



Instituto Tecnológico José Mario Molina Pasquel y Henríquez
Campus Chapala

Programación para Sistemas de Embebidos

Unidad 1. Elementos del lenguaje.

González Osuna Jennifer Cassandra

Minami Aguilera Jorge Satomi

Barajas Hernández Ismael

8°M

Diseño de una barra de leds multifunción

21 de febrero de 2022.

Contenido

Objetivo: 3

Justificación: 4

Marco Teórico: 5

Desarrollo: 15

Resultados: 37

Conclusiones: 52

Bibliografía: 54

Objetivo:

- a. **Objetivo del proyecto.**
Realizar una barra de leds de al menos 8 leds que se alimente a 12V y cuente con al menos 5 secuencias distintas de encendido de los leds, las cuales deberán ser manejadas con operadores de bits.
- b. **Objetivo académico.** Conoce las partes básicas de los lenguajes de programación para sistemas embebidos

Justificación:

En 1962, Nick Holonyak mientras colaboraba como científico asesor en un laboratorio de General Electric (Nueva York), inventó el primer LED rojo basado en semiconductores, aunque no se sabía que el diodo emitía fotones en el espectro infrarrojo, es decir, invisible al ojo humano. Cabe mencionar que antes de Holonyak, se considera a **Oleg Vladimírovich Lósev** (1903-1942) como el primero en desarrollar el LED (1927).

Hoy en día la tecnología LED ocupa ya un lugar importante en el mundo de la iluminación, ha sido una entrada explosiva y sin par, desplazando a su paso a otras tecnologías: volviéndolas obsoletas al reducir el área de su aplicación. Hoy creemos que 20W son demasiado para una lámpara cuando antes 40W eran muy poco para un foco incandescente.

Particularmente en la rama automotriz, la tecnología LED llegó a cambiar radicalmente la eficiencia de los vehículos. Lo que comenzó como una alternativa ahorrativa para las luces de freno en la parte trasera de algunos autos, se convirtió en 2003 en una revolución de la industria.

Algunos autos previos al modelo Nuyolari quattro de la empresa alemana, Audi en 2003 contaban ya con 24 LEDs en cada faro para las luces que se usan de día, pero ahora se implementaba por primera vez la iluminación LED en las luces intermitentes, las de cruce y las de carretera para las noches, esto se traduce en un total de 54 LEDs para todas las funciones de luz de los nuevos autos deportivos. Desde entonces, las más reconocidas casas automotrices han implementado esta tecnología que no solamente garantiza mayor potencia, sino que es más amable con el medio ambiente.

De esta manera nuestro proyecto es muy factible poder utilizarlo en la industria automotriz, pues si analizamos más afondo el funcionamiento de los LEDs en esta rama tenemos que, por un lado, tienen muy poca inercia lumínica, pues el tiempo que transcurre desde que comienzan a encenderse hasta que emiten todo su potencial de luz, es realmente muy bajo, de modo que no habrá variaciones de intensidad lumínica una vez que se enciendan.

Por otra parte, los LEDs consumen muy poca energía en relación a la luz que emiten, y esa es sin duda su mayor ventaja. Ya que son componentes eléctricos muy sencillos y la luz que emiten no proviene de un filamento incandescente o un arco eléctrico, la energía de los LEDs que se vuelve calor es realmente muy poca, y la gran mayoría de la electricidad que recorre el diodo se convierte en luz; esta es una de las ventajas que más ha favorecido la popularidad de la tecnología LED, pues los sistemas de luz son mucho más seguros al no generar calor, y pueden combinarse con una gran cantidad de materiales sin temor a ocasionar accidentes.

1. Marco Teórico:

a. Componentes:

LM7805

Es un regulador de tensión. que es capaz de modificar una señal de tensión que obtiene a su entrada y ofrecer una señal diferente de voltaje a su salida. En esa salida, el voltaje suele ser inferior y con unas características determinadas que se requieren para evitar riesgos o para el que el circuito al que alimenta funcione de forma adecuada, si es sensible a variaciones de voltaje, se muestra en la figura 1.1.

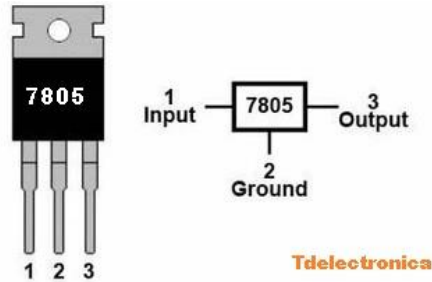


Figura 1.1 Regulador LM7805 (TD Electrónica, 2020).

Fusible

El fusible es un componente de instalaciones eléctricas que se interrumpe o funde cuando la corriente resulta excesiva, el componente se muestra en la figura 1.2.



Figura 1.2 Fusible (Euro Electrónica, 2018).

Los fusibles están compuestos por una lámina o un filamento hecho de una aleación o de un metal que se caracteriza por presentar un punto de fusión bajo. Este elemento está ubicado en un punto estratégico de la instalación eléctrica para que se funda si la intensidad de la corriente supera un cierto valor. Así, el fusible interrumpe la corriente y salvaguarda la integridad de los conductores minimizando el riesgo de incendio y/o avería.

TIP41C

El Transistor TIP41C es un transistor de potencia para bajas frecuencias. Tipo NPN está fabricado con silicio se puede ver en la figura 1.3.

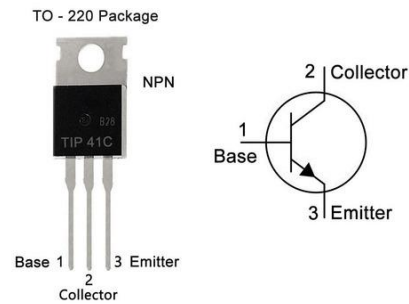


Figura 1.3 Tip 41C (Electro Store, 2015).

El TIP41C están diseñado para aplicaciones de potencia y manejar bajas frecuencias, por lo general se utiliza en fuentes de alimentación y como amplificador de audio de baja potencia.

Resistencia

Una resistencia es un elemento pasivo de un circuito eléctrico. Generalmente, una resistencia cualquiera provoca una restricción al paso de la corriente, limitándola y, específicamente, regulándola. Claro está, una resistencia combinada con otros elementos como condensadores (capacitores), bobinas, diodos... forman complejos circuitos con funciones concretas. Una resistencia se puede definir como cualquier medio material que limita el paso de la corriente eléctrica. A continuación, se muestran en la figura 1.4 los diferentes tamaños de las resistencias por potencia.

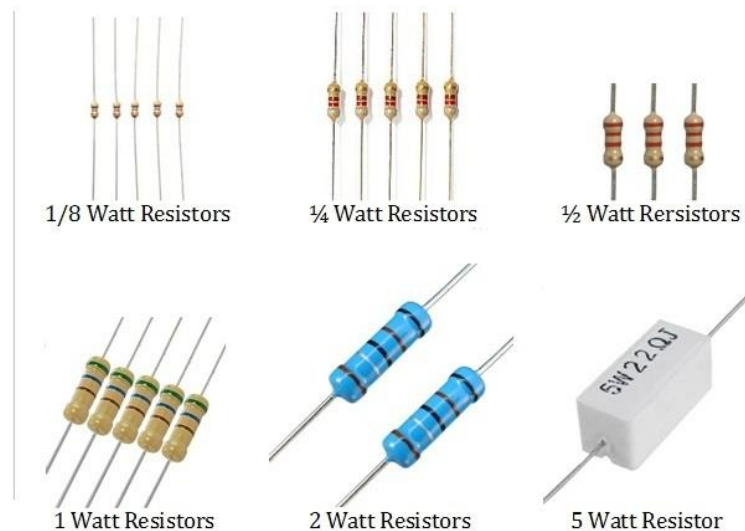


Figura 1.4. Tipos de resistencia según su potencia (Lugo, 2017).

Las resistencias se clasifican según su código de colores, mediante el cual es posible conocer el valor de la resistencia con tan solo mirarla de cerca. Es usual encontrar resistencias con 4 o 5 bandas de colores. Así como también varía su tamaño dependiendo su potencia.

Led de Potencia

Los diodos emisores de luz (LEDs) son elementos de estado sólido (semiconductores) que emiten energía luminosa al aplicar directamente energía eléctrica, los cuales, dependiendo de la aplicación pueden ser de baja o alta potencia. Los LEDs de alta potencia son diseños más completos que incluyen diversas alternativas de ópticas de control del flujo luminoso y se fabrican en potencias mayores a 1 W.



Figura 1.5. Led de Potencia 1W (Siled, 2016).

Características

Voltaje Operación: 3.2VDC ~ 3.4VDC

Potencia: 1W

Angulo de Luz: 120°

Vida útil: 10.000 HORAS

Este tipo de LEDs se utilizan principalmente en aplicaciones arquitectónicas de iluminación en exteriores e iluminación para calle, permitiendo tener más posibilidades de diseño y efectos de color.

Oscilador de Cristal

El oscilador de cristal es un componente electrónico capaz de generar una corriente eléctrica con una frecuencia muy precisa, esta frecuencia puede ser utilizada como reloj en una placa electrónica.



Figura 1.6 Oscilador de Cristal (Cesar Cinjordiz, 2018).

El símbolo más utilizado para representar un oscilador de cristal en un esquema electrónico es:

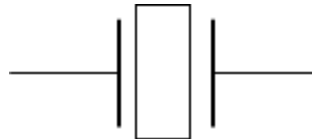


Figura 1.7 Símbolo del Oscilador de Cristal (Cesar Cinjordiz, 2018).

Capacitor

El condensador eléctrico o capacitor es un componente pasivo como los resistores, pero, que tienen la característica de almacenar energía en forma de campo eléctrico. Este campo es el resultado de una separación de la carga eléctrica. Está formado por un par de superficies conductoras, generalmente de láminas o placas las cuales están separadas por un material dieléctrico o por el vacío. Las placas sometidas a un diferencial de potencial adquieren una determinada carga eléctrica (positiva en una de ellas y negativa en la otra), siendo nula la variación de carga total. Un condensador es un dispositivo de dos terminales y puede tener polaridad en sus terminales.



Figura 1.8 Capacitores Cerámicos y Electrolíticos (HeTPro, 2016).

AT MEGA 328P-PU

El Microcontrolador ATMEGA328P cuenta con 8 bits de alto desempeño combina memoria flash ISP de 32 kB con capacidades de lectura mientras realiza escritura, RAM de 2 kB, 23 líneas de E/S, con convertidor A/D de 6 canales y 10 bits, temporizador de vigilancia (watchdog) programable con oscilador interno y cinco modos de ahorro de energía seleccionables por software. Un remplazo alternativo común al ATmega328 es el ATmega328P.

El Microcontrolador ATMEGA328P se puede ocupar para diferentes proyectos de electrónica, robótica y mecatrónica.

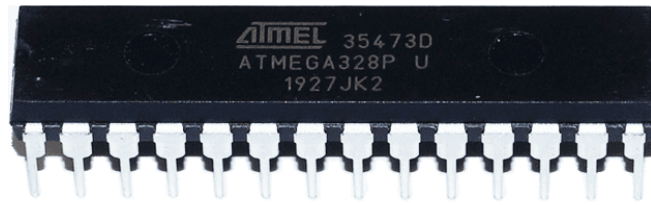


Figura 1.9. ATMEGA 328P PU

ESPECIFICACIÓN Y CARACTERÍSTICAS

- Familia: AVR ATmega
- CPU: 8-bit AVR
- Formato DIP
- Voltaje de alimentación: 1.8V a 5.5V
- Máxima frecuencia de funcionamiento: 20 Mhz
- Comunicación: I2C, SPI, UART
- Temperatura: -40°C ~ 85°C
- Pines: 28
- I/O: 23
- Tamaño de memoria del programa: 32 kB
- Memoria FLASH: 32KB
- RAM: 2 kB
- ROM: 1 kB
- EEPROM: 1024 bytes
- ADC: 6 canales

b. Programación:

C++

El lenguaje de programación c++ fue el lenguaje que ayudó a potenciar la *programación orientada a objetos*.

Las características principales del lenguaje de programación c++ son las siguientes.

- ✚ Sintaxis heredada del lenguaje C.
- ✚ Tiene un standard ISO, conocido como ANSI-C++. La última revisión fue en el 2011.
- ✚ Lenguaje fuertemente tipado. El programador debe saber cómo hacer y declarar el código para que funcione.
- ✚ Programación orientada a objetos, lo que comúnmente se puede encontrar por POO.
- ✚ *Abstracción.*
- ✚ *Encapsulado.*
- ✚ *Herencia.*
- ✚ *Polimorfismo.*
- ✚ Sobrecarga de operadores.
- ✚ Soporta expresiones Lambda, también llamadas funciones anónimas.
- ✚ Control de excepciones.

Código abierto y hardware extensible: El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo.

Variable

Una variable es un espacio en la memoria, en el cual el programador asigna un valor determinado por el tipo de dato que el lenguaje de programación va soportar, para declararla es necesario saber qué tipo de dato la contiene.

Es representada por un nombre que es asignado por el programador, y se nombra justo después de su tipo de dato.

Los siguientes son los tipos de datos esenciales:

- **int**; es numérico sirve para números enteros sin punto decimal (1,2,3,4..)
- **float**; es numérico y sirve para números con parte fraccionaria, es decir con punto decimal (1.2, 1.3, 1.4,)
- **char**; es de tipo carácter y cadena de caracteres, es decir que puedes asignar desde una letra, símbolo o número hasta una palabra o serie de caracteres (a, 2, !, &, hola, etc.)
- **bool**; es de tipo verdadero o falso, este tipo de dato mostrara en pantalla un "1" en caso de ser *true* o "0" en caso de ser *false*.

