Embedded Networking and Hardware-Accelerated Graphics with Satellite CCRMA

Edgar Berdahl
CCRMA
Stanford University
Stanford, CA, USA
eberdahl@ccrma.stanford.edu

Spencer Salazar
CCRMA
Stanford University
Stanford, CA, USA
spencer@ccrma.stanford.edu

Myles Borins
CCRMA
Stanford University
Stanford, CA, USA
mborins@ccrma.stanford.edu

ABSTRACT

Satellite CCRMA is a platform for making embedded musical instruments and embedded installations. The project aims to help prototypes live longer by providing a complete prototyping platform in a single, small, and stand-alone embedded form factor. A set of scripts makes it easier for artists and beginning technical students to access powerful features, while advanced users enjoy the flexibility of the open-source software and open-source hardware platform.

This paper focuses primarily on networking capabilities of Satellite CCRMA and new software for enabling interactive control of the hardware-accelerated graphical output. In addition, new results are presented showing that the Satellite CCRMA distribution allows the lifespan of the flash memory to be greatly increased in comparison with other embedded Linux distributions. Consequently, we believe that embedded instrument and installation designers will prefer using Satellite CCRMA for enhanced long-term reliability.

Keywords

Satellite CCRMA, embedded musical instruments, embedded installations, Node.js, Interface.js, hardware-accelerated graphics, OpenGLES, SimpleGraphicsOSC, union file system, write endurance

1. INTRODUCTION

Many recent NIME projects have been constructed around sensor circuits interconnected with a more powerful computer using an Arduino or other microcontroller. This strategy combines the flexibility of programmable hardware with the computational power provided by a laptop or desktop computer [3]. However, projects built using this kind of platform have the tendency to "die" with time. Some factors leading to the death of projects include forced software updates, changes in future iterations of hardware interfaces and/or cables (e.g. RS-232 serial to USB), etc. These factors are of course avoidable but may require maintenance resources that might not be readily available in the future when needed.

The aim of the Satellite CCRMA project is to enable makers to use a convenient, compact, and "embedded" computer in place of a laptop. Satellite CCRMA has been made feasible by the recent availability of relatively inexpensive embedded Linux computers. The *Satellite CCRMA distribution* is a distribution of Linux that is specially configured for interactive media. It contains an optimized Linux kernel, pre-installed audio and video software, a series of scripts to make it easier to use for beginners (see Section 4), and a special feature to preserve the life of the memory (see Section B). The Satellite CCRMA distribution is released in the form of the *Satellite CCRMA image*, which is an image of a flash memory card that can be used to boot specific embedded Linux boards. Currently the Beagle Board xM and the Raspberry Pi Model B (512MB) are supported.

The *Satellite CCRMA kit* is used for teaching workshops and courses. It consists of a sandwich holding an embedded Linux board, an Arduino, and a breadboard. Figure 1 shows how the kit could be employed during a three-hour workshop setting: a user squeezes a force-sensing resistor (FSR) that is installed into the breadboard, and sound is synthesized in headphones (or small portable loudspeakers [3], not shown here).

NIME'13, May 27 – 30, 2013, KAIST, Daejeon, Korea.

2. EMBEDDED PROJECTS

Embedded projects offer several advantages [3]. For example, they are self-contained, which makes them more convenient to use [2]. All required cables can be left connected, so they can be demonstrated at a moment's notice simply by taking them off of the shelf and powering them up. In a limited sense, the embedded projects could be considered to be "living" since they are self-sufficient [4]. If appropriate, projects can be kept disconnected from the Internet, so their software will never require updating in order to continue functioning. We believe that this feature is particularly important for instrument makers, who may require years of practice in order to become virtuosos on the new instruments that they design.

The Satellite CCRMA kit can be similarly employed for making installations. The kit can be left in installation spaces without fear of theft, as would often be the case if leaving a laptop or smartphone instead [3]. Also, because the kit components are becoming less and less expensive, significant numbers of kits can be connected together in a network installation without the cost becoming prohibitive.

