BeagleSNES v0.6 User and Developer Guide

Andrew Henderson

www.beaglesnes.org

Contents

1 — An Introduction to BeagleSNES

1.1 Overview

BeagleSNES is a complete software system that turns the ARM-based BeagleBoard-xM (BB-xM) and BeagleBone Black (BBB) hardware platforms[1] into a dedicated device capable of executing software that was originally developed for the Super Nintendo Entertainment System®(SNES) video game console[2]. It allows you to play SNES titles on your television or computer monitor via the DVI digital video output of the BB-xM and the HDMI output of the BBB. The BBB also supports using the LCD3 cape for video output.

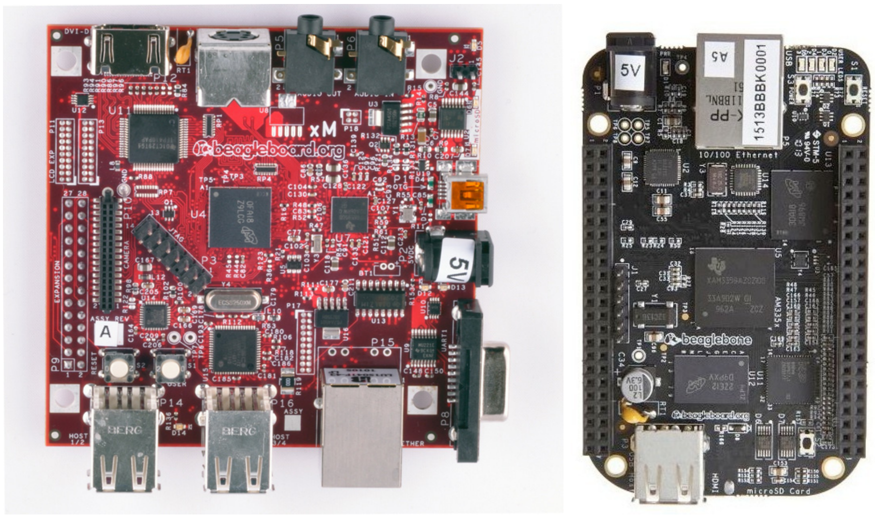


Figure 1.1: The BeagleBoard-xM (left) and BeagleBone Black (right) hardware platforms.

BeagleSNES originally began as a course project for a graduate embedded systems design class. While the scope of the project was to port the SNES9X emulator to the BB-xM and then tune the board's Linux kernel for performance, things got a little out of hand. The project kept

[1] Learn more about the BeagleBoard family of hardware platforms at `www.beagleboard.org`.

[2] Nintendo®and Super Nintendo Entertainment System®are registered trademarks of Nintendo Co. Ltd. and its subsidiary companies.

growing in complexity and scope. Over the span of four months, it matured into the initial release of the project that you see today. The bootloader, Linux kernel, file system, and emulator of BeagleSNES have all been customized for functionality, stability, and performance. These components were further customized when the project was ported to the BBB.

The source code for BeagleSNES, as well as the latest instructions for building, installing, and using the software, are freely available from `www.beaglesnes.org`. Almost all of BeagleSNES is licensed under the GNU Public License (version 3) as open source software (OSS), but the emulator itself is licensed under a custom, non-commercial license. What does this mean? It means that you are welcome to examine the code, modify it, learn from it, and even use it in projects of your own. But, you may not sell the system if it contains the emulator.

The hardware of the SNES (CPU, GPU, memory, audio chipset, etc.) is quite different from the hardware of the BB-xM and BBB. SNES software will never execute natively on these platforms, so BeagleSNES must *emulate* the hardware of the SNES in software. The inputs, outputs, and internal state of each piece of SNES hardware is represented as high-level source code (C and C++ code, in the case of BeagleSNES). The 16-bit microprocessor of the SNES (the WDC 65C816) runs at a clock speed of 3.58 MHz. This is much slower than the 32-bit ARM CPUs of the BB-xM (AM37x 800 MHz Cortex-A8) and BBB (AM335x 1 GHz Cortex-A8), meaning that there are enough resources available to dynamically translate every SNES program instruction, at the time that it would be executed on the SNES, into one or more ARM instructions and then execute them. This *just-in-time dynamic translation* approach has considerable overhead, but it is very effective in situations where the emulated system is much slower than the system performing the emulation.

Figure 1.2: The Super Nintendo Entertainment System®video game console.

Several SNES emulators currently exist, and BeagleSNES leverages this existing work (rather than "reinventing the wheel"). The SNES9X emulator[3] was selected to serve as the emulator base for the project for the following reasons:

- All source code for the emulator is available, and all optimizations and additions made to it as part of BeagleSNES can be freely released as long as the terms of the SNES9X license are followed.
- The source code for the emulator is cross-platform, meaning that its implementation is not tied to a particular display library or CPU architecture.

[3]www.snes9x.com

- Its emulation compatibility is quite good, allowing it to run many different pieces of SNES software accurately.

While many architecture-specific performance optimizations exist in SNES9X (such as using x86-based assembly cores to accelerate the emulation), many of these improvements can't be used with the ARM-based BB-xM and BBB. This limits the overall emulation speed of BeagleSNES, but the performance is generally acceptable.

1.2 Using This Manual

This guide was written to help you to install and configure BeagleSNES. It also contains technical information that provides details of how the system works and describes how to build the various components of the system from source. Hopefully, this information will not only allow you to enjoy BeagleSNES for your own entertainment, but also allow you to use this work as a basis for whatever BeagleBoard-based system you might want to create.

For any important aspects of the system that warrant additional attention, this manual uses the following box to highlight them:

> Be sure to pay attention to these boxes as you read through the manual. They will alert you to any important or useful information that you should be aware of.

For any commands executed on the command line, this manual uses the following box:

```
username@host$ sudo ./execute_this_command
```

Each section of the guide contains information on a different aspect of BeagleSNES. Section 1 is the introduction that you are reading right now. Section 2 describes how to download and install the pre-made file system image of BeagleSNES. For most end-users, this section will be the most important one to review. Section 3 provides instructions on how to add ROMs to the BeagleSNES system and configure the game selection GUI and gamepad button mapping. Section 4 is a troubleshooting guide to help you work through some of the common issues that people come across. Section 5 gives step-by-step instructions on how to build the various pieces of the BeagleSNES system. Section 6 provides more details about the configuration of the bootloader and Linux kernel used in BeagleSNES. Section 7 describes several details of the audio, video, and input subsystems of BeagleSNES. This will be of interest to developers that wish to review the technical details of how BeagleSNES interfaces with the BeagleBoard hardware. Section 8 describes how BeagleSNES might be made into a portable system and discusses the LCD3 video target and GPIO input. Section 9 acknowledges the people and projects that have contributed to the software and hardware that BeagleSNES is made from. Finally, section 10 serves as a changelog of the project from release to release.

Enjoy using BeagleSNES! Hopefully, you will learn a few things about embedded system design in the process, too!

1.3 BeagleSNES License

The bootloader, Linux kernel, and GNU utilities that make up the BeagleSNES system are all licensed under the GPL. These components can be freely modified, shared, and even sold, as long as the terms of the GPL are followed. Feel free to use them as a base for your own projects!

The license information for the BeagleSNES emulator application can be seen in figure 1.3. This license is a slightly modified version of the SNES9X license, which directs you to the stock SNES9X license for all of the details. It allows you to look at, learn from, and use BeagleSNES for personal use, but not to exploit it for commercial gain. This is NOT GPL licensed software, but it is freeware.

```
BeagleSNES - Super Nintendo Entertainment System (TM) emulator for the
BeagleBoard-xM and BeagleBone Black platforms.

(c) Copyright 2013        Andrew Henderson (hendersa@icculus.org)

BeagleSNES is a GUI front-end that is integrated into to the Snes9x
emulator, along with some hacks that shut off unneeded functionality
and tune the performance for execution on the BeagleBoard-xM and
BeagleBone Black platforms.

Please refer to the snes9x-license.txt file for more information on
the authors and license pertainting to the Snes9x codebase.  Specific
ports contains the works of other authors. See headers in individual
files.

BeagleSNES homepage: http://www.beaglesnes.org/
Snes9x homepage: http://www.snes9x.com/

Permission to use, copy, modify and/or distribute BeagleSNES in both
binary and source form, for non-commercial purposes, is hereby granted
without fee, providing that this license information and copyright
notice appear with all copies and any derived work.

This software is provided 'as-is', without any express or implied
warranty. In no event shall the authors be held liable for any damages
arising from the use of this software or it's derivatives.

BeagleSNES is freeware for PERSONAL USE only. Commercial users should
seek permission of the copyright holders first. Commercial use includes,
but is not limited to, charging money for BeagleSNES or software derived
from BeagleSNES, including BeagleSNES or derivatives in commercial game
bundles, and/or using BeagleSNES as a promotion for your commercial
product.

The copyright holders request that bug fixes and improvements to the code
should be forwarded to them so everyone can benefit from the modifications
in future versions.

Super NES and Super Nintendo Entertainment System are trademarks of
Nintendo Co., Limited and its subsidiary companies.
```

Figure 1.3: The `beaglesnes-license.txt` file included in the BeagleSNES source code.

2 — Installing BeagleSNES

2.1 Overview

BeagleSNES is more than just a single application. It is a complete POSIX operating system environment, complete with a custom Linux kernel, a full file system, and configuration files that tell BeagleSNES how to behave for the end user. Because of the general complexity of such a large system, it can sometimes be difficult to get everything up and running smoothly. This section will tell you how to download and install BeagleSNES so that you can get started using it as quickly and painlessly as possible.

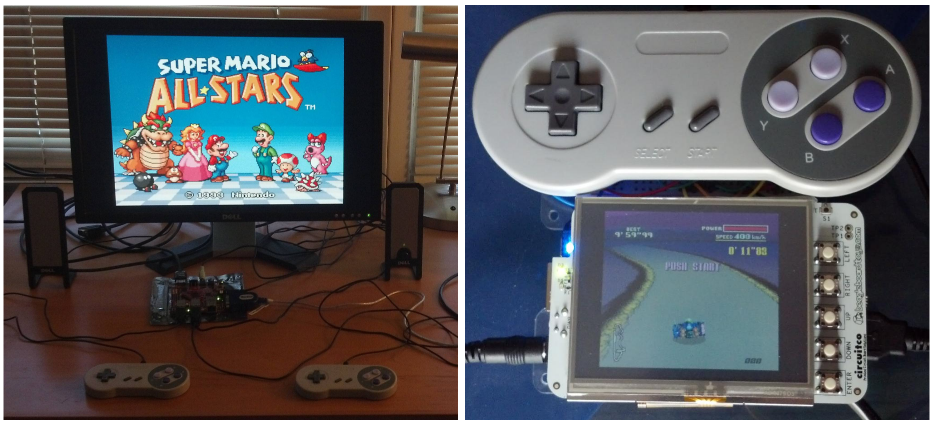


Figure 2.1: The 0.5 release of BeagleSNES, running on the BB-xM with DVI digital video output (left) and on the BBB with the LCD3 cape (right).

All source code for the project, as well as a full file system image of a complete BeagleSNES system, have been made available for download. Most users will only wish to try out BeagleSNES, rather than hack on its source code, so this section will focus on using the full file system image. Prior to starting, you'll need the following items:

- Either a BB-xM hardware platform that is at least revision C or a BBB platform that is at least revision A5A. Earlier revisions have not been tested and are not suggested.
- An external power supply to provide full power to the system. Powering via USB will not provide the necessary amount of current.

- An HDMI-to-DVI cable (BB-xM) or microHDMI-to-HDMI cable (BBB), and a television or monitor capable of displaying the digital video signal. Or, if you may wish to instead use a Circuitco LCD3 cape board for displaying video for the BBB.
- For the BB-xM, external speakers or a headset for listening to the audio output[1]. The audio for the BBB is sent over the HDMI connection.
- A microSD card with a capacity of at least 4 GB. BeagleSNES will rewrite the partition table on this card, so make sure that it does not contain any data that you need.
- Copies of the software ROMs that you wish to use with BeagleSNES.
- One or two Tomee™USB SNES Gamepad controller(s)[2]. The BB-xM supports using one or two controllers, but the BBB only supports one unless you use an external USB hub to provide additional USB ports. *USB controllers other than the Tomee™will work with BeagleSNES*, but they will most likely need their buttons remapped.

> While many SNES ROMs are widely available on the web, possession of these ROMs is generally considered to be piracy of commercial software. No ROMs are provided in the BeagleSNES file system images, and no help in obtaining ROMs will be given. You're on your own.

2.2 Downloading

There are two BeagleSNES file system images available for download. The first is a microSD card image that contains a complete BeagleSNES system (bootloader, kernel, file system, etc.). The second is a tarball of only the directory structure that includes the BeagleSNES application, its associated resources, and sample configuration files. This image is intended for more advanced users that wish to add the BeagleSNES emulator into an existing file system environment that they have created. The complete microSD card image will be referred to as the *full system image*, and the tarball of the BeagleSNES directory structure will be referred to as the *application image*. The contents of the application image already exist inside of the full system image.

Direct download links to the latest versions of all BeagleSNES software components are available at `www.beaglesnes.org`. SourceForge hosts all of the files for BeagleSNES[3], but `www.beaglesnes.org` should first be consulted for any additional information that you might need about the latest version of BeagleSNES prior to downloading.

2.3 Installing

The easiest way to install BeagleSNES is to write the full system image to a microSD card and then boot your BBB or BB-xM from that card. The speed class of the card is not all that important, but all of the BeagleSNES development was performed using a class 10 card. Slower speed classes may increase the time needed for a system boot, but otherwise the impact on the performance of BeagleSNES is assumed to be negligible.

Some advanced BeagleBoard users may already have a working file system, kernel, etc. on their system and only wish to install the BeagleSNES application. For those people, the application image can be copied into their existing file system. Libraries used by BeagleSNES (SDL, SDL_mixer, etc.)[4] are not shipped with the application image because you might not have

[1]Even when using an HDMI-to-HDMI cable for the video, audio will not be transmitted over the cable. This is because the signal being transmitted over the HDMI cable is actually a DVI signal (video only), rather than an HDMI signal that contains both video and audio data.

[2]Several online vendors offer this particular USB controller for sale. It is the only controller that will be officially supported for BeagleSNES.

[3]`https://sourceforge.net/projects/beaglesnes/files/`

[4]A full list of the libraries used can be seen in section 3.4.

built your particular kernel with the same features that are in the BeagleSNES kernel (ALSA, framebuffer console, etc.). It will be up to you to install the needed libraries on your system in a fashion that is compatible with your current kernel features.

