UNIT 7: INFRARED SENSORS

AIMS

You're going to study the basics of infrared light in this unit: how to generate it...and how to detect it too.

We use infrared light (IR) for lots of sensors and devices that detect objects, obstacles, the presence and movement of people and things or send data.

I'm sure you've seen lots of examples: sensors that start escalators working when they detect someone's presence, ones that open or shut the lift or garage doors, others that detect a motor's speed and the direction it turns in and ones that detect the temperature of bodies or objects etc.

Infrared light is used in a huge number of areas. We're not going to try and look at them all but we are going to use a couple of different IR devices to perform simple tasks and drive applications.

THEORY SECTION

- THE ELECTROMAGNETIC SPECTRUM
- INFRARED LIGHT
- HOW TO GENERATE IT AND HOW TO DETECT IT
- INFRARED SENSORS
  - Transmissive IR sensors
  - Reflective IR sensors
- ENCODING
  - "Our" protocol
  - "OurIR" Library
  - Other protocols

PRACTICE SECTION

- EXAMPLE 1: "Detecting" an object
- EXAMPLE 2: Counting objects

- EXAMPLE 3: Measuring an object
- EXAMPLE 4: Detecting the direction of movement
- EXAMPLE 5: Automatic Lighting
- EXAMPLE 6: Crowd control
- EXAMPLE 7: Reflective detectors
- EXAMPLE 8: Remote control, transmitter
- EXAMPLE 9: Remote control, receiver
- EXAMPLE 10: Remote adjustment, transmitter
- EXAMPLE 11: Remote adjustment, receiver
- EXAMPLE 12: Remote data transmitter
- EXAMPLE 13: Remote data receiver

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 4

1. THE ELECTROMAGNETIC SPECTRUM 4
2. INFRARED LIGHT 6
3. HOW TO GENERATE IT AND HOW TO DETECT IT 6
4. INFRARED SENSORS 8
   A. TRANSMISSIVE IR SENSORS 8
   B. REFLECTIVE IR SENSORS 9
5. ENCODING 11
   A. "OUR" PROTOCOL 11
   B. THE "OURIR" LIBRARY 13
   C. OTHER PROTOCOLS 17

PRACTICE SECTION 18

6. EXAMPLE 1: DETECTING" AN OBJECT 18
7. EXAMPLE 2: COUNTING OBJECTS 20
8. EXAMPLE 3: MEASURING AN OBJECT 21
9. EXAMPLE 4: DETECTING THE DIRECTION OF MOVEMENT 22
10. EXAMPLE 5: AUTOMATIC LIGHTING 24
11. EXAMPLE 6: CROWD CONTROL 24
12. EXAMPLE 7: REFLECTIVE DETECTORS 25
13. EXAMPLE 8: REMOTE CONTROL, TRANSMITTER 27
14. EXAMPLE 9: REMOTE CONTROL, RECEIVER 28
15. EXAMPLE 10: REMOTE ADJUSTMENT, TRANSMITTER 29
16. EXAMPLE 11: REMOTE ADJUSTMENT, RECEIVER 30
17. EXAMPLE 12: REMOTE DATA TRANSMITTER 31
18. EXAMPLE 13: REMOTE DATA RECEIVER 31

REFERENCES 32

THEORY SECTION

1. THE ELECTROMAGNETIC SPECTRUM

The electromagnetic spectrum is a collective term referring to the entire range and scope of frequencies of electromagnetic radiation and their respective, associated photon wavelengths.

Without going into the physics of the matter too much, we're going to start off by reviewing the parameters that define an electromagnetic signal or wave like the one in the Figure 1.

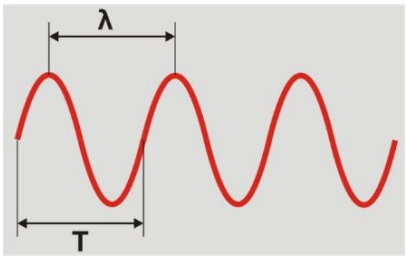


Figure 1

F = Frequency or number of cycles per second

T = Period or time that a cycle lasts: This is the inverse of the frequency T = 1 / F

For instance, a signal with a frequency of 1000 cycles or hertz per second (1 KHz) has a period of 0.001 seconds or 1 mS, okay?

λ = represents the distance in metres the wave travels in a period of time. This is new. As electromagnetic waves travel at the speed of light (300,000,000 m/s) the distance is calculated using the following formula:

$$\lambda = \frac{300.000.000}{F}$$

Thus, a signal with an **F** frequency of 1 KHz has a **T** period of 1 mS and a **λ** wave length of 300,000 metres.

Check and see: the greater the frequency the longer it is and vice versa.

Let's go back to the beginning. We're going to analyse the range, or spectrum, of all the known electromagnetic waves. Have a close look at the following Figure 2: it'll give you an idea of what this is all about.

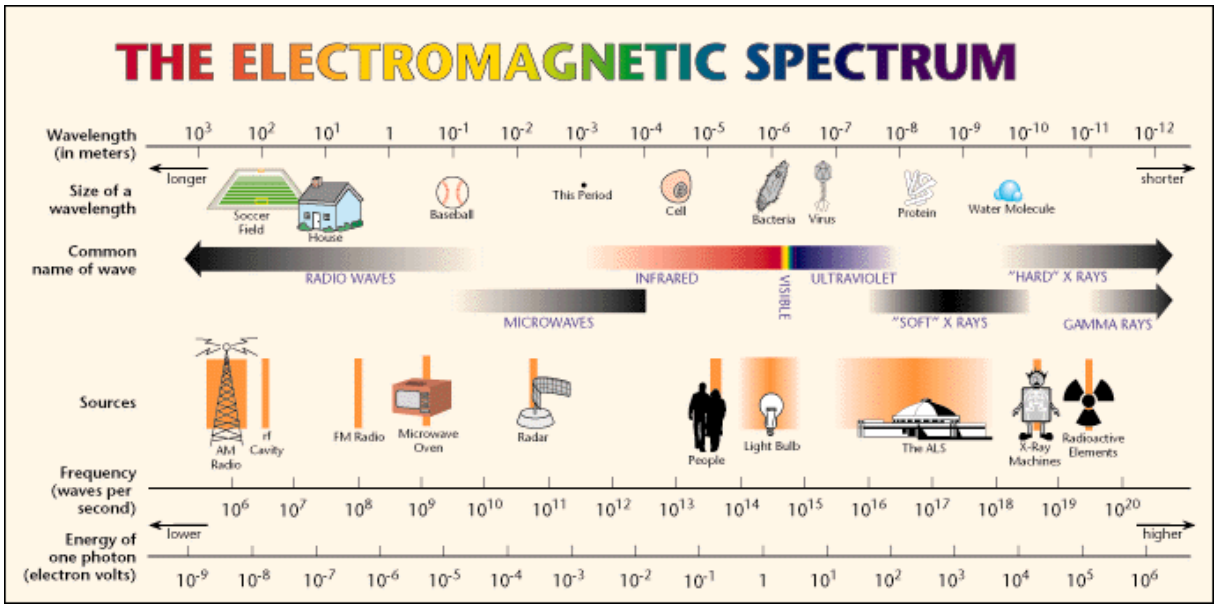


Figure 2: ELECTROMAGNETIC SPECTRUM

Radio waves: As you're aware, these are used for radio and television broadcasting. They can reach frequencies of hundreds or thousands of millions of hertz (109 Hz) and their wavelength oscillates between several thousand metres and less than a metre.

Remember: the greater the F frequency the longer the Λ length.

Microwaves: Amongst other things, these signals are used for mobile telephones and the well-known household appliance of the same name. They have frequencies of thousands of millions of hertz (1010 to 1012) and wavelengths of thousandths of a metre (10-3). To give you an idea, their size or length could be compared to a "." or full stop.

Infrared: These rays have an even higher frequency - around 1014 Hz - and a wavelength of around a millionth or a billionth of a metre (a micro or nanometre 10-6). They're about the size of a cell. Warm objects may give off this type of radiation.

Visible light: Rays with frequencies around 1015 belong to the radiation spectrum visible to the human eye. The colours between red and violet are found in this frequency range.