3. REDUCED COST

3.1 Raspberry Pi

In order to reduce the cost of the kit, the Raspberry Pi Model B (512MB) embedded Linux board is now supported. Table 1 gives the current cost of the essential parts that can be obtained by shopping around. Prices will fluctuate over time, but currently the kit parts can be purchased for as little as $68 USD before taxes.

It is important to point out one drawback from the Rasp-

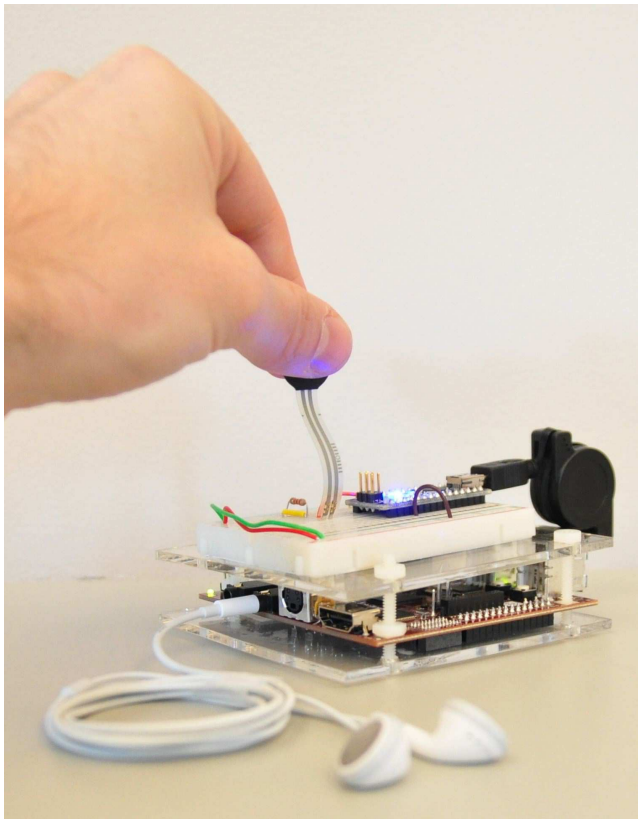

Figure 1: Simple demo of the *Satellite CCRMA kit* with a single force-sensing resistor (FSR) installed onto the breadboard in a workshop setting.

Table 1: Minimum cost of the essential parts in kit

Part	Price
Raspberry Pi	$35
Memory card	$5
Power supply	$6
Ethernet cable	$2
Breadboard	$5
Arduino Nano clone	$12
USB cable	$3
Subtotal (without Arduino-related parts)	**$48**
Total (with Arduino-related parts)	**$68**

berry Pi Model B (512MB) board, which is that the audio quality is not CD quality. Some users will prefer to use the Beagle Board xM board instead or to employ an inexpensive external USB audio interface such as the Guitar Link UCG102 (2 channels), which can be convenient because it incorporates 1/4" audio connectors, or the SIIG USB SoundWave 7.1 or equivalent Linux-compatible 7.1 interface with mini 1/8" connectors (2 inputs, 8 outputs).

3.2 Living Prototypes

Due to the low-cost of creating self-sufficient projects, it becomes more feasible for designers to create libraries of "living prototypes." In other words, designers can keep all older prototypes operational. This can be particularly helpful in the process of iterative design or evaluation by external people. Even years after its creation, a prototype can be demonstrated at a moment's notice, simply by taking it off of the shelf and powering it up.

3.3 International Workshops

Furthermore, due to the newly reduced cost of the Satellite CCRMA kit, it is more feasible to teach workshops at international conferences. For instance, we are teaching a workshop at NIME 2013 on building embedded musical instruments and embedded installations with Satellite CCRMA.