In addition, BeagleSNES needs to be started at boot. There is a script named *service.sh*[5] in the application image that should be called by a boot `cron` job, the `rc.local` script, or other similar mechanism upon system start-up. Modify the path to the BeagleSNES directory in *service.sh* to point to the path where you have installed the application image on your system.

To copy the full system image to your microSD card, first download the image and then use `bunzip2` to decompress it. Execute a Unix `dd` command similar to the following to write the image to your microSD card[6]:

```
username@host$ sudo dd if=beaglesnes_system.img of=/dev/sdX bs=1M
```

This command will take a little while to run, so be patient. The output file parameter (`of`) will may vary depending upon what device represents the microSD card on your particular Linux system. For most systems, `/dev/sdX` (`sda`, `sdb`, etc.) will represent the microSD card. On others, you might see `/dev/mmcblkX` instead, where the "X" is a number (0 for the first microSD card, 1 for the second, etc.). It is up to you to determine which it is. Don't guess and try different parameters without knowing for sure! A bad guess might end up with your accidentally overwriting data on some other device in your system!

Once the full system image is copied onto the microSD card, the card will contain two file system partitions. The first partition is the `VFAT` `/boot` partition. The kernel and bootloader live in this partition, as do the BeagleSNES configuration files and any ROMs and game images that you've copied to the system. The second partition is the `EXT4` `/rootfs` partition where the BeagleSNES application and base Linux distribution live. The `VFAT` filesystem on the `/boot` partition is understood by Windows, the various OSes of Apple hardware, and Unix OSes like Linux and FreeBSD, so it is simple to add new games and change configuration settings no matter which OS you use to do so.

Insert the microSD card with the BeagleSNES image into your board and then power the board up. After about 10-12 seconds, the game selection menu will appear, begin playing background music, and display with the default dummy game entries.

When using BeagleSNES on a BBB, you do *not* have to hold down the "user boot" button during boot. The BeagleSNES `uEnv.txt` file is written such that the bootloader on the BBB's eMMC understands that it should boot from the microSD card and it will do so automatically.

A freshly-downloaded BeagleSNES microSD card image has a default boot configuration for a BBB using HDMI for audio and video. It *will not boot* for other configurations (BBB with LCD3 or BB-xM) until you change the bootloader and `uEnv.txt` files. The next section explains how to change the default bootloader and `uEnv.txt` files over to their respective versions for booting BeagleSNES on the BBB with LCD3 or the BB-xM.

[5]This script not only launches the emulator, but it also uses `nice` to adjust the priority of the emulator by -20. This effectively makes the emulator the highest-priority user space process running on the system.

[6]Windows users can use the Win32 Disk Imager utility to write the image to the microSD card. Download it at: `http://sourceforge.net/projects/win32diskimager`

3 — Configuring and Using BeagleSNES

3.1 Overview

Within the full system image, the BeagleSNES application lives in the *application root directory*: `/home/ubuntu/beaglesnes`. To make things easier for the end-user, all of the standard configuration files that control game configuration, controller button mapping, etc. are located inside of the `/boot` partition. The `/boot` partition uses a `VFAT` filesystem, so files can be easily copied to it or modified by mounting the microSD card on a Windows, Mac, or Linux system and making the desired changes.

When logging into a running BeagleSNES system, use the username "ubuntu" and password "temppwd". You can use the "ubuntu" account to `sudo` any administrative commands that you feel need to be performed on the system.

3.2 Choosing a Boot Configuration

BeagleSNES supports three boot configurations:

- The BB-xM using DVI for video and the BB-xM audio jack for audio via external speakers.
- The BBB using HDMI for audio and video.
- The BBB using an LCD3 cape for video and some other hardware (such as a USB audio device) for audio.

The default configuration for the full system image is for the BBB using HDMI. While the BBB and BB-xM kernels have different names and can exist side-by-side in the `/boot` partition, the bootloader and `uEnv.txt` files for each platform have the same names and can't co-exist in the same location. To address this issue, three scripts have been added to the `/boot` partition to change the current boot configuration to one of the other configurations. The `setup_BBB_HDMI.sh` script makes the BBB with HDMI the active boot configuration by copying the bootloader and `uEnv.txt` files from the `BBB_HDMI` subdirectory into the root of the `/boot` partition. The `setup_BBB_LCD3.sh` script makes the BBB with LCD3 the active boot configuration in the same manner (copying files from the `BBB_LCD3` subdirectory). The `setup_BBxM.sh` script will do the same, but it will copy the files from the `BBxM` subdirectory and make those BBxM files the active ones for the system. You won't be able to execute these scripts while the system is running (i.e. a BBB-configured full system image won't even boot on a BB-xM, so you will not be able to shell into the system to run `setup_BBxM.sh`), so you must mount the `/boot` partition of the microSD card on a Linux system, `cd` into the partition, and then execute the script by using one of the following commands:

To set the system to BB-xM:

```
username@host$ sudo sh ./setup_BBxM.sh
```

To set the system to BBB with HDMI:

```
username@host$ sudo sh ./setup_BBB_HDMI.sh
```

To set the system to BBB with LCD3:

```
username@host$ sudo sh ./setup_BBB_LCD3.sh
```

Alternatively, you can simply copy the files from the desired boot configuration subdirectory (`BBxM`, `BBB_HDMI`, `BBB_LCD3`) into the root of the `/boot` partition to replace the existing boot files. This is the easiest approach for Windows users that are unable to execute the shell scripts.

All three `uEnv.txt` files have two different configuration settings within the `uEnv.txt` file itself: the "fast boot" configuration and the "development boot" configuration. By default, all of the `uEnv.txt` files are set up for fast boot. The differences between the fast and development boot configurations are the boot time and which OS features are enabled. Fast boot uses BeagleSNES as the `init` process of the Linux system. Only the bare minimum of system daemons are started, and only the file systems required for BeagleSNES operation are mounted. Networking is not enabled under this configuration, and you also won't be able to build the BeagleSNES in-place on the hardware. But, the boot time for the system is only about 10-12 seconds. The development boot configuration uses the standard Linux `init` daemon as the `init` process, and it enables all features of the system, but it requires a boot time of about 20 seconds. Instructions on how to switch between the fast boot and development configurations are given in Section 6. Unless you are planning on doing BeagleSNES development, using the microSD card image for your own development project, or need network access to transfer ROMs to your microSD card, stick with using the fast boot configuration.

3.3 Configuring BeagleSNES

Once the image has been copied onto the microSD card and configured for your platform, you have two options for configuring the BeagleSNES application and loading game ROMs into it. The first method is to mount the microSD card on another system and then access the `/boot` partition directly to copy and modify files. This is by far the easier method, since you are free to copy files back and forth between your system and the BeagleSNES file system quickly and easily.

The second method is to insert the microSD card into your BB-xM or BBB, power up the system, and then use the ethernet interface[1] to shell into your BeagleSNES system and access the file system remotely. The full system image uses DHCP to request an IP address when an ethernet cable is attached to the system. You can use utilities like `scp` to copy files to and from the system and `ssh` to shell into the system and edit files locally. You will have to manually determine what IP has been assigned to the device by performing a broadcast ping or

[1]For the BBB, you can also use an FTDI cable to shell into the system to make configuration changes (though you won't be able to copy files). The FTDI interface will work for both fast boot and networking configurations.

by looking at the administration interface of your router or DHCP server to see which IPs are in use. Your BeagleSNES system will show up in the administration interface with the hostname `beaglesnes`.

> If you are using a fast boot configuration, networking will not be enabled and this method can't be used.

There are three things that must be completed before your BeagleSNES is usable:

- Descriptions and information for each game listed in the selection menu must be added to the `games.xml` file.
- ROM and image files associated with each game listed in the selection menu must be copied into the file system.
- Gamepads and GPIO inputs must be configured to use the proper button mapping in the `games.xml` file if the default mapping isn't correct.

Adding game information is covered in Section 3.3.1. Copying ROMs and images into the file system is covered in Section 3.3.2. Configuring button mappings for gamepads is covered in Section 3.3.3.

For the current release, BeagleSNES's game selection menu supports having a large, but not unlimited, number of game titles. At least one game must exist in the actual system, of course, but there is no specified upper limit. As more games are added to the menu, more system RAM will be consumed and the start-up time for the BeagleSNES application will increase. While no stress tests have been performed to see just how many games can be added to the system, it should easily handle several hundred games.

All configuration and data files for BeagleSNES are located in the `beaglesnes` subdirectory in the root directory of the `/boot` partition. On the running BeagleSNES system, the location of the `beaglesnes` directory is `/boot/uboot/beaglesnes`. If the microSD card with BeagleSNES is mounted on a Windows system, the path to the `beaglesnes` directory is off the root of the drive that the `/boot` partition is mounted as. For example, if the microSD card is inserted into a Windows system and the `/boot` partition becomes drive `E:`, the path of the `beaglesnes` directory is `E:\beaglesnes`.

3.3.1 Modifying games.xml

The `games.xml` should be the only configuration file that you will need to modify for BeagleSNES. It is an XML file that describes the button mapping for your controllers, GPIO mapping for controls, and the game descriptions, box images, and ROM files associated with each game listed in the game selection menu. `games.xml` is located in the `beaglesnes` subdirectory of the `/boot` partition.

The XML tags used in the `games.xml` file are described in Table 3.1. A sample `games.xml` that demonstrates some of these tags is shown in Figure 3.1. Two sample game entries are listed. The first game, `GameTitle1`, has an entry that defines every possible field. The second game, `GameTitle2`, has an entry that defines only the bare minimum fields required to make a valid entry (`<title>` and `<rom>`). If the other tags are not defined, default values are used in their place. A full sample `games.xml` file is included in both the full and application file system images. It contains a dummy configuration for a menu with 10 ROMs, gamepad button mappings for the player 1/2 controllers, GPIO inputs, and a pause menu button combo. The sample will act as a template to help guide you as you configure BeagleSNES for your own use.

XML Tag	Purpose	Minimum	Maximum
<config>	Top level tag of XML file	1	1
<gpio>	Mark section of GPIO buttons	0	1
<gpio_*>	Specific GPIO button mapping	0	1 per <gpio>
<player1/2>	Mark section of button mapping	0	2
Various tags	Map buttons (<vaxis>, <select>, etc.)	0	1 per <player?>
<pause_combo>	Combo that triggers pause menu	0	1
<pause_key>	Button that is part of the pause combo	0	No limit
<snes>	Mark section of all SNES games	1	1
<game>	Mark section of a single game	1	No limit
<title>	Title of a game	1 per <game>	1 per <game>
<rom>	Filename of a game ROM	1 per <game>	1 per <game>
<image>	Filename of a game image	0	1 per <game>
<genre>	Genre of a game	0	2 per <game>
<year>	Year of a game	0	1 per <game>
<text>	Description of a game	0	5 per <game>

Table 3.1: XML tags of the BeagleSNES `games.xml` file.

There is no need to add path prefixes, such as `image/` or `rom/`, when specifying the ROM or box image filenames in `games.xml`. BeagleSNES already knows which directories to look in to find those files, so only the name of the file itself needs to be specified.

There are five special characters (“ “ ”, “&”, “‘”, “<”, “>”) that have a special meaning in XML, so you must replace each one with a special *predefined entity* in the XML file. Each predefined entity is a placeholder that will be replaced with the appropriate character when the text is actually rendered on the screen. The predefined entities for “ “ ”, “&”, “‘”, “<”, and “>” are “`"`”, “`&amp;`”, “`&apos;`”, “`<`”, and “`>`”, respectively. For example, the title “`Pocky & Rocky`” would be entered as “`Pocky &amp; Rocky`” in the XML file.

3.3.2 Adding ROMs and Images

Each SNES ROM must be copied into the `rom` subdirectory of the `beaglesnes` directory of the `/boot` partition. The downloadable image files for BeagleSNES do not include any SNES ROMs, so the system will not do anything useful until you add your own ROMs to the file system. The SNES ROMs used with BeagleSNES must be NTSC SNES titles. ROMs can be compressed (`.zip` or `.gz` file extension) or uncompressed (`.smc` file extension). If a particular title is able to run on the SNES9X emulator, then it will most likely run on BeagleSNES. Add as many SNES ROMs as you like into the `rom` directory, but only ROMs specified in the `games.xml` file will appear in the game selection menu.

Each box or screenshot image must be copied into the `image` subdirectory of the `beaglesnes` directory of the `/boot` partition. The game selection menu is designed to accommodate a 200x145 pixel PNG image for the box image. This image size has the same aspect ratio as the original SNES cartridge box. Some SNES titles (such as Japanese releases that were never released in the North American market) originally came in a box with a different aspect ratio. As long as the height of the image does not exceed 145 pixels, the extra width of the image should still fit in the menu screen without a problem.

```
<?xml version="1.0"?>
<config>
<snes>
  <game>
    <title>GameTitle1</title>
    <rom>filename1.smc</rom>
    <image>box_image1.png</image>
    <year>1999</year>
    <genre>Genre1</genre>
    <genre>Genre2</genre>
    <text>Description text line 1</text>
    <text>Description text line 2</text>
    <text>Description text line 3</text>
    <text>Description text line 4</text>
    <text>Description text line 5</text>
  </game>
  <game>
    <title>GameTitle2</title>
    <rom>filename2.smc</rom>
  </game>
  ... More <game></game> tags go here for more games ...
</snes>
</config>
```

Figure 3.1: Example of a `games.xml` file.

3.3.3 Configuring Gamepad Button Mapping

There are a variety of USB human interface devices (HIDs) that are recognized as joysticks by the Linux kernel's USB subsystem. Not all of these devices are true joysticks (they might be gamepads, or even devices that looks nothing like a game controller), but they all fall within the same category of "close enough in functionality to a joystick to be treated as such by the kernel". While these joysticks and gamepads can be used to provide input to BeagleSNES, the physical buttons of each controller must first be mapped to the "logical" buttons inside BeagleSNES that correspond to the buttons of the original SNES controller.

As a reference, the original SNES controller is shown in Figure 3.3. This controller has a directional pad that provides two "axes" for directional input (an up/down axis and a left-right axis). It also provides a total of eight buttons: the rectangular start and select buttons, four round input buttons for A/B/X/Y, and two shoulder buttons on the edge of the controller for L (left) and R (right).

