

FM Transmitter for Raspberry Pi on Secure Unix Systems

Kyle Daniel Martin

1 June 2016

PROJECT PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF BACHELOR OF SCIENCE (HONOURS) AT THE UNIVERSITY OF THE
WESTERN CAPE

SUPERVISED BY:

DR. JEAN-BAKA DOMELEVO ENTFELLNER

MENTOR:

MOGAMMAT WALEED DEANEY

Contents

Declaration	ii
Abstract	iii
Acknowledgements	iv
Glossary	v
1 User Requirements	1
1.1 Introduction	1
1.2 User's view of problem	1
1.3 Description of problem	1
1.4 Expectations of the software solution	1
1.5 Not expected of the software solution	1
1.6 Conclusion	2
2 Requirements Analysis Design	2
2.1 Introduction	2
2.2 Interpretation and break-down of the problem	2
2.3 Complete analysis of the problem	2
2.4 Other similar systems	3
2.5 Suggested Solution	3
2.6 Conclusion	3
3 User Interface Specification	4
3.1 Introduction	4
3.2 User Interface	4
4 High-Level Design	6
4.1 Introduction	6
4.2 Design diagram	6
4.3 Conclusion	7
5 Low-level Design	7
5.1 Introduction	7

5.2	Design diagram	7
5.3	Conclusion	9
	References	10

Declaration

I, KYLE DANIEL MARTIN, declare that this project titled: “FM Transmitter for Raspberry Pi on Secure Unix Systems” is my own work and that it has not been submitted before to any institution of learning for any degree and/or assessment. All the sources I have used and/or quoted have been indicated and acknowledged by means of complete reference.

Signature:.....

Date:.....

Name in print: Kyle Daniel Martin

Abstract

Frequency Modulation (FM) broadcasting is a Very High Frequency (VHF) broadcasting technology which began with Edwin Howard Armstrong. Armstrong made use of FM to provide high quality sound broadcasts over radio. The term FM band directly dictates the frequency band which is dedicated for FM transmission. Note that the term equates a FM method within a range of frequencies. The Raspberry Pi is a credit card-sized, single-board computer. The Raspberry Pi has on-board hardware that is used to generate spread-spectrum clock signals on the General Purpose Input/Output (GPIO) pins to output FM signal. My proposal is that we will achieve a solution that results in having a FM Transmitter composed solely of the Raspberry Pi, an optional but recommended passive antenna, as well as source code written in the C programming language.

Acknowledgments

I would like to give thanks to my supervisor Dr Jean-Baka Domelevo Entfellner and mentor Mogammat Waleed Deaney for their continuous support and encouragement during the duration of my Honours year. I would not be where I am today nor be able produce the work I produced without our weekly meetings and constant communication via emails.

I would also like to give thanks to my family for their constant physical and emotional support towards completing my Bachelor of Science (Honours) degree.

A thanks to Krehan “Press” Pillay for providing me with the audio files I’ll be using for the demonstration of the software.

Glossary

PWM: Pulse-Width Modulation. PWM is a method for modulating digital signals into a two-level pulsing signal with arbitrary frequency. PWM is notably used in applications controlling LEDs or stopping motors, but it may be used to emulate frequency modulation.

FM: Frequency Modulation. FM is a type of signal modulation used in radio applications. FM receivers are most widespread worldwide.

GPIO pins: General Purpose Input/Output pins. These are the 26 of the 40 physical pins that run along the edge of the Raspberry Pi. These pins materialize a programmable physical interface between the Pi and the outside world.

DMA: Direct Memory Access. DMA is a feature of most modern computers including the Raspberry Pi that allows certain systems to have direct access to main system memory without issuing memory read/write instructions through the CPU.

CPU: Central Processing Unit. The “brain” of the Raspberry Pi. This hardware is used for communication between the Raspberry Pi hardware, executing processes and performing calculations. The Raspberry Pi has a Broadcom system-on-a-chip based on an ARM CPU.

SoC: System-on-a-chip or System on Chip is an integrated circuit, i.e. a set of electronic circuits on one small plate of semiconductor material. This semiconductor material; usually silicon, is called a “chip.” The core of a SoC is that it integrates all components of a full-fledged computer such as the CPU, hard disk controllers, video card, DMA, PWM, sound card and etc into a single chip.

