

Creating a Linux Kernel Module (LKM) using C to interact with Arduino

Arturo Chinchilla Sánchez, Malcolm Davis Steele

March 27, 2016

Abstract

As part of the course Languages, Compilers and Interpreters, will be created a LKM for Linux 3.x which will be written in pure C and interact with an Arduino device. “A loadable kernel module (LKM) is a mechanism for adding code to, or removing code from, the Linux kernel at run time. They are ideal for device drivers, enabling the kernel to communicate with the hardware without it having to know how the hardware works. The alternative to LKMs would be to build the code for each and every driver into the Linux kernel. Without this modular capability, the Linux kernel would be very large, as it would have to support every driver that would ever be needed on the system. You would also have to rebuild the kernel every time you wanted to add new hardware or update a device driver. The downside of LKMs is that driver files have to be maintained for each device. LKMs are loaded at run time, but they do not execute in user space — they are essentially part of the kernel.”[1]

Contents

1	Introduction	3
2	Development environment	5
2.1	Overleaf:	5
2.2	Git and GitHub:	6
2.3	123D Circuits	6
2.4	Sublime Text:	7
3	Data Structures, functions and libraries	7
4	Instructions to use each of the programs	7
4.1	Arduino	9
4.2	Arduino LKM	10
5	Student Activity Log	10
5.1	Arturo's Time-Sheet	11
5.2	Malcolm's Time-Sheet	12
6	Project final status	12
6.1	Issues, Limitations and Challenges	13
6.2	Know Issues	13
7	Conclusions, Suggestions and Recommendations.	13

List of Figures

1	Overleaf Project.	5
2	Git Project Home.	6
3	123D Circuit Home.	7
4	Sublime Text, text editor.	8
5	Schematic Circuit.	8
6	Arduino and circuit Box.	9
7	Terminal Commands Example.	11

1 Introduction

Linux was originally developed by Linus Torvalds in 1991 to be a operating system for the IBM Personal computers, it's a Unix-Like operating system. The Linux Kernel is widely known for his flexibility and the power that grants to the programmer, this because even though his source code is big, it can be modified with all the tweaks that are looked for. One way to do this is with a Loadable Kernel Module or LKM. This is a piece of code that can be loaded to the Linux kernel so you can add functionality to the system [5].

LKMs are written in C programming language, this language was developed in the Bell Labs(AT&T), and one of his purposes was to develop the Unix kernel. "By early 1973, the essentials of modern C were complete. The language and compiler were strong enough to permit us to rewrite the Unix kernel for the PDP-11 in C during the summer of that year" [6].

For writing a LKM some specific headers for the kernel modules like the *"linux/module.h"* or the *"linux/module.h"* are included to the code, then the functions that the kernel will perform. Two important methods that the LKM must have are the one that loads the module to the kernel and the one that cleanup the module and dereference it, this methods can be named as the programmer wants if he specifies with the *"module_init(methodName)"* and the *"module_exit(methodName)"* functions. If not also can be used the default names *"init_module"* and *"cleanup_module"*.

The LKM after been written, can be compiled and inserted to the Linux with the make command on a terminal located on the same place as the *"module.c"* file, then you can insert the module with *"insmod module.ko"*. When the module is inserted the *"init_module"* method will be called, and after that module you can list the active modules with the method *"lsmod"* or show the module info with *modinfo module*, finally when up to clean the module from kernel, it's used the command *"rmmod module"*. Some of this commands needs root rights to use them so with the word sudo before the command the module is up to go[2]. An example code of a hello world module with his corresponding make file will be as follows.

```
1 | /*
2 | *  hello -2.c - Demonstrating the module_init() and module_exit
   | *  () macros.
3 | *  This is preferred over using init_module() and
   | *  cleanup_module().
```

```

4  */
5  #include <linux/module.h> /* Needed by all modules */
6  #include <linux/kernel.h> /* Needed for KERNINFO */
7  #include <linux/init.h>   /* Needed for the macros */
8
9  static int __init hello_2_init(void)
10 {
11     printk(KERNINFO "Hello, world 2\n");
12     return 0;
13 }
14
15 static void __exit hello_2_exit(void)
16 {
17     printk(KERNINFO "Goodbye, world 2\n");
18 }
19
20 module_init(hello_2_init);
21 module_exit(hello_2_exit);

