UNIT 5: ANALOG SIGNALS

AIMS

Up until now we've assumed that Arduino only uses digital signals with levels of "1" or "0". Input signals can come from pushbuttons, switches, detectors and many other sources but must be preceded by a "1" or a "0". Output signals can power LED lights, relay switches, motors and many other devices but once again we must write a "1" or a "0" level to them.

But not everything in the natural world is digital. There are also what are called "analog" signals and their value or voltage may vary between a minimum and maximum over a period of time. Even Arduino controller, like the majority of modern controllers, is capable of generating width modulated digital signals. They're called "PWM" ("Pulse Width Modulation") or "PDM" (Pulse-duration Modulation) signals.

In this unit you're going to study what these signals are like, how to use them and what you can do with them. You'll be learning about the functions in Arduino language that manipulate the controller's analog inputs.

THEORY SECTION

- INTRODUCTION
- DIGITAL CONVERSION
- RESOLUTION
    - And now it's your turn
- FUNCTIONS IN ARDUINO LANGUAGE
- ANALOG PERIPHERALS
    - Potentiometers
    - Photo sensors
    - IR reflective sensors
    - Air conditioners
- PSEUDO ANALOG OUTPUT
    - What are PWM signals?

- What are they used for?
- How are they generated?
  - The analogWrite() function:

- Further functions in Arduino language
  - The random() function
  - The randomSeed() function

PRACTICE SECTION

- EXAMPLE 1: ADC Conversion
- EXAMPLE 2: Thresholds
- EXAMPLE 3: Analog comparator
- EXAMPLE 4: Brightness control
- EXAMPLE 5: Electric rudder
- EXAMPLE 6: Photometer
- EXAMPLE 7: Lighting control
- EXAMPLE 8: Measuring reflections
- EXAMPLE 9: Detecting colours
- EXAMPLE 10: Temperature
- EXAMPLE 11: Air handler
- EXAMPLE 12: A PWM signal
- EXAMPLE 13: Optical effects
- EXAMPLE 14: Manual regulation
- EXAMPLE 15: Random lights

PRACTICE MATERIALS

-Lap top or desk top computer

-Arduino IDE work environment; this should include the supplementary material already installed and configured.

-Arduino UNO controller board

-A USB cable

TABLE OF CONTENTS

THEORY SECTION

1. INTRODUCTION

The expression “analog input signals” may have come up at some point during the course but just what are they, what are they used for, where are they and how are they controlled? The time has come to answer these questions.

You’re already aware that everything in the “digital” world works on the assumption that there are only two possible values or levels: level “1”and level “0”. A pushbutton, a switch or a detector can be on (“1”) or off (“0”). A LED light, relay switch or a motor can be turned on or off. Sound is nothing more than a signal that goes from level “1” to level “0” at a given speed or frequency. A PWM signal is digital and we can vary the length of the level “1” or duty cycle and thus regulate the voltage. Serial communication is nothing more than the transfer of bits with levels of “1” and “0”.

When we’ve got a number of bits we put them together into bytes and use them to create different sized numbers, encode characters and send messages. In any case, this is the way you’ve been working up until now.

Nevertheless, the “real” world isn’t like that. We come across physical quantities in the natural world that may have multiple values or other features. In any case, I’m sure you’re only too well aware that not everything is black and white; there are greys as well.

Have a think about room temperature for instance. This is a physical quantity that changes constantly. The temperature isn’t the same in the morning as it is at midday or in the evening. If we had a sensor that could measure the temperature and generate a voltage proportional to it, we’d see that it varied constantly over time more or less.

Temperature is an analogical physical quantity. The sensor delivers a voltage of 2 V at 5 0’clock in the morning. At 9 o’clock the temperature goes up and therefore the voltage does too; it rises to 4 V. At 3 o’clock the voltage reaches 5 V and from 4 o’clock onwards it starts to fall with the temperature. All we have to do is look for a relationship between the temperature and the voltage the sensor delivers.

Now have a think about the huge number of analogical physical quantities all around us. Using suitable sensors or “transducers” you can transform these physical quantities into equivalent analog voltages:

- ✓ Humidity and/or relative humidity. This enables us to calculate the amount of water vapour in the atmosphere.
- ✓ Atmospheric pressure. We can find out the pressure the air exerts on the Earth using a suitable sensor.
- ✓ Weight. We can measure the force a body exerts on a point at rest.
- ✓ Speed. If you had a suitable sensor you could measure how fast the air moves or how fast a vehicle moves or how fast you move.

- ✓ Light. You can measure the ambient light or the light that strikes an object. There are sensors that detect visible light, infrared light, ultraviolet light and others as well.
- ✓ Sound. We can measure, study and/or detect noises, ambient sound, volume and other acoustic phenomena.

There are many other areas sensors or “transducers” that can transform physical quantities into equivalent analog voltages.

2. DIGITAL CONVERSION

As you may well imagine, there are all types of sensors and “transducers” that are able to measure and deliver analog voltage equivalent to the physical quantity they’re measuring. Unfortunately, no digital system is able to handle or process these analog currents directly; not even Arduino.

The very first thing we have to do is convert the analog voltages into their equivalent digital or binary values. To perform this conversion we use electronic circuits called “analog-to-digital converters”, abbreviated to “ADC”.

Have a look at the Figure 1. It shows the components you need to process an analogical physical quantity like temperature, for instance.