Superuser: The Superuser or “root” is a special user account that is for system administration. In operating systems based on Unix, the Superuser has all rights (known as permissions) to all files and programmes on the device currently running the Unix Operating System.

scrot: A terminal command that is simply used for taking a screen-shot the Raspberry Pi.

1 User Requirements

1.1 Introduction

The following will describe the problem from the perspective of the end-user. It states the problem domain and the functionality of the programme expected by the end-user.

1.2 User's view of problem

The user's will have a simple view of the problem. Users want to transmit FM signals for the transmission of an audio source from their Raspberry Pi whilst making use of little-to-no additional hardware and as low as possible CPU overhead. The audio source may be a file or a stream from the Internet.

1.3 Description of problem

Using the Raspberry Pi to produce a FM signal may seem like a tricky task to perform as we're trying to make use of just the Raspberry Pi and the programming language C to produce the FM signal. In theory, we should be able to transmit the signal solely with the Raspberry Pi without connecting anything to the GPIO pins however, the range of the frequency will be <10 centimeters from the Raspberry Pi. If we make use of the only additional hardware we'll need i.e. a simple wire antenna connected to GPIO Pin 4 of the Raspberry Pi, it will boost the signal to a range of <10 meters which may be improved with additional power to the antenna.

1.4 Expectations of the software solution

The solution to the problem needs to ensure a user is able to transmit the audio as a FM signal that will be able to be tuned-in to by any FM receiver that is tuned onto the FM frequency transmitted by the Raspberry Pi while making use of minimal additional hardware. The audio produced from tuning into the FM signal needs to be clear and free of static noise.

1.5 Not expected of the software solution

The software will not be able to play multiple audio files at a given moment nor will it be able to emit the frequency modulation at multiple frequency bands. The software will also not produce stereo as the audio file must be in a 16-bit mono wav format. Finally not to be

expected from the software is being able to adjust the frequency band of the FM signal it broadcasts whilst broadcasting the FM signal.

1.6 Conclusion

The previous section describes how the end-user requires a system that will allow them to transmit FM signals from the Raspberry Pi by describing the expectations of the system to define the scope of the project as well as stating what is not expected of the solution. The following section will explain the solution from the designer's interpretation as to solving the problem.

2 Requirements Analysis Design

2.1 Introduction

In the previous section, the end-user specified the need for a transmitting a FM signal from a Raspberry Pi while making use of minimal additional hardware. The following section will explain how the developer will attempt to meet the user's requirements. This will be done by breaking down the problem into how the C code executes and identifying all applicable details.

2.2 Interpretation and break-down of the problem

The input of the solution will be simply an audio file which the user wishes to broadcast over a FM band. The user will run the solution by entering a single terminal command which will provide the code with the name of the audio file as well as possibly the FM band they want to transmit the signal at provided it remains in valid FM frequency ranges for their specific country. In order to ensure quality of the signal, a simple single wire antenna will be attached to the Raspberry Pi on GPIO pin 4. This will result in the range of the signal being increased from <10 centimeters to <10 meters.

2.3 Complete analysis of the problem

The problem is one of security. It would be relatively simple to make use of existing code for emulating the FM transmitter on the Raspberry Pi 1 but that code makes use of a "hack"; a "hack" that will only work on GNO/Linux operating systems that can read/write to the

special file “/dev/mem” and will not work on other Raspberry Pi operating systems such as FreeBSD or NetBSD. The entire objective for this project is to come up with a solution that will allow for the transmission of FM signals on a Raspberry Pi enabling the operating system to make use of two hardware features, PWM and DMA, on the Raspberry Pi in a safe way from **kernel-space**, not making use of a wild and potentially dangerous memory map accessible from user space. This entails writing complex **kernel-drivers** for the PWM and DMA on the Raspberry Pi running any genuine Raspberry Pi operating system; we will begin with NetBSD. To do this we will have to not use the “/dev/mem” special file at all and instead make use of Direct Memory Allocation (DMA) and the Pulse-Width Modulator (PWM) to allow the mapping between the memory and device.

2.4 Other similar systems

Computer Scientists over at Imperial College Robotics Society have designed a simple programme that “hacks” the Raspberry Pi into an FM transmitter. This C programme alters the peripheral bus in physical memory into its virtual address space using the “/dev/mem” special file and mmap commands. To do this however, root access is needed. As it makes use of the “/dev/mem” special file, it is seen a security risk as this special file is a direct call to the main memory (RAM) of the Pi from **user-space**; as all information of the current session on the Pi may be accessed using this special file which may include confidential information.