```

Listing 1: Hello World Module [3]

```

1  obj-m += hello-2.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6  clean:
7      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

Listing 2: Makefile [3]

The LKM can be used for many purposes, but the most common are device drivers, file system drivers, network drivers and system calls.[2] As the main function is to communicate the hardware with the user space, the most uses are implementation of drivers.

The Arduino hardware consists of a printed circuit board with a micro-controller, usually Atmel AVR (in this case Atmega328), and digital and analog input / output, which can be connected to expansion boards (shields) that expand the features operation of the Arduino. Arduino platform provides its own language, for programming the same, which is very easy to learn and has its own functions to handle the Arduino in a more correct way, , also has its own IDE, which provides very useful characteristics; for interaction with the same, so very rare to see someone wants to program your arduino in a different language as is C, this causes the documentation

is very sparse. Not having enough required documentation, it is taken as a reference Data Sheet micro-controller (Atmega328) which indicates how to manipulate all pins of the Arduino, which is a task a little tangled, for the impressive amount of codes for setting each one of the functions of each pin.

2 Development environment

2.1 Overleaf:

Free online tool used to write academic documents. This service allow create, edit and share your documents easily using \LaTeX . Provide two principal windows, in the first you can write your document, obviously using the \LaTeX syntax, and the second you can see your compiled document. \LaTeX also is composed of a large set of macros getting documents have a typographic high quality. Therefore it is widely used for the creation of academic papers, theses and technical books, since the typographic quality and made with \LaTeX documents is comparable to that of a great scientific publishing. The source can be found on [7]. Fig 1.