4. USABILITY

Most users program Satellite CCRMA by logging in over an Ethernet connection from a laptop [3]. For instance, to program in the Pure Data (pd) language, a user forwards the graphical user interface (GUI) window over the Ethernet connection. In other words, the embedded application runs on the Satellite CCRMA kit, but the user programs it from the laptop (see Figure 2).

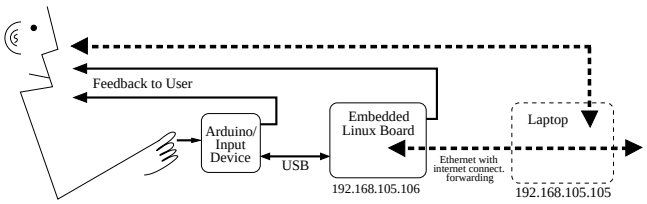

Figure 2: Standard configuration of the Satellite CCRMA kit—the laptop is only needed during programming.

Various human computer interfaces can be employed to link a human user with Satellite CCRMA, including loudspeakers, monitors, HDMI projectors, many input devices as supported by Ubuntu Linux drivers, and/or custom input devices constructed using the Arduino [6].

A new series of scripts makes it easier for artists and beginning technical students to access powerful features without needing to understand how to use Linux. Power users can drill down further into the platform by reading the scripts to learn how they work and to extend them.

4.1 Default Network Configuration

In an instructional setting where students may have a combination of OS X, Linux, and Windows laptops, the most reliable configuration involves directly connecting each student's kit with each student's laptop. By default, the Satellite CCRMA kit's Ethernet device will be automatically set to the static IP address 192.168.105.106. Then, each student should set his or her laptop's Ethernet device to the address 192.168.105.105 (see the IP addresses listed at the bottom of Figure 2). After installing an x11 server with ssh support, each student can login using the following command:

```
ssh -XY ccrma@192.168.105.106
```

where the default password is `temppwd`.

4.2 DHCP-Based Network Configuration

Power users may prefer a Dynamic Host Configuration Protocol (DHCP)-based network configuration in which the kit and the laptop are connected directly to the same Internet router. To change the kit to run in this configuration, simply run the `ethernet-use-dhcp` script, `halt` and shut down the kit, connect it directly to the router (typically via Ethernet), and power the kit back on.[1] As the kit boots up again, it will assign itself an IP address using DHCP and should have access to the Internet (see Figure 3). The kit also runs the Avahi zeroconf daemon, which enables any users with compatible systems on the same router to connect to the kit directly using the kit's hostname:

[1]The script `ethernet-use-staticIP` sets the kit back to the original, default network configuration.

`ssh -XY ccrma@satellite.local`
The user's laptop must also support zeroconf. OS X currently ships with zeroconf support, most Linux package managers allow easy installation of avahi-daemon, and Windows users can obtain support by installing "Bonjour Print Services," a no cost application available from Apple.

With the DHCP-based network configuration, it is easier for multiple people to log into the same kit and also to control multiple kits simultaneously (see Section 5.2). In the case of multiple kits connected to the same router, the `set-hostname` script is useful for changing the kit's hostname from the default `satellite` to something distinctive. It is also advisable to change each kit's password using the `passwd` command — otherwise, students will start hacking into each other's kits, which luckily so far has only been observed at CCRMA in playful contexts.

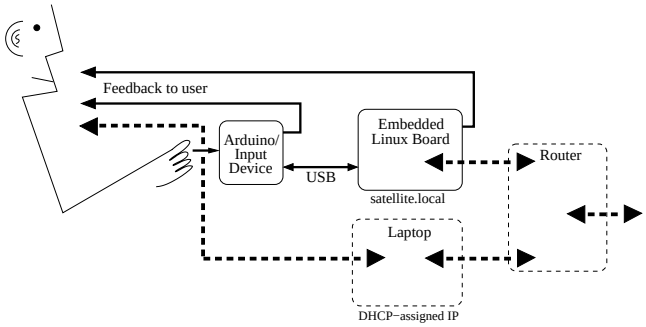

Figure 3: DHCP-based network configuration— during programming, it is necessary to connect the laptop and router.

