Sistemas con Microprocesadores

Práctica 2: Prácticas con *displays* de 7 segmentos

Resumen:

Utilizar *displays* de 7 segmentos conectados a Arduino, tanto de forma directa como multiplexada.

Entrega:

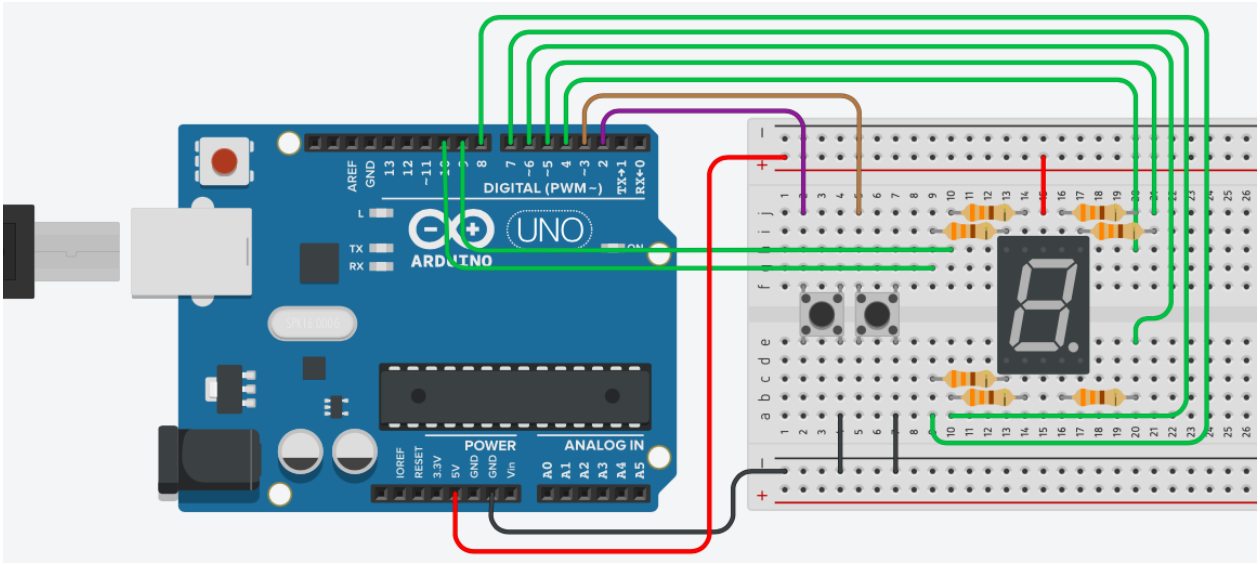
Hacer públicos los diseños Tinkercad de los ejercicios 1 y 2.
Entregar un PDF de 3 páginas aproximadamente que contenga:

- Nombre y apellidos
- Dirección web de los proyectos Tinkercad.
- Captura de pantalla del funcionamiento del contador.
- Captura de pantalla del funcionamiento del segundero con 2 dígitos.
- Cálculo del valor de la resistencia para los *displays* y de su potencia.
- Código fuente de los proyectos pegado como texto en el documento.
  (Antes de pegar el código fuente en el documento PDF lo colorearemos, por ejemplo, utilizando la página: https://highlight.hohli.com/?language=cpp)

Desarrollo:

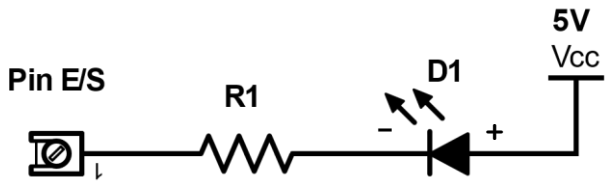
Esta práctica está dividida en 4 apartados:

1) Utilizar un *display* de 7 segmentos (de ánodo común) para realizar un contador de 0 a 9 con dos botones.

