E27radio development report

Jeroen Vennegoor op Nijhuis

December 22, 2015

Online project archive:

- `https://github.com/jevontech/e27radio-image`
- `https://github.com/jevontech/e27radio`

©2015 Jeroen Vennegoor op Nijhuis

Contents

1. Introduction **4**
1.1. What is E27radio 4
1.2. Background 4

2. Design and development **6**
2.1. System architecture 6
2.2. Software overview 7
2.3. Feasibility assessment 7
2.4. Software development 9
2.4.1. Audio playback 9
2.4.2. I got the Bluez 9
2.4.3. User interface 10
2.4.4. Statemachine 11

3. Conclusion **12**

Appendices **13**

A. About the author **14**

1. Introduction

1.1. What is E27radio

E27radio is a device that makes internet radio accessible to everybody. It enables people to listen to music and news from many countries around the world. The E27 radio is extremely simple to use once it has been configured properly, so it can be used by anybody, no matter age or education.

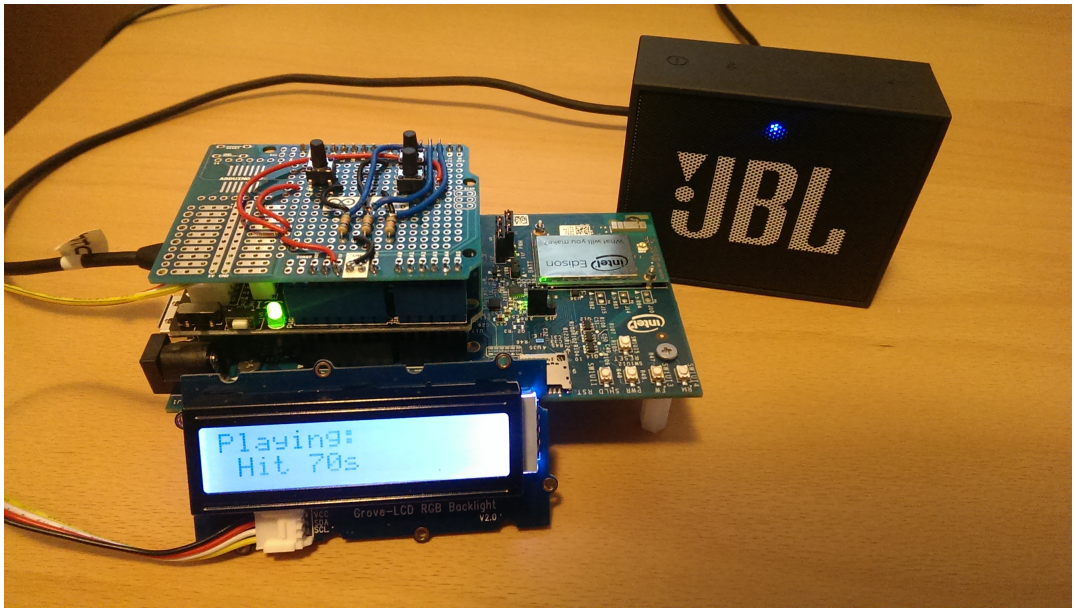

Figure 1.1.: E27 radio

1.2. Background

E27radio was developed as an entry for the Elektor Edison Challenge (end of 2015). I decided to take part in this competition in order to improve my product development skills and learn more about software development for embedded Linux. My idea was to use the Intel Edison to build a basic internet radio. Th initial plan was to use a external DAC(Digital to Analalog converter) in order to create an "analog out" signal that could be input for any audio amplifier. I ordered a Audio Codec Board from MikroElektronika. It connects to the Intel Edison using I2C for configuration and I2S for the data. Right from the start I ran into problems trying to use the I2C and I2s bus. I don't own a logic analyzer, so it was really difficult to see what was actually going wrong on the I2C bus. I gave up after a couple of weeks...

