



Subdirección Académica
Departamento de Sistemas y Computación
Ingeniería en Sistemas Computacionales
Semestre: Enero - Junio 2017
Materia: Sistemas Programables (3SC8A)

Nombre del tema:
Practica 11

Nombre del alumno:
Salcedo Morales José Manuel (13211419)

Nombre del catedrático:
Ingeniero Luís Alberto Mitre Padilla

Índice

1	Introducción	3
2	Componentes utilizados	3
3	Marco Teórico	4
4	Desarrollo	5
4.1	Imagenes	6
4.2	Diseño	13
4.3	Codigo	18
5	Conclusión	21

1 Introducción

En esta practica, se utiliza un arduino para controlar la uliminacion de unos led's en conjunto con el muestreo de numeros en una pantalla de 7 segmentos.

Esto, controlado por un fototransistor.

2 Componentes utilizados

- Arduino
- Cables Jumper
- Fuente de alimentacion para arduino
- Resistencias
 - 1KΩ
 - 2KΩ
- Pantalla de 7 segmentos
- Optointerruptor Itr8102

3 Marco Teórico

- Arduino: Arduino se refiere a una plataforma o placa de electrónica de código abierto y al software utilizado para programarlo. Arduino está diseñado para hacer la electrónica más accesible a los artistas, diseñadores, aficionados y a cualquiera interesado en la creación de objetos interactivos o entornos. Un tablero de Arduino se puede comprar pre-ensamblado o, porque el diseño de hardware es de código abierto, construido a mano. De cualquier manera, los usuarios pueden adaptar las tablas a sus necesidades, así como actualizar y distribuir sus propias versiones.
- Optointerruptor: Este sensor se compone de un emisor infrarrojo en un vertical y de un detector blindado del infrarrojo en el otro. Al emitir un haz de luz infrarroja desde una posición vertical a la otra, el sensor puede detectar cuando un objeto pasa entre los montantes, rompiendo la viga. Se utiliza para muchas aplicaciones, incluyendo interruptores ópticos de límite, dispensación de pellets, detección general de objetos, etc...
- Pantalla de 7 segmentos: La pantalla de 7 segmentos, también escrita como "pantalla de siete segmentos", consta de siete LED (de ahí su nombre) dispuestos en forma rectangular como se muestra. Cada uno de los siete LEDs se denomina segmento porque ilumina las formas de segmento de un dígito numérico (tanto Decimal como Hex) que se mostrarán. Un 8º LED adicional se utiliza a veces dentro del mismo paquete, permitiendo así la indicación de un punto decimal (DP) cuando dos o más pantallas de 7 segmentos están conectadas entre sí para mostrar números mayores de diez.

4 Desarrollo

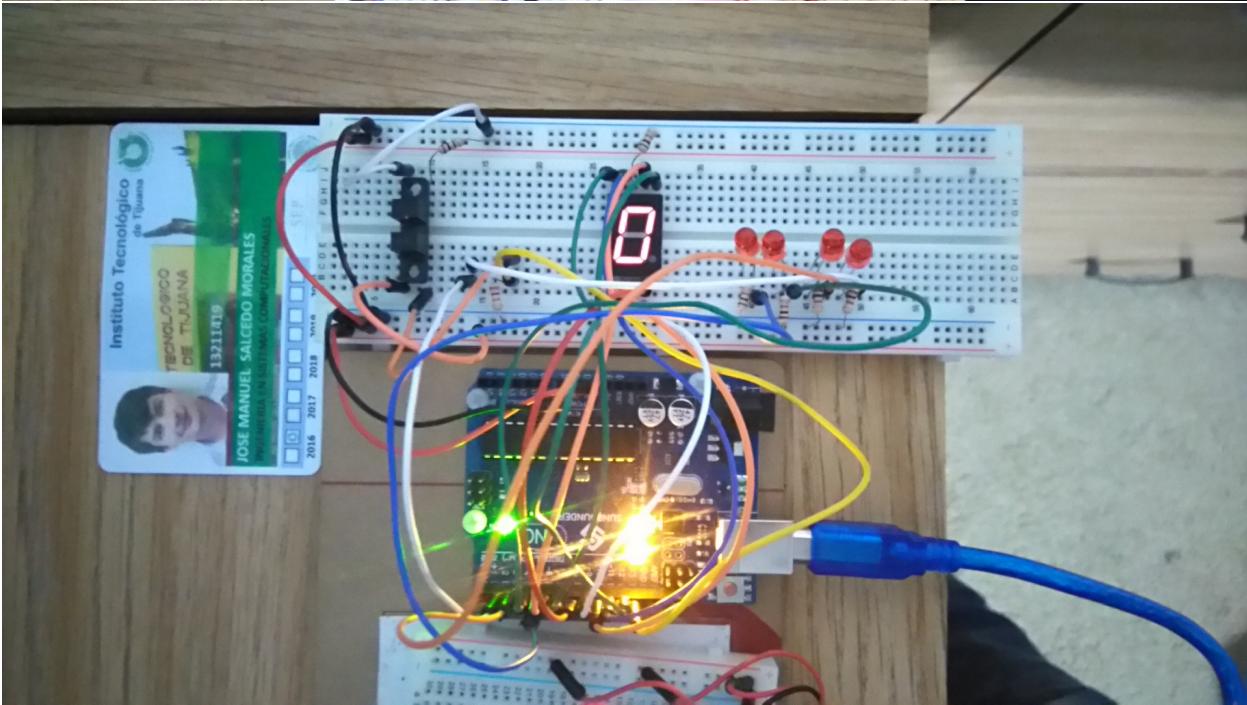
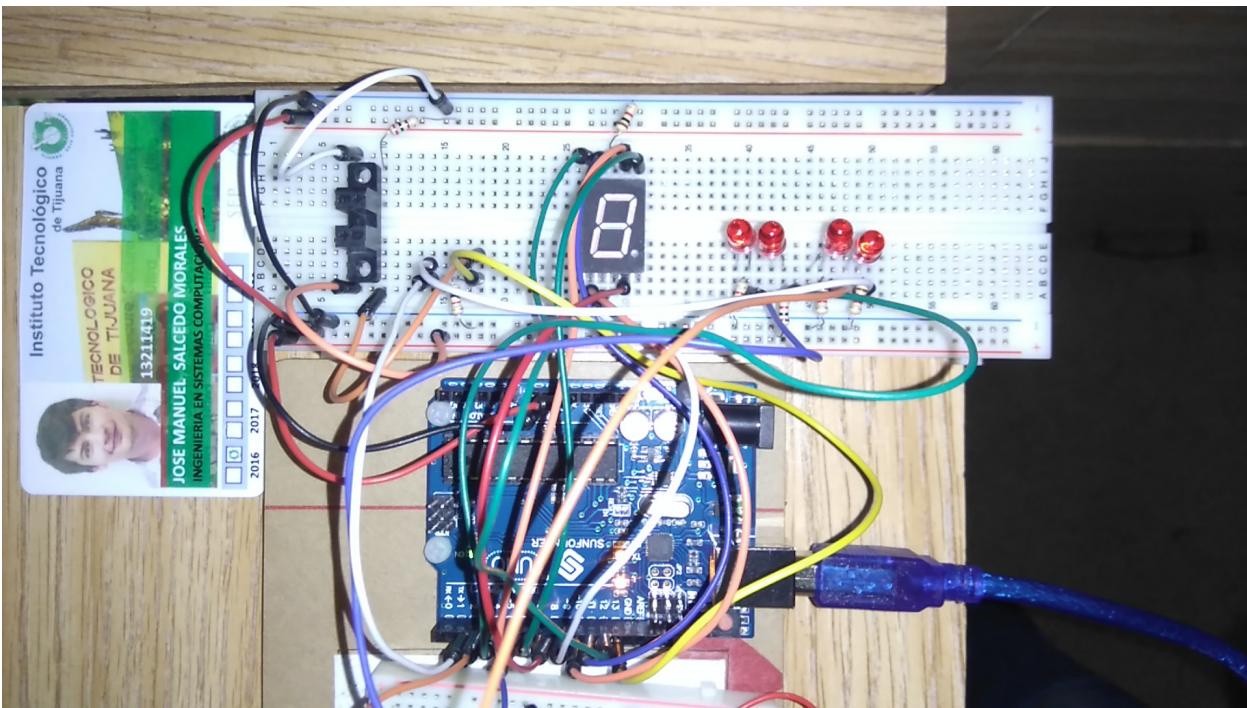
La principal funcion de la pantalla de 7 segmentos es mostrar un numero (del 0 al 9).

