Embedded Sound Synthesis

Victor LAZZARINI, Joseph TIMONEY and **Shane BYRNE**
Sound and Digital Music Technology Group
Maynooth University
Maynooth, Co.Kildare
Ireland,
{Victor.Lazzarini, Joseph.Timoney, Shane.Byrne.2011}@nuim.ie

Abstract

This article introduces the use of the Intel Galileo development board as a platform for sound synthesis and processing, in conjunction with the Csound sound and music computing system. The board includes an Arduino-compatible electronics interface, and runs an embedded systems version of the Linux operating system. The paper describes the relevant hardware and software environment. It introduces a port of Csound, which includes custom frontends that take some advantage of the board capabilities. As a case study, a MIDI synthesizer is explored as one of the many potential applications of the system. Further possibilities of the technology for Ubiquitous Music are also discussed, which use the various interfacing facilities present on the Galileo.

Keywords

Embedded systems, sound synthesis and processing, music programming, Ubiquitous Music

1 Introduction

The Intel Galileo board[1] (fig.1) is an embedded systems development board based on the Quark System-on-Chip (SoC), which includes an Arduino-like functionality (and compatibility with some existing extension shields and software). The board can be used as a straight replacement for the Arduino Uno boards, with a customised Arduino Integrated Development Environment (IDE) that allows programming of applications (sketches) using the Wiring library. The Galileo, however, runs under a Linux-based operating system, and thus allows other modes of application that are not restricted to Arduino IDE sketches, and which can take more complete advantage of the board capabilities.

In this article, we examine the use of the Galileo board for sound synthesis and processing, with the Csound[ffitch et al., 2014][Boulanger, 2000] sound and music computing system. We demonstrate the scalability of Csound, which has been shown to run on a great variety of platforms, from supercomputers[2] to mobile[Lazzarini et al., 2012][Yi and Lazzarini, 2012] and web[Lazzarini et al., 2014], and now on embedded systems such as the Galileo. The hardware and software combination discussed in this paper has the potential of opening up a variety of new applications for electronic music composers and performers.

As a case study, we have developed a complete software image for the system, which allows it to be booted up as an outboard MIDI synthesizer. This paper is organised as follows: we first describe the hardware and software environment that is available to Galileo developers. We then discuss the details of the port of the Csound system, and its custom frontend that takes advantage of the board's Arduino-like capabilities. This is followed by a report on our case study, the MIDI synthesizer. Finally, we propose some further applications of the technology.

2 Galileo hardware and software

Galileo boards have been produced under two slightly different hardware configurations, namely, original (GEN1, pictured in fig.1), and a revised specification (GEN2, pictured in fig.2)[3]. They generally run under custom, specially-designed, Linux for embedded systems images, created and supported by the Yocto Project[4]. The board can be booted up from the flash memory (containing a minimal/small linux image), or from the SD card, which can contain more complete operating system images.

[1] `http://arduino.cc/en/ArduinoCertified/IntelGalileo`

[2] Csound was used as part of two class C projects at the Irish Centre for High-Performance Computing (ICHEC) exploring parallel processing for audio

[3] `http://www.intel.ie/content/www/ie/en/do-it-yourself/galileo-maker-quark-board.html`