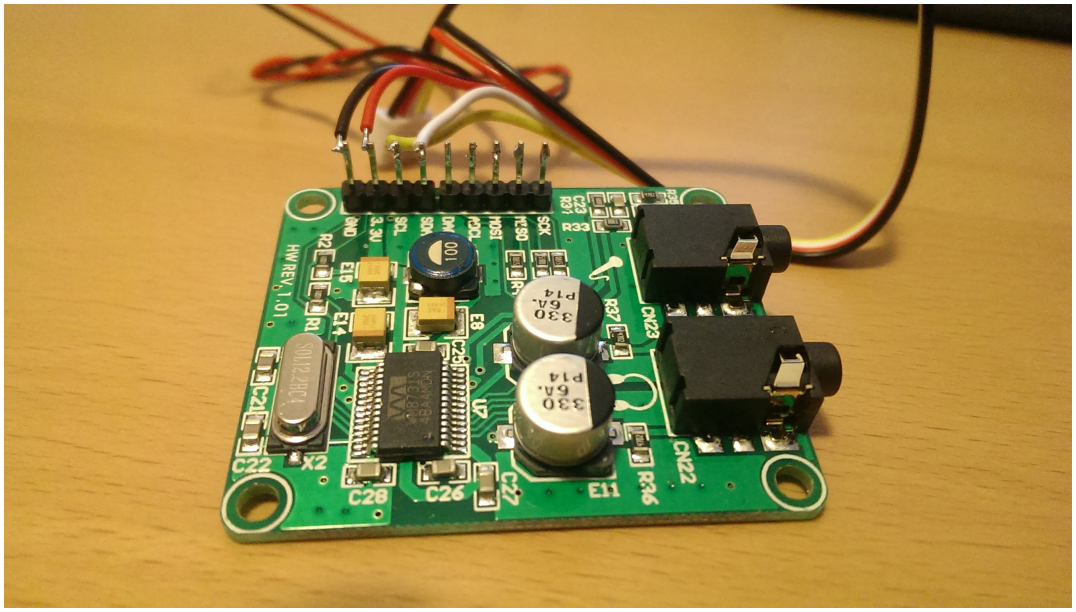

Figure 1.2.: Audio Codec Board

I then decided to take a step back and rethink the whole project. As the Intel Edison features both WLAN and Bluetooth, I might as well use both in this project. So I decided to build an internet radio that actually streams to a Bluetooth speaker. This introduced a lot of new challenges , as you can read in the rest of this report. I decided to call the project E27radio..

2. Design and development

2.1. System architecture

E27radio is build on the Intel Edison platform. It uses the onboard Wifi to connect to the Internet , so it can connect to Internet radio stations around the world. Audio output is provided by connecting via Bluetooth to an external speaker. The user interface consists of a small LCD and a number of push buttons

The application software runs on Yocto Linux (the default for Intel Edison) and is written in C++.

