Université Mohamed Khider de Biskra Faculté
des Sciences et de la Technologie Département
de Génie électrique

MÉMOIRE DE MASTER

Filière : Électrotechnique

Option : Commande Électrique

Réf :

Présenté et soutenu par :

MANSOURI Zakaria

Le : 06 juillet 2019

Realization of PWM control interface for a single-phase full-bridge inverter controlled by Arduino board

---

Jury :

Mr.	GOLEA AMMAR	**Prof**	Université de Biskra	Président
Mr.	DENDOUGA ABDELHAKIM	**MCA**	Université de Biskra	Encadreur
Mr.	MOHAMMEDI MESSAOUD	**MCB**	Université de Biskra	Examinateur

Année universitaire : 2018-2019

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique

Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Électrique
Filière : Électrotechnique
Option : Commande Électrique

Mémoire de Fin d'Études
En vue de l'obtention du diplôme :

MASTER

Thème

Realization of PWM control interface for a single-phase full-bridge inverter controlled by Arduino board

Présenté par :
MANSOURI Zakaria

Avis favorable de l'encadreur :
Mr DENDOUGA Abdelhakim

Avis favorable du Président du Jury

GOLEA Ammar .

Cachet et signature

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique

Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Électrique
Filière : Électrotechnique
Option : Commande Électrique

Thème :

Réalisation d'une interface de commande MLI pour un onduleur monophasé en pont contrôler par Arduino

Proposé par : MANSOURI Zakaria
Dirigé par : DENDOUGA Abdelhakim

Abstract

The current project has as major aim the design of a single-phase inverter for educational purposes. The main distinctive feature is the digital implementation of the PWM modulation with a graphical user interface.

Since it's lunch in 2005, Arduino it is a highly recommended option for the first approach to digital programming for students. In this work we aim to use the Arduino to make it easy to generating PWM control signal for single-phase inverter.

There are already circuits available for generating PMW signals, we went further in customizing the way PWM is generated, and made a desktop application that provides a complete control of the generated PWM signal.

At first, complete theoretical analysis will be made, including its applications and basic elements. Afterwards, we will be defined the specific characteristics of the desired inverter, and select components required.

An essential part of the work relies on the programming desktop interface. For this reason, an insight into the technologies available will be made. Some options for will be discussed and the most suitable technology will be selected.

Finally, the circuit will be implemented in a protoboard. And measurement of the control signals will be done, and compared with the theoretically calculated signals.

ملخص

يهدف المشروع الحالي إلى تصميم عاكس كهربائي أحادي الطور للأغراض التعليمية. النقطة المميزة الرئيسية هي إمكانية إصدار إشارة (نمذجة عرض الموجة) PWM بواجهة مستخدم.

منذ إصداره عام 2005، يعد الـ Arduino خيارًا موصى به للغاية في البرمجة الرقمية للطلاب. في هذه المذكرة، نهدف إلى استخدام الـ Arduino لتسهيل توليد إشارة تحكم PWM لعاكس أحادي الطور.

على الرغم من وجود دارات لتوليد إشاراتPMW ، في هذا العمل تعمقنا في تخصيص طريقة إنشاءPWM ، وقمنا بإنشاء تطبيق سطح مكتب يوفر تحكمًا كاملاً في إشارة PWM التي تم إنشاؤها.

في البداية، سيتم إجراء تحليل نظري كامل، بما في ذلك تطبيقاته وعناصره الأساسية. بعد ذلك، سيتم تعريف الخصائص المحددة للعاكس المطلوب، وتحديد المكونات المطلوبة.

جزء أساسي من العمل يعتمد على واجهة برمجة سطح المكتب. لهذا السبب، سيتم تقديم نظرة ثاقبة للتقنيات المتاحة. سيتم مناقشة بعض الخيارات وسيتم اختيار التقنية الأكثر ملاءمة.

بعد الانتهاء من الجزء النظري، سيتم تنفيذ الدارة في اللوحة الأولية. وسيتم قياس إشارات التحكم، ومقارنة مع الإشارات المحسوبة نظريا.

Thanks

Above all, we thank God Almighty for giving us courage, and health during all these years and that thanks to him this work could be realized.

Our thanks also go to the **Dr. DENDOUGA Abdelhakim**, for helping me writing this memoir as well as monitoring the process of this work and preaching us.

I also thank the members of the jury who increases the honor of residing my work

Dedications

Dedicate this work to the dearest people to my heart my mother and my father who always helped me to consecrate.

To my brothers and sisters.

To my friends, my casemates.

To all my teachers.

List of tables

Chapter I

Chapter II

List of figures

Chapter I

Chapter II

Chapter III

List of abbreviations

PWM: Pulse width modulation.

AC: Alternative current.

DC: Direct current.

IGBT: Insulated-gate bipolar transistor.

MOSFET: Metal–oxide–semiconductor field-effect transistor.

SPI: Serial Peripheral Interface.

USB: Universal Serial Bus.

Abstract

The current project has as major aim the design of a single-phase inverter for educational purposes. The main distinctive feature is the digital implementation of the PWM modulation with a graphical user interface.

Since it's lunch in 2005, Arduino it is a highly recommended option for the first approach to digital programming for students. In this work we aim to use the Arduino to make it easy to generating PWM control signal for single-phase inverter.

There are already circuits available for generating PMW signals, we went further in customizing the way PWM is generated, and made a desktop application that provides a complete control of the generated PWM signal.

At first, complete theoretical analysis will be made, including its applications and basic elements. Afterwards, we will be defined the specific characteristics of the desired inverter, and select components required.

An essential part of the work relies on the programming desktop interface. For this reason, an insight into the technologies available will be made. Some options for will be discussed and the most suitable technology will be selected.

Finally, the circuit will be implemented in a protoboard. And measurement of the control signals will be done, and compared with the theoretically calculated signals.

Key words: single-phase inverter, PWM, Arduino, desktop user interface.

Table of Contents

General introduction

The main objective of this work is the design and testing a full-bridge single-phase inverter on the one hand, and on the other hand the practical implementation of the MLI control using an Arduino board. In this context, a graphical interface has been created to control the Arduino.

Since the development of the first Arduino board in 2005, digital control in power electronics has grown considerably. For those reasons, this work has the aim to apply this tool to ease the PWM implementation on a single-phase inverter, substituting analogical circuitry.

Although nowadays it is possible to find integrated circuits with single – phase inverters, the circuit will be designed and built piece by piece. Thus, it is easier to understand and visualize all the components required, as well as to check the waveforms obtained on the internal elements.

In this context, a first theoretical analysis will be carried out, including its applications and its basic elements. Then, the specific characteristics of the desired inverter will be defined, thus allowing the calculation and selection of the required components.

A fundamental part of the work is the development of a control interface for the Arduino to generate the PWM signal for the inverter. For this reason, a general overview on the PWM control and its characteristics, as well as on the programming side of the Arduino was presented in detail.

After the theoretical approach, the complete circuit will be implemented in a protoboard. Some measurements and tests will be also done in order to check the performance of the device and its efficiency.

Chapter I: Single phase inverter

I.1 Introduction

This chapter presents a general overview of the inverter concerning its operating principle, its different topologies, as well as the various control strategies, etc. In addition, the PWM control strategy has been studied in detail.

I.2 Definition

A power inverter, or inverter, is an electronic device or circuitry that changes direct current (DC) to alternating current (AC) [1]. Inverters, are constructed from power switches and the AC output waveforms are therefore made up of discrete values. This leads to the generation of waveforms that feature fast transitions rather than smooth ones. [2]

Depending on the number of phases of the AC output, there are several types of device the famous ones are: single-phase and three-phase inverters.

Figure I- 1 Ring Automotive Voltage Transformer Inverter Power Source Pro 2100W

I.3 Applications

Inverters are used to convert DC electricity from sources like solar panels, batteries, or fuel cells to AC electricity.

Solar inverters convert DC power from solar panels to AC for the electric grid. Grid-tied inverters are designed to feed into the electric power distribution system [3]. They transfer synchronously with the line and have as little harmonic content as possible. Uninterrupted power supplies use inverters to power programmable logic controllers (PLC) and to supply AC power when main power is not available. Given that direct connection is not possible, an inverter is required as an intermediate device.

There is a growing demand for power in a wide range of applications: portable consumer devices, hybrid/electric vehicles, industrial control systems, and solar power systems. This has led to increasing adoption of inverters to ensure a better supply of high efficiency, high reliability power.

Inverters provide increased energy savings over conventional on/off control. Faster switching times mean more precise temperature control. In fan motors, inverters reduce audible noise.

The most widespread converters are the voltage inverters, also called power-reducers, as the output voltage is lower than the input.

Figure I- 2 Examples of applications of power inverters

I.4 Types of inverters by power source nature

There are serval types of inverters that can be grouped together by number of considerations. In this section, we classify them by some of those considerations: the power source nature.

I.4.1.1 Voltage inverter

A voltage inverter is an inverter that is powered by a DC voltage source.

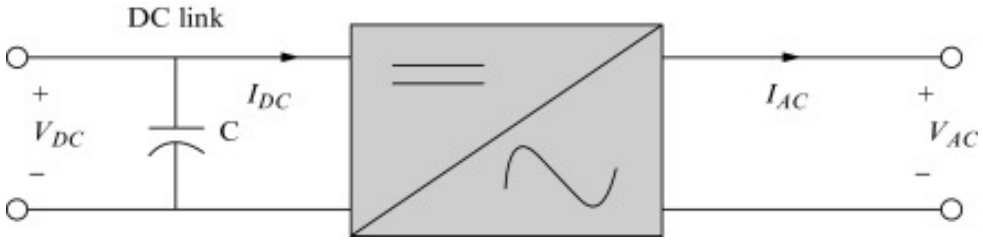

Figure I- 3 Voltage source inverter. [4]

Single-phase voltage source inverters (VSIs) can be found as half-bridge and full-bridge topologies. Although the power range they cover is the low one, they are widely used in power supplies, single-phase UPSs, and currently to form elaborate high-power static power topologies [5]

I.4.1.2 Current-source inverter

In single-phase current-source inverter, the DC source is provided by a bridge rectifier and is connected by a DC link inductor in series, and the RL load is connected by a capacitor in parallel. The purpose of the capacitor is to make the current of the effective load lead that of the voltage so that the load commutation of the thyristors can be realized. [6]

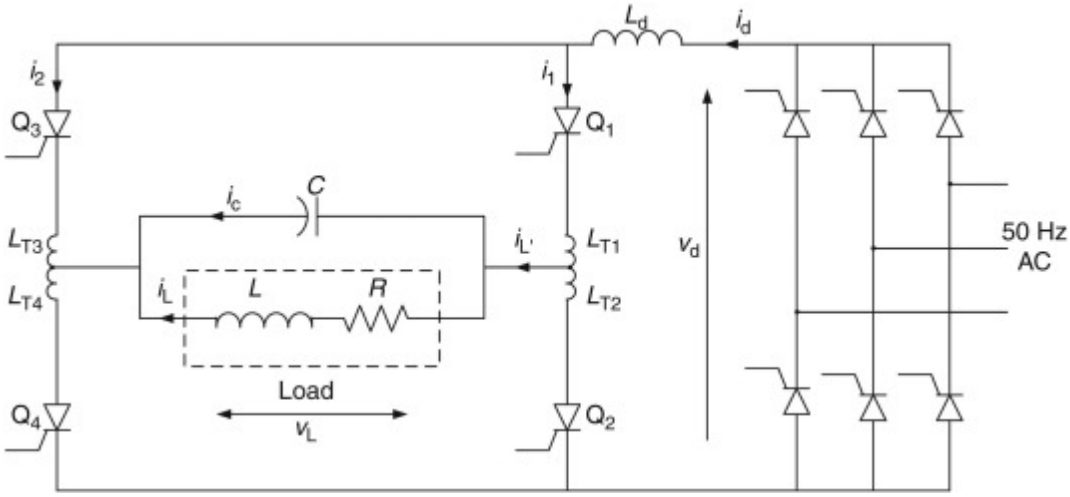

Figure I- 4 Single-phase current-source inverter

The circuit is generally utilized for high-frequency induction heating applications. It is assumed that the DC link inductance is sufficiently large to smooth the DC current-source ripples and the capacitor has near perfect filtering of harmonic currents. [2]

I.5 Output

With regard to their output, three different types of power inverters can be found: square wave, modified sine wave and pure sine.

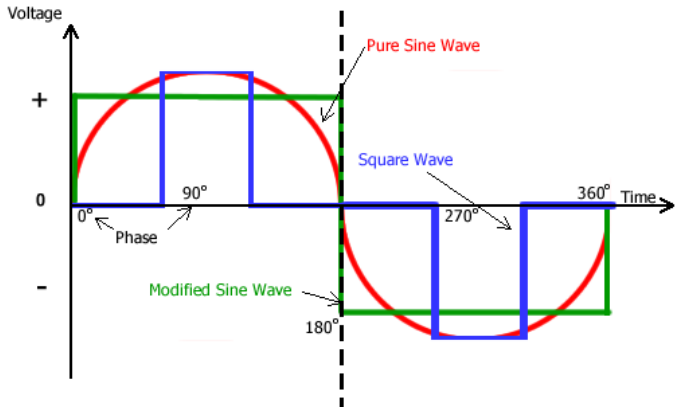

Figure I- 5 square wave, modified sine wave and pure sine wave

I.5.1 Square wave

This is the basic type of inverter. Its output is an alternating square wave. The harmonic content in this wave is very large. This inverter is not efficient and can give serious damage to some of the electronic equipment. But due to low cost, it has some limited number of applications in household appliances.

I.5.2 Modified sine wave

A modified sine wave inverter actually has a waveform more like a square wave, but with an extra step or so. Because the modified sine wave is noisier and rougher than a pure sine wave, clocks and timers may run faster or not work at all. A modified sine wave inverter will work fine with most equipment, although the efficiency or power will be reduced with some. But with most of the household appliances it works well.