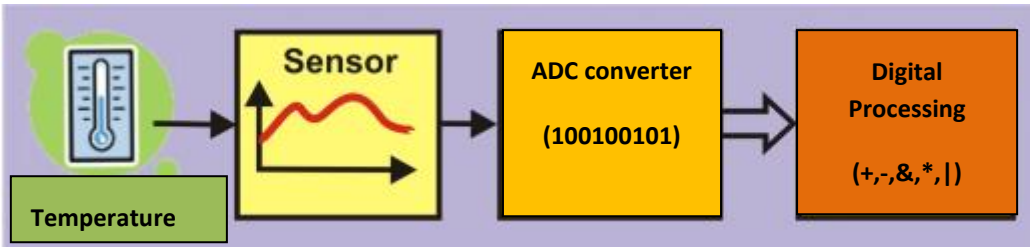

Figure 1

1. The sensor detects the physical quantity it’s designed to measure; in this case it’s temperature. It usually generates a voltage proportional to the physical quantity.
2. The voltage is then fed into the analog-to-digital converter (ADC). The circuit gives out a binary value equivalent to the input voltage.
3. The binary value can now be read by a controller like Arduino.
4. The controller can perform any kind of process with this binary value: store it in the memory, perform arithmetical or logical operations, display it, transfer it to another system and lots of others.

Just about all modern controllers including Arduino have an integrated ADC converter circuit. All you have to do is connect up the appropriate sensor or transducer to the analog input pin. The type of sensor or transducer you use will depend on the physical quantity you’re going to measure.

And what’s more, these controllers usually have a number of input pins for analog signals. Have a look at the figure on the left. Most controllers only have one ADC converter circuit with a number of

input pins or “analog channels” connected to it. Arduino UNO has a single converter and six analog input pins connected to it: numbers A0 to A5. This enables it to take samples of analog voltages from as many as six different sensors.

Of course you can only take one sample at a time. By using suitable functions you can convert the voltages on each analog input channel one at a time. We say that the input channels are “multiplexed”.

Have a close look at the figure Figure 2. Every time you request a conversion the ADC converter takes a sample of the analog voltage on the input channel you’ve specified. The result, or its binary equivalent, is obtained in 100 µS (0.0001”), which is how long it takes to perform the conversion. Arduino can perform approximately 10,000 conversions a second.

The analog voltage is 2 V at the first point in time. The ADC converter generates a value or binary number equivalent to this voltage. The analog voltage is 2.8 V at the second point in time, 3.5 V at the third, 4.2 V at the fourth and so on and so forth. You can see what an analog signal is like: it fluctuates constantly between a minimum – 0 V in the example - and a maximum - 5 V.

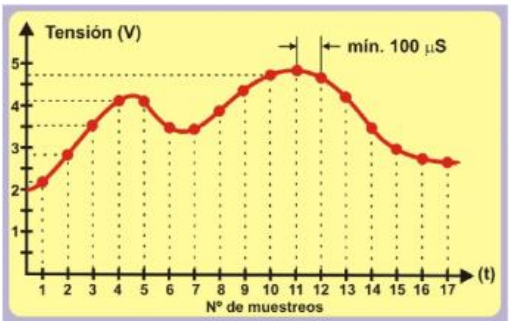

Figure 2

I’ve already said that Arduino UNO takes approximately 100 µS to perform a conversion. This is quite fast but there are others that can do it a lot faster. Have a look at the Figure 3. It shows an analogical signal with a frequency, F, of 1000 Hz. Its period, T, is 1000 µS (1 mS). This means that Arduino would be able to take ten samples of the signal at the most (1000 µS / 100 µS).

If you had an analogical signal with a frequency, F, of 500 Hz and a period, T, of 2000 µS, you could take a total of twenty samples of it (2000 µS / 100 µS). This means that you’d get a more exact “digitalization” than when the frequency of the analog signal was 1000 Hz.

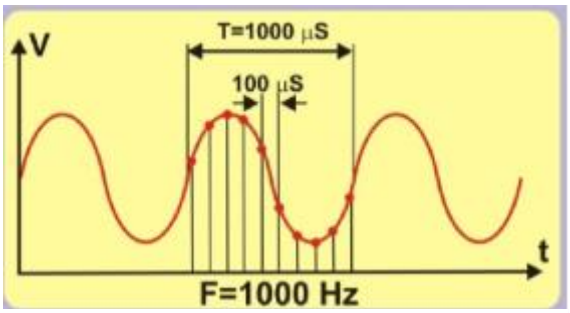

Figure 3

You don't always have to take that many samples per second. Remember the sensor that measures the room temperature? This is a physical quantity that doesn't modulate very much. The temperature doesn't rise from -10ºC to +15ºC in 0.0001 seconds. The same thing happens with natural light. We don't go from night to day in 100 µS. And what about weight? Nothing weighs 75 kg one moment and 31 kg 100 µS later. The same applies to other physical quantities like speed, atmospheric pressure, humidity and many others.

Arduino UNO's conversion speed is more than enough to measure most analogical physical quantities!

3. RESOLUTION

In addition to the conversion speed, another important factor in an ADC converter circuit is its precision or "resolution". We say that the converter incorporated into the Arduino UNO has a resolution of 10 bits. This means that the result of a conversion may have 1024 possible binary values (2^{10}).

How do we establish a relationship between the analog voltage and the binary value? We need to know a constant called the "reference voltage" or V_{REF} which is the voltage the converter circuit uses to perform its internal operations. We calculate the resolution per bit using the following equation which depends on both the V_{REF} and the number of bits the converter's got. Assuming that V_{REF} = 5V:

$$Resolución = \frac{V_{REF}}{2^{10}} = \frac{5}{1024} = 0.0048V/Bit \cong 0.005V = 5mV$$

If you know this you can predict the binary output value the converter gives you based on the analogical input voltage. Have close look at the following table.