Ultraviolet: These rays belong to frequencies beyond the colour violet, which is why they're called ultraviolet.

X and Gamma rays: X and gamma rays are found beyond ultraviolet rays. Their frequencies exceed 10^{17} Hz and their lengths remain below 10^{-12} metres (billionths of a metre). They're radioactive.

2. INFRARED LIGHT

Infrared light is radiation released by any body with a temperature above 0º Kelvin (-273º C) or absolute zero. Its frequency is above microwaves but below visible light; that's why we can't see it. Its wavelength ranges from 0.7 µm (700 nm) to a 1000 µm which corresponds to frequencies from 4.28^{14} Hz to 3^{11} Hz.

Ours eyes can only detect a tiny part of the electromagnetic spectrum; how differently we'd perceive the world if we could see infrared light! Bearing in mind that any warm body can emit infrared light these are the kind of things we'd see Figure 3:

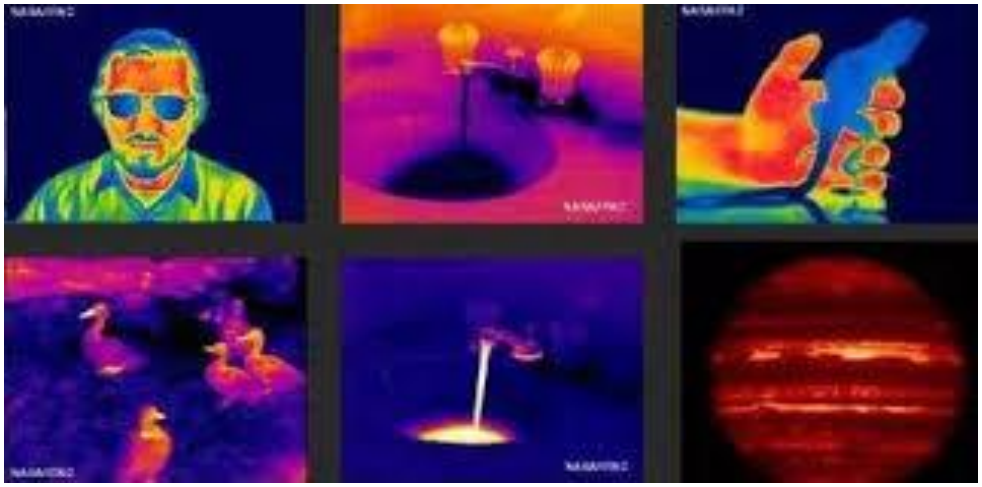


Figure 3

The greater the temperature the greater the amount of IR radiation. Warmer bodies are seen as lighter colours like yellow or white through an infrared camera and colder ones as darker.

We all use UR radiation on a daily basis: whenever you press a button on the television remote control; bar codes are read with IR light; for opening and closing doors; for detecting the presence or absence of a person; there are alarm systems that use IR light to detect the presence of an intruder; for transferring information between a computer and a peripheral; for devices that measure temperature, for night vision cameras and a host of other applications.

3. HOW TO GENERATE IT AND HOW TO DETECT IT

Depending on the chemical composition of the materials used to manufacture LED lights, when they are directly polarized they give off light of different colours: red, amber, green, blue and even white - you must have used them. What I'm trying to say is that they give off radiation that's within the range of visible light in the electromagnetic spectrum.

If the material used is gallium arsenide (GaAS) then the radiation given off is between 0.7 μm (700 nm) and 1,000 μm. This is within the range of IR radiation on the electromagnetic spectrum. To sum up, IR LEDs give off light but we don't see it.

As you may well imagine there are IR LED lights of different shapes and sizes for use in all kinds of situations. And the same applies to LEDs that give off visible light. There are even ones with different wavelengths but these always fall within the infrared spectrum.

As far as practical considerations are concerned an IR LED looks exactly the same as a conventional one and has very similar connections. As shown in the diagrams below (Figure 4) it has to be directly polarized so that it gives off an IR light that you can't see.

Figure 4

The anode has to be positive and the cathode negative. Both diagrams are the same. They also need an absorption resistance which is calculated in the same way as for a conventional LED that gives off visible light. If we apply +5 V to an IR LED where the Id = 20mA and the VAK= 1.5 V, we need a resistor of 180 Ω.

There are lots of LED lights that give off IR light and there are also lots of devices that detect it. The figure shows the symbol and a common diagram of the connections for an IR detector; it's called a "photoresistor". When it comes into contact with IR light, there's a directly proportional flow of intensity (I) from the transmitter to the collector; this causes a fall in VR voltage in the R resistor.

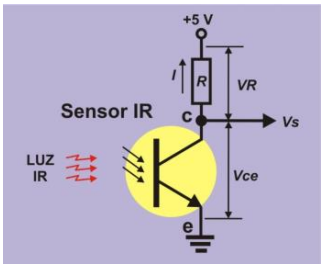


Figure 5: PHOTORESISTOR

In the circuit in the figure, the total voltage (+5V) is shared between the resistor (**VR**) and the photoresistor (**Vce**). The more light there is the greater the **I** intensity. This in turn causes a rise in the **VR** and a consequent fall in the **Vce**. The **Vs** output voltage falls to "0". If there is no IR light, the I intensity is non-existent, the **VR** disappears and the **Vce** rises to the maximum (+5V). The **Vs** output voltage goes to level "1". In other words, this is what happens:

1. If there's IR light present the Vs goes to level "0".

2. If there's no IR light present the Vs goes to level "1".

The above circuits are certainly very simple examples. You'll find a huge number of similar circuits both for generating IR light and for detecting its presence.

4. INFRARED SENSORS

An infrared sensor usually consists of a transmitter (T) and a detector (D) similar to the ones mentioned above. They can be divided into two broad groups: Transmissive IR sensors and IR reflective sensors.

A. TRANSMISSIVE IR SENSORS

The transmitter is usually opposite the detector. When an object comes between the two the transmission of IR light is interrupted and a signal is generated. You've almost certainly come across them in lifts or garage doors: when something or someone interrupts the transmission the door starts to open.

You'll find them in all sorts of shapes, sizes and ranges and they have a huge variety of applications.

You'll be using a very common transmissive sensor in the practice area; it's economical and simple - the H22A1.(Figure 6)

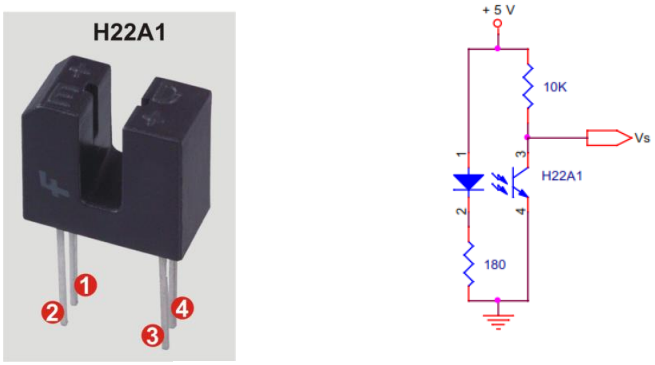


Figure 6: H22A1

The transmitter and the detector are contained in the same U-shaped capsule. On the left the diode transmitter corresponds to pins 1 (the anode) and 2 (the cathode). The phototransistor detector is on the right; pin 3 corresponds to the collector (c) and 4 to the transmitter (t).

When we insert an object in the slot it cuts the beam of infrared light between the transmitter and the detector and creates a particular logic level.

Here are the symbols and the most common circuit diagram you'll be using in the following exercises.

The IR emitting diode, between pins 1 and 2, is directly polarized with the 180 Ω absorption resistor. The phototransistor, between pins 3 and 4, receives the IR light. The Vs output voltage goes through the collector on pin 3. As long as it receives light it stays on level "0". When we insert an opaque object in the slot and cut the beam of light, the output signal goes to level "1".

B. REFLECTIVE IR SENSORS

Light reflection is another way of detecting objects. In this case the transmitter and the reflector are positioned next to each other and not opposite. When the IR light that's being transmitted strikes an object it's reflected and "bounces" back towards the detector.