Parámetro:

Los parámetros son variables locales a los que se les asigna un valor antes de comenzar la ejecución del cuerpo de una función. Su ámbito de validez, por tanto, es el propio cuerpo de la función. El mecanismo de paso de parámetros a las funciones es fundamental para comprender el comportamiento de los programas en C.

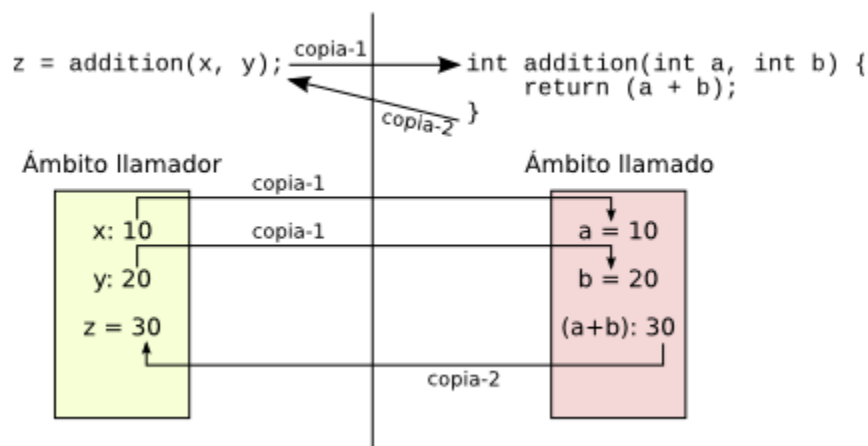


Figura 1.10 Parámetros (Félix García, 2002).

El paso de parámetros y la devolución de resultado en las funciones C se realiza **por valor**, es decir, **copiando** los valores entre los dos ámbitos.

Bitwise (NOT)

El operador NOT bit a bit (~) invierte los bits de su operando. Al igual que otros operadores bit a bit, convierte el operando en un entero con signo de 32 bits.

Syntax: ~a

El operando se convierte en un entero con signo de 32 bits y se expresa como una serie de bits (ceros y unos). Los números con más de 32 bits obtienen sus bits más significativos descartados. Por ejemplo, el siguiente número entero, con más de 32 bits, se convertirá en un número entero de 32 bits con signo:

Before: 111001101111101000000000000000110000000000001

After: 101000000000000000110000000000001

Cada bit en el operando se invierte en el resultado.

La tabla de verdad para la operación NOT se muestra en la tabla 1.1.:

a	NOT a
0	1
1	0

Tabla 1.1 Tabla de verdad NOT (MDN, 2018).

Bitwise (OR)

El operador OR bit a bit (|) devuelve un 1 en cada posición de bit para la cual los bits correspondientes de uno o ambos operandos son 1.

Syntax = a | b

Los operandos se convierten en enteros de 32 bits y se expresan mediante una serie de bits (ceros y unos). Los números con más de 32 bits obtienen sus bits más significativos descartados. Por ejemplo, el siguiente entero con más de 32 bits se convertirá en un entero de 32 bits:

Before: 111001101111101000000000000000110000000000001

After: 101000000000000000110000000000001

Cada bit del primer operando se empareja con el bit correspondiente del segundo operando: primer bit con el primer bit, segundo bit con el segundo bit, y así sucesivamente.

El operador se aplica a cada par de bits y el resultado se construye bit a bit.

La tabla de verdad de la operación OR se muestra en la tabla 1.2:

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 1.2 Tabla de verdad OR (MDN, 2018).

PORTD y PORTB

En principio, es importante recordar que los puertos de un microcontrolador de 8 bits tienen esa misma cantidad de entradas/salidas, o sea ocho líneas. Esto nos haría pensar que el ATmega328P, que posee tres puertos (B, C y D), dispone de $3 \times 8 = 24$ líneas de entrada/salidas disponibles. Sin embargo, utilizado en un Arduino no es así, como veremos.

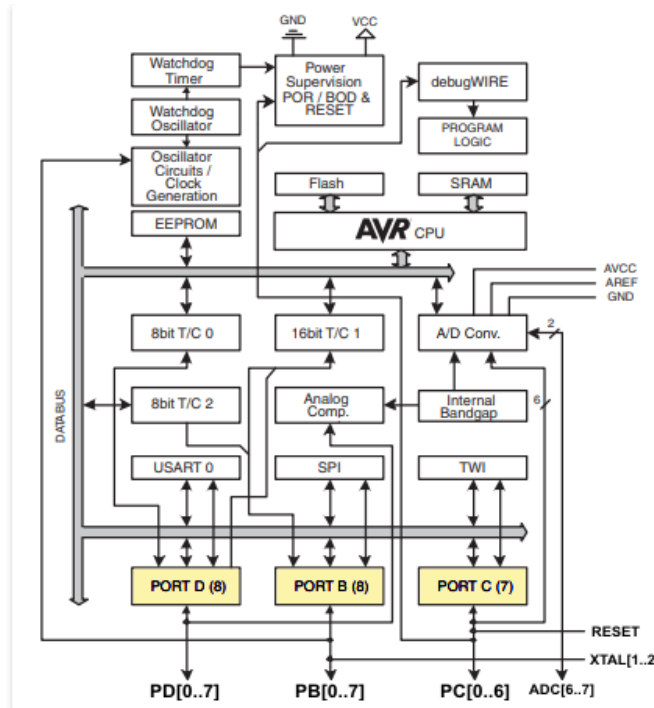


Figura 1.11 Puertos. (Daniel Weller and Sharat Chikkerur, 2010).

El **PORTB** (puerto B) tiene ocupadas dos líneas de entrada/salida que se utilizan para conectar el cristal oscilador. Estos pines, el **PORTB** bit-6 y **PORTB** bit-7, pueden quedar libres si se configura al chip para utilizar el oscilador interno, pero esta opción no podemos utilizarla en el Arduino debido a que ya tiene su sistema basado en la velocidad de cristal de 16 MHz, además de que el cristal está soldado a esos pines en el circuito de la placa.

Dos bits del **PORTD**, el **PORTD** bit-0 y el **PORTD** bit-1, se utilizan durante la programación del Arduino, ya que están conectados a la interfaz **USB**, además de ser los pines **TX** y **RX** utilizados para la *comunicación serie*. Estos pines se pueden utilizar para comunicación serie asincrónica hacia el exterior, y también como entradas o salidas cuando no se está grabando un programa. Pero no deben tener conexiones instaladas mientras se programa el Arduino.

En consecuencia, no se llega a disponer de la cantidad de 24 entradas/salidas que ofrecerían tres puertos de 8 bits.

2. Desarrollo:

a. 2.1 Materiales:

En la parte de electrónica, primeramente, se investigó un poco de los leds de potencia para decidir cual se iba a utilizar. Los candidatos ganadores fueron los leds de 3 Watts a 3.3V ya que es de los más utilizados en la industria automotriz como se muestra en la figura



Figura 2.1.1 LED 3 Watts a 3.3V

Luego se comenzaron a hacer pruebas en proteus para ir armando el circuito y en base a eso se hizo el circuito en proto para y probar con algunos componentes electrónicos. Primeramente, se comenzó a hacer pruebas con un UNL2003 para poder alimentar únicamente 1 led de potencia, así que se realizó la compra de estos por vía internet. Y una vez que los tuvimos en mano, se comenzó con el circuito.

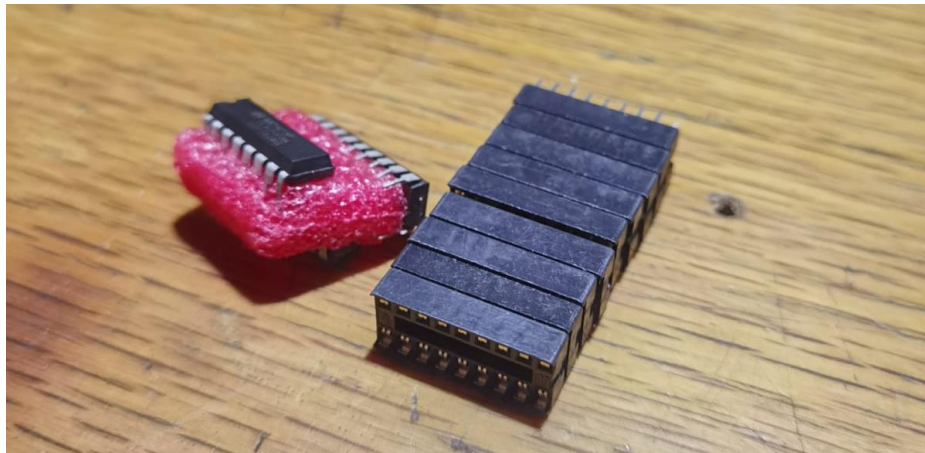


Figura 2.1.2 ULN2003 (Ismael Barajas, 2022).

En base a las pruebas con el ULN2003 se tuvo como resultado que no funcionaba tan bien como se deseaba, debido a que no alcanzaba el 100% la luminosidad del

led, así que se tomó la decisión de cambiarlos por el TIP41C y posteriormente haciendo pruebas.

b. 2.2 Proceso:

Electrónica:

El uso del TIP41C hizo que el LED encendiera a la luminosidad deseada, estas pruebas se hicieron en protoboard que se muestra en la figura 2.2.1

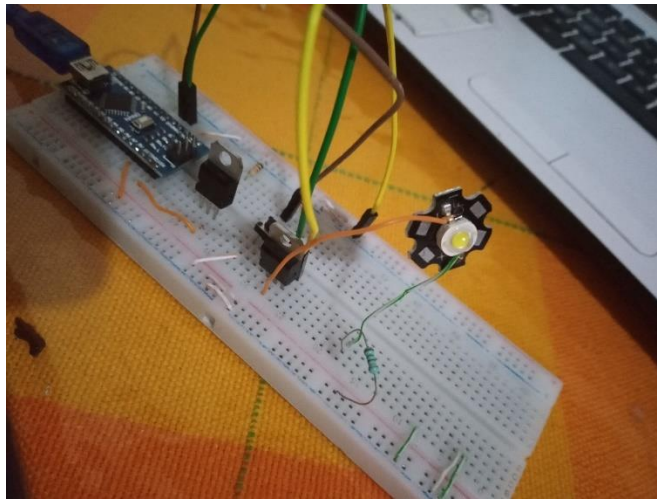


Figura 2.2.1 Prueba de conexión con TIP41 en protoboard (Ismael Barajas, 2022).

Una vez comprobando que funcionaba como se deseaba, se pasó al esquemático para analizar en qué manera se haría implementarían el resto de los leds y que estos funcionaran en conjunto. El primer esquemático realizado (al cual más adelante se le hicieron grandes cambios) se muestra en la figura 2.2.2.

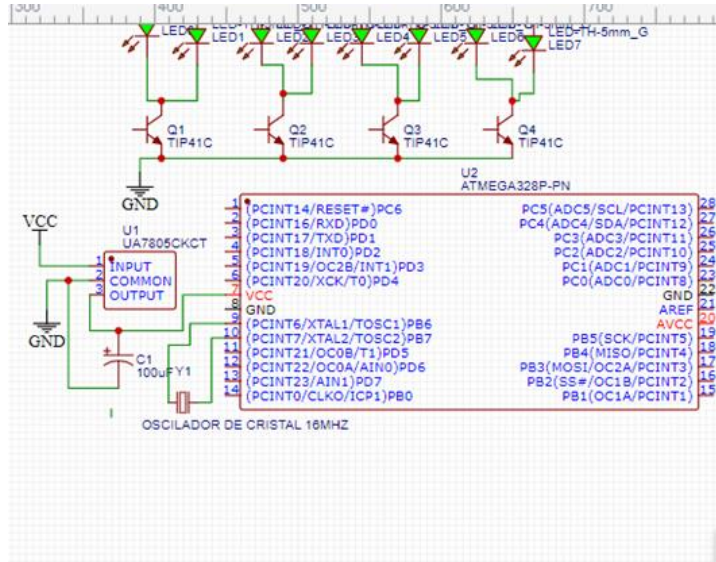


Figura 2.2.2 Primer esquemático (Ismael Barajas, EasyEDA 2022).

Se hicieron 2 esquemáticas, debido a que al analizar el primero, tenía algunos errores para poder activar los leds. Entonces hay se plantea hacer el propio circuito de fuente, que se tiene que acondicionar para proporcionar los 5 volts para alimentar el microcontrolador, para esto se utilizó un 7805, lo cual permite tener 12 Voltios de entrada (IN) y 5 Voltios de salida (OUT).

Así que se modificó haciendo calculas para colocar las resistencias que se necesitaban para enviar la señal proviniendo del microcontrolador al transistor. Lo cual dicho calculo dio 440 ohms y se logró activar esa señal.