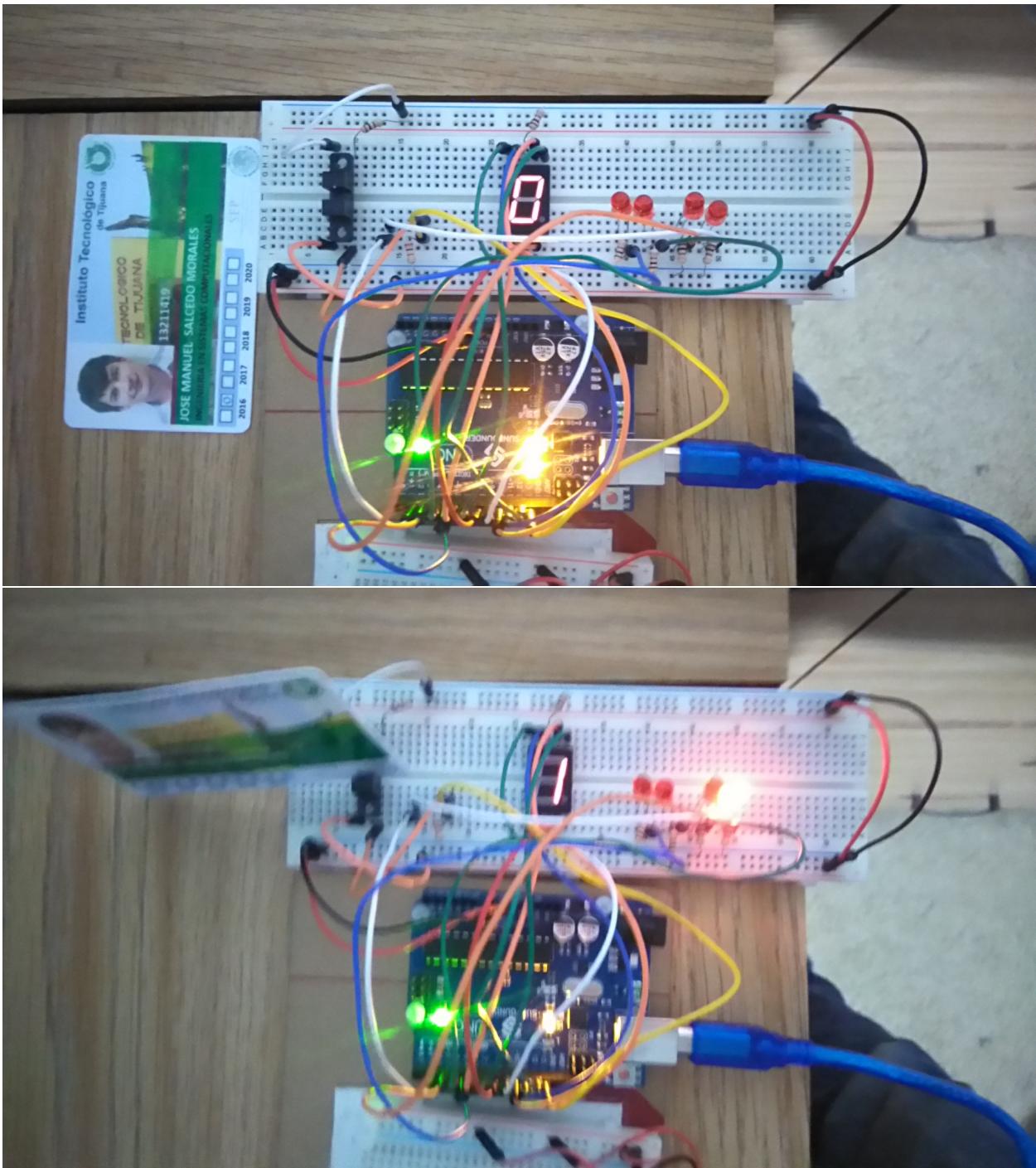
Igual que los 4 led's que existen, solo que en binario.

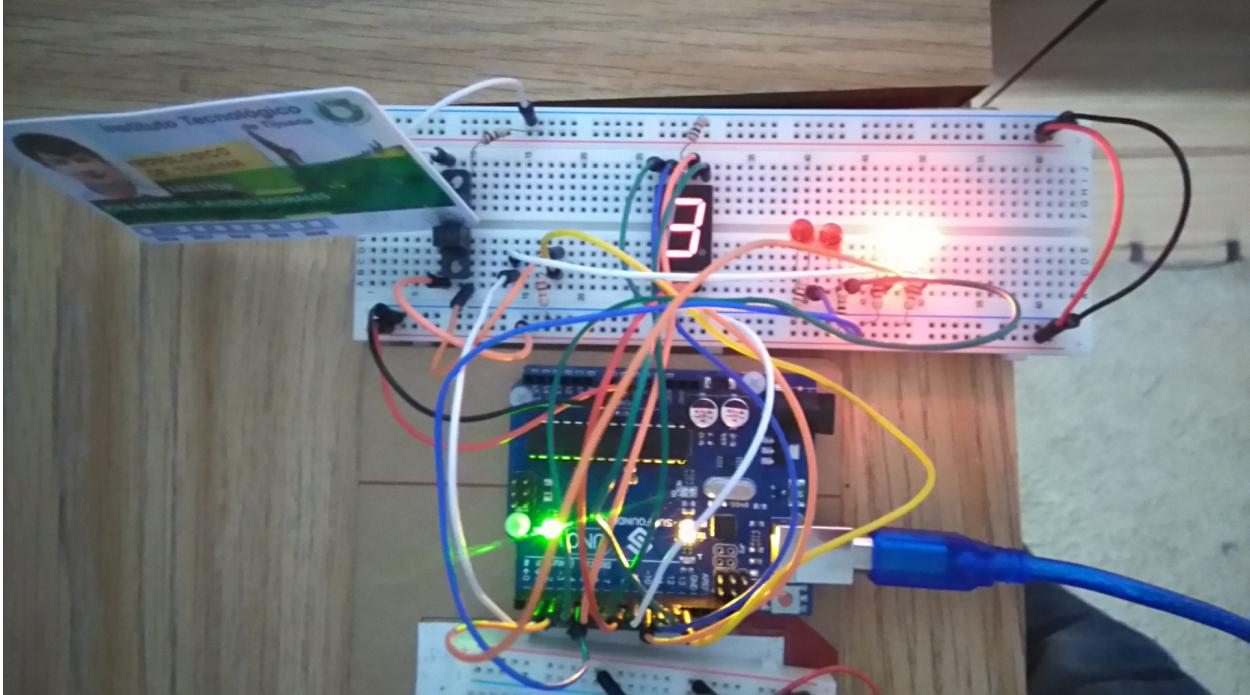
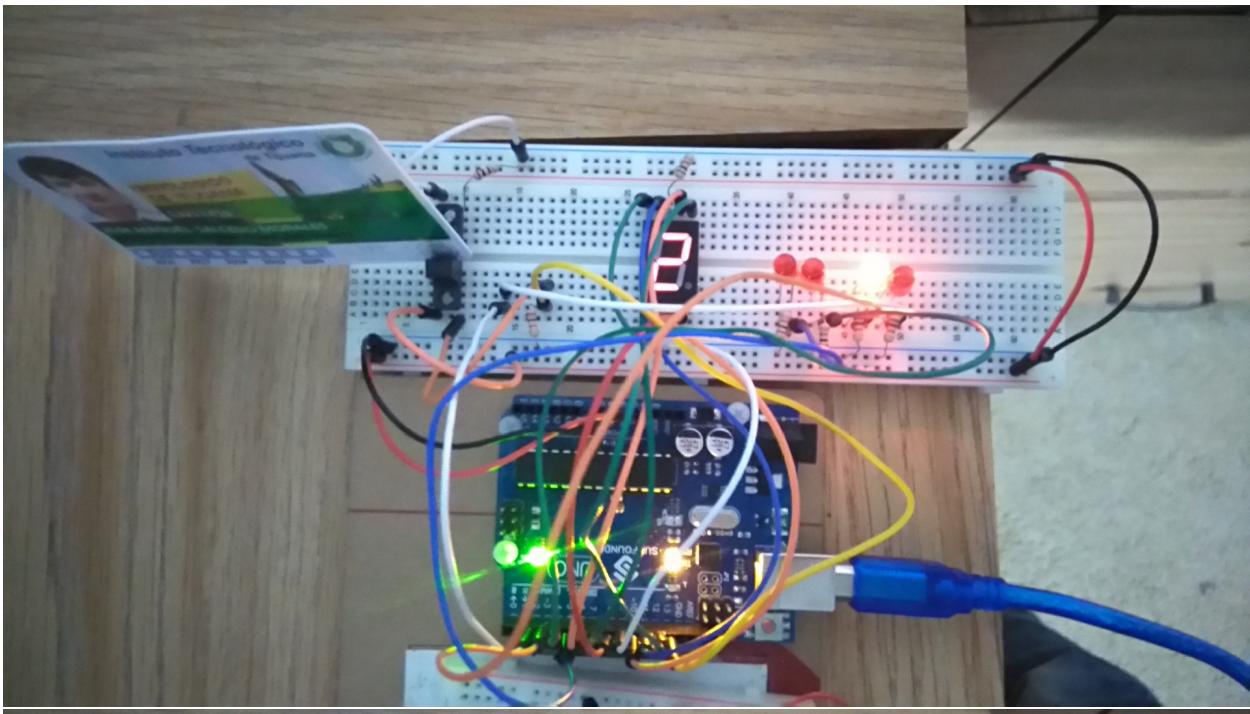
Cada vez que la transmision de luz se interrumpa, incrementara el numero por 1, esto hasta regresar al numero 0.