[4] `https://www.yoctoproject.org`

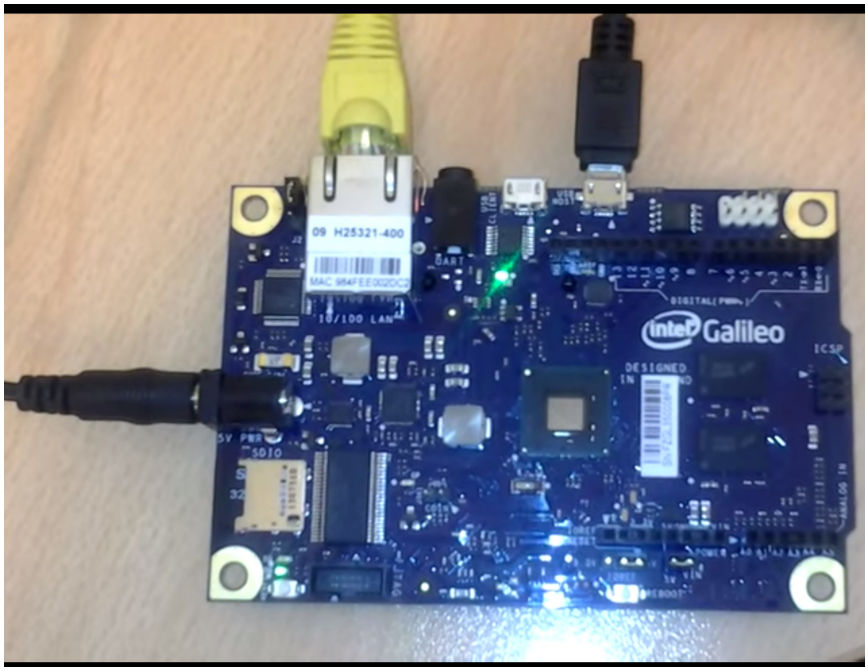

Figure 1: The Intel Galileo (GEN1) with ethernet and USB connections

2.1 Hardware specifications

The two share some basic attributes that include a Quark processor, which has the same instruction set to the Pentium, or i586, CPU, and contains a single core running at 400 MHz (also known as 'Clanton')[5], 10/100Mbit ethernet, PCI Express, USB 2.0 device and host interfaces, and microSD card reader. GEN1 boards have a 3.5 mm RS-232 connector, whereas GEN2 replaces this with a 6-pin Transistor-Transistor Logic (TTL) Universal Asynchronous Receiver-Transmitter (UART) header that is compatible with standard adaptors.

The Galileo uses the standard Arduino pin layout, which includes 20 General-Purpose Input/Output (GPIO) pins (6 multiplexed as analog inputs), plus power and Serial Peripheral Interface (SPI) headers. The hardware implementation of these is different in GEN1 and GEN2. In the former, an external GPIO expander chip (Cypress CY8C9540A) is used to control most of the pins in the shield, with only two Quark GPIOs connected directly (accessible via a multiplexer switch). The latter has 12 Quark GPIOs fully accessible to the headers, and uses a different GPIO expander chip layout (3 NXP PCAL9535A), mostly to control multiplexing (leaving eight available for input/output functions). The Quark GPIOs allow faster switching performance, through a dedicated software interface. Analog IOs are implemented in GEN1 via an Analog Devices AD7298 ADC IC, providing 12 bits of resolution, and in GEN2 via a Texas Instruments ADS108S102 IIO-ADC, which is 10-bit (scaled to a 12-bit range for compatibility purposes). Pulse-width modulation (PWM) is also implemented differently on the GEN2 board, providing higher resolution.

[5] http://ark.intel.com/products/79084/Intel-Quark-SoC-X1000-16K-Cache-400-MHz

2.2 Software systems

The board is generally run under a specially-built Linux OS image, although a Debian-based system has also recently been tested, and Microsoft has also provided a cut-down version of Windows 8 for it. We will discuss here the original Linux software that has been designed for the board. There are two types of Linux images that are used in the Galileo, based on different versions of the standard C library. The smaller image, mostly meant to be run from the limited space in the board flash memory, is built with uClibc. This library was originally designed for embedded Linux systems not using memory management units, but also runs on standard Linux. The other type of image is based on eglibc, which was designed for embedded systems but is generally compatible with the standard glibc. This library is more suit-