As with transmissive sensors, there's a huge variety of reflective sensors on the market and they can be adapted for use in any kind of situation. They come in all shapes, sizes and ranges. Sometimes both the transmitter and the reflector are in the same capsule like the ones in the figure. Other times the transmitter and the reflector are separate. But they all work exactly the same way. The transmitter transmits the IR light and it's reflected off the object. This reflection is captured by the detector which transmits a digital signal.

You'll be using an LD274-3 IR emitting diode in the practice area. This IR light diode has a wavelength of 950 nm (1 nanometre = 1^{-9} metres).

You'll be using the Sharp GP1UX511QS or something similar as a detector. It consists of an IR light sensitive phototransistor with a wavelength of 950 nm. This feature of this detector is that it includes a 38 KHz filter. In other words it only detects light that's adjusted to that frequency. To put it yet another way, it only detects beams of IR light that go on and off at a rate of 38,000 times a second.

The filter makes the detector impervious to unwanted sources of IR radiation. Remember that all warm objects transmit IR radiation and the detector could easily malfunction. Nevertheless, it's unlikely that an object transmit IR light with a wavelength of 950 nm adjusted to a frequency of 38 Hz.

The symbols and the most common connections are in the figure opposite: OUT = the digital output when IR light adjusted to 38 Hz is detected; GND = the earth and VCC = positive voltage of +5V.

Figure 8: LD274-3

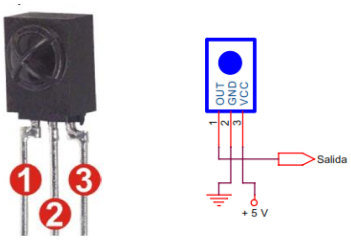


Figure 7: GP1UX511QS

You can see (Figure 9) what these beams are like and the logic levels the sensor generates each time it detects one.

The IR emitting LED light goes on and off at a frequency of 38 KHz for a period of 800 μm for instance. This beam or "burst" illuminates the detector which generates a logic level of "0". Then for another period of say 600 μm the transmission is interrupted. The detector generates a logic level of "1" on the output.

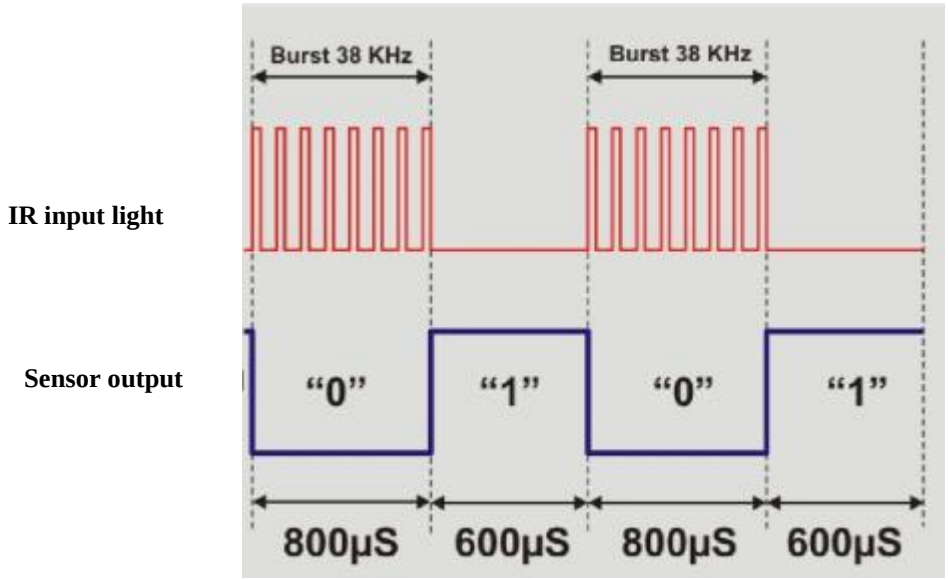


Figure 9

If the transmitter emits a burst of IR light and this is reflected off an object, the burst "bounces" back to the detector which generates a logic level of "0" at the output: object detected. If there's no object there won't be any reflection. The detector doesn't receive the "burst" and no object is detected.

On the other hand, imagine that the emitter is opposite the detector. You already know that when a “burst” is emitted the detector generates a logic level of “0”; if there’s no “burst” it generates a logic level of “1”. **You’re transmitting bits and that’s data!** That’s the principle of IR remote control.

5. ENCODING

A practical application of transmitting bits of data by IR light is remote control; this is something you use every day to switch the TV on and off and change the channels.

That’s right: the remote control contains an IR light emitter similar to or the same as the LD274-3 LED that you’re going to use. The television has a detector sensor similar to our Sharp model GP1UX511QS.

When you press a button on the handset it sends out a stream of pulses of infrared light. Each button has a unique binary code. The infrared light pulses encode it the receiver in the device recognizes the code, decodes it and converts it back to the original binary one. The television then responds accordingly by switching itself on or off, turning the volume up or down or changing channels etc.

A. “OUR” PROTOCOL

How do we encode or modulate the IR light to transmit or receive binary codes? The simplest and most obvious way is the one we’ve already explained. You know our detector generates a logic level of “0” every time it receives a beam or “burst” of IR light of 38 KHz. If it doesn’t receive the beam it generates a logic level of “1”. So for example if you want to transmit the 8 bit binary code 10110001, all you have to do is generate or not generate a series of beams with the LED light emitter for the unit of time *t*.

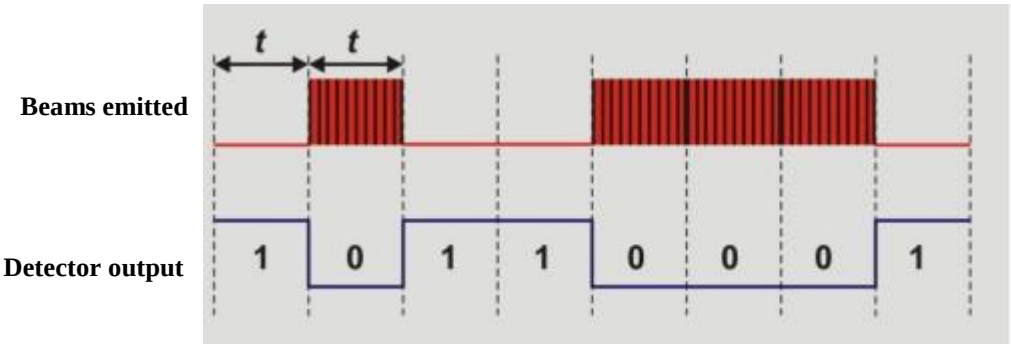


Figure 10

At fixed intervals (+) of 1 mS for instance, the 38 Hz beams of IR light are emitted or not. During this same 1 mS period of time read the detector pin: it’ll be on “0” if the beam is being received or “1” if it’s not.

Nevertheless, it's a bit too simple to work perfectly and there are a couple of problems with the system:

1. How do we distinguish one code from the next? This is important to remember when a code ends with the same bit as the next one begins with. Suppose a code ends with the bit "1" and the next code begins with "1" too.
2. Say two bits are received one after the other and they've both got the same value of "1" or "0". You read the detector output in the time periods *t* but if these get slightly out of synch, how do you distinguish between the previous bit and the next one?

We have to look for a sure way of transmitting and receiving the codes and being able to clearly identify where a code begins and ends and where each one of the bits that comprises the code begins and ends.

There are a number of different techniques or "protocols" that are capable of doing this. We're going to make up our own one and we're going to call it "Our" protocol.

Let's start off with some basic working premises:

1. These are 38 KHz beams, like the ones we've been talking about up till now. For your information, there are systems that work with beams of 32.75 KHz, 36.7 KHz, 36 KHz, 40 KHz etc...
2. The beam for the level "1" bit lasts for 1,200 µm and the one for the level "0" bit for 600 µm.
3. The beams are separated by fixed intervals of 600 µm.