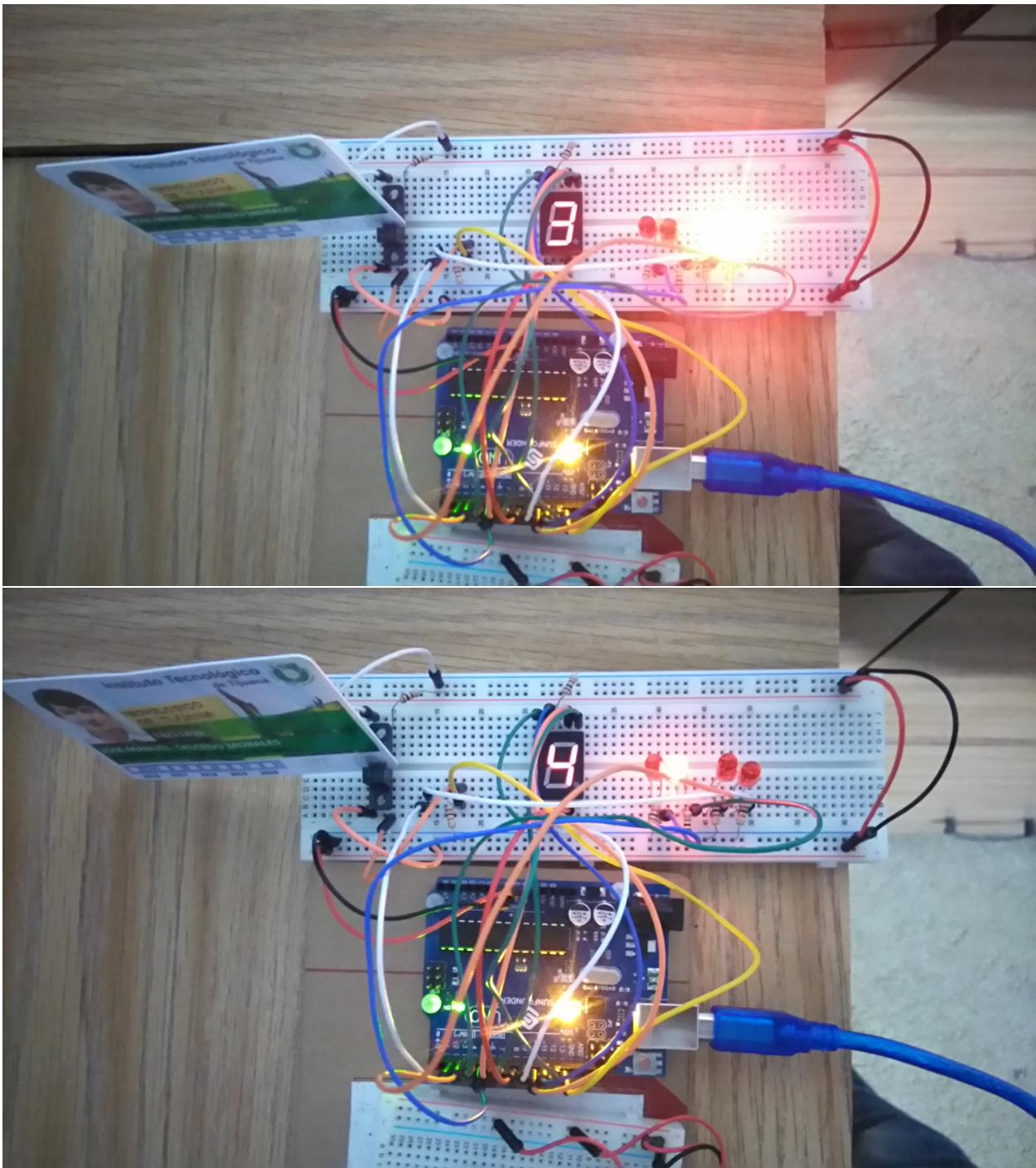
Todo esto controlado por el arduino, conteniendo la logica para controlar cada led dentro de la pantalla al mismo tiempo que los led's separados de la pantalla para indicar dada numero en binario.