INPUT VOLTAGE	OUTPUT			OUTPUT VOLTAGE	OUTPUT		
	BINARY	DEC.	HEX.		BINARIO	DEC.	HEX.
0.000	0000000000	0	0x000	**2.500**	1000000000	512	0x200
0.005	0000000001	1	0x001	**3.000**	1001100110	614	0x266
0.010	0000000010	2	0x002	**4.000**	1100110011	819	0x333
0.015	0000000011	3	0x003	**4.985**	1111111100	1020	0x3FC
0.020	0000000100	4	0x004	**4.990**	1111111101	1021	0x3FD
1.000	0011001100	204	0x0CC	**4.995**	1111111110	1022	0x3FE
2.000	0110011000	408	0x198	**5.000**	1111111111	1023	0x3FF

All you have to do is divide the analog input voltage by the converter's bit resolution, which in this case is 0.0048.

A. AND NOW IT'S YOUR TURN

Assuming that the V_{REF} reference voltage is 3 V and the converter has a resolution of 8 bits, complete the following table for each different analog input voltage.

BIT RESOLUTION							
INPUT VOLTAGE	OUTPUT			**OUTPUT VOLTAGE**	OUTPUT		
	BINARY	**DEC.**	**HEX.**		**BINARIO**	**DEC.**	**HEX.**
0.00				**1.50**			
0.01				**1.67**			
0.50				**1.85**			
0.65				**2.12**			
0.83				**2.57**			
0.95				**2.93**			
1.15				**3.00**			

.

THE FOLLOWING IS VERY IMPORTANT! The analog input voltage that you measure must NEVER exceed the V_{REF} reference voltage. For instance, if the V_{REF} = 5 V, the analog input voltage must also be 5 V at the most.

4. FUNCTIONS IN ARDUINO LANGUAGE

Using the ADC converter incorporated in the Arduino UNO couldn't be any easier. You need just two functions from the Arduino programming language.

- **The analogReference() function**

This function makes it possible to set the value of the reference voltage (VREF) the ADC converter circuit has to use to convert an analog voltage into its binary or decimal equivalent.

This is important because once you know what it is, you can calculate the resolution per bit as you did previously.

The VREF voltage must not exceed the voltage that powers the Arduino UNO controller (5 V). On the other hand, the analog input voltage you convert may not exceed the VREF.

Syntax:

analogReference(type);

type: Configures the V_{REF} used for analog input (i.e. the value used as the top of the input range). The options are:

DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards).

INTERNAL: This is a V_{REF} generated within the controller. In the case of Arduino UNO it's 1.1 V

EXTERNAL: The required V_{REF} is delivered on the controller's AREF pin.

Example:

analogReference(INTERNAL); *//The internal voltage of 1.1 V is used*

- **The analogRead() function**

This is the sentence you're going to use when you want to perform an analogical to digital conversion. Each time it's executed it takes a sample of the voltage on the analog pin or channel specified and then performs the conversion.

Syntax:

analogRead(pin);

pin: The pin number that corresponds to the analog channel you want to convert. In the case of Arduino UNO this could be from A0 to A5.

Example:

int V;

V = analogRead(A2); *//Performs the conversion of the voltage on A2*

- **The map() function**

Even though this function isn't expressly designed for ADC conversion it may well be of interest for what we're doing at the moment. It makes it possible to remap, reassign or redefine a value between a minimum and a maximum.

Let's see; you already know that the Arduino ADC converter has a resolution of 10 bits and can express a value between 0 and 1023 (2^{10}). On the other hand, the numbers Arduino works with come in packets of bytes (8 bits) or multiples of bytes (int, unsigned int, long and unsigned long). Sometimes it's better to use the result of a conversion (10 bits) as if it were a byte between 0 and 255 (8 bits) or an int (16 bits) and so on. Bytes or integers are much more standard formats.

Syntax:

map(value, fromLow, fromHigh, toLow, toHigh);

value: the number to map. It's usually an int (16 bits) or a long (32 bits). Float values can't be used.

fromLow: Expresses the lower bound of the value's current range.

fromHigh: Expresses the upper bound of the value's current range.

toLow: Expresses the lower bound of the value's target range.

toHigh: Expresses the upper bound of the value's target range.

Example:

```
int V;

V = analogRead(A2);          //Performs the conversion of the voltage on A2

V = map(V,0,1023,0,255);     //Reassigns the value it's read and converts in into a byte
```

The analog value read on pin A2 lies between 0 and 1023 and is stored in the "V" variable. This value is remapped to an equivalent between 0 and 255. According to Arduino this function is derived from the following mathematical calculation; we include it purely for interest's sake:

$$Resultado = \frac{(valor - min) * (Nmax - Nmin)}{(max - min) + Nmin}$$

5. ANALOG PERIPHERALS

There's a broad range of sensors on the market that can provide analog voltage between a minimum and maximum depending on the physical quantity in question. You can regard them as analog peripherals.

- ✓ **Analog Potentiometers**. Moving the shafts gives a variable analog voltage of between 0 and 5 V. They're connected to the A0 and A1 pins and you can regard them as analogical motion sensors.
- ✓ **Light sensor**. This is connected to the A2 pin. The voltage it generates depends on the amount of ambient light that strikes it.
- ✓ **Infrared Sensor**. This is non-visible infrared light (IR) reflective sensor. It's connected to the A3 pin and measures the IR light that strikes it.
- ✓ **Temperature Sensor**. This measures the room temperature and generates voltage in proportion. It's connected to the A4 pin.