The figure below contains a summary of the basic working premises:

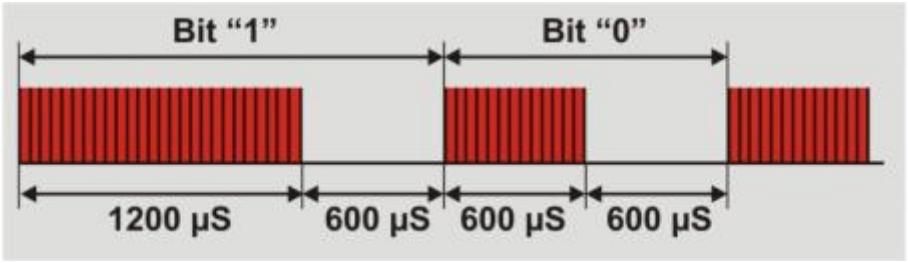


Figure 11

Now you can see that transmitting a "0" bit or a "1" bit isn't just a matter of sending out a beam of light. The beam has to last for a certain period of time (600 *µ*m or 1,200 *µ*m) and, in addition to this, it has to be followed by a "blank" space of 600 *µ*m for instance.

We should also make the following points clear:

- All the codes transmitted or received contain 8 bits which is equal to 1 byte.
- We start off by transmitting or receiving the least significant bit (LSB), the one on the extreme right in the byte.
- All the codes begin with an initial state that comprises a beam that lasts for 2,400 µm followed by a space of 600 µm. This bit, known as “start”, enables us to see where the new byte or code begins.

Having established all these working premises, we’re now going to have a look at what the transmission of the 10110001 code we used before would be like. Have a close look at the figure: bearing in mind that the least significant bit (LSB) is transmitted after the start bit, the code bit sequence would read like this from left to right: 10001101.

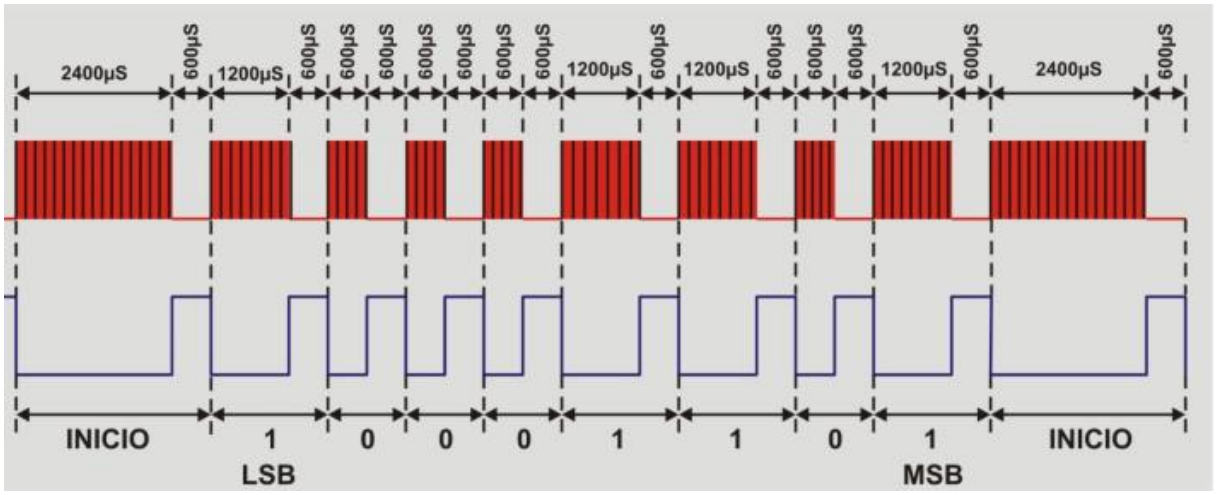


Figure 12

You’ll notice that that the basic idea is to control the time the detector output level stays on “0” for each bit. If it stays on “0” for 2400 *µ*S we’re dealing with the start bit which introduces a new byte or code. If, on the other hand, it stays on “0” for 1200 *µ*S we’re dealing with a “1” bit and if it stays on “0” for 600 *µ*S we’re dealing with a “0” bit. Between one bit and the next the detector output has to stay on level “1” for 600 *µ*S.

B. THE “OURIR” LIBRARY

We’ve created another library called “OurIr” with the functions we need to send and receive 8 bit data using the “OUR” protocol we’ve just made up.

The first thing you have to do is copy the OurIr folder included in the supplementary folder at c: Program files (x86)****Arduino****libraries****. Remember that a library must comprise at least two kinds of text files. These are the relevant ones in this case:

OurIr.h: this file contains the names of all the functions in the library.

OurIr.cpp: this file contains the code or instructions for executing the functions.

This library is very simple, easy to study and easy to use: it's only got three functions.

- **The burst() function**

This function generates a 38 KHz burst on the pin indicated and for the desired time.

Syntax

Burst(pin,time);

pin: the output pin for the 38 KHz burst

time: duration of the burst in microseconds

Example

burst(3, 1200); //38 KHz burst on pin 3 for 1200 µS

Have another look at the OurIr.cpp file and the code for this function from the library and you'll see it's very simple:

```
void OurIr::burst(byte Tx_Pin, int uS)
{
unsigned long T=micros();          //Sample time
tone(Tx_Pin,38000);                //38 KHz frequency output on the indicated pin (Burst)
while((micros()-T) < uS);          //Wait until the burst emission time elapses
noTone(Tx_Pin);                    //Burst OFF
}
```

We take a sample of the time with micros(). Using the old and well-known tone() function we generate a burst of 38 KHz on the indicated pin. This continues until the indicated time elapses. In the end the burst disappears with the noTone().

- **The IrTx() Function**:

This function sends out a piece of data on the pin indicated in accordance with "*Our*" protocol.

Syntax:

var.IrTx(pin, data);

pin: output pin

data: 8 bit data to be transmitted

Example:

```
#include <OurIr"        //Include the library
OurIr ITx               //Current variable
Tx.IrTx(3,0x83);        //Sends the 0x83 data on the D3 pin
```

You can also study the code used to create this function in the OurIr.cpp file. It's not difficult.

```
void OursIr::IrTx(byte Tx_Pin, byte data)
{
pinMode(Tx_Pin,OUTPUT);                 //Configures the pin as an output
burst(Tx_Pin, 2400);
delayMicroseconds(600);                 //Sends out the start bit
for(int c=0; c<=7; c++)                 //Sends out 8 bit data
{
if(bitRead(dato,c))                     //If the c bit is "1"...
burst(Tx_Pin, 1200);                    //1200 uS burst
else                                    //If it isn't...
burst(Tx_Pin, 600);                      //600 uS burst
delayMicroseconds(600);                 //600 uS pause
  }
}
```

After the pin indicated has been configured as an output, a burst is transmitted for 2400 µS and then there's a 600 µS pause. This is the start bit for Our protocol. The for() loop transmits the 8 bit data. bitRead() checks this data's c bit (from 0 to 7). If it's a "1" it sends out a 1200 µS burst; if it's a "0" it sends out a 600 µS burst, In both cases there'a pause of 600 µS.

- **The IrRx() Function**

This function returns the 8 bit data received on the indicated pin in accordance with the "Our" protocol.

Syntax

var.IrRx(pin);

pin: Input pin

Example:

```
#include <OursIr"        //It includes the library
OursIr Rx                //The most common variable
int Data=Rx.IrRx(2);     //It receives data from pin 2
```

It's not difficult to understand the code that makes up this function either. According to the "*Our*" protocol the idea is to detect the length of the level "0" pulses. Just remember that a pulse that lasts longer than 1000 µS corresponds to a "1" bit; if it lasts less than 650 µS it's because a "0" bit has been received.

```
byte OursIr::IrRx(byte Rx)
{
byte Data;                          //Value received
long Pulse;                         //Width of the pulse
while(!digitalRead(Rx));            //Waits until the start bit arrives (Max. 2400 uS)
for(byte c=0; c<=7; c++)            //Receives 8 bits
{
Pulse=pulseIn(Rx,LOW);              //Measures the length of pulse |_____| (1-0-1)
if(Pulse > 1100)                    //If it's longer than 1100 uS...
bitSet(Data,c);                     //a "1" bit has been received
if (Pulse < 650)                    // If it's shorter than 650 uS...
bitClear(Data,c);                   // a "0" bit has been received
}
return(Data);                       //Returns the value received
```

It starts off by waiting until the receiving pin is on level “1”; this is the end of the start bit. It waits to receive 8 bits from the for() loop. pulseIn() measures the length of the level “0” pulse received. It it’s longer than 1000 µS it’s because a “1” bit has been received. bitSet() writes a “1” on the c bit “Data” variable (from 0 to 7). It it’s shorter than 650 µS it’s because a “0” bit has been received. bitClear() writes a “0” on the c bit “Data” variable (from 0 to 7). It’s simple and original, don’t you think?