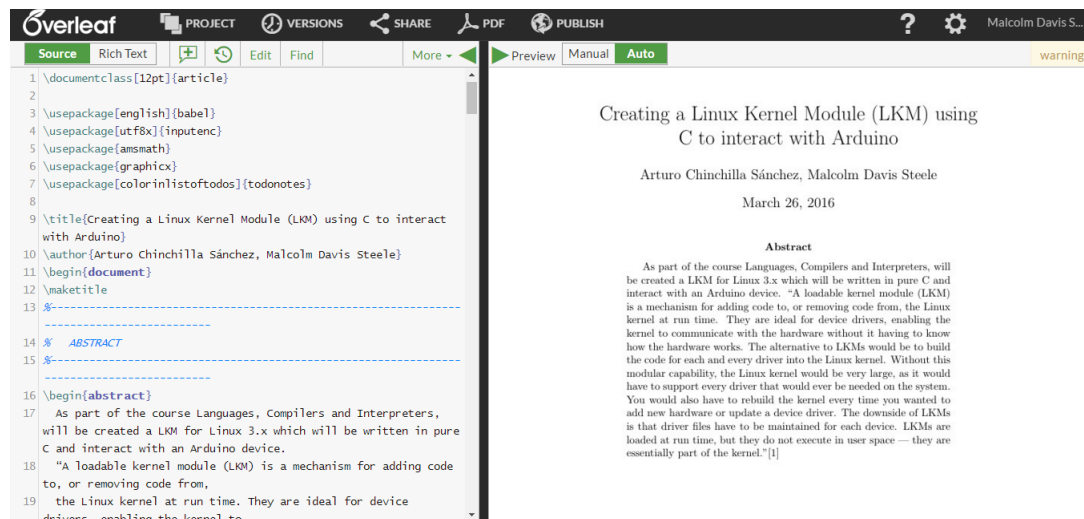


Figure 1: Overleaf Project.

2.2 Git and GitHub:

To store the code in a practical way and have a versioning control Git is used for creating repositories and GitHub as service and cloud storage. This code manager (Git) let us to create branches for each member of the group and work separately, change the branch work, see and edit code of another branches, make commits for later if you want to go back in your code and when each part needed is done they can be merged into master branch to have a final version of the code. The code can be found on [8]. Fig 2.

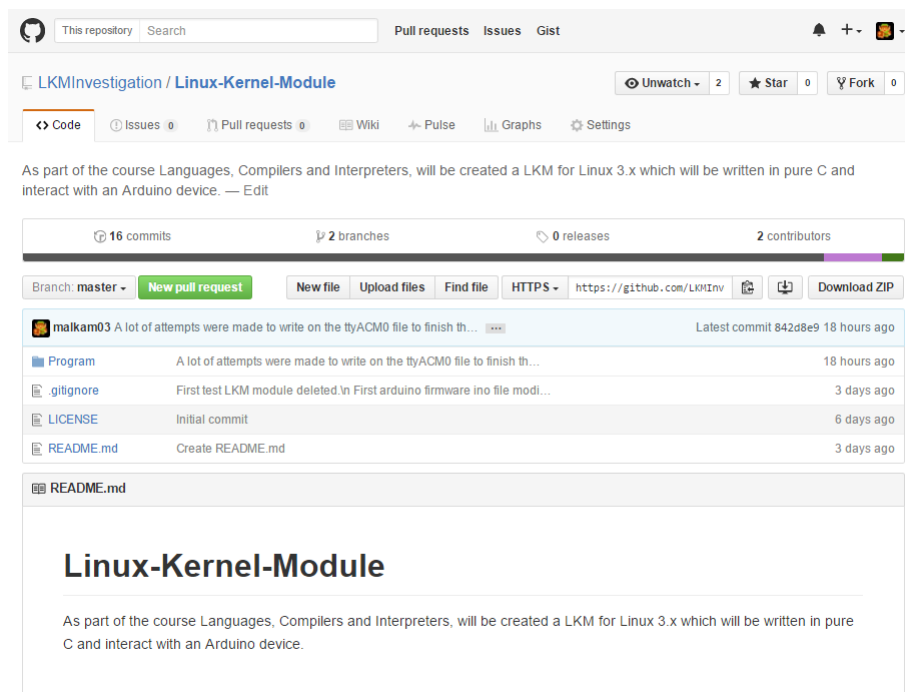


Figure 2: Git Project Home.

2.3 123D Circuits

This tool it's useful for sketch the circuits of a short project, and can be used to simulate the circuits and the program on the arduino(just arduino code). Also the circuit can be viewed in 3 different modes and design a PBC board. It was used for the Arduino-Led Circuit and for simulate the code before flashing the arduino. Fig 3.

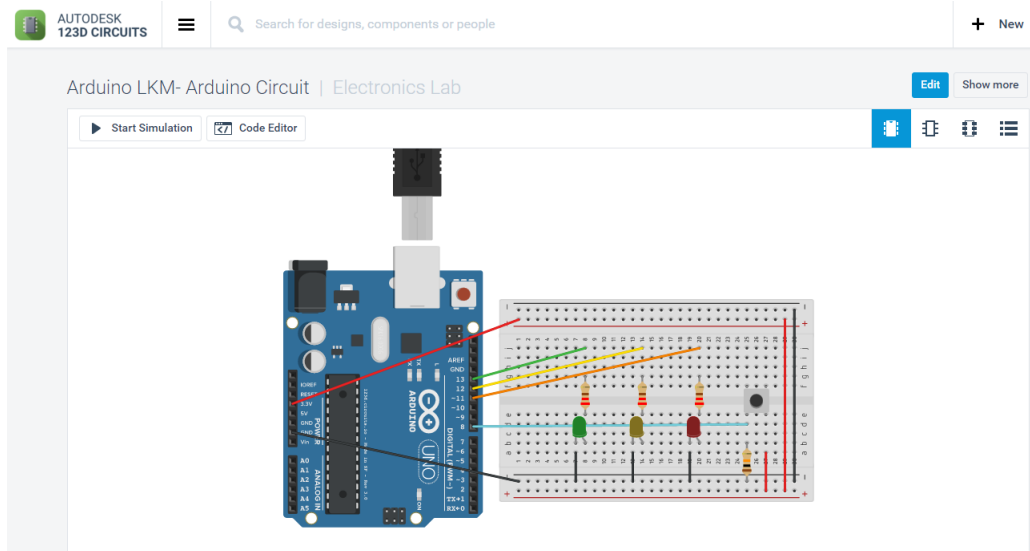


Figure 3: 123D Circuit Home.

2.4 Sublime Text:

It's a text editor for code, markup and prose. It contains functionalities very useful for programmers and facilitates the code development process. It proved to be very versatile and stable, giving good impressions, simple but with great features. Things like word suggestion, coloring restricted words, minimap, multi selection and multi cursor. Also supports multiple languages like Java, C, C++ and others. Fig.4

3 Data Structures, functions and libraries

Provide a description of all the functions, libraries and data structures used for the development of this project.

4 Instructions to use each of the programs

Before using the program it's needed to have the hardware requirements, for this project the Arduino Uno is used, but can be used with other Arduino types. The circuit of the Arduino used is the one of Fig. 5.

