

Projet de Fin d'Année II

H-Bridge DC motor, PID regulator control based on BeagleBone Black : Processing Real Time Unit

Developped by :

Shourouk DALHOUMI

Mohamed El Hedi BOUSSAMA

Class :

2AGE2

Supervised by:

Samir SAKRANI

Lotfi CHARAABI

Defended the 09/05/2018

In front of the jury:

President

: M^{me} KSOURI Moufida

Reviewer

: M^r BOURGUIBA Riadh

Supervisors

: M^r Samir SAKRANI

M^r Lotfi CHARAABI

Academic year 2017/2018

Acknowledgments

As a matter of first importance, We are thankful to God for His incalculable favors and for giving us the knowledge and strength to fulfill our research work.

We express our thanks and earnest appreciation to our supervisor, Mr. Samir SAKRANI, for the continuous support of our end of the year project, for his motivation, patience, and knowledge. This work would have not been done without his advice and valuable comments.

Our sincere thanks also go to our co-supervisor, Lotfi CHARAABI, for his help throughout this period. We would like to thank our dear Mr Habib OUERGHMI for all the help and assistance he provided for us throughout our period.

Abstract

A motor controller is a device or group of devices that serves to govern in some predetermined manner the performance of a motor. A motor controller might include a manual and automatic means for starting and stopping the motor, selecting forward or reverse rotation, selecting and regulating the speed, regulating or limiting the torque, and protecting against overloads and faults. We are implementing DC motor controller using PWM, thereby regulating the speed of the motor and implementing the working of an H-bridge.

In this project, the BeagleBone Black drives a DC motor with pulse-width modulation and uses a PID regulator for speed control.

The purpose of a PID regulator is to take a signal representing the demanded speed, and to drive a motor at that speed. The controller may or may not actually measure the speed of the motor.

Key words : BeagleBone Black, H-bridge, DC motor, PID regulator.

Contents

List of figures	v
Liste of tables	vi
General introduction	vii
1 DC motor control	1
1.1 Introduction	1
1.2 The DC motor	1
1.2.1 History and applications	1
1.3 PID algorithms for speed regulation.....	2
1.3.1 Why PID ?.....	3
1.3.2 Ziegler-Nichols method.....	3
1.3.3 Tuning the Pid controller.....	3
1.4 H-bridge	4
1.4.1 The operating principle.....	4
1.342 Half-bridge vs Full-bridge.....	5
1.5 PWM different techniques and types	6
1.5.1 Analog PWM	6
1.5.2 Digital PWM	6
1.5.2.1 Different PWM techniques	6
1.6 Some controlling devices and circuits	8
1.7 BeagleBone Black	9
1.7.1 Presentation and features	9
1.7.2 Why BeagleBone?	10
1.7.3 Application examples	10
1.7.4 PRU subsystem	11
1.7.5 IoT subsystem interfacing	11
1.8 Embedded Linux	12
1.9 Embedded system	12
1.10 Conclusion.....	13

2 DC motor control based on BeagleBone Black	14
2.1 Introduction	14
2.2 The motor	14
2.3 Hardware design	16
2.3.1 Auto transformer	16
2.3.2 Stabilized power supply	16
2.3.3 H-bridge.....	17
2.3.3.1 GT15J101 IJBT	18
2.3.3.2 IRF840 MOSFET	18
2.3.3.3 Bootstrap circuit	19
2.3.4.3 Command lines	20
2.3.4 CNX36U opto coupler	22
2.3.5 HEF40106B inverting trigger.....	23
2.3.5.1 why double inversion ?	24
2.3.6 Tests and simulations	24
2.4 softawre side	24
2.4.1 Getting started	26
2.4.2 Linux development on the BeagleBone Black card	27
2.4.2.1 Cross compilation	28
2.4.2.2 Direct development on the card	28
2.4.3 PRU-ICSS	29
2.4.4 LibPRUio	30
2.5 User interface.....	34
2.6 PID implementaion	35
2.7 Conclusion.....	36
 General conclusion	 37
Bibliography	4
Annexe 1	5
Annexe 2	6

Table of figures

1.1 DCMotor model	1
1.2 PID terms functions and effects on control system.....	3
1.3 PID terms and effects on control system.....	4
1.4 Mounting of motor powered by an H-bridge	4
1.5 Difference between full H-bridge and half H-bridge	5
1.6 Unipolar PWM mounting	6
1.7 Bipolar PWM mounting	7
1.8 Unipolar 4-quadrant PWM mounting	7
1.9 BeagleBone Black and Raspberry Pi comparision.....	10
2.1 Motor control model.....	15
2.2 The namplate signal	16
2.3 Auto-transformer	16
2.4 12 and 15 voltage supply	17
2.5 MOSFET H-bridge	18
2.6 Typical connection of the IR2210 driver	19
2.7 Bootstrap circuit mounting.....	19
2.8 Bootstrap circuit schematic	20
2.9 Opto coupler schematic.....	21
2.10 Opto coupler output signal	22
2.11 Schmitt trigger	23
2.12 Schmitt trigger output signals	23
2.13 High speed Opto coupler mounting	24
2.14 Power card with components	24
2.15 Power card with inputs\outputs	24
2.16 Ifconfig.....	25
2.17 IP adress changing.....	26
2.18 Geany enviroment	27
2.19 PRU-ICSS architecture	27
2.20 Device tree configuration as start up	30
2.21 Device tree loaded.....	31

2.22 UIO driver activated	31
2.23 Isleep function	31
2.24 PWM eCap test connection	32
2.25 User interface	32
2.26 PID tuning program	33

Liste of tables

1.1 BeagleBone features.....	8
2.1 Different vaues of Rinductor.....	13
2.2 Different values of Rarmature.....	14

General introduction

In 1935, Alan Turing, a British mathematician and cryptologist, published an article of 34 pages in which the term « computing machine » appeared for the first time. 10 years later, the invention of transistor marked the opening of a new period of information that allowed physicists to substantiate recently invented theories.

Nowadays, industries seek employees who are able to use developed technologies with a high level of abstraction which need deep concentration and reflex, people with a huge capacity to analyze complex problems, to fetch efficient solutions, to take decisions and implement suitable plans for different issues. These qualities are compulsory to solve technical problems, in order to face and adapt to various situations. Obviously, these people can only be engineers. In fact, what humankind did among this period is the fact of finding the best method to take a step forward and to build what has been made until the time we're passing through by enhancing science and technology.

The embedded system is frequently present in so many industrial fields as of late. That's why our supervisors guided us to work in such a domain and emphasized on learning and controlling "Embedded Linux" environment which facilitates development using the BeagleBone embedded card, then applying it to command a system such as DC motors. In fact, the aim of this project is to apply a command a DC motor after exploring its different peripherals

DC motor is a system known for its complexity because of its touchy parameters. The system is not steady, especially while changing the frequency in which the motor operates.

The PWM technique we chose for the command is not the best, but it is the most widespread in various DC motor's applications. Besides, any of the DC motors need a PID algorithm implementation to control speed.

Thus, the implementation of such a project in academic domain should not be underestimated given its presence among a variety of crucial applications such as elevators, ski-lift drivers and similar applications requiring smooth and reliable operation.

Chapter 1 :

DC motor control

Introduction

This chapter discusses a DC motor control project, which has always been used for the purpose of teaching motor speed control theory. As we all know, DC motor operating principle is the simplest to understand that's why DC motor control is crucial for electrical engineering curriculum given the various benefits of such a project which deals with different fields.

DC motor

A direct current, or DC, motor is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. It is the most common type of motor; it relies on the forces produced by magnetic fields. DC motors normally have just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

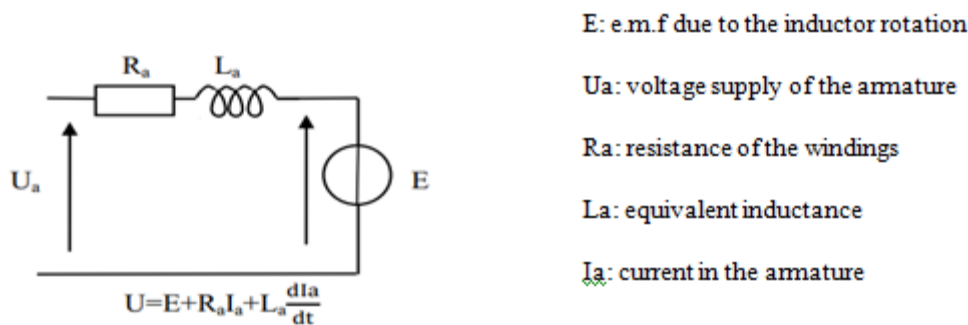


Fig 1.1: DC motor model

History and applications

DC motors were very popular 20-25 years back due to their capabilities of speed variation which was not possible in induction motors as its speed was dependent on the frequency of mains. DC motors despite their complex construction and high cost were a compulsion for

designer due their very good speed regulation which could not be achieved with any other kind of existing motors.

At present, although the speed problem of AC motor has been solved, however, DC motor is still used in machine needed for speed adjustment, positive/reverse rotation or frequent starting/braking. Since the DC motor has a good start and speed performance, it is often used to the field that have a higher requirements of speed and starting such as a large reversible rolling mills, mine hoist, hotel high-speed elevators, the goal planer, electric locomotives, diesel locomotives, large precision machine tools and large cranes and other production machinery.

1.3 PID algorithms for speed regulation

DC motors lose speed when they are loaded and increase their speed when they are unloaded. For applications where constant speed is required and the load is unknown or fluctuates, a controller will be required to fix the optimal set point of the engine; this controller may be a **proportional–integral–derivative** controller.

The determination of corresponding PID parameter values for getting the optimum performance from the process is called tuning. This is obviously a crucial part in case of all closed loop control systems. Some of these PID tuning methods are given below.

The Good Gain method, and the Ziegler-Nichols' method are experimental, that is, they require experiments to be made on the process to be controlled. If you have a mathematical process model, you can of course use these experimental methods on a simulator based on the model. One method — Skogestad's method — is model-based, we can get good PID parameter values directly from the transfer function model of the process, without doing any experiment.

1.3.1 Why PID?

The PID controller is everywhere; temperature, motion, position, speed. Yet, it is available in analog and digital forms. Why use it? It helps get your output (velocity, temperature, position, speed..) where you want it, in a short time, with minimal overshoot, and with little error. The PID controller can do the job and evenr more; it can enhance the output quality. After toutching the PID's imoortance, we will get a chance tune a PID controller.

Tuning a system means adjusting three multipliers K_p , K_i and K_d adding in various amounts of these functions to get the system to behave the way we want. The table below summarizes the PID terms and their effect on a control system.

Term	Math Function	Effect on Control System
P Proportional	$K_P \times \text{Verror}$	Typically the main drive in a control loop, K_P reduces a large part of the overall error.
I Integral	$K_I \times \int \text{Verror} \, dt$	Reduces the final error in a system. Summing even a small error over time produces a drive signal large enough to move the system toward a smaller error.
D Derivative	$K_D \times d\text{Verror} / dt$	Counteracts the K_P and K_I terms when the output changes quickly. This helps reduce overshoot and ringing. It has no effect on final error.

Fig 1.2: PID terms functions and effects on control system

1.3.2 Ziegler-Nichols method

In order to ensure the control of the DC motor, it is advantageous to find a model that best represents the real system. Approximate a model to our system is the subject of the automatic part of this project.

In fact, the issues of reliability and robustness of the system being sentenced in a mockup make ordering the system more and more difficult. Indeed, the stakes of the mechanism present a constraint for the determination of the real parameters.

For this reason, our supervisors opted for us to use the Ziegler-Nichols method which is a heuristic method of setting a PID regulator. This method was developed by Nathaniel B. Nichols and John G. Ziegler.

1.3.3 Tuning the PID controller

Although we can find many methods and theories on tuning a PID, Ziegler-Nichols's is the most known approach to get us up quickly.