C. OTHER PROTOCOLS

You’ve probably realized that there are a huge variety of protocols for transmitting and receiving data using infrared light. In practice, almost all manufacturers of electrical appliances such as televisions, CD players, DVDs, VCRs etc have their own one.

We’re talking about well-known manufacturers like Phillips Nec, Nokia, Sharp, JVC and lots of others. Without going any further, the “*Our*” protocol which we just made up is almost exactly the same as the one Sony uses. Their protocol transmits and receives 12, 15 and 20 bit codes instead of 8 bit ones.

We’ve haven’t really got enough time to talk about all of these protocols but if you have a look in Internet you’ll find lots of information about them: how many bits they work with, how they modulate the IR light, how they encode the various binary codes. Here’s a very interesting website I suggest you have a look at:

http://www.sbprojects.com/knowledge/ir/index.php.

The “*Our*” protocol we just “made up” works pretty well and the most important thing is that it’s easy to understand. It meets all the needs of a course like this one.

However the protocols employed by the above mentioned companies in the household appliances they manufacture are much more reliable, robust...and complex. If you have a look on the Internet, you’ll find Arduino libraries that provide access to these commercial protocols.

Go to http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html .

Download the IRremote library: it contains functions for the remote transmission and reception of data and it’s multiprotocol as well. After you’ve downloaded it, install it. You can try some of the examples suggested using the components from this course. Give it a go – you’ll enjoy it and you’ll be learning at the same time. It’s all up to you.

PRACTICE SECTION

6. EXAMPLE 1: Detecting" an object

In this first example you're going to use a transmissive IR sensor to detect when an object passes between the emitter and the detector. Have a look at the circuit diagram (Figure 14) below and then a picture of what it looks like when it's been assembled (Figure 13).

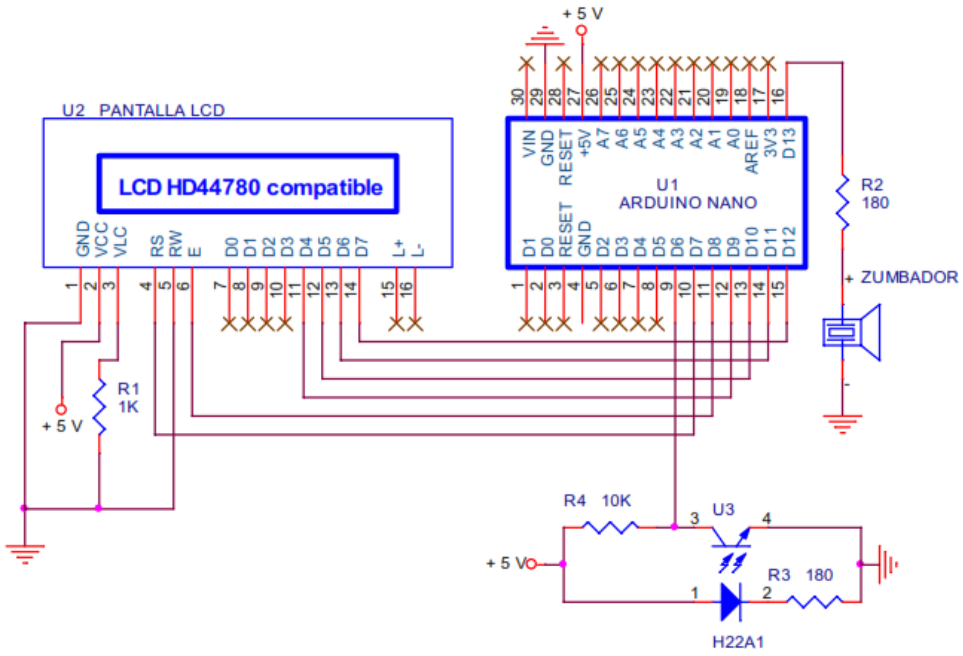


Figure 14

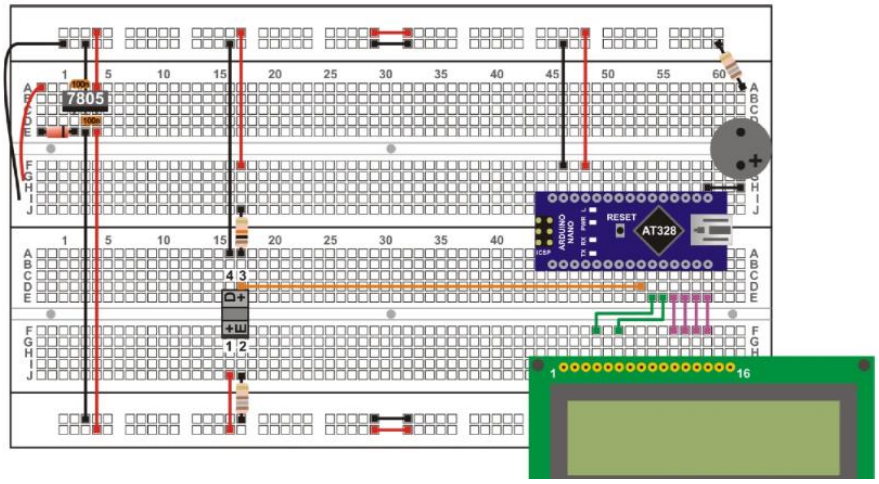


Figure 13

The program's really simple: the sensor is connected to the D6 input pin. When something passes between the sensor and the detector it breaks the beam of light and sets off the piezoelectric buzzer connected to the D13 output.

Now it's your turn

This example will help you to experiment with objects of different colours, textures and thicknesses. You'll be able to see to what point they break the beam of infrared light or whether they pass between the emitter and the detector without being detected. Use the table below to note down your observations.

OBJECT	Breaks beam	COMMENTS
White cardboard		
Green cardboard		
Red cardboard		
Black cardboard		
Blue cardboard		
Transparent plastic		
Opaque plastic		
Metal (a coin)		
A thin wooden panel		

7. EXAMPLE 2: Counting objects

Every time an object goes across the slot on the H22A1 sensor and breaks the beam of IR light, it produces a pulse or logic transition of 0-1-0. These pulses can be counted and you can tell how many objects have gone through the sensor. This is what you're going to do in this example. Use the same circuit diagram and assembly illustration as in the previous example. You'll be able to watch the objects being counted on the LCD screen.

The program is pretty simple and you've already done other things like it. This time, however, you're using an IR sensor. The first advantage is that when you use light to detect objects the signal doesn't suffer from the "bounce back" effect which you're already familiar with. There are no sheets of metal or springs that move. In short, there are no mechanical parts.

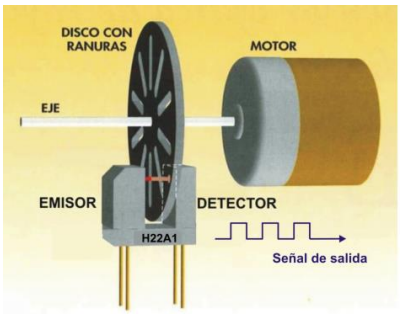
Now it's your turn

I don't think I need to talk about all the possible applications of a device that can count objects but I am going to mention an idea that may well not have occurred to you.

Do you know what an encoder is? It's a system that tells you how fast a motor is going, which way it's turning, how much the axle has moved or how many times it's gone round etc.

What's it made up of? Have a look at the figure: there's a disc with say ten slots and it's fitted around the axle of the motor. The disc is inserted in the H22A1 sensor. When the motor's turning over, each slot lets the IR light through and the sensor sends out a signal with a sequence of digital pulses.