BeagleSNES uses the configuration file `snes9x.conf` to define the *default* mappings between physical joystick buttons and axes to the logical buttons and axes used by the emulator. `snes9x.conf` is located in the `beaglesnes` directory of the `/boot` partition. The button mapping settings are located in a section within the `snes9x.conf` named `[BeagleSNES Controls]`. Previous versions of BeagleSNES required users to change this section to map SNES gamepad buttons to the buttons of your joystick. But, the `snes9x.conf` configuration file is now completely ”off-limits” for the end-user. All button mappings are now set using the `games.xml` file.

Figure 3.2: The included 200x145 pixel `blank_box.png` placeholder image that is used if no image is specified.

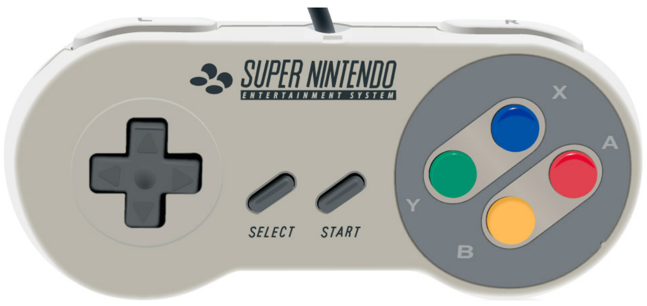


Figure 3.3: The original SNES controller. The L (left) and R (right) shoulder buttons are located on the top edge of the controller.

Don't touch the `snes9x.conf` file! All configuration settings are now made via the `games.xml` file.

The `<player1>` and `<player2>` tags in the `games.xml` file describe button mappings between your joystick and the SNES gamepad. Figure 3.4 demonstrates the use of these tags. The default button mappings are used for any controller or button not explicitly mentioned in the `games.xml`. The only exception to this is the `<pause>` tag. If you have a joystick with enough buttons to map all of the SNES controller buttons and still have at least one button to spare, the extra button can be mapped to trigger the pause menu. Unless the `<pause>` is explicitly mapped, though, no default is mapped.

3.3.4 Configuring A Pause Combination

If you can't spare an extra button to trigger the pause menu, you can map the pause to a combination of buttons. When this button combo is pressed on the player 1 controller, the pause menu will trigger. For example, you can map the left and right trigger buttons and the start and select buttons together as a pause button combo. When all four are pressed simultaneously, the pause menu triggers. Because you don't want the pause button combo to be a set of buttons that will interfere with gameplay, selecting three to four buttons for the combo is advised.

The pause combo is defined by using two tags: `<pause_combo>` and `<pause_key>`. An example pause combo that uses the left and right trigger buttons and the start and select buttons is shown in Figure 3.5.

This particular pause combo assumes the same button mapping as seen in the `<player1>` tag in Figure 3.4. For example, the first `<pause_key>` of the combo is for button `4`. The `<player1>` tag maps 4 to the left trigger button in the `<L>` tag. If the button mapping changes in the `<player1>` tag, the `<pause_key>` tags must be changed to match it.

```
<?xml version="1.0"?>
<config>
<player1>
  <vaxis>1</vaxis>
  <haxis>0</haxis>
  <L>4</L>
  <R>5</R>
  <A>1</A>
  <B>2</B>
  <X>0</X>
  <Y>3</Y>
  <select>8</select>
  <start>9</start>
  <pause>11</pause>
</player1>
<player 2>
  ... similar tags go here for the player 2 controller ...
</player2>
... other tags go here ...
</config>
```

Figure 3.4: Controller button mapping portion of the `games.xml` file.

```
<pause_combo>
  <pause_key>4</pause_key>
  <pause_key>5</pause_key>
  <pause_key>8</pause_key>
  <pause_key>9</pause_key>
</pause_combo>
```

Figure 3.5: XML for a pause button combo in the `games.xml` file.

3.3.5 Configuring GPIOs For Input

The BBB platform has support for mapping GPIO inputs as player 1 controller inputs. The SNES directional pad and buttons, and even the pause button, can be mapped to GPIOs. A maximum of 13 GPIO inputs (four for the d-pad, eight for buttons, and one for pause) can be mapped using a contiguous block of the BBB's pins (P8.7 through P8.19). Mapping the GPIOs uses the `<gpio>` tag and a series of `<gpio_*>` tags that specify particular SNES controller buttons and controls. Figure 3.6 shows a sample GPIO mapping.

Each individual `<gpio_*>` tag contains a number from one through thirteen. These numbers correspond to the pins on the P8 header of the BBB. Pin P8.7 is 1, pin P8.8 is 2, etc. up through pin P8.19, which is 13. If a particular pin number is not mapped, it won't be initialized by BeagleSNES. For more information on the hardware details of trigger GPIOs for input to BeagleSNES, check out Section 8.4.

```
<gpio>
  <gpio_gpleft>1</gpio_gpleft>
  <gpio_gpright>2</gpio_gpright>
  <gpio_gpup>3</gpio_gpup>
  <gpio_gpdown>4</gpio_gpdown>
  <gpio_L>5</gpio_L>
  <gpio_R>6</gpio_R>
  <gpio_A>7</gpio_A>
  <gpio_B>8</gpio_B>
  <gpio_X>9</gpio_X>
  <gpio_Y>10</gpio_Y>
  <gpio_select>11</gpio_select>
  <gpio_start>12</gpio_start>
  <gpio_pause>13</gpio_pause>
</gpio>
```

Figure 3.6: XML for mapping GPIOs to SNES controls in the `games.xml` file.

3.4 Using BeagleSNES

Once the system has been configured, it is now ready for use. BeagleSNES expects either one or two USB gamepads to be plugged in. Two host USB ports on the BB-xM system have been designated by BeagleSNES as the gamepad ports: one for "player one" and the other for "player two". *Any gamepad that is not plugged into one of these designated USB ports is ignored.* Figure 3.7 shows which two BB-xM USB ports are designated for the gamepads.

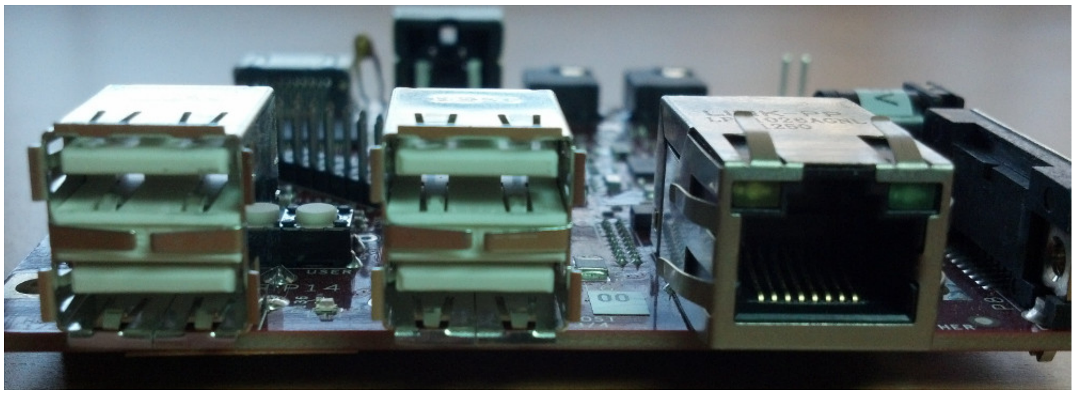


Figure 3.7: The USB ports of the BB-xM. The top-left USB port is reserved for the "player one" gamepad, and the top-right USB port is reserved for the "player two" gamepad.

The BBB has only one host USB port, so there are two options for using gamepads with the system. The first option is to plug a gamepad into the host USB port. This is the port for the "player one" gamepad, and no "player two" gamepad can be used. The second option is to plug a USB hub into the host USB port and then plug one or two gamepads into the hub. Much like the host USB ports of the BB-xM, one port of the hub will be the "player one" port, and another will be for "player two". Whichever physical ports on the hub report as the first and second logical ports will be used as the "player one" and "player two" ports, respectively. *Any gamepad that is not plugged into one of these designated USB ports on the hub is ignored.*

3.4.1 Game Selection Menu

Assuming that you have successfully completed the steps of the adding game descriptions, ROMs, and box images to your BeagleSNES system, it will now boot up to a complete menu of SNES titles that are ready to be played. This game selection menu will *only* respond to the "player one" controller, and the menu will display the text "PLEASE CONNECT PLAYER 1 GAMEPAD" if no gamepad is connected to the "player one" USB port[2]. Press up and down on the gamepad's direction-pad to change the currently selected title on the menu. Press either the "select" or "start" gamepad button to launch the currently selected title. If there are GPIO inputs that are mapped for up/down and start/select, they can also be used to navigate the menu and launch a title. Once a title has been launched, the menu will fade out and the emulator will begin execution.

Once you have launched a title and left the game selection menu, you must use the pause menu to return to the game selection menu to select a different game to play.

3.4.2 Pause Menu

The pause menu can only be triggered while playing a game. Its purpose is to allow the user to load and save *snapshots* of the current game state, exit the current game and return to the game selection menu, and, of course, pause the game. Figure 3.8 shows the pause menu.

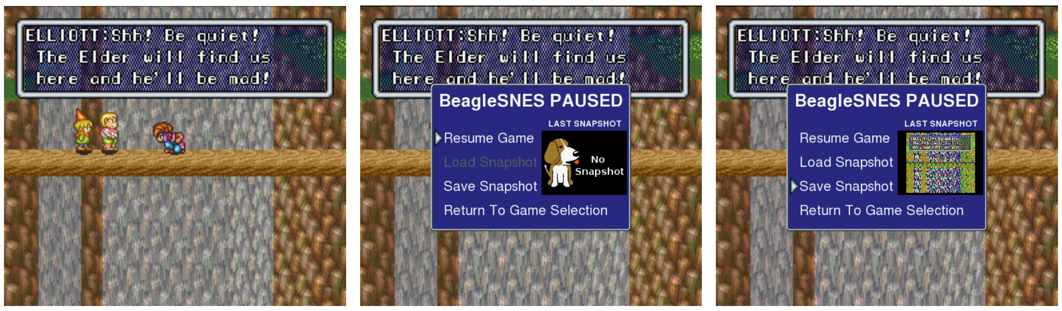

ELLIOTT:Shh! Be quiet! The Elder will find us here and he'll be mad!

Figure 3.8: The pause menu. While in-game (left), triggering the pause menu will bring up a set of menu options (middle). If a snapshot for the current game has been previously saved, the "Load Snapshot" option will become active and a thumbnail will be shown in the "LAST SNAPSHOT" preview window.

There are three ways to trigger the pause menu:

- If the pause menu has been mapped to a single gamepad button in `games.xml`, press that button.
- If a pause button combination has been mapped in `games.xml`, press that button combo.
- If the pause menu has been mapped to a GPIO in `games.xml`, trigger that GPIO.

Pressing up and down on the player 1 gamepad, or triggering the appropriate GPIOs, will navigate the pause menu. Likewise, using select/start or its GPIO equivalent will select a menu option. Selecting "Resume Game" will remove the pause menu and unpause the game. Also, pressing the pause button or button combo, or triggering the pause GPIO, a second time will also remove the pause menu and unpause the game.

The "Load Snapshot" and "Save Snapshot" menu options allow you to save the current state of the game and then load it to return to it at any time. This is useful for saving the progress of the current play session when you need to take a break and are not near a point in the game

[2]The LCD3 display target will not display this warning.

where progress can be saved. It is also useful for saving your game just prior to making that difficult jump or fighting a tough boss battle. Snapshots are named after the filename of the ROM that they are associated with, so renaming ROMs will make your save states seem to disappear (although they are still present on the microSD card). If no snapshot for the current game exists, the "Load Snapshot" option will be grayed-out and unavailable. Only one snapshot can exist for each game. Once a snapshot is saved, the prior snapshot is overwritten.

> Snapshots are stored in the `savestate` subdirectory of the `beaglesnes` directory on the `/boot` partition.

The "Return to Game Selection" menu option exits the current game and return you to the game selection menu. Be sure to save a snapshot of your current game if you want to return to it later. Selecting this option is roughly the same as turning off the power on your SNES console: your current game will be lost forever.

4 — Troubleshooting

4.1 Overview

If you are unable to `dd` the full system image to your microSD card:

- Do you not have `bunzip2` on your system? For Ubuntu users, you can `apt-get` it by installing the `bzip2` package.
- Is your download corrupt? Does the MD5 hash for the file match the hash listed for the file at SourceForge?
- Did you disable the write-protect switch on your microSD card?
- Do you have permissions to write to the raw microSD card device file?
- Are any pre-existing partitions on the microSD card currently mounted? Unmount all partitions on the microSD card before performing the `dd`.

If you can't get your system to boot after writing the full system image to your microSD card:

- Are you using an external power supply (5VDC 2A)? The system won't boot when using only USB power.
- Did you disable the write-protect switch on your microSD card? The BeagleSNES still needs to write out log files to the local file system.
- Is there any output coming from the debug serial or FTDI port? If there is nothing, `dd` the full file system to the microSD card again after you verify the MD5 hash of the downloaded file.
- Did you cleanly unmount the microSD card from the Linux system that you used to `dd` the image to the card? Some Linux systems will auto-mount the microSD card as soon as it has a valid file system on it. If you just pull the microSD card out when it is mounted, it is possible to damage the boot partition.
- Did you change the boot configuration from the default BBB HDMI configuration if you needed to do so? If you are using a BBxM, you will need to change the boot configuration to the BBxM configuration to use the proper bootloader.

If you can see the boot splash screen, but the emulator won't start:

- If you revert to the backup of the original `games.xml` file (you made that backup, right?) does the emulator start? If so, you have a mistake in your `games.xml` file.
- If you do not see anything wrong with your `games.xml` file, does the debug output over serial or FTDI show any errors in parsing the XML? Any errors will give you hints as to

where troubles may lie in your `games.xml`.

- Make sure that all box image and ROM filenames are correct. The filenames are case-sensitive, and you do not need to add the directories to the filenames.
- If nothing else works, shell into the system via the network and try to manually execute the `service.sh` script in the application root directory. You will need to `sudo` the command or be a superuser in order to do so. Watch for any debug output that might give you a hint as to why it is not working.
- If all else fails, send us a bug report and we'll check it out.