2.5 Suggested Solution

The suggested solution to the problem would be to modify the open-source code of the Imperial College Robotics Society so that it does not make use of the “/dev/mem” special file to perform the FM signal transmission. This will be done by making use of Direct Memory Allocation (DMA) to map information between the Raspberry Pi and its’ memory (RAM) without needing to contact the central processing unit.

2.6 Conclusion

This final section stated how the designer interpreted and broke-down the problem. A basic and simple framework for a solution to the problem was provided in order to solve how the Raspberry Pi will be used in order to produce a FM signal using solely the on-board

Raspberry Pi hardware, the programming language C and a small wire antenna to act as a signal booster.

3 User Interface Specification

3.1 Introduction

The User Interface Specification will cover all possible actions that the user may perform as well as all visual, auditory and other interaction responses the system will provide to the user.

3.2 User Interface

The software will be operated from the command-line of the secure Unix system currently operating on the Raspberry Pi. From the command-line the user must then navigate into the folder that contains the source code for the software which is called `pifm.c` as well as the file `fm_transmitter` which is an executable. (Note that the “`scrot`” command in the terminal is simply used for screen-shotting the Raspberry Pi.)

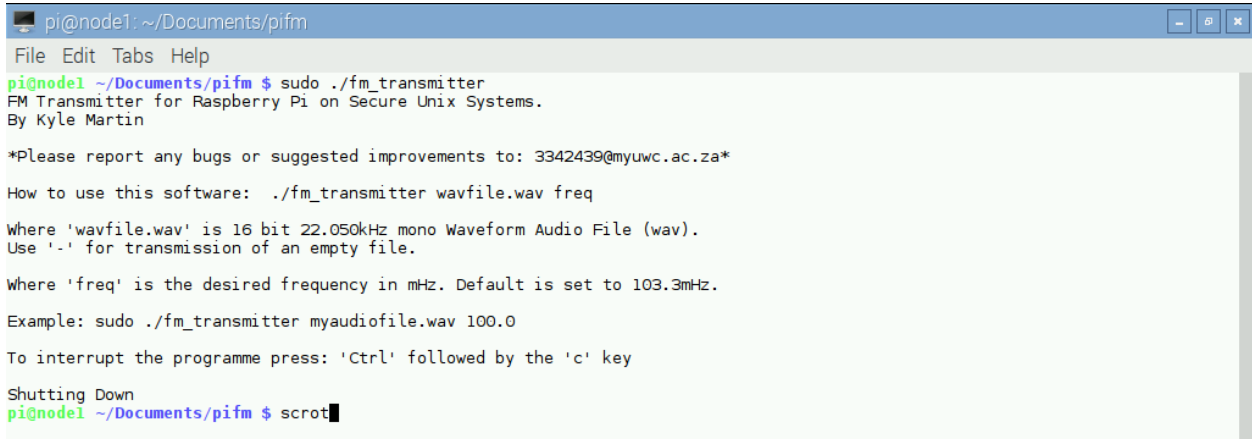
The first step is optional but highly recommended, the user should recompile the code on their Raspberry Pi through the command-line using the `g++` compiler. “`g++ pifm.c -o fm_transmitter`” without the quotation marks.

A terminal window screenshot showing a successful compilation. The window title is 'pi@node1: ~/Documents/pifm'. The terminal output shows the command 'g++ pifm.c -o fm_transmitter' being executed, followed by 'scrot' being entered. There is no output from the compilation command.

```
pi@node1: ~/Documents/pifm
File Edit Tabs Help
pi@node1 ~/Documents/pifm $ g++ pifm.c -o fm_transmitter
pi@node1 ~/Documents/pifm $ scrot
```

Figure 1: The result of compiling the code successfully. Note there is no output to the terminal on successful compilation.