A. POTENTIOMETERS

These are variable resistors and you can modify their value by moving the shaft or a control called the "wiper". I've no doubt that you've used one lots of times without realizing it; the control you use to adjust the volume of a radio, television or sound system are some examples.

They're the simplest and most economical analog peripherals you'll find. They come in all shapes and sizes.

It's got three terminals or pins. Pins 1 and 2 are at each end of the resistor. They represent its total value. Pin 3 is the wiper. There's a mechanism that moves it from one end of the resistor to the other thus varying its value. If you were to position it right at the mid-point of its travel between pins 1 and 3 you'd get half the resistance. If you put it between pins 3 and 2 you'd get the other half.

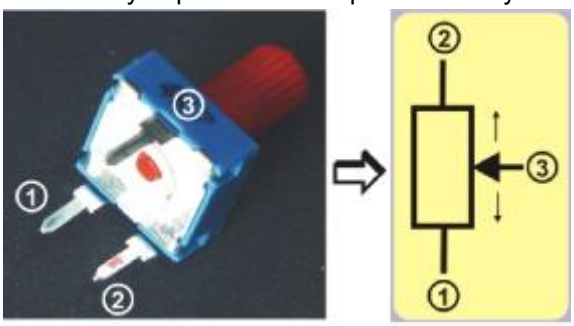

Figure 4

Now have a look at what the connections of both potentiometers. The ends are connected to 0 V and +5 V. If you move the wiper towards the bottom end, to the left that is, the analog voltage will decrease until it reaches 0 V. If you move it in the opposite direction the voltage will increase until it reaches a maximum of +5 V. The wipers of both potentiometers are connected to the analog input pins A0 and A1 on the controller.

B. PHOTO SENSORS

This is based on a small device called a “phototransistor”. Dispensing with lengthy technical explanations, we could say that this component increases or decreases the amount of current that passes through it based on the amount of light that strikes it. There are ones that are visible or ambient light sensitive and others that are infrared light (IR) sensitive. They come in a range of shapes and sizes.

As is shown in the figure on the right, we can use a torch to modulate the amount of light that strikes the sensor. This increases or decreases the electrical current, I, that flows through it. This current, I, circulates through a resistor, R, and produces a voltage, V, that also varies proportionally: V = R * I. To sum up: when the light changes, the intensity of the current, I, does too and this in turn modulates the voltage, V, between entre 0 and 5 V. This voltage is delivered to the A2 analog input of the Arduino.

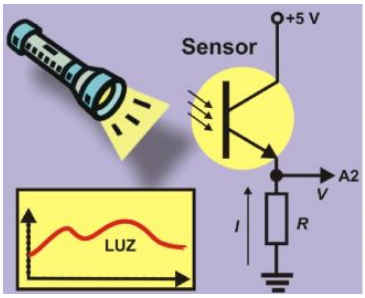

Figure 5

C. IR REFLECTIVE SENSORS

This is another kind of light sensor: it detects infrared light (IR) not visible to the human eye. It detects the amount of reflected light that strikes it. The sensor comprises two components. The LED light or emitter (E), which runs on +5 V, gives off a constant beam of infrared light. When this is reflected off an object, it strikes the photodiode or receiver (R) which absorbs the light. Depending on the amount of reflected light, the receiver increases or decreases the intensity, I, it feeds into the resistor, R, which in turn increases or decreases the analog voltage, V. Remember: V = R * I. To sum up: the closer the object, the greater the amount of reflected light. The intensity, I, is greater and therefore the voltage, V, is too. This voltage is applied to the A3 analog input pin on the Arduino UNO.

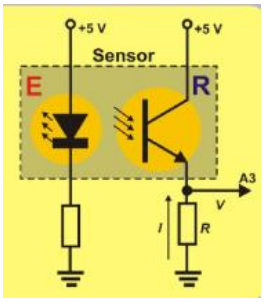

Figure 6

D. TEMPERATURE SENSOR

LM35 device, this component has three pins. Two of them are connected to the supply voltage of 0 and +5 V. The intensity, I, which circulates through the third pin, is directly proportional to the temperature. This intensity goes across the resistor, R, and this creates a voltage, V (V = R * I). It's connected to the A4 analog input pin.

According to the manufacturer of this device, the sensor's resolution is 10 m V/ºC. You can use the following equation based on the analog voltage (Va) to calculate the room temperature in degrees centigrade

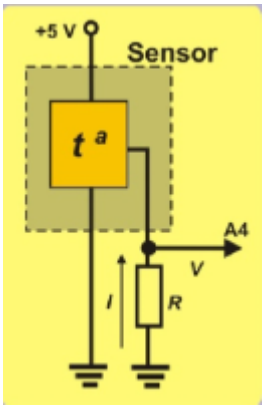

Figure 7

$$^{o}C = \frac{5 * Va * 100}{1024} = \frac{Va * 500}{1024}$$

6. PSEUDO ANALOG OUTPUT

The Arduino platform and digital inputs and outputs. You saw that Arduino UNO controller board has 14 pins that can be configured as digital inputs or outputs. In fact you've used most of them in exercises that you've been doing up until now.