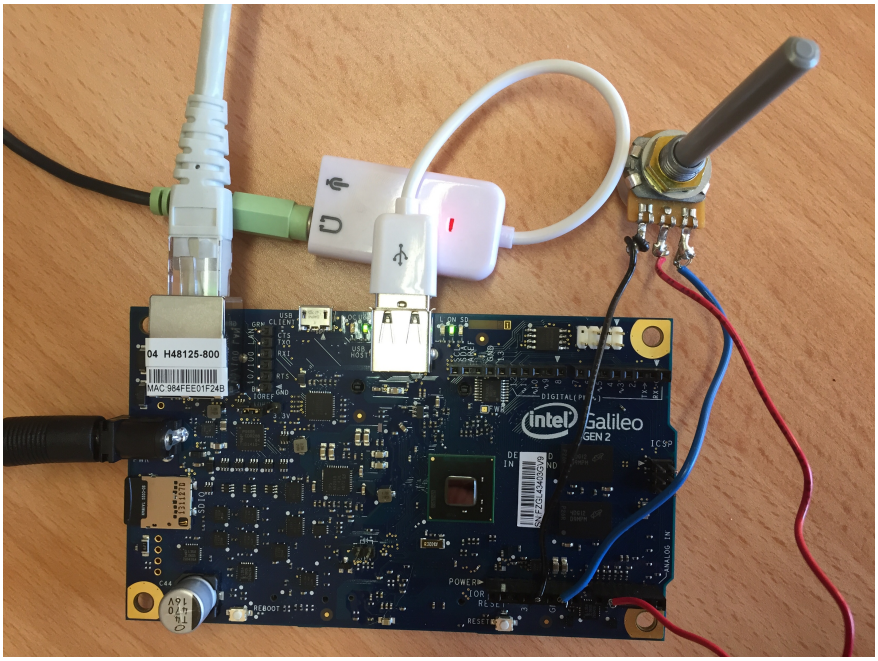

Figure 2: The Intel Galileo (GEN2) with ethernet, USB audio and a potentiometer connected to analog input 1 (pin A1)

able for SD card-based installations with no size constraints, as it is more fully-featured and provides better performance than uClibc. Software built with the Arduino IDE normally depends on uClibc, and therefore will not run on an eglibc image (although the Arduino IDE can be rebuilt from source to target this).

2.2.1 Development environment

Although it is possible to include all the development tools and use the board itself to build software, it is more advisable to set up a cross-compiling toolchain on a host computer. This is done by building an image and the toolchain from the sources, through a Linux Board Support Package (BSP) provided by Intel for the Clanton platform. The BSP is a collection of scripts (shell scripts, python scripts, recipes, etc) built with the resources provided by the Yocto Project, that allows us to build full Linux-based operating systems for specific embedded platforms. It uses the `bitbake` tool to collect all the information in the various scripts, download from sources, patch, build, and install the operating system software. The Galileo Yocto BSP can be used to build a fully-functional eglibc-based Linux standard base distribution, and a Software Developer Kit (SDK) containing a gcc/g++ toolchain. Most importantly for us, this Linux image contains the alsa library, and with it, we can access soundcards connected via the USB or PCI Express interfaces.

3 Csound for the Galileo

A fully-functional port of Csound was built for the Galileo board using the cross-compilation environment described above. The only two basic dependencies for Csound are libsndfile (for soundfile access) and the ALSA library (for realtime audio and MIDI). Although the image built with the provided BSP contains both, there are still a few issues to be resolved before we can build the system. Firstly, the cross-compilation environment installation does not appear to include the ALSA headers, so these need to be copied manually from the sysroot in the Yocto build to the installed toolchain sysroot.

Secondly, the libsndfile originally provided by the Yocto build is broken, as it depends on large file offset support that is not provided as standard by the system. So we have to modify the bitbake build recipe (provided in `./poky/meta/recipes-multimedia/libsndfile/libsndfile1_1.0.25.bb`) to configure the build with `-D_FILE_OFFSET_BITS=64`, and rebuild the image and SDK. With this in place, we can proceed to build Csound in the usual manner, using the CMake tools.