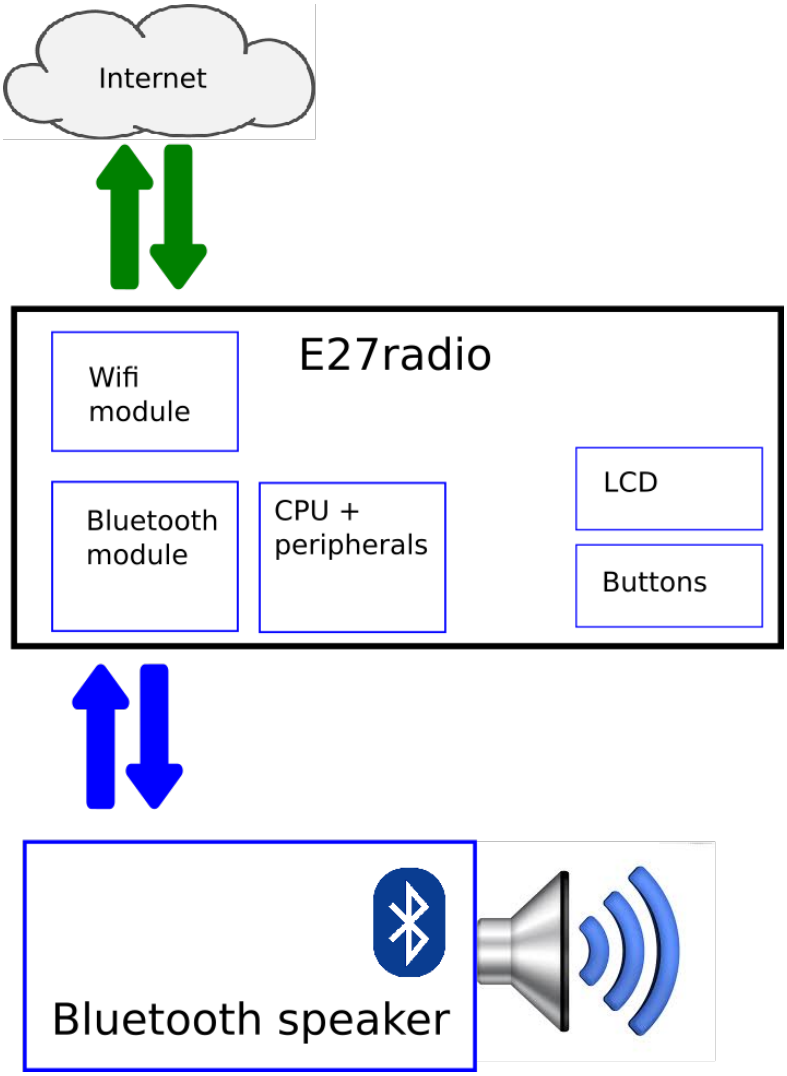

Figure 2.1.: System architecture

2.2. Software overview

In order to play audio streams via Bluetooth to external speakers, several software functionalities are required. Fortunately there is a lot of open source software out there: we just need to find all the bits and pieces and then integrate them in to 1 application.

First of all we need to be able to connect to an Internet radio stream and decode it. This requires a audio player that is able to decode most common Internet radio audio streams (e.g. MP3). For this I chose "mpg123" (`http://mpg123.org/`). Another set of tools is required to actually push this audio stream to a Bluetooth speaker. After some research on the Internet, I decided to go for a setup using PulseAudio. For the actual Bluetooth setup and configuration, a Bluetooth stack is required. For Linux, this is Bluez (`http://www.bluez.org/`). Bluez is an open source project and exists for quite some time already, but it has almost zero documentation and even less example code. So, that will be a challenge !

2.3. Feasibility assessment

I did a small test with shell scripts and commandline tools to assess whether it is possible to play an audio stream from the Internet to a Bluetooth device using the Intel Edison. For this project I am using a JBL portable Bluetooth speaker.

Figure 2.2.: Bluetooth speaker

In order to play audio, I decided to use 'mpg123' . Unfortunately this software

is not available in the official Intel Edison opkg repository. There is an unofficial Intel Edison repository by AlexT (`http://alextgalileo.altervista.org/`) and this does include 'mpg123'. But unfortunately this software is build to use Alsa as the default audio sink and not Pulse.

Thus I decided to set up my own BitBake environment. I downloaded the Intel Edison image source from the Intel Edison website and followed the instructions to build the default edison image on my Linux Ubuntu 14.04 PC. The initial bitbake takes a lot of time (it builds everything from source) and a LOT of space (you will need several GigaBytes). Once this toolchain worked, I added the recipe for 'mpg123' to the bitbake configuration and I made sure that 'mpg123' uses Pulse audio as default and not Alsa. I could then create a new image for the Edison and flash it on to the board. Once the new image was running on the Intel Edison, I configured the Intel Edison to connect to my local WiFi network(see the Intel Edison website for more information). But we are not done yet: we still need to connect the Intel Edison to the Bluetooth speaker.

Let's go through the manual Bluetooth configuration step-by-step:

- Log in to the Intel Edison
- Enable the Bluetooth interface:

  ```
  rfkill unblock bluetooth
  ```

- Open the bluetooth agent command line tool:

  ```
  bluetoothctl
  ```

- Enable scanning for Bluetooth devices

  ```
  scan on
  ```

- Pair the Intel Edison with the Bluetooth speaker:

  ```
  pair xx:xx:xx:xx (the address of the discovered BT speaker)
  ```

- Connect to the Bluetooth audio device:

  ```
  connect xx:xx:xx:xx
  ```

- Quit the Bluetooth agent

  ```
  quit
  ```

If everything went fine, the audio device you just connected also created an "audio sink" in Pulse audio. We now need to select the sink for this Bluetooth audio device as the "default" audio sink for Pulse audio.

Let's first find it by using:

```
pactl list | grep bluez_sink
```

Look for a name that looks like:

```
bluez_sink.XX_XX_XX_XX_XX_XX_XX
```

We need to set that sink as the default:

```
pactl set-default-sink bluez_sink.XX_XX_XX_XX_XX_XX_XX
```

Now we can finally select an audio stream from the Internet and play it :

```
mpg123  http://pub1.radiotunes.com:80/radiotunes_tophits
```

This works, so the idea of playing audio via Bluetooth with a Intel Edison is feasible.