You may remember that six of these pins can function as PWM signal output pins as well. Just for good measure, have a look at the figure on the right. I'm talking about pins 3, 5, 6, 9, 10 and 11, the ones preceded by the "~" sign; they're got an arrow pointing at them too.

You're going to use them as PWM signal outputs in this unit. You can get an idea of what the pins look like when they're connected to the development board by looking at the figure on the left.

PWM signal outputs numbers 6, 9, 10 and 11 are connected to the LEDs. Numbers 3 and 5, which are also PWM signal output pins, make it possible to connect up two servomotors or a DC motor; we'll connecting two servomotors in this unit.

A. WHAT ARE PWM SIGNALS?

The abbreviation "**PWM**" stands for "*Pulse Width Modulation*", an "*asymmetric*" periodic digital signal of "1"s and "0"s which is repeated constantly at the same frequency, **F** (there are X number of cycle repetitions in a second). This means that the time the signal is at level "1" may be completely different to the time the signal is at level "0".

Have a look at the signal in the Figure 8; we're going to work on it now. It's a digital signal with a period, **T**, of 2 mS. In other words, there are 500 identical cycles in a second (1000 mS/2). The frequency, **F**, is therefore 500 Hz (F=1/T).

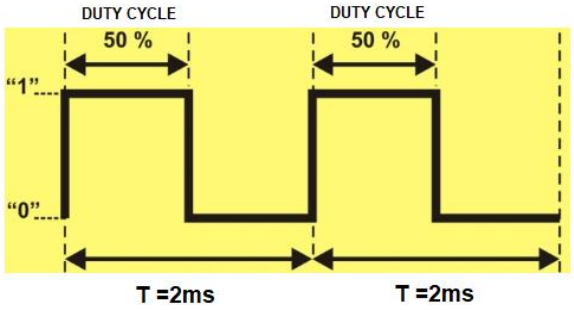

Figure 8

In this case the level "1" of each cycle lasts the same time as the level "0": 1 mS (if we add level "1" and level "0" they equal 2 mS of the period, T). This is called a "symmetrical" signal. The time the signal is on level "1" is called a "duty cycle". Its value is 50% (1 mS) of the total value of the period in the example. This is called a 50 % PWM signal.

Let's have a look at the four PWM signals in the figure below. They've all got the same period, **T**, of 2 mS, or a frequency, **F**, of 500 Hz, which actually amounts to the same thing. However the duration of the level "1", that is to say the duty cycle, is different in all of them.

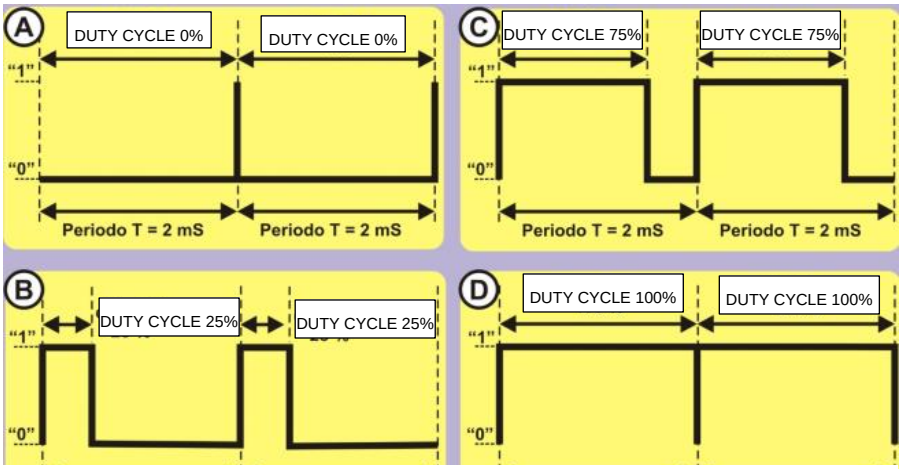

Figure 9

- ✓ **Signal A**. This is a PWM signal with a duty cycle of 0% of the period length. In other words, it remains at level “1” for 0 mS (0% * 2 / 100) and at level “0” for 100% of the remaining time, that is to say, 2 mS (100% * 2 / 100).
- ✓ **Signal B**. This is a PWM signal with a duty cycle of 25% of the period length. In other words, it remains at level “1” for 0.5 mS (25% * 2 / 100) and at level “0” for 75% of the remaining time, that is to say, 1.5 mS (75% * 2 / 100).
- ✓ **Signal C**. This is a PWM signal with a duty cycle of 75%. It remains at level “1” for 1.5 mS (75% * 2 / 100) and at level “0” for 25% of the remaining time, that is to say, 0.5 mS (25% * 2 / 100).
- ✓ **Signal D**. This is a PWM signal with a duty cycle of 100%. It remains at level “1” for 2 mS (100% * 2 / 100) and at level “0” for 0% of the remaining time, that is to say, 0 mS (0% * 2 / 100).

I’ve already mentioned that most modern controllers are capable of generating one or more PWM signals on some of their pins. In the case of Arduino UNO these pins are 3, 5, 6, 9, 10 and 11. When you use any of them as PWM outputs you must know the frequency, F, of the signals that Arduino generates. Have a look at the table:

PIN Nº	FREQUENCY, F	PERIOD, T
5 and 6	**980 Hz**	**1.02 mS or 1020 µS**
3, 9, 10 and 11	**490 Hz**	**2.04 mS or 2040 µS**

B. WHAT ARE THEY USED FOR?

I’m going to try and avoid complicated technical terms. We can control the time we remain on level “1” for each period with PWM signals; this is called the duty cycle. We can also use them to control and adjust the voltage we supply to certain output peripherals like light globes, LED lights, a motor, a servo and others.