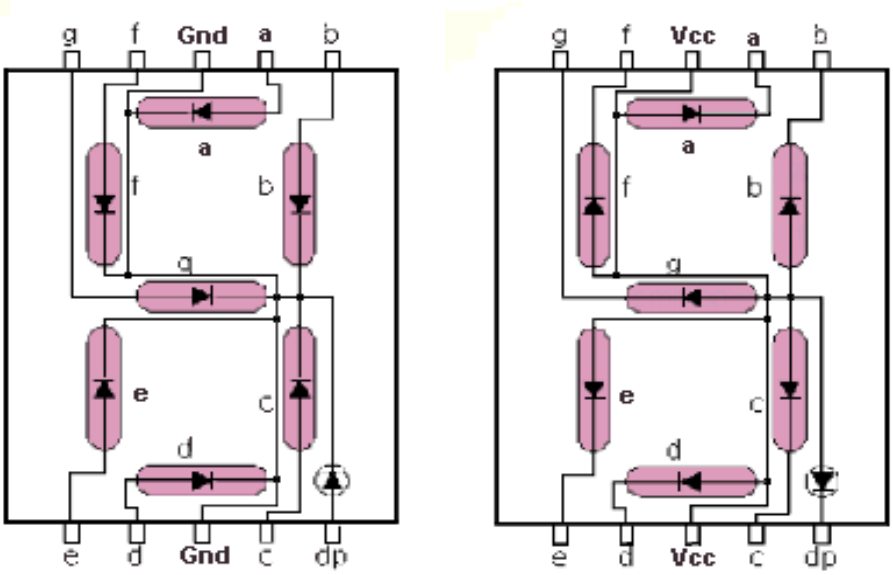


Implementaremos un contador de un dígito con un *display*, de forma que incremente su valor en una unidad cada vez que se pulse un botón y se decremente cuando se pulse el otro botón. Las patillas del microcontrolador a las que se conecten los botones deben ser configuradas como entradas con resistencias internas de *pull-up*.

Un *display* de 7 segmentos está compuesto de diodos emisores de luz (led). Cuando se aplica un voltaje suficientemente positivo en el terminal ánodo (+) con respecto al voltaje del cátodo (-) el diodo conduce la corriente, y en el caso de un led, emite luz.



Los *displays* de 7 segmentos de ánodo común y cátodo común son los que podemos encontrar como componentes comerciales. Son diodos led conectados mediante una configuración en la que todos los ánodos están unidos o bien todos los cátodos están unidos.

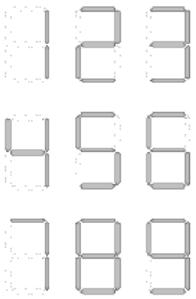


Cátodo común **Ánodo Común**

Mediante estas configuraciones podemos controlar cada segmento de forma independiente. Utilizaremos un *display* de 7 segmentos de ánodo común (seleccionamos el tipo de *display* en Tinkercad después de añadirlo al diseño) y 7 resistencias iguales (calculamos el valor de estas resistencias usando el Anexo 1 del final). En la configuración ánodo común el ánodo de los ledes se suele conectar a alimentación (Vcc), es decir, a 5 V en nuestro caso, y los cátodos a los pines que controlarán qué segmentos encender. Por tanto, para que circule corriente por el led y se encienda, el microcontrolador debe establecer el valor de ese pin a baja (0 V), o en alta (5 V) para apagarlo. Así que conectamos el *display* a 7 pines de la placa Arduino intercalando una resistencia en serie en cada una de estas conexiones para limitar la corriente que llega a cada led. Cada led se suele designar con una letra dependiendo de su posición en el *display*: a, b, c, d, e, f o g.

Una opción de implementación es crear un vector (**`tabla7seg`**) codificando la forma de los dígitos que se muestran en la siguiente tabla, otro vector para indicar qué pin está conectado a cada segmento (**`pines_display`**) y una variable para contar pulsaciones (**`contador`**). Cada elemento del vector **`tabla7seg`** indica qué segmentos del *display* hay que encender para formar ese número, en concreto, cada bit a "1" de cada elemento indica que hay que encender el correspondiente segmento.