If the game selection menu appears, but it does not respond to your USB gamepad:

- Do you have the Tomee USB gamepad? If not, you may have to play with the button mappings in the `games.xml` file.
- Is your gamepad plugged into the correct USB port? Do you see the message telling you to plug in the "player one" gamepad? Only a gamepad that is plugged into the "player one" USB port can control the menu selection screen.
- For the BBB, there is a known kernel bug that causes USB devices to not be enumerated by the USB bus. Press the "Reset" button (button S1) or unplug and plug in the power cable to reset the system and try again. It may take a few tries, unfortunately. Watch the FTDI output for an *Error -110* to see if this is the particular problem that you are seeing. This problem appears to occur less often once the system has been running for a while, and it may take several tries before the errors go away and the USB subsystem enumerates properly.
- For the BBB, plugging an external USB hub into the host USB port can be very tempermental. If using an external hub, plug in the hub and all controllers prior to starting the system. Sometimes the external hub will stop responding if all controllers are removed from it. An unpowered external hub will work, but it makes the BBB far more sensitive and can create problems when hotplugging controllers.
- Verify that your gamepad is operating properly by testing it with another computer.

If the game selection menu starts, but is missing one or more of your titles:

- If you do not see anything wrong with your `games.xml` file, does the debug output over serial or FTDI show any errors in parsing the XML? Any errors will give you hints as to where troubles may lie in your `games.xml`.
- Did you use predefined entries in the XML for any special characters (Section 3.3.1)? Parsing of the XML will stop once an error is reached.

If the game selection menu starts, and you can launch a title, but then the screen goes blank and nothing happens:

- Make sure the filename for the ROM is correct in the `games.xml` file.
- Make sure that the ROM is valid by executing it under another emulator.
- Make sure that the ROM is in the `rom` subdirectory of the `beaglesnes` directory in the `/boot` partition.

> If you run into some other problem that is not covered here, please feel free to submit a bug report to Andrew at `hendersa@icculus.org`.

5 — Building BeagleSNES

5.1 Overview

BeagleSNES is more than just a single application. It is a complete operating system with a custom Linux kernel, a full file system, and configuration files that tell BeagleSNES how to behave for the end user. Because of this, it can be sometimes be difficult to get up and running. This section focuses on building the individual components of BeagleSNES using the build scripts that have been included to assist you. Unless you are interested in installing/uninstalling/changing configurations/libraries/binaries in the file system, it is probably best to leave the file system as-is and concentrate on modifying only the bootloader, kernel, and BeagleSNES application.

5.2 Bootloader

BeagleSNES's bootloader is a modified version of the v2014.01 release of U-Boot. The BeagleSNES source code archive for the bootloader is just a `tar`'d-up version of the hacked-on, working code base of what is running inside of the BeagleSNES system. Use the following steps to build it:

- Install a cross-compiler tool chain on your development system. BeagleSNES's development was done using a tool chain built using `crosstool-ng`[1], though you could probably install an ARM development tool chain via `apt-get` or whatever package manager your particular Linux distribution is using. It is also possible to use the tool chain that is included with the BeagleSNES kernel source.
- Edit the `beaglesnes-build.sh` shell script in the `beaglesnes_u-boot` code base to point to the location of your cross-compiler tool chain. You'll need to modify the `TOOL_PATH` and `TOOL_CHAIN` variables to point to the path where your tool chain is located and the prefix of each tool chain binary, respectively. `crosstool-ng` installs the tool chain in `$(HOME)/x-tools`, so that will most likely be the value that you need to set for `TOOL_PATH`. Tool chain binaries are typically named following the format of `$TARGET_ARCH$TOOL`. $TOOL is `gcc`, `g++`, etc., and `$TARGET_ARCH` is probably something similar to `arm-cortex_a8-linux-gnueabi-`.
- Edit the `beaglesnes-build.sh` shell script to choose to build the bootloader for the BB-xM or the BBB. To do this, there are two lines in the shell script that build a configuration

[1] Download `crosstool-ng` at: `http://crosstool-ng.org`

for `omap3_beagle_config` (BB-xM) and `am335x_evm_config` (BBB). Uncomment only one of these two lines to select the hardware to build the bootloader for. The comments in the shell script will guide you.

- Once you have set up your `beaglesnes-build.sh` script, execute it. This will start the build process and generate a variety of files in the `beaglesnes_u-boot` directory. The two files that you are interested in are `MLO` (the first stage bootloader) and `u-boot.img` (the second stage bootloader that actually loads the kernel).
- Mount the first partition (the boot partition) of a microSD card that has had the full BeagleSNES file system image written to it. *Back up the `MLO` and `u-boot.img` binaries in the root of that mounted partition.*
- Replace the `MLO` and `u-boot.img` binaries on the microSD card (*after you have backed them up*) with the `MLO` and `u-boot.img` that you just built.
- `sync` the microSD card's mounted boot partition's file system, then unmount it.

You've just built the BeagleSNES bootloader and installed it. Insert the microSD card into the system, power it up, and it should now boot using your new bootloader. Remember that your new bootloader files will be replaced with the stock BeagleSNES ones if you execute the `setup_BBB.sh` or `setup_BBxM.sh` scripts in the boot partition.

5.3 Kernel

BeagleSNES's BB-xM kernel is built from a snapshot of the Linux 3.7.10 kernel branch of Robert Nelson's stable kernel project[2] that was retrieved using `git` on 11 March 2013. The BBB kernel is built from a snapshot of the 3.8.13 kernel branch from the same project, retrieved on 02 June 2013. Both of these projects contains a script called `build_kernel.sh` that performs a number of useful activities:

- Installs a cross-compilation tool chain for building the kernel.
- Checks for new kernel patches.
- Applies kernel patches (over 200 of them!) to the kernel source tree for all BeagleBoard platforms.
- Builds the kernel.

The BeagleSNES kernel source trees contain the `build_kernel.sh` file, but it has been modified. The script no longer checks for kernel patches and updates or downloads the tool chain. The tool chain is already present in the BeagleSNES kernel source package[3], so there is no need to download it again. To build one of the kernels, use the following steps:

- *Optional:* If you want to rebuild the splash screen that displays during kernel boot, execute the `build_beagle_splash.sh` script. This will convert the splash screen image file[4] into the proper format and then copy it to its place in the kernel source tree. Note that the splash screen image is different in the two kernel trees. For BB-xM, it is 640x480. For BBB, it is 720x480.
- Execute the `build_kernel.sh` script.
- Choose to "Exit" the kernel configuration tool without changing any settings. *Note: Don't change the configuration unless you know what you are doing!* The kernel will then begin building, which will take a few minutes. All generated binaries will be under the `deploy` directory of the BeagleSNES kernel source.
- Mount the first partition (the boot partition) of a microSD card that has had the full BeagleSNES file system image written to it. *Back up the `zImage` (BB-xM) or `3.8.13-bone20.zImage`*

[2] `https://github.com/RobertCNelson/stable-kernel`
[3] The tool chain is in the `dl/gcc-linaro-arm-linux-gnueabihf-4.7-2012.12-201214_linux` directory.
[4] `beaglesnes_kernel_splash.png`

kernel binary in the root of that mounted partition.

- Replace the kernel binary on the microSD card (*after you have backed it up*) with the `deploy/3.7.10-x9.uImage` or `deploy/3.8.13-bone20.zImage` kernel binary that you just built. Rename the `3.7.10-x9.uImage` kernel to `zImage` in the `/boot` partition.
- `sync` the microSD card's mounted boot partition's file system, then unmount it.

Optional: The BeagleSNES does not load any kernel modules (everything necessary is built directly into the kernel), but if you also want to install the kernel modules created when the kernel is built, use the following steps (performing the commands using `sudo`):

- Mount the second partition (the rootfs partition) of a microSD card that has had the full BeagleSNES file system image written to it. *Back up the `/lib/modules/3.7.10-x9` (BB-xM) or `/lib/modules/3.8.13-bone20.zImage` (BBB) directory in the root of that mounted partition.*
- If building for a BB-xM, replace the `/lib/modules/3.7.10-x9` directory on the microSD card (*after you have backed it up*) with the `deploy/mod/lib/modules/3.7.10-x9` directory that you just built.
- If building for a BBB, copy the file `deploy/3.8.13-bone20-modules.tar.gz` to the `/lib/modules` directory on the microSD card. `gunzip` and `untar` the file. Replace the `/lib/modules/3.8.13-bone20` directory on the microSD card (*after you have backed it up*) with the `/lib/modules/lib/modules/3.8.13-bone20` directory.
- `sync` the microSD card's mounted rootfs partition's file system, then unmount it.

You've just built the BeagleSNES kernel and installed it. Insert the microSD card into the system, power it up, and it should now boot using your new kernel.

Advanced Users: If you are interested in using the latest kernel source and patches, fetch the latest code from Robert Nelson's Git repository and replace the `.config` in the kernel source with the `beaglesnes_config` file from the top-level directory of the BeagleSNES kernel source. The `beaglesnes_config` file is a copy of the kernel configuration that is used when building the BeagleSNES kernel. It will ensure that you build a kernel with the proper features enabled. By default, the kernel from Git tries to be generic as possible by building a *large* number of hardware drivers as modules. BeagleSNES does not build most of these drivers, and the ones that it does need are linked directly into the kernel, rather than being built as modules.

5.4 BeagleSNES Application

The BeagleSNES application is both the game selection menu front-end and the SNES emulator back-end. Both components are built as a single binary. The application has the following library dependencies:

- The Simple DirectMedia Library[5] for general audio, video, and input.
- The SDL_ttf library[6] for rendering TrueType fonts.
- The SDL_image library[7] for loading image files of various formats.
- The SDL_mixer library[8] for loading and playing music and sound effects.
- The Expat library[9] for parsing the XML `games.xml` file.

Because it can be difficult to build and install all of these libraries correctly using a cross-compiler tool chain, it is easiest to build both the libraries and the BeagleSNES application on the BB-xM or BBB. The BeagleSNES full system image contains all of these libraries and header files, as

[5] `http://www.libsdl.org`
[6] `http://www.libsdl.org/projects/SDL_ttf`
[7] `http://www.libsdl.org/projects/SDL_image`
[8] `http://www.libsdl.org/projects/SDL_mixer`
[9] `http://expat.sourceforge.net`

well as a complete development tool chain. These instructions will assume that you are building BeagleSNES inside of the full system image.

> If you decide to use a package manager (like `apt-get`) to install the needed libraries, you're going to drag in a number of additional dependencies that you won't need. Typically, SDL is built with X11 support, so your package manager will install a large number of X-Windows applications and libraries. It is better to build the libraries manually than to install them via a package manager. If you don't have the patience or experience to manually install the libraries and their dependencies, stick with using the libraries and headers already installed in the full file system image. These libraries include only the most basic features of SDL (framebuffer console graphics, ALSA audio, and the joystick input subsystem).

The BeagleSNES application source code is available as a downloadable source code archive. The source code is also installed inside the full system image[10]. To build the BeagleSNES application on a BB-xM or BBB system, use the following steps:

- Shell into the running BeagleSNES system (username "ubuntu", password "temppwd"). The easiest way to do this is via the serial debug interface[11], though you can also use `ssh` to shell into the system via the ethernet interface.
- Enable swap space. `swapfile` is a one gigabyte swap file that has already been created for you in the full system image. You only need to turn on swapping by using the command: `sudo swapon /var/cache/swap/swapfile`
- `cd` into the `/home/ubuntu/build/beaglesnes_src/sdl` directory.
- Execute the `configure` script in this directory. The script should find the tool chain and all of the necessary libraries and headers to build the application. For best performance, execute the `configure` script with the `--enable-neon` option:

```
ubuntu@beaglesnes$ sudo ./configure --enable-neon
```

- Edit the `beagleboard.h` file to select which platform you are building for. There are two `#define` statements: one for `BEAGLEBONE_BLACK` and one for `BEAGLEBOARD_XM`. Uncomment the `#define` that describes your system, and comment out the other one.
- If you have chosen to enable the `BEAGLEBONE_BLACK` build via `beagleboard.h`, you can also comment in the "`#define CAPE_LCD3 1`" in `beagleboard.h` to enable the support for the LCD3 display instead of the HDMI video output.
- Execute a `make` command in this directory. The application will now build, and it will take a while. If you did not enable swapping, the system will exhaust its RAM and fail on this step.
- *Optional:* Shrink the generated binary (and reduce its loading time) by stripping it via the command: `strip ./snes9x-sdl`
- Copy the generated `snes9x-sdl` binary to the `/home/ubuntu/beaglesnes` directory. If you built a binary for BBB with HDMI video, name it `snes9x-sdl.BBB.HDMI`. If you built a binary for BBB with HDMI video, name it `snes9x-sdl.BBB.LCD3`. If you built a binary for BB-xM, name it `snes9x-sdl.BBxM`. These are the files that the `service.sh` script looks for.
- Execute a `sync` command.

[10]It is in the `/home/ubuntu/build/beaglesnes_src` directory within the full system image.

[11]Read `http://elinux.org/BeagleBoardFAQ` for more information on setting up and using the serial debug interface

6 — BeagleSNES Kernel and Bootloader

6.1 Overview

BeagleSNES uses the U-Boot bootloader[1] and a trimmed-down Linux kernel. The configuration of both of these components impacts both the runtime performance of the system and the time necessary to boot the system. A general timeline of the events that occur as the system boots is as follows:

Time (seconds)	Events Taking Place
0	System is powered up Bootloader begins executing immediately Bootloader fetches its configuration file from the boot partition Bootloader initializes DVI video at a resolution of 1280x720 (BB-xM)
2	Bootloader has finished loading and decompressing the kernel Kernel begins bootstrapping Bootloader terminates execution
5	Kernel initializes DVI video at a resolution of 640x480 (BB-xM) Kernel turns on framebuffer console (BB-xM) Boot splash screen is displayed (BB-xM)
8	Kernel initializes HDMI video at a resolution of 720x480 (BBB) Kernel turns on framebuffer console (BBB) Boot splash screen is displayed (BBB) `init.sh` script in the BeagleSNES image is executed
10	Boot splash screen disappears BeagleSNES is launched via the `service.sh` script Game selection menu logo and gradient bar appear on the screen
12	Game selection menu displays entire screen and begins playing audio System is now accepting user input and is ready for use

Table 6.1: Timeline of BeagleSNES events, from power up to "usable system".

While waiting 12 seconds for an embedded system to achieve a usable state is still quite a

[1] `http://www.denx.de/wiki/U-Boot`

long time, earlier development versions of BeagleSNES would take as long as *35 seconds* to go from power up to usable system. There is still room for a great deal of improvement in the boot speed of BeagleSNES. Other projects, such as SwiftBeagle[2], have done an excellent job discovering various ways that the bootloader and kernel can be changed in order to decrease boot time. In this section, several of these points will be discussed as the bootloader and kernel of BeagleSNES are examined.