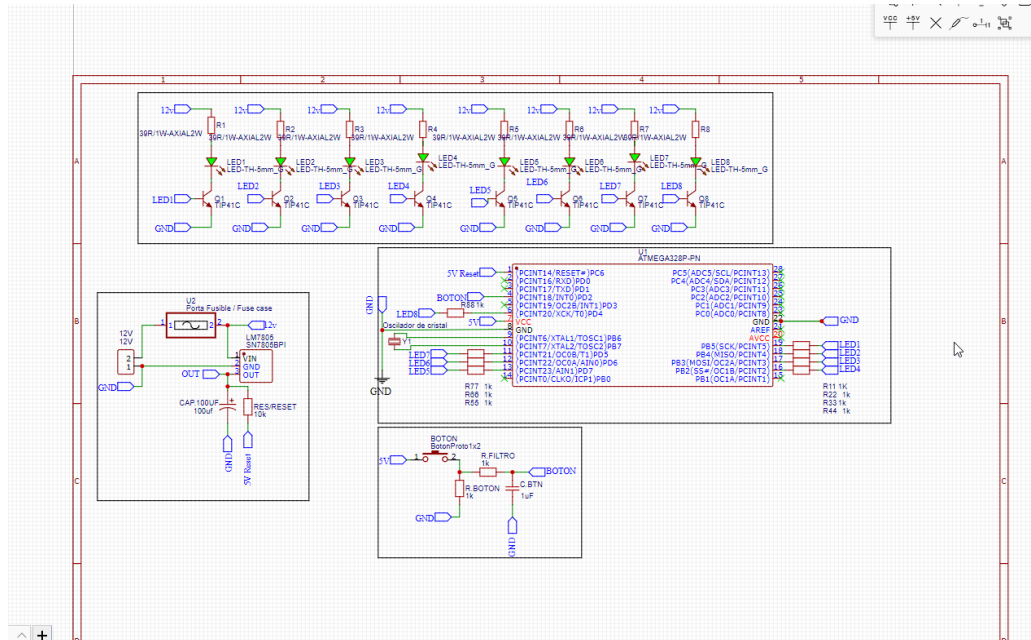


Figura 2.2.3 Segundo esquemático (Ismael Barajas y Satomi Minami, EasyEDA 2022).

En la siguiente figura se muestra el armado en proto.

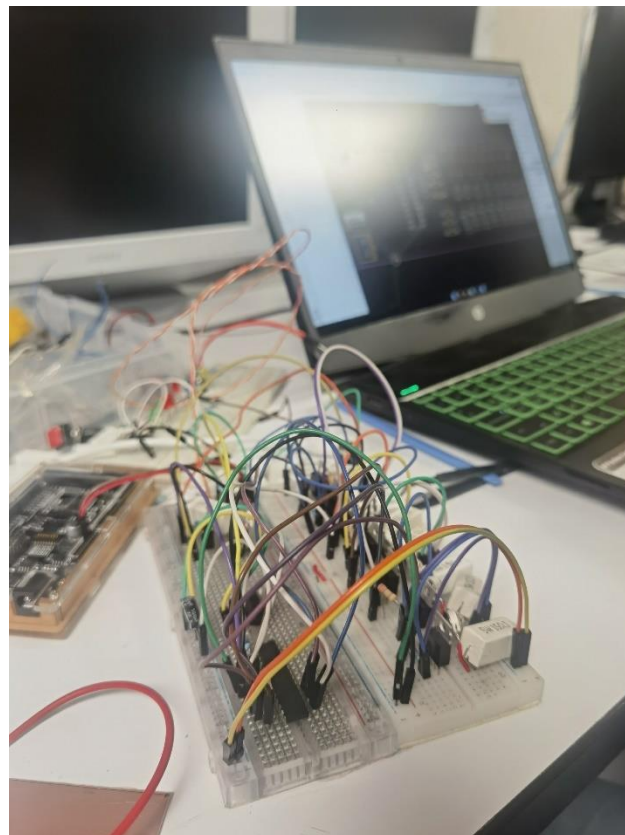


Figura 2.2.4 Prueba del circuito completo en protoboard (Satomi Minami, 2022).

Al concluir que el funcionamiento era el correcto se comenzó con el diseño de la PCB, en la figura 2.2.5 se muestra el diseño terminado en el software de EasyEDA.

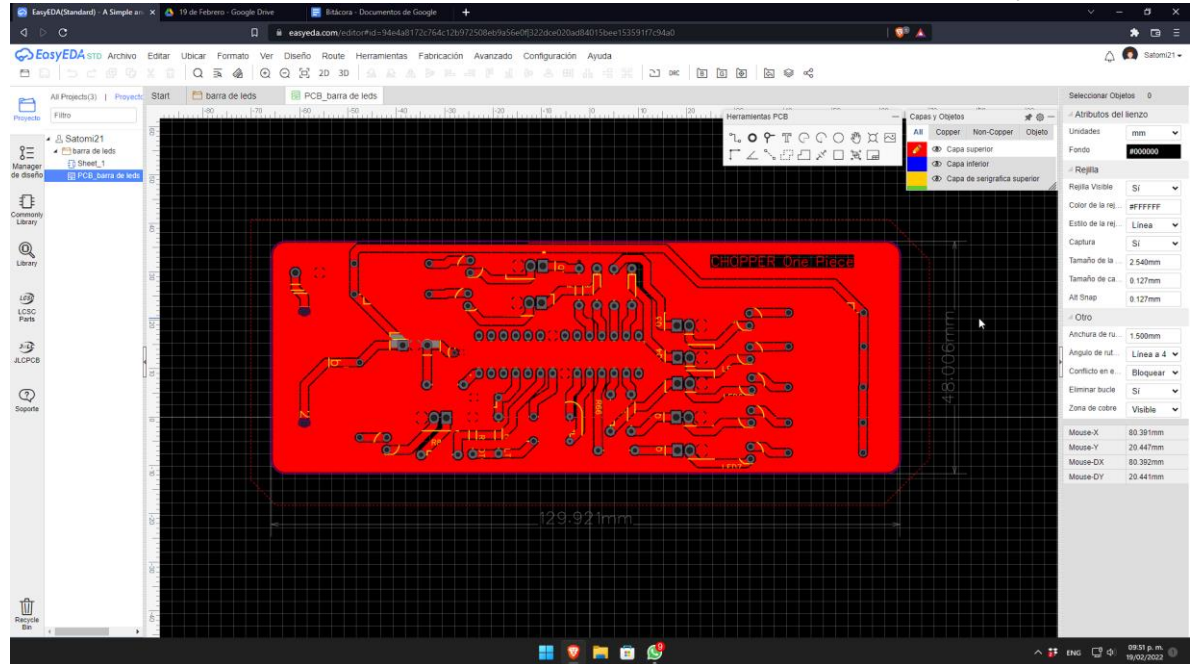


Figura 2.2.5 Diseño de PCB (Satomi Minami, Ismael Barajas y Jennifer González, EasyEDA, 2022).

Para poder realizar la PCB se utilizó una cortadora laser, se pintó la baquela sin perforar y luego la máquina comenzó a marcar las pistas, una vez finalizado ese trabajo. Se paso por el proceso de ácido férrico combinado con agua oxigena para tumbar el cobre. En las figuras 2.2.6, 2.2.7 y 2.2.8 se muestra el procedimiento mencionado.



Figura 2.2.6 Corte Láser (Satomi Minami, 2022).



Figura 2.2.7. Proceso de PCB por ácido férrico y agua oxigenada (Satomi Minami, 2022).



Figura 2.2.8 Proceso de PCB finalizado (Satomi Minami, 2022).

Programación:

Se comenzó investigando cómo funciona el bitwise para poder implementarlo en el código. En las siguientes figuras se muestran los ejemplos y definiciones del bitwise que se utilizó para implementarlo en el código del proyecto, comenzando con estos ejemplos (Figura 2.2.9, 2.2.10, 2.2.11, 2.2.12).

Description

There is a somewhat unusual operator in C++ called bitwise EXCLUSIVE OR, also known as bitwise XOR. (In English this is usually pronounced "eks-or".) The bitwise XOR operator is written using the caret symbol `^`. A bitwise XOR operation results in a 1 only if the input bits are different, else it results in a 0.

Precisely,

```
0 0 1 1   operand1
0 1 0 1   operand2
-----
0 1 1 0   (operand1 ^ operand2) - returned result
```

Example Code

```
int x = 12;    // binary: 1100
int y = 10;    // binary: 1010
int z = x ^ y; // binary: 0110, or decimal 6
```

The `^` operator is often used to toggle (i.e. change from 0 to 1, or 1 to 0) some of the bits in an integer expression. In a bitwise XOR operation if there is a 1 in the mask bit, that bit is inverted; if there is a 0, the bit is not inverted and stays the same.

```
// Note: This code uses registers specific to AVR microcontrollers (Uno, Nano, Leonardo, Mega, etc.)
// it will not compile for other architectures
void setup() {
  DDRB = DDRB | 0b00100000; // set PB5 (pin 13 on Uno/Nano, pin 9 on Leonardo/Micro, pin 11 on Mega) as OUTPUT
  Serial.begin(9600);
}

void loop() {
  PORTB = PORTB ^ 0b00100000; // invert PB5, leave others untouched
  delay(100);
}
```

Figura 2.2.9. Bitwise XOR (Arduino, 2022).

Description

The bitwise NOT operator in C++ is the tilde character `~`. Unlike `&` and `|`, the bitwise NOT operator is applied to a single operand to its right. Bitwise NOT changes each bit to its opposite: 0 becomes 1, and 1 becomes 0.

In other words:

```
0 1   operand1
-----
1 0   ~operand1
```

Example Code

```
int a = 103; // binary: 000000001100111
int b = ~a;  // binary: 111111110011000 = -104
```

Notes and Warnings

You might be surprised to see a negative number like -104 as the result of this operation. This is because the highest bit in an int variable is the so-called sign bit. If the highest bit is 1, the number is interpreted as negative. This encoding of positive and negative numbers is referred to as two's complement. For more information, see the Wikipedia article on [two's complement](#).

As an aside, it is interesting to note that for any integer `x`, `~x` is the same as `-x - 1`.

At times, the sign bit in a signed integer expression can cause some unwanted surprises.

See also

Figura 2.2.10. Bitwise NOT (Arduino, 2022).

Description

The bitwise OR operator in C++ is the vertical bar symbol, |. Like the & operator, | operates independently each bit in its two surrounding integer expressions, but what it does is different (of course). The bitwise OR of two bits is 1 if either or both of the input bits is 1, otherwise it is 0.

In other words:

```
0 0 1 1   operand1
0 1 0 1   operand2
-----
0 1 1 1   (operand1 | operand2) - returned result
```

Example Code

```
int a = 92;    // in binary: 000000001011100
int b = 101;   // in binary: 000000001100101
int c = a | b; // result:    000000001111101, or 125 in decimal.
```

One of the most common uses of the Bitwise OR is to set multiple bits in a bit-packed number.

```
// Note: This code is AVR architecture specific
// set direction bits for pins 2 to 7, leave PD0 and PD1 untouched (xx | 00 == xx)
// same as pinMode(pin, OUTPUT) for pins 2 to 7 on Uno or Nano
DDRD = DDRD | 0b1111100;
```

Figura 2.2.11. Bitwise OR (Arduino, 2022).

Description

The bitwise AND operator in C++ is a single ampersand &, used between two other integer expressions. Bitwise AND operates on each bit position of the surrounding expressions independently, according to this rule: if both input bits are 1, the resulting output is 1, otherwise the output is 0.

Another way of expressing this is:

```
0 0 1 1   operand1
0 1 0 1   operand2
-----
0 0 0 1   (operand1 & operand2) - returned result
```

In Arduino, the type int is a 16-bit value, so using & between two int expressions causes 16 simultaneous AND operations to occur.

Example Code

In a code fragment like:

```
int a = 92;    // in binary: 000000001011100
int b = 101;   // in binary: 000000001100101
int c = a & b; // result:    000000001000100, or 68 in decimal.
```

Each of the 16 bits in a and b are processed by using the bitwise AND, and all 16 resulting bits are stored in c, resulting in the value 01000100 in binary, which is 68 in decimal.

Figura 2.2.12. Bitwise AND (Arduino, 2022).

Description

The left shift operator << causes the bits of the left operand to be shifted **left** by the number of positions specified by the right operand.

Syntax

```
variable << number_of_bits;
```

Parameters

variable: Allowed data types: byte, int, long.

number_of_bits: a number that is <= 32. Allowed data types: int.

Example Code

```
int a = 5; // binary: 000000000000101
int b = a << 3; // binary: 000000000101000, or 40 in decimal
```

Notes and Warnings

When you shift a value x by y bits (x << y), the leftmost y bits in x are lost, literally shifted out of existence:

```
int x = 5; // binary: 000000000000101
int y = 14;
int result = x << y; // binary: 010000000000000 - the first 1 in 101 was discarded
```

If you are certain that none of the ones in a value are being shifted into oblivion, a simple way to think of the left-shift operator is that it multiplies the left operand by 2 raised to the right operand power. For example, to generate powers of 2:

Figura 2.2.12. Operador left shift (Arduino, 2022).

Description

The right shift operator >> causes the bits of the left operand to be shifted **right** by the number of positions specified by the right operand.

Syntax

```
variable >> number_of_bits;
```

Parameters

variable: Allowed data types: byte, int, long.

number_of_bits: a number that is <= 32. Allowed data types: int.