Suppose a PWM signal like the one shown in the Figure 10 is connected to a LED light. The duty cycle is 50% so the LED is on for half the time and off for the other half. Believe it or not, the LED will only shine at half brightness for each period.

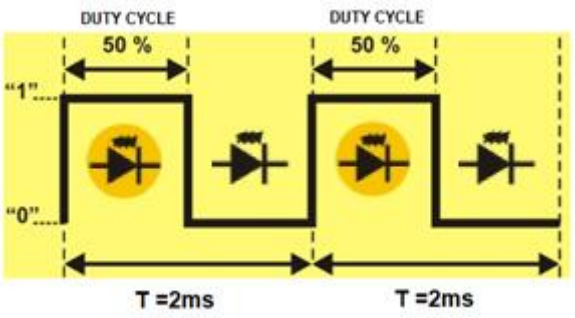

Figure 10

For the same reasons, if the duty cycle is 0%, the signal will remain on level “0” permanently. The LED won’t shine. If the duty cycle is 25% the LED will shine at a quarter of its brightness and if the duty cycle is 75% it will shine at three quarters of its full brightness. If the duty cycle is 100% the signal remains on level “1” permanently. The LED will shine at full brightness. This gives us a range of between 0% and 100% of the PWM signal to achieve the required brightness.

In the same way that you can adjust the brightness of a LED light or a lamp, you can also adjust the speed of a motor or the position of a servo’s shaft. The principle is the same and you’ll soon be able to experiment with it. You should also be aware that there a good number of peripherals that are controlled using PWM signals.

C. HOW ARE THEY GENERATED?

The controllers capable of generating PWM signals have a series of built-in electronic circuits like oscillators, timers, comparators, registers and others; they’re fairly complex. By synchronizing all these instruments and making them work together, we can generate one or more signals of this type. However, you need quite a lot of technical expertise to get all that working.

Fortunately Arduino makes things really easy for us. Forget about electrical circuits, how they work and how to use them. All you need is a single function from the programming language to generate a PWM signal; Arduino takes care of the rest.

- **The analogWrite() function**

This function makes it possible to adjust the length of the duty cycle of a PWM output signal.

Syntax:

analogWrite(pin, value);

pin: the pin to write to. This refers to the pin that’s going to generate the PWM signal. Remember that our Arduino UNO controller can use pin numbers 3, 5, 6, 9, 10 and 11.

value: determines the length of the duty cycle. It’s a byte number between 0 and 255. The value 0 represents 0% of the duty cycle, 127, a duty cycle of 50%, and 255 corresponds to a PWM signal with a duty cycle of 100%.

Example:

```
byte Voltage = 25;                  //States the % of the required voltage
int Cycle= Voltage*255/100;         //Calculates a duty cycle length of between 0 and 255
analogWrite(6,Cycle);               //Generates a PWM signal of 25% on pin 6
```

You don’t have to configure the pin that generates the PWM signal as an output; it’s already in the function. The pin generates the PWM output signal indefinitely until you execute the analogWrite() function again with a different value or the digitalRead() or digitalWrite() functions on the same pin. In all of these three cases the output of the PWM signal terminates.

7. FURTHER FUNCTIONS IN ARDUINO LANGUAGE

Even though the following functions don't have anything to do with PWM signals, we're going to take a look at them any way so you can broaden your knowledge of the language itself; they're also interesting to talk about.

They're related to the generation of random numbers, that is to say, numbers of any value within certain limits.

- **The random() function**

This function generates a random number between a minimum and a maximum. It returns a 32 bit signed integer value (long).

Syntax:

random(min, max);

min: establishes and includes the minimum value of the random number to be generated; it's optional. If the minimum value isn't indicated it's 0.

max: establishes but excludes the maximum value of the random number to be generated.

Example:

//Imitate the rolling of two dice

byte Die_1;

byte Die_2;

Die_1 = random(1,7); *//Generates a random number between 1 and 6*
Die_2 = random(1,7); *//Generates a random number between 1 and 6*

- **The randomSeed() function**

This is another system Arduino uses to generate random numbers. It consists of a sequence or long list of numbers and it's always the same. You can start generating the random numbers from anywhere in the list or sequence.

Syntax:

randomSeed(value);

value: this can be any 16 bit integer (int) or 32 bit integer (long). It's used as a "seed" to start off the sequence of random numbers at any point. The more random the "seed" is, the more random the sequence will be.

PRACTICE SECTION

8. EXAMPLE 1: ADC conversion

Here's the first exercise and it's about as easy as they come. All you have to do is read the analog value that you feed in using the potentiometer connected to the A0 pin. The conversion is performed each time you press the pushbutton connected to pin 4. The idea of this exercise is to reinforce the knowledge you've acquired in the previous units so serial communication and a range of formats are used to deliver the result of the conversion to the PC.

The really novel aspect of this example has been highlighted in the Figure 11. The analogRead(A0) function reads and converts the analog signal on the AO input into its equivalent in binary code and saves it in the "AN0" variable.

```
while(digitalRead(4));
delay(20);                  //Esperar a recibir un pulso desde D4

AN0=analogRead(A0);         //Realiza la conversión de A0 (potenciome
V=AN0*0.00488;              //Calcula la tensión equivalente

Serial.print(AN0);          //Transmite la medida en decimal
Serial.print("\t");         //Tabulación
Serial.print(AN0,BIN);      //Transmite la medida en binario
```

Figure 11

This value is converted into voltage by multiplying it by the constant, 0.0048 (the resolution in bits), and is saved in the "V" variable. Last of all the results are transmitted using serial communication in decimal, binary and hexadecimal formats and in voltage. The figure on the left shows a number of conversions. Start with the potentiometer all the way to the left (0 V) and move it gradually until you get all the way to the right (5 V).