5. THE INTERNET OF THINGS

5.1 Introduction

The "Internet of Things" vision predicts that in the future, most everyday objects will be connected to the Internet, which could have a transformative effect on how humans interact with these objects [1]. While current applications include smart/programmable control of devices' energy usage, automatic inventory control or checkout at supermarkets, etc., artistic applications will also arise, for instance in the area of social media. Due to physical constraints, only a limited number of people can interact physically with an object, but via the Internet a large number of people can interact with an object, including people at remote locations. Because Satellite CCRMA is compact, highly reconfigurable, and can be easily connected to networks and the Internet, it becomes one of the currently most ideal platforms for prototyping musical objects for the Internet of Things. In this section, the new SatelliteCCRMA.Node collection of tools is described, which can be used to this end.

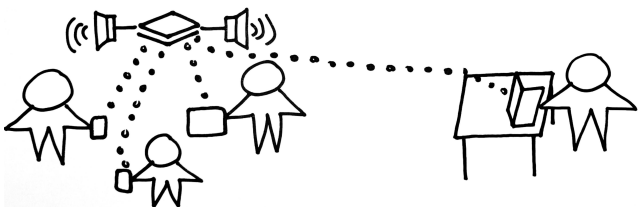

Figure 4: Users can connect to Satellite CCRMA from a variety of networked devices.

5.2 SatelliteCCRMA.Node

The latest distribution of Satellite CCRMA comes bundled with Node.js, a JavaScript environment for developing scalable Internet applications [11]. Node.js is based on Google's V8 engine and offers a unique approach to rapid prototyping of web-driven interactions. These web technologies and standards allow users to extend control to a wide array of external devices (see Figure 4) without requiring users to write platform-specific code.

5.2.1 The Power of Sockets

The WebSocket standard allows data to be piped directly from a web server to a client and vice versa over a full-duplex channel. This enables the server to easily display the current state of the system to the client as well as monitor client inputs, even on multiple external devices in real time. This technology is ideal for creating social media interactions in which users can interact with each other in the real world and also with Satellite CCRMA.

5.2.2 Controlling Sound From Network Clients

Communication between Node.js and many sound synthesis environments is simplified by `node-osc`, a library for sending and receiving Open Sound Control (OSC) messages within a Node.js program. Example code is included on the Satellite CCRMA image showing users how to encapsulate client interactions such as touches, clicks, and device orientation into OSC messages for parsing by sound synthesis environments, such as Pure Data, ChucK, SuperCollider, and many more.

For instance, `Interface.js` is a fully functional web application offering a simple OSC interface to parse the state of five multi-touch fingers and three separate axes of accelerometer data. The application works on any device that conforms to w3c standards, and it been tested with both Android and iOS. Figure 5 shows a user causing OSC messages to be sent on SatelliteCCRMA.Node by interacting with a Nexus tablet running a web browser. A video demonstation can be found at
`https://secure.vimeo.com/59331713`

Figure 5: Multi-touch interaction with a Nexus tablet controlling sound synthesis on Satellite CCRMA via `Interface.js`.

6. ACCELERATED GRAPHICS

Several current embedded prototyping platforms incorporate advanced graphics processing units (GPUs) into their designs for rendering hardware-accelerated graphics. For instance, both the Beagle Board xM and Raspberry Pi feature such hardware, which is interfaced with an HDMI output port that can drive a monitor, projector, etc. Popular

graphics programming environments for artists and designers such as Processing [10] and GEM for Pd [5] are not presently designed to take advantage of the GPUs on Satellite CCRMA's target hardware. Such environments fall back to software rendering, all but forbidding real-time usage on these devices since it tends to use practically all of the computational power of the main processing unit while running at a substandard frame rate.