3.1 Custom frontends

In order to access the basic Arduino-like functionality of the Galileo, specific frontends were developed: `gcsound` (GEN1) and `gcsound2` (GEN2). This functionality can be divided into two groups: access to analog inputs, and access to the GPIO digital input and output. Such connections to the pins on the board is accomplished via the Linux Sysfs interface. This provides access to GPIOs via a number of files under `/sys/class/gpio` (for digital IO) and `/sys/bus/iio/devices/iio:device0/`. GPIOs need to be exported first by writing their number to the `/sys/class/gpio/export` file, and their direction ("in" or "out") needs to be written to `/sys/class/gpioN/direction`, where N is the GPIO number. Then its value (0,1) can be read/written to `/sys/class/gpioN/value`. The analog inputs can be accessed by reading the `/sys/bus/iio/devices/iio:device0/in_voltageN_raw` file (values in the range 0 - 4095), where N is the analog input number (0-5). The interface expects text (ASCII) characters as it was originally designed to work with `echo` and `cat`. Note that Sysfs is a regular interface for all GPIOs, and in order to take advantage of the fast IO provided by the GPIOs directly connected to Quark, a different interface (through `/dev/uio0` and `ioctl()` calls) is required. This has not yet been implemented in the two custom frontends.

3.2 Analog inputs

The analog inputs on the Galileo board are marked A0-A6. These pins are reserved for this purpose by the `gcsound` and `gcsound2` programs, ie. they cannot be used for digital input and output (although there is software support for this function). These inputs are offered to Csound orchestra in the software bus channels named as "analog$N$" where N is the analog port number. The application can then read these channels as required (using `chnget`). Access to the analog IO is present throughout the performance, continuously, and is implemented asynchronously (ie. non-blocking).

The signal is delivered as a floating-point number normalised between 0 and 1 (corresponding to a 0 - 5V input range). For instance, to access a potentiometer connected to the A1 analog input (as shown in fig.2), we use

```
ksig chnget "analog1"
```

3.3 Digital input and output

The remaining 14 pins can be used for general-purpose digital input or output. Access is provided as requested, through blocking reading/writing operations. This functionality is implemented as new opcodes in the system:

```
ival   gpin inum
kval   gpin inum
       gpout ival, inum
       gpout kval, inum
```

where `ival` and `kval` are the GPIO values (0 or 1), and `inum` is the GPIO number.

3.4 Pins and signals

Depending on the board version (GEN1, GEN2), digital signals at the various pins are mapped to different GPIO numbers. Access to some of these require the switching of one or more multiplex controls (also identified by specific GPIO numbers). The mapping of pins to GPIOs and multiplexers is shown below for the two versions of the Galileo board.

3.4.1 GEN1 board GPIO mapping

Table 1 can be used as reference for the GEN1 board pins and GPIO numbers used. Pins 4 - 9 are directly connected, other pins will require a multiplexer selector to be set before use. Most of the GPIO sources are provided through the Cypress CY8C9540A I/O Expander; two sources connected directly to the Quark SoC are available through pins 2 & 3 (as indicated on Table 1).

For the pins that require multiplexing, the `gpout` opcode will need to be used to select the correct source before accessing the pin from that source. For instance to access the GPIO for pin 0 and set it to 1, we have to use

```
gpout 1, 40
gpout 1, 50
```

so that GPIO 40 accesses the multiplex selector, selecting the source as GPIO 50, and we then set this to 1.

3.4.2 GEN2 board GPIO mapping

GEN2 board has a significantly different mapping scheme, as it employs a different hardware setup. Most of the 14 GPIO digital IO pins are connected directly to the Quark SoC, and so they are controlled in a slightly different way. In/out direction needs to be switched on for each pin by a separate GPIO setting. Most pins are multiplexed, so they also need to be