6.2 Bootloader

BeagleSNES's U-Boot bootloader has had two noteworthy modifications: the standard 3-second delay prior to loading the kernel has been removed[3] and preliminary support has been added for a bootloader splash screen[4] for the BB-xM. Version 2013.04 (used in BeagleSNES v0.4) is the earliest version of U-Boot that will work for BeagleSNES because it is the first version to include *device tree* support[5] for the BBB's cape bus.

A BeagleSNES bootloader configuration file, `uEnv.txt`, exists for the BB-xM and both BBB boot configurations. Each of the uEnv.txt files set a U-Boot environment variable called `bootargs`. This value of this variable is the set of command line arguments that are passed to the Linux kernel when the kernel begins its bootstrapping process. There are two sets of `bootargs` in each of the `uEnv.txt` files: one for a quicker-booting configuration that specifies the BeagleSNES `init.sh` script as the "init" process for the Linux kernel, and one for a slower-booting configuration that uses the default Linux `systemd`[6] as the "init" process. By default, the "fast boot" configuration is enabled, which enables BeagleSNES to get up and running as quickly as possible.

The quicker-booting configuration allows the system to boot in about half of the time it takes the slower-booting configuration. This is because the quicker configuration does not start many of the system daemons that are started in the slower configuration (since `systemd` is not used and the usual init scripts are not executed). The quicker configuration performs only the bare minimum of remounting the root filesystem read-write, starting `udevd` (to create the device files for the USB gamepads in the `/dev` filesystem), and adjusting the ALSA mixer levels. The downside to this is that many other important services, such as networking, are never started. Aside from the difference in the "init" process that is used, both configurations for each platform specify `bootargs` that turn on debug output to the serial port, enable the DVI or HDMI video output at specific resolutions, and shut off the framebuffer's blinking cursor.

Switching between the faster and slower boot configurations requires editing the `uEnv.txt` file. The `bootargs` for both configurations are in each of the `uEnv.txt` files, so it is only a matter of commenting out the configuration that isn't needed and commenting in the configuration that is. Make sure that only one configuration is commented in at a time. Comments have been included in the `uEnv.txt` files to guide you. The `uEnv.txt` files that come with BeagleSNES can be seen in Figures 6.1 (for the BB-xM), 6.2 (for the BBB with HDMI), and 6.3 (for the BBB with LCD3).

[2]`http://code.google.com/p/swiftbeagle/`

[3]This setting is located in the file `include/configs/omap3_beagle.h` (BB-xM) and `include/configs/am335x_evm.h` (BBB) in the bootloader source. It is the `CONFIG_BOOTDELAY` pre-processor define.

[4]These modifications have been added to the file `board/ti/beagle/beagle.c` in the bootloader source. Check out `dss_init()`, which sets up the memory buffer and does some register adjustments.

[5]To learn more about the device tree and how it is used in the Linux kernel to describe hardware peripherals for embedded devices, go to `http://www.devicetree.org`

[6]To learn more about `systemd` and its role as a system and service manager for Linux, go to `http://www.freedesktop.org/wiki/Software/systemd`

```
#kernel_file=3.7.10-x9.uImage
kernel_file=zImage
console=ttyO2,115200n8
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait fixrtc
boot_fstype=fat
xyz_load_image=${boot_fstype}load mmc 0:1 0x80300000 ${kernel_file}
xyz_load_dtb=${boot_fstype}load mmc 0:1 0x815f0000 /dtbs/${dtb_file}
xyz_mmcboot=run xyz_load_image; echo Booting from mmc
video_args=setenv video vram=12MB omapfb.mode=dvi:640x480-16@60 omapdss.
    def_disp=dvi
device_args=run video_args; run expansion_args; run mmcargs

# This is for a slower boot, development setup
#mmcargs=setenv bootargs console=${console} ${video} root=${mmcroot}
    rootfstype=${mmcrootfstype} ${expansion} coherent_pool=1M mpurate=800
    console=blank vt.global_cursor_default=0
# This is for a quick boot, non-development setup
mmcargs=setenv bootargs console=${console} ${video} root=${mmcroot}
    rootfstype=${mmcrootfstype} ${expansion} coherent_pool=1M mpurate=800
    console=blank vt.global_cursor_default=0 init=/home/ubuntu/beaglesnes/
    init.sh

expansion_args=setenv expansion buddy=${buddy} buddy2=${buddy2} camera=${
    camera} wl12xx_clk=${wl12xx_clk}
loaduimage=run xyz_mmcboot; run device_args; bootm 0x80300000
```

Figure 6.1: BeagleSNES's BB-xM `uEnv.txt` bootloader configuration file.

6.3 BeagleBoard-xM Kernel

BeagleSNES was originally developed specifically for the Rev C BB-xM. While the Linux kernel has good support for the BB-xM hardware (OMAP3, TWL4030, etc.), every embedded system kernel must be tuned and patched for the specific environment in which it will be deployed. The BeagleBoard platforms are no exception. BeagleSNES's kernel codebase uses a series of 207 kernel patches that have been contributed and vetted by the BB community. While not every patch is specific to the BB-xM (BeagleBoard- and BeagleBone-specific patches are also applied), many of the patches increase system stability by improving voltage core management and thermal management.

The kernel itself has been compiled to remove many features that aren't needed for BeagleSNES. This has reduced the compressed size of the kernel from 3.4 megabytes with additional modules dynamically loaded down to a little less than two megabytes with no additional modules loaded. While this might not seem like a big difference, it has reduced the boot time of BeagleSNES by about 6 seconds. Roughly 200 ms of these savings are attributed to the reduction in time needed to load and decompress the smaller kernel. The bulk of the savings comes from eliminating the time needed to initialize various kernel subsystems that are never be used. For example, many security features (SELinux, address space layout randomization (ASLR), stack overflow protection), symmetrical multi-processing support, support for various file system types, SCSI support, etc. were all originally compiled into the kernel.

The BB-xM's DM3730 CPU begins running at a clock speed of 600 MHz, but is quickly changed to 800 MHz per the `mpurate` kernel command line option that is provided by the bootloader. The "performance" governor[7] forces the CPU clock rate to remain at its maximum

[7] Option `CPU Power Management -> CPU Frequency scaling` in the kernel configuration menu.

```
kernel_file=3.8.13-bone20.zImage
console=ttyO0,115200n8
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait fixrtc

loadkernel=load mmc ${mmcdev}:${mmcpart} 0x80200000 ${kernel_file}
loadfdt=load mmc ${mmcdev}:${mmcpart} 0x815f0000 /dtbs/${fdtfile}
boot_ftd=run loadkernel; run loadfdt

# This is for a slower boot, development setup
#mmcargs=setenv bootargs acpi=noirq console=blank vt.global_cursor_default=0
     mpurate=1000 video=720x480-16@60 console=${console} root=${mmcroot}
    rootfstype=${mmcrootfstype}
# This is for a quick boot, non-development setup
mmcargs=setenv bootargs acpi=noirq console=blank vt.global_cursor_default=0
    mpurate=1000 video=720x480-16@60 console=${console} root=${mmcroot}
    rootfstype=${mmcrootfstype} init=/home/ubuntu/beaglesnes/init.sh

uenvcmd=run boot_ftd; run mmcargs; bootz 0x80200000 - 0x815f0000
```

Figure 6.2: BeagleSNES's BBB HDMI `uEnv.txt` bootloader configuration file.

rate of 800 MHz. Originally, `mpurate` was set to "auto" and the "ondemand" governor was used by the kernel. The high CPU utilization of the emulator would force the governor to adjust the clock speed from 600 MHz to 800 MHz once the BeagleSNES application began executing in user space. This caused the system to boot more slowly (since the CPU clock was at 600 MHz for the entire kernel bootstrap process), so the `mpurate` and governor were changed accordingly.

While this CPU can theoretically be clocked at 1GHz, there are a few problems with doing so. The BeagleSNES kernel source lacks a patchset that existed in earlier kernel versions (2.6.36, 3.0) that handle the VDD1 voltage adjustment of the CPU. It is non-trivial to port this patch set to newer kernels. While the clock speed could be forced to 1GHz without this patch set, you will slowly fry your BB-xM's CPU while doing so. The system also becomes a bit unstable when you're pushing the CPU at 100% at 1GHz, which leads to application crashes (including BeagleSNES). Therefore, the governor will only push the clock speed to a stable 800 MHz.

An interesting note: Titles that use Super FX[8] chips, such as "Star Fox", run really well when the system is clocked to 1GHz (5 FPS at 800 MHz versus 20-25 FPS at 1 GHz). The extra clock speed does make a big difference.

6.4 BeagleBone Black Kernel

When the BBB was first released in April 2013, the 3.8 Linux kernel was the OS kernel for the board. The BBB's kernel branch consisted of the mainline 3.8 kernel source tree with hundreds of patches applied to it to support the BBB's functionality. While the BBB's hardware is largely based upon that of its predecessor, the BeagleBone, the BBB's 3.8 kernel is quite different from the mature 3.2 kernel used by the BeagleBone. Thus, the change from 3.2 to 3.8 for BBB led to new kernel features and new problems.

The main advantage of using the new 3.8 kernel was the introduction of the Device Tree[9] data structure. This data structure allows for a more-uniform way of defining processor pin multiplexing ("muxing"), identifying which kernel resources and drivers are associated with

[8]For more information on Super FX, refer to the Wikipedia article: `http://en.wikipedia.org/wiki/Super_fx`

[9]Learn more about the Device Tree at `http://www.devicetree.org/Main_Page`.

```
kernel_file=3.8.13-bone20.zImage
console=ttyO0,115200n8
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait fixrtc

loadkernel=load mmc ${mmcdev}:${mmcpart} 0x80200000 ${kernel_file}
loadfdt=load mmc ${mmcdev}:${mmcpart} 0x815f0000 /dtbs/${fdtfile}
 boot_ftd=run loadkernel; run loadfdt

# This is for a slower boot, development setup
#mmcargs=setenv bootargs acpi=noirq capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN console=blank vt.global_cursor_default=0 mpurate=1000 console=${console} root=${mmcroot} rootfstype=${mmcrootfstype}
# This is for a quick boot, non-development setup
mmcargs=setenv bootargs acpi=noirq capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN console=blank vt.global_cursor_default=0 mpurate=1000 console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} init=/home/ubuntu/beaglesnes/init.sh

uenvcmd=run boot_ftd; run mmcargs; bootz 0x80200000 - 0x815f0000
```

Figure 6.3: BeagleSNES's BBB LCD3`uEnv.txt` bootloader configuration file.

particular processor pins. Unfortunately, the 3.8 kernel is a combination of different kernel subsystems and can be a bit tempermental at times. BeagleSNES will be moving to the 3.12/3.13 kernel tree once more testing has been completed and BeagleSNES is properly adapted to use it. Other than the newer features of the 3.8 kernel, the BBB's kernel is quite similar in nature to that of the BBxM: the "performance" CPU governor is used and the kernel has been trimmed down to reduce the kernel size, eliminate unneeded device drivers, and shorten kernel boot time.

6.5 Other Improvements to Boot Speed

Other system customizations that contributed to a faster boot time are:

- Shrinking the size of the root file system partition from 7.5 gigabytes to about 3.5 gigabytes. This reduces the file system mount time at system start up.
- Shutting off unneeded daemons. By default, the Ubuntu Linux image that BeagleSNES is based on started standard daemons like the print spooler, Apache web server, lvm, dns-clean, fetchmail, etc.. These daemons slow down system start up and use the CPU for no purpose.
- Reducing the number of `tty` terminals. There is no need for a dozen terminals on an embedded device, and it takes time and resources to start and maintain them all.

For further performance improvements, the place to begin is the removal of even more daemons. The next place would be to remove unneeded files from the root file system, convert chunks of the remaining file system into read-only `cramfs` files, and then mount them to form a mostly read-only file system. This will greatly reduce the portion of the file system that actually needs to be read-write (such as `/dev`, `/sys`, `/tmp`, etc.). Moving to a lightweight shell, such as busybox[10], would allow for even more files to be removed.

The BeagleSNES menu and emulator are currently built in-place on the BB-xM and BBB platforms, so a complete toolchain and debugger are installed inside of the current BeagleSNES image for convenience. These tools could also be removed to reduce the size of the system even further. The 1 gigabyte swap file could also be removed, saving a great deal of space.

[10]`http://www.busybox.net/`

7 — BeagleSNES Subsystems

7.1 Overview

This section is currently BB-xM only. It must be rewritten to include BBB information. The overall concepts are still the same, though.

BeagleSNES uses the Simple DirectMedia Layer (SDL)[1] library to provide audio, video, and input event functionality for both the game selection menu and the emulator. SDL is a cross-platform library which has a simple programming interface that abstracts away many of the details involved when creating multimedia applications. The BeagleSNES emulator is based on an SDL port of SNES9X[2]. The mainline version of SNES9X only supports an X11 video display target, which is unsuitable for the framebuffer video of BeagleSNES. By using the SDL port, BeagleSNES is able to support a variety of interfaces, including those offered by the the BB-xM platform.

While the game selection menu and the emulator are really two separate applications (after all, the only purpose of the game selection menu is to provide the emulator with the filename of the ROM to be loaded), both exist in the same binary. Early testing showed that there was a visible screen flicker and a delay of several seconds when transitioning from one to the other when each was broken out as a separate binary. Therefore, the game selection menu will initialize all of the needed SDL subsystems (audio, video, and joystick), execute, and then shut down the audio and joystick subsystems prior to transitioning to the emulator. The emulator will then reinitialize the audio and joystick subsystems while leaving the video subsystem alone[3]. By not shutting down the video subsystem and reinitializing it, the visible screen flicker is removed.

[1] `http://www.libsdl.org`

[2] The original SNES9X-SDL source, prior to the BeagleSNES modifications, can be fetched from the Git repository at: `https://github.com/domaemon/snes9x-sdl`

[3] SDL allows an application to change the video resolution on-the-fly without reinitializing the video subsystem. When the game selection menu and the emulator used to use different screen resolutions, the emulator had to change the resolution when it began executing. But, it did not need to restart the video subsystem to do so.

7.2 Audio Subsystem

The BeagleSNES audio subsystem uses the ALSA[4] interface that is exposed by kernel. The interface is provided by the system-on-chip (SoC) audio driver for boards that have TWL4030 audio CODEC hardware[5]. SDL offers an ALSA back-end, so many of the details involved in interfacing with ALSA are abstracted away from BeagleSNES by SDL.