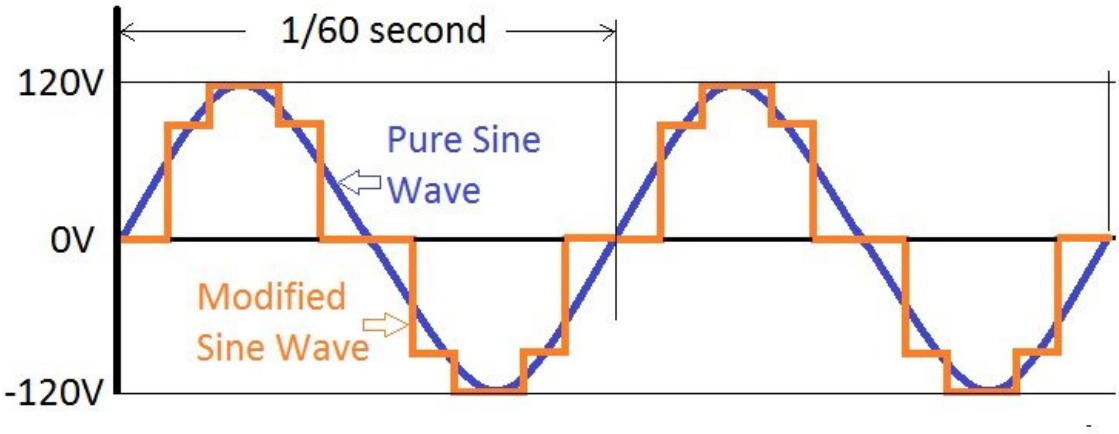

Figure I- 6 Modified sine wave inverter output

I.5.3 Pure sine wave

This type of inverter provides output voltage waveform which is very similar to the voltage waveform that is received from the Grid. The sine wave has very little harmonic distortion, what makes it ideal for running electronic systems such as computers and other sensitive equipment without causing problems or noise.

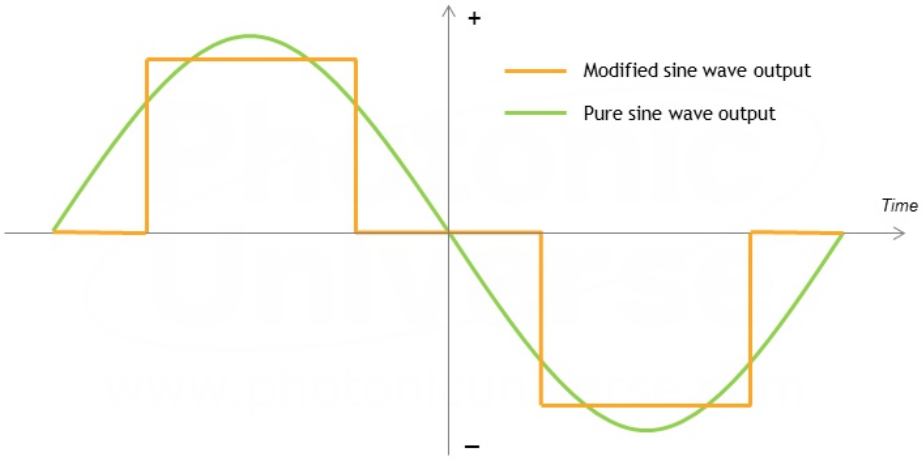

Figure I- 7 Pure sine wave inverter output

Benefits of using pure sine wave Inverter:

- Most of the electrical and electronic equipments are designed for the sine wave.
- Electronic clocks are designed for the sine wave.
- Lesser harmonic distortion.

I.6 Inverter topologies: Half-Bridge and Full-Bridge

There are several topologies available, in this section we will briefly explain the two most common ones: Half Bridge and Full Bridge.

I.6.1 Half-Bridge

Only two switchers are required in this topology. The DC input is divided in two identical sources and the output is referenced to the middle point.

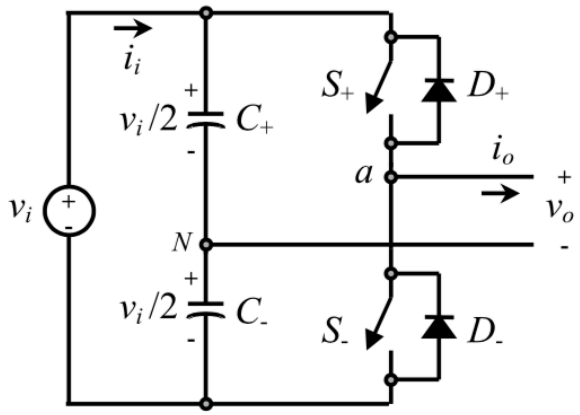

Figure I- 8 Single-phase Half-Bridge voltage inverter

A capacitor divisor is used to achieve the medium voltage point (N). By controlling the voltage in N, direct current injected in the alternate side is assured to be zero.

In order to obtain the same value of power, higher currents are required, as voltage is lower. In case a high voltage is needed in the output, an elevator is commonly used as first step, as the input voltage must be double than the output desired. Regarding switching losses, the semiconductors must be designed for $2V_o$. This fact makes this topology the worst in performance, as switching losses become excessively high. [2]

I.6.2 Full H-Bridge

In a Full H-Bridge, the alternate output voltage (V_o) is obtained by the difference between two branches of switching cells. Therefore, four switchers are needed. To maximize the fundamental component of the output voltage, the fundamental component of the voltage on each branch (V_{ao} and V_{bo}) must be 180º out of phase. The semiconductors of each branch are complementary in performance, which is to say when one is conducting the other is cut-off and vice versa. This topology is the most widely used for inverters.

The semiconductors must be designed only for V_o, but as a disadvantage, four switchers are required and therefore, losses can become elevate. [2]

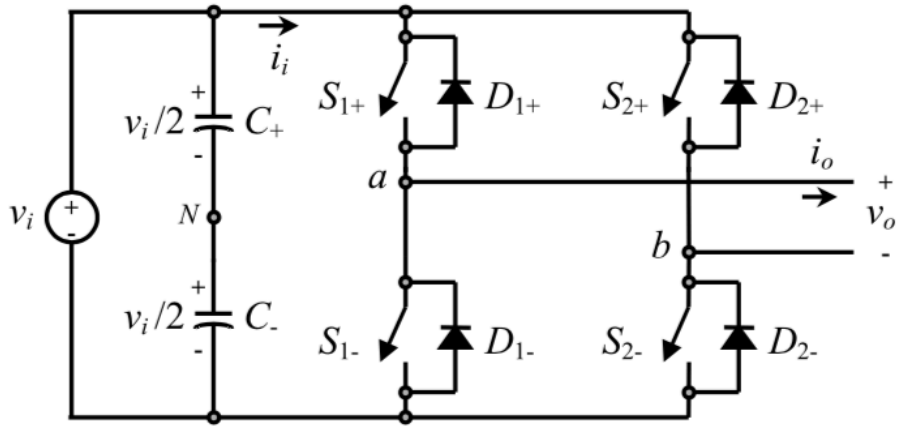

Figure I- 9 Single-phase Full H-Bridge voltage inverter

I.7 Controlling the single-phase inverter

The idea is to compare a modulation signal with a signal of the type "triangle" or "sawtooth". The result of this comparison creates a control signal for the triggers of the switches (MOSFET) in a complementary manner.

For a single-phase voltage inverter, two types of modulation are mainly distinguished:

- The fixed cyclic ratio (plain wave, shifted) modulations, or the duty cycle of each of the switching cells is kept constant.
- Pulse width modulations (PWM), or duty cycle, are sinusoidally variable.

I.7.1 Fixed duty cycle control

The control signal is generated by comparing a constant signal (**Vm**: modulation signal) with triangular one (**V**c carrier signal)

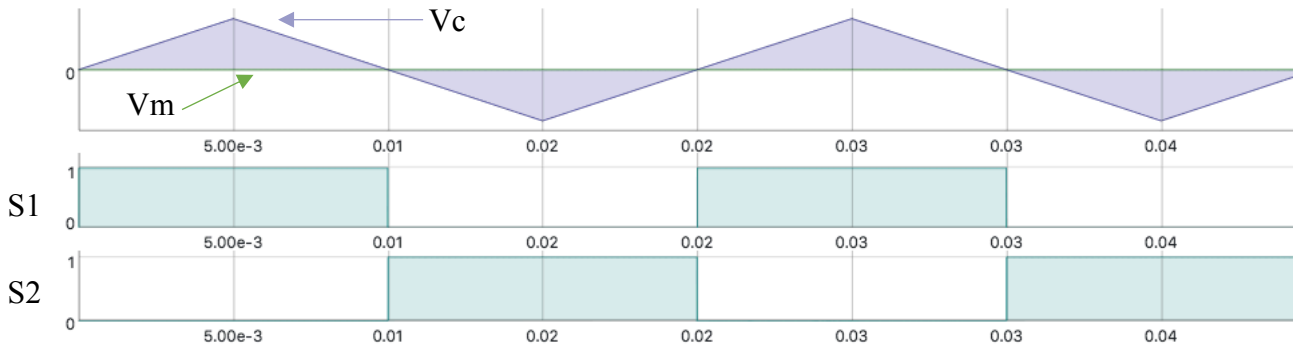

Figure I- 10 fixed duty cycle control signal

This will produce a control signal that can be modeled this algorithm:

- When **Vc** > **Vm**: S1 switch is on, and S2 is off
- When **Vc** < **Vm**: S1 switch is off, and S2 is on

I.7.2 Shifted duty cycle control

The control signal is generated the same way, by comparing a constant modulation signal (**Vm**) with triangular one (**Vc**), but this time, there a dead time where the output signal settle at the null value for a bit of time

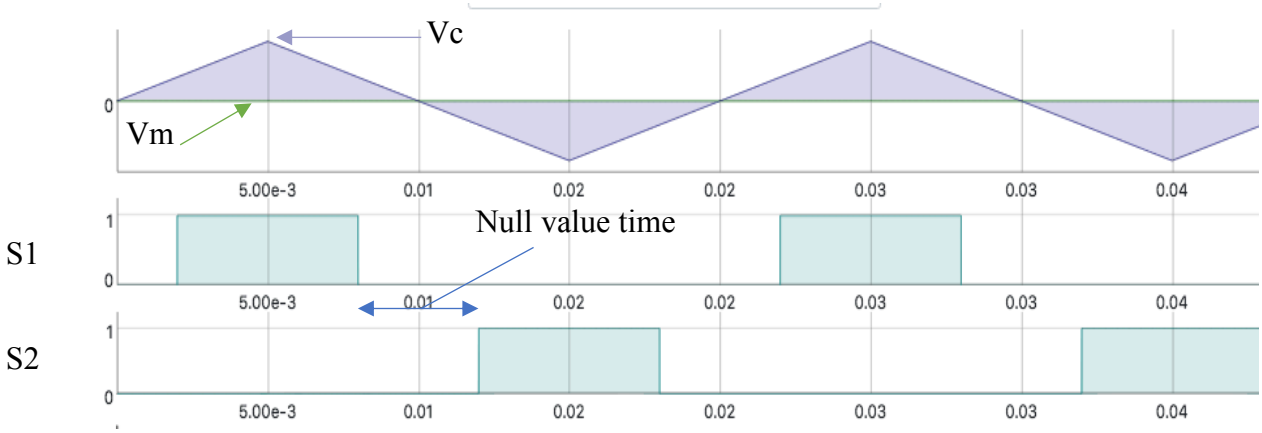

Figure I- 11 Shifted duty cycle control

I.7.3 Variable duty cycle control (SPWM)

One of the methods used to reduce the low frequency harmonics in the inverter waveform is sinusoidal pulse-width modulation. In this method, a reference copy of the desired sinusoidal waveform, the modulating wave, is compared to a much higher frequency triangular waveform, called the carrier wave. The resulting drive signals cause multiple turn-on of the inverter switches in each half-cycle with variable pulse width to produce a quasi-sine wave of load voltage. The pulse width increases from a very narrow width at the start of each cycle to a maximum width in the middle of each cycle. Then the pulse width reduces again after maximum until its minimum width at the end of the half-cycle period. [7]

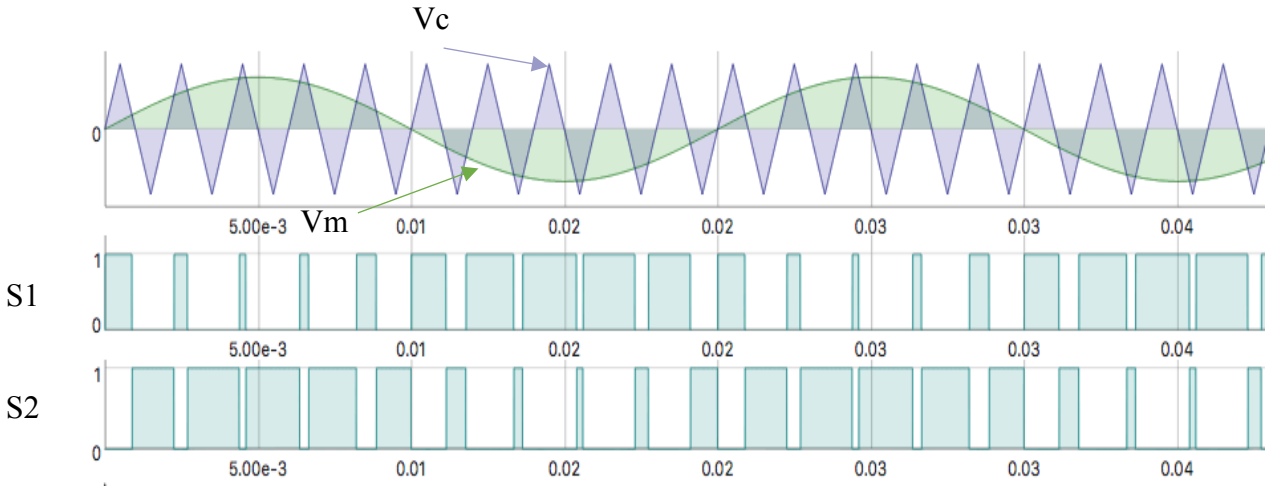

Figure I- 12 the sinusoidal pulse-width modulation method

SPWM is a special case when the modulating signal $\mathbf{V_m}$ is a sinusoidal at frequency $\boldsymbol{f_m}$ and amplitude $\widehat{\boldsymbol{V}}_\mathbf{m}$, and the triangular signal $\mathbf{V_c}$ is at frequency $\boldsymbol{f_c}$ and amplitude $\widehat{\boldsymbol{V}}_\mathbf{c}$. In this case, the modulation index $\mathbf{m_a}$ (also known as the amplitude-modulation ratio) is defined as [2]

$$\boldsymbol{m_a} = \frac{\widehat{\boldsymbol{V}}_{\boldsymbol{m}}}{\widehat{\boldsymbol{V}}_{\boldsymbol{c}}}$$

and the normalized carrier frequency $\mathbf{m}_f$ (also known as the frequency-modulation ratio) is

$$\boldsymbol{m}_\mathrm{f} = \frac{\mathrm{f}_{\boldsymbol{c}}}{\mathrm{f}_{\boldsymbol{m}}}$$

I.8 Driver

A driver is a device which adapts the connection function to the requirements of the semiconductors. As the connection function only oscillates between two values, sometimes it is necessary to modify the signal in order not to damage the semiconductors.

Main functions:

- Amplification of the control signal in order to adapt it to the desired levels of voltage and current.
- Galvanic isolation: to avoid electrical contact between two parts of the circuit.
- Protection against low feeding voltages or current that could damage the semiconductors.