The Ziegler-Nichols method consists in fixing the gains I (integral action) and D (action Derivative) to zero. P gain (Proportional action) is then increased from zero until it reaches the maximum K_u gain at which the signal at the output of the loop of command oscillates with a constant amplitude. K_u (The gain margin for loop stability) and the oscillation period T_u (the period of the oscillation frequency at stability limits) are used to set the P, I and D gains according to the type of controller used.

Rule Name	Tuning Parameters
Classic Ziegler-Nichols	$K_p = 0.6 K_u$ $T_i = 0.5 T_u$ $T_d = 0.125 T_u$
Pessen Integral Rule	$K_p = 0.7 K_u$ $T_i = 0.4 T_u$ $T_d = 0.15 T_u$
Some Overshoot	$K_p = 0.33 K_u$ $T_i = 0.5 T_u$ $T_d = 0.33 T_u$
No Overshoot	$K_p = 0.2 K_u$ $T_i = 0.5 T_u$ $T_d = 0.33 T_u$

Fig 1.3: PID terms functions and effects on control system

1.4 H-bridge

We need some kind of drivers to control the speed and direction of motors. Using H-bridge is the easy way for controlling the motor using microcontroller.

An H bridge is an electronic circuit that enables a voltage to be applied across a load in opposite direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards.

1.4.1 The operating principle

The term H-bridge is derived from the typical graphical representation of such a circuit. An H bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening T1 and T4 switches and closing T2 and T3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches T1 and T2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches T3 and T4.

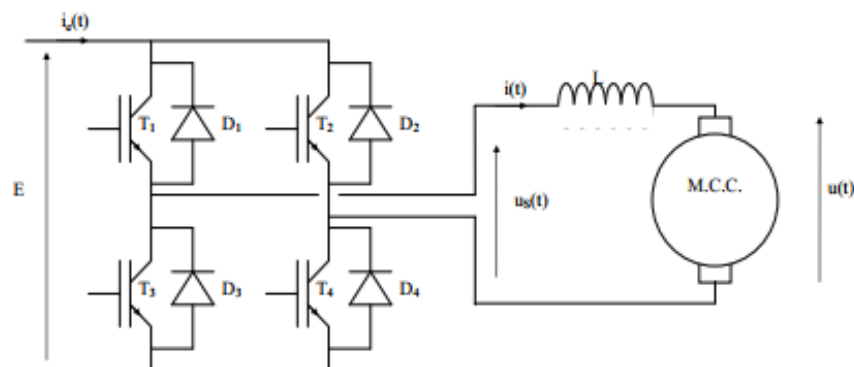


Fig 1.4: Mounting of motor circuit powered by H-bridge

1.4.2 Half-bridge vs Full-bridge

The same current doesn't go through the load; the applied volts are half for the half bridge. Hence the load's impedance needs to be $1/2$ if we want the same current. In fact; we will need bigger MOSFETs for the same power in a half bridge. Besides, reactive loads behave differently in a half bridge.

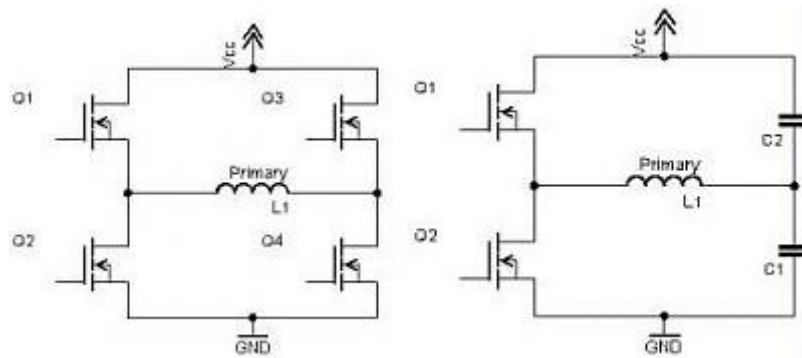


Fig 1.5: difference between full H-bridge and half H-bridge

While watching both configurations, it seems almost the same with a difference that in half bridge there are capacitors instead of MOSFETs, yet it is no longer the same current that goes through the load.

1.5 PWM different types and techniques

A pulse width modulator (PWM) is a device that may be used as an efficient DC motor speed controller.

1.5.1 Analog PWM

A circuit utilizes a triangle wave generator and comparator to generate a pulse-width-modulated (PWM) waveform with a duty cycle that is inversely proportional to the input voltage. An operational amplifier and comparator generate a triangular waveform which is passed to an inverting input of a second comparator. By passing the input voltage to a non-inverting comparator input, the PWM waveform is produced.

1.5.2 Digital PWM

Digital PWM is the most known approach to control various machines. Its implementation only needs a timer and the different signals to compare in order to obtain a duty cycle ranging from 0 to 100.

1.5.2.1 Different PWM techniques

So, which PWM technique is best for your motor control application? In case you haven't guessed, this is somewhat of a trick question. It's kind of like asking "which ice-cream flavor is the best". There aren't quite as many flavors of PWM as there are ice cream, but there are perhaps more of them than you realize.

Some PWM topologies will prevent regeneration of energy back into your electrical supply. In fact, some will only allow the motor to spin in one direction and generate torque in that same direction. These drives are called single-quadrant drives.

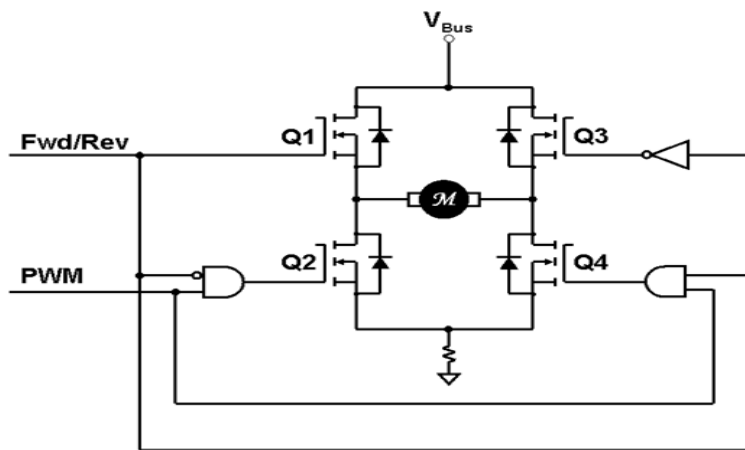


Fig 1.6: Unipolar PWM mounting

For Unipolar PWM operation in quadrant 1, Q1 is turned ON continuously while we apply a PWM signal to Q4. When Q4 is switched ON, a current path is created from V_{bus} , through Q1, through the motor, through Q4, and returning through ground. At the end of this PWM state, Q4 is switched OFF. Since the motor winding has inductance, it will fight to keep the motor current flowing in the same direction. An inductor protects its current just like a mother protects her child. It effectively says, "Don't mess with my current! If you do, I will generate whatever voltage is necessary to keep my current flowing." As a result, the inductor forces the back-body diode of Q3 to conduct. But since Q1 is always ON, the motor current will return through Q1, not the DC supply. When you think about it, you realize that since Q1 is ON continuously, this circuit behaves exactly like the single quadrant drive discussed earlier with one exception...if you want the motor to spin in the other direction, simply turn Q3 ON all the time and PWM Q2 instead. This results in quadrant 3 operation where the motor is running in reverse, and generating negative torque.

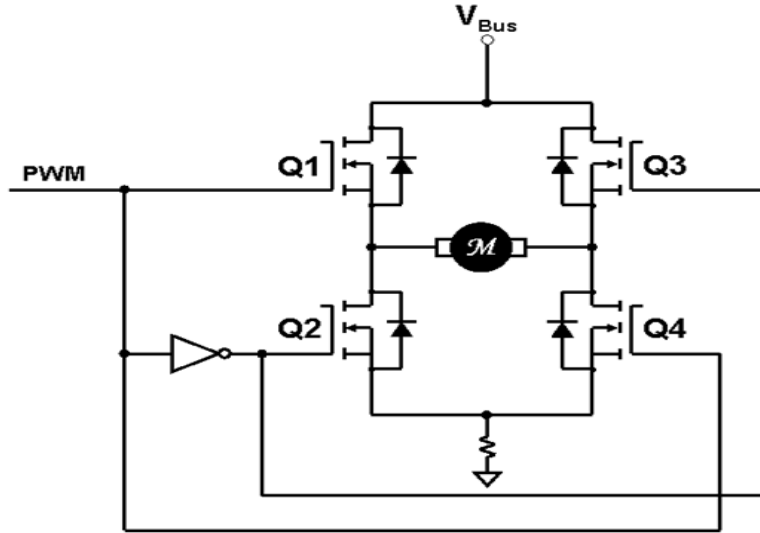


Fig 1.7: Bipolar PWM mounting

The bipolar PWM technique is inherently a 4-quadrant technique. As long as the average applied motor voltage is of the same polarity as the motor's back-EMF voltage, *and* it is greater in amplitude than the back-EMF, then the motor will operate in motoring mode. However, if the average applied motor voltage is of the same polarity as the back-EMF, but its amplitude is less than the back-EMF, then the motor will operate in generating mode.

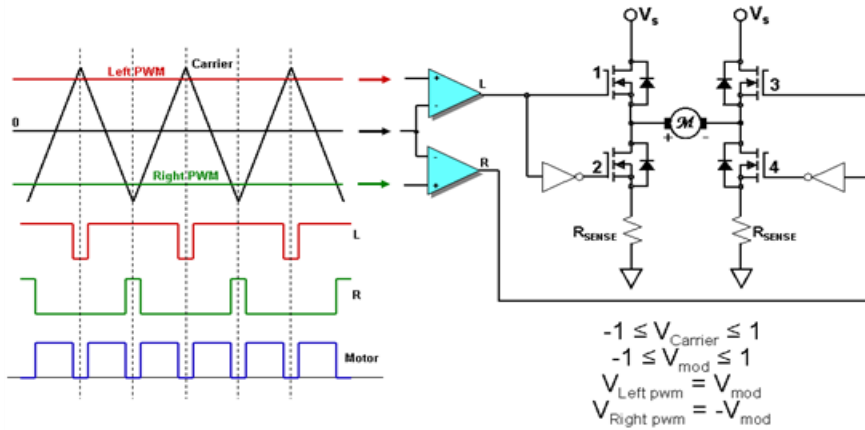


Fig 1.6: Unipolar 4-Quadrant PWM mounting

As with the Bipolar PWM technique, in Unipolar 4-Quadrant PWM Technique, there is no forward/reverse signal.

Since each half of the H-Bridge is driven in a complementary fashion, we must include dead-time between the top and bottom PWM signals. Assuming dead-time is supplied internally by the PWM module, this technique requires four PWM pins on the processor. this technique has an insatiable appetite for PWM signals.

1.6 Some controlling devices and circuits

A speed control device for a DC motor controls the rotational speed of the DC motor through adjustment of a duty cycle of a semiconductor switching element.

In some of the electronics projects you may want to control a DC Motor with 8051 or AT89C51 microcontroller. The maximum current that can be sourced or sunk from such a microcontroller can reach a few hundred Amperes.

Or we can simply use the PIC16F684 microcontroller to do such a duty. Besides, by connecting an L298 bridge IC to an Arduino, you can control a DC motor. Moreover, committed to motor control for more than 20 years, ST offers a complete and wide range of microcontrollers for DC motors. Finally, the enhanced Raspberry and BeagleBone boards can do the job in a prestigious way.

1.7 BeagleBone Black

The BeagleBone is one of the most developed microprocessors of nowadays which is used for different applications, including motors command and control.