Alternatively, programming OpenGL ES code directly is straightforward and shifts the majority of the graphics computations to the GPU hardware, achieving real-time performance [8]. However, programming in OpenGL ES requires competence with C++, GLSL, and numerous details of low-level graphics rendering, which are skills that artists and musicians may not possess. For this reason, we have developed SimpleGraphicsOSC, a small program exposing hardware-accelerated graphics on embedded platforms via Open Sound Control (OSC) messages [12].

6.1 SimpleGraphicsOSC

The goal of SimpleGraphicsOSC is to provide an OSC interface to a set of primitive graphical shapes, which may be combined and transformed to display intricate visualizations. The set of primitives offered is modeled loosely after similar environments, such as Processing and openFrameworks. However, unlike those environments, SimpleGraphicsOSC runs as a separate background process. To use it, a programmer runs the `SimpleGraphicsOSC` binary in the background and sends instructions via Open Sound Control to UDP port 7000 from the desired programming environment such as Pd, SuperCollider, or ChucK.

Commands sent to SimpleGraphicsOSC indicate primitives that are to be created, destroyed or modified. Table 2 lists currently available primitives and commands. SimpleGraphicsOSC maintains a list of active objects and their visual properties, and renders these using OpenGL ES [8] at an appropriate frame rate, typically 30 frames per second. See Figure 6 for an overall outline of this system.

SimpleGraphicsOSC is provided as part of the Satellite CCRMA distribution and is also available as a standalone open-source application at:
`http://github.com/spencersalazar/SimpleGraphicsOSC`

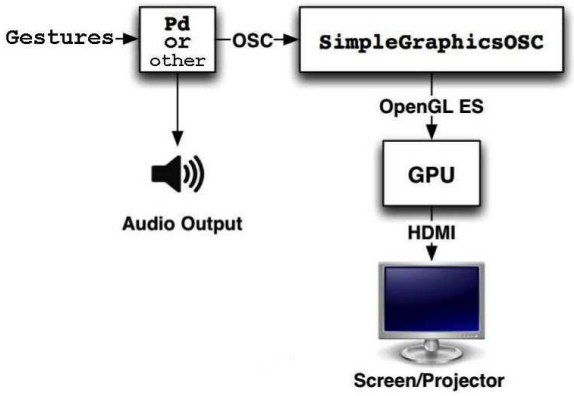

Figure 6: Components for implementing audio-visual interactions using Satellite CCRMA with SimpleGraphicsOSC.

6.2 Spectrum Overdrive

Spectrum Overdrive is a small project designed to demonstrate the unique capabilities of the Satellite CCRMA platform in tandem with SimpleGraphicsOSC. Spectrum Overdrive is a basic overdrive distortion guitar effects pedal with one twist — it displays a real-time spectrum of the overdriven signal via a small pico-projector.

Command	Description
line	Create/modify a straight line between two points.
triangle	Create/modify triangle with three independent vertices.
rect	Create/modify rectangle or square.
ellipse	Create/modify an ellipse or circle.
image	Create/modify an arbitrary PNG, JPEG, BMP, or TIFF image; the image file is loaded from the filesystem from a provided filepath.
remove	Remove a shape from the list of active objects.

Table 2: Available graphics commands in SimpleGraphicsOSC. In addition to shape geometry, each command accepts red, green, blue, and alpha/transparency (RGBA) color parameters.

Spectrum Overdrive consists of Satellite CCRMA (BeagleBoard xM and Arduino) housed within a generic project box. The configuration is further outfitted with a USB audio interface specialized for guitar input and a wallet-sized pico-projector. Two knobs mounted to the box control overdrive level and system gain, and a switch toggles the effect bypass. These controls are polled by the Arduino, which sends their instantaneous values to a Pd patch running on the BeagleBoard xM. This patch applies overdrive and gain to the input signal and computes the FFT of the result. The FFT values are transformed into SimpleGraphicsOSC commands to render each bin as a line. The height of each line is proportional to amplitude, and horizontal position is proportional to bin number, i.e. frequency, resulting in a conventional amplitude spectrum. The graphics are displayed by the pico-projector, which the user can freely move and point at various surfaces to create a light show. A video demonstration is available at the following link:
`http://youtu.be/pTgP0aL48wo`