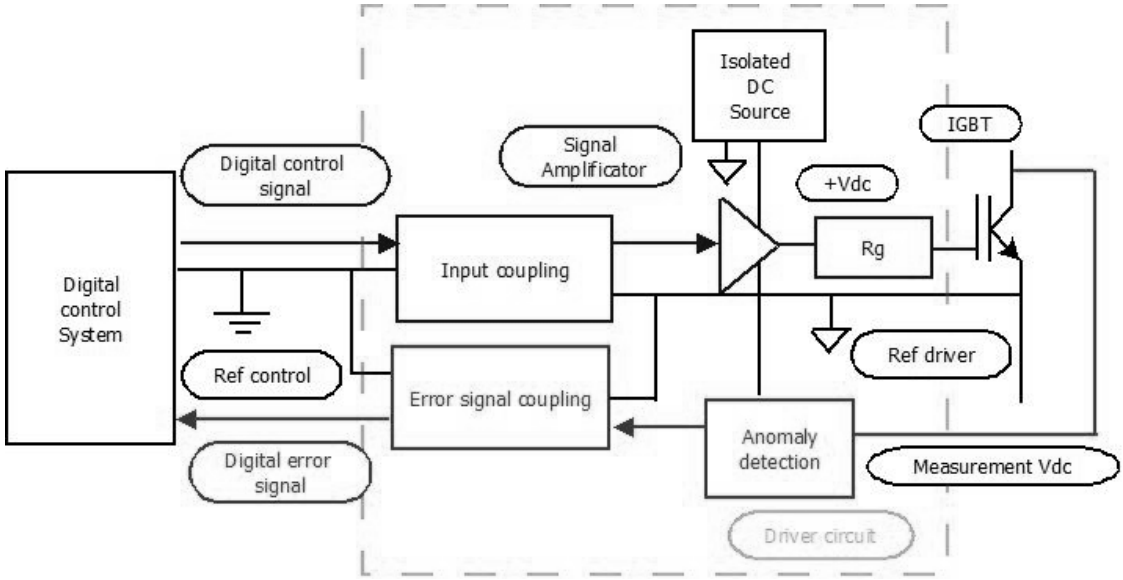

Figure I- 13 Functional schematic of a driver

I.9 Filter

Because the output voltage waveform is squared, a filter is required in order to obtain a sine wave form, as well as helping in reducing harmonics, which may disturb the correct operation of the output load connected.

A filter is, in its most basic sense, a device that enhances and/or rejects certain components of a signal [8]. It can be analogic or digital and depending on its behavior, there are mainly four types.

Type of filter	Features
Low pass	They let pass low frequencies and attenuate the ones over a cut point
High pass	They attenuate frequencies below a cut point
Band pass	They let pass frequencies between a range
Band rejection	They block frequencies between a range

Table I- 1 Types of filters and main features.

Once the type of circuit is selected, the cut point and the ranges of frequencies can be chosen by modifying the components.

As in this case the objective is to attenuate harmonics, a low pass filter will be selected. For a low pass filter, there are two options regarding circuitry: RC (resistor - capacitance) or LC (inductance - capacitance).

I.10 Conclusion

In this chapter, we explained the general operating principle of the inverter and the applications that used it, as well as the different control techniques of the inverter. In the next chapter we will try to explain the operating principle of the Arduino board which will be used as a control platform of our inverter. Also, we will talk about the desktop interface that let us change the control techniques.

Chapter II: Arduino and the desktop interface

II.1 Introduction

In this chapter, we will get an overview of what the Arduino is, and see its diagram, inputs and outputs, and how to power up the board. Then, we will get more specific and talk about Arduino Uno board mentioning its characteristics and operating principle. We will take a look at how programming the Arduino is done, then move on to the desktop interface programming, and explore the technologies that will let us develop the desired application.

II.1.1 What is Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. [9]

II.2 Arduino Uno

II.2.1 Overview

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again. [10]

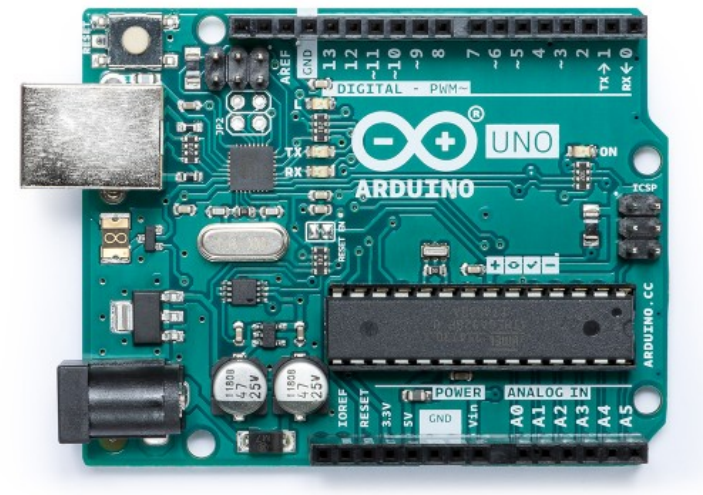

Figure II- 1 Arduino Uno Rev3 board [10]

II.2.2 The ATmega328P

The Arduino Uno is based on the ATmega328P, which is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328P achieves throughputs approaching 1MIPS per MHz allowing the system designer to optimize power consumption versus processing speed. [11]

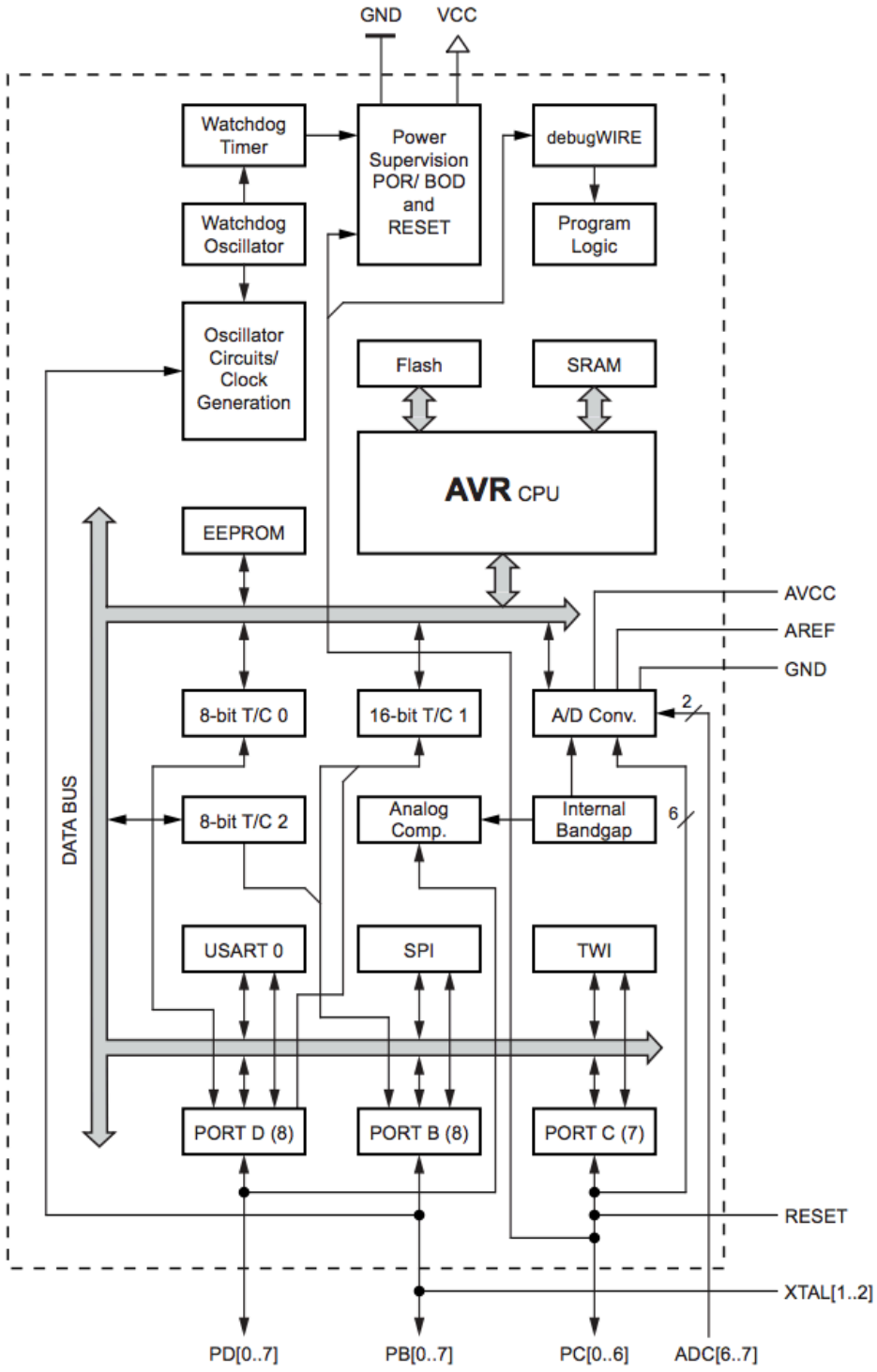

Figure II- 2 The Atmel® ATmega328P Block Diagram [11]

II.2.3 Features

The table below summarizes the features of the Arduino Uno:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Table II- 1 Arduino Uno Tech Specs [10]

II.2.4 Powering the Arduino Uno [10]

The Arduino Uno board can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

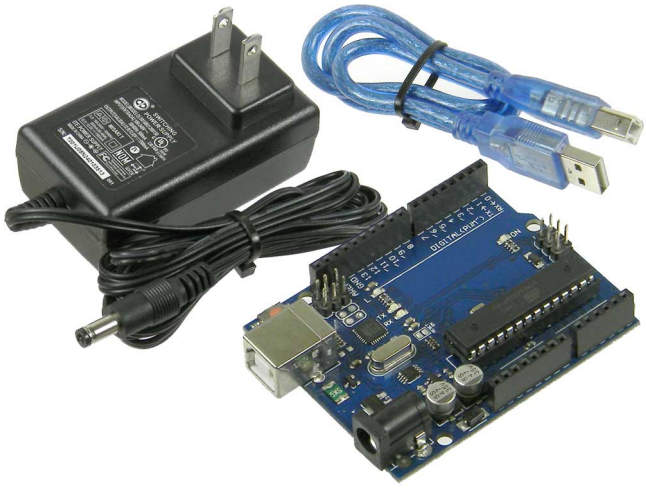

Figure II- 3 Arduino Uno power sources

The power pins are as follows:

- **Vin**. The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- **5V**. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3**. A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND**. Ground pins.
- **IOREF**. This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.

II.2.5 Inputs/outputs

Each of the 14 digital pins on the Uno can be used as an input or output, using *pinMode()*, *digitalWrite()*, and *digitalRead()* functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller. [10]

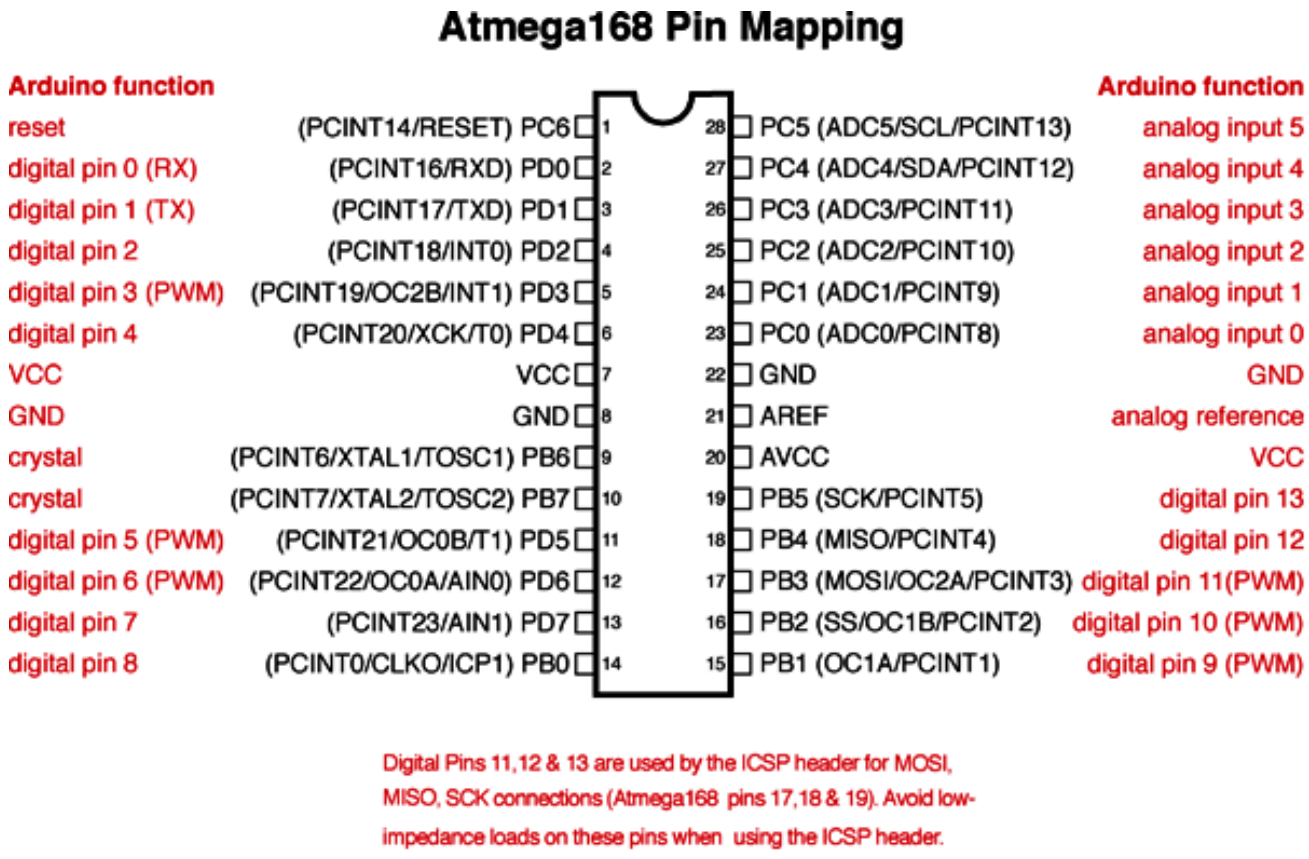

Figure II- 4 ATmega168/328P-Arduino Pin Mapping [12]

In addition, some pins have specialized functions: [10]

- **Serial**: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts**: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value using the *attachInterrupt()* function.
- **PWM**: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the *analogWrite()* function.
- **SPI**: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- **LED**: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI**: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

II.3 Programming the Arduino

The Arduino Uno can be programmed with the (Arduino Software (IDE)). Select "Arduino/Genuino Uno from the Tools > Board menu (according to the microcontroller on your board).