Table 1: Pin to GPIO mapping, Galileo GEN1

pin	mux selector, value	source/function
0	40, 0	UART0 RXD (/dev/ttyS0)
	40, 1	50 (GPIO)
1	41, 0	UART0 TXD (/dev/ttyS0)
	41, 1	51 (GPIO)
2	31, 0	14 (Quark GPIO)
	31, 1	32 (GPIO)
3	30, 0	15 (Quark GPIO)
	30, 1	18 (GPIO)
4	-	28 (GPIO)
5	-	17 (GPIO)
6	-	24 (GPIO)
7	-	27 (GPIO)
8	-	26 (GPIO)
9	-	19 (GPIO)
10	42, 0	SPI1_CS (Quark)
	42, 1	16 (GPIO)
11	43, 0	SPI1_MOSI (Quark)
	43, 1	25 (GPIO)
12	54, 0	SPI_MISO (Quark)
	54, 1	38 (GPIO)
13	55, 0	SPI_SCK (Quark)
	55, 1	39 (GPIO)

switched on via another GPIO. In addition, the board has pullup/pulldown 22k resistors connected to the pins that can be switched on and off, also through GPIOs. Table 2 shows the mapping for each pin and their associated GPIO numbers

The GPIOs controlling the direction of the Quark GPIO pins are set as 0 = output, 1 = input. Note that this is not necessary for the two PCAL9535A GPIO (pins 7 & 8). The resistor GPIO selectors are set as 0 = pulldown, 1 = pullup; if they are set to input, the resistor is disconnected. These allow the voltage to lower to the ground, or to rise to the operating voltage (5V), when pins are disconnected.

Multiplex selection works as with GEN1, by accessing and setting the relevant GPIO, and if there are mulitplexers involved, both need to be used. For example, to make the onboard led (which is connected to pin 13) light up, we need to set GPIO 46 to 0 to select the Quark GPIO, then 30 to 0 (output direction), and finally set 7 to 1 (to turn it on). Using this, an instrument that is equivalent to the Arduino *blink* sketch can be written as:

```
instr blink
  kcnt init 0
  kLed init 0
  gpout 46, 0
  gpout 30, 0

  if kcnt == 100 then
     kLed = (kLed == 0 ?  1 : 0)
     gpout kLed, 7
     kcnt = 0
   endif
  kcnt += 1
endin
```

3.5 Other possibilities

Presently, the Csound frontends `gcsound` and `gcsound2` do not implement other Arduino-like capabilities which are available in the system. These include PWM output, access to SPIO and Inter-Integrated Circuit (I2C) busses, fast GPIO, and tty interfaces. It is envisaged that some of these will be explored in the near future for specific applications. For instance, we plan to take advantage of SPIO connections to provide integrated audio DAC/ADC capabilities, either via custom or commercially-available shields. Fast GPIO will be made available alongside the regular Sysfs implementation, and

Table 2: Pin to GPIO mapping, Galileo GEN2

pin	mux 1, value	mux 2, value	dir	22k res	source/function
0	-	-	-	-	UART0 RXD (/dev/ttyS0)
	-	-	32	33	11 (Quark GPIO)
1	45, 1	-	-	-	UART0 TXD (/dev/ttyS0)
	45, 0	-	28	29	12 (Quark GPIO)
2	77, 1	-	-	-	UART1 RXD (/dev/ttyS1)
	77, 0	-	34	35	13 (Quark GPIO)
	77, 0	-	-	35	61 (PCAL9535A GPIO)
3	76, 1	-	-	-	UART1 TXD (/dev/ttyS1)
	76, 0	64, 0	16	17	14 (Quark GPIO)
	76, 0	64, 0	-	17	62 (PCAL9535A GPIO)
4	-	-	36	37	6 (Quark GPIO)
5	66, 0	-	18	19	0 (Quark GPIO)
6	68, 0	-	20	21	1 (Quark GPIO)
7	-	-	-	39	38 (PCAL9535A GPIO)
8	-	-	-	41	40 (PCAL9535A GPIO)
9	70 0	-	22	23	4 (Quark GPIO)
10	70, 0	-	26	27	10 (Quark GPIO)
11	44, 1	72, 0	-	-	SPI_MOSI (spidev1.0)
	44, 0	72, 0	24	25	5 (Quark GPIO)
12	-	-	-	-	SPI_MISO (spidev1.0)
	-	-	42	43	15 (Quark GPIO)
13	46, 1	-	-	-	SPI_SCK (spidev1.0)
	46, 0	-	30	31	5 (Quark GPIO)