```

File Edit Selection Find View Goto Tools Project Preferences Help
led.c
1  /**
2   * @file   arduinoFirmware.c
3   * @author Arturo Chinchilla S.
4   * @author Malcolm Davis S.
5   * @date   March 21, 2016
6   * @version 0.4
7   * @brief   A simple firmware, which was written in C language using
8   *          which handles some features on the Arduino, as multimode
9   *          using the serial port with a Linux kernel module.
10  */
11  #include <avr/io.h>
12  #include <util/delay.h> // Header to use delay_ms() function
13  #include <stdbool.h> // Header to use boolean types in C
14  #include <avr/interrupt.h> // Contain functions to manage interrupt
15  #define delayTime 250 // Define a delay time used in the code
16  #define F_CPU 16000000 // Define the Frequency of the Arduino C
17  #define BUAD 9600 // Amount of bauds (characters per second)
18  #define BRC ((F_CPU/16/BUAD)-1) // Baud Rate Calculate
19  #define RX_BUFFER_SIZE 128 // Size of Buffer used to receive serial
20  #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~BV(bit)) // Provide access
21  #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= BV(bit)) // Provide access
22
23  void ledModeZero(void); // Manage the first LED animation, turn ON,
24  void ledModeOne(void); // Manage the second LED animation, LED bur
25  int checkButton(void); // Manage the button actions
26  void serialTx(void); // Transmit data function
27  void serialRx(void); // Receive data function
28  char getChar(void); // Gets the first character, give us the opt
29  char* getBuffChar(void); // Returns the buffer received
30  void selectionMode(void); // Select the current animation
31  char buffChar[RX_BUFFER_SIZE]; // Char with the value to change

```

Figure 4: Sublime Text, text editor.

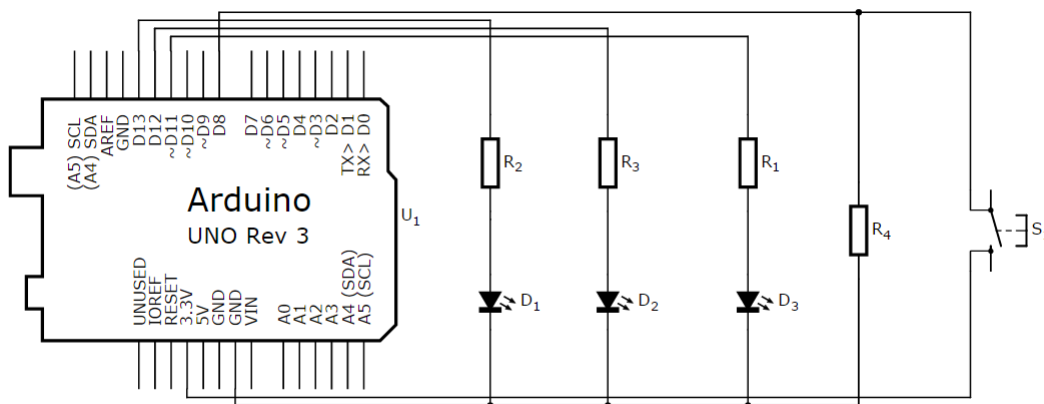


Figure 5: Schematic Circuit.

The R1, R2 and R3 resistors are rated 220 ohms, and are connected to the 13, 12 and 11 pins on the arduino respectively and then in series with them a led polarized to grown, the R4 resistor or Pull Down resistor is between the

ground and the lecture of the 8th pin of the arduino, this for assurance that the read will be clear a logic 1 or a logic 0, without floating ground errors. The switch or button is connected between the 3.3 v pin of the arduino and the 8th pin for the logic lecture.