The ATmega328 on the Arduino Uno comes preprogrammed with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol. [10]

II.3.1 The Arduino IDE

Arduino IDE is a software that is mainly used for writing and compiling the code into the Arduino Module. It's a cross-platform application (for Windows, macOS, Linux) that is written using Java.

The IDE environment is mainly distributed into three sections:

1- Menu Bar.
2- Text Editor.
3- Output Pane.

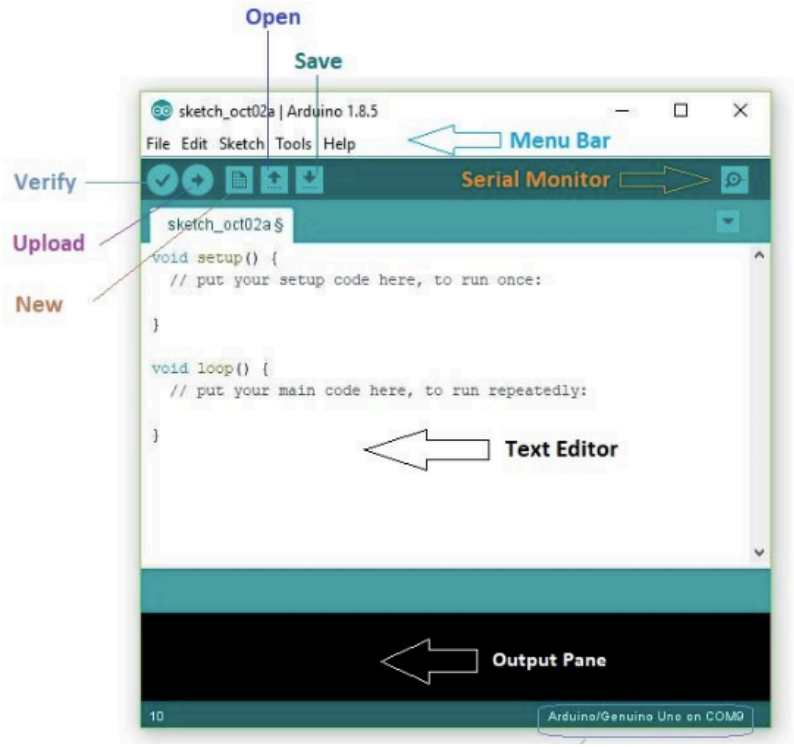

Figure II- 5 The Arduino IDE interface

II.3.1.1 Menu Bar

The bar appearing on the top is called Menu Bar that comes with five different options as follow

- **File** - You can open a new window for writing the code or open an existing one.
- **Edit** - Used for copying and pasting the code with further modification for font
- **Sketch** - For compiling and programming
- **Tools** - Mainly used for testing projects. The Programmer section in this panel is used for burning a bootloader to the new microcontroller.
- **Help** - In case you are feeling skeptical about software, complete help is available from getting started to troubleshooting.

The Six Buttons appearing under the Menu tab are connected with the running program as follow:

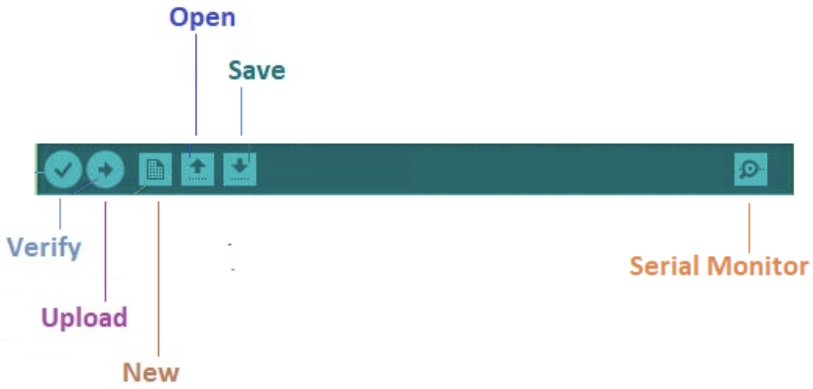

Figure II- 6 The Buttons under the Menu tab in Arduino IDE

- The check mark appearing in the circular button is used to verify the code. Click this once you have written your code.
- The arrow key will upload and transfer the required code to the Arduino board.
- The dotted paper is used for creating a new file.
- The upward arrow is reserved for opening an existing Arduino project.
- The downward arrow is used to save the current running code.
- The button appearing on the top right corner is a Serial Monitor

II.3.1.2 Text Editor

The main screen below the Menu bard is known as a simple text editor used for writing the required code.

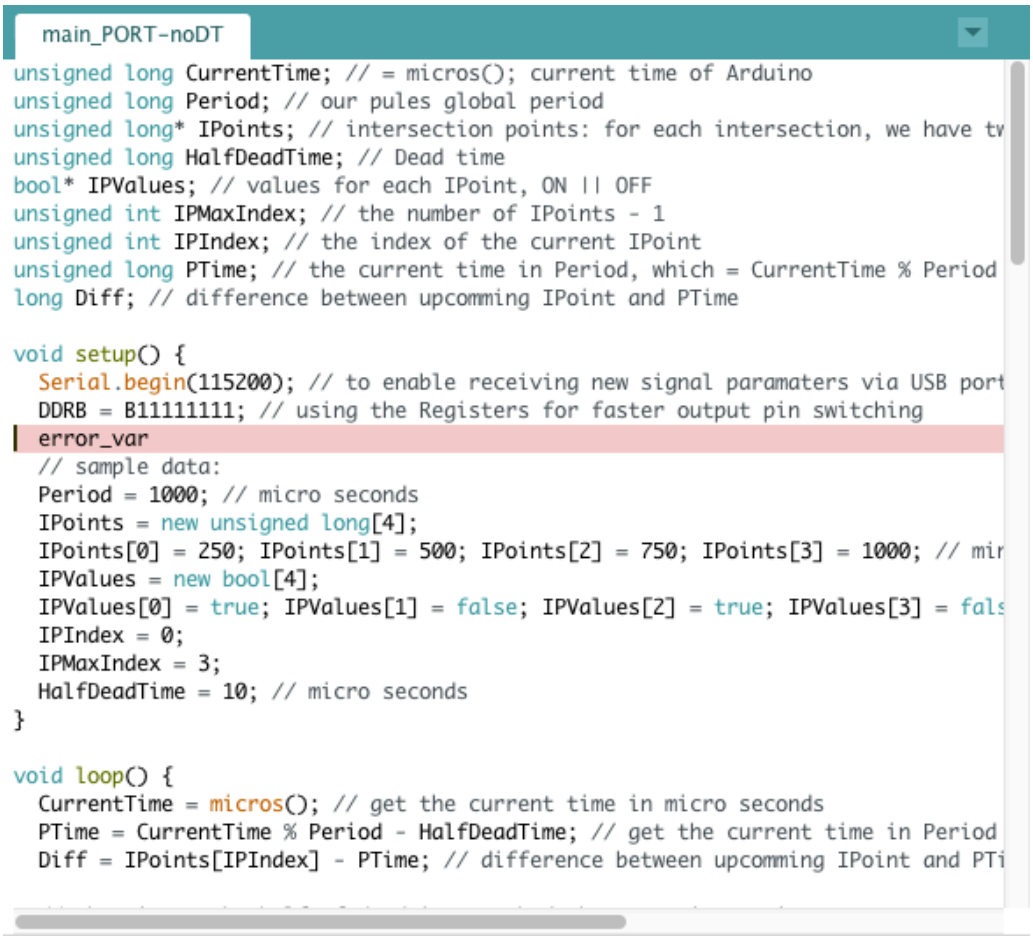

Figure II- 7 The text editor in Arduino IDE

II.3.1.3 Output Pane

The bottom of the main screen is described as an Output Pane that mainly highlights the compilation status of the running code: the memory used by the code, and errors occurred in the program. You need to fix those errors before you intend to upload the hex file into your Arduino Module.

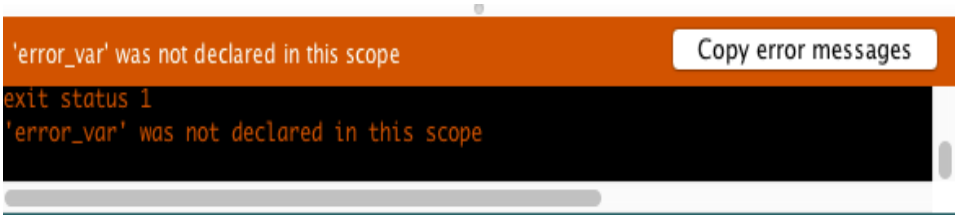

Figure II- 8 The output pane in Arduino IDE

II.3.2 Basics of Arduino language [13]

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

II.3.2.1 Functions

For controlling the Arduino board and performing computations.

Digital I/O
digitalRead()
digitalWrite()
pinMode()

Analog I/O
analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family
analogReadResolution()
analogWriteResolution()

Advanced I/O
noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time
delay()
delayMicroseconds()
micros()
millis()

Math
abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry
cos()
sin()
tan()

Characters
isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Random Numbers
random()
randomSeed()

Bits and Bytes
bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

External Interrupts
attachInterrupt()
detachInterrupt()

Interrupts
interrupts()
noInterrupts()

Communication
Serial
Stream

USB
Keyboard
Mouse

II.3.2.2 Variables

Arduino data types and constants. They are grouped in 5 main groups:

Constants	**Data Types**	**Variable Scope & Qualifiers**
Floating Point Constants	String()	const
Integer Constants	array	scope
HIGH | LOW	bool	static
INPUT | OUTPUT | INPUT_PULLUP	boolean	volatile
LED_BUILTIN	byte	
true | false	char	**Utilities**
	double	PROGMEM
Conversion	float	sizeof()
(unsigned int)	int	
(unsigned long)	long	
byte()	short	
char()	size_t	
float()	string	
int()	unsigned char	
long()	unsigned int	
word()	unsigned long	
	void	
	word	

II.3.2.3 Structure

The elements of Arduino (C++) code. They are grouped in 9 main groups:

Sketch	**Arithmetic Operators**	**Pointer Access Operators**
loop()	% (remainder)	& (reference operator)
setup()	* (multiplication)	* (dereference operator)
	+ (addition)	
Control Structure	- (subtraction)	**Bitwise Operators**
break	/ (division)	& (bitwise and)
continue	= (assignment operator)	<< (bitshift left)
do...while		>> (bitshift right)
else	**Comparison Operators**	^ (bitwise xor)

for	!= (not equal to)	| (bitwise or)
goto	< (less than)	~ (bitwise not)
if	<= (less than or equal to)	
return	== (equal to)	**Compound Operators**
switch...case	> (greater than)	%= (compound remainder)
while	>= (greater than or equal to)	&= (compound bitwise and)
		*= (compound multiplication)
Further Syntax	**Boolean Operators**	++ (increment)
#define (define)	! (logical not)	+= (compound addition)
#include (include)	&& (logical and)	-- (decrement)
/* */ (block comment)	|| (logical or)	-= (compound subtraction)
// (single line comment)		/= (compound division)
; (semicolon)		^= (compound bitwise xor)
{} (curly braces)		|= (compound bitwise or)

II.4 The Desktop Interface

II.4.1 Connecting to the Arduino board (Serial port)

For our application to connect to the Arduino board, there must be a way to send data back and forth between both ends, for this we will use the USB on the computer with the serial pin on the Arduino Uno 0(RX), 1(TX), and establish a serial port communication.

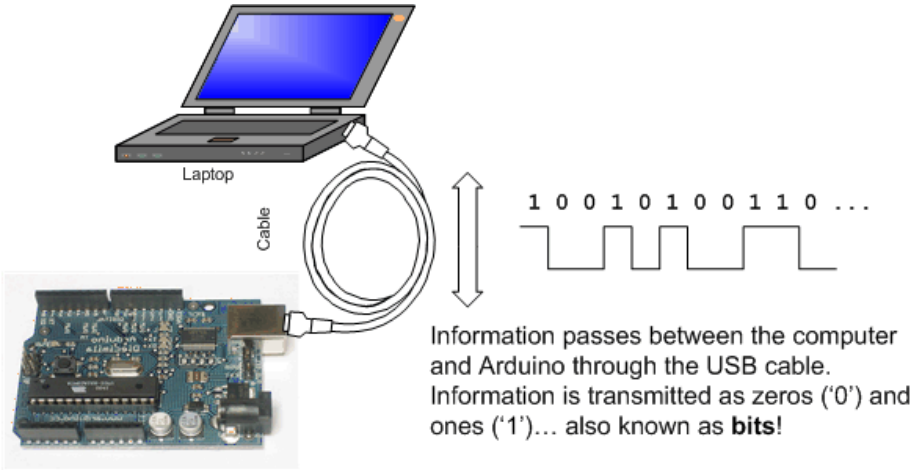

Figure II- 9 Serial communication between computer and Arduino

A serial port is a serial communication interface through which information transfers in or out one bit at a time (in contrast to a parallel port). [14] Throughout most of the history of personal computers, data was transferred through serial ports to devices such as modems, terminals, and various peripherals

II.4.2 Available technologies for application development

Currently there are many different technologies that can be used to develop a desktop application, each has its own advantage and limits, here are some of the most common ones:

- **Java SE**: Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require. [15]
- **.NET Framework**: A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services. [16]
- **Electron JS**: Electron is an open source library developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript.

II.4.3 Electron JS

Electron JS is chosen here, mainly because of the ease of use and the fast development time, thanks to the web technologies (HTML, CSS, and JavaScript), Electron accomplishes this by combining Chromium and Node.js into a single runtime and apps can be packaged for Mac, Windows, and Linux. [17]

II.4.4 JavaScript

JavaScript (JS) is a high-level, interpreted programming language that conforms to the ECMAScript specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. [18]

```
function factorial(n) {
    if (n === 0) {
        return 1;  // 0! = 1
    }
    return n * factorial(n - 1);
}

factorial(3); // returns 6
```

Figure II- 10 A simple recursive JavaScript function

II.4.4.1 Nodejs

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications. [19]

II.4.4.2 NPM

NPM is a package manager for the JavaScript programming language. And it is the default package manager for the JavaScript runtime environment Node.js.

NPM is the world's largest software registry. Open source developers from every continent use NPM to share and borrow packages, and many organizations use NPM to manage private development as well.
For developing the application, we use serval NPM packages, including:

- **ReactJS**: for designing the graphical user interface
- **Dygraphs**: for visualizing the graphs.
- **Node Serialport**: for managing communication with Arduino via serial port.