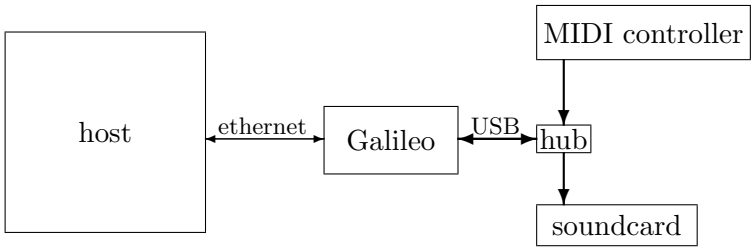

Figure 3: The Galileo Synthesizer layout

PWM will also be added to extend the output capabilities of the Galileo Csound implementation.

4 Case study: MIDI synthesizer

As a case study to assess the potential of the Galileo board for music-making, a fully-fledged MIDI synthesizer was developed. This employs Csound as the sound engine, interfacing with external USB audio and MIDI hardware. A Csound image for the board was developed, using the principles outlined in the previous sections, and including a realtime preemptive kernel. This image can be simply copied into any microSD card compatible with the board (sizes between 2 and 32GB). It will then be ready to use. It contains the standard Csound command-line frontend `csound`, the custom frontends `gcsound` and `gcsound2`, the Csound library (plus some plugin opcodes),

and the basic MIDI-based orchestra (midisynthesizer.csd).

These files are located in the `/home/root` directory. On boot up, the board will start a Csound process and load midisynthesizer.csd. This is set to use the default audio USB card and a MIDI input device, and it contains three instruments, accessible via MIDI program change messages. The three instruments are

1. Supersaw synth: a simple design based on five detuned sawtooth oscillators[Timoney et al., 2014]. Modulation controls detuning, CC 02 controls envelope attack, and CC 03 controls envelope release.

2. Pluck string: a Karplus-Strong-like[Karplus and Strong, 1983] instrument. Modulation controls brightness, CC 02 controls envelope attack, and CC 03 controls envelope release.

3. Voice: a ModFM[Lazzarini and Timoney, 2010] formant synthesizer. Modulation controls vowel types, CC 02 controls attack, and CC 03 controls envelope release.

Programs are set circularly to these three instruments (PGM mod3), and the synthesizer works in multimode, ie. it is possible to assign different programs to different channels. Although in this case study, we did not explore the possibilities offered by the Arduino-like electronics interfaces, these can be easily incorporated in the synthesizer design.

4.1 Testing the synthesizer

There is only a single host USB port on the board (there is also a client USB port, but this is used to connect to it from a host computer), so to use both an audio IO card and a MIDI controller, a hub is required. We tested the Galileo Synthesizer with a dynex USB hub, to which an M-Audio Oxygen 25 keyboard, and Behringer U-Control audio interface were connected. The synthesizer is generally very responsive, with low-latency audio output. Depending on the instrument, different polyphony limits apply. The vocal synthesizer is monophonic, but both the Supersaw and the Pluck string instruments allow up to six concomitant voices.

A video showing the synthesizer in action can be seen in

`http://youtu.be/ogYdJsKKxJk`

4.2 Connecting to the board

The board is fully accessible via `ssh`, through the use of a DHCP server (which can itself be run in the host computer) (fig.3). This can be used to debug, develop, and add new instruments to the existing ones. In this case, from the host terminal,

```
$ ssh root@<IP address>
```

In order to locate the IP address for the board, you need to have access to the network router, where you should see it listed alongside the board MAC address. You can find the board MACaddress on the top part of the ethernet socket. Once logged in, `vi` is available for simple editing. Files can also be copied to and from the board via `scp`. Host connection to the board is not required for synthesizer operation, as the board boots up into a Csound process directly. However, all USB connections should be present before booting the system.

5 Further applications

The technology described in this article has significant potential for further exploitation, beyond the simple case study discussed above. In particular, it has a direct application as a platform for Ubiquitous Music research and practice[Keller et al., 2015].