Example Code

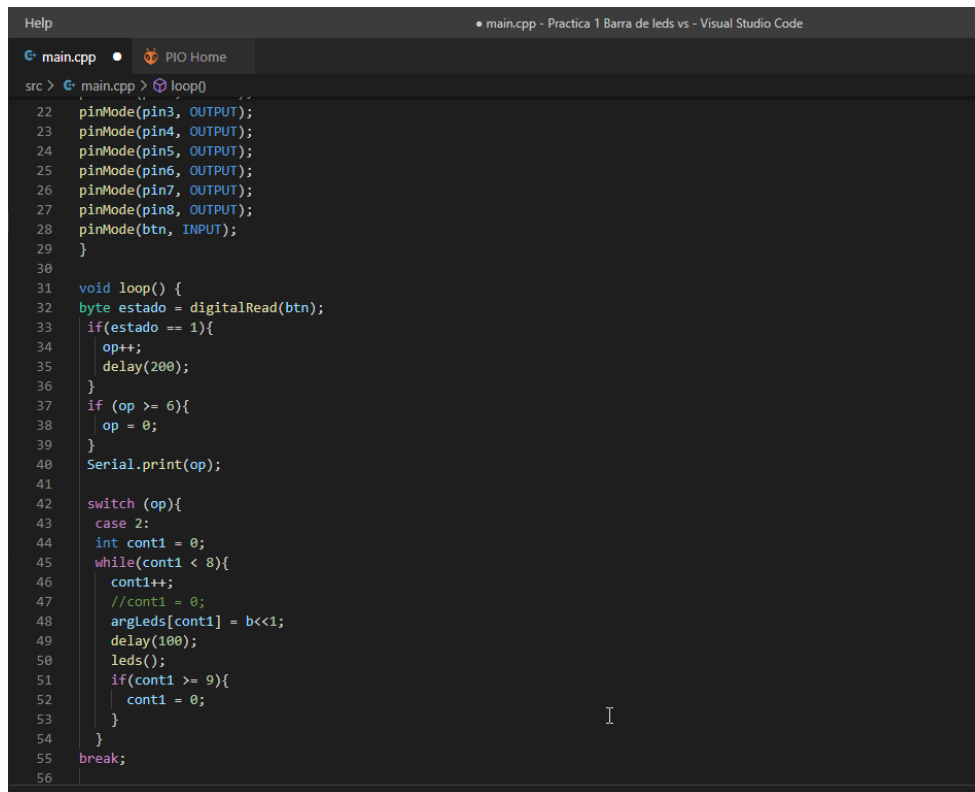
```
int a = 40; // binary: 000000000101000
int b = a >> 3; // binary: 00000000000101, or 5 in decimal
```

Notes and Warnings

When you shift x right by y bits (x >> y), and the highest bit in x is a 1, the behavior depends on the exact data type of x. If x is of type int, the highest bit is the sign bit, determining whether x is negative or not, as we have discussed above. In that case, the sign bit is copied into lower bits for esoteric historical reasons:

Figura 2.2.13. Operador right shift (Arduino, 2022).

Después se trabajó en el código pues se estuvo experimentando los métodos del bitwise. Este proceso se muestra en la figura 2.2.14.



```
Help
main.cpp - Practica 1 Barra de leds vs - Visual Studio Code
main.cpp
src > main.cpp > loop()
22 pinMode(pin3, OUTPUT);
23 pinMode(pin4, OUTPUT);
24 pinMode(pin5, OUTPUT);
25 pinMode(pin6, OUTPUT);
26 pinMode(pin7, OUTPUT);
27 pinMode(pin8, OUTPUT);
28 pinMode(btn, INPUT);
29 }
30
31 void loop() {
32   byte estado = digitalRead(btn);
33   if(estado == 1){
34     op++;
35     delay(200);
36   }
37   if (op >= 6){
38     op = 0;
39   }
40   Serial.print(op);
41
42   switch (op){
43     case 2:
44       int cont1 = 0;
45       while(cont1 < 8){
46         cont1++;
47         //cont1 = 0;
48         argleds[cont1] = b<<1;
49         delay(100);
50         leds();
51         if(cont1 >= 9){
52           cont1 = 0;
53         }
54       }
55       break;
56 }
```

Figura 2.2.14. Código: Probando métodos del bitwise (Visual Studio Code, 2022).

Una vez que se entendió claramente cómo funcionaba el bitwise y las pruebas terminaban bien, se hizo un avance en las secuencias y poco a poco se fue trabajando con ellas hasta lograr las 5 secuencias del objetivo para los leds.

```
#include <Arduino.h>
int pinOut[11] = {13,12,11,10,9,7,6,5};
#define btn 8
int op = 0;

int d = B111000;
int b = B111110;
// ----> DECLARACION DE FUNCIONES <---- \\

int boton();
void secIntercambio ();
void sec2Faro ();
void apagado ();
void sec3Sirena();
```



```
void sec4Direccional ();
void sec5inter();

void setup() {
  DDRD = B111111;
  DDRB = B111110;

  Serial.begin(9600);
  pinMode(btn, INPUT);
  d = B111100; //0-7 B111000; D5-0 controlados --> D5->3
  b = B011100; //8-13 B111110; D13-8 controlados --> d13->9
  PORTD = B1000000;
  PORTB = b;
}

void loop() { //////////////////////////////////////
  int estado = boton();
  Serial.println(estado);
  estado = 5;
  if(estado == 1){
    sec2Faro();
  }
  if(estado == 2){
    secIntercambio ();
  }
  if(estado == 3){
    sec3Sirena();
  }
  if(estado == 4){
    sec4Direccional ();
  }
  if(estado == 5){
    sec5inter();
  }
  if(estado == 6){
    apagado();
  }
}

int boton (){
  byte estado = digitalRead(btn);
  if(estado == 1){
    op++;
    delay(200);
  }
}
```

```
}
if (op > 6){
    op = 0;
}
return op;
}

void secIntercambio (){
    int a = B101000;
    int c = B010100;
    for (int i = 0; i < 5; i++){
        PORTB =~c;
        PORTD =~a;
        delay(100);
        PORTB =c;
        PORTD =a;
        delay(100);
    }
}

void sec2Faro (){
    int x=0, i = 0;
    int a = B000000;
    int A = B000000;
    int s = B000100;
    int q = B100000;
    for(int x=0; x<5; x++){ //Puerto B
        a = q>>x;
        PORTB = a;
        delay(50);
    }
    for(int i=0; i<4; i++){ // Puerto D
        A = q>>i;
        PORTD = A;
        delay(50);
    }
    for(int i=0; i<5; i++){ // Puerto D
        A = s<<i;
        PORTD = A;
        delay(50);
    }
    for(int x=0; x<4; x++){ //Puerto B
        a = s<<x;
        PORTB = a;
        delay(50);
    }
}
```

```
    }  
}  
  
void sec3Sirena(){  
    int u = B111111, count = 0, r =0;;  
    if(count == 0){  
        count = 1;  
        r =~u;  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(25);  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(25);  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(180);  
    }  
    if(count == 1){  
        count = 0;  
        r =~u;  
        PORTB = u;  
        delay(25);  
        PORTB = r;  
        delay(25);  
        PORTB = u;  
        delay(25);  
        PORTB = r;  
        delay(25);  
        PORTB = u;  
        delay(25);  
        PORTB = r;  
        delay(180);  
    }  
}  
  
void sec4Direccional () {  
    int r = B100000;  
    int c = B000100;  
    for(int x=0; x<4; x++){  
        int res = 0,res2 = 0;  
        res = r>>x;
```

```
    res2 = c<<x;
    PORTD = res2;
    PORTB = res;
    delay(100);
}
for(int x=0; x<4; x++){
    int res = 0, res2 = 0;
    res = r>>x;
    res2 = c<<x;
    PORTD = res;
    PORTB = res2;
    delay(100);
}
}

void sec5inter(){
    int b1 = B111010, b3 = B100000;
    int b2 = B011101, b4 = B000100;
    int res=0, res2=0, count=0;
    if(count == 0){
        count = 1;
        res = b1&b2;
        PORTD = res;
        PORTB = res;
        delay(200);
    }
    if(count == 1){
        count = 0;
        res2 = b3|b4;
        PORTD = res2;
        PORTB = res2;
        delay(200);
    }
}

void apagado (){
    PORTB = B000000;
    PORTD = B000000;
}
```

En la figura 2.2.15 se muestra el funcionamiento de los leds (junto con el armado en proto) utilizados como prueba para verificar que las secuencias se estuvieran cumpliendo.



Figura 2.2.15. Prueba de secuencia con leds en protoboard (Satomi Minami, 2022).

Finalmente se concluyó el código.

Carcasa:

Se hizo una lluvia de ideas entre todo el equipo, en base a eso se crearon los primeros bocetos, para opciones para al final elegir una. En la figura 2.2.16 a la 2.2.19 se muestran estos bosquejos.

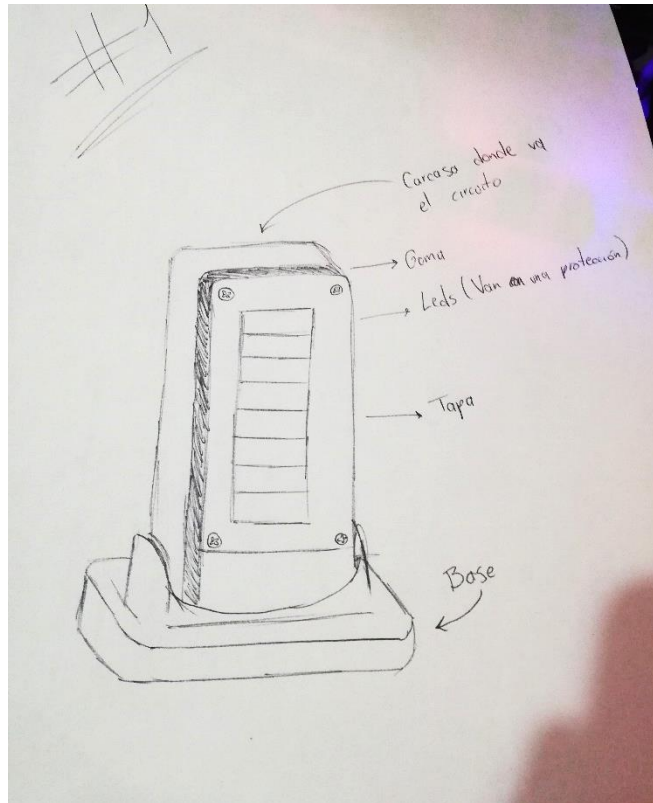


Figura 2.2.16. Bosquejo 1 (Jennifer González, 2022).

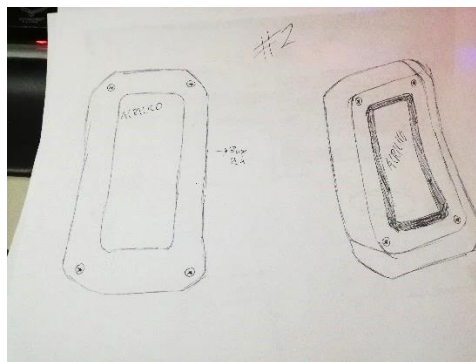


Figura 2.2.17. Bosquejo 2 (Jennifer González, 2022).

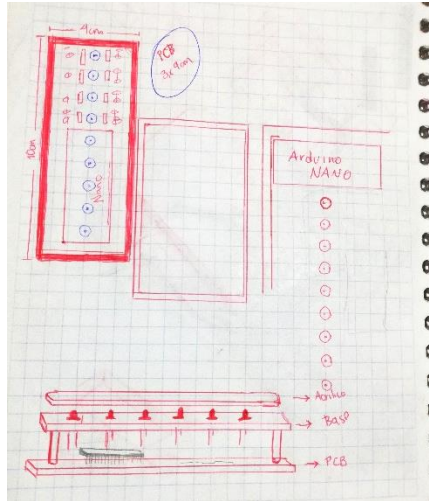


Figura 2.2.18. Bosquejo 3 dimensiones y vista superior (Jennifer González, 2022).

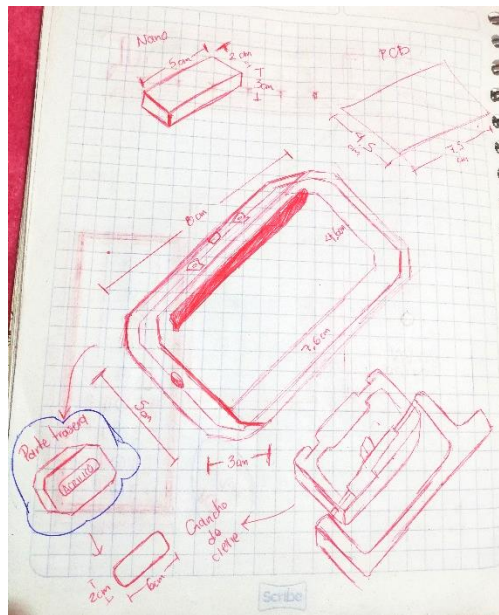


Figura 2.2.19. Bosquejo 3 Tridimensional con dimensiones (Jennifer González, 2022).

Al final del día se decidió por un diseño, el cual se comenzó a modelar y como se iba avanzando en el proceso se iban haciendo cambios. El primer modelo decisivo se hizo pieza por pieza tomando en cuenta las dimensiones del bosquejo, al cual llamaremos "Diseño #1".

Sin embargo, la primera pieza modelada en Inventor fue el soporte de los leds, para ver la distancia entre ellos (esto se muestra en la figura 2.2.20) al ver que coincidían, al final se hizo un ensamblaje de todas las piezas para verificar que las dimensiones coincidían entre sí, este resultado se muestra en la figura 2.2.21.

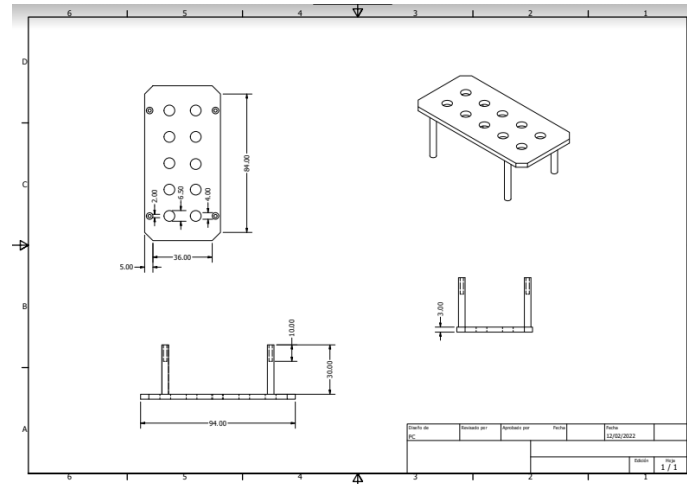


Figura 2.2.20 Plano de la pieza 'LedsSupport" (Borrador) (Jennifer González, Inventor 2022).