The next step is to run the executable file as the “Superuser”. To do this from the command line enter “sudo ./fm_transmitter”. This will execute the code provided there was no errors during compilation and bring the user to the second screen displayed below:



```
pi@node1: ~/Documents/pifm
File Edit Tabs Help
pi@node1 ~/Documents/pifm $ sudo ./fm_transmitter
FM Transmitter for Raspberry Pi on Secure Unix Systems.
By Kyle Martin

*Please report any bugs or suggested improvements to: 3342439@myuwc.ac.za*

How to use this software: ./fm_transmitter wavfile.wav freq

Where 'wavfile.wav' is 16 bit 22.050kHz mono Waveform Audio File (wav).
Use '-' for transmission of an empty file.

Where 'freq' is the desired frequency in mHz. Default is set to 103.3mHz.

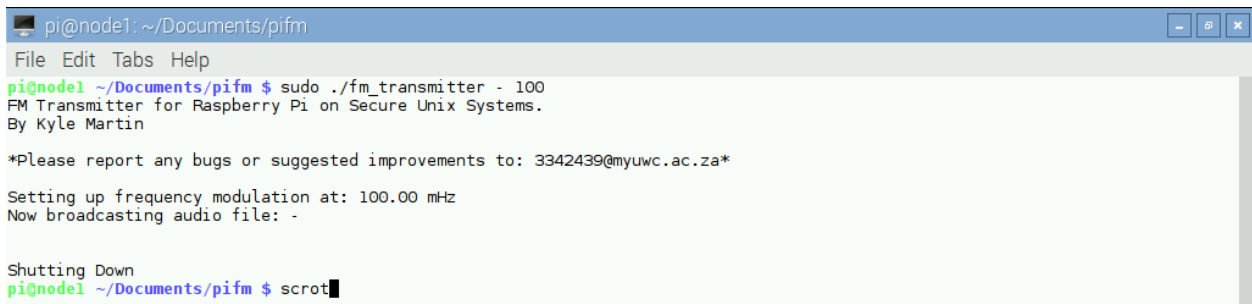
Example: sudo ./fm_transmitter myaudiofile.wav 100.0

To interrupt the programme press: 'Ctrl' followed by the 'c' key

Shutting Down
pi@node1 ~/Documents/pifm $ scrot
```

Figure 2: The result of executing the code successfully.

The screen in figure 2 explains to the user how to execute the software as well as the optional arguments the software accepts during execution time. It also explains how to interrupt the software while it is in executing. The user may now re-execute the code using the correct syntax and will be brought to the following screen seen bellow in figure 3. (Note for this example the audio file “sound.wav” and frequency “100.0” mHz will be used.)



```
pi@node1: ~/Documents/pifm
File Edit Tabs Help
pi@node1 ~/Documents/pifm $ sudo ./fm_transmitter - 100
FM Transmitter for Raspberry Pi on Secure Unix Systems.
By Kyle Martin

*Please report any bugs or suggested improvements to: 3342439@myuwc.ac.za*

Setting up frequency modulation at: 100.00 mHz
Now broadcasting audio file: -

Shutting Down
pi@node1 ~/Documents/pifm $ scrot
```

Figure 3: The result of executing the code successfully with valid arguments.

4 High-Level Design

4.1 Introduction

High-level design explains the architecture that will be used for developing software. The high-level design diagram will provide an overview of the entire system while identifying the main components of the system that will be developed for the product.

4.2 Design diagram

Below, in figure 4, is a visual representation of a multilayer description of a computer.

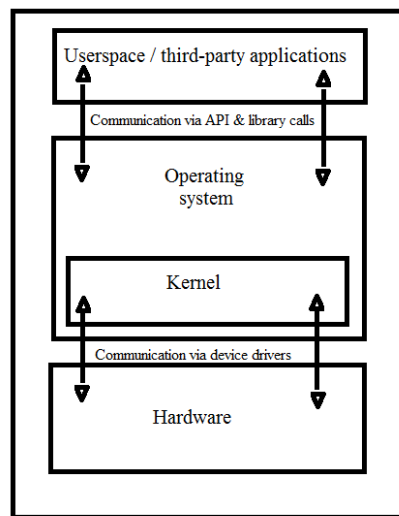


Figure 4: Multilayer description of a computer.

As seen above, one of the key features of a secure operating system is to protect the hardware. This is done by ensuring that various applications access the hardware in a safe and non-concurrent manner. I.E, as seen in figure 4, if the user wants to access the hardware, from user-space the computer will make a library call to the operating system. Finally, the operating system, which has control over the kernel, will make use of drivers to access the hardware via the kernel.

As mentioned in section 2, subsection 2.4; a similar system exists but this system cannot be used for the implementation of this project as it makes use of what is essentially accessing the hardware of the computer directly from user-space. This is seen in figure 5 on the following page.