The game selection menu uses a helper library, SDL_mixer[6], to provide an even simpler interface to SDL's audio subsystem. SDL_mixer allows BeagleSNES to easily fade music in and out, load audio in a variety of formats, and mix several channels of audio together. SDL only provides a standard interface for pushing audio data out to the audio driver back-end, so the more advanced functionality of SDL_mixer provides the additional features needed to create a full-featured audio solution.

BeagleSNES configures the audio hardware of the BB-xM to use signed 16-bit stereo audio samples with a sampling rate of 32000 Hz. Both the game selection menu and the emulator use these same audio settings for simplicity. The audio subsystem is initialized in the game selection menu so that audio feedback and background music[7] can be played while the end-user examines the list of games that are available for play.

When a game is launched from the menu, any playing audio fades out and the audio subsystem is shut down. The emulator then reinitializes the audio subsystem using its original program logic. The emulator's implementation of the SNES's audio DSP performs all of the audio mixing, so SDL_mixer is no longer needed. While the real SNES hardware will play audio at almost the same time that its DSP generates the audio data, the BeagleSNES emulator has audio latency of approximately 100 ms. This is caused by the time needed to place the generated DSP audio into the kernel's circular audio buffer and the time needed before the pre-existing audio data ahead of the new data in the buffer is consumed. In practice, this delay is hardly ever noticed. This latency can be reduced by shrinking the size of the kernel audio buffer (12000 bytes), but this increases the amount of CPU resources dedicated to refilling the buffer and risks the possibility of an audio buffer underrun.

7.3 Video Subsystem

BeagleSNES system's video output is displayed via the BB-xM's DVI digital output. The initial release of BeagleSNES used the S-Video analog output for NTSC video output, but this ended up being a poor choice for the following reasons:

- The analog video signal is very noisy. The horizontal sync varies from scanline to scanline, there is a lot of static, and the color quality is poor. Whether this is due to poor drivers or poor hardware quality is unknown.
- The 720x482 NTSC resolution is larger than what is needed for BeagleSNES. This means that more pixels are being updated than are actually needed, which degrades performance.
- During development, profiling of BeagleSNES showed that displaying the framebuffer was much slower for the analog output than it was for the digital output.
- The NTSC analog output has a refresh rate of 29.97 Hz, while the DVI digital output has a refresh rate of 60 Hz. This means that the digital output can potentially display at a rate of 60 FPS, while the analog signal artificially limits the frame rate to half that amount.
- NTSC signals suffer from overscan.

[4]The Advanced Linux Sound Architecture. Read more about it at: `http://www.alsa-project.org`

[5]This audio driver is located in the file `sound/soc/omap/omap-twl4030.c` in the kernel source tree.

[6]`http://www.libsdl.org/projects/SDL_mixer`

[7]The background music that is played behind the menu comes from the group ViRiLiTY. It was originally the tracked background music of an illicit serial number generator used for the piracy of the "ALO Power Audio Converter" software.

All components of BeagleSNES render their graphics to the framebuffer console, which is set to a native resolution of 640x480 and a color depth of 16 BPP. These particular settings are hard-coded into the "omapfb" framebuffer driver[8] within BeagleSNES's Linux kernel. The refresh rate of 60 Hz means that whatever data is present in the framebuffer is pushed out as a single frame of video and is displayed for roughly 0.0167 seconds. At the end of that time period, the data in the framebuffer will then be pushed out as the next single frame of video.

The emulator runs at a logical 60 FPS internally, but this does not necessarily mean that it will render video at 60 FPS. The emulator can potentially skip the rendering of many frames of video if it falls behind schedule and needs to make up the difference. The skipped frames will not even be noticed in many cases, though performance-intensive titles that require a large amount of CPU time to emulate the SNES hardware will not have enough CPU time left to perform the rendering of enough frames each second to make the gameplay "smooth".

7.3.1 Video During Boot

One of the defining characteristics of a consumer-oriented embedded device is that it "turns on" immediately after it is powered up. The device must give the consumer some indication that it is now going through its start-up process. Displaying some sort of graphical splash screen immediately after power up is an excellent way to achieve this. Unfortunately, BeagleSNES does not provide such a splash screen. Or, at least, it does not do so immediately upon powering up. To do so would require adding functionality to the BeagleSNES bootloader to display the splash screen[9].

Roughly five seconds after power up, a much simpler method to display a splash screen becomes available. At that point in time, the bootloader has loaded, decompressed, and begun bootstrapping the kernel. The kernel initializes the graphical framebuffer and displays the first graphical output of the system: an 80x80 pixel, 224 color image of Tux the Penguin. This icon is displayed in the upper left corner of the framebuffer console. BeagleSNES turns this image into a splash screen by replacing the original image[10] with a much larger image: a 640x480 pixel, 224 color image of the BeagleSNES logo.

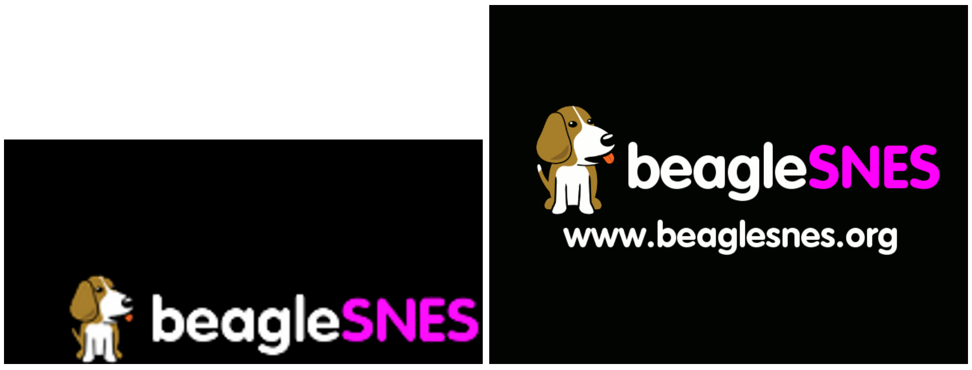


Figure 7.1: The original BeagleSNES's splash screen (left) was the smallest size it could be (640x300) for NTSC (720x482), but the current one (right) is the same size as the screen resolution (640x480) for simplicity.

Even though BeagleSNES uses a full-screen splash image, it is not really necessary for the

[8]This framebuffer video driver is located in the file `drivers/video/omap/omapfb_main.c` in the kernel source tree.

[9]There actually is preliminary support for a splash screen in the bootloader, but it has not been "hooked up" yet and needs further testing.

[10]This image is located at `drivers/video/logo/logo_linux_clut224.ppm` in the kernel source tree.

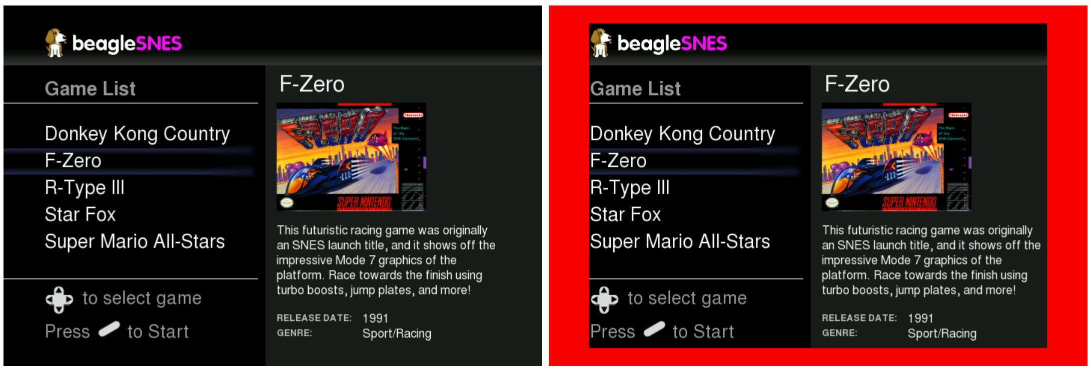
splash screen to be the full resolution of the screen. The background color of the framebuffer is black, and the image is displayed in the upper left corner of the screen, so it is only necessary to include the bare minimum amount of splash screen image needed to center the BeagleSNES logo on the screen. The splash screen is compiled into the kernel, so a smaller image reduces both the size of the kernel and the time it takes for the bootloader to load the kernel into memory from the microSD card. Adding a splash screen in this fashion increases the compressed kernel's size by only about 20kb. Normally, debug and information messages are printed to the framebuffer console as the kernel boots, but BeagleSNES suppresses the printing of these messages[11] so that they do not overwrite the splash screen image.

The splash screen remains on the screen while the kernel continues to bootstrap. Once it is finished, the splash screen disappears as the "icon" in the upper left corner is removed. The screen remains blank until the game selection menu appears approximately 17-18 seconds after power-up.

7.3.2 Game Selection Menu

The BeagleSNES game selection menu uses the entire 640x480 screen. The video subsystem of SDL is used to request an SDL video surface to render to. This surface is not requested using the SDL_FULLSCREEN flag, even though the resolution we want to use is the entirety of the screen. Profiling the use of SDL_FULLSCREEN has shown that it is much slower than not using it, so it is always better for BeagleSNES to use a "windowed" mode for all of its framebuffer graphics. For the simple animations of the menu, though, the impact of slower execution is minimal.

When BeagleSNES used analog video, not all of the space on the screen was usable. Due to NTSC overscan, some portion of the edges of the screen may be "cut off" by the television that the image is being displayed on. The amount that is cut off varies depending upon the particular television, so it is important to design the interface so that there is a "safety buffer" around the edges of the screen where text and other important items must not be placed. Typically, no more than a few pixels are cut off on each edge, though the left edge of the screen loses far more: a cut off of 12-16 pixels is common. Digital video does not have this issue.



This futuristic racing game was originally an SNES launch title, and it shows off the impressive Mode 7 graphics of the platform. Race towards the finish using turbo boosts, jump plates, and more!

RELEASE DATE: 1991
GENRE: Sport/Racing

Figure 7.2: A screenshot of the old game selection menu (left), and the same menu with the potential NTSC overscan regions marked in red (right).

7.3.3 Emulator Execution

NOTE: This section contains information about the video of the emulator from when BeagleSNES used analog video output. It remains in the documentation because it may be of some technical interest to other BeagleBoard developers.

[11] These messages can still be viewed via the serial debug interface (`/dev/ttyO2`), though.

<Analog video material begins here>

Once a selection has been made via the game selection menu, the screen fades to black prior to starting the emulator. This was not a choice based upon aesthetics (well, not directly, anyway). Internally, the emulator requests a 616x478x16 SDL video surface to render to. This resolution is different than what was previously set, so the SDL video subsystem actually has to be reconfigured with the new resolution. If the framebuffer is completely black, the transition between resolutions is never noticed by the end user. It also ensures that portions of the screen that are never updated by the emulator's rendering remain black during its execution.

The odd 616x478 resolution of the emulator was determined by the following method. Internally, each rendered scanline of SNES graphics is 512 pixels long. If we subtract the 512 pixels from the 720 pixel screen width, we have 208 pixels left in each scanline of the screen framebuffer (the screen framebuffer has a pitch of 1440 bytes, or (720 pixels * 2 bytes), so there are no extra offscreen pixels). If we divide these 208 pixels in half (since half will fall to the left of our SNES data and half will fall to the right of it), we have 104 pixels. 512 pixels + 104 pixels = 616 pixels, which is the resolution we request. Since we are not requesting a full-screen SDL video mode (via the SDL_FULLSCREEN flag when calling SDL_SetVideoMode()), the SDL surface will render into the left-most 616 pixels of the screen. With the offset of 104 pixels, the 512 pixels of the SNES scanline will render in the center of the screen. The emulator renders at a vertical resolution of 478 pixels. It is actually rendering each of the SNES's internal 239 scanlines twice to pixel double up to 478 scanlines. The SDL surface will render into the top-most 478 pixels of the screen, leaving the bottom 4 scanlines of the framebuffer untouched. Some, if not all, of these 4 scanlines will never even be seen, since they are usually lost to overscan.

<Analog video material ends here>

Finally, by using an SDL_UpdateRect() call to only update the portion of the framebuffer containing the 512x478 portion of SNES image data, we leave the chunks of black in the framebuffer that are to the left and right of the SNES graphics untouched. Why go to all of this trouble? Speed. There is a considerable difference between the speed of using the SDL_FULLSCREEN mode (slow) and using only a portion of the screen (much faster). Titles that would run at close to 30 FPS would drop to 8-12 FPS when running in a full screen mode.

7.4 Input Subsystem

BeagleSNES uses USB gamepad events as its only method for receiving input from the end user. Both SDL and the Linux kernel view these gamepads as a USB human interface device (HID) class joystick devices. SDL's input event mechanism for gamepads is split over two subsystems: video and joystick. The video subsystem captures most events that are experienced by SDL-based applications: keyboard, mouse, window manager, etc.. Once the video subsystem is initialized, SDL's internal event queue is initialized as well. Joystick events (the type of event that would be received from a gamepad) are an optional addition to the SDL event mechanism. They are not received by the general SDL event handling code until the joystick subsystem is initialized. Once that initialization occurs, gamepad events are also generated and placed into SDL's event queue alongside the other types of events.

When the joystick subsystem is initialized, the underlying limitations of the Linux kernel's joystick interface become apparent. The number of connected joysticks is reported, and any reported joysticks can then be "opened" to begin receiving events from them. Unfortunately, there are *many* things that could go wrong:

- If the USB hardware is unreliable (connections to devices are lost, but then quickly restored), SDL will stop responding to it. This is because the "old" joystick is gone, but a "new" one has taken its place, and the new joystick must then be opened to use it.

- Joystick devices vary in the number of axis and buttons that the device provides to the software. If an end-user plugs in a USB joystick device that reports a different number of buttons or axes, or reports buttons that are mapped in a different order than expected, then the behavior of the device will not be correct.
- There are no "new joystick plugged in" or "joystick unplugged" events generated by SDL. There is only a mechanism to query how many joystick devices are currently plugged into the system.

BeagleSNES does its best to work around these limitations. There is a known problem that the BB-xM has with briefly losing power to its USB devices. This problem has the same effect as rapidly unplugging and then plugging in a gamepad: the gamepad appears to stop working because the joystick driver device file in the file system that SDL listens to for joystick events becomes invalid. BeagleSNES addresses this problem with a kernel patch[12] that properly handles this issue.