Figura 2.2.21. Ensamble final del diseño #1 (Jennifer González, Inventor 2022).

Al mostrar el ensamble del diseño #1 el equipo estuvo desacuerdo en el acomodo de leds, entonces todos dijeron que lo mejor era acomodarlos en una sola hilera y así cumpliría mejor el objetivo de que tiene que ser una tira led. Entonces se realizaron modificaciones de las piezas para hacerlas un poco más grandes, y de ser 10 leds se redujeron a 8, para que fueran en 1 sola hilera. Las modificaciones se muestran en la figura 2.2.22 y el plano de ensamble en la figura 2.2.23.

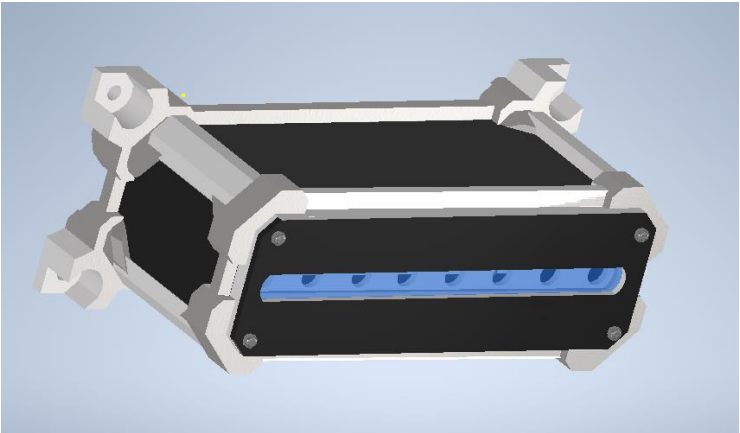


Figura 2.2.22. Ensamble final del diseño #2(Jennifer González, Inventor 2022).

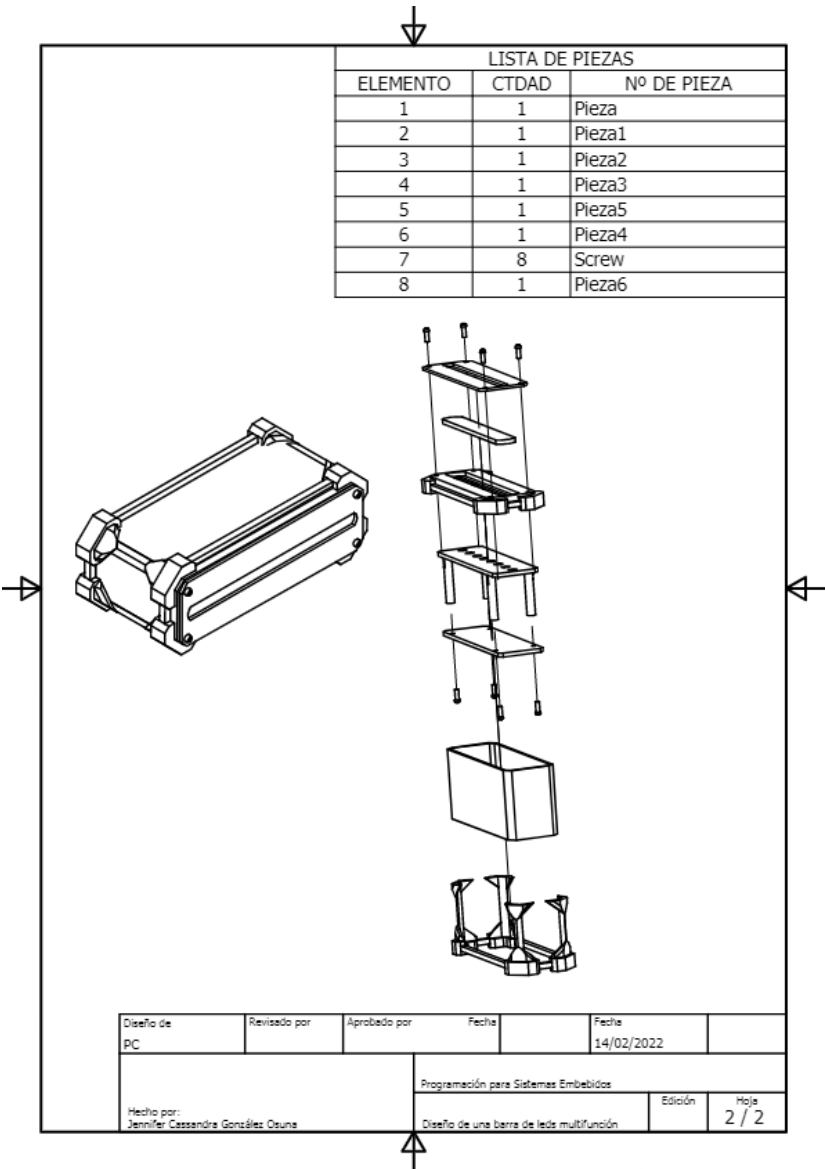


Figura 2.2.23. Plano del ensamble final del diseño #2(Jennifer González, Inventor 2022).

Después se plantea la idea de como hacer el sujetador que hará posible que la carcasa pueda ajustarse a una parrilla de un auto. Para esto se crea un boceto, como objetivo principal es buscar un diseño sencillo, con cuerpo capaz de no romperse. En la figura 2.2.24 se muestra el boceto mencionado.

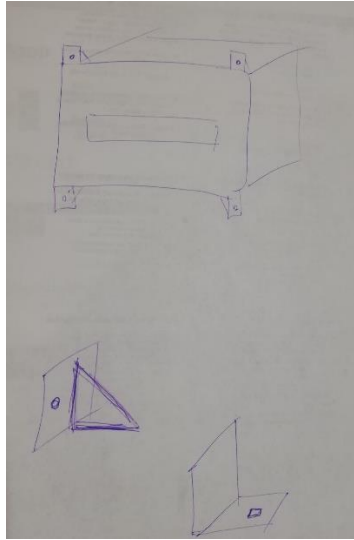


Figura 2.2.24 Boceto del sujetador implementado en la carcasa. (Jennifer González, 2022).

Después de esto se modifica la parte de la base de la carcasa para agregar la idea del boceto anterior, para poder adaptar esta carcasa a la parrilla de un automóvil mostrado en las figuras 2.2.25 y 2.2.26.

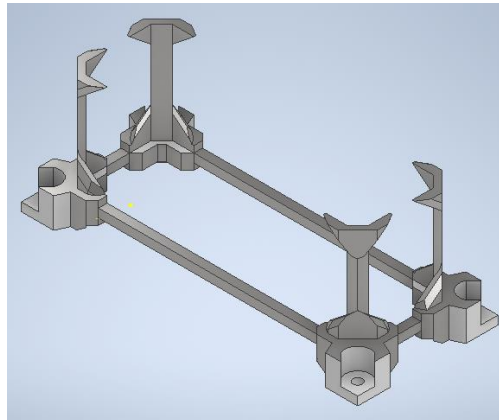


Figura 2.2.25. (Jennifer González, Inventor 2022).

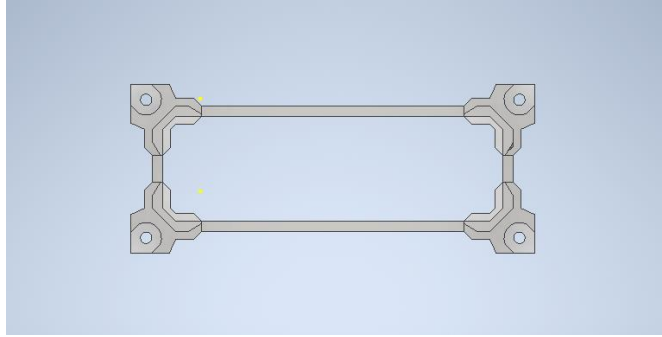


Figura 2.2.26. Vista superior (Jennifer González, Inventor 2022).

Después de eso se comenzó con la impresión, pero hubo errores con unas piezas, algunas secciones de la pieza base estaban muy débiles, por ende, se rompieron, también unas perforaciones muy pequeñas debido a que con el PLA está se fue reduciendo más de lo que debía. En la figura 2.2.27 se observa las primeras piezas impresas y la base de está rota.



Figura 2.2.27 Defecto de impresión y de diseño (Jennifer González, 2022).

En base a esto, se decidió extender el tamaño de la carcasa un centímetro por lado, para no tener tan limitado el espacio de la carcasa y por ende que algunas paredes de ciertas piezas quedaran más resistentes.

Al modificar nuevamente la carcasa se imprimieron las piezas, en la siguiente figura se muestran algunas de estas.

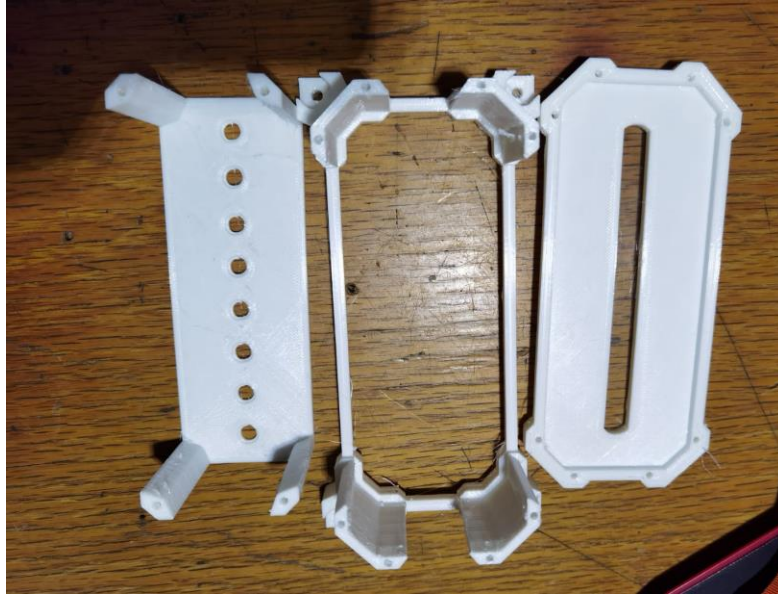


Figura 2.2.28 Impresión de las piezas (Jennifer González, 2022).

En esta última etapa donde ya se obtienen las piezas en PLA, se tenía contemplado de realizar un pequeño prototipo para hacer un pulido de PLA, esto para evitar eliminar cualquier poro de la impresión y las marcas de cada capa. Sin embargo, con el tiempo que se tenía asignado para terminar el proyecto, no nos alcanzó el tiempo para hacer esta parte así que tuvo que ser omitida.

Por último, se cortaron las caras que llevan lámina, y se perforaron para poder ensamblarlas a la carcasa final. En la figura 2.2.29 se muestra las piezas cortadas en lámina y en la figura 2.2.30 el resultado final después de pintarlas de negro mate.

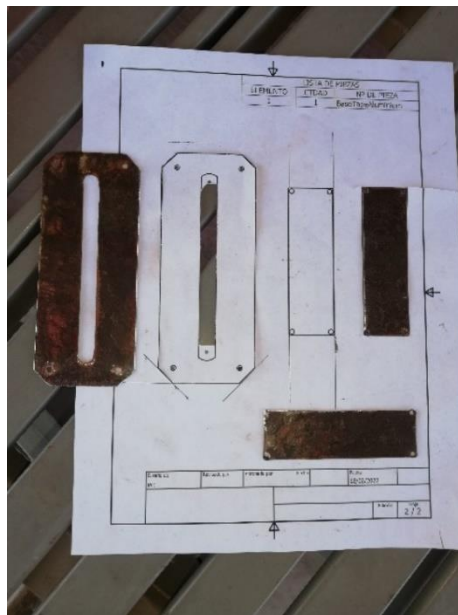


Figura 2.2.29. Tapas de la carcasa cortadas en lámina (Jennifer González, 2022).



Figura 2.2.30. Piezas cortadas pintadas en negro mate (Jennifer González, 2022).

3. Resultados:

a. 3.1 Diseños.

A continuación, se muestra la carcasa final, pieza por pieza y así como el ensamblaje final a partir de la figura 3.1.1 a la 3.1.20.

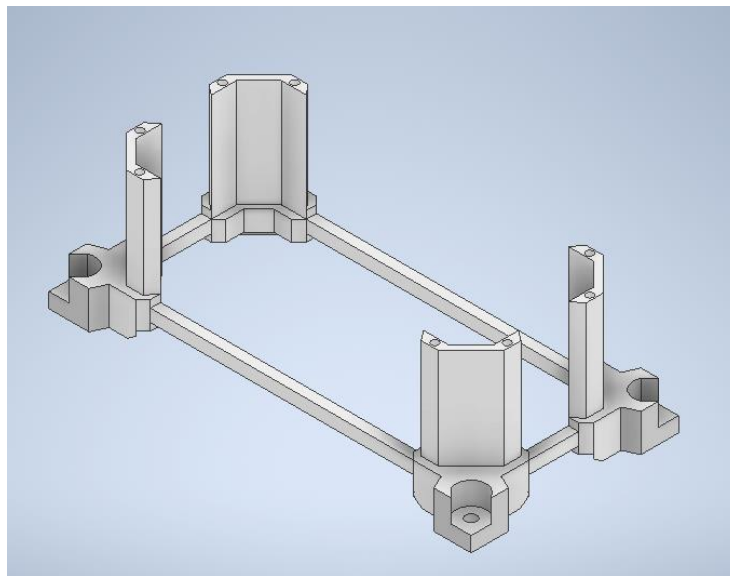


Figura 3.1.1 Base de la carcasa (Jennifer González, Inventor 2022).