What conclusions can you draw?



1. The disc in the example has ten slots so by the time you've counted ten pulses the wheel will have completed a 360º revolution.

2. If you count how many groups of ten pulses there are in a minute you'll be able to calculate the turning speed (r.p.m. or revolutions per minute).

3. Assuming the slots are evenly distributed on the disc, there's an arc of 36º (360º/10) between one slot and the next. If you count ten pulses then the axle will have turned 180º. So you know how far it's moved.

Funny, isn't it? Can you think of anything else you can do?

8. EXAMPLE 3: Measuring an object

Here's another very simple example that uses the same circuit diagram and assembly illustration as example 1. The idea is to measure how long an object takes to go through the middle of the sensor or how long it breaks the beam of IR light for.

You're going to use the pulseIn() function to do it. We studied this function in the previously. It's a function that measures the duration of a pulse or signal of logic level "1" or "0" in µS. It returns a full unsigned long value between 0 µ and 4,294,976,295 µS (2^{32}) which is equivalent to 70 minutes.

Syntax:

pulseIn(pin, value, timeout);

pin: the number of the pin which you want to read the pulse on. (*int*)

value: type of pulse to read: either HIGH or LOW"1" or "0". (*int*):

timeout: (optional): the number of microseconds to wait for the pulse to be completed. The function returns 0 if no complete pulse was received within the timeout. Default is 1.000.000 µS (1") or one second (*unsigned long*).

You already know that when an object passes through the slot it breaks the beam of light and the H22A1 sensor generates a logic level of "1". When there aren't any obstacles in the slot the sensor generates a logic level of "0". So what we have to do is measure the length of the level "1" when the object is going through the slot; it's displayed on the LCD screen.

Now it's your turn

When you record the example and see the result maybe it won't seem all that important. Got any ideas? Well, guess what? We can create a system capable of measuring the size and length of the object which goes through the slot.

All we have to do is make it go through the slot at a constant speed and use this famous formula: S = D / T (Speed = Distance / Time). And from that we get the following:

D = S x T (Distance = Speed x Time).

Imagine that the object goes through the slot at a constant speed of 0.1 m/s, or in other words, it takes 10 seconds to cover a distance of 1 metre. This is the speed at which it goes through the slot on the H22A1 sensor. Suppose a measurement of 300,000 µS appears on the LCD screen. This means that the pulse lasts 0.3 seconds which is how long the object took to go through the sensor.

If we apply the formula D = S x T it works out that the distance covered by the object is equal to 0.1 x 0.3 = 0.03 m or 3 cm. This distance is nothing less than the length or size of the object itself. What do you think about that?

9. EXAMPLE 4: Detecting the direction of movement

You can detect the direction an object or a person is moving in with the help of two sensors. Have a look at the figure. You've got two sensors, A and B, right next to each other.

The person's going to the left in the top picture. He goes through sensor A first and then B and generates the sequence of digital signals shown.

In Figure 15 the person's going to the right. He goes through sensor B first and then A and generates the sequence of digital signals shown.

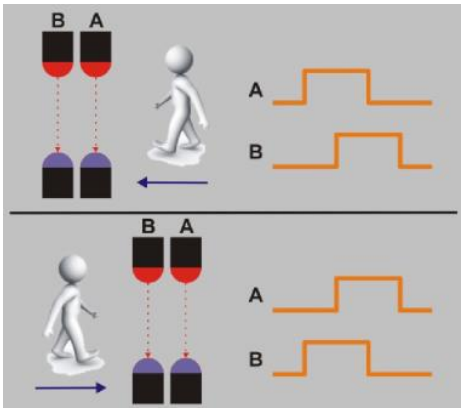


Figure 15

This is what you're going to do in this example: detect the sequence of signals that enable you to tell which direction the object is moving in. Here's the circuit diagram (Figure 17) and the assembly illustration (Figure 16):

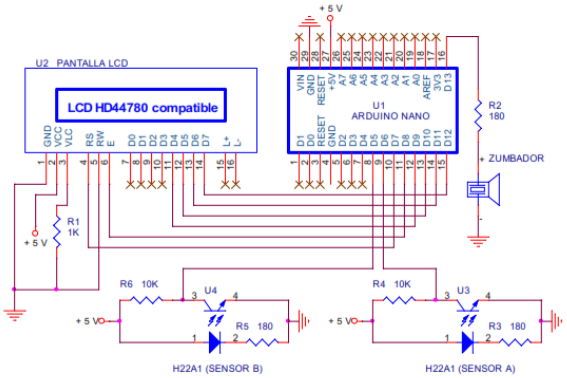


Figure 17

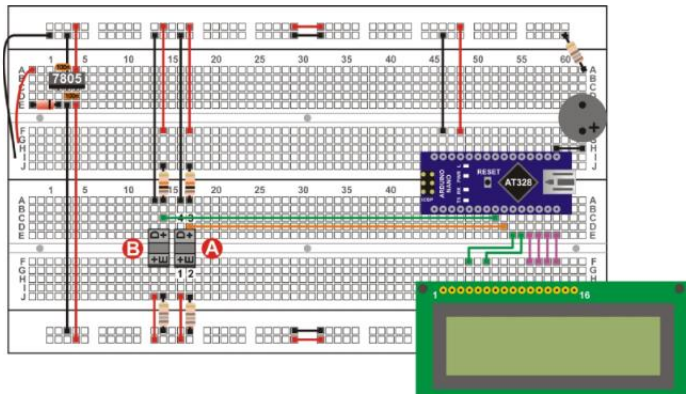


Figure 16

Sensor A is connected to the D6 input and it's on the right of sensor B which is connected to the D5 input. The Figure 18 will help you understand what the program does to detect the sequence of signals that are generated when an object moves to the left.

1. If an object is detected coming from the right (A)...
2. The program waits until it reaches the sensor on the left (B).
3. It waits until the object's finished going through the one on the right (A).
4. It waits until the object's finished going through the one on the left (B).

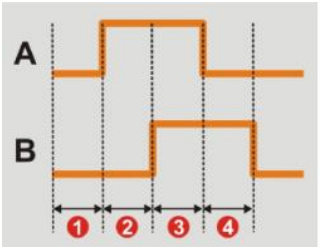


Figure 18

10. EXAMPLE 5: Automatic lighting

Imagine an exhibition room at an art gallery or museum. We're going to install an automatic lighting system. The idea is to detect whether or not there's a person in the room and when there's nobody in the room, the lights stay off. The system detects if the person's coming in or going out of the room.

Here's the circuit diagram. It's similar to the one in the last practice exercise. The A sensor is connected to the D6 input and the B sensor to the D5 input. In addition to this, a LED light connected to the D6 input simulates the lighting system. The LED light goes on when it detects human presence in the room.

The sequence you used in the previous example to detect the direction an object or a person was moving in will also tell you when someone is coming into the room or leaving it.

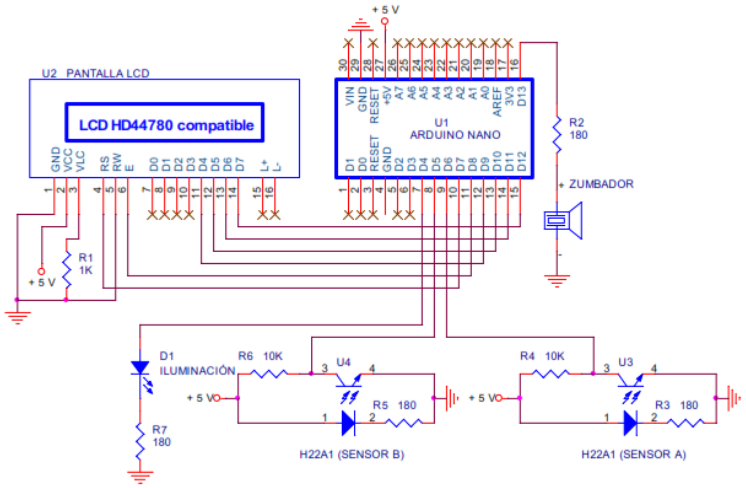


Figure 19

When you record the program, try simulating a couple of people coming in and going out of the room. Count them and make sure that whenever there's someone in the room the lighting system stays on.