1.7.1 Presentation and features

The BeagleBone is a property of Texas Instrument. It is considered as an « open hardware » and « open source ». In fact, its architecture is free and open. That's why many companies produce this sort of card according to the field on which they work. We can mention industrial BeagleBone produced «Element14» know for its sturdiness, BeagleBone green wireless which works on IoT field. These two cards have almost the same hardware and software than BeagleBone black produced by the same company, Texas Instrument.

The BBB is suspected of a 720 MHz high performance device of ARM A8, with 4GB 8-bit eMMC on-board flash storage and 512MB DDR3RAM. With Ethernet host, USB support. Besides, it has 64 GPIOs, ADCs, HDMI host and 3D graphic accelerator. And the different "capes" that can be added. Finally, BBB uses a MicroSD card with embedded Linux (Angstrom).

Tab 1.1: BeagleBone features

	BeagleBone
Processor	AM3358 ARM Cortex-A8
Maximal speed	1GHz
Analog pins	2*46
Digital pins	65(3.5V)
Memory	512MB DDR3 microSD
USB	miniUSB 2.0 client port, USB 2.0 host port
video	microHDMI, cape add-ones
Audio	microHDMI, cape add-ones
Supported interfaces	4*UART, 8*PWM, LCD, GPMC, MMC1, 2*SPI, 2*CAN Bus, 4 Timers

1.7.2 Why BeagleBone ?

The BeagleBone is a microcomputer with which we can innovate and make a plenty of projects. By simply connecting it through a USB cable after putting the SD card which contains an embedded Linux.

Comparing to other microprocessors, BBB is known for a huge number of connections which is very important especially while making a sophisticated project.

The access to the architecture of such microprocessor via the page of Texas instruments, this fact highlights the “open source” mentality which is not the case for Raspberry Pi.

	Raspberry Pi (Model B)	BeagleBone Black
Processor	Arm11	AM335x
Speed	700MHz	1GHz
RAM	512MB	512MB
USB		2
Audio	HDMI, Analog	HDMI
Video	HDMI, Analog	Mini-HDMI
Ethernet	10/100	10/100
I/O	8 GPIO	69 GPIO, LCD, GPMC, MMC1, MMC2, 7 AIN, 4 Times, 4 Serial Ports, CAN0
Size	3.37" x 2.125"	3.4" x 2.1"
Operating System	Linux	Android, Linux, Windows, Cloud9, CE, etc
Dev Environments/Toolkits	Linux, IDLE, OpenEmbedded, QEMU, Scratchbox, Eclipse	Python, Scratch, Linux, Eclipse, Android ADK
Cost	\$35	\$45

Fig 1.9: BeagleBone Black and Raspberry Pi comparison

1.7.3 Application examples

A variety of projects which may be made with BBB:

Ninja Blocks: a system able to ensure communication between many captors as part of Internet of Things.

BeagleStache OpenCV faces detection: an innovative project that consists of face recognition; a very useful application for security measures for big companies.

- Economic car : solar car made with BeagleBone.

1.7.4 PRU-ICSS subsystem

The Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) on the BBB's AM335x processor contain two 32-bit 200 MHz RISC cores, called PRUs. These PRUs have their own local memory allocation, but they can also use the BBB P8/P9 header pins and trigger interrupts and share memory with the Linux host device. The PRU-ICSS is a valuable addition to a general embedded Linux platform, as it can provide support for interfacing applications that have hard real-time constraints. It is important to note that the PRU-ICSS is not a hardware accelerator—it cannot be used to improve the general performance of code that is executing on the Linux host device.

Rather, it can be used to manipulate inputs, outputs, and memory-mapped data structures to implement custom communication interfaces (e.g., simple I/O manipulation, bit-banging, SPI, UARTs).

1 Uses a Xilinx Spartan 6 LX9 TQFP-144 FPGA with 9,152 logic cells, 11,440 CLB flip-flops, 16 DSP48A1 slices, and 576Kb RAM. The board also contains a 256Mb SDRAM and Arduino compatible headers, and is accessed using GPMC, SPI, or I2C from the BBB. There are two independent 32-bit RISC PRU cores (PRU0 and PRU1), each with 8 KB of program memory and 8 KB of data memory. The program memory stores the instructions to be executed by each PRU, and the data memory is typically used to store individual data values or data arrays that are manipulated by the program instructions. The PRU0 uses Data RAM0 and the PRU1 uses Data RAM1; however, each PRU can access the data memory of the other PRU, along with a separate 12 KB of general-purpose shared memory.

1.7.5 IoT subsystem interfacing

The terms *Internet of Things* (IoT) and *cyber-physical systems* (CPS) are broadly used to describe the extension of the web and the Internet into the physical realm, by the connection of distributed embedded devices. Presently, the Internet is largely an internet of people. The IoT concept envisions that if physical sensors and actuators can be linked to the Internet, then a whole new range of applications and services are possible. For example, if sensors in a home environment could communicate with each other and the Internet, then they could be “smart” about how they function.

Here are some software communication architectures that can be used to realize IoT or CPS:

The BBB Web Server: A BBB that is connected to a sensor and running a web server can be used to present information to the web when it is requested to do so by a web browser. Communications take place using the Hypertext Transfer Protocol (HTTP). The BBB Web Client: A BBB can initiate contact with a web server using HTTP requests to send and receive data. A C/C++ program is written that uses the TCP and TCP sockets to build a basic web browser, which can communicate over HTTP, or if necessary, securely over HTTPS.

The BBB TCP Client/Server: A custom C++ client and server are presented that can intercommunicate at high speeds with a user-defined communications protocol.

1.8 Embedded Linux

It is crucial to mention embedded Linux because it is compulsory for a BeagleBone Black card to have such a tool within.

Embedded Linux is an adaptation of the Linux kernel to an embedded system. It has important advantages for embedded systems since it allows porting on processors other than x86 such as PowerPC, ARM, MIPS, Cold Fire, etc. In addition, the size of the kernel is compatible with the memory sizes used in an embedded system (less than 500 KB. Finally,

Linux offers support for dynamic loading of modules which makes it possible to optimize the size of the kernel.

Embedded Linux distributions occupy a large part of the market .Depending on the capabilities of the system; it is possible to reduce the functionality of the kernel to lighten the memory required and to adapt to technical constraints.

The word embedded in the term embedded Linux is used to convey the presence of an embedded system. We cannot deny that embedded systems are present everywhere in everyday life.

1.9 Embedded system

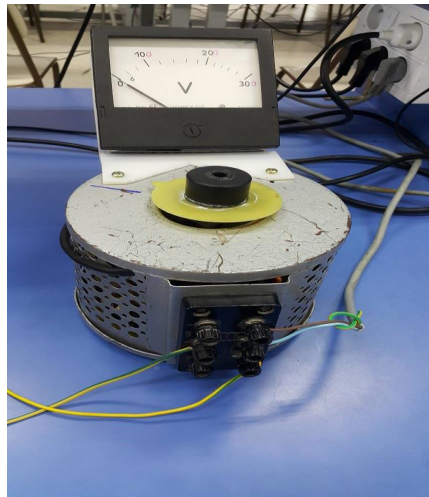
An embedded system integrates hardware and software designed specifically to perform data functions that control the systems. The hardware and software system are closely related and immersed in the hardware system so that their separation is not as easy as in a computer-like work tool.

1.10 Conclusion

DC motor speed control is an important academic subject given the different ways of its implementation, from the choice of H-bridge to the choice of the PWM technique. For our project, we made many choices to get our project of command and speed control.

Chapter 2 :

DC motor control based on BeagleBone Black



2.1 Introduction

Our main focus in this chapter is to describe in detail the laboratory part in which we worked on over a complete semester. The aim of this project was the development of a code to command and to control the speed of the DC motor. The project also includes the design and development of a power circuit including the variety of needed components.

2.2 The DC motor

The DC motor we worked on during our end year project is one of the American G.K HELER CORP mock-up.

We faced some problems because the parameters of the motor were not mentioned so we could not easily get a model for the engine. We made a variety of essays to know approximately the values of the parameters

As we all know, a DC motor includes two parts; the armature and the inductor and each side of them has its internal resistance. So we had to measure two resistances; the bigger one belongs to the inductor and the small one is for the armature. Once traversed by a direct current, we can measure the resistance of the motor and then one applies at its terminals an alternating tension to measure the impedance and then we can deduct the inductance of the two different parts. We took practically three measures in order to get the average which can potentially reach the fair value.

Tab 2.1: Different values of Rinductor

Voltage (V)	60	55	50
Current (mA)	70	65	53
Resistance(Ω)	850	840	850

Rinductor=845 Ω

Tab 2.2: Different values of Rarmature

Voltage (V)	20	25	30
Current (A)	1.59	1.59	2.32
Resistance(Ω)	12.69	12.84	12.93

Rarmature=12.7 Ω

Now, after looking to the motor nameplate as shown below, we can notice that is both the inductor and the armature are powered from the same voltage supply, this might be meaning that we need 115V for each one of them separately to make the engine work properly.



Fig 2.2 : The nameplate signal

2.3 Hardware design

"Confusion and clutter are the failure of design, not the attributes of information"
-Edward Tufte-

Controlling a DC motor actually needs an association of different components to operate in an efficient way; this part talks about the variety of components we dealt with in order to get different rates of power supply to our DC motor. The components are put in the order of which the signal flows.

2.3.1 Auto transformer

An Auto-transformer is an electrical transformer with only one winding. The "auto" prefix refers to the single coil acting alone and not to any kind of automatic mechanism. In an autotransformer, portions of the same winding act as both the primary and secondary sides of the transformer. In contrast, an ordinary transformer has separate primary and secondary windings which are not electrically connected.

Autotransformers are often used to step up or step down voltages in the 110-115-120 V range and voltages in the 220-230-240 V range. For example, providing 110 V or 120 V from 230 V input, allowing equipment designed for 100 or 120 V to be used with a 230 V supply which is exactly the case of our project; we can provide voltage for the engine from the mains voltage.

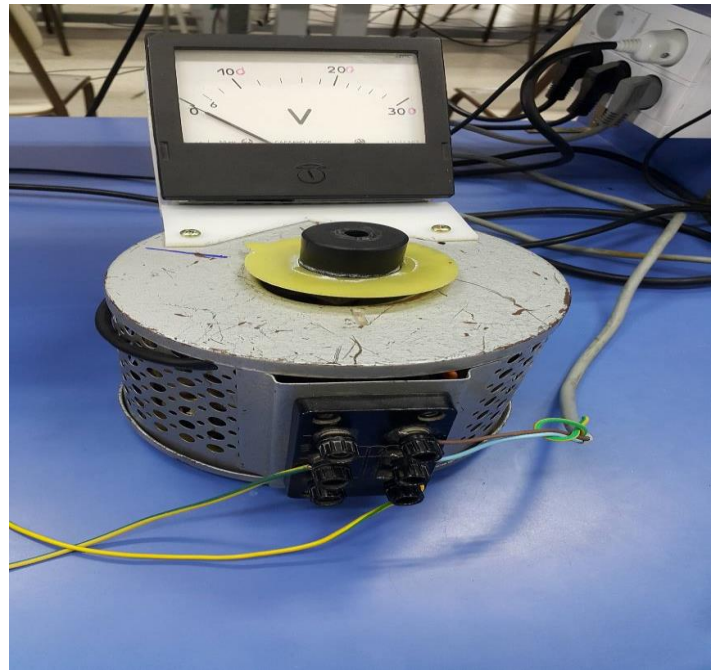
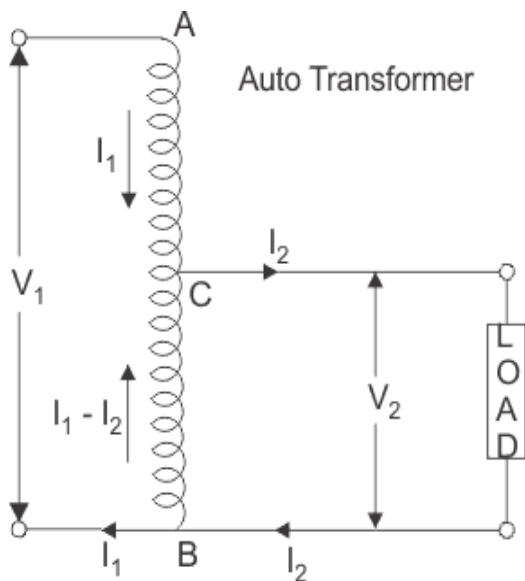


Fig 2.3 : Auto transformer

2.3.2 Stabilized power supply

A stabilized power supply is a circuit which converts unregulated AC into a constant DC. Its function is to supply a stable voltage, to a circuit or device that must be operated within certain power supply limits. The output from the regulated power supply may be alternating or unidirectional, but is nearly always DC.