4.1 Arduino

Once you have mounted the circuit and connected to the Arduino, connect this to the computer, you must open the Arduino's Firmware folder in the Linux terminal and run the following commands:

1. `$ avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o ArduinoInt.o ArduinoInt.c`
2. `$ avr-gcc -mmcu=atmega328p ArduinoInt.o -o ArduinoInt`
3. `$ avr-objcopy -O ihex -R .eeprom ArduinoInt ArduinoInt.hex`
4. `$ avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:ArduinoInt.hex`

For this project it was mounted in a little box. See the Fig. 7



Figure 6: Arduino and circuit Box.

Yo can push the button for see the LEDs animation, by default the animation is in Mode One, that means that the LEDs go to turn ON/OFF. Then you can send a serial message with the following codes:

- 00: To select mode one, that are the ON/OFF mode.
- 01: To select the mode two, that are the burst mode, this mode makes a LED burst, that means each LED go turn ON/OFF sequentially, one by one.

Note: It is recommended to use the Arduino IDE and the tool Serial Monitor for sent the serial message.

Now you can push the button and enjoy the LEDs animation, and if change the mode, if you want.

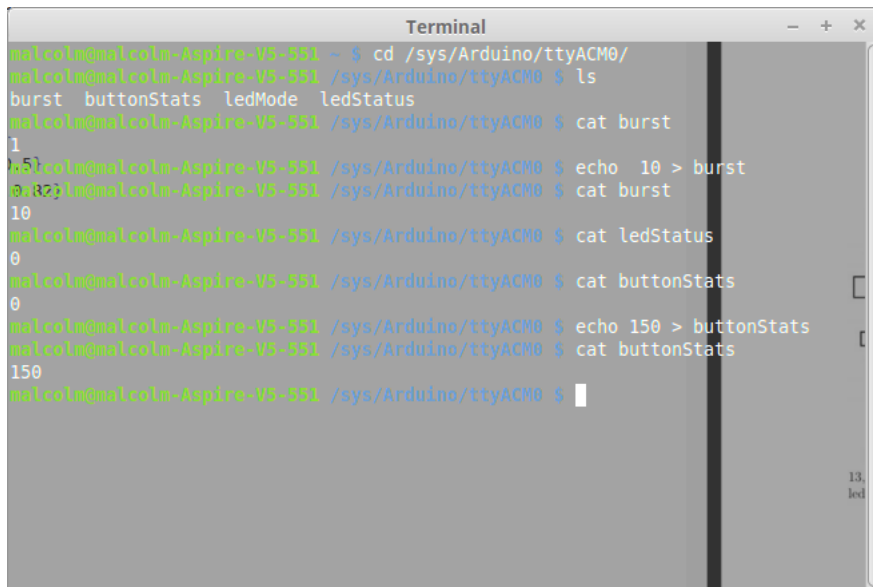
4.2 Arduino LKM

For the LKM part you just have to compile it and insert it to the kernel. IMPORTANT: Connect the arduino before loading the module.

1. Go to the program directory and locate the ArduinoLKM.c file and the Makefile.
2. Open a terminal there, and type `$ make` to build the module.
3. Use the command `$ sudo insmod ArduinoLKM.ko` to insert the module to the kernel, in this part a password entry will be prompt. Use your root password to gain super user privileges.
4. At this point the module is already loaded and you can modify his attributes at the `/sys/Android/ttyACM0` directory this with a `$ cd /sys/Android/ttyACM0`
5. You can `$ls` (list the configuration files), `$echo` to all the configuration files but the ledStatus, and `$cat` to all of them. Fig 7.
6. To unload the module a simple `$ sudo rmmod ArduinoLKM` will do.

5 Student Activity Log

Include here the details of the activities performed by each of the students in this assignment. Include one line per each activity, providing details like:



```
malcolm@malcolm-Aspire-V5-551 ~ $ cd /sys/Arduino/ttyACM0/
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ ls
burst buttonStats ledMode ledStatus
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ cat burst
1
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ echo 10 > burst
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ cat burst
10
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ cat ledStatus
0
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ cat buttonStats
0
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ echo 150 > buttonStats
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $ cat buttonStats
150
malcolm@malcolm-Aspire-V5-551 /sys/Arduino/ttyACM0 $
```

Figure 7: Terminal Commands Example.

- Description of the activity.
- Amount of time in hours/minutes spent on each activity.
- Total amount of hours spent in the development of this project per student.
- Use a time-sheet format (or tablet) to present this activity log.

5.1 Arturo's Time-Sheet

Date	Task Description	Spent Time
03/10/2016	Installed the klinux headers and gcc-avr packages	0.5 hours
03/10/2016	Read an article and make a "Hello World LKM"	1.5 hours
03/19/2016	Documentation process start	2 hours
03/20/2016	Create organization and repository on GitHub	0.5 hours
03/21/2016	Investigation about LKM's	16 hours
03/22/2016	Investigation about LKM's	16 hours
03/23/2016	Investigation about C code for Arduino and av-gcc	16.5 hours
03/24/2016	Write the first version of firmware	16.5 hours
03/25/2016	Fix some bugs in the firmware	15 hours
03/26/2016	Write the final version and complete the documentation	14.5 hours
	Total	99 hours

5.2 Malcolm's Time-Sheet

Date	Task Description	Spent Time
03/10/2016	Installed the klinux headers and gcc-avr packages.	0.5 hour
03/10/2016	Make the "Hello World LKM"	1 hour
03/19/2016	Documentation process start	2 hours
03/21/2016	Investigation about LKM's	16.5 hours
03/22/2016	Investigation about tty communication	16.5 hours
03/23/2016	Investigation about tty communication	16.5 hours
03/25/2016	Try to enable the communication between the programs	16 hours
03/26/2016	Fix and debug the code, complete the documentation	14.5 hours
	Total	115.5 hours

6 Project final status

The project could not be completed in its entirety, this because the Linux kernel module could not establish communication with the Arduino firmware. You can not directly access the serial port Arduino (ttyACM0) because there is already a driver module running on that port and can not be replaced. About the Arduino, it can't change the amount of the repetitions from by the serial port, by default the repetitions is 1.

6.1 Issues, Limitations and Challenges

- The limited documentation on matters of creating Linux kernel modules, and the few code examples that have internal documentation explaining how it worked.
- Trying to manipulate (read / write) files in the user space from the Linux kernel module, many of the references found indicated that this was not allowed. "The most common question asked in this don't-do-that category is, "How do I read a file from within my kernel module?" Most new kernel developers are coming from user-space programming environments or other operating systems where reading a file is a natural and essential part of bringing configuration information into a program. From within the Linux kernel, however, reading data out of a file for configuration information is considered to be forbidden. This is due to a vast array of different problems that could result if a developer tries to do this" [4]
- Reading the Arduino ports with C language and AVR-GCC compilers, because the major part of documentation was for Arduino language code. Because Arduino language is simplest and also all is documented.