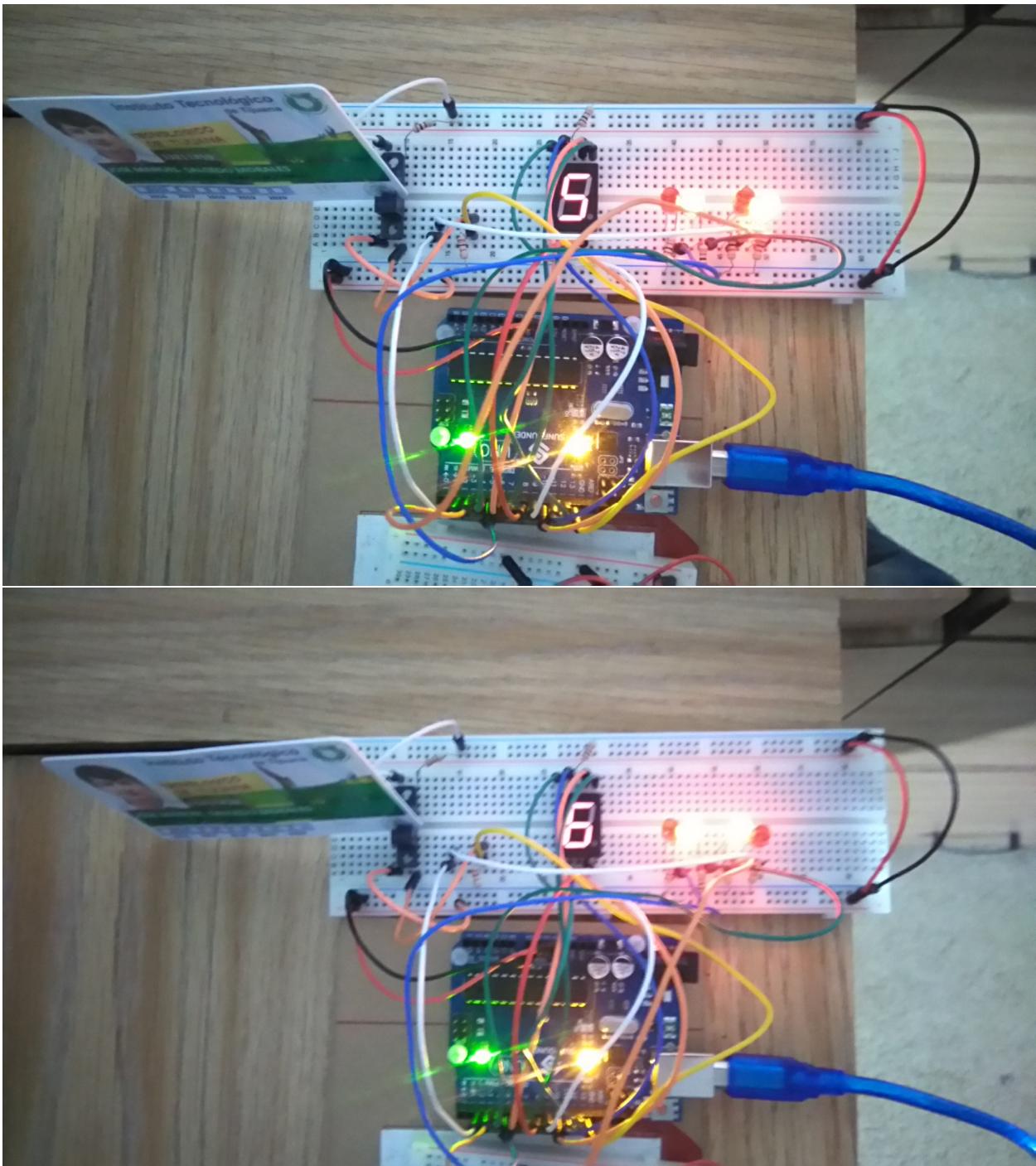
4.1 Imagenes

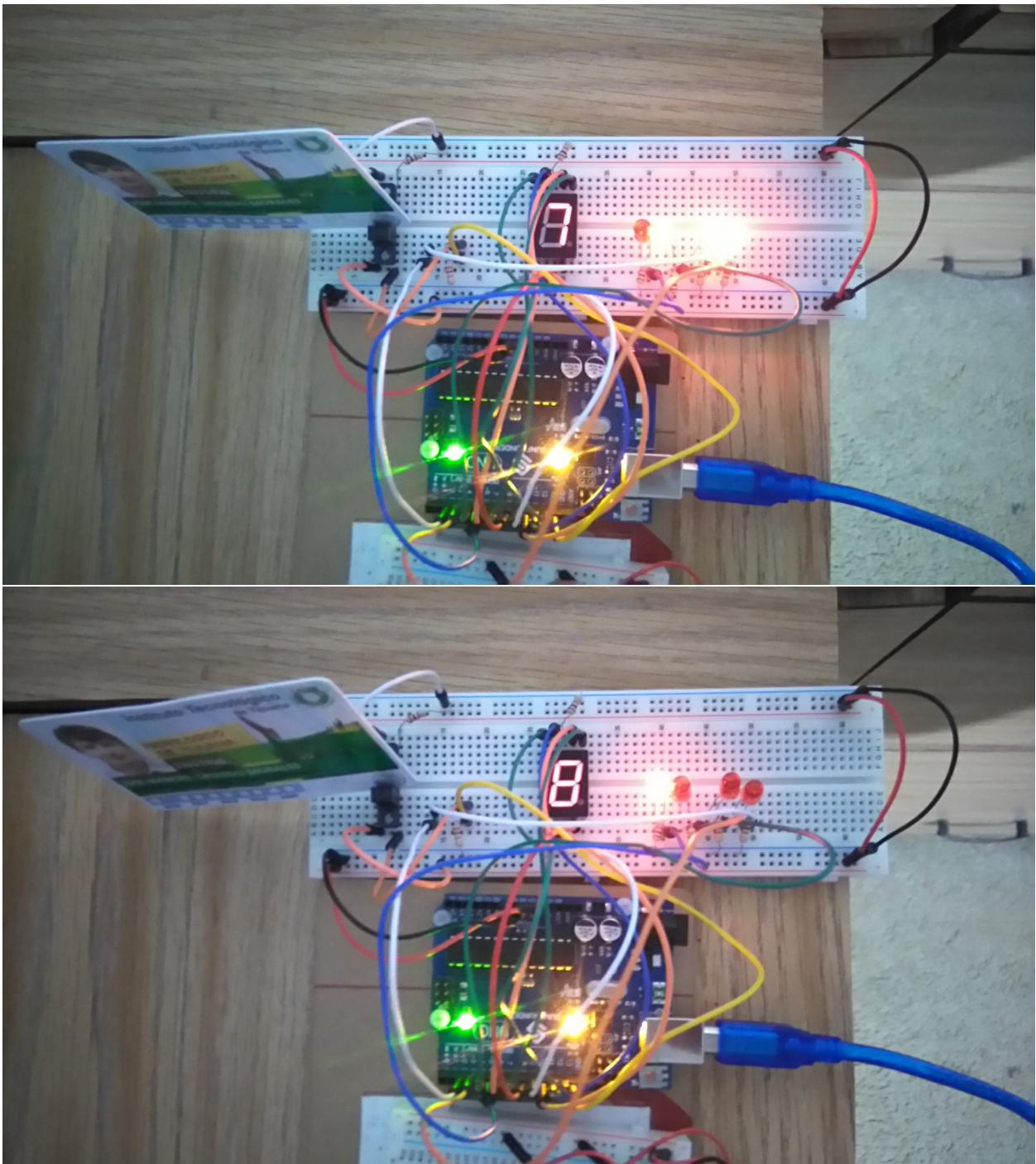


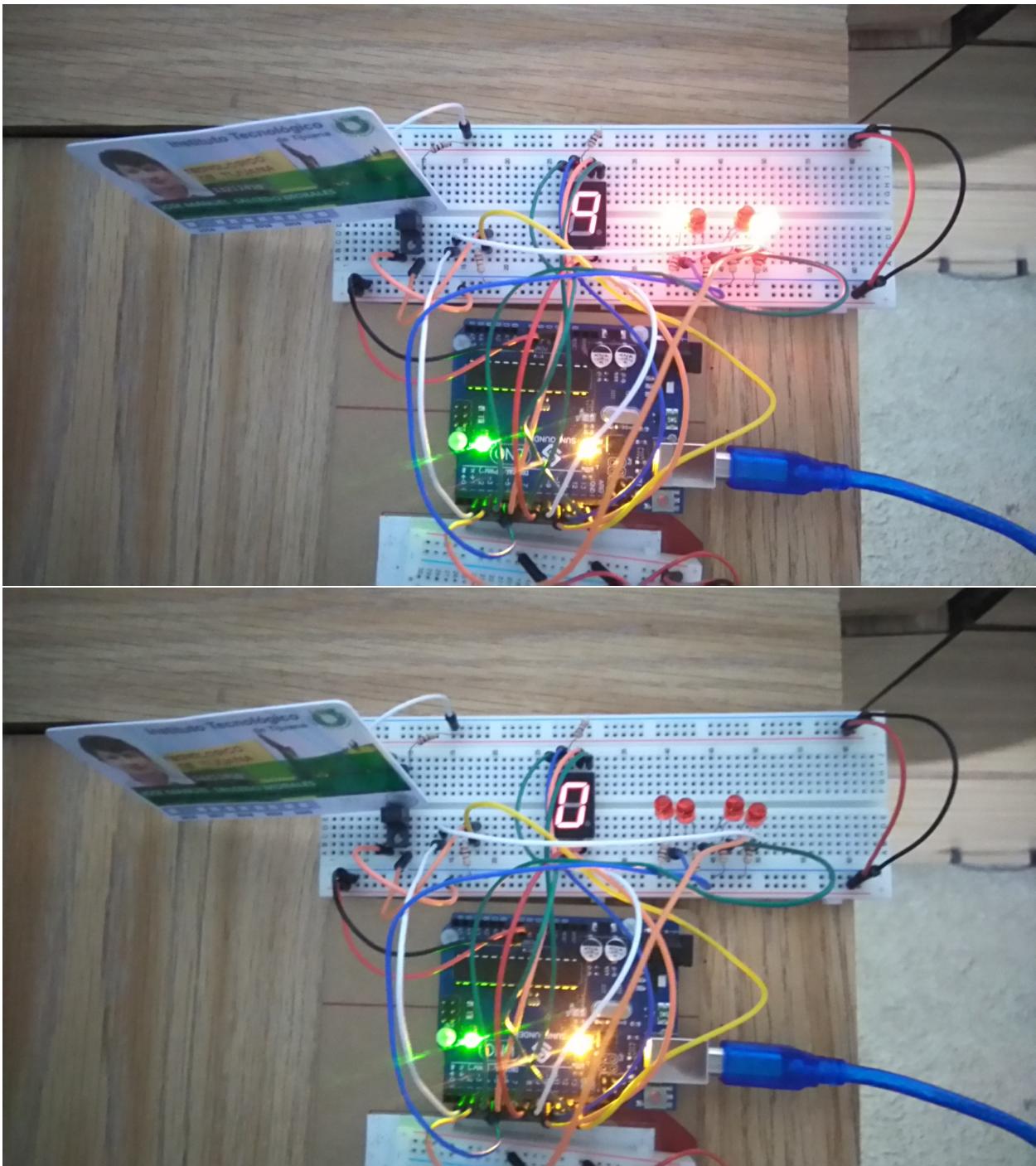