In our project, it provides well regulated and stabilized output, which is essential for most stages of circuits in the power card to give proper results.

The power supply to the circuit ‘under test’ can be cut-off immediately to save the valuable components from damage. The circuit provides three different regulated outputs 15V and 12V. The circuit uses a standard 17V–0–17V, 2.5A step-down transformer to generate 17V AC.

A rectifier diode provides 18V DC, which is smoothed by capacitor and given to the combination of regulated 7800s (7815 to provide voltage for IR2110 driver, 7812 to provide voltage for speed sensor). The regulated 87xx’s produce fixed, regulated outputs of 15V and 12V, respectively, which are connected to the different mentioned parts of the circuit.

The picture below shown a stabilized power circuit of the motor (at the right) and of the H-Bridge circuit (at the left).

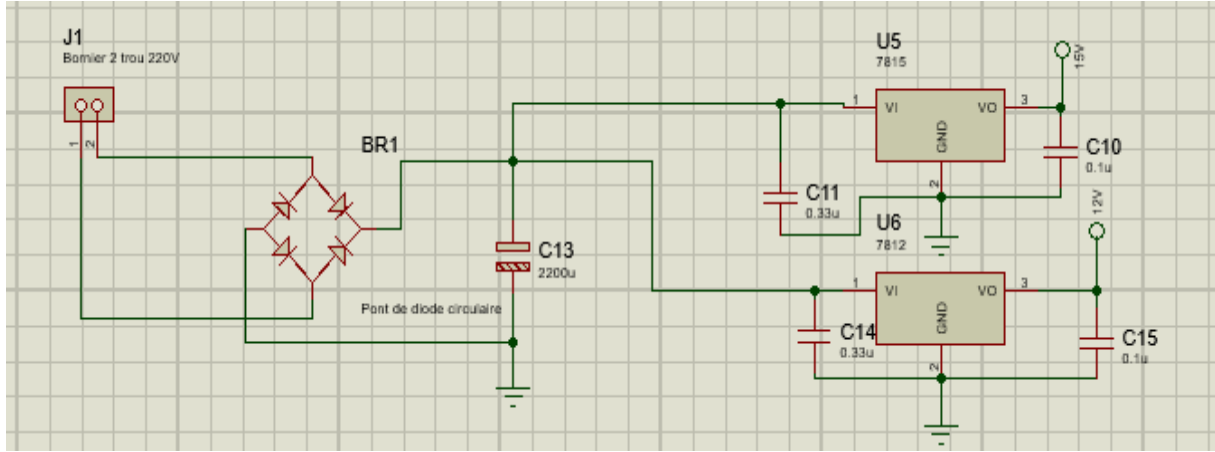


Fig 2.4: 12 and 15 voltage supply

2.3.3 H-bridge

In this project, we dealt with two different architectures of full H-bridge, either with IGBT or MOSFET.

2.3.3.1 GT15J101 IGBT

The IGBT is used in medium- to high-power applications like switched-mode power supplies, traction motor control and induction heating. Actually, it can have high current-handling capability with blocking voltages. An IGBT features a significantly lower forward voltage drop compared to a conventional MOSFET in higher blocking voltage rated devices, although, MOSFETS exhibit much lower forward voltage at lower current densities.

2.3.3.2 IRF840 MOSFET

The main advantage of a MOSFET is that it requires almost no input current to control the load current, when compared with bipolar transistors. In an enhancement mode MOSFET, voltage applied to the gate terminal increases the conductivity of the device. In depletion mode transistors, voltage applied at the gate reduces the conductivity. Besides, MOSFETs are generally used while working in a high rate of frequency.

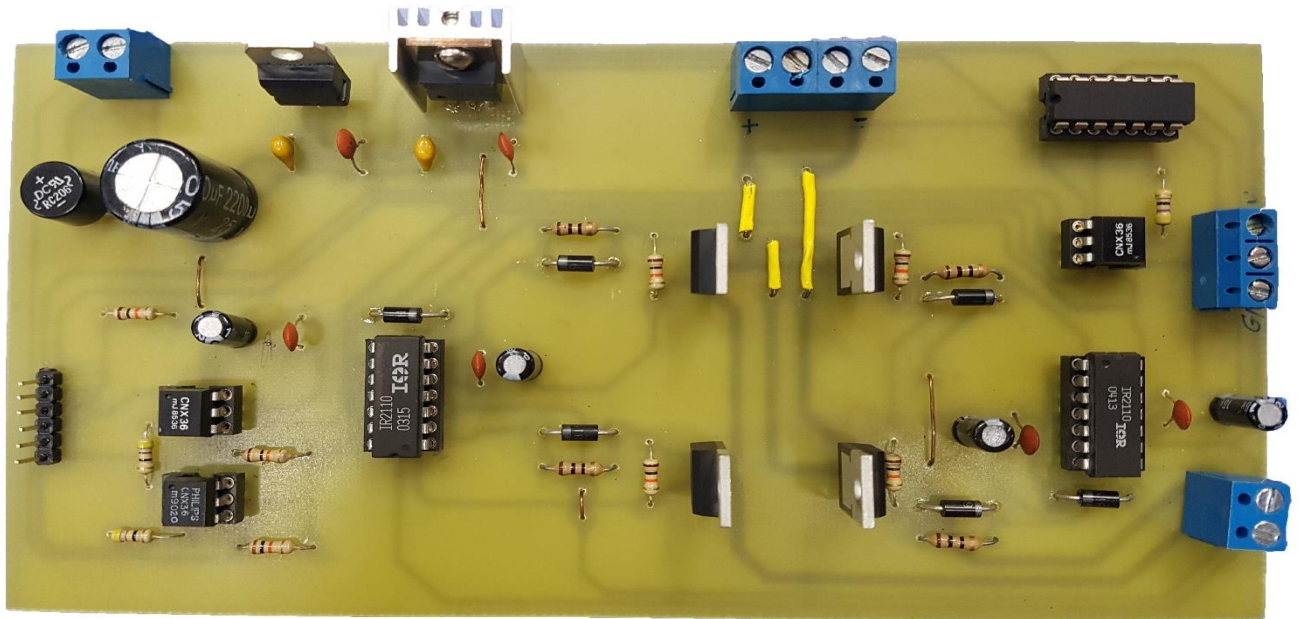


Fig 2.5: MOSFET H-bridge

This two architectures need to be driven with a specified Integrated control circuit. Many integrated control circuits are available, it all include an output stage behaving as a power interface with bipolar transistors or MOSFETs.

2.3.3.3 IR2110 driver

The IR2110 are high voltage, high speed power MOSFET drivers with high and low side output channels.

but we must mention that it is not that easy as the picture show. There is some problems of controlling the high side switch, the source of the latter being at a potential that varies between 0V and VDC. By high side, we call the transistor stors whose source electrode for the MOSFETs or emitter for IGBTs which are placed at a floating potential that can be high, and varies rapidly from the value of the reference circuit of the command.

It is therefore necessary to create a supply of the high-side control circuit which can follow theses very rapid variations of potential, and also Isolate or shift the potential of the logic command to the potential of the close-up control circuit of the high-side switch.

Typical Connection

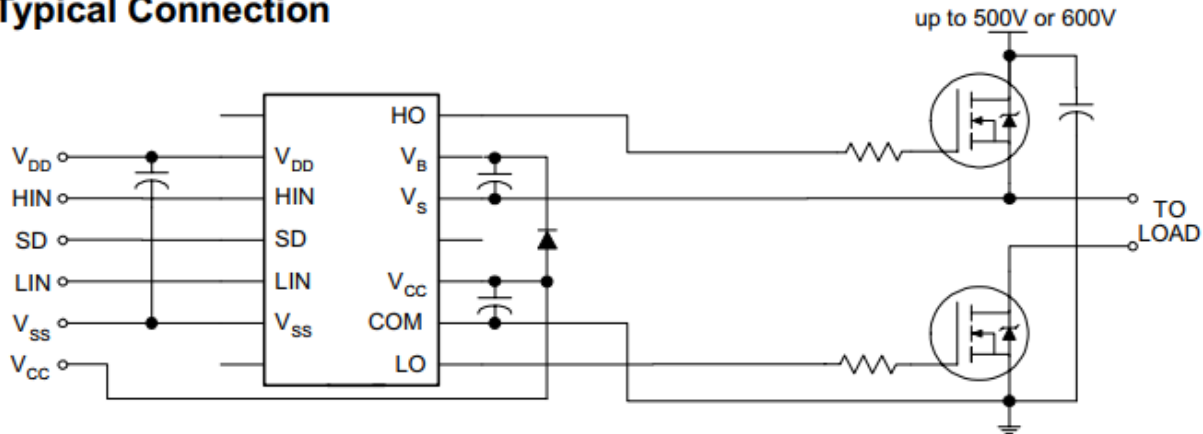


Fig2.6: IR2110 driver

2.3.3.4 Bootstrap circuit

In the field of electronics, a bootstrap circuit is one where part of the output of an amplifier stage is applied to the input, so as to alter of a system is used at startup is described as bootstrapping. input impedance of the amplifier. When applied deliberately, the intention is usually to increase rather than decrease the impedance. Generally, any technique where part of the output.

The bootstrap circuit is useful in a high-voltage gate driver and operates as follows. When the VS goes below the IC supply voltage VDD or is pulled down to ground (the low side switch is turned on and the high-side switch is turned off), the bootstrap capacitor, CBOOT, charges through the bootstrap resistor, RBOOT, and bootstrap diode, DBOOT, from the VDD power supply, as shown in Figure 2. This is provided by VBS when VS is pulled to a higher voltage by the high-side switch, the VBS supply floats and the bootstrap diode reverses bias and blocks the rail voltage (the low-side switch is turned off and high-side switch is turned on) from the IC supply voltage, VDD.

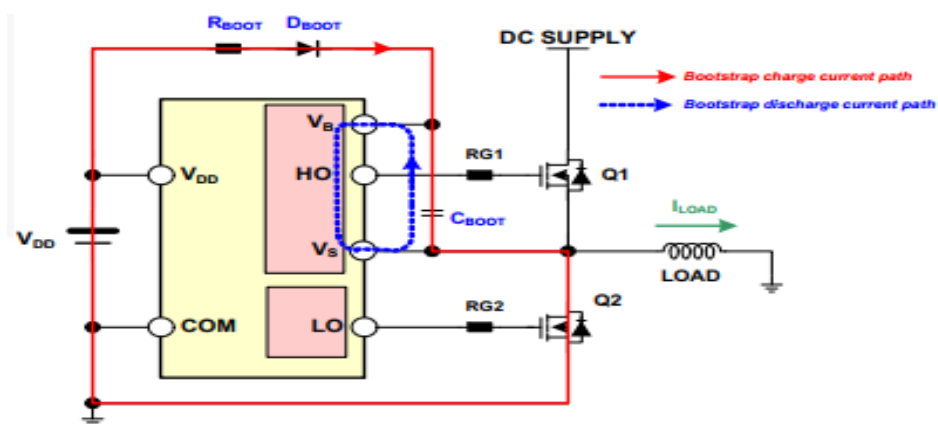


Fig 2.7: Bootstrap circuit mounting

- The bootstrap resistor (RBOOT) must be considered in sizing the bootstrap resistance and the current developed during initial bootstrap charge. If the resistor is needed in series with the bootstrap diode, verify that VB does not fall below COM (ground), especially during startup and extremes of frequency and duty cycle.
- The bootstrap capacitor (CBOOT) uses a low-ESR capacitor, such as ceramic capacitor. The capacitor from VDD to COM supports both the low-side driver and bootstrap recharge. A value at least ten times higher than the bootstrap capacitor is recommended. Here we use 22uF ceramic capacitor paralleled with 100nF ceramic capacitor to gate a low-ESR capacitor.
- The bootstrap diode must have a lower forward voltage drop and switching time as soon as possible for fast or even ultra-fast recovery . 1N5819 can be the perfect choice .