II.5 Conclusion

In this chapter we presented the main diagram of the Arduino, then talked specifically about Arduino Uno, we also gave the tools of development of a program especially for the Arduino C. Then discussed the technologies available for developing the desktop application, and got an overview of the chosen technology: Electron JS.

Chapter III: Realization and Results

III.1 Introduction

This chapter is devoted to the experimental validation of the studies and simulation presented in the first and second chapters. We will present the results of the control application and the main experimental results to confirm the validation of the control algorithms.

An oscilloscope is employed for the display and measurement of the results. Furthermore, some pictures of the circuits built can also be found in these pages.

III.2 Diagram of the project

This diagram below shows the flow of information from the Desktop interface, to the Arduino, that connect to the control circuit where the control signal gets amplified, then sent to the gates of the inverter switches, which in turn operates according the control signal sent by the Arduino.

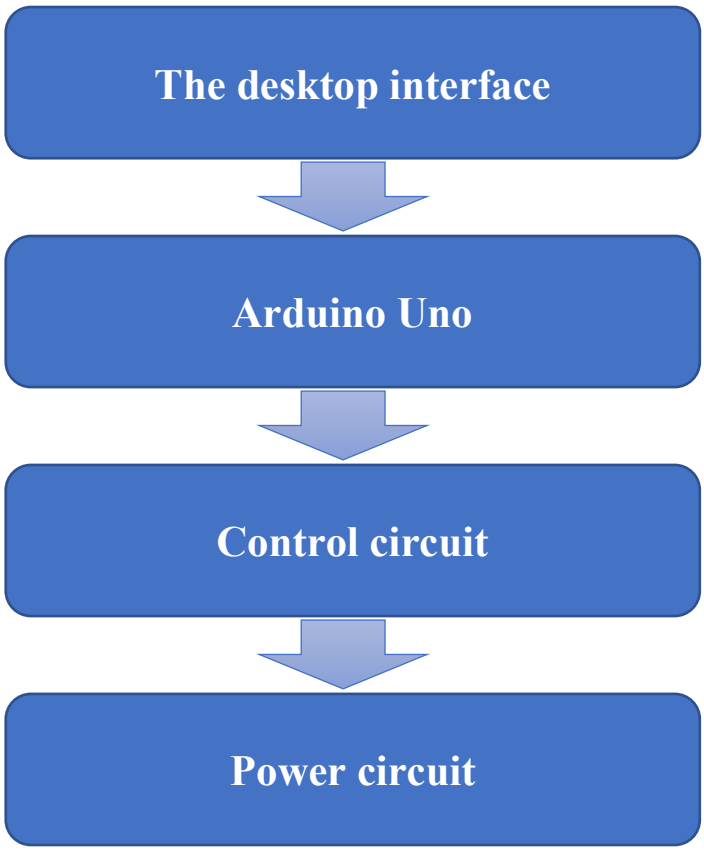

Figure III- 1 Diagram of the project

III.3 Desktop interface

The first step is to define the control signal, and to do that, we have to setup the PWM parameters using the desktop interface.

Figure III- 2 The desktop interface

After entering the parameters of both the modulation and the carrier signals, as well as general parameters like sampling time. The app will automatically generate a preview of the comparison, and the output results for both direct and inverse pulses signals.

The PWM desktop interface is divided into 4 main components: send to Arduino, signals parameters, general parameters, comparison results.

III.3.1 Send to Arduino

After entering all the necessary parameters, we need to send the control signal parameters to the Arduino, and that can be done by clicking the “Send to Arduino” button.

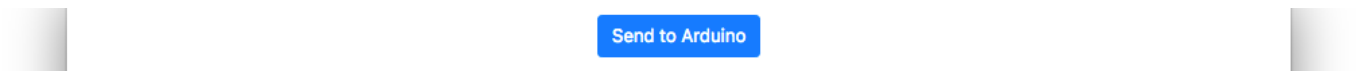

Figure III- 3 The desktop interface: Send to Arduino Button

What happens is that, the application sends the necessary information to the Arduino board, and this information is basically the intersection points resulted from the comparison between both the modulation and carrier signals, in one period.

III.3.2 Signals parameters

The first step in setting up the control signal is to define the parameters for both modulation and carrier signals.

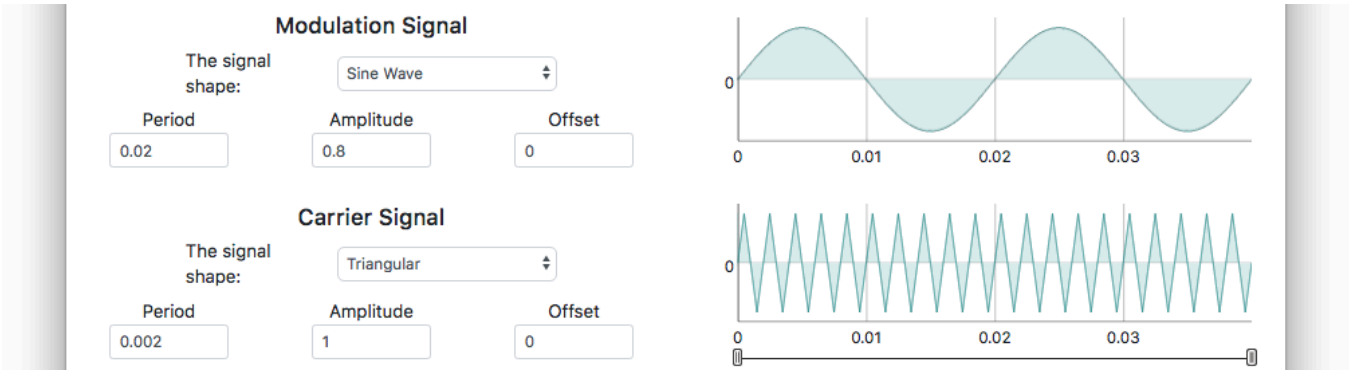

Figure III- 4 The desktop interface: signals parameters

For both signals, we have to enter the following parameters:

- **Signal shape:** we need to choose the shape of each signals, for modulation signal we have: sine wave, cosine wave, constant. For carrier signal, we only have the triangular shape for now.
- **Period:** the period of each signal in seconds, this value is ignored when constant shape signal.
- **Amplitude:** the amplitude value for each signal, typically from 0 to 1. This is not the actual amplitude value of the control signal.
- **Offset:** the offset for each signal if needed, defaults to 0.

III.3.3 General properties

After setting up the parameters for each signal, three other parameters needed to be defined, sampling time, deadtime, duration.

Figure III- 5 The desktop interface: signals parameters: the general parameters

These parameters represent:

- **Sampling time:** the sampling time used to calculate the intersection points that will eventually be sent to the Arduino board.
- **The deadtime:** a small interval during which both the upper and lower switches in a phase-leg are off [20], so that short circuit can be avoided and switches are not damaged due to high current.

III.3.4 Comparison results

Immediately after entering all the necessary parameters, the application automatically generates a preview of the control results in three graphs synchronized by time.

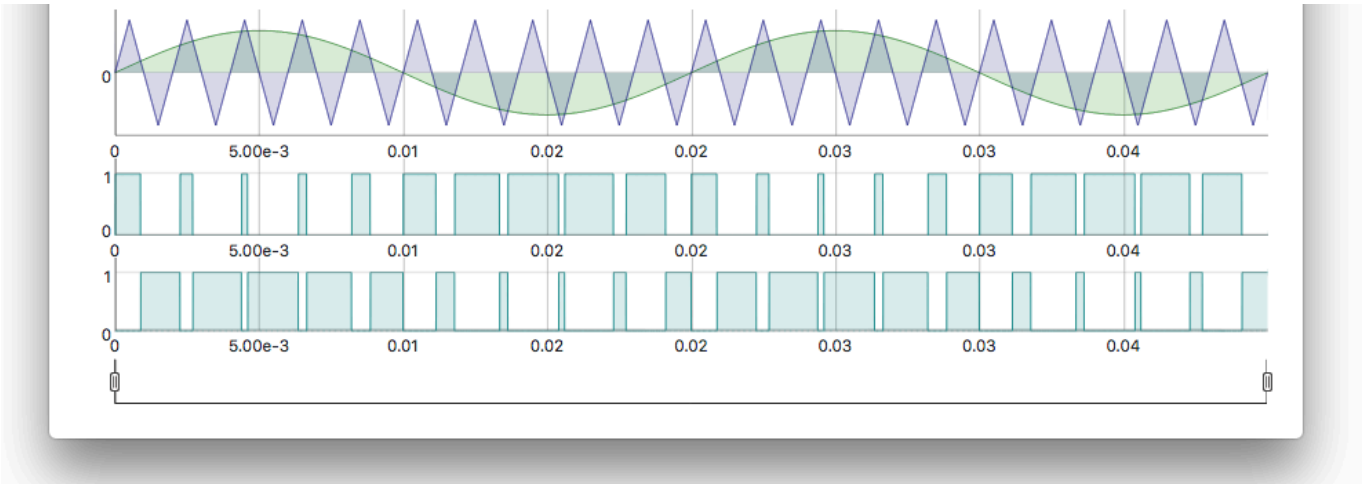

Figure III- 6 The desktop interface: comparison result output

The first graph is basically the two signals: modulation and carrier in top of each other, this will let us see the intersection point easily.

The second two graphs are the result of the comparison, with the dead time value in mind. The first is the direct pulses signal, the second is inverse pulses signal.

III.4 Arduino Program

After defining the control signal, and sending its information to the Arduino board, the last will have all the information needed to generate the control signal.

The code that is used to generate the control signal using Arduino Uno and serial port, can be divided into 4 parts: variables declarations, setup function, the loop function, serial listening function

III.4.1 Variables declarations

Here we declare the necessary variables, we note the main ones: CurrentTime, IPoints, IPValues.

```
unsigned long CurrentTime; // = micros(); current time of Arduino
unsigned long Period; // our pules global period
unsigned long* IPoints; // intersection points: for each intersection, we have two points wit
unsigned long HalfDeadTime; // Dead time
bool* IPValues; // values for each IPoint, ON || OFF
unsigned int IPMaxIndex; // the number of IPoints - 1
unsigned int IPIndex; // the index of the current IPoint
unsigned long PTime; // the current time in Period, which = CurrentTime % Period
long Diff; // difference between upcomming IPoint and PTime
```

Figure III- 7 III.4 Arduino Program: variables section

III.4.2 Setup function

In the setup function, we begin by setting up the serial port to listen for incoming parameters. Then we initialize the registry state of the pins, we use the registry method instead of digitalWrite() for speed. And finally, we define a default signal with a period of 1ms, this signal is generated for the first time the Arduino board boots, and gets replaced when new control parameters comes via the serial port.

```
void setup() {
  Serial.begin(115200); // to enable receiving new signal paramaters via USB port
  DDRB = B11111111; // using the Registers for faster output pin switching

  // sample data:
  Period = 1000; // micro seconds
  IPoints = new unsigned long[4];
  IPoints[0] = 250; IPoints[1] = 500; IPoints[2] = 750; IPoints[3] = 1000;
  IPValues = new bool[4];
  IPValues[0] = true; IPValues[1] = false; IPValues[2] = true; IPValues[3] = false;
  IPIndex = 0;
  IPMaxIndex = 3;
  HalfDeadTime = 10; // micro seconds
}
```

Figure III- 8 Arduino Program: setup section

III.4.3 The loop function

The loop function is where the signal generation algorithm takes place, this algorithm insures two things:

- In each intersection point, the board pins values get updated according the current intersection value.
- Between every update on pins values, there's a small interval of time where all the pins values are OFF, this interval is called the dead time.

```
void loop() {
  CurrentTime = micros(); // get the current time in micro seconds
  PTime = CurrentTime % Period - HalfDeadTime; // get the current time in Period
  Diff = IPoints[IPIndex] - PTime; // difference between upcomming IPoint and PTime

  // the time - the half of deadtime reached the upcomming IPoint
  // also, if the upcomming IPIndex is 0, and we still in the last IPIndex interval, then pas
  if (Diff <= 0 || (IPIndex == IPMaxIndex && Diff >= Period - IPoints[IPMaxIndex - 1])) {

    // set both pins off for deadtime:
    PORTB = B00000000;
    delayMicroseconds(HalfDeadTime * 2);

    // set the upcomming IPValue to the pins
    if (IPValues[IPIndex]) { // direct pulse ON, opposite pulse OFF
      PORTB = B00000001; // pin 8
    } else { // direct pulse OFF, opposite pulse ON
      PORTB = B00000010; // pin 9
    }

    // move to the next point
    if ( IPIndex < IPMaxIndex ) {
      IPIndex++;
    } else {
      IPIndex = 0;
    }
  }
}
```

Figure III- 9 Arduino Program: loop function section

III.4.4 Serial listening function

This function waits for incoming parameters information via serial port from the desktop interface, and updates the control signal parameters according to the new information.

```
// when new signal paramaters recieved
void serialEvent() {
  //  update: Period, IPoints, IPValues, IPCount, IPIndex = 0, Diff=0
  int inChar = Serial.read();
  if (isDigit(inChar)) {
    // convert the incoming byte to a char and add it to the string:
    inString += (char)inChar;
  }
  if (inChar == '|') { // new paramaters are comming
    IPMaxIndex = inString.toInt() - 1;
    Serial.println("clearing...");
    delete[] IPoints;
    delete[] IPValues;
    IPoints = new unsigned long[IPMaxIndex + 1];
    IPValues = new bool[IPMaxIndex + 1];
    IPIndex = 0;
    NewIPIndex = 0;
    // clear the string for new input:
    inString = "";
    Serial.println("updating...");

  }
  if (inChar == 'h' || inChar == 'l') {
    IPoints[NewIPIndex] = inString.toInt();
    if (inChar == 'h') {
      IPValues[NewIPIndex] = true;
    } else {
      IPValues[NewIPIndex] = false;
    }
    NewIPIndex++; // move the index to the next new IPoint
    // clear the string for new input:
    inString = "";
  }
  if (inChar == 'd') {
    HalfDeadTime = inString.toInt() / 2;
    // clear the string for new input:
    inString = "";
  }
  if (inChar == 'p') {
    Period = inString.toInt();
    // clear the string for new input:
    inString = "";
  }
  // if you get a newline, then it's done
  if (inChar == '\n') {
    IPIndex = 0;
    NewIPIndex = 0;
    // clear the string for new input:
    inString = "";
    Serial.println("done.");
    Serial.print("Count "); Serial.println(IPMaxIndex + 1);
    Serial.print("Period "); Serial.println(Period);
    Serial.print("DeadTime "); Serial.println(HalfDeadTime * 2);
  }
}
```