6.3 Playback of Video Files

The `omxplayer` application on the Raspberry Pi has the ability to play video files using hardware acceleration. Currently, only H.264 and MPEG2 file formats are supported, and each user is required to buy a license for hardware accelerated playback for a small fee:
`http://www.raspberrypi.com/mpeg-2-license-key/`

For example, one can use the `shell` object in pd to call `omxplayer` from within a pd patch. In this fashion, one can build interactive installations that play video clips based on inputs from Arduino. The Satellite CCRMA image for Raspberry Pi contains a demo patch that can be modified by users.

7. STAND-ALONE MODE

7.1 Defaults

By default, stand-alone mode is *enabled*. That means whenever the kit is powered on, it boots up and automatically starts the default patch `~/on-startup/default.pd`. The default patch simply outputs a sinusoid if it does not find an Arduino connected to the embedded Linux board. On the other hand, if an Arduino is connected and has the sketch `StandardFirmata` loaded on it, then the kit will synthesize "wind" sounds, where the sound intensity depeonds on the analog input pin A0. Even if a sensor circuit is not attached to A0, the user can "play" the demo patch simply by touching the pin A0.

If the user desires to edit the default patch, the user must first login via ssh and stop the default patch from running using the `stop-default` command. Then the user can edit the default patch by loading pd with `pd &` and then opening the default patch.

7.2 Configuration

To exit stand-alone mode for future boots, the user can simply run the command `exit-stand-alone`.[2] Alternatively, to use stand-alone mode with other software, the user merely needs to edit the file `~/on-startup/load_default_patch`, so that the desired alternate software is loaded instead.

8. CONCLUSIONS

While preparing these enhancements, we have recompiled the Linux kernel many times and experimented with various compiler switches to improve the performance and incorporate bug fixes. We are happy to report that the platform operates at high quality standards for practical use in creating embedded musical instruments and embedded installations. In summary, while the Satellite CCRMA distribution is not the only way to prototype embedded instruments and installations, it currently has the following advantages over other embedded Linux distributions:

- it comes preloaded with applications for working with audio,
- audio is processed reliably even with CPU loads as high as 80% or more,
- a set of scripts makes it easier for artists and beginning technical students to reconfigure the kit,
- hardware-accelerated graphics can be easily controlled via OSC messages,
- it comes with software examples for prototyping new interactions, such as starter code for allowing users to login via http to control sound, and
- the memory card should generally last an order of magnitude longer if flash writes are disabled (see Section B).

Interested users may wish to visit a link[3] to a list of useful commands.

9. ACKNOWLEDGMENTS

We would like to graciously thank Wendy Ju, Paul DeMarinis, Chris Chafe, Cathy Wicks, Fernando Lopez-Lezcano, Carr Wilkerson, Robert Nelson, Gerald Coley, T.I., Chris Jubien, the Alexander von Humboldt Foundation, and the Satellite CCRMA user base.

10. REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[2] E. Berdahl and C. Chafe. Autonomous New Media Artefacts (AutoNMA). In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 322–323, Oslo, Norway, May 30-June 1 2011.

[3] E. Berdahl and W. Ju. Satellite CCRMA: A musical interaction and sound synthesis platform. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 173–178, Oslo, Norway, May 30-June 1 2011.

[4] E. Berdahl and Q. Llimona. Tangible embedded linux. In *Proc. Conference on Tangible, Embedded and Embodied Interaction*, Barcelona, Spain, February 10-13 2013.

[5] M. Danks. Real-time image and video processing in gem. In *Proceedings of the International Computer Music Conference*, pages 220–223, 1997.

[6] D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe. Arduino: An open electronic prototyping platform. In *Proc. CHI, 2007*, 2007.