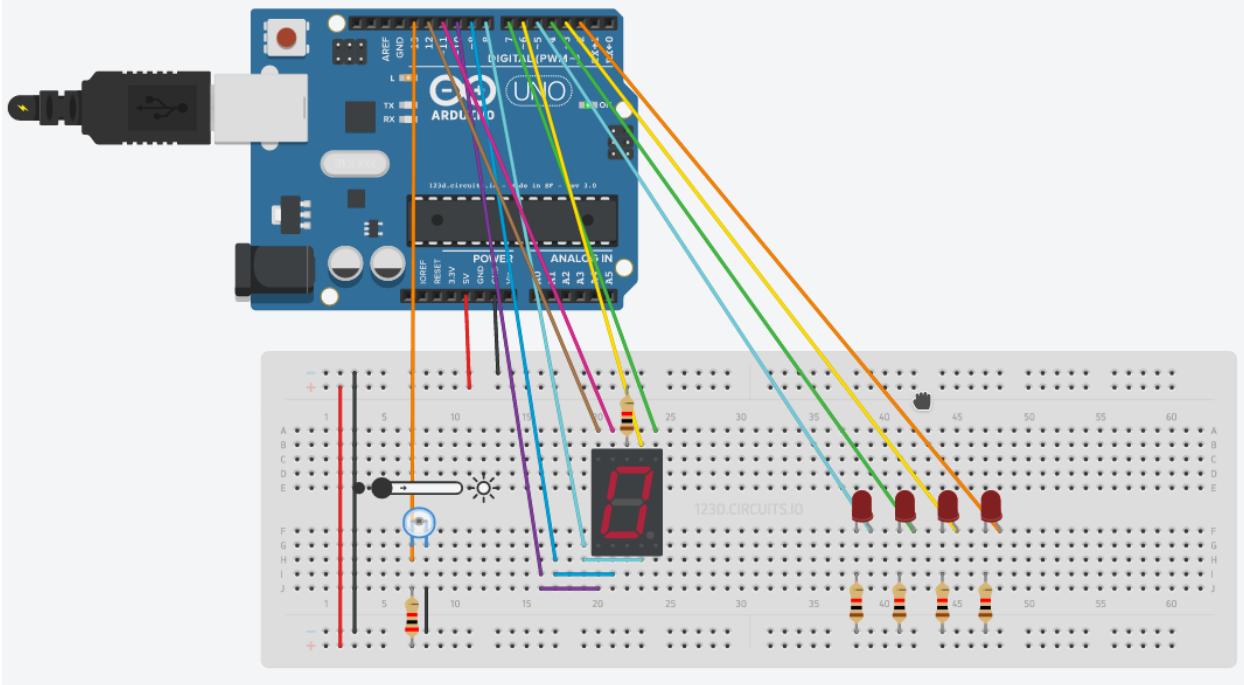
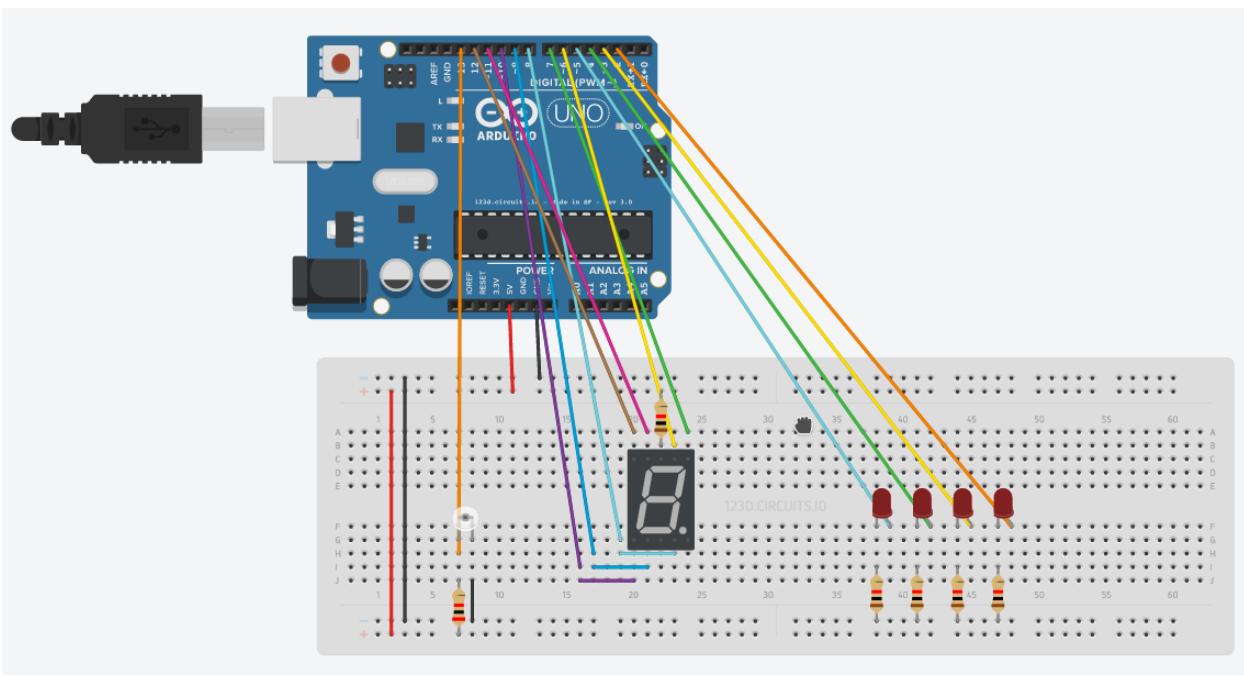


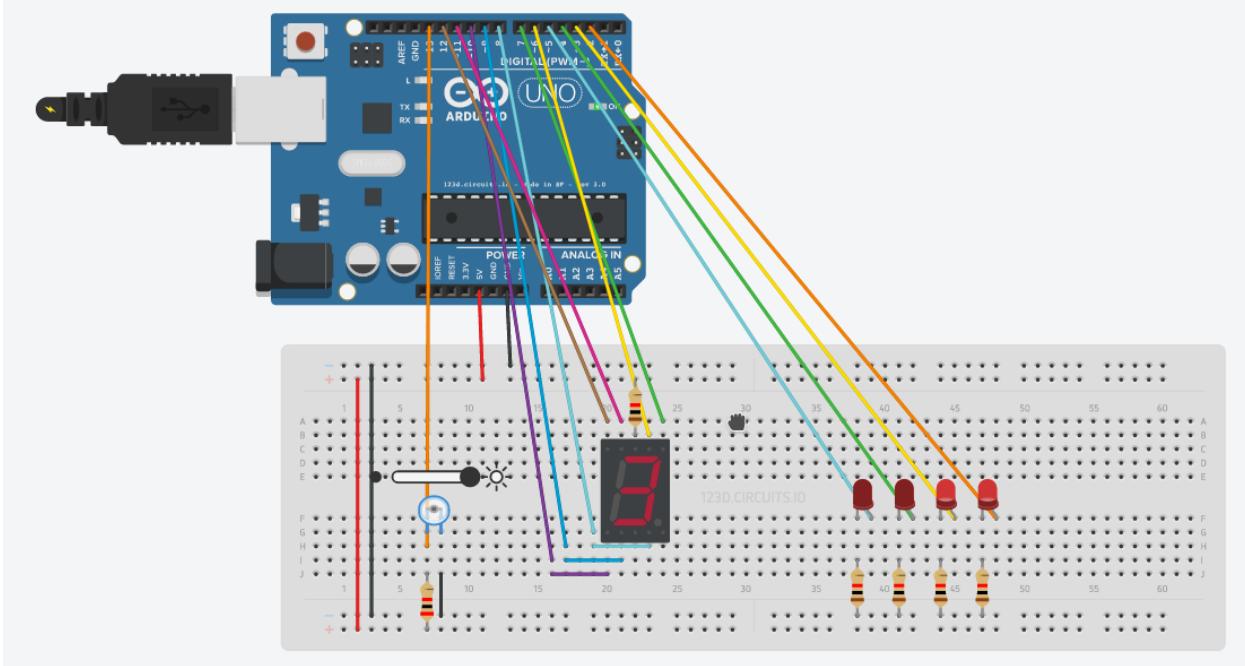
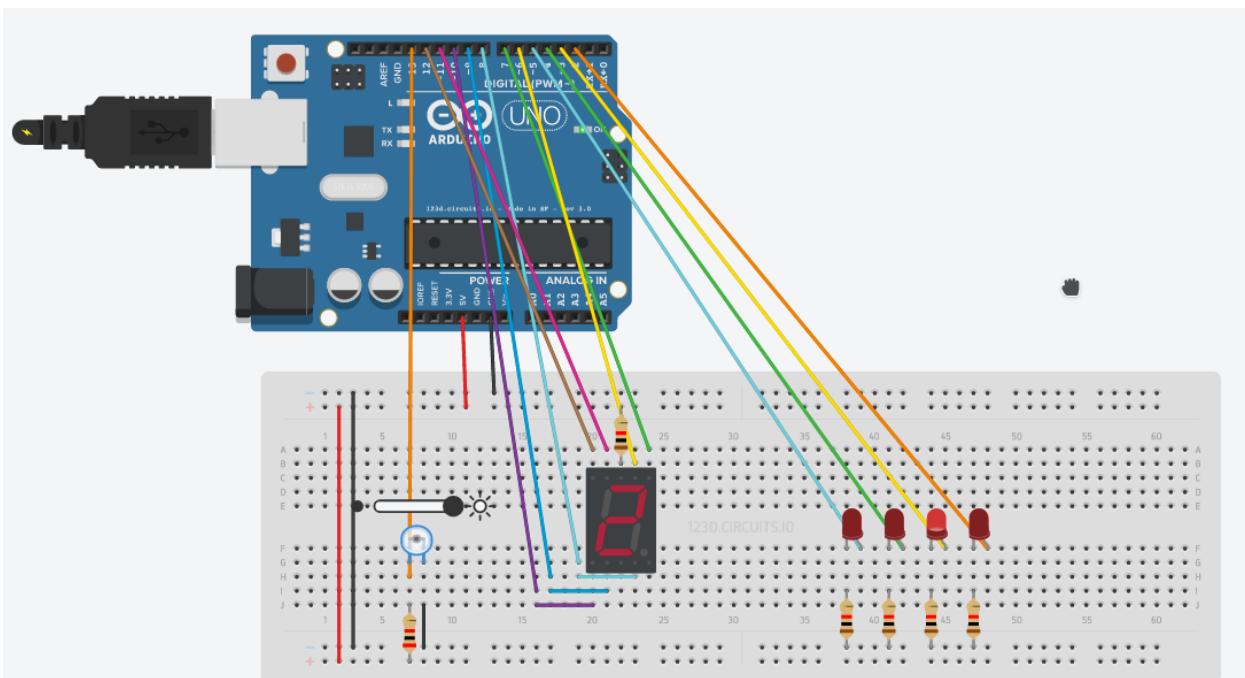