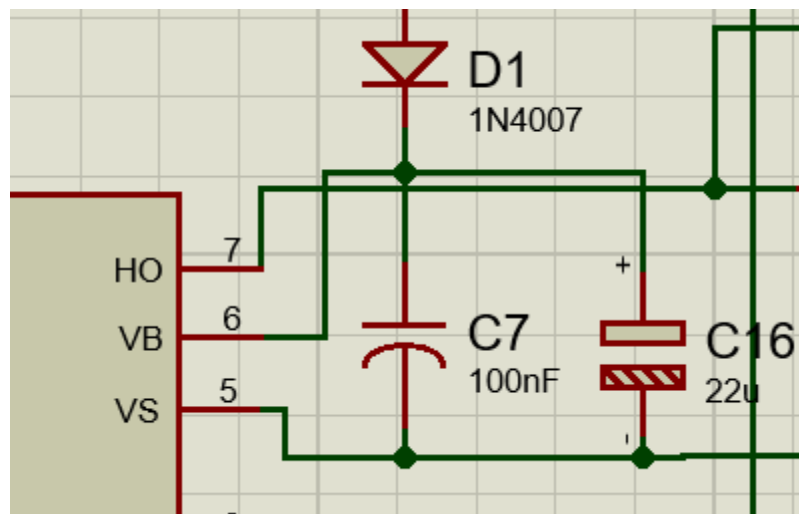


Fig 2.8:Bootstrap circuit schematic

2.3.3.5 Command signals

Will each of the two IR2110 driver needs to be controlled with LIN and HI signals to control the High and the Low side of the H-bridge to get two direction rotation

First direction : HIN = PWM
LIN = GND

HIN =GND
LIN = PWM ;

Second direction : HIN = GND
LIN = PWM

HIN = PWM
LIN = GND ;

2.3.4 CNX36U opto coupler

In our project, galvanic isolation is guaranteed by the use of an optocoupler from the CNX36U family; optically coupled isolators consisting of an infrared emitting GaAs diode and a silicon NPN phototransistor with accessible base. In fact, it drives the 3.3V voltage coming from the BeagleBone through the serial link bus to the H-bridge's driver.

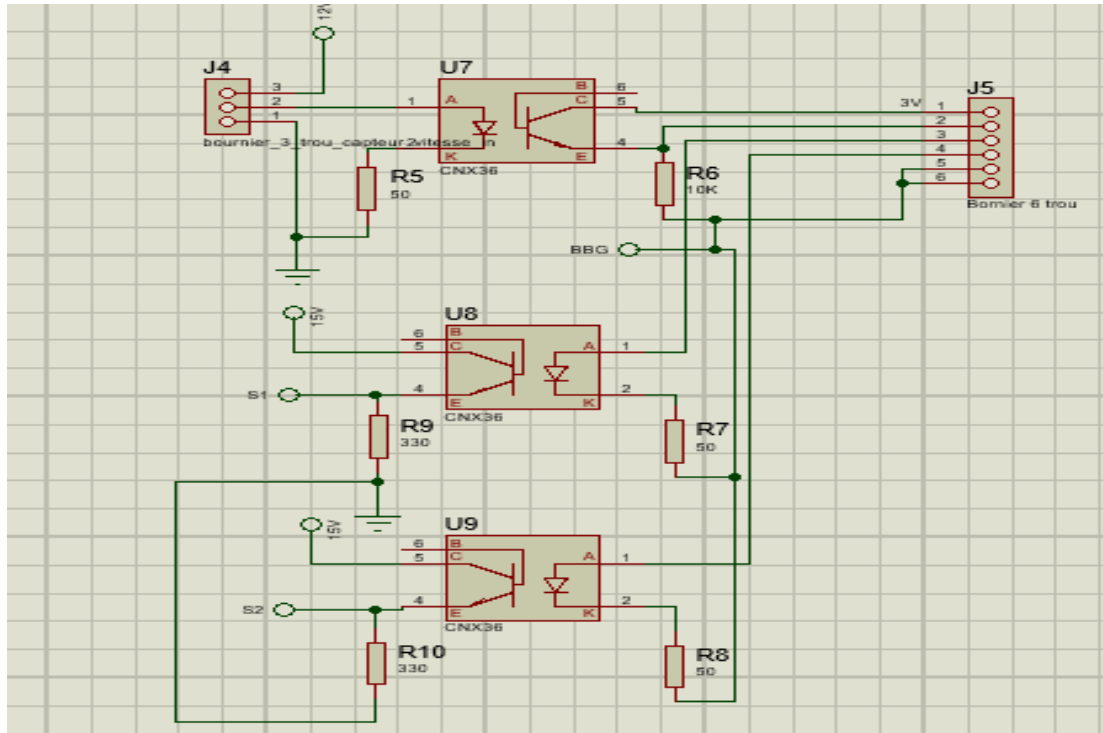


Fig 2.9: Opto coupler schematic

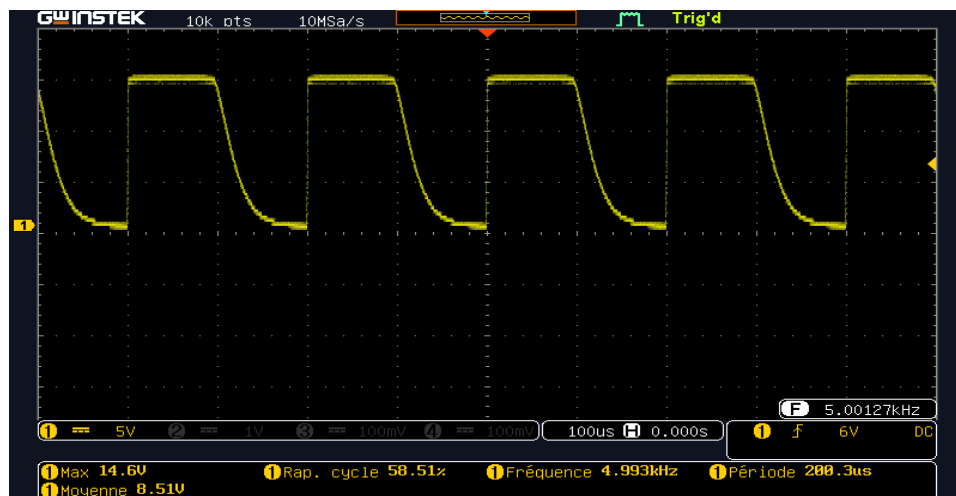


Fig 2.10: Optocoupler output signal

2.3.5 HEF40106B inverting trigger

Each circuit of the HEF40106B functions as an inverter with Schmitt-trigger action. The Schmitt trigger switches at different points for the positive and negative-going inputs signals. The difference between the positive-going voltage and the negative-going voltage is defined as hysteresis voltage. The device may be used for enhanced noise immunity or to “square up” slowly changing waveform.

Noise elimination is one of the main problems in achieving higher data rates and longer communication cables. There is quite some noise on the line, and if the center voltage is used to determine if the input is in the on, or off state, there can be four high periods detected in the signal as shown in the second graph. If we however use two trigger levels with hysteresis, it is possible to reduce the number of detected logical highs to only one.

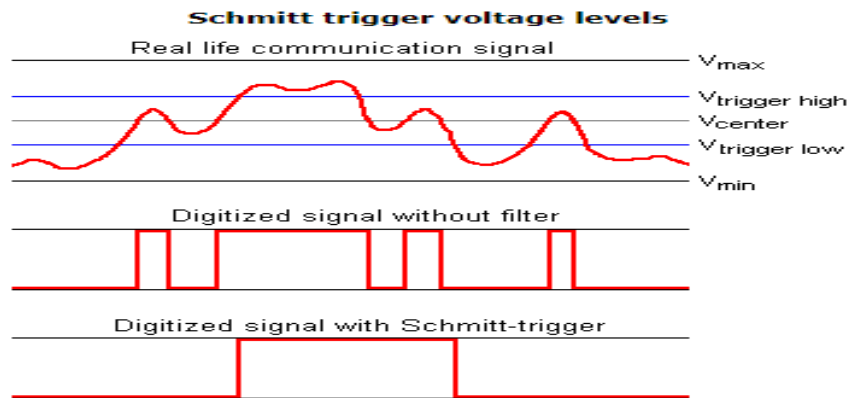


Fig 2.11: Input and output signals of the schmitt trigger

2.3.5.1 why double inversion ?

As we can notice, there are signals which are passed through the inverter two times and it seems useless to do so. The output of the trigger is amplified and re-organized, hence, we make a signal goes into the device twice to obtain the wanted form without dimming.

As we can show in the picture below the inverting trigger can correct the optocoupleur output signal to insure that our H-bridge can work correctly even with high frequency.

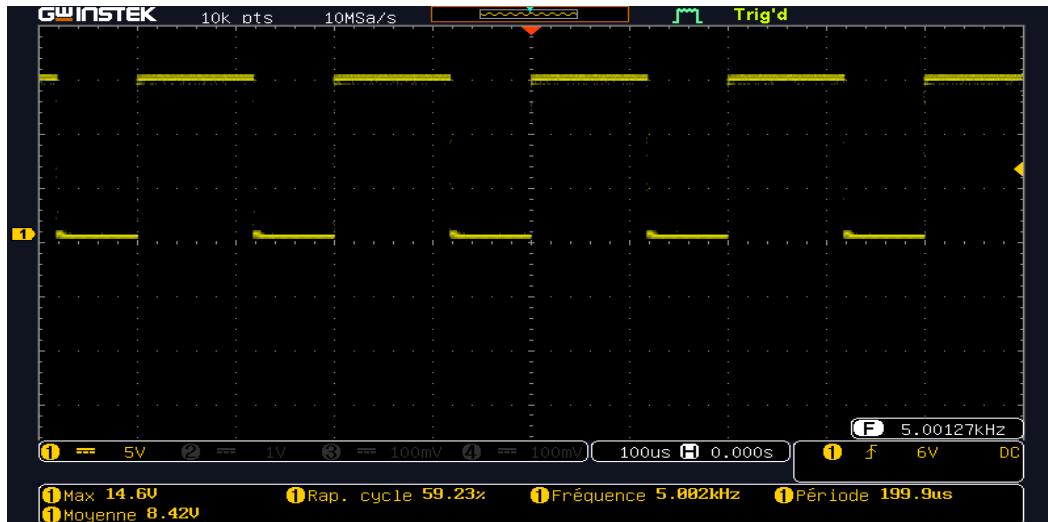


Fig 2.12: Input and output signals of the schmitt trigger

2.3.5.2 Why high frequency ?

PWM controls produce smoother current, without any fluctuations, at higher switching frequencies, typically between 1 and 20 kHz. At 20 kHz, the switching frequency is inaudible to humans, thereby eliminating the hum which switching at lower frequency produces.

2.3.6 Tests and simulations

The phase of simulation is made to valid the theoretical analysis, to detect potential errors so as we can avoid so many damages while working on other steps of the project. The software weapon used in this section is Proteuse8 ISIS.