[7] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. Sta. How I learned to stop worrying and love flash endurance. In *Proceedings of the 2nd HotStorage Workshop with the USENIX Annual Technical Conference*, Boston, MA, June 2010.

[8] A. Munshi. OpenGL ES common profile specification 2.0. *Khronos group September*, 2007.

[9] D. Quigley, J. Sipek, C. Wright, and E. Zado. Unionfs: User- and community-oriented development of a unification file system. In *Proceedings of the Linux Symposium*, volume 2, Ottawa, Ontario, Canada, July 2006.

[10] C. Reas and B. Fry. *Processing: a programming handbook for visual designers and artists*. Mit Press, 2007.

[11] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *Internet Computing, IEEE*, 14(6):80–83, Nov.-Dec. 2010.

[12] M. Wright and A. Freed. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference*, pages 101–104, Thessaloniki, Hellas, September 1997.

APPENDIX

A. RELIABILITY TESTING WITH POWER CYCLING

In order to provide the musician with a convenient vehicle for developing virtuosity, a high-quality embedded musical instrument should work for many years without requiring maintainance. One important way to assess the reliability of an embedded musical instrument is to check whether it works properly each time it boots up when being repeatedly power cycled.

Power cycling is also a particular concern for art installation designers. To save power, museums and exhibition halls typically turn off the power without warning when no visitors are there. As soon as visitors arrive, they turn the lights and installations back on. The authors were interested in verifying that Satellite CCRMA would repeatedly boot up despite unprotected power downs as well as verifying that the sound drivers started properly. At the suggestion of Paul DeMarinis, we employed the circuit shown in Figure 7, in which an Arduino board automatically turns the Satellite CCRMA kit on and off.

To equivalently turn on and off the 220V power input, the electromechanical relay MKP2-I from Rayex Electronics was employed. However, a 5V digital output from the Arduino board was not powerful enough to switch the relay, so a TIP122 was used to switch the relay on and off using a 12V DC power supply (see Figure 7). Finally, the

[2]To put Satellite CCRMA back into stand-alone mode for pd patches, the user can run the command `stand-alone-pd`

[3]`https://ccrma.stanford.edu/wiki/Useful_Commands_for_Satellite_CCRMA`

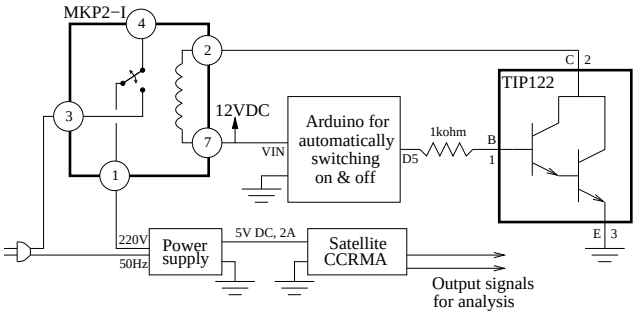

Figure 7: Power cycling circuit with relay for reliability testing.

output audio from the Satellite CCRMA kit was monitored. Even before performing the final tests with the power cycling circuit, it was easy to learn what default settings to use, for instance so that audio started properly for every single boot. For this reason, the circuit is enthusiastically recommended to others who are testing the reliability of hardware projects.

B. RELIABILITY OF FLASH MEMORY

Wear and tear of the flash memory over time is a more complex issue. Most notably, blocks on a flash memory device are reported to endure only a limited number of writes. This effect is known as "write endurance," and manufacturers are reported to provide ratings of 10,000 to 100,000 writes/block for their flash memory products. In fact, blocks may well survive more writes depending on history of the writes [7].

To prolong the lifespan of the flash memory cards, it was necessary to implement a method for disabling all writes to the flash memory. To enter this mode, the user can execute the following command:

```
disable-flash-writes
```

and reboot. Then for subsequent boots, the system starts up using the *aufs union filesystem*, allowing the flash memory to be mounted as read-only [9]. Any apparent writes are written instead to a RAM disk. The file system presented to the user consists of the *union* between the read-only flash memory and the RAM disk (see Figure 8), allowing the file system to apparently change with time. However, since these changes are written only to RAM, none of the changes persist following a reboot. This means that users should disable flash writes only after they have completely finished preparing and calibrating a project.