11. EXAMPLE 6: Crowd control

Here's another very practical example. This time the aim is to control the capacity of an enclosed space like a bingo hall for example. The sytem's got a green light to indicate that more players can go in and a red one to indicate that the hall is full. The LCD screen shows how many free places there are available.

The circuit is very like the last one we see in Figure 19. There's a red LED that wasn't in the previous example; it's is controlled from the D3 output.

As in the two preceding examples the A and B IR transmissive sensors connected to the inputs detect which way people move: whether they enter or leave the bingo hall.

12. EXAMPLE 7: Reflective detectors

Here's another example that involves detecting objects and people. This time you're going to use a LD274-3 IR LED emitter and a Sharp GP1UX511QS IR detector.

First of all, as far as possible, try and prevent the IR light emitted by the LED LD274-3 dispersing. The more direct the light the better. To do this, you're going to "wrap up" the LED in a piece of black thermoretractible tubing. Have a look at the photo in the figure. You insert the LED in the tubing until 3 mm of the pins are covered and apply some heat – you could use a lighter for example. The idea is that just the end of the LED remains visible.

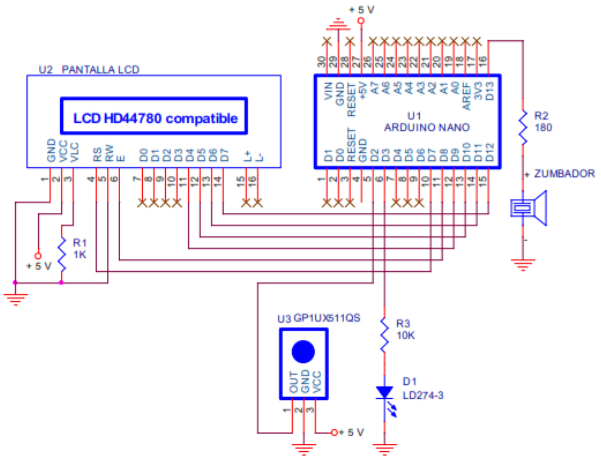


Figure 20

You can see the circuit diagram in Figure 20. The LED emitter is connected to the D3 output by a 10KΩ absorption resistor - in theory any way. Later on you'll be changing the resistor's value to study and experiment with different ranges. The detector is connected to the D2 pin. Every time it detects an object, the piezo buzzer connected to the D13 output plays a tone.

The LED emitter and the detector are placed right next to each other; this way the detector receives the IR light from the emitter when it reflects and "bounces off" an object. As has already mentioned in the theory area, the detector only detects modulated IR light at a frequency of 38 KHz.

Have a look at the program in the example. It contains a newly created function called serial.find().This function reads data from the serial buffer until the target string of given length is found. The function returns true if target string is found, false if it times out. After a 2 mS pause a 38 KHz signal is given off on the D3 output; this modulates the IR light that's given off at this frequency. We use the old and already well-known tone() function here. The micros()function controls the time the light is given off which is fixed at 1 mS. What this means is that at least every 2 mS a 38 KHz "burst" is given off for 1 mS. Have a look at the oscillogram in the Figure 21.

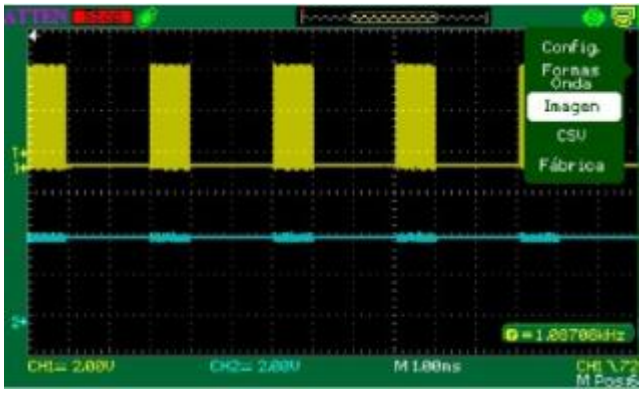


Figure 21

Whilst the “burst” is being given off (for 1 mS) the D2 pin is explored at the detector output connection. If a level “0” is detected this means that the detector has received a “rebound” from the “burst” of IR light modulated to 38 KHz. The serial.find() function returns a “*true*” value. You can see the relationship between the burst of light emitted (in yellow) and the detector output when it receives the reflected light (in blue).

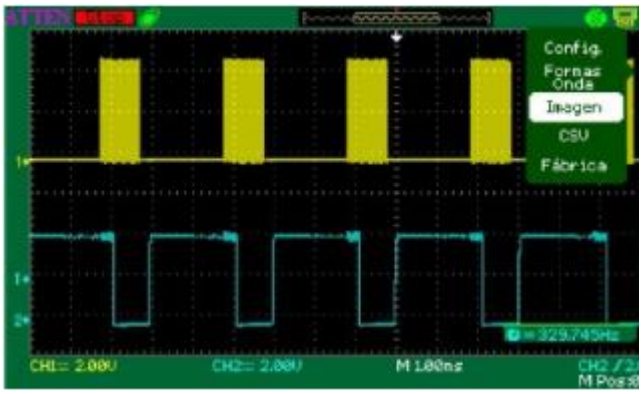


Figure 22

The main program is pretty basic really: the D13 pin, the one the piezo buzzer is connected to, goes on or stays off according to the “*true*” or “*false*” result the serial.find() function returns.

13. EXAMPLE 8: Remote control, transmitter

Do you remember Section 5 in the Theory Area? It suggested a fairly reliable protocol called “Our” that we could use to transmit any 8 bit binary code using IR light.

You’re going to build a remote control system but you’ll need two Arduino controllers to do it:

- ✓ One that functions as an IR transmitter. Binary codes are encoded and transmitted using IR light and the “Our” protocol based on the state of pushbuttons or sensors.
- ✓ The other controller acts as the IR receiver. It receives the IR signal with the “Our” protocol, decodes the code and reacts appropriately.

We’ll need to do a bit of DIY though. First solder an absorption resistor of between 100 and 180 Ω to the pin that corresponds to the anode of the LD274-3 light emitting diode. Have a look at Figure 23:

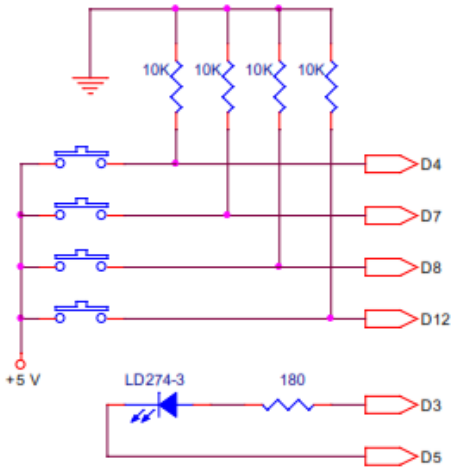


Figure 23

The pushbuttons connected to the D4, D7, D8 and D12 pins are for controlling the device. Now you’ve got a basic remote control with four digital pushbuttons and three analogue sensors as well. They’re all potentiometers: a light sensor, an IR reflective sensor and a temperature sensor. Now everything depends on the program you record on the Arduino UNO board to read the pushbuttons and the analogue sensors and then encode the information and send it.

The program in the example uses the “IrOur.h” library. Using the IrTx() function contained in the library it sends out a code every time a pushbutton is pressed.

Have a close look at it; it’s not very difficult. All it does is explore the state of the four pushbuttons and then send out the appropriate code each time we push one of them. If you connect an oscilloscope to

the D3 and D5 outputs you can see the bursts that do the encoding. The Figure 24 contains a diagram of the code 0xAA(10101010) being transmitted after someone pressed the D4 pushbutton.

Remember that after the start bit the least significant bit (LSB) is transmitted. If you read from left to right you'll see the code that was transmitted back to front: 01010101.