After many tests, we found that we can attend 10Khz on the MOSFET H-bridge. This limitation was due the optocoupler. Even with the shmit-trigger, the signal will be completely destroyed after 12Khz.

Many research and efforts led us to find that we can increase the optocoupler speed by:

connecting a parallel RC (270K // 470pF) network on the B-E junction to double the turn-off delay and to avoid erratic operation in the presence of large electrical disturbances.

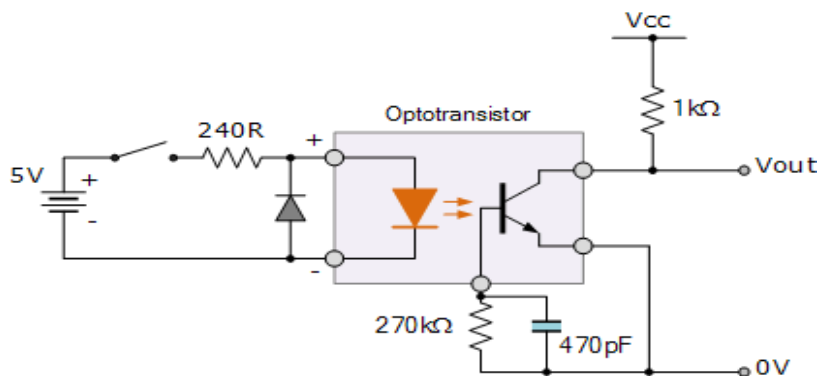


Fig 2.13: high speed opto coupler mounting

Finally, we ended up with two power cards so as we can control the DC motor differently; either with IGBT H-bridge or MOSFET H-bridge.

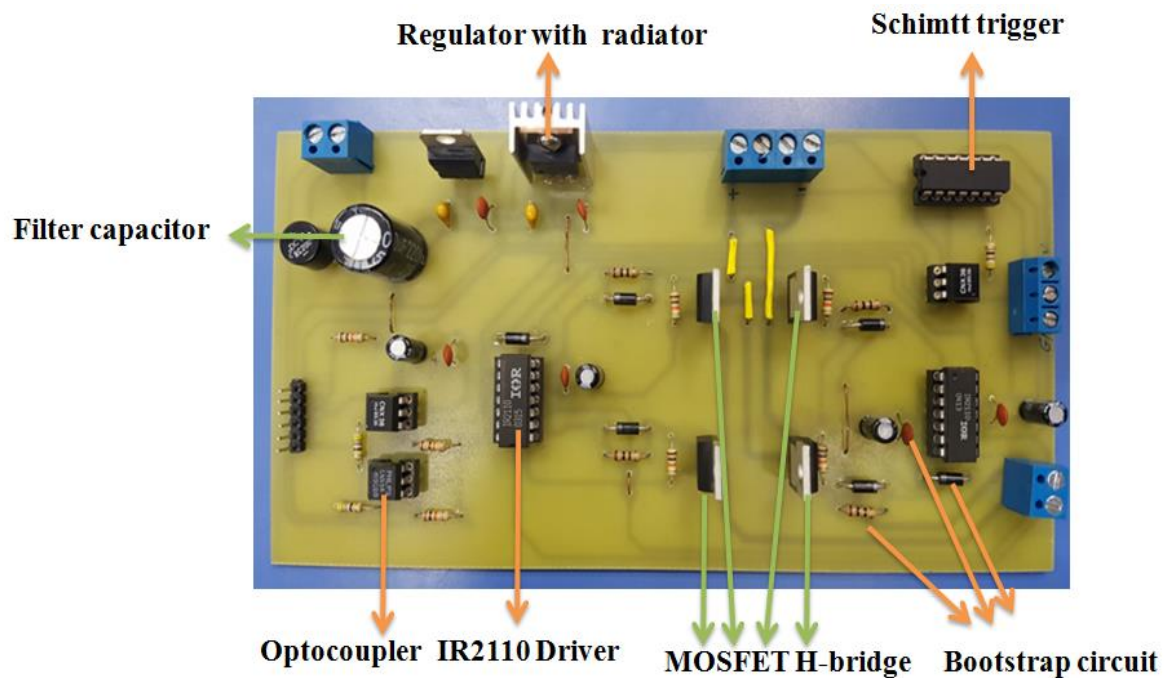


Fig 2.14: Power card with different components

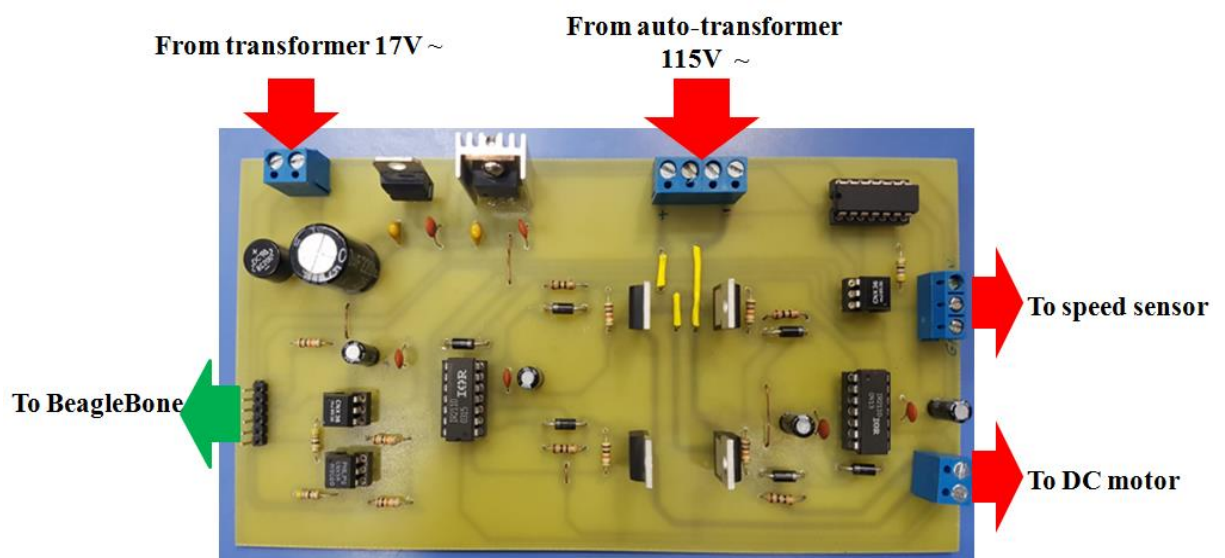


Fig 2.15: Input and output of the power card

Now let's move to software part, in which, we will create our PWM signal from the command board and get the current speed for the PID implementation.

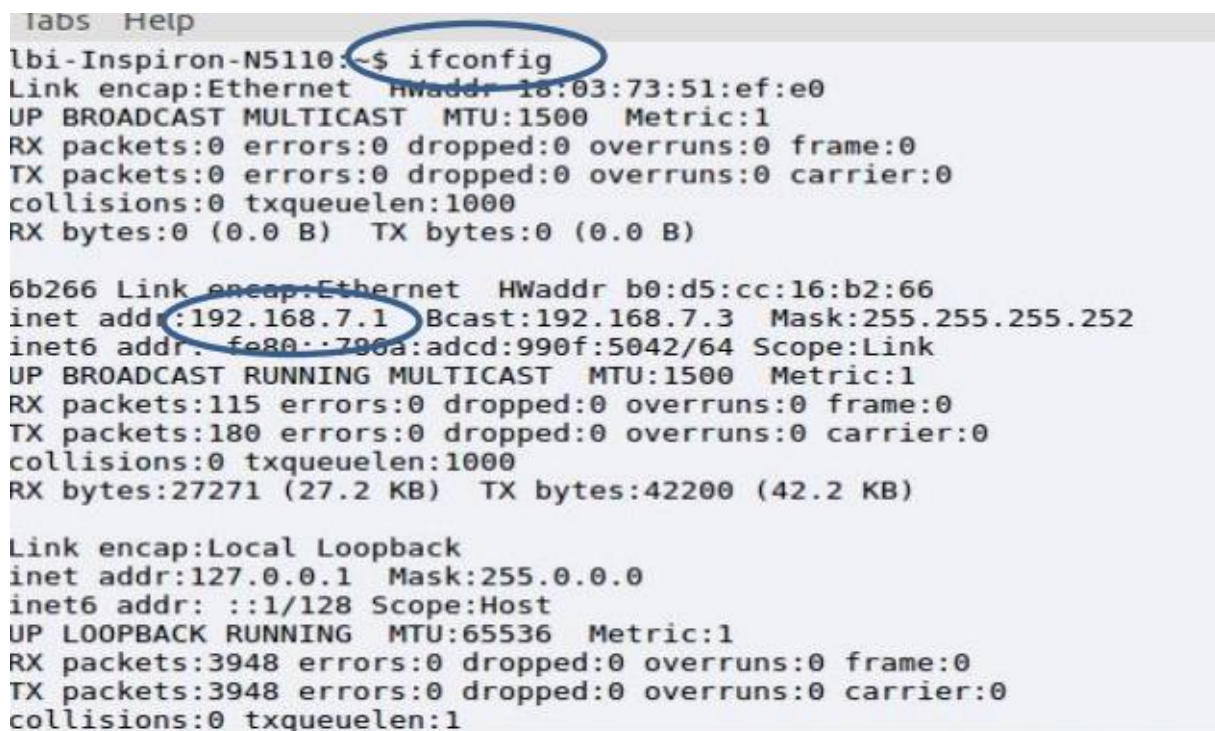
2.4 Software side

This section deals with the different steps we did to complete the functionality of the hardware grouping.

2.4.1 Getting started

All Beaglebone versions have a USB port that allows for a LAN between the Beaglebone card and other machines. The access to the card is done via the protocol "ssh" (secure shell), it is a protocol of secure communication that allows remote access to a machine when both machines are on the same network.

To start the connection, connect the card to a computer via a USB 2.0 cable or 3.0, then type the command "ifconfig" to check the connection with the card.



```
lbi-Inspiron-N5110:~$ ifconfig
Link encap:Ethernet HWaddr 18:03:73:51:ef:e0
UP BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

6b266 Link encap:Ethernet HWaddr b0:d5:cc:16:b2:66
inet addr:192.168.7.1 Bcast:192.168.7.3 Mask:255.255.255.252
inet6 addr: fe80::796a:adcd:990f:5042/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:115 errors:0 dropped:0 overruns:0 frame:0
TX packets:180 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:27271 (27.2 KB) TX bytes:42200 (42.2 KB)

Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:3948 errors:0 dropped:0 overruns:0 frame:0
TX packets:3948 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
```

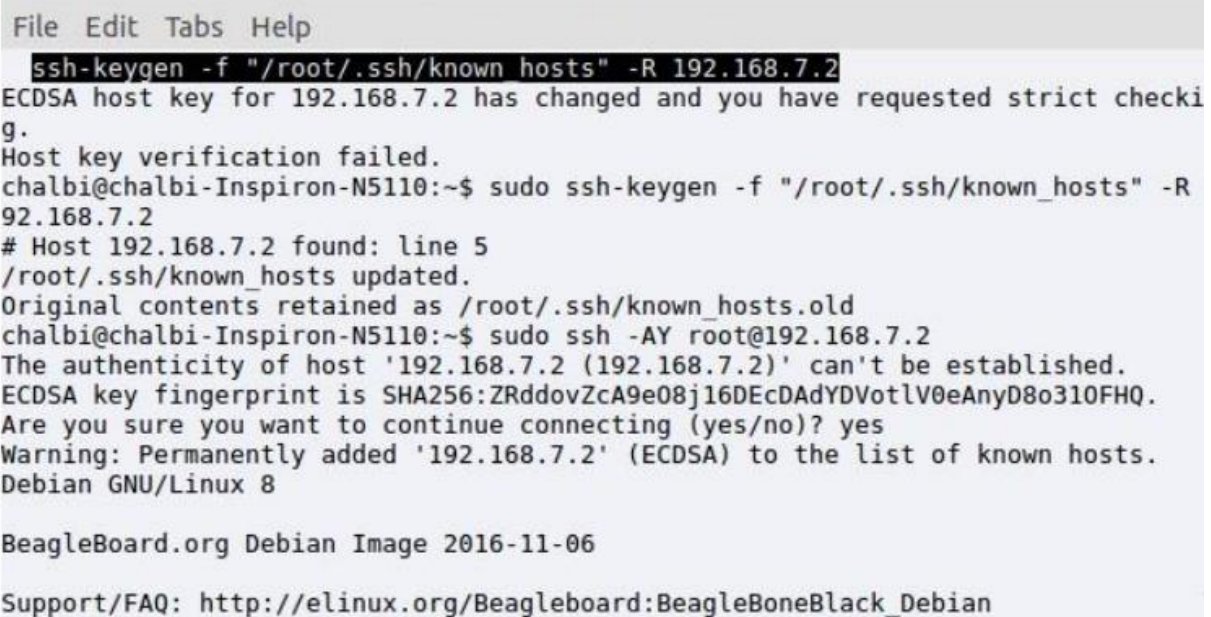
Fig 2.16: Ifconfig

Then we type the command "**Sudo ssh -X username @ ip_BeagleBone**".