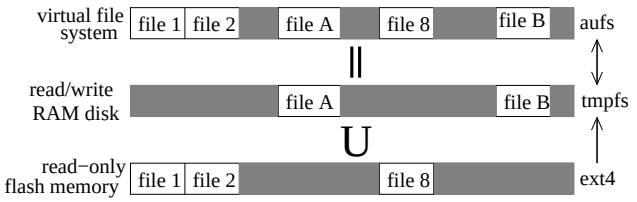

Figure 8: When flash writes are disabled, any apparent changes to the file system take place only in the RAM disk.

One further important advantage of disabling flash writes is that then it is safe to disconnect the power to the kit at any moment. Since no changes are made to the flash memory, the file system cannot become inconsistent due to unprotected power downs. This is a nice feature because then users do not need to implement a method for "shutting down" their embedded musical instruments or installations. If users change their minds later and wish to make further changes to a project, they can run the following script to enable flash writes once again:

```
enable-flash-writes
```

When flash writes are disabled, it is theoretically possible for the user to write a program that will run out of RAM because there is no virtual memory. The authors have not experienced this situation yet, as the supported boards all have 512MB of RAM. Nonetheless, concerned users can watch the `~/.memory_log` file, to which the currently occupied RAM in megabytes is appended every 15 minutes. For instance, a user could follow the memory log in real time by opening a terminal, connecting to the kit, and then running the following command:

```
tail -f ~/.memory_log
```

Table 3: Power cycling the Satellite CCRMA image (except the PdPi image for the last, starred entry)

Card type	Board	Flash writes	# boots
Sandisk 4GB	Beagle xM	*disabled*	**>6900**
Sandisk 4GB	Beagle xM	enabled	143
Samsung 4GB	Beagle xM	*disabled*	**>3900**
Samsung 4GB	Beagle xM	enabled	147
Kingston 8GB	Rasb. Pi	*disabled*	**>10000**
Kingston 8GB	Rasb. Pi	enabled	432
Kingston 8GB*	Rasb. Pi	enabled	143

C. POWER CYCLING TEST RESULTS

Table 3 demonstrates that disabling flash writes is essential for making robust projects using Satellite CCRMA. These tests were performed using class 4 flash memory cards on both the Beagle Board xM and Raspberry Pi boards using unprotected power downs. When flash writes are enabled (the default configuration), the kits only boot up successfully to run a sound synthesis pd patch for a few hundred times. Presumably during the boot process, one or more sectors are written to many times, meaning that the memory cards can fail after a relatively small number of boot cycles. In contrast, when flash writes are disabled, the kits boot up an order of magnitude more times. So far, kits for which flash writes were disabled have always booted up thousands of times. Due to limited resources for performing these tests, none of the flash-write-disabled cards have failed yet.

Clearly the feature for disabling flash writes is very useful, but to the authors' knowledge, no other embedded Linux distributions for inexpensive boards incorporate an equivalent feature in a switchable fashion. The authors would like to take this opportunity to note that the last card listed in Table 3 was loaded with the PdPi image found on the Internet for running Pure Data on the Raspberry Pi instead of Satellite CCRMA. Thus, the results in Table 3 indicate that Satellite CCRMA performs much more reliably than PdPi. The authors also verified that the PdPi image was subject to an audio dropout approximately every five minutes, whereas the Satellite CCRMA image did not drop any audio vectors when run for eight hours continuously, both when using the internal sound output and an external sound interface.

One could conceive of projects for which it is necessary to keep flash writes enabled. Satellite CCRMA handles this border case by allowing monitoring of the number of times a memory card has been booted with flash writes enabled. This record is stored in the file `~/.number_boots`.