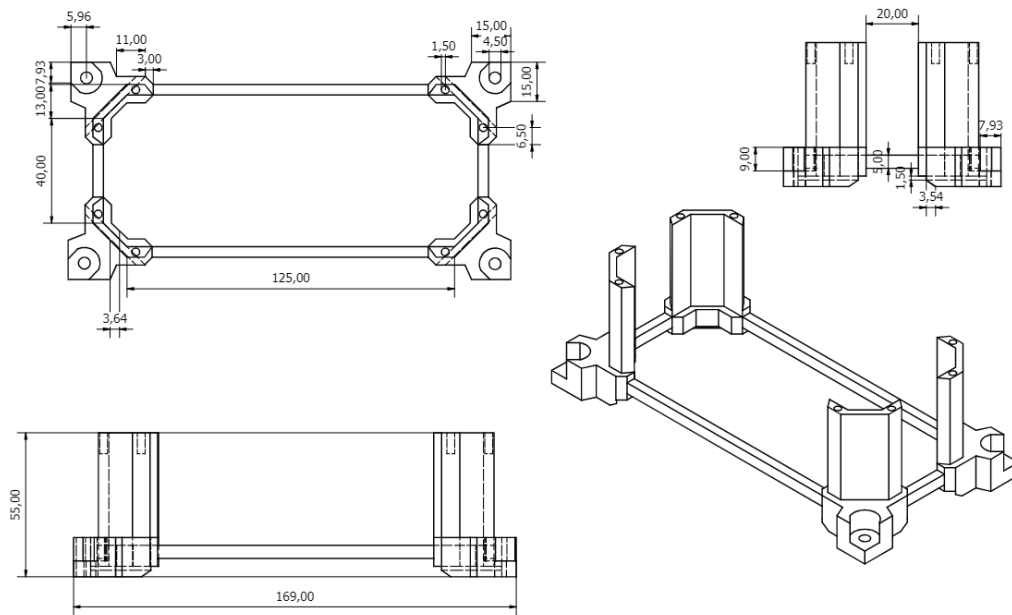


Figura 3.1.2 Plano de la base de la carcasa (Jennifer González, Inventor 2022).

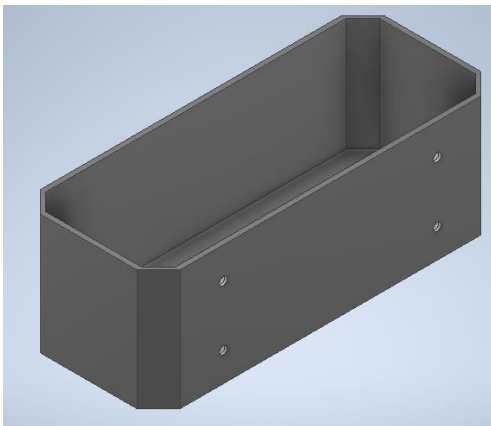


Figura 3.1.3 Caja (Jennifer González, Inventor 2022).

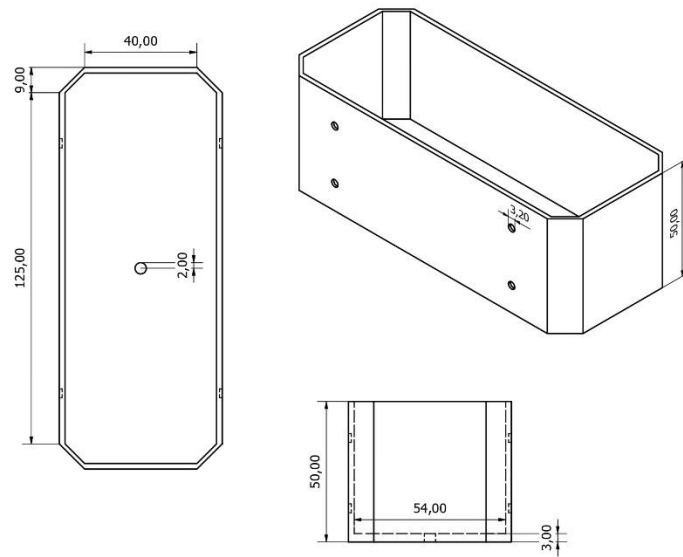


Figura 3.1.4. Plano de caja (Jennifer González, Inventor 2022).

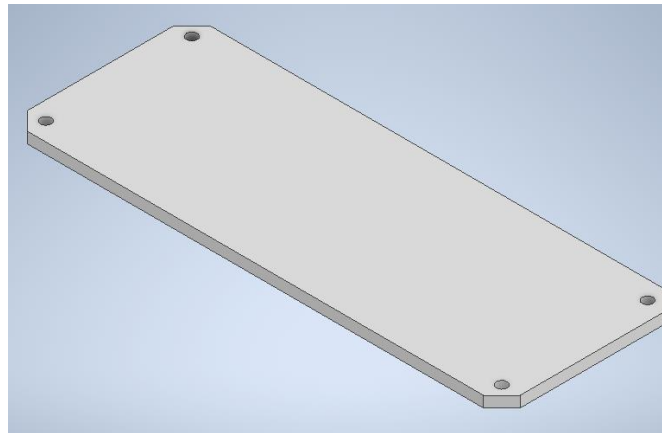


Figura 3.1.5. Base para PCB (Jennifer González, Inventor 2022).

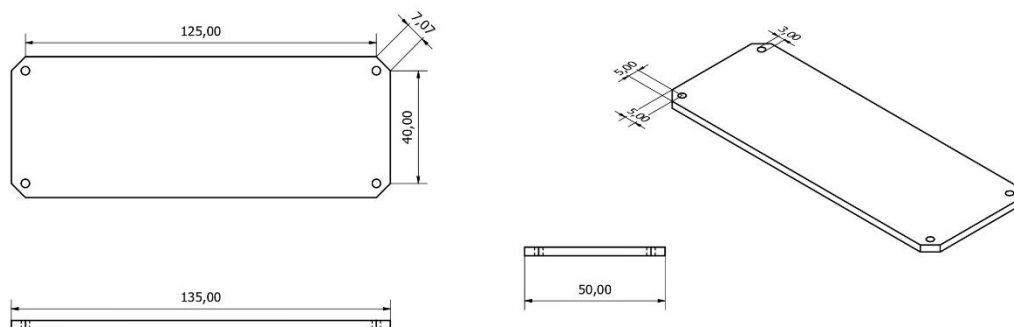


Figura 3.1.6 Plano de base para PCB (Jennifer González, Inventor 2022).

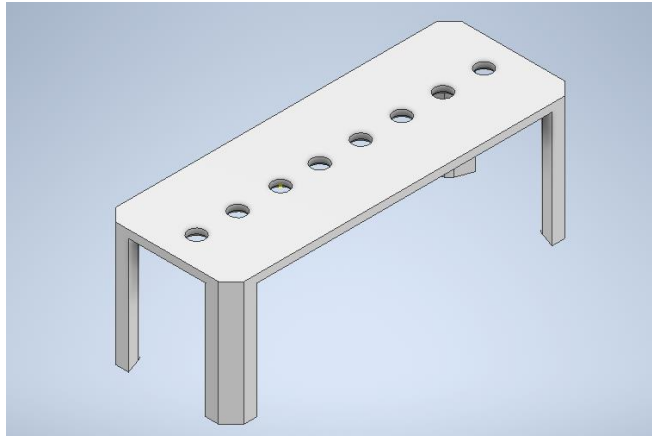


Figura 3.1.7 Soporte para LEDs (Jennifer González, Inventor 2022).

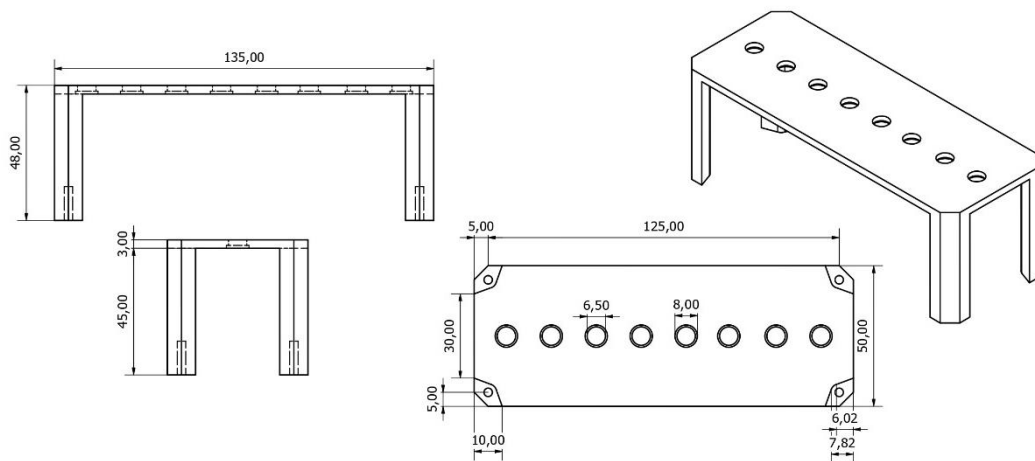


Figura 3.1.8. Plano de soporte para LEDs(Jennifer González, Inventor 2022).

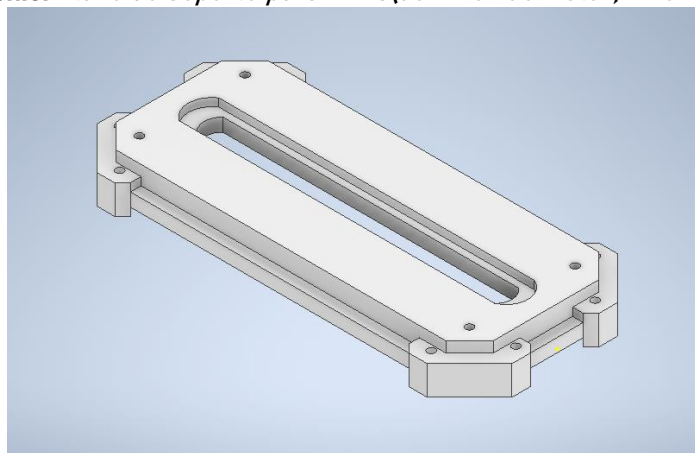


Figura 3.1.9. Cabeza de la carcasa (Jennifer González, Inventor 2022).

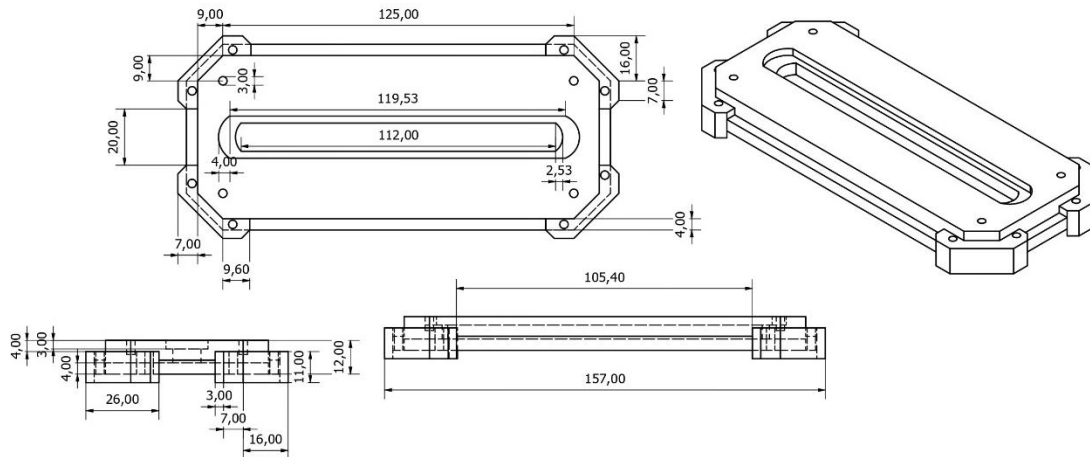


Figura 3.1.10. Plano de la cabeza de la carcasa (Jennifer González, Inventor 2022).

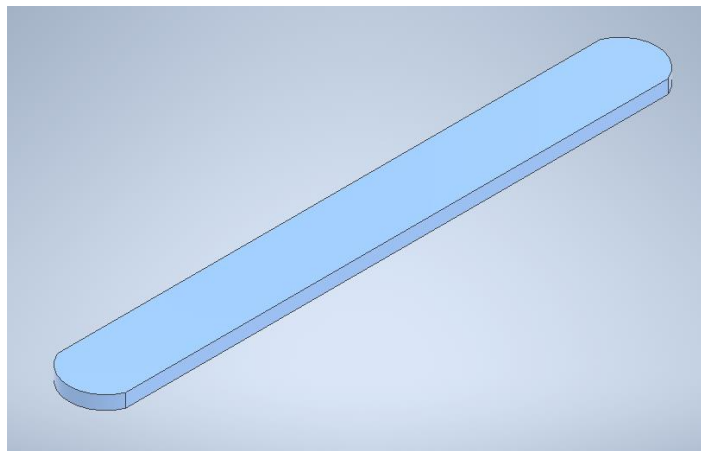


Figura 3.1.11. Tapa de acrílico para la cabeza (Jennifer González, Inventor 2022).

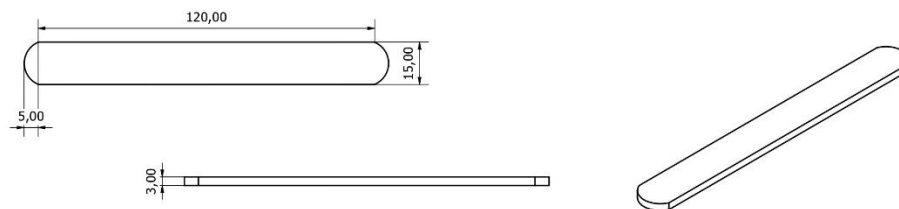


Figura 3.1.12. Plano de la tapa de acrílico para la cabeza (Jennifer González, Inventor 2022).