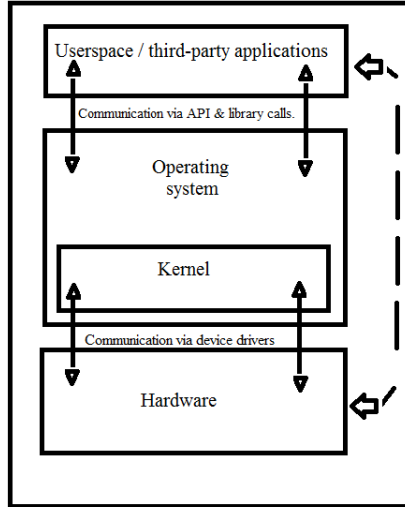


Figure 5: Multilayer description of a computer with direct hardware access via “/dev/mem” represented by the dashed arrow.

4.3 Conclusion

It is clear to see that at the very essence of this project I will have to write some Raspberry Pi specific drivers that will remove the need for the direct pathway between user-space and the hardware.

5 Low-level Design

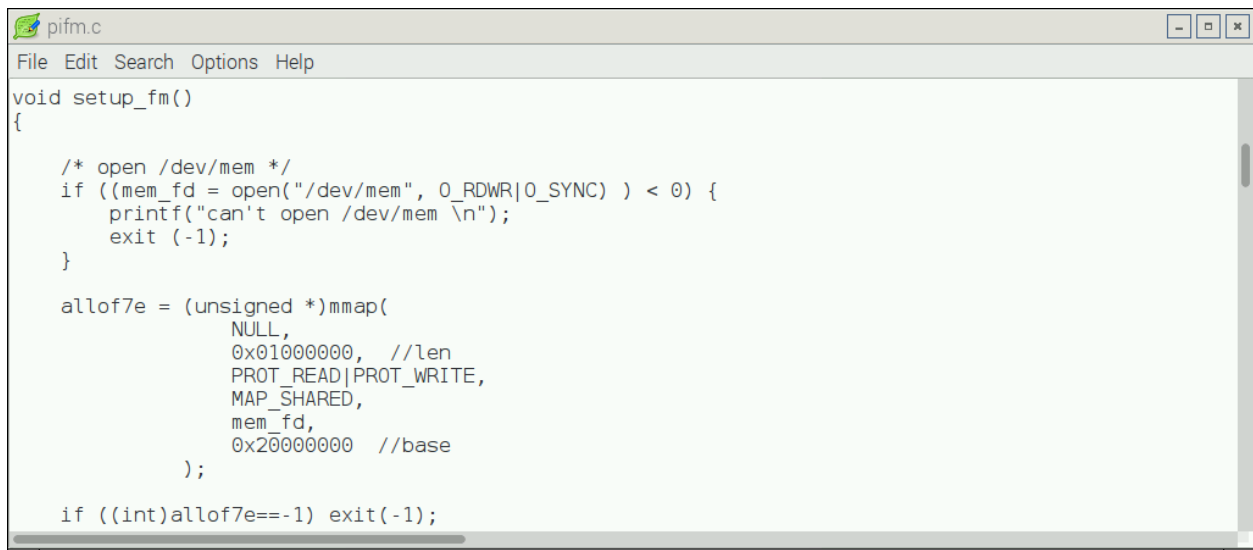
5.1 Introduction

Low-level design is a modular-level design process which follows a step-by-step refinement process. This may be used for designing data structures, algorithms, required software architecture and source code. Overall, the data organization may be defined during requirement analysis and then redefined during implementation. Low-level design phase is generally where software components are actually designed.

5.2 Design diagram

To have a better idea of what exactly needs to be developed, a strong understanding is needed of the “hacked” solution’s source code. This is to ensure why the “hacked” solution

generates cause for concern and why it cannot be used as a permanent solution but more as the basis for my solution.



```
void setup_fm()
{
    /* open /dev/mem */
    if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
        printf("can't open /dev/mem \n");
        exit (-1);
    }

    allof7e = (unsigned *)mmap(
        NULL,
        0x01000000, //len
        PROT_READ|PROT_WRITE,
        MAP_SHARED,
        mem_fd,
        0x20000000 //base
    );

    if ((int)allof7e==-1) exit(-1);
}
```

Figure 6: Snippet of the setup_fm() method of the pifm.c source code file.