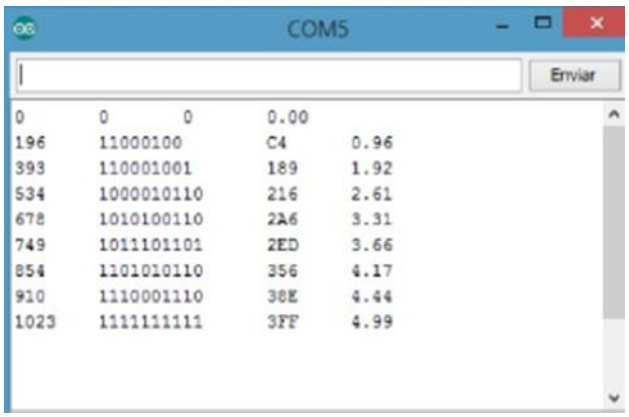

Figure 12

9. EXAMPLE 2: Thresholds

You can of course perform any sort of operation with the binary value of a conversion. You're going to perform two conversions in this exercise: one with the potentiometer connected to the A0 analog input and one with the potentiometer connected to the A1 analog input. If the result of the conversion on the A0 input is more than 4 V, the red light goes on. If the result on the A1 input is more than 2.5 V, the white light goes on. If the results are less than the voltages indicated the LED lights remain off.

When you upload the program make sure the red light goes on when you turn the shaft of the potentiometer almost all the way to the right; also make sure that you only have to turn the shaft of the potentiometer halfway to the left so that the white light goes on.

Remember that a potentiometer performs like a kind of joystick and you can detect what position it's in.

10. EXAMPLE 3: Analog comparator

Based on the previous exercise, we can create programs that compare two voltages or signals and say if one is greater than the other or if they're both the same. This is what we want you to do in this exercise.

We compare two analog voltages, V1 and V2, connected to the A0 and A1 pins; both come from the potentiometers. The output LEDs go on as shown in the table on the right.

IF...	LED
V1 = V2	Amber
V1 > V2	Red
V1 < V2	Green

11. EXAMPLE 4: Brightness control

You must have seen a system for controlling the brightness of the lighting in a room. You can brighten or dim the lights and create different atmospheres in the room using the control. And that's exactly what you're going to do in this exercise.

We use the analog voltage coming from one of the potentiometers and going to the A0 pin to generate a PWM signal that regulates the voltage going to the white LED connected to pin 6.

Have a look at the extract from the main body of the program (Figure 13). It's a pushover! The analogRead(A0) function reads the analog value on the A0 pin.

```
void loop()

  AN0=analogRead(A0);
  AN0=map(AN0,0,1023,0,255);

  analogWrite(6,AN0);
```

Figure 13

As you are already aware the result of the conversion may be anywhere between 0 and 1023. Using the map() function you can round off that value to another equivalent between 0 and 255, which is the parameter that the analogWrite() function needs to generate a PWM signal on output 6 (the white LED).

Upload the program and amaze your family and friends. You can dim or brighten the light by moving the potentiometer from left to right or vice versa.

12. EXAMPLE 5: Electric rudder

We're going to use the same principles as in the previous exercise. Just suppose you're out sailing. By moving the potentiometer, you move the servomotor's shaft which in turn moves the boat's rudder.

If you have a close look at the program you'll see that map() function rounds off the result of the conversion to a value between 0 and 180, the maximum number of degrees that the servomotor shaft can turn.

13. EXAMPLE 6: Photometer

We're going to work with the light sensor now and imitate a photometer or instrument for measuring ambient light. Every time we press the pushbutton connected to pin 4, the voltage coming from the light sensor connected to pin A2 is converted into its analog equivalent.

The program measures the analog value on the A2 input based on the light that strikes the sensor and then calculates the voltage. The results are transmitted using serial communication.

You can regard this exercise as an experimental program. You can make a number of measurements like the ones in the figure on the right. The first measurement you make is with the sensor in the dark and the last one in maximum brightness.

You can also look for a relationship between these measurements and the ones provided by a commercially manufactured instrument. Based on what you find out, you can create your own instrument for measuring light in lux or in lumens.

14. EXAMPLE 7: Lighting control

This exercise doesn't offer anything especially new but it does have practical applications: in public lighting systems for instance. I've no doubt it's used to light the streets of your town or city.

When it starts to get dark, the street lights go on. In this exercise, the light sensor connected to the A2 analog input measures the ambient light. There's sure to be a similar sensor in your street; you'll find it in some out of the way but secure location. The LED connected to pin 6 simulates the street light.

It's interesting to note that once again a function has been created: Measure_light(). This new function takes a series of ambient light samples at regular intervals and then calculates and returns the average of them all.

This is used to "filter" the analog signal coming from the light sensor; this avoids very small variations that make the street lights go on and off for no reason.

The main program calls our **Measure_light()** function and compares the average with the minimum value established for turning on the white LED.

- **And now it's your turn**

Have a think about this example: it's night time and the street lights are on. What happens if there's a storm and a streak of lightning strikes the sensor? The street lights will probably go off as our program will think that it's day time. They'll go on again later of course. This is a real example; things like this happen. But should we find a way to avoid it? Find a solution that avoids this happening and improve the program.