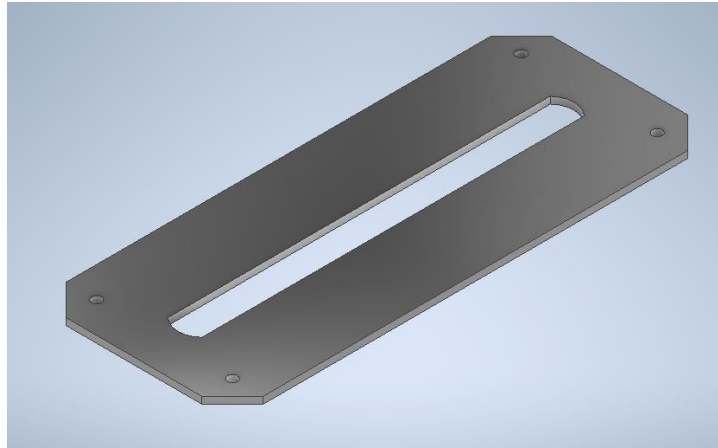


Figura 3.1.13. Tapa de lámina para la cabeza (Jennifer González, Inventor 2022).

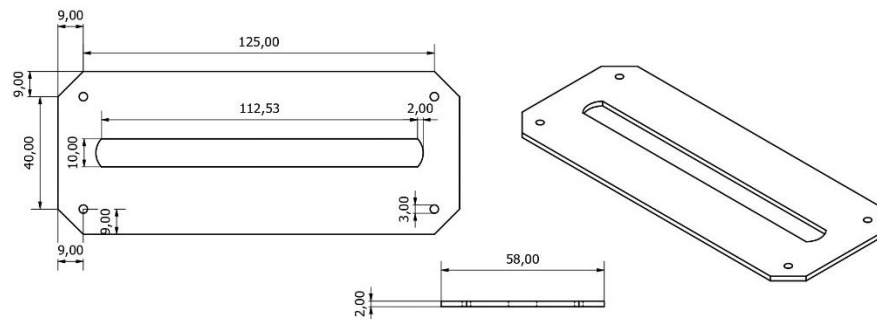


Figura 3.1.14. Plano de tapa de lámina para la cabeza (Jennifer González, Inventor 2022).

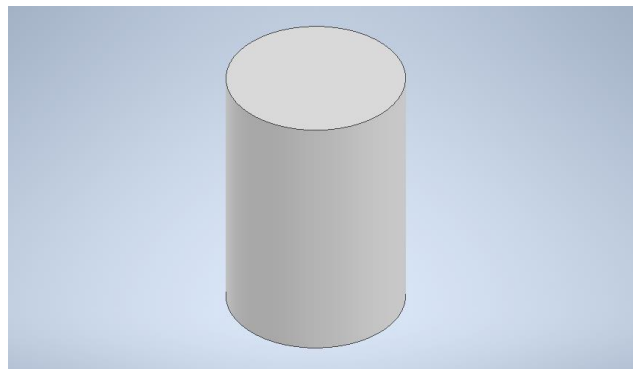


Figura 3.1.15. Cilindro, soporte de tapas laterales (Jennifer González, Inventor 2022).

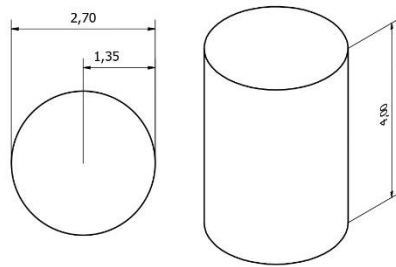


Figura 3.1.16. Plano de cilindro, soporte de tapas laterales (Jennifer González, Inventor 2022).

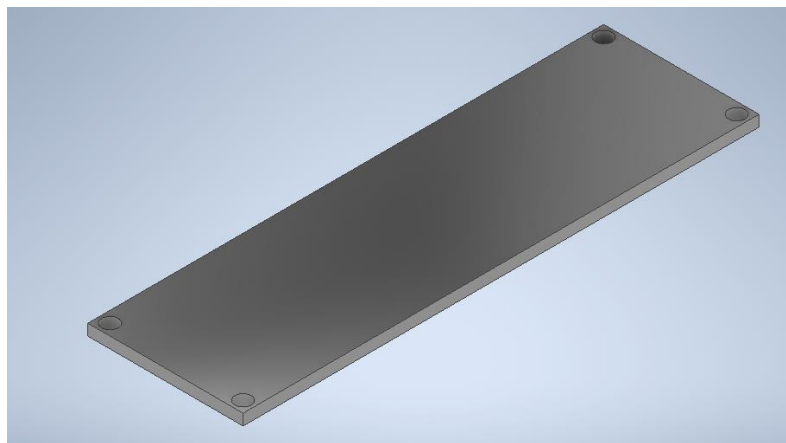


Figura 3.1.17. Tapa lateral de lámina (Jennifer González, Inventor 2022).

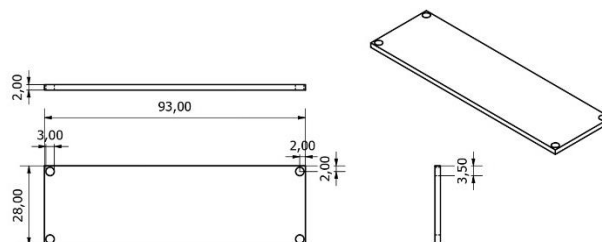


Figura 3.1.18. Plano de tapa lateral de lámina (Jennifer González, Inventor 2022).

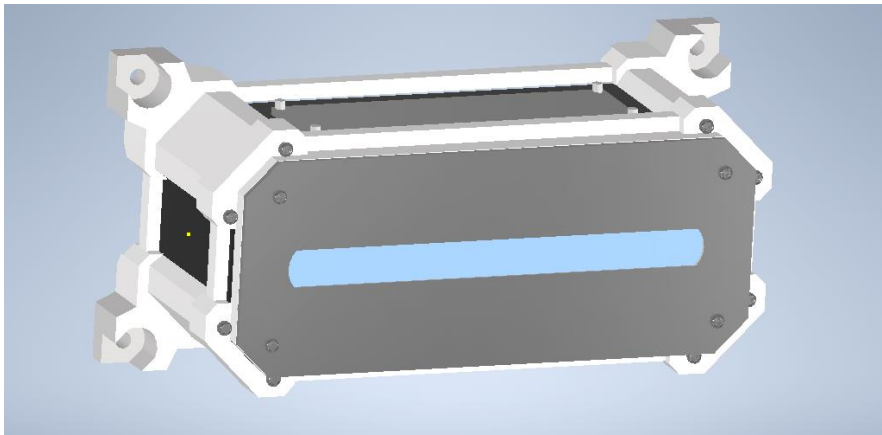


Figura 3.1.19. Ensamble final (Jennifer González, Inventor 2022).

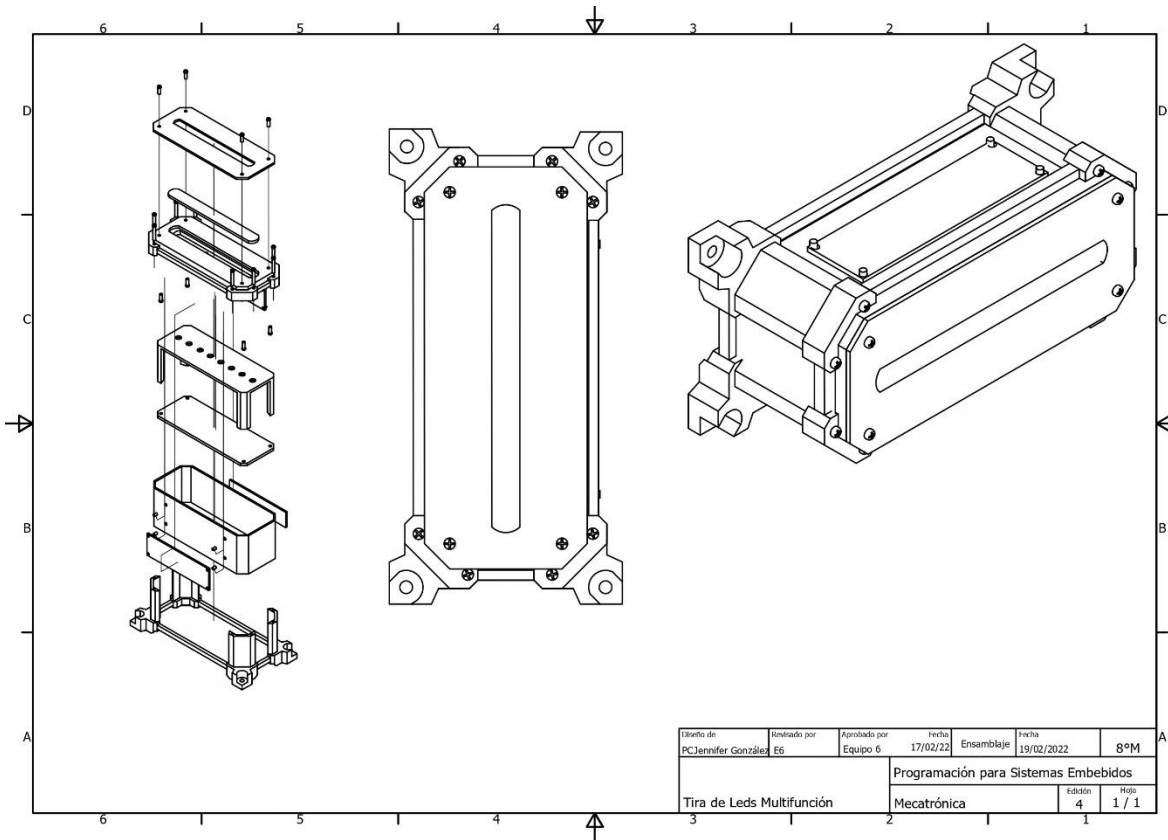


Figura 3.1.20. Plano de ensamble final (Jennifer González, Inventor 2022).

b. 3.2 Circuitos.

Como se menciona durante el proceso del circuito eléctrico, conforme a las pruebas que se realizaron en protoboard y a los cálculos realizados para saber que resistencias se necesitarían en ciertas partes del circuito, se hicieron cambios muy notorios y el esquemático final se muestra en la figura 3.2.1.

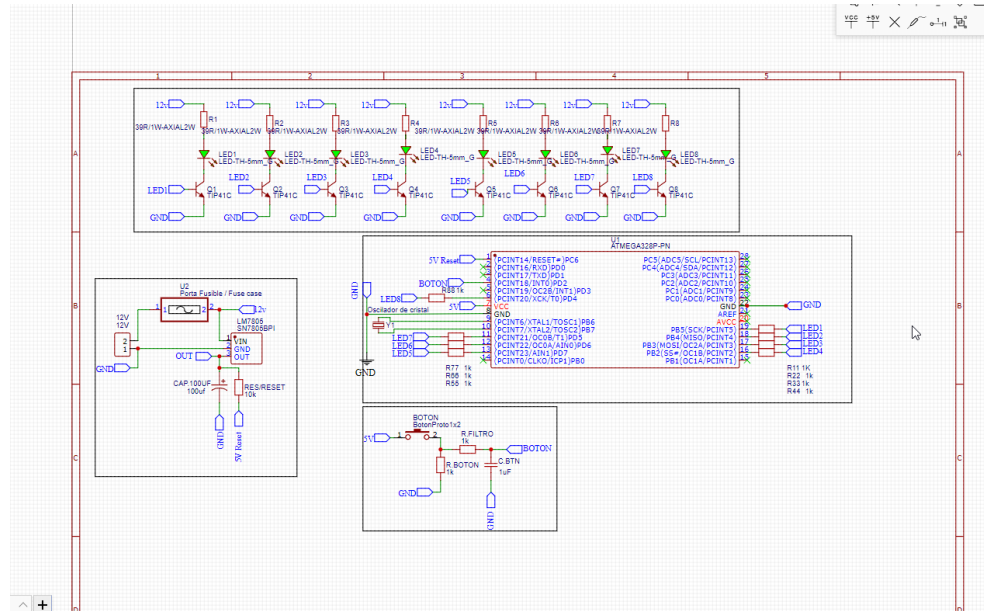


Figura 3.2.1 Diagrama Final (Ismael Barajas, EasyEDA, 2022).

El resultado final de la PCB con los componentes ya ensamblados y soldados de muestra en la figura 3.2.2.

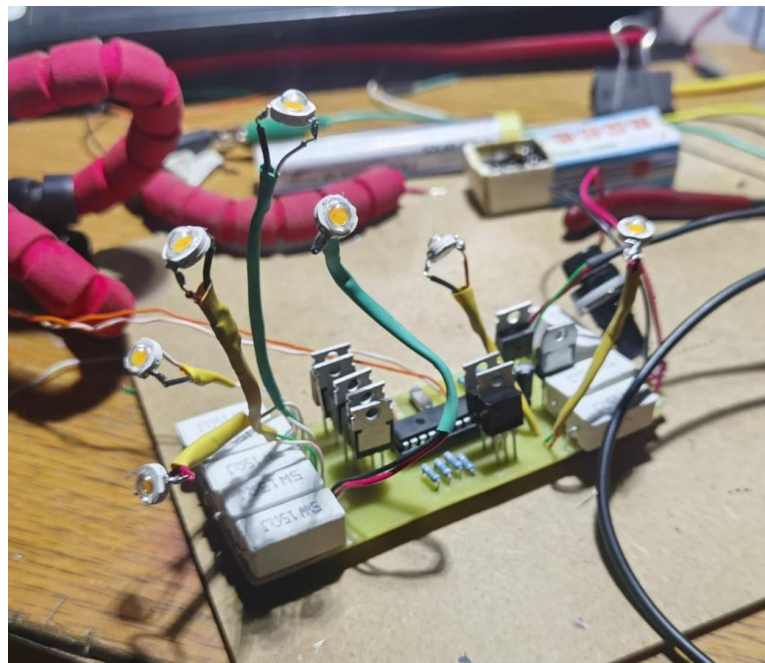


Figura 3.2.2 PCB Final (Satomi Minami, 2022).

c. Códigos.