"-X": allows us to run X Window graphical interfaces via the IP address.

"Username": it can be either the super user "root" (without password) or the default personal user "Debian" (password: tpw). To ensure security, we have to change the passwords since they are standard by default. To modify the authentication, we use the "passwd" command.

"Ip_BeagleBone" is the address '192.168.7.2' which is a static address in the case which uses a USB cable. Since linux is multiuser, other users can access to the beagleBone card. Yet, we just have to connect it to a local network, either wirelessly (by wifi key) or by an RJ45 Ethernet cable. In this case, we type the IP address of the Beaglebone (we can know by typing the command "ifconfig" on the terminal Beaglebone) in this network. This address is dynamic, this is the address that the DHCP server assigns to it. As all Beaglebone cards have the same static IP address '192.168.7.2' and the protocol "ssh" assigns each card an IP address linked to a single address mac (physical address), so if we want to connect another card to the machine via a USB cable, we must remove the link from the IP address of the other card and replace it with that of the new by typing the command "sudo ssh-keygen -f /root/.ssh/known_hosts -R 192.168.7.2 "



```
File Edit Tabs Help
ssh-keygen -f "/root/.ssh/known_hosts" -R 192.168.7.2
ECDSA host key for 192.168.7.2 has changed and you have requested strict checking.
Host key verification failed.
chalbi@chalbi-Inspiron-N5110:~$ sudo ssh-keygen -f "/root/.ssh/known_hosts" -R 192.168.7.2
# Host 192.168.7.2 found: line 5
/root/.ssh/known_hosts updated.
Original contents retained as /root/.ssh/known_hosts.old
chalbi@chalbi-Inspiron-N5110:~$ sudo ssh -AY root@192.168.7.2
The authenticity of host '192.168.7.2 (192.168.7.2)' can't be established.
ECDSA key fingerprint is SHA256:ZRddovZcA9e08j16DEcDAdYDVotlV0eAnyD8o310FHQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.7.2' (ECDSA) to the list of known hosts.
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2016-11-06

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
```

Fig 2.17: IP adress changing

2.4.2 Linux development with the Beaglebone card

Actually, there are two kinds of development with the BeagleBone card.

2.4.2.1 Cross Compilation

it is a kind of programming whose development is done under another machine (Desktop). It compiles it with a compiler compatible with the processor of the platform used (gcc-arm in our case), then we just send the executable file to the BeagleBone card.

IDEs (Integrated Development Environment) such as Qtcreator and Eclipse ensure this method. This process becomes more and more complicated when using libraries that

are not standard in the compiler.

2.4.2.2 Direct development on the BeagleBone card

Direct development is present while using publishers who are not bulky (like "geany") with small libraries.

Development with the "geany" editor requires the use of "Makefile" files. The "Makefile" file is a simple way to organize code compilation, in which one specifies the compiler, the paths of the files "header" and "Source" and the path of the bin directory where the compiler creates the executable file. "Geany" is a text editor of different languages like C, C ++, python, verilog, etc. It is small and simple to use. To install it, type the command "apt-get install geany". In fact, it requires libraries that must be in the system, to do so, if they did not exist, we must type the "apt-get update" command to download its libraries, then retype the "apt-get install geany" command. We can also check the installation by typing "Whereis geany". Finally, to execute it, we must type "geany &> / dev / null &". This command allows you to execute it without blocking the terminal. It is also better to download a file explorer like "xfe" or "Pcmanfm" to facilitate the manipulation of files. To use "geany", we have to configure it according to the type of development and the type of desired compiler.

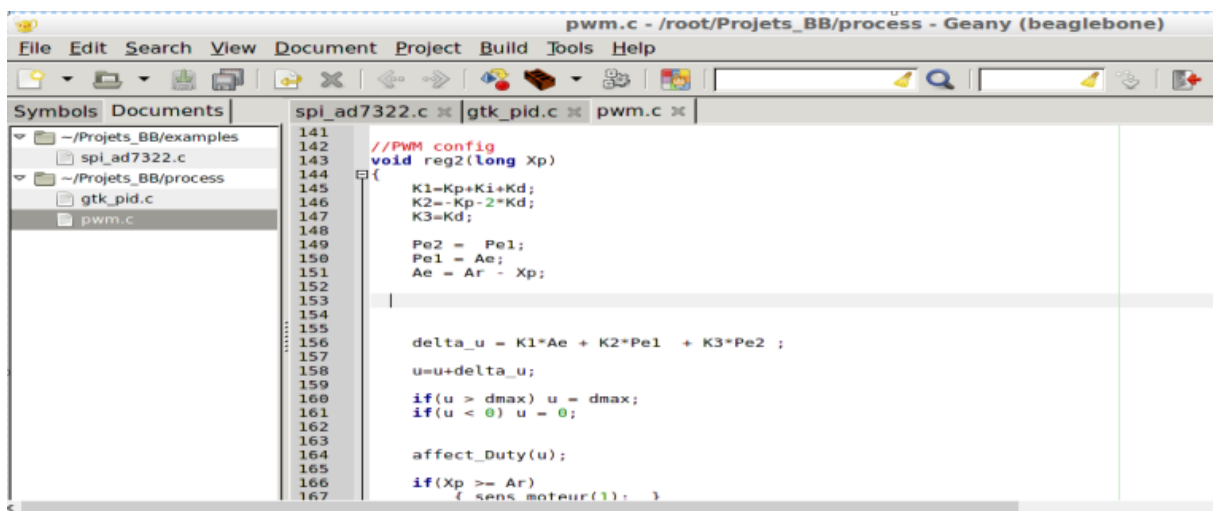


Fig 2.18: Geany environment

In this stage we are almost ready. In fact, we decided to use one of the most advanced feature in the Beaglebone board : PRU-ICSS. Obviously, the beaglebone board is a SoC (System on Chip) subsystem Which contains two PRUs; programmable real-time unit (PRU): PRU0 and PRU1.

2.4.3 PRU-ICSS

During the implementation of our project, we dealt with PRU-ICSS Which contains two PRUs; PRU0 and PRU1.

A programmable real-time unit (PRU) is a fast (200-MHz, 32-bit) processor with single-cycle

I/O access to a number of the pins and full access to the internal memory and peripherals on the AM335 processor on BeagleBone Black. It is designed to provide software-defined peripherals as part of the Programmable Real-time Unit Industrial Control SubSystem (PRU-ICSS) and are capable of implementing things like 25 pulse-width modulators, stepper motor drivers and much more. Having these controllers integrated is really handy to avoid throwing in another device to control or interface to a new peripheral.

- Programmable Real-Time Unit (PRU) is a low-latency microcontroller subsystem
- Two independent PRU execution units
 - 32-Bit RISC architecture
 - 32 General Purpose Registers
 - 200MHz – 5ns per instruction
 - Single cycle execution - No pipeline
 - Dedicated instruction and data RAM per core
 - Shared RAM
- Includes Interrupt Controller for system event handling
- Fast I/O interface
 - Up to 30 inputs and 32 outputs on external pins per PRU unit

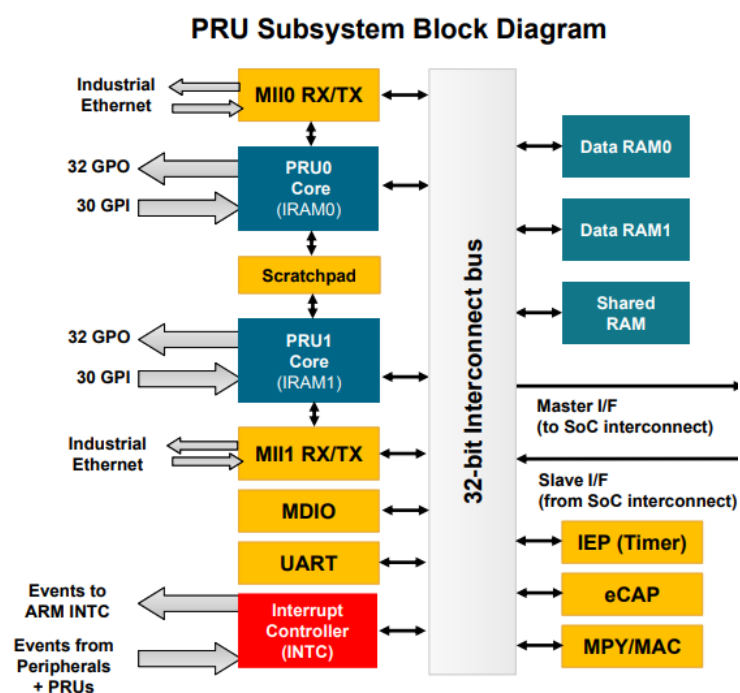


Fig 2.19: PRU-ICSS architecture

2.4.4 Libpruio

To make this two Subsystems work properly, we used a whole library called “Libpruio” in which there is many pre-defined functions to implement. But we must install this library first, in the beaglebone it self.