Originally, BeagleSNES required that all gamepads must be plugged in prior to power up and not removed during system operation. This situation is far from ideal, since gamepads can accidentally become unplugged during use. The current approach to this problem is to tie the internal representation of a particular gamepad to the physical USB port on the system that the gamepad is plugged into. Two of the BB-xM USB ports are designated explicitly for BeagleSNES gamepad use: one being the port for "player one" gamepad and the other for "player two". Unless a gamepad is plugged into either of those particular ports (i.e. the "player one" or "player two" port), it will be completely ignored.

When a USB gamepad is plugged into the BB-xM, new files are dynamically created in the `/sys` and `/dev` directories of the root file system, thanks to the `sysfs` support in the kernel[13]. Plugging a USB gamepad into the "player one" port creates a file[14], as does plugging a USB gamepad into the "player two" port[15]. Each of these files is a symbolic link to one of the `/dev/input/jsX` joystick device files. These device files are opened from user space applications to access joystick events. By watching the creation and destruction of these files via a `stat()` or `readlink()` function call, BeagleSNES can determine whether a gamepad is present or not in those particular physical USB ports.

The `/dev/input/jsX` device files enumerate the various joystick devices that are connected to the system, in the order that they are connected, using the device files `js0`, then `js1`, etc. BeagleSNES will dynamically remap joystick events so that the events are attributed to the correct gamepad. On each pass through the event handler loop, the "player one" and "player two" `sysfs` files are examined to see if they exist. If either has changed, the joystick subsystem of SDL is shut down and restarted to reconfigure the gamepads accordingly. This expensive operation only occurs if a controller has been plugged in or unplugged. Otherwise, the only overhead created is by the two checks to see if the `sysfs` files exist and a table lookup to remap joystick events to the correct gamepad as those events occur.

[12]This patch is for 3.2+ kernels. Older kernels still suffer from this issue.

[13]`sysfs` is a virtual file system that is accessible from user space. It exports information about devices that are registered with the kernel, as well as a variety of driver information. By reading these files from user space applications, the devices that these files represent can be interacted with easily.

[14]`/dev/input/by-path/platform-ehci-omap.0-usb-0:2.2:1.0-joystick`

[15]`/dev/input/by-path/platform-ehci-omap.0-usb-0:2.4:1.0-joystick`

8 — Making BeagleSNES Portable

8.1 Overview

With the small physical size and flexibility of the BeagleBone family of boards, it is natural to consider using BeagleSNES as the software to drive a portable video game system. To this end, some research has gone into developing such a system and has resulted in the new LCD3 cape display target for BeagleSNES. A prototype portable unit is shown in Figure 8.1.

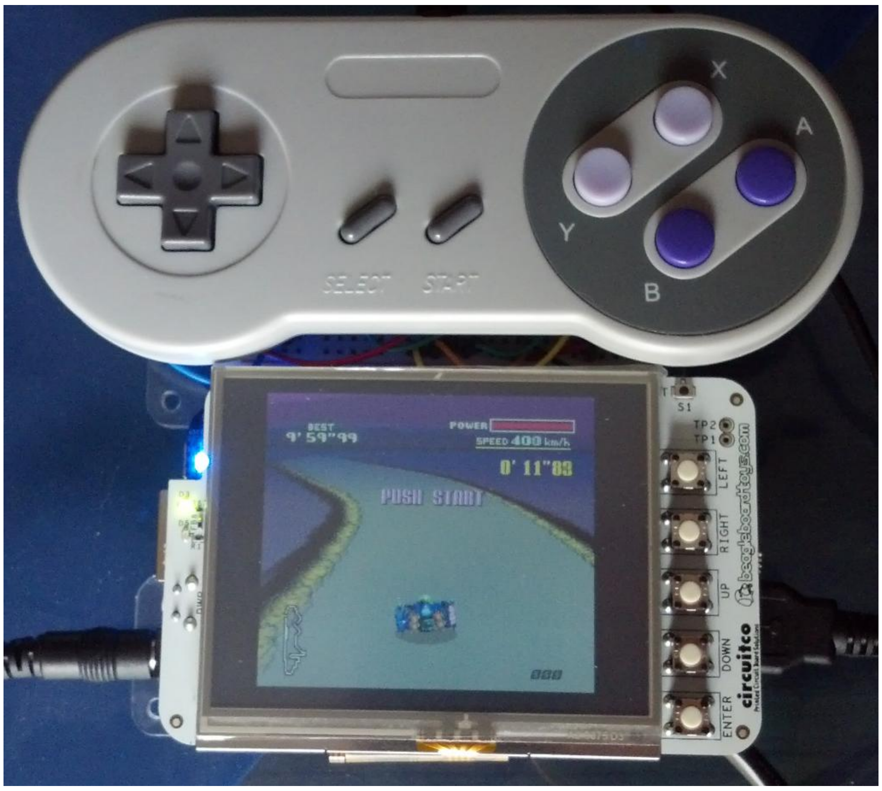


Figure 8.1: A prototype of a portable BeagleSNES system. This prototype still requires a USB gamepad and power via the +5VDC barrel connector, so it isn't fully "portable".

This target varies from the existing BBB target in the following ways:

- The LCD3 cape has a resolution of 320x240. This requires a rework of the BeagleSNES front-end GUI. On the bright side, such a small resolution allows the emulator to render at the native 256x239 SNES resolution without requiring the use of a software-based pixel-doubler to scale the SNES video output to the higher-resolutions used by the BBB and BB-xM.
- Additional audio hardware is needed. For the BBB HDMI target, a dummy CODEC in the kernel provides an ALSA interface to userspace. Audio data written to the dummy CODEC is passed to the HDMI framer chip in I2S format. The framer then packages up the audio data and sends it to the display via the HDMI protocol. When not using the HDMI framer chip, a different audio device must be provided.

8.2 Video

The LCD3 cape, seen in Figure 8.2, provides the video display functionality for a portable target. It weighs approximately 70 grams and has a current consumption of 100mA. While it is possible to use a smaller display, or at least a display module that has a smaller footprint, the easy availability and pre-packaged nature of the LCD3 cape makes it appealing for quick prototyping without having to worry about device drivers.

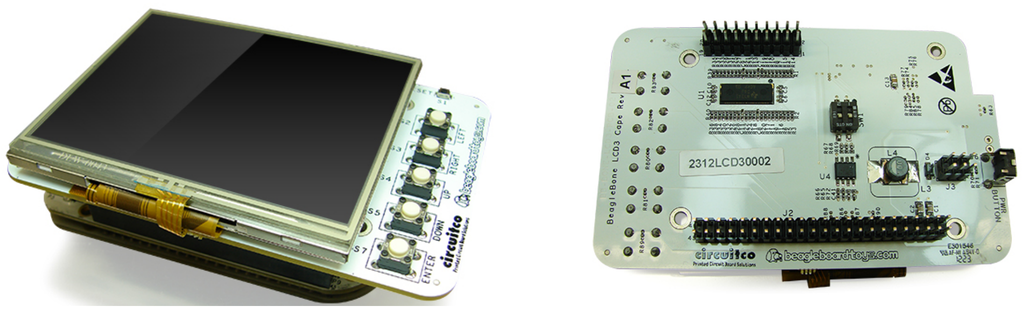


Figure 8.2: The LCD3 cape board made by CircuitCo. This 320x240 LCD has a touchscreen and five input buttons (left), and it plugs into the BBB's P8/P9 connectors via pins on the underside of the cape board (right). Photo credit: `http://elinux.org/CircuitCo:BeagleBone_LCD3`

Because the LCD3 cape is defined in the default Device Tree for the BeagleSNES kernel, it will be automatically detected and configured when it is connected to the BBB. The LCD3 uses the `TILCDC` driver in the kernel to provide a 16 BPP data bus between the BBB and the LCD3 cape. This is the same driver that is used to communicate with the built-in HDMI cape of the BBB, so userspace software need not be aware of the differences between communicating with the LCD3 cape versus communicating with the HDMI framer chip on the HDMI cape. As long as the 320x240 resolution is configured as a valid framebuffer resolution (by adding the appropriate framebuffer mode to the `/etc/fb.modes` file), BeagleSNES can simply request a 320x240 resolution and then adjust its rendering accordingly to use the LCD3 cape.

8.3 Audio

The simplest solution for adding audio hardware to the prototype is to use a USB sound device. Such devices are inexpensive (usually around 5-10 USD), quite small (as seen in Figure 8.3), and readily supported via the USB sound device driver in the kernel. The configuration to enable this audio driver in the kernel is seen in Figure 8.4.

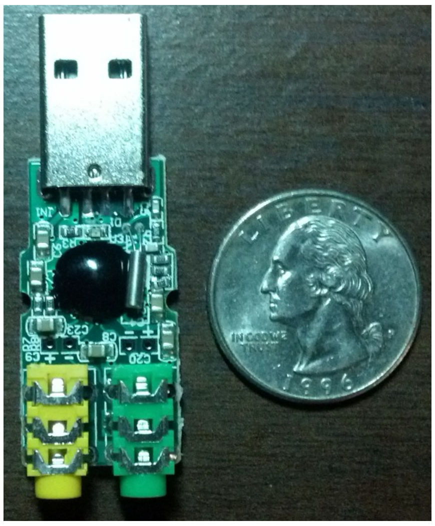


Figure 8.3: A USB audio device with its casing removed.

```
Device Drivers  --->
   <*> Sound card support  --->
      <*> Advanced Linux Sound Architecture  --->
         [*] USB sound devices  --->
            [*] USB Audio/MIDI driver
```

Figure 8.4: The hierarchy within the 3.8 Linux kernel `kconfig` configuration utility for enabling the USB audio device driver. BeagleSNES now enables this driver in its BBB kernel for both HDMI and LCD3 video output.

The BBB only has a single USB port available. Plugging the USB audio device into it is convenient from an implementation standpoint, but it is inconvenient because the device will extended beyond the footprint of the BBB and LCD3 cape. Worse yet, a USB port is still needed for the USB gamepad. To resolve these issues, a small, 4-port USB hub is plugged into the single USB port and then used to provide USB to the gamepad and USB audio device. The hub, when its casing was removed, is so small that it fits into the space between the LCD3 cape and BBB when the cape is plugged into the BBB.

Figure 8.5 shows the positioning of the USB hub and the USB audio device when both components are positioned on top of the BBB. The USB audio device is soldered directly to one port of the USB hub, allowing the audio jacks to be positioned in a convenient location that isn't limited by the physical position of the USB ports on the hub. Both the hub and audio device are wrapped in electrical tape to avoid shorting and inadvertent electrical connections. Because the LCD3 cape only uses a portion of the pins on the P8 connector, the audio jacks are positioned on the connector, next to the ethernet jack. The assembled unit with all components is shown in Figure 8.6.

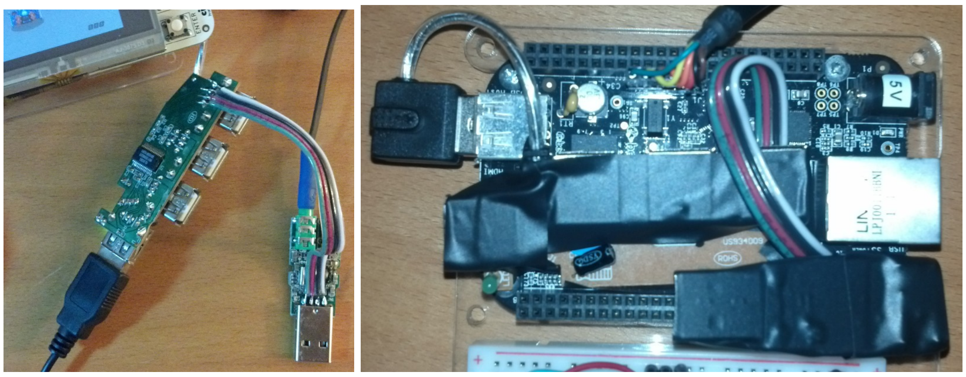


Figure 8.5: The USB hub and audio device. The audio device is soldered directly to a USB port on the hub (left). Both the hub and device are wrapped in electrical tape and positioned on the BBB.

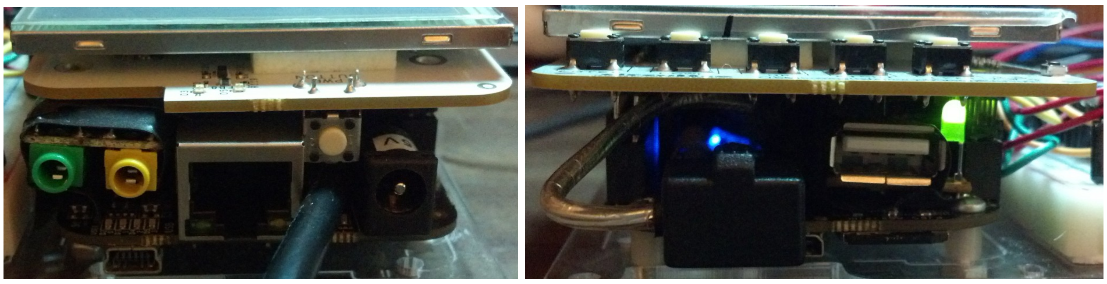


Figure 8.6: The assembled prototype. The edge of the unit with the power connector (left) exposes the audio jacks and the FTDI debug cable. The edge of the unit with the BBB USB port (right) exposes the edge of the USB hub and the USB port exposed by the hub. The slight “tilt” of the LCD3 cape is due to the height of the FTDI debug cable preventing one side of the cape from being completely inserted.

The CircuitCo Audio Cape requires the use of GPIO3[19], which is also used by the LCD3 cape. So, as the moment, there is no way of using the two together. A copy of the device tree overlay for the audio cape has been added to `/lib/firmware` so that you can experiment with the audio cape, but it won’t just “plug-and-play” like most capes will. You will have to manually load the overlay for the audio cape to use it.

8.4 GPIO Input

While requiring the user to use a USB gamepad for input is convenient for the developer, it isn’t convenient for the user. A proper implementation of a portable unit would use GPIO buttons for input and eliminate the need for the USB hub completely. Luckily, BeagleSNES supports GPIO input. This allows for a more elegant design for a portable platform, since a custom controller can now be built around the LCD and BBB to minimize the unit’s size and weight.