El código final quedó de la siguiente manera:

```
// --> VARIABLES <--
boolean pos = false;
int count = 0;

// --> FUNCIONES <--
void btnf (); // función de la interrupción
void apagado (); //FUNCION DE APAGADO
void secIntercambio (); //SECUENCIA 1
void sec2Faro (); //SECUENCIA 2
void sec3Sirena(); //SECUENCIA 3
void sec4Direccional (); //SECUENCIA 4
void sec5inter(); //SECUENCIA 5

void setup(){
  Serial.begin(9600);
  pinMode(btn, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(btn), btnf, RISING);
  DDRD = B11110000;
  DDRB = B00111100;
  PORTD = B00000000; // 7 6 5 4 -3 2 1-
  PORTB = B00000000; // c c 13 12 11 10 -9 8-
}

void loop(){
  if(pos == true){
    pos = false;
    count++;
    PORTD = B00000000;
    PORTB = B00000000;
    Serial.println(count);
  }

  if(count == 1){
    secIntercambio();
  }
  if(count == 2){
    sec2Faro();
  }
  if(count == 3){
    sec3Sirena();
  }
  if(count == 4){
    sec4Direccional ();
  }
  if(count == 5){
```

```
    sec5inter();
}
if(count == 6){
    count = 0;
}

}

void secIntercambio (){ // SECUENCIA 1
    int a = B10100100,A = B01010100;
    int c = B00101000;
    for (int i = 0; i < 6; i++){
        PORTB =c;
        PORTD =~a;
        delay(100);
        PORTB =~c;
        PORTD =~A;
        delay(100);
    }
}

void sec2Faro (){ //SECUENCIA 2
    int a = B00000000;
    int A = B00000000;
    int s = B00010000, s2 = B10000000;
    int q = B00100000, q2 = B00000100;
    for(int x=0; x<5; x++){ //Puerto B
        a = q>>x;
        PORTB = a;
        delay(50);
    }
    for(int i=0; i<5; i++){ // Puerto D
        A = s2>>i;
        PORTD = A;
        delay(50);
    }
    for(int i=0; i<5; i++){ // Puerto D
        A = s<<i;
        PORTD = A;
        delay(50);
    }
    for(int x=0; x<4; x++){ //Puerto B
        a = q2<<x;
        PORTB = a;
        delay(50);
    }
}
```

```
    }  
}  
  
void sec3Sirena(){ //SECUENCIA 3  
    int u = B11110000, count = 0, r = 0, r2=0, u2 = B00111100;  
    if(count == 0){  
        count = 1;  
        r = ~u;  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(25);  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(25);  
        PORTD = u;  
        delay(25);  
        PORTD = r;  
        delay(180);  
    }  
    if(count == 1){  
        count = 0;  
        r2 = ~u2;  
        PORTB = u2;  
        delay(25);  
        PORTB = r2;  
        delay(25);  
        PORTB = u2;  
        delay(25);  
        PORTB = r2;  
        delay(25);  
        PORTB = u2;  
        delay(25);  
        PORTB = r2;  
        delay(180);  
    }  
}  
  
void sec4Direccional () { //SECUENCIA 4  
    int r = B00100000, r2 = B00000100;  
    int c = B00010000, c2 = B10000000;  
    for(int x=0; x<4; x++){  
        int res = 0, res2 = 0;  
        res = r2<<x;
```



```
    res2 = c2>>x;
    PORTD = res2;
    PORTB = res;
    delay(100);
}
for(int x=0; x<4; x++){
    int res = 0,res2 = 0;
    res = r>>x;
    res2 = c<<x;
    PORTD = res2;
    PORTB = res;
    delay(100);
}
}

void sec5inter(){ //SECUENCIA 5
int b1 = B11101000, b3 = B10000000,d1 = B00111010, d3 = B00100000;
int b2 = B01110100, b4 = B00010000,d2 = B00011101, d4 = B00000100;
int res=0, res2=0,res3=0,res4=0, count=0;
if(count == 0){
    count = 1;
    res = b1&b2;
    res3 = d1&d2;
    PORTD = res;
    PORTB = res3;
    delay(200);
}
if(count == 1){
    count = 0;
    res2 = b3|b4;
    res4 = d3|d4;
    PORTD = res2;
    PORTB = res4;
    delay(200);
}
}

void apagado (){ //FUNCION DE APAGADO
    PORTB = B00000000;
    PORTD = B00000000;
}

void btnf (){ // FUNCION DE LA INTERRUPCION
    pos = true;
}
```

d. Prototipo.

Una vez ensamblando todas las piezas de la carcasa, las piezas de lámina, el acomodo de la PCB dentro de la carcasa y el sellado de la misma para evitar la filtración del agua se muestran en las figuras 3.2.3, 3.2.4, 3.2.5 y 3.2.6.

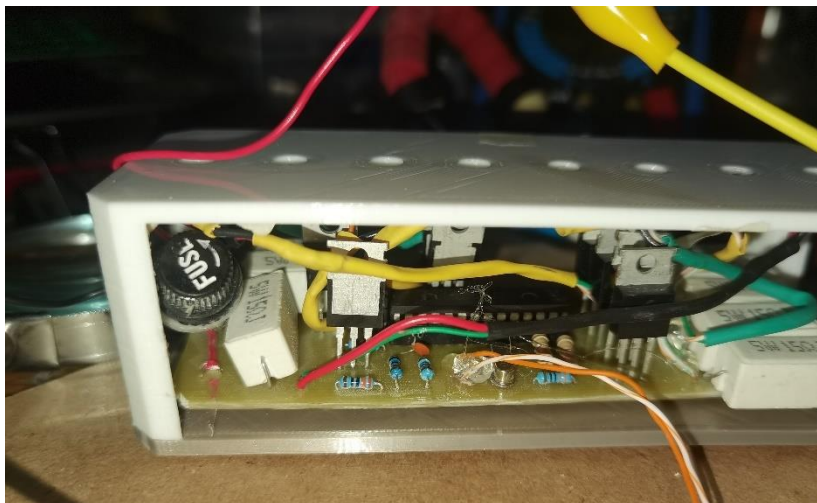


Figura 3.2.3 PCB dentro de la carcasa (2022).

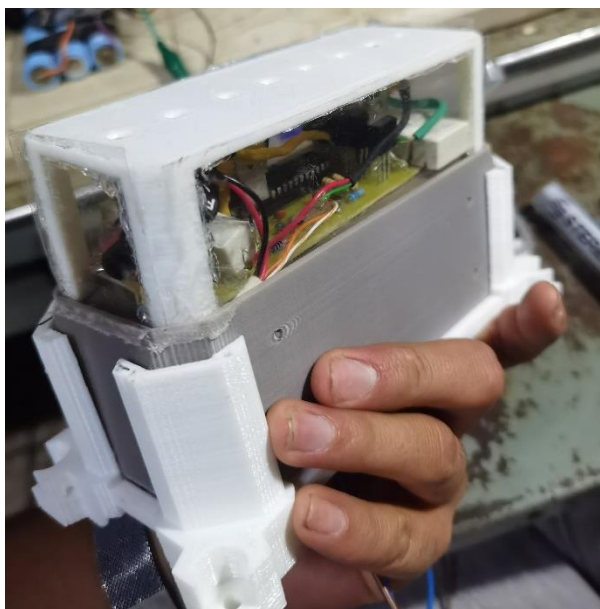


Figura 3.2.4 Ensamble de las piezas (2022).

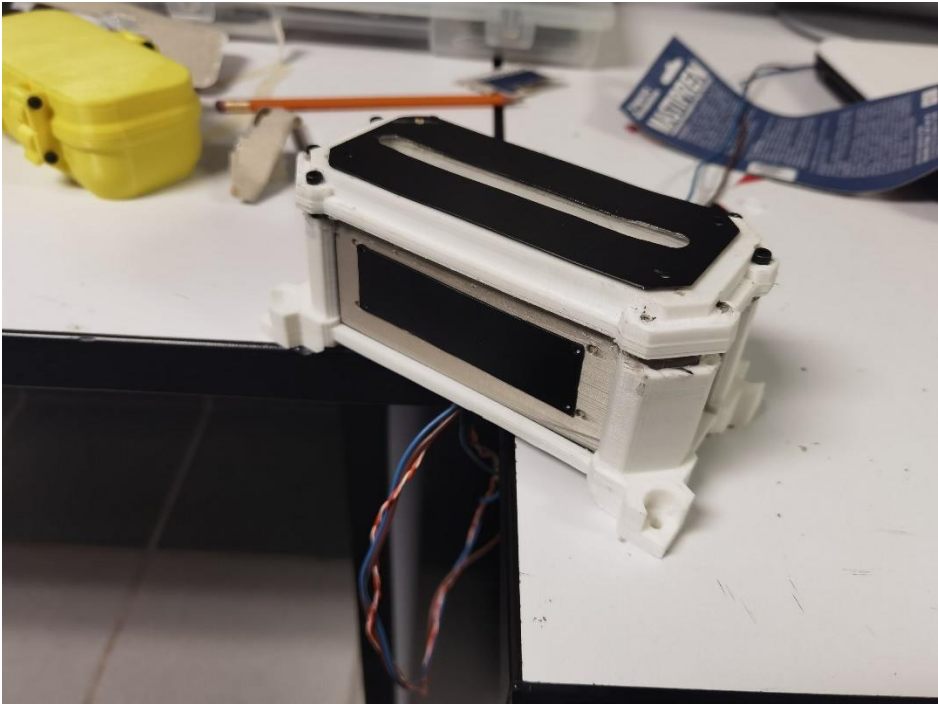


Figura 3.2.5. Prototipo 1.0 (2022).



Figura 3.2.6. Prototipo Encendido (2022).

e. Análisis de costos.

CANTIDAD	ARTÍCULO	PRECIO UNITARIO	TOTAL
1	AT MEGA 328P PU	68.09	68.09

10	LEDs de Potencia 3 Watts 3.3V	8.4	84
8	TIP41C	4.31	34.81
11	Resistencias	1	11
8	Resistencias 10 Watts	2.59	20.72
1	LM7805	4.74	4.74
1	Fusible	5	5
1	Porta fusible	13	13
1	Switch	6.90	
1	PLA p/kilo	350	350
2	Capacitores	4.29	8.58
1	Oscilador de Cristal	8.62	8.62

4. Conclusiones:

Los registros por puertos de Arduino nos ayudan mucho para realizar un control más dinámicos de los pines, el Puerto D nos permite controlar los pines del 0 al 7 mientras que el puerto B controla los pines del 8 hasta el 13 y dos mas que van al cristal del Arduino que en este caso utilizamos un Atmega328p y es del que hablamos, cada puerto se refiere a bits y es como hace uso de estos, cada puerto se trabaja con 8 bits, es decir, 8 pines del Arduino.

Los operadores Bitwise nos ayudan para cambiar los estados de los pines como si estos fueran compuertas lógicas, teniendo disponibles XOR, AND, OR, NOT y retroceder y avanzar posiciones de los bits.

En Arduino podemos inicializar/declararlos colocando $DDR(B-D) = B$ (es un indicativo) según el registro que se desee, para enviarle las indicaciones por medio de $PORT(B-D) = 00000000 B$ (es un indicativo) 00000000 seguido de los pines representados por 8 bits, 0 es estado bajo y 1 es estado alto.

Combinando los bitwise y los registros de puertos podemos lograr controlar los pines digitales para por ejemplo crear el proyecto de este documento, una barra de leds con secuencias, pero de igual forma se puede hacer automatizaciones.

Bibliografía:

- Aguilar, F. L. (2014). Variable. *DEV*.
- Arduino. (2014). Bitwise en Arduino IDE. *Arduino*.
- Argentina, R. (2017.). Manipulación de puertos. *Robots Didácticos*.
- ATS. (2016). C++. *Lenguajes De Programación*.
- Auto, H. (2020). Fusible. *Hello Insurance Group*.
- Brook, R. (2002). El lenguaje de programación C: diseño e implementación de programas. *El mundo de la programación*.
- Chikkerur, D. W. (2010). Practical Programming in C.
- Cinjordiz, C. (2018). Oscilador de cristal. *INFOOTEC*.
- Electronics, U. (2014). TIP41C.
- Martinez, R. (2017). AT MEGA 328P-PU. *UNIT Electronics*.
- MDN. (2017). bitwise (NOT). *Developer Mozilla*.
- MDN. (2017). bitwise (OR). *Developer Mozilla*.
- Mendoza, E. (2021). Capacitor. *Mecatrónica LATAM*.
- News, E. (2017). Resistencia. *Fluke*.
- Newsletter, I. (2015). LM7805 Todo sobre el regulador de tensión. *Hardware Libre*.
- Rojas, M. (2018). Historia del Led. *Iliminet*.
- Siled. (2016). LEDs de Potencia. *UNIT Electronics*.
- Vision. (2017). Los autos del futuro: iluminación LED en la industria automotriz. *VISION DIGITAL*.