	g	*f*	*e*	*d*	*c*	*b*	*a*	*HEX*
0	*0*	*1*	*1*	*1*	*1*	*1*	*1*	***3F***
1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	***06***
2	*1*	*0*	*1*	*1*	*0*	*1*	*1*	***5B***
3	*1*	*0*	*0*	*1*	*0*	*1*	*1*	***4F***
4	*1*	*1*	*0*	*0*	*1*	*1*	*0*	***66***
5	*1*	*1*	*0*	*1*	*1*	*0*	*1*	***6D***
6	*1*	*1*	*1*	*1*	*1*	*0*	*1*	***7D***
7	*0*	*0*	*0*	*0*	*1*	*1*	*1*	***07***
8	*1*	*1*	*1*	*1*	*1*	*1*	*1*	***7F***
9	*1*	*1*	*0*	*1*	*1*	*1*	*1*	***6F***



```
// segmentos a activar para cada valor del digito
const uint8_t
tabla7seg[10]={0x3f,0x6,0x5b,0x4f,0x66,0x6d,0x7d,0x7,0x7f,0x6f};
// pines a utilizar para el display
uint8_t pines_display[7]={4,5,6,7,8,9,10};
uint8_t contador = 0;
```

Podemos usar un bucle **`for`** para configurar el estado de los 7 segmentos en función del dígito a mostrar. Podemos consultar en la ayuda la función **`bitRead()`** para extraer de los números de **`tabla7seg`** los bits correspondientes a los segmentos a encender para cada valor del dígito. Como nuestro *display* es de ánodo común, para encender el segmento hay que escribir un 0 en el pin, por tanto, hay que negar los bits que extraemos de los elementos de la tabla **`tabla7seg`** antes de escribirlos en el pin correspondiente. Debemos escribir código fuente modular, creando, por ejemplo, una función que nos permita actualizar el *display* con un nuevo dígito.

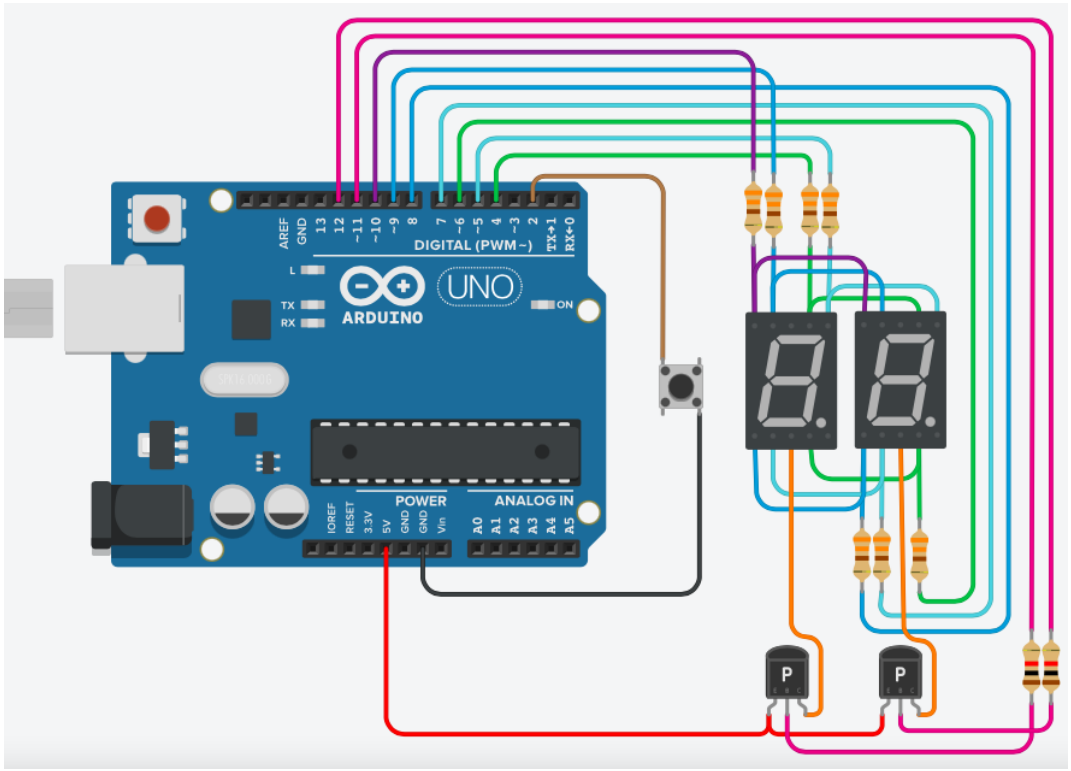
Debemos comprobar qué ocurre cuando al pulsar los botones el contador se sale de rango (0…9) y resolver.

2) Añadir un segundo *display* de 7 segmentos multiplexado y realizar un segundero con función de pausa.