15. EXAMPLE 8: Measuring reflections

This exercise is very simple and doesn't offer anything especially new. The idea is to measure the reflected light detected by the infrared (IR) sensor connected to the A3 analog input. The result is transmitted using serial communication.

- **And now it's your turn**

When you record the program open up the serial port monitor as well so you can see the different measurements. As the converter has a resolution of 10 bits, the values will be between 0 and 1023 (2^{10}). Here are a few things you should check up on:

- ✓ If you don't put anything in front of the sensor, you'll notice that the reading is quite small. The IR light that's emitted disperses, doesn't bounce back and the sensor therefore only detects a small amount of reflected light.
- ✓ If you place something brightly coloured at some 10mm from the sensor, you'll notice that the reading increases considerably.

Bright colours are reflected better that IR light; they turn round and bounce back towards the sensor more.

- ✓ Try the same experiment but this time with a darker object. The reading on the sensor drops. Darker colours absorb IR light more and the sensor receives a smaller quantity of reflected light.
- ✓ You can also experiment with moving the object nearer to the sensor or further away. You'll notice that the reading changes when you move the object closer to the sensor or further away. The closer the object is to the sensor, the greater the amount of reflected light the sensor receives.

16. EXAMPLE 9: Detecting colours

This is a purely experimental exercise and there are ways of improving it. It's all about detecting the colour of an object. Use the values on the table in the previous exercise as a reference. We're going to try and distinguish between black and white in this exercise.

The program waits until you press the pushbutton connected to pin 12. Then the IR sensor connected to the A3 analog input performs the measurement. If this is less than 300 it transmits the message "BLACK". If the value is equal to 300 or greater it transmits the message "WHITE". These messages are transmitted using serial communication.

As you can see from the above figure you can create a black and white object with a simple piece of cardboard. You put it in front of the sensor at a distance of approximately 15 mm and you press D12 to make the measurement. The object's colour will appear on the serial port monitor.

17. EXAMPLE 10: Temperature sensors

This exercise is very similar to the previous ones. Here the idea is to measure and display the room temperature detected by the sensor.

The measurement is performed by pressing the pushbutton connected to the D4 pin. The result of the conversion (AN4), its equivalent in voltage (V = AN4 * 0.0048) and the room temperature in ºC (T = AN4 * 500 / 1024) are transmitted using serial communication.

18. EXAMPLE 11: Air conditioners

Here's another example with an obvious practical application. We're going to simulate a simple air-conditioning system. The figure below will give you a better idea.

When the temperature exceeds a value established in the "Max" variable the green LED light on pin 9 goes on; this is to simulate that the cooling system has just gone on. If the room temperature drops below a certain value established in the "Min" variable, the red LED light on pin 11 goes on; this is simulate that the heating system has just gone on. If the sensor detects a temperature between the "Max" and "Min" both the cooling and the heating systems go off. This is assumed to be the temperature comfort zone.

19. EXAMPLE 12: A PWM SIGNAL

This is a very simple exercise and a good opportunity to use the analogWrite() function. The idea is to generate a PWM signal on pin 6 which is connected to the white LED on the development board.

You have to state the percentage of the duty cycle or the required voltage in the “Power” variable. The “Cycle” variable calculates the value of the equivalent duty cycle which is between 0 and 255.

The setup() function is empty but it’s compulsory to include it.

The main body of the program is in the loop() function. It generates a PWM signal with the duty cycle you requested on pin 6 using analogWrite();.

Here’s an important detail: have a close look at the while(1); function. These loops continue infinitely until the expression inside the parenthesis () becomes false. Something must change the tested variable or the while loop will never exit. It forms an infinite loop. This means that the controller doesn’t execute any other function.

But the PWM signal is still there on pin 6 all the same. Why? Because the controller’s internal electronic circuits generate the PWM signals on pins 3, 5, 6, 9, 10 and 11. Once the analogWrite() function starts generating them they continue indefinitely until you give an order to the contrary. We refer to them as “Hardware” generated signals.

20. EXAMPLE 13: Optical effects

This example will clearly demonstrate the optical effect created with the brightness of a LED light by a PWM signal with a duty cycle that gradually increases and decreases.

The first for() loop increases the duty cycle from 0 to 255 in steps of 5. The LED’s brightness will increase from minimum to maximum.

The second for() loop decreases the duty cycle from 255 to 0 insteps of 5 too. The Led light’s brightness will decrease from maximum to minimum.

21. EXAMPLE 14: Manual regulation

This is really an excellent practical exercise. You’re going to manually regulate the brightness of the white LED connected to pin 6. You’re going to make the LED light brighter by increasing the PWM signal duty cycle by multiples of five using pushbutton 4. You can fade it by using pushbutton 7 to reduce the length of the duty cycle, once again by multiples of five.

Let me remind you that you can apply exactly the same regulation principles to an electric motor, for instance; you’ll be able to control its turning speed using the same principle.

22. EXAMPLE 15: Random lights

This exercise is a bit of a novelty item. Maybe it'll help you to decorate your Christmas tree, shop window, bedroom, garden or whatever you fancy. We're going to generate four simultaneous PWM signals on pins 6, 9, 10 and 11.

The random() function we studied in this unit will randomize the length of each PWM signal's duty cycle. This in turn randomizes the voltage received by the Led lights.

REFERENCES

BOOKS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

WEB SITES

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com