Figure III- 10 Arduino Program: serial listening section

III.5 Control circuit

the signal generated by the Arduino board, can't be directly use on the switches triggers of the Inverter, instead, in needs to go through a circuit that contains a driver, with a voltage regulator as shown below:

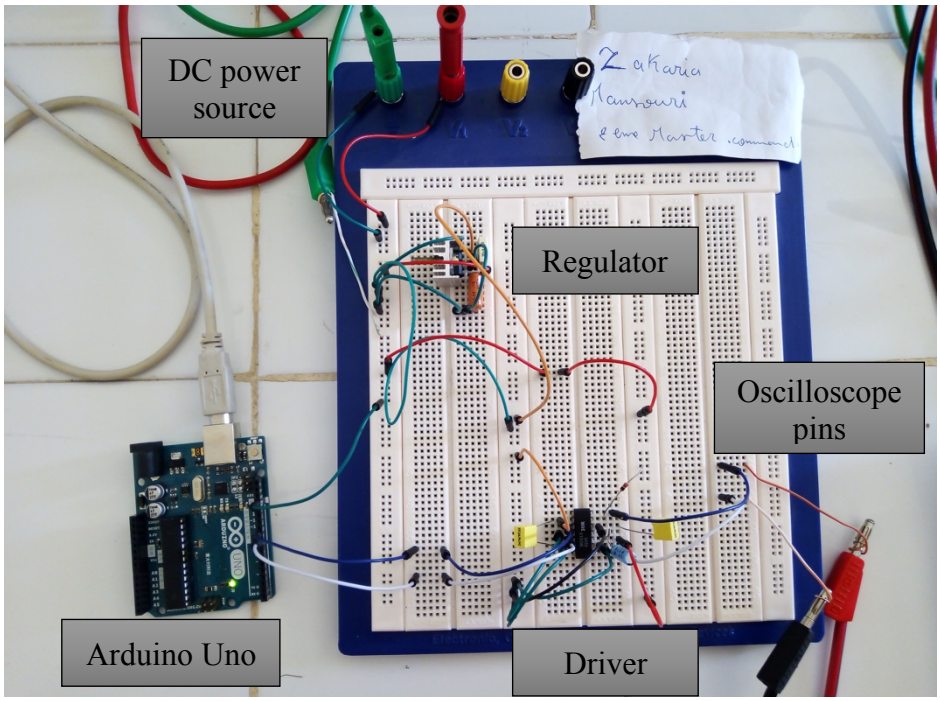

Figure III- 11 the control circuit

this is the schematic for the control circuit, where PWM1 and PWM1d are the Arduino pins 8 and 9:

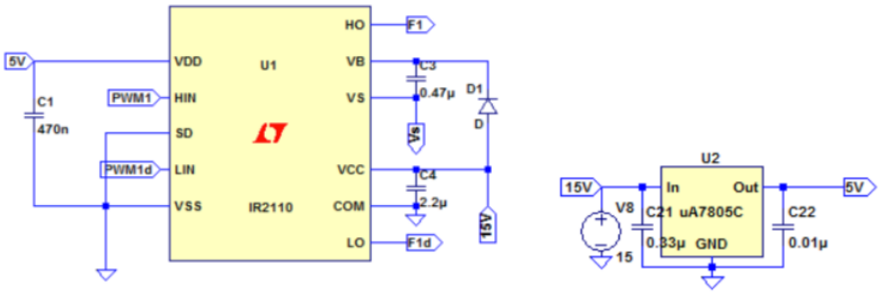

Figure III- 12 schematic representation of the control circuit

The driver has the aim to adapt the voltage from the microcontroller's output (5V) to the requirements of the switch gate of the inverter (15V). Having a look into the recommendations in the datasheet, the above schematic is chosen.

HIN and **LIN** are the logic inputs, corresponding to the PWM signal that comes from pins 8 and 9. **V_{ss}**, **SD** and **COM** are set into ground.

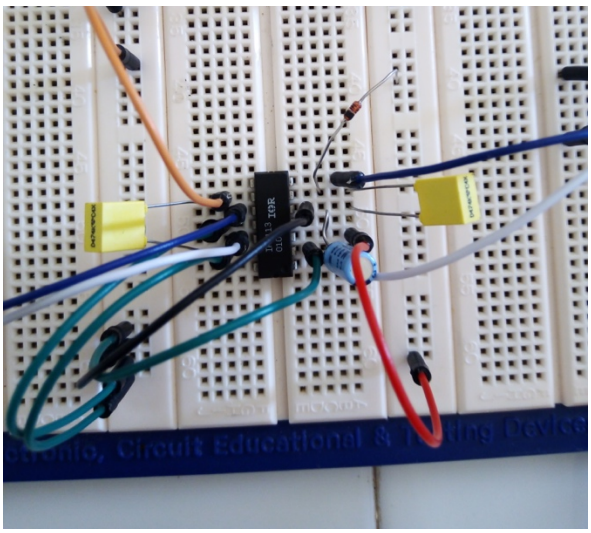

Figure III- 13 the circuit for IR2110 driver

III.6 Power circuit

The power circuit uses a Full-Bridge configuration. A capacitor voltage divisor is set and the output is referred to the middle point. The load used in this case a is a resistor of 500 Ω and the input voltage is 15V. The drivers' Vs pin is connected to the middle point of the leg of each Half Bridge. The schematic is shown below.

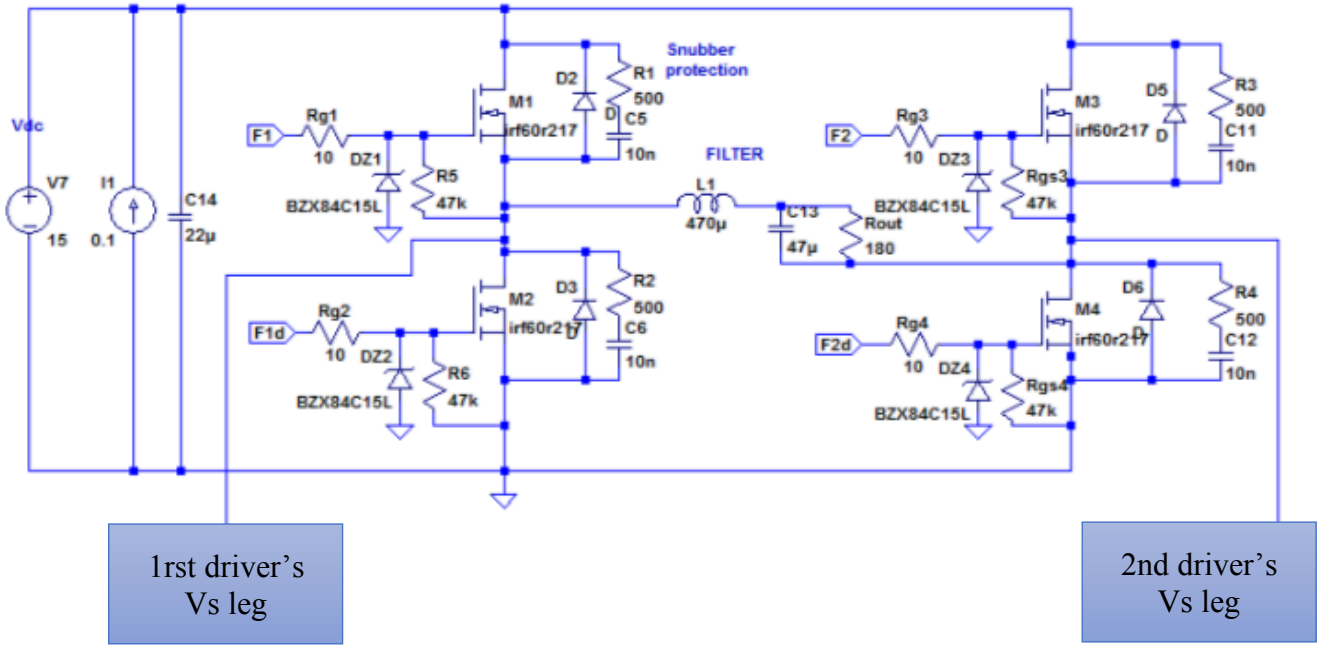

Figure III- 14 the schematic for H-bridge circuit

III.7 Results

These results are from the laboratory, the setup doesn't include the power circuit, instead we used an oscilloscope to visualize the two control signals as shown below.

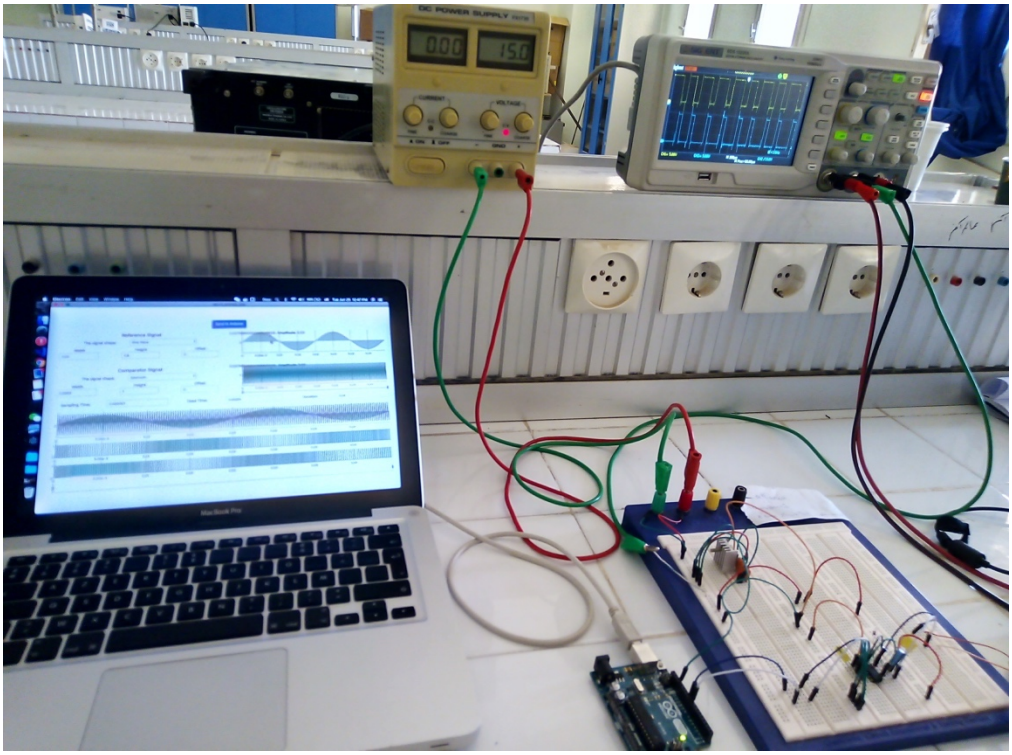

Figure III- 15 project setup in the laboratory

we used three deferent control signals: square waves, modified sine wave, and pure sine wave. On the oscilloscope, the first channel is for direct pulses signal, and the second channel is for the inverse pulses signal. The results are as follows:

III.7.1 Square waves

This is the result of a square wave control signal, with the parameters:

- Modulation signal: constant signal with the amplitude of 0.
- Carrier signal: a triangular signal with amplitude of 1 and period of 0.02s.
- Dead time: 0.00001s.

Figure III- 16 Desktop interface preview: square waves

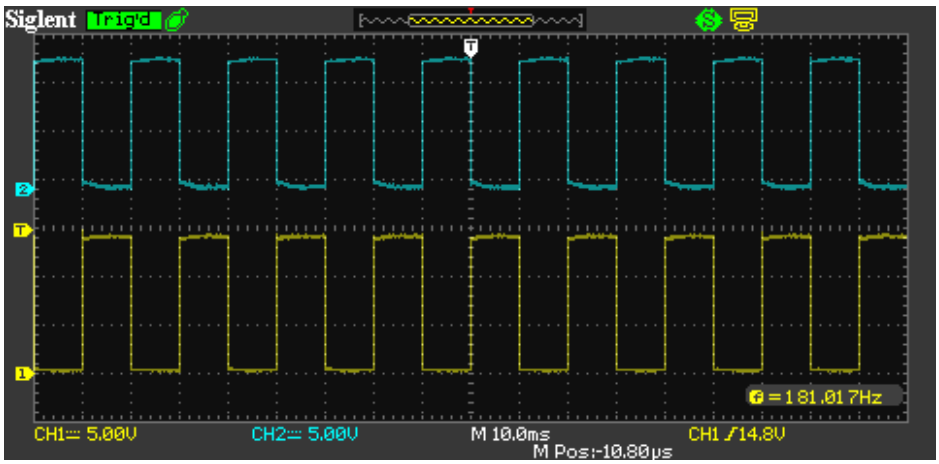

Figure III- 17 Oscilloscope output (10ms): square waves

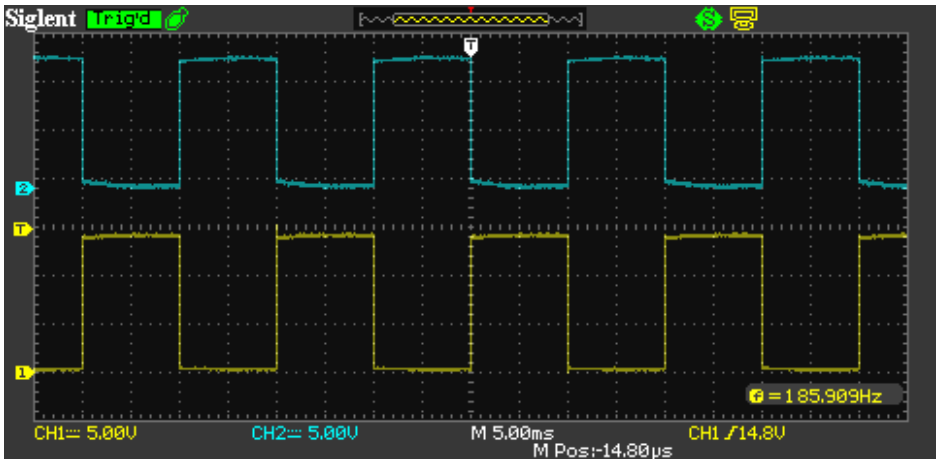

Figure III- 18 Oscilloscope output (5ms): square waves

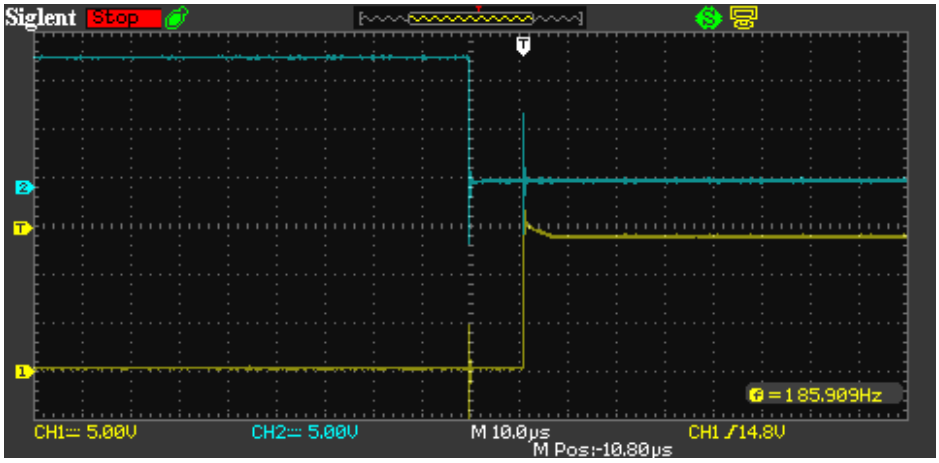

Figure III- 19 Oscilloscope output (10us): square waves

The results are exactly the same as the preview on the desktop interface. We can see the dead time (10us) when zooming on the oscilloscope. We can also see the oscillation when switching from 0 to 15V (or vice versa) due the fast response from the driver.