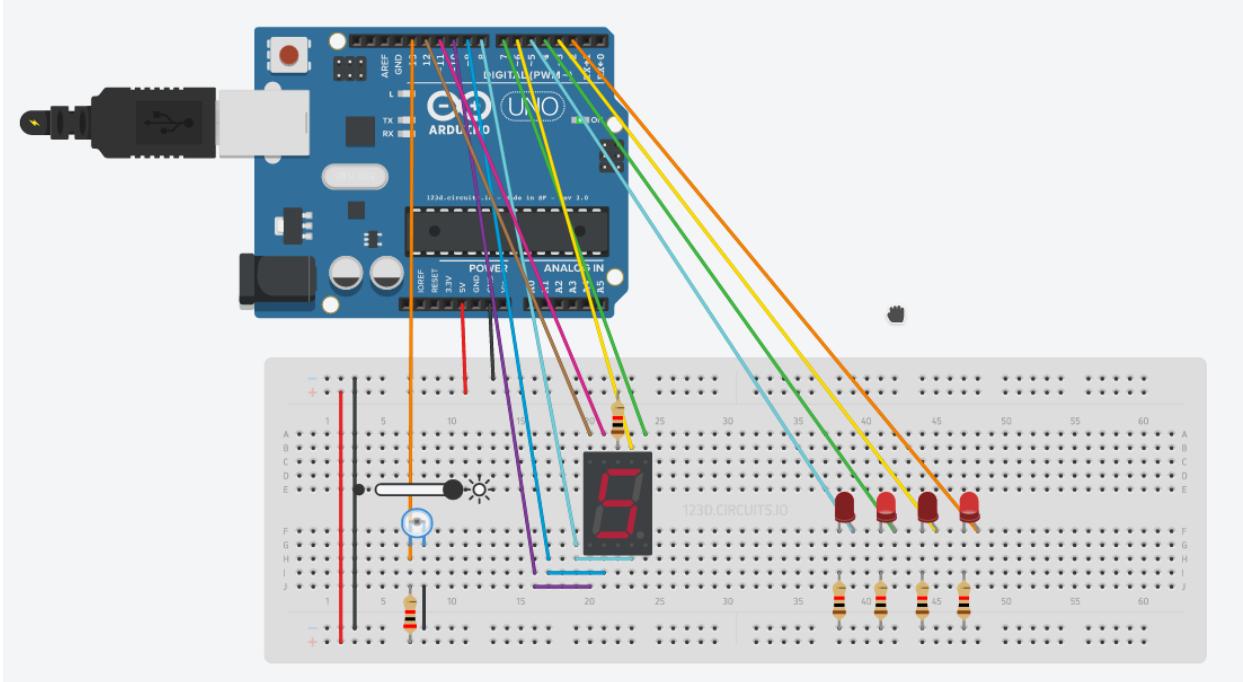
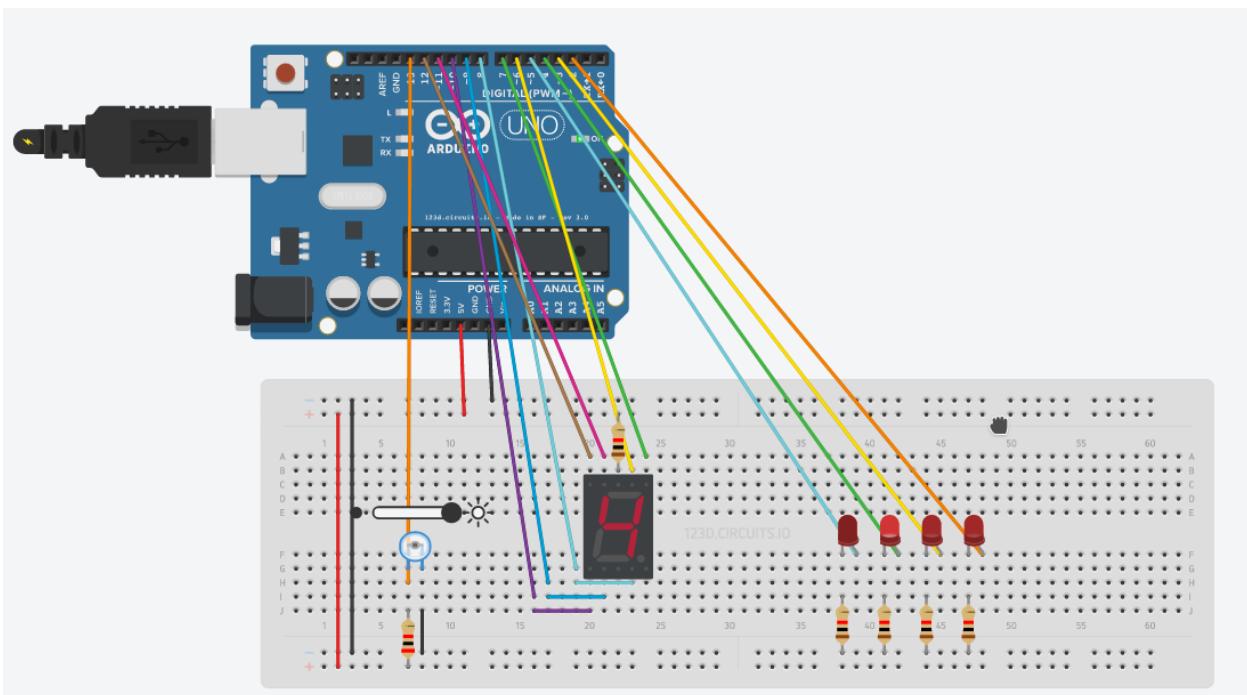