In this method there are a few things to note. The code tries to open the special file “/dev/mem” and if it cannot open that file, the entire programme terminates with the exit(-1) method. This is problematic because in good practice any user should **NEVER** have direct access to the “/dev/mem” file.

Another cause for concern comes from the “mmap()” method’s “PROC_READ” and “PROC_WRITE”. “PROC_READ” is allowing the “mmap()” method to read from the “/dev/mem” file which may and will be an issue as that file contains all the related information about the user for that session. This may include, but is not limited to personal confidential information such as banking credentials. “PROC_WRITE”, unlike “PROC_READ”, is always a cause for concern. “PROC_WRITE” is allowing the source code to directly access **and manipulate** the hardware from the user-space. This is an extremely risky practice as if incorrect information is sent to the hardware, it will result in improper functionality of the Raspberry Pi, corruption of data and/or physical damage to the Raspberry Pi’s hardware.

The user is prevented from having direct access to the “/dev/mem” to try and ensure no undesired manipulation of the “/dev/mem” file occurs. If the user enters “/dev/mem” into the terminal of a secure Unix-based operating system, they will get a “permission denied”

error seen below in figure 7.

```
pi@node1: ~/Documents/pifm
File Edit Tabs Help
pi@node1 ~/Documents/pifm $ /dev/mem
bash: /dev/mem: Permission denied
pi@node1 ~/Documents/pifm $ ls -l /dev/mem
crw-r----- 1 root kmem 1, 1 Jun  5 16:14 /dev/mem
pi@node1 ~/Documents/pifm $ scrot
```

Figure 7: Result of the “ls -l” command on the “/dev/mem” special file on a secure unix system. (The example seen here was taken on Raspbian)

As seen in figure 7, it is only the “root” that has the permission to read from and write to the “/dev/mem” file. This is why the hacked solution will only execute correctly if the command is given by the “root” by making use of “sudo” before initiating the executable file.

My solution will take on the following structure as seen below in figure 8.

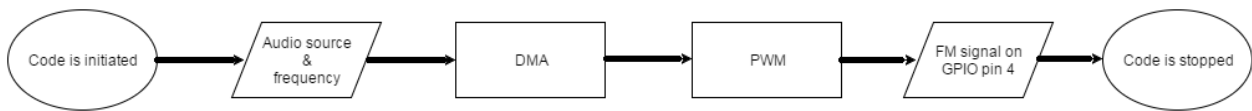


Figure 8: Visual representation of the functional diagram to my solution.

Essentially, my solution will take the audio source input and make use of DMA or Direct Memory Allocation to use the PWM or Pulse-Width Modulator without needing to overhead the CPU with the read/write commands needed if making use of the “/dev/mem” special file; which is a security risk. The Pulse-Width Modulator will then modulate the digital signals at the FM frequency specified by the user on the GPIO pin 4. These digital signals will be broadcasting the audio received from the audio source.

5.3 Conclusion

My solution will be an improvement over the “hacked” solution as it eliminates the security concerns generated by making use of the “/dev/mem” special file. As such I will be using and writing device drivers for secure Unix systems such as FreeBSD and NetBSD that will reduce CPU overhead by managing the DMA and PWM of a Raspberry Pi.

References

- Matt Richardson and Shawn Wallace. *Getting Started with Raspberry Pi*. O'Reilly Media, Inc., 2012.
- Eben Upton and Gareth Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.
- Justin Ellingwood. A comparative introduction to freebsd for linux users. *DigitalOcean*, 2015.
- Ligu Yu, Stephen R Schach, Kai Chen, Gillian Z Heller, and Jeff Offutt. Maintainability of the kernels of open-source operating systems: A comparison of linux with freebsd, netbsd, and openbsd. *Journal of Systems and Software*, 79(6):807–815, 2006.
- Simon Monk. *Raspberry Pi Cookbook*. O'Reilly Media, Inc., 2013.
- John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.
- Timothy L Warner. *Hacking Raspberry Pi*. Que Publishing, 2013.
- Oliver Mattos and Oskar Weigl. Turning the Raspberry Pi Into an FM Transmitter. http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter, 2015. [Online; accessed 12 April 2016].
- Christophe Jacquet. FM-RDS transmitter using the Raspberry Pi's PWM . <https://github.com/ChristopheJacquet/PiFmRds>, 2014. [Online; accessed 12 April 2016].