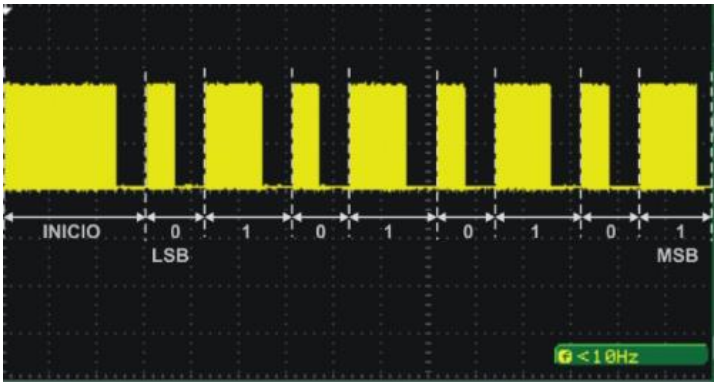


Figure 24

14. EXAMPLE 9: Remote control, receiver

Having made the IR remote transmitter you won't be completely happy until you've got a receiver to go with it. This is exactly what we're going to do in this next exercise. The Figure 25 below contains the circuit diagram

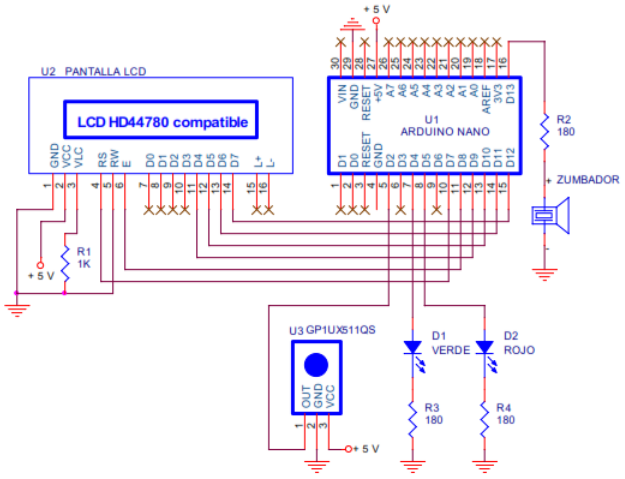


Figure 25

The input pin (D2 in the examples) shows a level "0" when it starts to receive information: while(digitalRead(Pin_Rx));. This is the beginning of the start bit in the "Our" protocol. Have a look back at section 5.

The receiver calls up the IrRx(Pin_Rx) function straight away, receives one of the orders from the transmitter in the previous example, decodes it and then carries out the order.

Transmitter		Receiver	
Pushbutton	**Order**	**Order**	**Action**
D4	0xAA	0xAA	**Green LED ON**
D7	0x55	0x55	**Green LED OFF**
D8	0x0F	0x0F	**Red LED ON**
D12	0xF=	0xF0	**Red LED OFF**

Once you've finished recording the program, try it out. Every time someone pushes a button on the transmitter , one of the LEDs in the receiver responds. According to the tests we've run in our laboratory the reception is good at a distance of 2 m. You can experiment by changing the value of the 180 Ω resistor that you connected to the anode of the IR emitting LED to increase its range. The minimum recommended value is around 100 Ω.

15. EXAMPLE 10: Remote adjustment, transmitter

You can of course transmit any kind of data using infrared light. This example looks at transmitting the converted analogical values supplied by the two potentiometers connected to inputs A0 and A1, you've already got everything you need, as was the case in example 8.

The potentiometers run on +5V. Their cursors are connected to the analogue inputs A0 and A1 of the Arduino board. These will vary between 0 and +5 V. The LD274-3 IR emitting LED is connected between the D3 and D5 outputs.

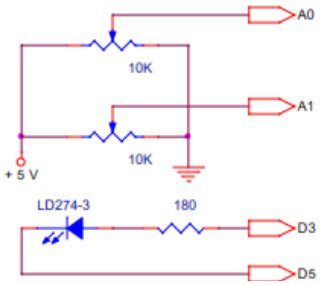


Figure 26

And there you go: now you've got your remote transmitter! The program reads the value of both analogue inputs and rounds them off to a value between 0 and 255. If one of the two inputs changes values compared to the previous reading, or in other words, if one of the two potentiometers moves, the program transmits a two bit message:

Potentiometer	***Channel***	***Message transmitted***
P1	A0	00 XX (XX is the result of the conversion)
P2	A1	01 00 XX (XX is the result of the conversion)]

The first byte identifies the potentiometer or the analogue channel: 00 for the A0 channel and 01 for the A1 channel. The second byte gives the value of the conversion of the channel.

16. EXAMPLE 11: Remote adjustment, receiver

Now it's time to make the receiver. This is very like example 9. We have to change the green and red LEDs. In the other example they were connected to the D4 and D5 pins; this time we connect them to the D5 and D6 pins respectively. These pins have PWM outputs. Have a look at the circuit Figure 27

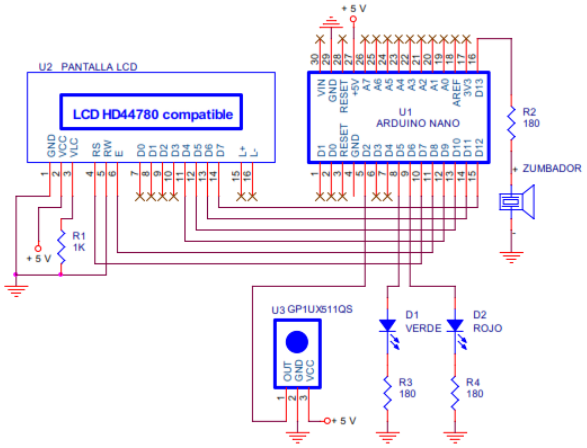


Figure 27

The program receives a two byte field by IR. The first byte establishes the channel number which is either 00 or 01. The second establishes a value between 0 and 255 and also the PWM duty cycle generated on the D5 (channel 0) and D6 (channel 1) pins. These are the pins you connected the green and red LEDs to.

In other words the aim is to regulate the brightness of the LED lights by remote control. And, in fact, example 5-10 shows that the transmission of readings of the potentiometers connected to the A0 and A1 channels of the transmitter. These readings are picked up by the receiver in example 5-11 and create a system for regulating the green and red LED lights by remote control.

Have a close look at both examples and then record them onto their respective controllers (transmitter and receiver) to make sure they work properly. The laboratory trials we did were successful. Occasionally we notice that the receiver stops working when the IR light is interrupted in the middle of a transmission. Maybe you can improve the system by adding start and end codes to each field to guarantee good reception; you could even improve the functions in the IrOur.h library. I'll leave it all up to you.

17. EXAMPLE 12: Remote data transmitter

We're going to finish unit 5 with another example of remote data transmission. In this case the transmitter receives characters typed into the PC one bit at a time and transmits them by IR using the "Our" protocol that you've been using up until now.

Just as in the previous examples, you'll need an Arduino controller or something similar to act as a transmitter. Connect an LD274-3 emitting diode with an absorption resistor to the controller between the D3 and D5 pins.

The program is very simple: all it does is check to see if a character has been received from the serial port. You can use the serial monitor included in the Arduino IDE or Integrated Development Environment.

The characters are received one bit at a time and extracted from the reception buffer and transmitted one by one using IR light.

18. EXAMPLE 13: Remote data receiver

This example complements the previous one. Just use the same assembly on the board as in previous examples.

The characters received by the IR detector are displayed on the LCD screen. What you've done is create a "bridge" between the PC and the screen to transfer characters between the two without using cables. The medium used to transfer the characters is, as you are already aware, IR light.

Here you are at the end of unit 7: if you've been able to do all the examples and they've all worked properly, congratulations!

Now, sit back and relax for a while and think about the thousands of ideas, applications and projects that arise as a result of what you've learnt. As has already been said on a number of occasions, the examples we suggest are purely didactic. The most important criterion for us is that the programs be as simple as possible even if they could use a little bit of improvement or fine-tuning and, in short, work better.

We don't pretend to provide definitive solutions for specific applications; that'd be impossible. What we do hope in all humility is that these examples serve as basic models that you can develop and use for your own ideas. Go on and give it a try.

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