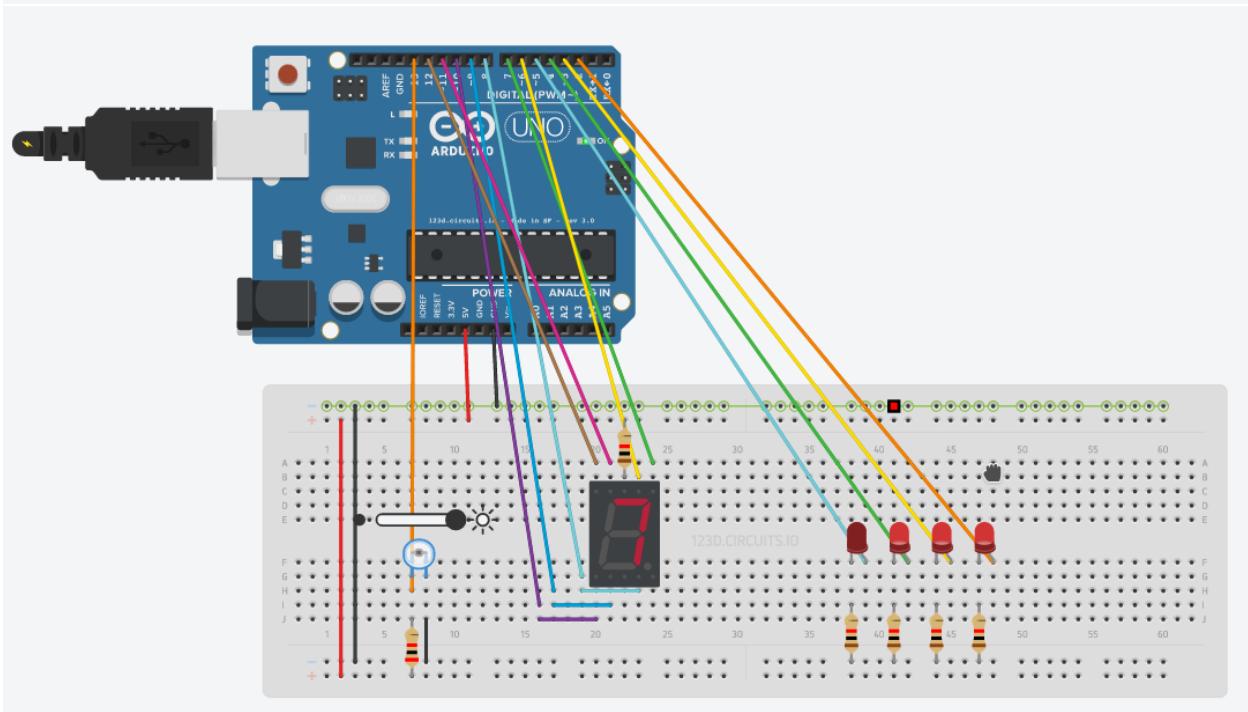
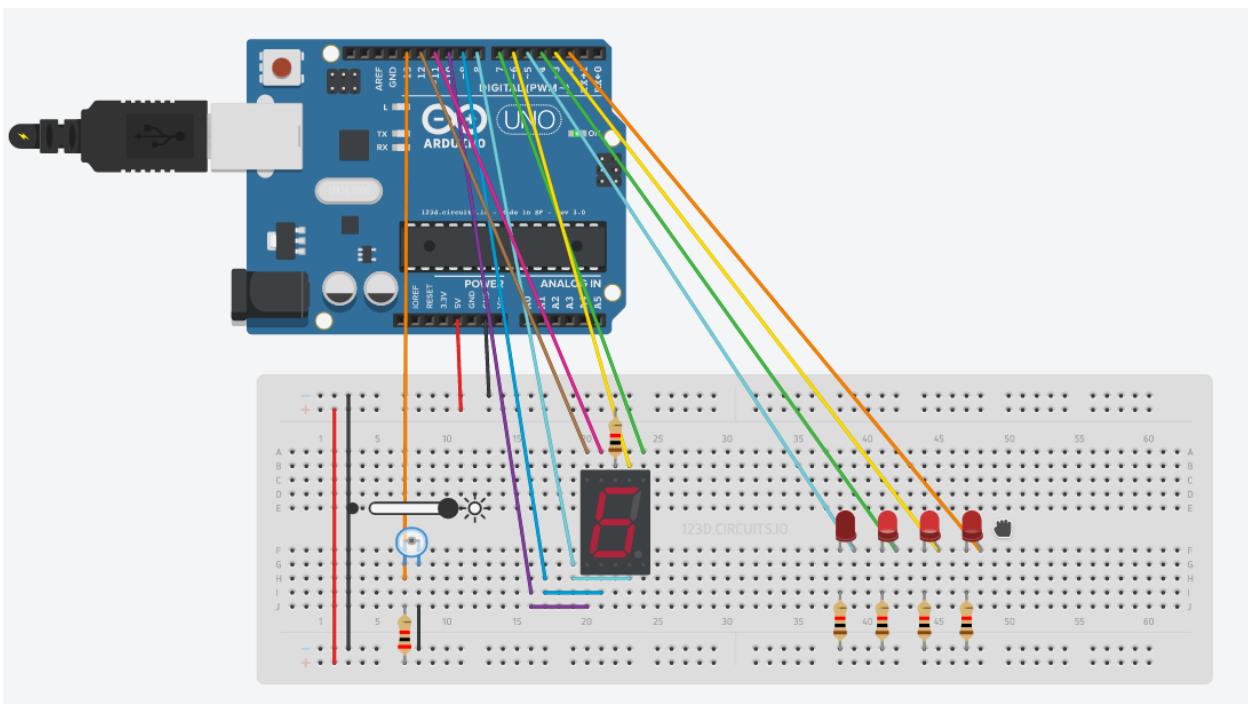


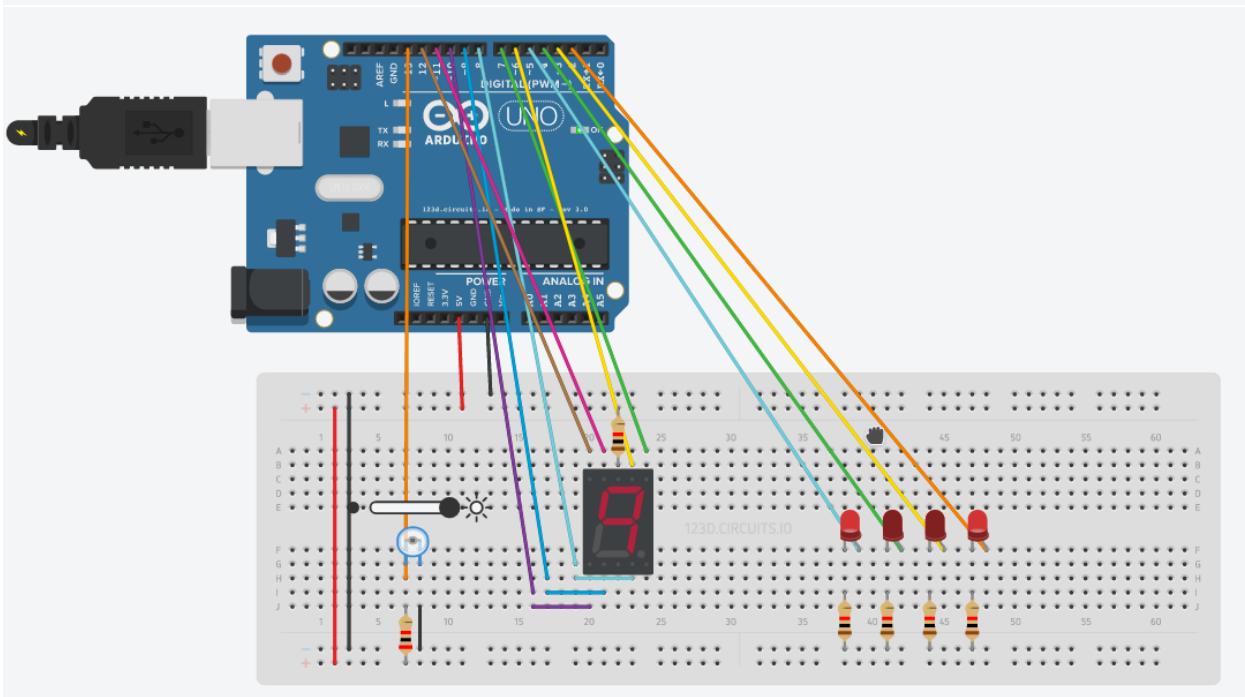
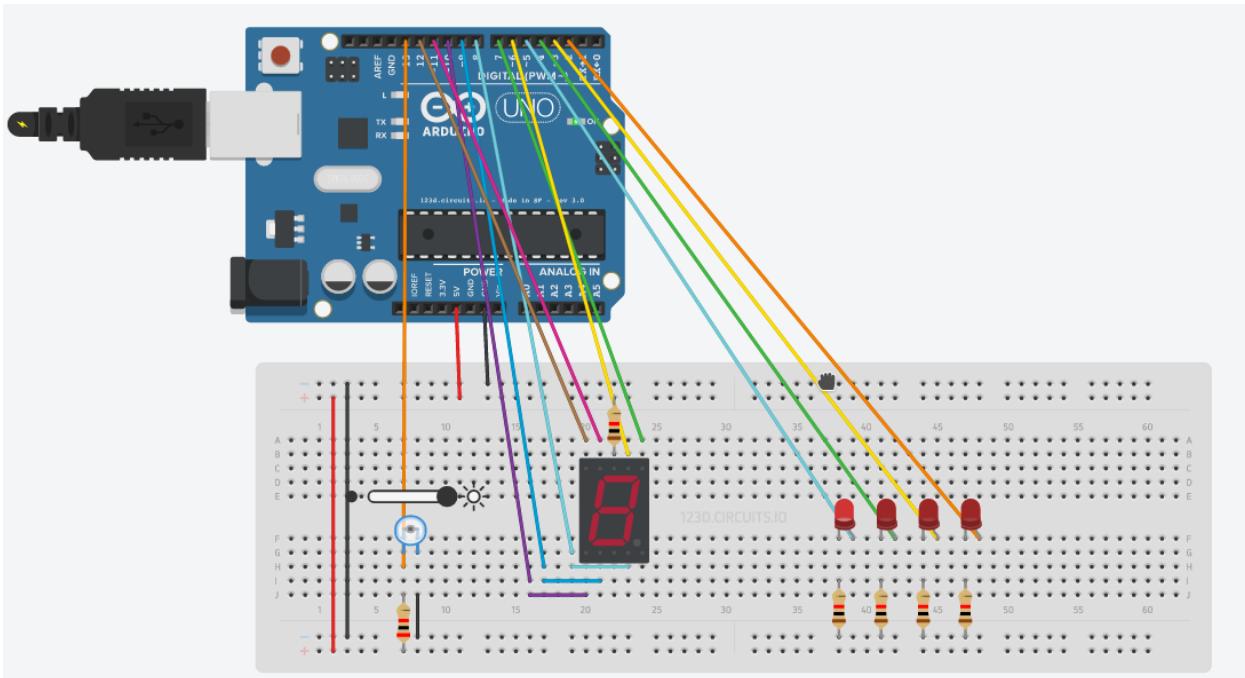
4.2 Diseño