By default, the GPIO support for BeagleSNES on the BBB uses all of the 13 pins on P8.7 through P8.19. These pins are configured as input pins, and the `games.xml` file maps these pins to SNES controller events (see Section 3.3.5). In addition, pin P8.2 is used as a ground, and pin P9.3 is used to provide 3.3V. When 3.3V is applied to a GPIO pin, a key down event is placed into the internal application event queue. When the GPIO pin goes to ground, a key up event is triggered in the same fashion. Figure 8.7 shows an example implementation of GPIO input for

BeagleSNES. Each pin has a 10k ohm resistor in a pull-down configuration.

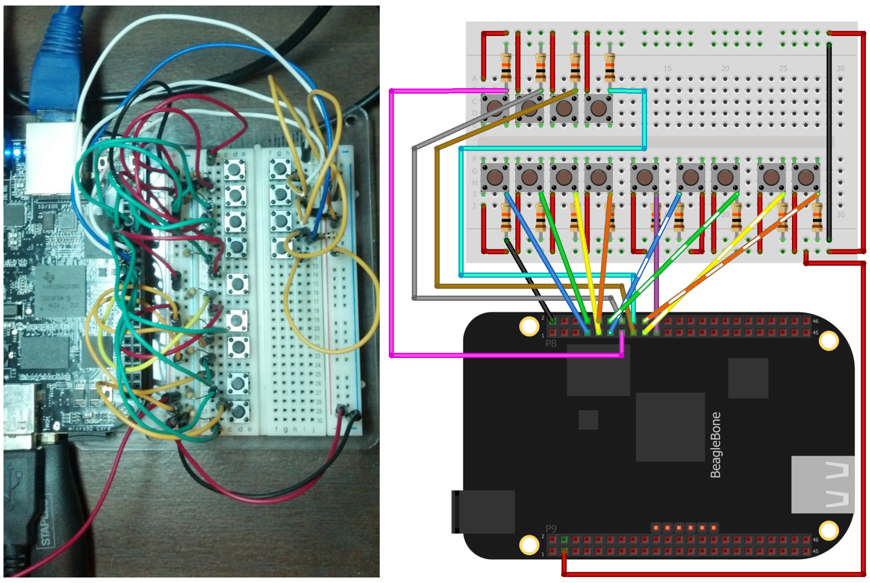


Figure 8.7: Input via GPIO. Thirteen pins are in use in the contiguous block from P8.7 through P8.19. Each pin uses a pull-down resistor configuration. These pins implement the directional pad (4), A/B/X/Y/L/R buttons (6), start/select buttons (2), and a pause button.

While most of these pins can be easily accessed while the LCD3 cape is attached, pin P9.3 is used to supply power to the LCD3. To connect to this pin, either put an intermediary "prototyping" cape in place that will give you access to the pin or snap a connector onto the P9.3 pin of the LCD3 cape to make a connection. The prototype in Figure 8.7 used a logic analyzer probe connector to grab onto the P9.3 pin of the LCD3 cape.

Always be very, very careful when working with pins on the BBB that supply voltage. While pins P9.3 and P9.4 supply 3.3V, pins P9.5 and P9.6 supply 5V. Connecting a 5V line to a GPIO is a sure-fire way to ruin your BBB, so always exercise caution!

8.5 Other Considerations

While the prototype is a good start, there is still work to be done in making a true "portable" system:

- Power is a consideration. How much battery life will you get if the emulator is maxing out the CPU? You'll still need a 5VDC, 2A power supply to power the board with the AM3359 running at maximum capacity. How heavy will this battery be? The design of the (now discontinued) CircuitCo battery cape[1] might be a good reference point to start designing from.

Ultimately, it will come down to how fancy you wish to make your portable unit design and how much money you'd like to spend prototyping it. The sky is the limit!

[1] `http://elinux.org/CircuitCo:BeagleBone_Battery`

9 — Acknowledgements

9.1 General Acknowledgements

While a project as complex as BeagleSNES can be organized and put together by one person, such an effort is only made possible by leveraging the hard work that others have already done. The following people have all contributed to the pieces that make up the foundation that BeagleSNES is built upon:

- The team at `beagleboard.org`, who have developed the BeagleBoard family of hardware platforms. Their efforts over the past several years have led to a variety of affordable, open-source hardware that is helping to encourage education and experience in embedded systems.
- Canonical (`http://www.canonical.com`), for their work on Ubuntu, which forms the base OS and file system environment of BeagleSNES.
- Robert C. Nelson (`http://www.rcn-ee.com`), whose kernel work has formed the basis of BeagleSNES's kernel. His wonderful Git repository of BeagleBoard kernel patches, as well as his numerous postings on various mailing lists and message boards, has provided a wealth of knowledge regarding the BB-xM platform.
- The SDL library team (`http://www.libsdl.org`), for their on-going work in providing a straight-forward interface to the low-level multimedia systems of both desktop and embedded platforms.
- The SNES9X team (`http://www.snes9x.com`), for their excellent, cross-platform SNES emulator.
- Yann Morin, for `crosstool-NG` (`http://crosstool-ng.org`), the cross-compiler toolchain-building scripts that have made the process of building a cross-compiler for BeagleSNES's development a snap.
- Petr Stetiar, for his `beagleboard-uboot-logo` project (`https://github.com/ynezz`) that served as a starting place for the bootloader splash screen code in BeagleSNES.

There is more to BeagleSNES than just code. The following people have also provided assistance and support during the development of the project:

- Dave Vedder at changeMode design (`http://www.chmoddesign.com`), who developed the layout and look of the game selection menu. If you need a GUI designed, I highly recommend him.
- Internet Janitor and Suspicious Dish of the Something Awful forums, who have provided

useful feedback pertaining to the game selection menu's fonts and animations.

- Mathias Legrand, creator of the Legrand Orange Book LaTeX template[1] that this document was created with.
- Elijah Hall (`http://mrkittie.newgrounds.com`),
- ... Genraltweet (`http://genraltweet.newgrounds.com`),
- ... conorstrejcek (`http://conorstrejcek.newgrounds.com`),
- ... and MarioMan94 (`http://marioman94.newgrounds.com`), the creators of several pieces of music that have served as the background audio for the BeagleSNES release video trailers.
- Robert Beagley, who has lent his surname to the cause and who has also greatly enlightened the world at large with his many philosophical musings on bacon.
- GameFAQS (`http://www.gamefaqs.com`), for their extensive collection of SNES game box images.
- The EECS department at Syracuse University (`http://www.syr.edu`), who gave guidance and support to the BeagleSNES project during its initial development as a graduate course project.

Finally, a big thank you to Nintendo®for developing the SNES hardware platform. Over 700+ game titles were released for the SNES[2], and it would be a shame if this fantastic platform faded away.

[1]`http://www.latextemplates.com/template/the-legrand-orange-book`
[2]`http://en.wikipedia.org/wiki/List_of_Super_Nintendo_Entertainment_System_games`

10 — Project Changelog

10.1 Changelog

Version 0.6 Release - 03 July 2014

General:

- Updated the documentation.

Application:

- Added GPIO input support (BBB targets).
- Added a splash screen to keep the user distracted while a background thread loads the GUI assets.
- Added support for the pause menu with snapshot load/save.
- Added gamepad and GPIO configuration support to the XML-based `games.xml` file.
- Cleaned up the emulator shutdown code so that a game can be cleanly exited and control will return to the front-end GUI.
- Changed from sync()-ing the filesystem to fsync()-ing specific files when saving snapshots, screenshots, saved games, etc.
- Fixed hi-res blitters that were smashing the stack on some games (Donkey Kong Country, Secret of Mana, Seiken Densetsu 3).
- Added a shrinking blitter to make the hi-res blitters work with a 320x240 display.
- Reenabled support for loading compressed ROMs (.zip, .gz).
- Added Expat libraries into the `configure` script.
- Cleaned up the XML parser a bit more to eliminate closing tag errors where there weren't actually errors.
- Added more aggressive optimization flags to the compile options to squeeze out a few more FPS.

File System:

- Added entries for `eth1` to `/etc/network/interfaces` for users having difficulty getting the BBB ethernet to initialize and pick up an IP via DHCP.
- Placed a device tree overlay for the CircuitCo Audio Cape (RevB) (`BB-BONE-AUDI-02`) in `/lib/firmware` in case anyone wants to try using it with the BBB.

Version 0.5 Release - 18 February 2014

General:

- Updated the documentation.

Application:

- Changed the old game selection menu configuration file from `games.cfg` to the new XML-based `games.xml` format.
- Added support for the 320x240 LCD3 display.
- Added software volume control (mostly used by the LCD3 target).
- Changed the name of the `snes9x-sdl.BBB` binary to `snes9x-sdl.BBB.HDMI` (for the BBB HDMI target).
- Added the `snes9x-sdl.BBB.LCD3` binary (for the BBB LCD3 target).
- Changed the game selection GUI to pull its button mappings from the emulator configuration file.
- Moved the configuration, ROM, and image files to the `/boot` partition.

Kernel:

- Added controller driver support for ACRUX, DragonRise, GreenAsia/Pantherlord, I-Force, Sony PS3, X-Box, and ZeroPlus.

Bootloader:

- Updated the U-Boot bootloader for all targets to v2014.01.
- Modified the `setup_BBB.sh` script so that it is now two scripts: `setup_BBB_HDMI.sh` (for the BBB HDMI target) and `setup_BBB_LCD3.sh` (for the BBB LCD3 target).
- Setup the kernel command line arguments in the `setup_BBB_LCD3.sh` script to disable the HDMI cape.

File System:

- Increased the `/boot` partition size from 64 megabytes to ~950 megabytes.
- Decreased the `/rootfs` partition size to fill the remainder of the 4 gigabyte microSD card image.
- Changed the `service.sh` script to detect whether the LCD3 cape is attached for BBB systems.
- Modified the `/etc/modes.fb` file to include a 320x240 mode for the new LCD3 display target.
- Added the Expat 2.1.0 library and headers to `/usr/local`.

Version 0.4 Release - 07 June 2013

General:

- Reduced boot time from 20 seconds down to about 10-12 seconds in "fast boot" mode.
- Updated the documentation.

Application:

- Added support for two gamepads via an external USB hub (BBB).
- Added support for an arbitrary number of games to be listed in the game selection menu.
- Removed the user space workaround that adds USB hotplugging (BBB).

Kernel:

- Updated the kernel tree from 3.8.11 to 3.8.13 (BBB).
- Patched the kernel to fix issues with USB hotplugging and "babble interrupts" (BBB).

Bootloader:

- Added the kernel command line `init=` option in `uEnv.txt` to use `init.sh` as the `init` process of the system for "fast boot" mode.

File System:

- Added the `init.sh` script to act as the new `init` process of the system.
- Modified `init.sh` to mount various file systems, start the `udevd` daemon, and launch BeagleSNES's `service.sh` script.

Version 0.3 Release - 17 May 2013

General:

- Added support for the BeagleBone Black platform.
- Added `setup_BBB.sh` and `setup_BBxM.sh` scripts in the boot partition to switch the full system image from BBB to BB-xM and back.
- Made BBB the default state of the full system image.
- Updated the documentation.

Application:

- Modified the graphics to support 720x480 (BBB) and 640x480 (BB-xM) based upon preprocessor defines.
- Added support the dynamic plugging/unplugging of gamepads for BBB.
- Fixed a timing bug where the selection GUI might freeze.
- Fixed a bug where SRAM (saved games) was not being written out to the microSD card. Also added a `sync()` call on SRAM activity to ensure saved games are stored on the microSD card.
- Modified the `service.sh` launch script to detect whether the system is BBB or BB-xM and launch the appropriate binary.

Kernel:

- Added a new Linux kernel tree (3.8.11) to support the BBB. The BB-xM is still using the 3.7.10 kernel tree.
- Added a 720x480 kernel splash screen for the BBB port.

Bootloader:

- U-boot has been upgraded from v2013.01 to v2013.04.
- The `beaglesnes_build.sh` build script has been changed to have an additional configuration target for the BBB.

File System:

- Both the BBxM and BBB kernels are in the `/boot` partition. `BBB` and `BBxM` subdirectories were added to hold platform-specific bootloader and `uEnv.txt` files.

Version 0.2 Release - 23 April 2013

General:

- Changed from analog video (S-Video) to digital video (DVI).
- Decreased boot time from about 25 seconds down to about 20 seconds.
- Improved gamepad support.
- Added a kernel splash screen.
- Published the first version of this documentation!

Application:

- Changed resolution of game selection menu from 720x482 to 640x480.
- Changed resolution of emulator from 616x478 to 640x480.
- Added support the dynamic plugging/unplugging of gamepads.
- Removed several checks for movie playback and saved states from main emulation loop.
- Changed the internal emulator scaling algorithm from "Simple2x1" to "Smooth2x2" to accommodate for the increased sharpness of the digital video signal.
- Moved the loading of background music and the rendering of dynamic text so that they occur after the BeagleSNES logo and gradient bar are loaded and rendered to the screen. This gets *something* up on the screen ASAP for the end-user.

Kernel:

- Removed video encoder (VENC) support (S-Video support removed).
- Added parallel display interface (DPI) support, generic DPI panel driver, and TFP410 DPI-to-DVI chip driver (DVI support added).
- Built SMC9XXX network support directly into the kernel, rather than as a module. Now no kernel modules are dynamically loaded.

- Added a 640x480 splash screen displayed during kernel bootstrapping by replacing the 80x80 tux boot icon with the splash image.
- Removed a bunch of device driver modules from the build, since they weren't actually being used anymore.

Bootloader:

- Preliminary support for a bootloader splash screen was added (`dss_init()` in `beagle.c`).
- Changed `mpurate` from `auto` to `800` in `uEnv.txt` to force the kernel to set the clock to 800 MHz immediately, rather than wait for the governor to adjust it.
- Cleaned up a bad PLL4 clock register setting that was preventing logging in through the debug serial output (`beagle.c`).
- Changed the video configuration in `uEnv.txt` to use digital 640x480@60 Hz (DVI) instead of analog 720x482@30 Hz (NTSC).

File System:

- The `service.sh` launch script for the BeagleSNES application is now started via the script `/etc/rc.local`, rather than as an "at reboot" `cron` job in `crontab`. This gets the game selection menu up and available much quicker (prior to `tty` initilization).
- The `loadcpufreq`, `cpufrequtils`, and `ondemand` services were removed from all `/etc/rc?.d` directories.
- The rootfs file system was shrunk down from 7.5 gigabytes to 3.5 gigabytes.

Initial Release - 15 March 2013

- Registered the `beaglesnes.org` domain name.
- Created the BeagleSNES SourceForge project.
- Created the project launch trailer video.
- Cleaned up the files and made them available on the web.