5.1 Portable live-electronics platform

For composers who employ live-electronics rigs regularly, having small, portable devices that can be used to interact with other performers on stage is very useful. With the Galileo+Csound system, it is possible to design various concert setups, with one or more boards, to deliver sound synthesis, processing, and audio playback. Due to the small size of these devices, they can be integrated in wearable components, or into augmented mechanic-electronic instruments.

5.2 Programmable effects units

The Galileo+Csound system can also be the basis for general-purpose audio processing "boxes", as fully programmable effects units. Because of the presence of a complete music programming language, it is possible to go beyond the usual categories of effects, and include more advanced signal processing algorithms, including frequency-domain methods such as the phase vocoder and sinusoidal modelling. Custom controls can be added via the electronics

interfacing capabilities offered by the board, as well as the usual MIDI and Open Sound Control (OSC).

5.3 Internet of Things sound devices

One of the original targets for the development of Galileo is to meet the potential of the Internet of Things (IoT) concept. Because of its networking capabilities (built-in ethernet, and easy wifi implementation via a PCI card), the board can be used as a remote sound device. It can run small web servers, node.js, and similar services, which can be linked up with the Csound sound synthesis engine. In addition, Csound can work as a networked sound server, which is capable of accepting control directly via UDP messages, and/or the OSC protocol.

5.4 Low-cost cluster computing for audio

With the built-in ethernet, it is possible to design a low-cost cluster, with the use of a network switch. The custom Linux OS image described in the earlier sections of this article also includes the OpenMPI library, which is an implementation of the Message Passing Interface (MPI), a standard technology for cluster networking. With this it is possible to construct a Cluster-based Csound frontend, that takes advantage of parallel processing to implement a high-performance audio engine. Such a setup would most likely be low cost in comparison to other comparable alternatives.

6 Conclusions

This paper reported on the implementation of an audio processing system using the Intel Galileo development board and Csound, running under a customised version of Linux for embedded devices. After a detailed discussion of the relevant hardware and software environment, we have explored the porting of Csound and the development of customised frontends to access the electronics interfacing capabilities of the board. A case study based on a simple MIDI synthesizer was used to demonstrate the platform as a viable sound and music programming environment. The article concluded with a number of possible application scenarios. While we have concentrated on a specific embedded platform, the ideas and principles discussed here can be applied elsewhere. In particular, we hope to develop similar systems for the Intel Edison[6] in the near future.

7 Acknowledgements

We would like to acknowledge the support of Intel Ireland, who very kindly supplied our research group with GEN1 and GEN2 Galileo development boards.

References

Richard J. Boulanger, editor. 2000. *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press, February.

John ffitch, Steven Yi, and Victor Lazzarini. 2014. Csound on GitHub. `http://csound.github.io`.

Kevin Karplus and Alex Strong. 1983. Digital Synthesis of Plucked String and Drum Timbres. *Computer Music Journal*, 7(2):43–55.

Damian Keller, Victor Lazzarini, and Marcelo Pimenta (eds.). 2015. *Ubiquitous Music*. Springer Edition, New York.

Victor Lazzarini and Joseph Timoney. 2010. Theory and practice of modified frequency modulation synthesis. *J. Audio Eng. Soc*, 58(6):459–471.

Victor Lazzarini, Steven Yi, Joseph Timoney, Damian Keller, and Marcelo Pimenta. 2012. The Mobile Csound Platform. In *Proceedings of ICMC 2012*.

Victor Lazzarini, Edward Costello, Steven Yi, and John ffitch. 2014. Csound on the Web. In *Linux Audio Conference*, pages 77–84, Karlsruhe, Germany, May.

Joseph Timoney, Victor Lazzarini, Jari Kleimola, and Vesa Valimaki. 2014. Examining the Oscillator Waveform Animation Effect. In *Proceedings of DAFx 2014*.

Steven Yi and Victor Lazzarini. 2012. Csound for Android. In *Linux Audio Conference*, volume 6.

[6] `https://www-ssl.intel.com/content/www/us/en/do-it-yourself/edison.html`