Install libpruio

Download and uncompress package from this commands

```
wget http://www.freebasic-portal.de/dlfiles/539/libpruio-0.2.tar.bz2
```

```
tar xjf libpruio-0.2.tar.bz2
```

Copy files

```
cd libpruio-0.2
```

```
cp src/c_wrapper/libpruio.so /usr/local/lib
```

```
ldconfig
```

```
cp src/c_wrapper/pruio*.h* /usr/local/include
```

```
cp src/config/libpruio-0A00.dtbo /lib/firmware
```

```
cp src/pruio/pruio*.bi /usr/local/include/freebasic/BBB
```

```
cp src/pruio/pruio.hp /usr/local/include/freebasic/BBB
```

Activate the PRUSS by enabling the tree overlay.

```
echo libpruio > /sys/devices/bone_capemgr.slots
```

Pre-conditions

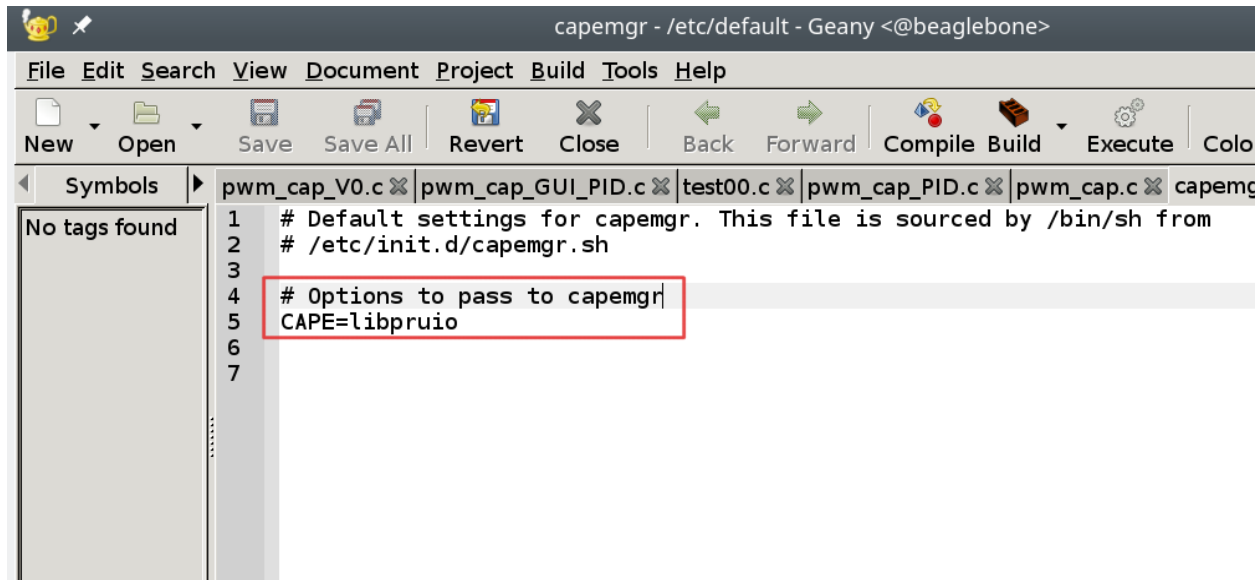
After installing the library package we must insure that the kernel driver *uio_pruss* is loaded, and the PRU subsystems are enabled by loading an appropriate device tree overlay.

The Linux Device tree controls the configuration of pins: mux mode, direction, pullup/pulldown, etc. It is a generic, standardized way of dealing with constantly updating hardware configurations. In short, it describes the hardware in a system. It is the linux version of ARM board files.

The device tree is compiled into a `.dtb` file that is then loaded upon the kernel's startup. All specified pins and peripherals are loaded and configured for use in the system. We can install and load the universal overlay (file `src/config/libpruio-00A0.dtbo` must get copied to folder `/lib/firmware`) shipped with the libpruio package by executing `:sudo echo libpruio > /sys/devices/bone_capemgr.slots`

It's also possible to activate the device tree overlay at system startup by adapting the configuration file:

`sudo geany /etc/default/capemgr` to make it look like :



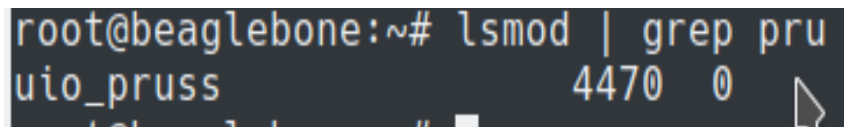
/Fig 2.20: Device tree configuration at start up

After booting the BeagleBone, we can verify that the device tree is correctly loaded by default in the bone_capemgr folder.

```
root@beaglebone:~# cat /sys/devices/bone_capemgr.*/slots
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
7: ff:P-O-L Override Board Name,00A0,Override Manuf,libpruio
```

Fig 2.21: Device tree loaded

As we can see in the picture below by typing **lsmod | grep pru**, the uio driver for pru microcontroller is working now:



```

root@beaglebone:~# lsmod | grep pru
uio_pruss                4470  0
  
```

Fig 2.22: UIO driver activated

Now we will talk about the main programm based on these principle points :

*** isleep fonction :** this fnction will create a delay time to set sampling time and take as input an unsigned int that represent how many mellisecond betwin two sampling.

```

41  int isleep(unsigned int mseconds)
42  {
43      fd_set set;
44      struct timeval timeout;
45
46      /* Initialize the file descriptor set. */
47      FD_ZERO(&set);
48      FD_SET(STDIN_FILENO, &set);
49
50      /* Initialize the timeout data structure. */
51      timeout.tv_sec = 0;
52      timeout.tv_usec = mseconds * 1000;
53
54      return TEMP_FAILURE_RETRY(select(FD_SETSIZE,
55                                     &set, NULL, NULL,
56                                     &timeout));
57  }
  
```

Fig 2.23:Isleep fuction

In the main fonction we will essentialy do this steps :

*** pruIo *Io = pruio_new(PRUIO_DEF_ACTIVE, 0x98, 0, 1) :** create new driver structure

*** pruio_pwm setValue(Io, P_OUT, f0, d0) :** this will affect f0 to frequency and d0 to duty cycle of the Io stucture

*** pruio_cap Value(Io, P_IN, &f1, &d1) :** this will affect the input signal of the eCap to the f1 input frequency and d1 input duty cycle .

As first test we put the output PWM signal P9_21 to the eCAP input P9_42 Wiring diagram for pwm_cape.

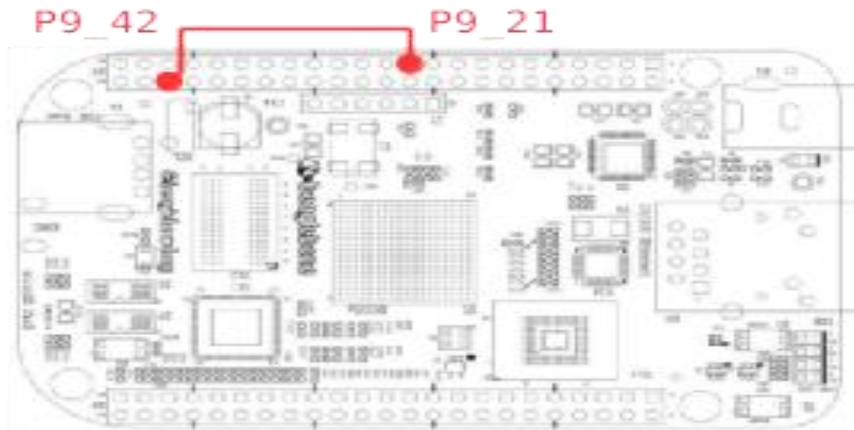


Fig 2.24: Pwm eCap test connexion

the first line show us the input parameters for the PWM signal and the second show us thw eCap values. In this stage we try to assamble all the project parts(H-Bridge + BBB) together and its work very will.

```
--> Frequency: 1024000.000000 , Duty: 0.700000
      Frequency: 1020408.187500 , Duty: 0.714286
```

Fig 2.14: PRU-ICSS architecture

Now it's time to go further with this project and to add some features.

2.5 User interface

The user interface is a type of computer interface that allows a user to manipulate the machine. It coordinates human-machine interactions and is part of the design of our control project.

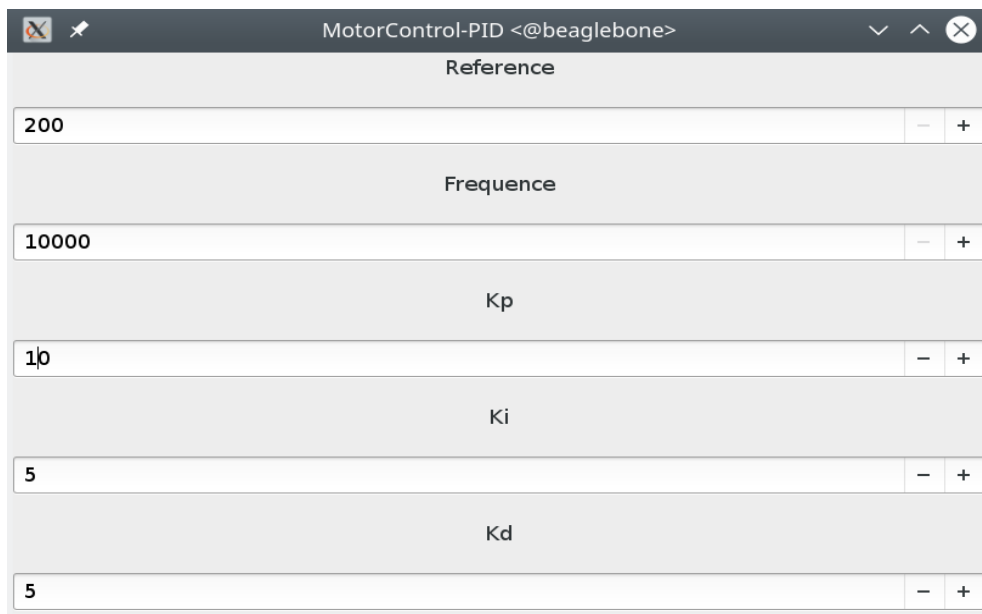


Fig 2.25: User interface

During the implementation of the interface, we used the library "gtk_lib" applied to the C language. Geany software was the tool for building the design of the window.

2.6 PID implementation

A standard "textbook" equation of PID controller is:

$$u(t) = K \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right)$$

But, as we are dealing with a digital control project, we need a compatible equation which bests the situation of BeagleBone that's why we need a discretization for this formula, we obtain the equation below:

$$U(z) = \left[K_p + \frac{K_i}{1 - z^{-1}} + K_d (1 - z^{-1}) \right] E(z)$$

$$U(z) = \left[\frac{(K_p + K_i + K_d) + (-K_p - 2K_d)z^{-1} + K_d z^{-2}}{1 - z^{-1}} \right] E(z)$$

the used variables in this PID algorithm are:

$$K_1 = K_p + K_i + K_d$$

$$K_2 = -K_p - 2K_d$$

$$K_3 = K_d$$

```

82  ///***** PID_1 *****
83  void reg2(float_t Fm)
84  {
85      K1=Kp_val+Ki_val+Kd_val;
86      K2=-Kp_val-2*Kd_val;
87      K3=Kd_val;
88
89      Pe2 = Pe1;
90      Pe1 = Ae;
91      Ae = (Fr - Fm);
92
93      delta_u = K1*Ae + K2*Pe1 + K3*Pe2 ;
94
95      d0=d0+delta_u;
96      d0=d0*1;
97
98      if(d0 >= dmax) d0 = dmax;
99      if(d0 <= dmin) d0 = dmin;
100
101  }
102  /*****

```

Fig 2.26: PID tuning program

2.7 Conclusion

The different phases of working on this project show the huge number of details which contribute in the achievement of the goal we've put when starting this end of the year project.

General conclusion

The purpose of our End Year Project was to pursue a well known embedded system

through a DC motor. To accomplish this project, we began by a preliminary study to understand embedded system field and to touch 'Linux' environment. Then, we've presented both the hardware and software parts: embedded BeagleBone Black card, the DC motor and the power card, to end up with mentioning all the steps which led to the accomplishment of our command project. Indeed, the portability of embedded Linux gave us the capacity to work on other platforms other than BeagleBone Black.

This work allowed us to enhance our capacity to develop a similar project with the evolution of our autonomy in solving problems. In fact, it was a good opportunity to confront our theoretical knowledge with practical situations and finally to extend our knowledge.

We would like to emphasize that the control of our system can be improved by changing the user interface by an android application that uses internet of things.

Bibliography

- [1] DEREK MOLLOY: Exploring BeagleBone Tools and Techniques for building with Embedded Linux.
- [2] AN-6076 Design and Application Guide of Bootstrap Circuit for High-Voltage Gate-Drive IC
- [3] <http://beagleboard.org/>
- [4] <http://processors.wiki.ti.com>
- [5] <https://www.alldatasheet.com/datasheet-pdf/pdf>
- [6] <http://www.ecircuitcenter.com/Circuits/pid1/pid1.htm#top>
- [7] https://e2e.ti.com/blogs_/b/motordrivecontrol/archive/2012/03/19/so-which-pwm-technique-is-best-part-1#pi318947=89
- [8] <https://www.embeddedrelated.com/showarticle/586.php>

Annex