4.3 Código

```

// / [ ] [ ] / \ / [ ] [ ] [ ] / [ ] [ ] [ ] / [ ] [ ]
// [ ] / [ ] \ D ) ( ( [ ] [ ] ) / ( [ ] [ ] ) ( ) ( ) [ ]
// [ ] [ ] / [ ] / \ / [ ] [ ] [ ] / [ ] [ ] / [ ] [ ] / [ ]
//
// ITT 8Sem SP Practica 11
//
// Made by Salcedo Jose
// License: CC-BY-SA 3.0
// Downloaded from: https://circuits.io/circuits/4447129-itt-8sem-sp-practica-11

// DATOS ABSTRACTOS.
enum ledDisplay {
    LED_A = 6,
    LED_B = 7,
    LED_C = 8,
    LED_D = 9,
    LED_E = 10,
    LED_F = 11,
    LED_G = 12
};

// CONSTANTES GLOBALES.
const unsigned int PIN_DISPLAY_INICIAL = 6;
const unsigned int PIN_DISPLAY_FINAL = 12;
const unsigned int pinLed[] = {2, 3, 4, 5};
const unsigned int CANTIDAD_LEDS = 4;
const unsigned int PIN_OPTOINTERRUPTOR = 13;

// Obtener un numero en binario.
String ObtenerNumeroEnBinario(unsigned long numero)
{
    // Maximo de bytes a usar para un numero binario.
    const int BYTES_BINARIO_MAXIMO = 32;

    // Definir texto del numero binario a retornar.
    String numeroBinario = "";

    // Si el numero es igual o menor a 0, definir el numero como 0.
    if (numero <= 0) {
        numeroBinario = "0";
    } else {
        // Repetir por cada byte posible en un numero binario.
        for(int numeroByte = BYTES_BINARIO_MAXIMO; numeroByte >= 0; numeroByte--) {
            // Obtener el numero de potencia de acuerdo al numero del byte actual.
            const unsigned long numeroPotenciaActual = (unsigned long)(round(pow(2, numeroByte)));

            // Si el resultado de la potencia obtenida es menor o igual al numero a convertir,
            // a esta posicion de byte le pertenece un 1.
            if (numeroPotenciaActual <= numero) {
                numeroBinario += "1";

                // Restar la potencia actual al numero para no repetir el mismo byte.
                numero = numero - numeroPotenciaActual;
            }
            // Si el numero no es 1, si ya hay numeros en el texto binario agregar un 0.
            } else if (numeroBinario != "") {
                numeroBinario += "0";
            }
        }
    }

    return numeroBinario;
}

// Prender una colección de led's de acuerdo a un numero en binario.
void RepresentarNumeroBinarioEnPines(unsigned int numero, const unsigned int colecciónPines[], const unsigned int cantidadPines)
{
    const String numeroBinario = ObtenerNumeroEnBinario((unsigned long)numero);

    // Obtener la cantidad de digitos en el numero binario.
    const unsigned int cantidadDigitos = numeroBinario.length();

    // Iniciar desde el ultimo digito.
    unsigned int digitoActual = cantidadDigitos - 1;
    // Recorrer todos los pines existentes.
    for (unsigned int pinActual = 0; pinActual < cantidadPines; pinActual++) {
        // Capturar el pin a encender.
        const unsigned int pinEncender = colecciónPines[pinActual];

        // Continuar si el numero del digito actual es menor
        // a la cantidad de pines permitidos a usar.
        // Y el pin actual es menor a la cantidad de digitos a representar.
        if (digitoActual < cantidadPines && pinActual < cantidadDigitos) {
            // Capturar el digito actual del numero binario.
            const char numeroDigito = numeroBinario.charAt(digitoActual);

            // Prender o apagar el pin de acuerdo al equivalente del numero binario.
            if (numeroDigito == '1') {
                digitalWrite(pinEncender, HIGH);
            } else if (numeroDigito == '0') {
                digitalWrite(pinEncender, LOW);
            }

            digitoActual -= 1;
        } else {
            digitalWrite(pinEncender, LOW);
        }
    }
}

```

```

        }
    }

// Prender o apagar led's definidos en el display.
void ApagarLedsDisplay(bool A, bool B, bool C, bool D, bool E, bool F, bool G)
{
    // Definir los estados a aplicar.

    if (A == true)
        digitalWrite((byte)LED_A, HIGH);
    else
        digitalWrite((byte)LED_A, LOW);

    if (B == true)
        digitalWrite((byte)LED_B, HIGH);
    else
        digitalWrite((byte)LED_B, LOW);

    if (C == true)
        digitalWrite((byte)LED_C, HIGH);
    else
        digitalWrite((byte)LED_C, LOW);

    if (D == true)
        digitalWrite((byte)LED_D, HIGH);
    else
        digitalWrite((byte)LED_D, LOW);

    if (E == true)
        digitalWrite((byte)LED_E, HIGH);
    else
        digitalWrite((byte)LED_E, LOW);

    if (F == true)
        digitalWrite((byte)LED_F, HIGH);
    else
        digitalWrite((byte)LED_F, LOW);

    if (G == true)
        digitalWrite((byte)LED_G, HIGH);
    else
        digitalWrite((byte)LED_G, LOW);
}

// Representar un numero del 0 al 9 en un display de 7 segmentos.
void RepresentarNumeroEnDisplay(unsigned int numero)
{
    // Prender los leds del display de acuerdo al numero a representar.
    switch(numero) {
        case 0:
            ApagarLedsDisplay(false, false, false, false, false, false, true);
            break;

        case 1:
            ApagarLedsDisplay(true, false, false, true, true, true, true);
            break;

        case 2:
            ApagarLedsDisplay(false, false, true, false, false, true, false);
            break;

        case 3:
            ApagarLedsDisplay(false, false, false, false, true, true, false);
            break;

        case 4:
            ApagarLedsDisplay(true, false, false, true, true, true, false);
            break;

        case 5:
            ApagarLedsDisplay(false, true, false, false, true, false, false);
            break;

        case 6:
            ApagarLedsDisplay(false, true, false, false, false, false, false);
            break;

        case 7:
            ApagarLedsDisplay(false, false, false, true, true, true, true);
            break;

        case 8:
            ApagarLedsDisplay(false, false, false, false, false, false, false);
            break;

        case 9:
            ApagarLedsDisplay(false, false, false, true, true, false, false);
            break;

        default:
            ApagarLedsDisplay(true, true, true, true, true, true, false);
            break;
    };
}

// the setup routine runs once when you press reset:
void setup() {
    // Inicializar lectura de serial.
    Serial.begin(9600);

    // Definir pines de salida para leds.
}

```

```

unsigned int cantidadPinesLed = sizeof(pinLed);
for (byte led = 0; led < cantidadPinesLed; led++) {
    pinMode(pinLed[led], OUTPUT);
}

// Definir pines de salida para display.
for (unsigned int pinLedDisplay = PIN_DISPLAY_INICIAL; pinLedDisplay <= PIN_DISPLAY_FINAL; pinLedDisplay++) {
    pinMode(pinLedDisplay, OUTPUT);
}

// Definir pin de entrada para activacion por optointerruptor.
pinMode(PIN_OPTOINTERRUPTOR, INPUT);

// VARIABLES GLOBALES.
unsigned int valorOptointerruptorAnterior = 1;
byte pasadasOptointerruptor = 0;

// the loop routine runs over and over again forever:
void loop() {
    // Capturar valor del optointerruptor.
    unsigned int valorOptointerruptor = digitalRead(PIN_OPTOINTERRUPTOR);
    Serial.println(valorOptointerruptor);

    // Continuar si el valor digital del optointerruptor es 0
    // y el valor sea distinto al anterior.
    if (valorOptointerruptor == 0 &&
        valorOptointerruptor != valorOptointerruptorAnterior) {

        // Incrementar el numero de pasadas si no ah superado el numero maximo
        // a representar.
        if (pasadasOptointerruptor < 9)
            pasadasOptointerruptor += 1;
        else
            pasadasOptointerruptor = 0;

        // Representar el numero en los led's y en el display.
        RepresentarNumeroEnDisplay(pasadasOptointerruptor);
        RepresentarNúmeroBinarioEnPines(pasadasOptointerruptor, pinLed, CANTIDAD_LEDS);

        // Guardar el valor obtenido del optointerruptor.
        valorOptointerruptorAnterior = valorOptointerruptor;
    }
}

```

5 Conclusión

Desplegar informacion al usuario es de vital importancia para muchas (si no todas) las creaciones de software y/o hardware.

Con esta practica, se puede apreciar la complejidad de la cual es incluso mostrar solo los primeros 10 numeros decimales.

Pero con ello, un aprendizaje de como se realizan estas funciones.

Referencias

- [1] What is Arduino? - Definition from Techopedia. (n.d.). Retrieved March 26, 2017, from <https://www.techopedia.com/definition/27874/arduino>
- [2] Sparkfun. (n.d.). Photo Interrupter - GP1A57HRJ00F. Retrieved February 13, 2017, from <https://www.sparkfun.com/products/9299>
- [3] 7-segment Display. (n.d.). Retrieved April 25, 2017, from <http://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>