6.2 Know Issues

- Could not establish communication between the module Kernel Linux and Arduino, this because the LKM ought to manipulate user space files , which can not be done, the only documented way is through the use of an application running on user space, which according to the approach of the specification given in the project was not right.

7 Conclusions, Suggestions and Recommendations.

- Tools like the handler code versions is very useful, because in case of any failure in the new version allows us to return to a previous, plus the possibility of linking it to the storage service in the cloud, to share, and work together in a very easy way.

- Sublime Text can be one of the better text editors, that should not make lack in the programmer's computer, for its flexibility and support for multiple programming languages, in addition to the vast amount of functionality adapted for this use.
- 123D Circuits is a very practical tool for circuits simulation, provides a wide range of electronic elements, the simulation gives us the assurance that the circuit is good, to proceed to assemble it without breaking our devices.
- Create Linux kernel Modules for Kernel 3.x is not easy, because it is relatively new (2011), and much of the information refers to older versions, especially the kernel version 2.x.
- Create the firmware for the Arduino using pure C and avr-gcc compiler is very tedious, because there is little amount of information about how to do it. Apart from this internal documentation of the examples is too little.
- The approach taken for the LKM of writing on the device file to manage the configuration parameters and the state of the circuit is not the best, this is because of the must of the must not of the kernel. It's recommended look for other approach to make this task.
- It's important to separate the user space functions or programs and the kernel space, for all the power that it's within it.
- For the project and the requirements purposes other approach that doesn't do much is the one that use the mayor number of the device to link it, this is because as far our research go this only create new tty ports and does not link a one that the O.S already mount.

References

- [1] D. Molloy, "*Writing a Linux Kernel Module — Part 1: Introduction*" derekmolloy.ie, April 14th, 2015. [Online]. Available: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>. [Accessed: 20- Mar- 2016].
- [2] B. Henderson, "*Linux LKM*", The Linux Documentation Project, 2006. [Online]. Available: <http://www.tldp.org/HOWTO/Module-HOWTO/index.html> [Accessed: 20- Mar- 2016].
- [3] P. Salzman, M. Burian and O. Pomerantz, "*The Linux Kernel Module Programming Guide*", The Linux Documentation Project, 2007. [Online]. Available: <http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html>. [Accessed: 27- Mar- 2016].
- [4] G. Kroah-Hartman "*Driving Me Nuts - Things You Never Should Do in the Kernel*". Linux Journal, 2005. [Online]. Available: <https://www.linuxjournal.com/article/8110?page=0,0>. [Accessed: 26- Mar- 2016].
- [5] G. Kroah-Hartman, "*Linux kernel in a nutshell*". Sebastopol, CA: O'Reilly, 2007, p. 16.
- [6] D. Ritchie, "*The Development of C Language*". Murray Hill: Bell Labs, 1993, pp. 1-9.
- [7] A. Chinchilla and M. Davis, "*Creating a Linux Kernel Module (LKM) using C to interact with Arduino*", Overleaf.com, 2016. [Online]. Available: <https://www.overleaf.com/read/zcbqrvszmsrb>. [Accessed: 26- Mar- 2016].
- [8] A. Chichilla and M. Davis, "*LKMInvestigation/Linux-Kernel-Module*", GitHub, 2016. [Online]. Available: <https://github.com/LKMInvestigation/Linux-Kernel-Module>. [Accessed: 26- Mar- 2016].