III.7.2 Modified sine wave

The modified since wave control signal has the same as square wave parameters, with a much bigger dead time value:

- Modulation signal: constant signal with the amplitude of 0.
- Carrier signal: a triangular signal with amplitude of 1 and period of 0.02s.
- Dead time: 0.003s.

Figure III- 20 Desktop interface preview: Modified sine wave

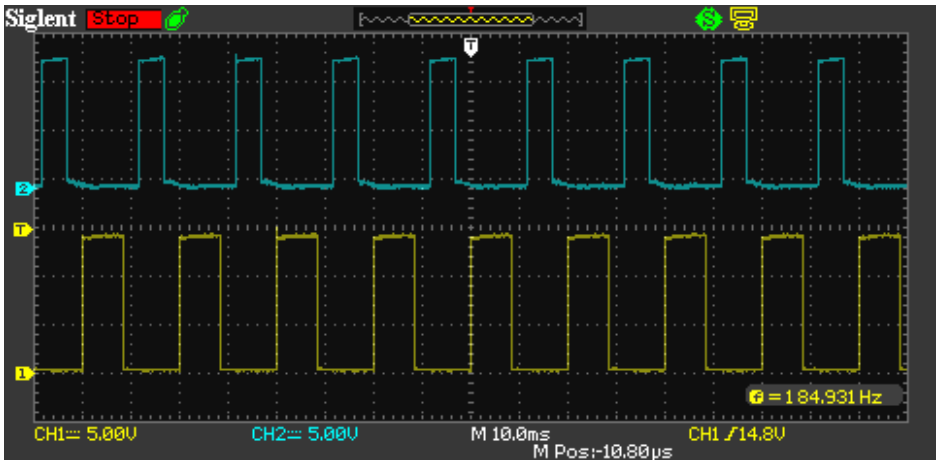

Figure III- 21 Oscilloscope output (10ms): Modified sine wave

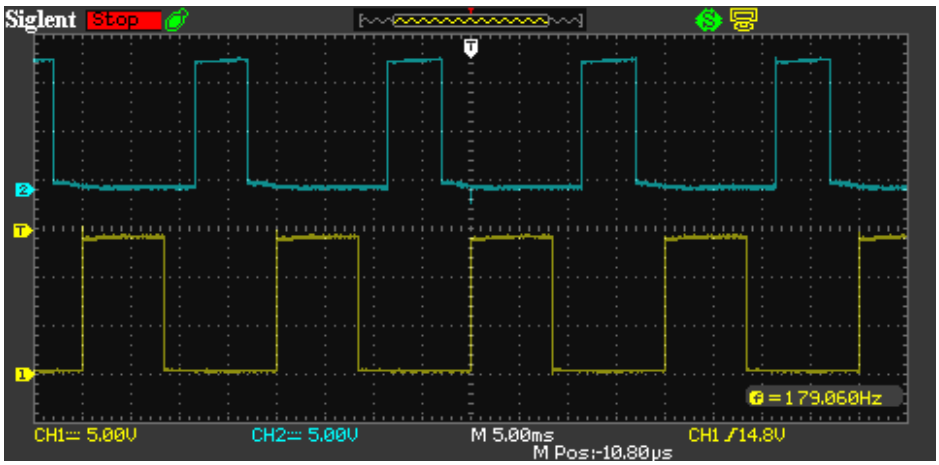

Figure III- 22 Oscilloscope output (5ms): Modified sine wave

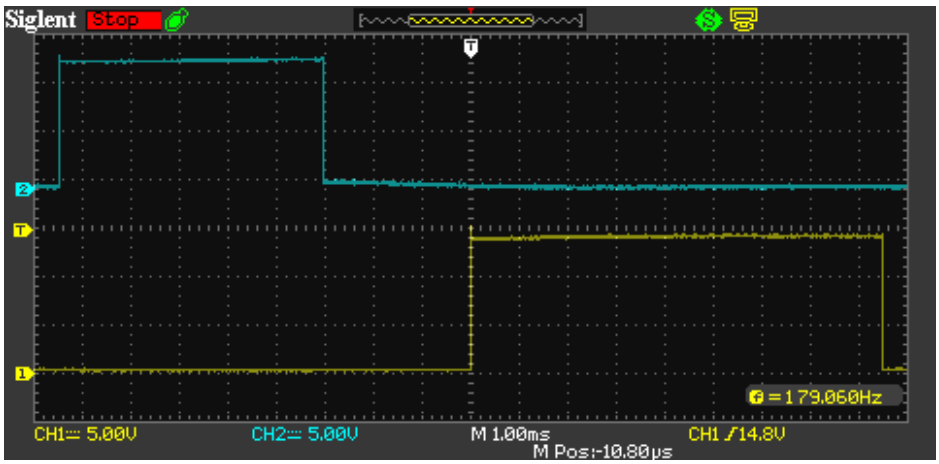

Figure III- 23 Oscilloscope output (1ms): Modified sine wave

The results are almost the same as the preview on the desktop interface. Now we can see the dead time (0.003s) on a 10ms oscilloscope zoom. As we can see the direct pulses signal has a bit lesser duty cycle than the inverse pulses signal, and that's due to the algorithm is not well optimized.

III.7.3 Pure sine wave

On pure sine wave, we tested the output on two different values of the carrier signal, one with a period of 0.0002s, and other with a period of 0.002s.

III.7.3.1 Pure sine wave with carrier period of 0.0002s

This is the result of a Pure sine wave control signal, with the parameters:

- Modulation signal: sine wave signal with the amplitude of 0.8 and period of 0.02s.
- Carrier signal: a triangular signal with amplitude of 1 and period of 0.0002s.
- Dead time: 0.00001s.

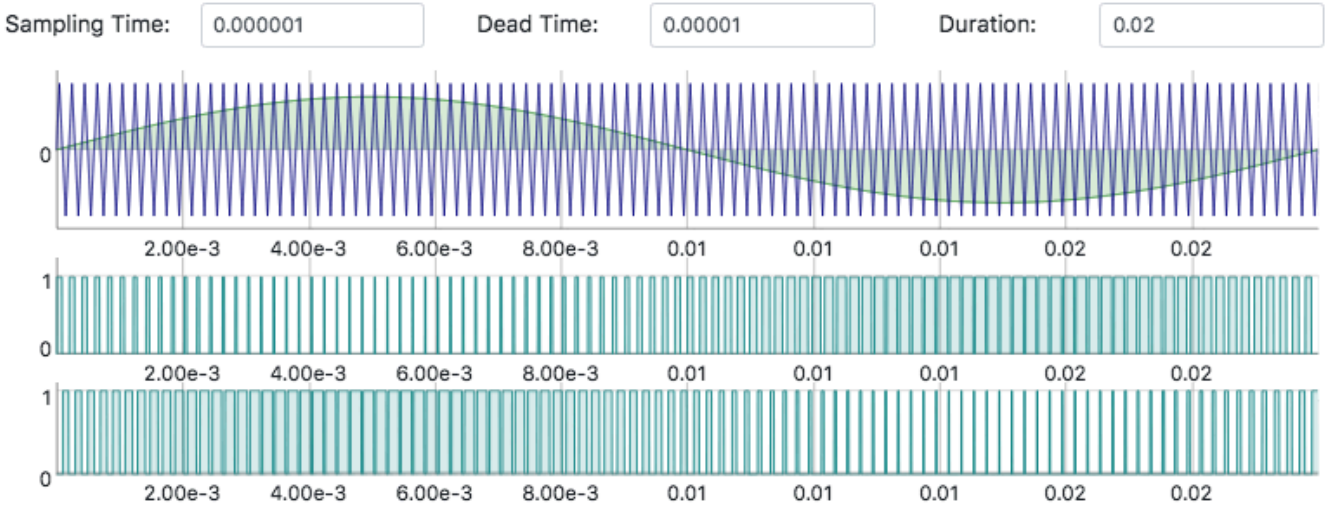

Figure III- 24 Desktop interface preview: pure sine wave (carrier period=0002s)

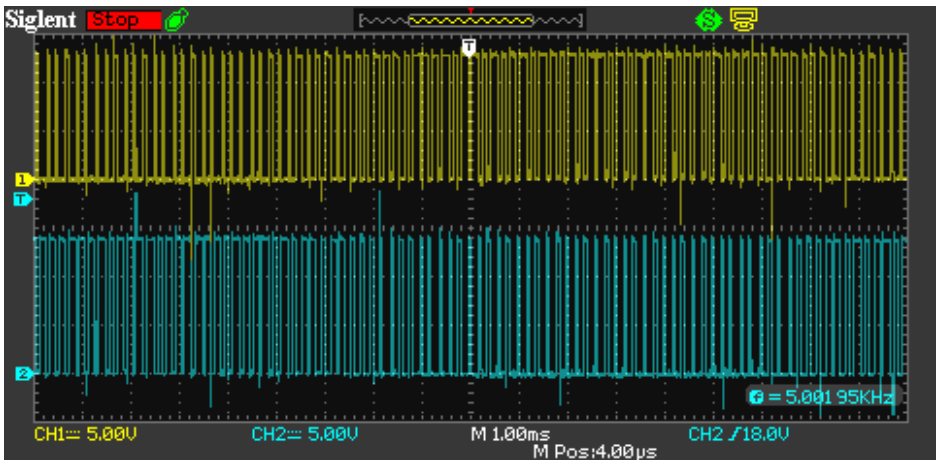

Figure III- 25 Oscilloscope output (1ms): Pure sine wave (carrier period=0002s)

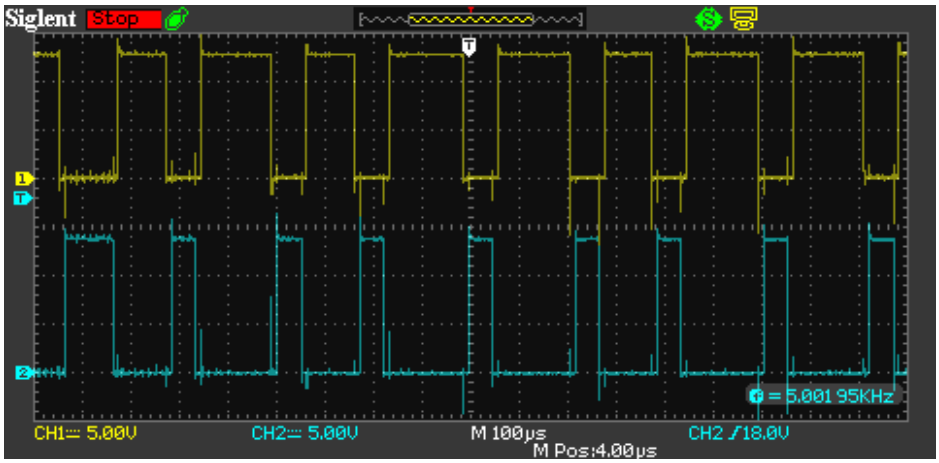

Figure III- 26 Oscilloscope output (100us): Pure sine wave (carrier period=0002s)

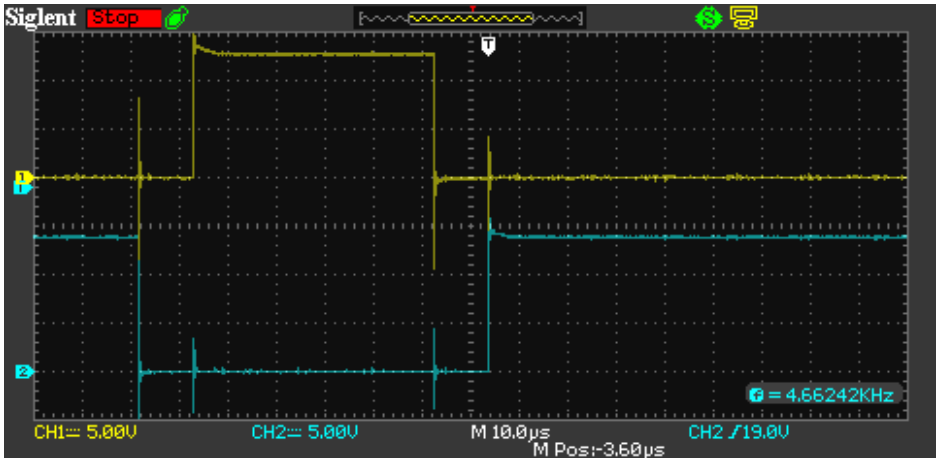

Figure III- 27 Oscilloscope output (10us): Pure sine wave (carrier period=0002s)

As expected, the results are exactly the same as the preview on the desktop interface. We can see the dead time (10us) when zooming on the oscilloscope. We can also see the oscillation when switching from 0 to 15V (and vice versa) due the fast response from the driver.

III.7.3.2 Pure sine wave with carrier period of 0.002s

This is the result of a Pure sine wave control signal, with the parameters:

- Modulation signal: sine wave signal with the amplitude of 0.8 and period of 0.02s.
- Carrier signal: a triangular signal with amplitude of 1 and period of 0.002s.
- Dead time: 0.00001s.