2.4. Software development

I decided to use my Linux PC for the software development in this project, because I already use it for "BitBaking" the Edison image. I first used bitbake to create a development toolchain for Linux 64bit. This toolchain is required to cross-compile the software for the Intel Edison platform. Cmake (`http://cmake.org/`) will be used as the build tool for this project: I have used it before in other projects. Cmake has the advantage that I don't need to write makefiles by hand and it should be fairly easy to switch between crosscompiling using the toolchain and building the software using bitbake. I created a "toolchain file" that provides cmake with all the information to generate Makefiles that use the compilers for the Intel Edison platform(cross compilation).

2.4.1. Audio playback

Because mpg123 seemed to work quite well in the feasibility phase I will use mpg123 in the E27radio software as well. mpg123 includes a C library that can be used by other projects, but I was not quite sure how to integrate it in my own software. I found some ispiration online (`http://hzqtc.github.io/2012/05/play-mp3-with-libmpg123-and-libao.html`) and decided to go for a similar approach with curl, libmpg123 and libao. Curl downloads the data, libmpg123 decodes it and then libao plays it. But I do need to some extra bits and pieces... The audio playback needs to run in a seperate thread in order to keep the user interface responsive when audio plackback has started. Stopping the playback was not so trivial, but it can be done by adjusting the return value in the callback function that curl is calling.

2.4.2. I got the Bluez

The software for E27radio needs to be able to connect the Intel Edison to a Bluetooth speaker, similar to what we did earlier using the 'bluetoothctl' tool. In order to do this, the E27radio software needs to 'talk' to the Bluez Bluetooth stack. This

was one of the hardest parts of the project, as there is almost no information or example code available. As far as I could tell, Bluez5(the default version in the Intel Edison image) uses only D-Bus for it's API, so I needed to learn how to use D-Bus as well. D-bus (`http://dbus.freedesktop.org`) is a message bus system for IPC and RPC. It should be cross-platform, but it is currently only used on Linux, as far as I could tell. I started looking for any D-Bus library that had a nice clean C++ API, but most D-Bus libraries are a bit outdated and C based (for example libdbus). I then found out that Qt also has a D-Bus library. The interface seemed nice and clean, so decided to use Qt. Qt is normally used for applications that have a graphical user interface, but it should be possible to use it for "embedded applications" as well.

In order to use Qt5 in my project, I needed to include Qt5 in the bitbake project for Intel Edison. This way I can build the require Qt5 libraries, so I can link my own application to those libraries. For this I started with the Qt5 bitbake layer from the OpenEmbedded project (https://github.com/meta-qt5/meta-qt5). I had to make some changes in the configuration in order to get Qt5 build for the Intel Edison platform and then I was able to successfully compile Qt5 for my Intel Edison image.

The Pulse audio sink also needs to be configured, so I created a small class that finds the 'bluez_sink' and sets it as the default sink. This shouldn't be required if the Bluetooth device has been used before, but I'll do it on every connect. In this class I just call an external executable (pactl) , because this was the fastest way to get it working.

2.4.3. User interface

The user interface for this project is very simple: it consists of

- 3 push butttons (start/stop, up, down)
- Simple LCD display

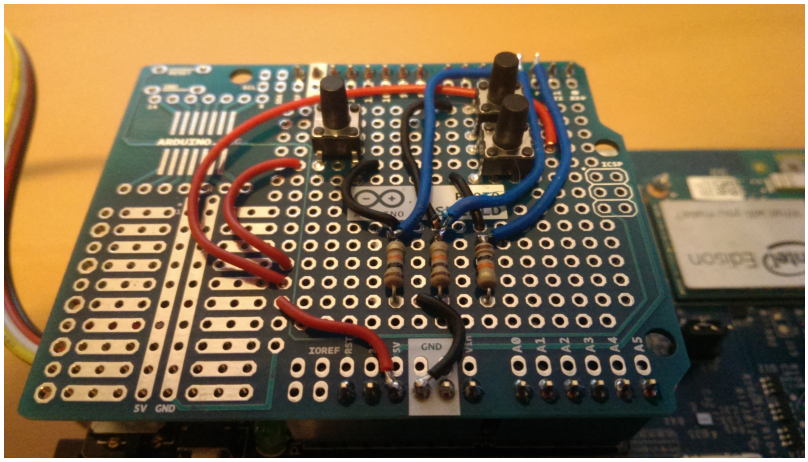

Figure 2.3.: Buttons

The 3 pushbuttons are mounted on an Arduino proto shield and are "active-high" (this means 0V signal when the button is not pressed and 5V signal when the button is pressed). In order to access GPIO in my application, I used the MRAA library. Each button is connected to a digital pin on the arduino header (pin 2, pin 3 and pin 4).In my software I am configuring these pins as input pins. Furthermore interrupts are used to handle a button press. I didn't implement any button debouncing in hardware, so I just added simple debouncing in software using a timer.

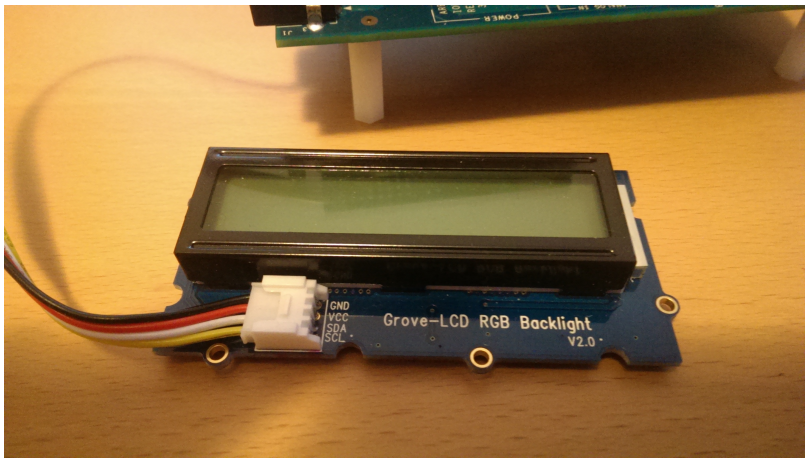

Figure 2.4.: LCD

The LCD I used is part of the Seeed Grove Starter Kit Plus. The LCD was connected to the Intel Edison Arduino board using the Grove base board. Using the LCD in software is very easy , as the driver for this LCD is already in the Intel UPM repository. The only thing I added is a very simple C++ wrapper for the existing C code.

2.4.4. Statemachine

The software now has code for audio playback and Bluetooth configuration and also code for the user interface(buttons and LCD), but we still need something to connect it all together. The "statemachine" pattern is commonly used for this type of problem. I could have build a simple state machine in C++ direcly, but I decided to use the State Machine Compiler (`http://smc.sourceforge.net/`) instead. SMC allows me to make a clean seperation between the design of the state machine itself and the implementation in code. SMC basically compiles a statemachine design (in a text file) to sourcecode in a number of languages (C++, Java, etc.). This also enables easy extensions of the sourcecode in the future.

3. Conclusion

The development of the E27radio was a bumpy road, but it works quite well in the end. I was able to complete this first basic prototype of E27-radio just in time for the contest and I already identified quite a few things that still need to be done. For example:

- Include e27radio bitbake recipe in the image
- Implement/improve exception handling in software (currently only very basic 'happy flow' is supported)
- Simplify setup procedure (currently the initial setup needs to be done via the command line , for example pairing the Intel Edison with a Bluetooth device).
- Implement extraction of metadata from Internet radio streams (for example title and artist for the current song)
- Volume control for the radio stream

I hope this project helps/inspires other people to build their own internet radio (or any other application for that matter). The Intel Edison is a nice platform for building an internet radio, as it has Wifi, Bluetooth and enough processing power. The only disadvantage is that its limited graphical processing capabilities make it more difficult to use a graphical LCD in the user interface for an Internet radio client.

Appendices

A. About the author

Jeroen Vennegoor op Nijhuis is a Dutch engineer. He lives in Eindhoven(the Netherlands) . He studied at the University of Twente and has a MSc degree in Mechanical Engineering. His professional interests and skills range from mechanical design to electronics and software. His professional experience has been mostly in R&D and the development of new products.
For more information:
LinkedIn: `https://nl.linkedin.com/in/jeroenvennegoor`