Figure III- 28 Desktop interface preview: pure sine wave (carrier period=002s)

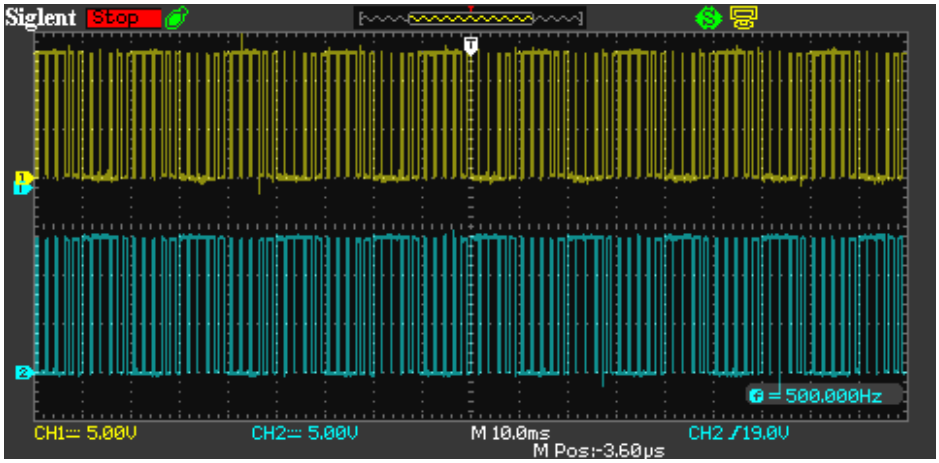

Figure III- 30 Oscilloscope output (10ms): Pure sine wave (carrier period=002s)

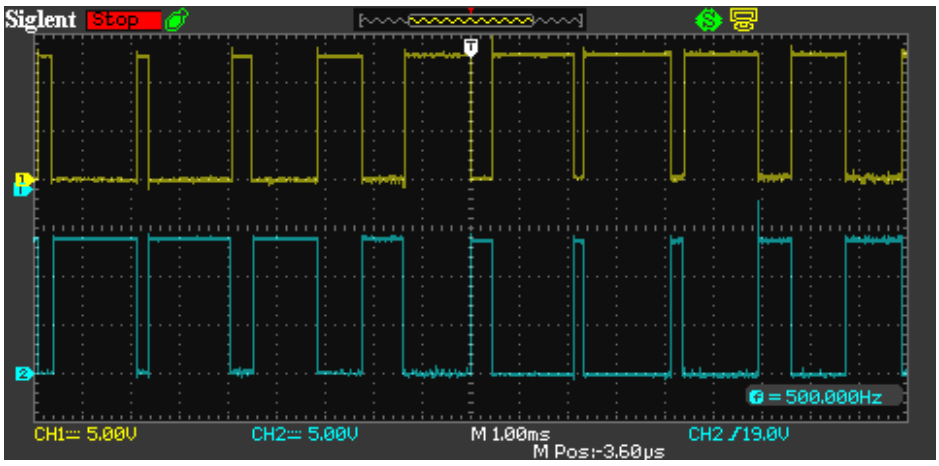

Figure III- 30 Oscilloscope output (1ms): Pure sine wave (carrier period=002s)

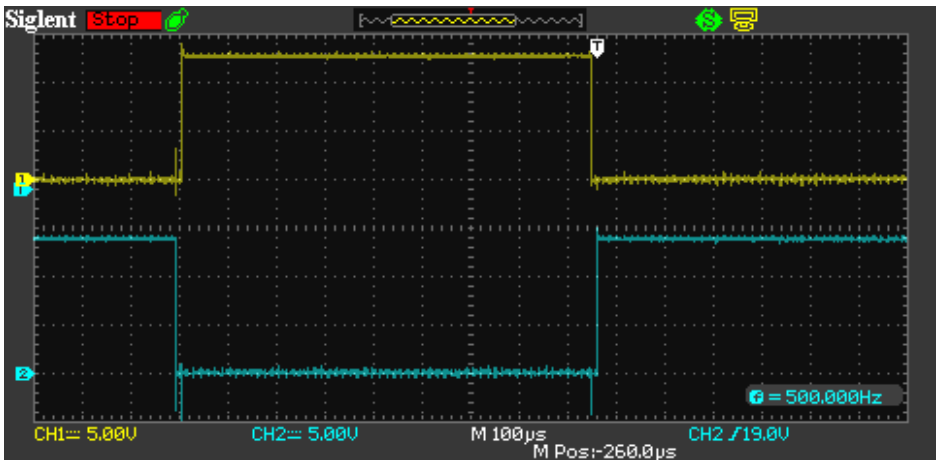

Figure III- 31 Oscilloscope output (100us): Pure sine wave (carrier period=002s)

Same as the previews signal, the results are exactly the same as the preview on the desktop interface.

III.8 Conclusion

This chapter was dedicated to the presentation of the results of control signals of the inverter, allowed us to confirm, the reliability of the control technic realized with the association of the control board (Arduino), programmed in Arduino C, which allowed the simplification of the implementation of the control algorithm (fixed duty cycle, SPWM) using a desktop interface on the computer.

General conclusion

As part of the preparation of the Master's degree in Electrical Engineering, this work aims to present a theoretical study of the single-phase inverter, and practical realization of its control signals, that took place at the laboratory of the department of electrical engineering at the university of Mohammed Khider Biskra.

This work is organized in three chapters, starting with an introduction. In the first chapter, after the presentation we presented general notions about inverters and its control strategies and its applications.

The second chapter was devoted to explain the operation of the Arduino board and the desktop interface, and how they connect together; we gave the diagram of the internal structure of the board and talked about the Arduino Uno features. We used the environment to create a sketch, which the main objective of the sketch is to control the inverter. We discussed the technologies available for the developing a desktop application, including Electron JS. We gave an overview of Electron JS and explained why we choose it.

The third chapter we talked about the realization of the project, we gave an overview diagram, then went through each part of the project, from desktop interface, to Arduino board, passing by the control circuit, and finally to the power circuit. We presented the results of experimental in three different control technique (both fixed duty cycle controls, and the SPWM technique).

1. Objectives achieved

The control strategies (square wave pulse, PWM), are successfully realized and tested, and proved to be working, despite the limitation of the Arduino board. The power circuit has not been realized because of the lack of power components required at the laboratory on the one hand, and secondly the short time allocated to the completion of this work.

2. Further developments

There are some parts of this project that can be improved even more. The Arduino code algorithm can be optimized, when looking for intersection points. The desktop interface can also add other features, like the automatic calculation of signal parameters based on a specific control target e.g.: Electric motor speed control.

Bibliography

[1] I. o. E. a. E. Engineers, "IEEE 100: The Authoritative Dictionary of IEEE Standards Terms," Standards Information Network, IEEE Press, 2000, p. 588.

[2] M. H. Rashid, Power electronics handbook, Butterworth-Heinemann, 2017.

[3] R. Du and P. Robertson, "Cost-Effective Grid-Connected Inverter for a Micro Combined Heat and Power System," *IEEE Transactions on Industrial Electronics,* vol. 64, no. 7, pp. 5360-5367, 2017.

[4] H. Abdi, R. R. Nezhad and M. Salehimaleh, "Chapter 5 - Fuel Cells," in *Distributed Generation Systems*, Butterworth-Heinemann, 2017, pp. 221-300.

[5] M. H. Rashid, Electric Renewable Energy Systems, Academic Press, 2015.

[6] M. H. Rashid, "16 - DC–AC inverters," in *Electric Renewable Energy Systems*, Boston, Academic Press, 2016, pp. 354 - 381.

[7] D. Fewson, "4 - DC to AC inverters," in *Introduction to Power Electronics*, Oxford, Butterworth-Heinemann, 1998, pp. 66-94.

[8] S. K. Mitra and J. F. Kaiser, Handbook for digital signal processing, John Wiley \& Sons, Inc., 1993.

[9] A. LLC, "Arduino - Introduction," Arduino LLC, [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 30 6 2019].

[10] A. LLC, "Arduino Uno Rev3," Arduino LLC, [Online]. Available: https://store.arduino.cc/arduino-uno-rev3. [Accessed 30 6 2019].

[11] Atmel®, *8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash,* Atmel®, 2015.

[12] A. LLC, "Arduino - PinMapping168," Arduino LLC, [Online]. Available: https://www.arduino.cc/en/Hacking/PinMapping168. [Accessed 30 6 2019].

[13] A. LLC, "Arduino Reference," Arduino LLC, [Online]. Available: https://www.arduino.cc/reference/en/. [Accessed 06 30 2019].

[14] V. Beal, "What is serial port? - A Word Definition From the Webopedia Computer Dictionary," webopedia, [Online]. Available: https://www.webopedia.com/TERM/S/serial_port.html. [Accessed 7 1 2019].

[15] Oracle, "Java SE | Oracle Technology Network | Oracle," Oracle, [Online]. Available: https://www.oracle.com/technetwork/java/javase/overview/index.html. [Accessed 1 07 2019].

[16] V. Beal, "What is .NET Framework? Webopedia Definition," webopedia, [Online]. Available: https://www.webopedia.com/TERM/D/dot_NET_Framework.html. [Accessed 01 07 2019].

[17] electronjs.org, "About Electron | Electron," Github, [Online]. Available: https://electronjs.org/docs/tutorial/about. [Accessed 01 07 2019].

[18] D. Flanagan, "Introduction to JavaScript," in *JavaScript - The definitive guide (6 ed.)*, O'Reilly Media, 2011, p. 1.

[19] nodejs.org, "About | Node.js," Node.js Foundation, [Online]. Available: https://nodejs.org/en/about/. [Accessed 01 07 2019].

[20] L. Chen and F. Z. Peng, "Dead-Time Elimination for Voltage Source Inverters," *IEEE Transactions on Power Electronics,* vol. 23, no. 2, pp. 574-580, 2008.

Annex

The Arduino code:

```
unsigned long CurrentTime; // = micros(); current time of Arduino
unsigned long Period; // our pules global period
unsigned long* IPoints; // intersection points: for each intersection, we have two points with a distance of DeadTime between them
unsigned long HalfDeadTime; // Dead time
bool* IPValues; // values for each IPoint, ON || OFF
unsigned int IPMaxIndex; // the number of IPoints - 1
unsigned int IPIndex; // the index of the current IPoint
unsigned long PTime; // the current time in Period, which = CurrentTime % Period
long Diff; // difference between upcomming IPoint and PTime

void setup() {
  Serial.begin(115200); // to enable receiving new signal paramaters via USB port
  DDRB = B11111111; // using the Registers for faster output pin switching

  // sample data:
  Period = 1000; // micro seconds
  IPoints = new unsigned long[4];
  IPoints[0] = 250; IPoints[1] = 500; IPoints[2] = 750; IPoints[3] = 1000;
  IPValues = new bool[4];
  IPValues[0] = true; IPValues[1] = false; IPValues[2] = true; IPValues[3] = false;
  IPIndex = 0;
  IPMaxIndex = 3;
  HalfDeadTime = 10; // micro seconds
}

void loop() {
  CurrentTime = micros(); // get the current time in micro seconds
  PTime = CurrentTime % Period - HalfDeadTime; // get the current time in Period
  Diff = IPoints[IPIndex] - PTime; // difference between upcomming IPoint and PTime
```

```
  // the time - the half of deadtime reached the upcomming IPoint
  // also, if the upcomming IPIndex is 0, and we still in the last IPIndex interval, then pass to the next loop
  if (Diff <= 0 || (IPIndex == IPMaxIndex && Diff >= Period - IPoints[IPMaxIndex - 1])) {

    // set both pins off for deadtime:
    PORTB = B00000000;
    delayMicroseconds(HalfDeadTime * 2);

    // set the upcomming IPValue to the pins
    if (IPValues[IPIndex]) { // direct pulse ON, opposite pulse OFF
      PORTB = B00000001; // pin 8
    } else { // direct pulse OFF, opposite pulse ON
      PORTB = B00000010; // pin 9
    }

    // move to the next point
    if ( IPIndex < IPMaxIndex ) {
      IPIndex++;
    } else {
      IPIndex = 0;
    }
  }
}

String inString = "";
unsigned int NewIPIndex = 0; // the index of the new IPoints

// when new signal paramaters recieved
void serialEvent() {
  //  update: Period, IPoints, IPValues, IPCount, IPIndex = 0, Diff=0
  int inChar = Serial.read();
  if (isDigit(inChar)) {
    // convert the incoming byte to a char and add it to the string:
    inString += (char)inChar;
  }
```

```
if (inChar == '|') { // new paramaters are comming
  IPMaxIndex = inString.toInt() - 1;
  Serial.println("clearing...");
  delete[] IPoints;
  delete[] IPValues;
  IPoints = new unsigned long[IPMaxIndex + 1];
  IPValues = new bool[IPMaxIndex + 1];
  IPIndex = 0;
  NewIPIndex = 0;
  // clear the string for new input:
  inString = "";
  Serial.println("updating...");

}
if (inChar == 'h' || inChar == 'l') {
  IPoints[NewIPIndex] = inString.toInt();
  if (inChar == 'h') {
    IPValues[NewIPIndex] = true;
  } else {
    IPValues[NewIPIndex] = false;
  }
  NewIPIndex++; // move the index to the next new IPoint
  // clear the string for new input:
  inString = "";
}
if (inChar == 'd') {
  HalfDeadTime = inString.toInt() / 2;
  // clear the string for new input:
  inString = "";
}
if (inChar == 'p') {
  Period = inString.toInt();
  // clear the string for new input:
  inString = "";
}
```

```
  // if you get a newline, then it's done
  if (inChar == '\n') {
    IPIndex = 0;
    NewIPIndex = 0;
    // clear the string for new input:
    inString = "";
    Serial.println("done.");
    Serial.print("Count "); Serial.println(IPMaxIndex + 1);
    Serial.print("Period "); Serial.println(Period);
    Serial.print("DeadTime "); Serial.println(HalfDeadTime * 2);
  }
}
```

Desktop interface code: *the intersection calculation code:*

```
export const getIntersections =
  (payload: { modSignal: ISignal, carSignal: ISignal, globalParams: IGlobalParams }) => {

    const intersections = [];

    const refSignal = payload.modSignal;
    const compSignal = payload.carSignal;
    const samplingTime = payload.globalParams.samplingTime;
    const deadTime = payload.globalParams.deadTime;
    const period = refSignal.width;

    const refSignalFunction = getFuncByShape(refSignal.shape);
    const compSignalFunction = getFuncByShape(compSignal.shape);

    let rp: number;
    let cp: number;
    let dpp: number;
    let oldDpp: number;
    for (let i = 0; i <= period + samplingTime; i += samplingTime) {
      rp = refSignalFunction(refSignal.width, refSignal.height, i, refSignal.offset);
```

```
    cp = compSignalFunction(compSignal.width, compSignal.height, i, compSignal.offset);
    dpp = (cp > rp ? 1 : (cp < rp ? 0 : dpp));
    if (oldDpp !== dpp || i === period) { // intersection point found !!
      intersections.push(((i) * 1000000).toFixed(0) + (dpp ? 'h' : 'l'));
    }
    oldDpp = dpp;
  }
  // add the period
  intersections.push((period * 1000000).toFixed(0) + 'p');
  // add the deadtime
  intersections.push((deadTime * 1000000).toFixed(0) + 'd');
  intersections.shift();
  return intersections;
};
```