Modifiquemos el código para implementar un contador con dos *displays* que actúe como segundero. Para ello podemos utilizar la función **`millis()`** para medir el tiempo transcurrido. Además, al pulsar un botón se debe detener la cuenta del segundero hasta que se suelte el botón y continúe la cuenta por donde iba. No es necesario que se muestren los segundos en el *display* mientras se tiene pulsado el botón. Podemos utilizar el siguiente código:

```
uint8_t contador = 0;
        ...
if(++contador == 60) contador = 0;
```

Para mantener la visualización de los dos dígitos simultáneamente utilizamos dos *displays* de 7 segmentos de ánodo común. Unimos los segmentos iguales de los dos *displays*, y controlamos de forma independiente los ánodos. Para esto utilizaremos transistores (bipolares) PNP (con resistencias de 1 kΩ en la base) que desconectan/conectan los ánodos a alimentación (5 V). Utilizamos transistores en vez de conectar los ánodos directamente a patillas del microcontrolador, ya que la intensidad que circula por ellos para ciertos dígitos es superior a la que puede suministrar directamente una patilla del microcontrolador. Con esta configuración, cuando ponemos en baja (0 V) el pin conectado a la base de un transistor se activa el *display* correspondiente.



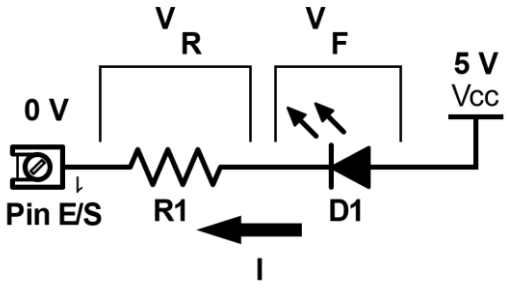
Debemos tener en cuenta que, con esta configuración, y estrictamente hablando, no podemos mostrar dos números (distintos) en los dos *displays* simultáneamente, pero si conmutamos los *displays* alternadamente y muy rápidamente (mostrando un número en un *display* durante una pequeña fracción de segundo (10 ms por ejemplo) y otro número en el otro *display* otra pequeña fracción), para el ojo humano dará la impresión de que ambos *displays* están encendidos a la vez. Ver la función **`delay()`** de Arduino. Es decir, el algoritmo sería: a) configurar el estado de los segmentos para mostrar el dígito del *display* 1, b) encender el *display* 1, c) esperar una fracción de segundo, d) apagar el *display* 1, repetir los 4 pasos para el *display* 2 con su dígito correspondiente, y vuelta a empezar.

ANEXO 1: CÁLCULO DE LA RESISTENCIA PARA ALIMENTAR UN LED

Cálculo del valor de la resistencia

Vamos a calcular qué valor debe tener la resistencia en serie con cada led del *display* (segmento) para que funcione a un tercio de la intensidad máxima de corriente eléctrica que soporta. Para esto

supondremos que el *display* que estamos utilizando es SA56-11SURKWA del fabricante Kingbright y seguimos los siguientes pasos:
- Descargamos el *datasheet* de este *display*, por ejemplo, del sitio web del fabricante (https://www.kingbright.com).
- Localizamos en la sección Absolute Maximum Ratings del *datasheet* la corriente máxima (mA) que soporta cada led del *display* (*DC forward current*).
- A continuación, consultaremos la tensión que caerá en cada led para esta corriente. Esta característica del led, la caída de tensión (*forward voltage*, V_F) típica para una corriente de 10 mA, está especificada en la tabla Electrical / Optical Characteristics.



- Ahora, asumiendo un voltaje de salida de 0 V en las patillas del microcontrolador y 5 V de alimentación, podemos calcular el valor de tensión que caerá en la resistencia: $V_R = 5\ V - V_F$
- Como ya sabemos la tensión que cae en la resistencia y la intensidad de corriente que pasa por ella, podemos calcular su valor con la ley de Ohm: $V = R \cdot I \rightarrow R = V / I$ (utilizamos unidades enteras en la ecuación: ohmios, voltios y amperios).
- Una vez tenemos el valor de la resistencia debemos buscar un valor que esté disponible comercialmente. Es decir, un valor normalizado. Nos restringiremos a la serie de resistencias E12, cuyos valores son muy comunes. En esta serie encontramos resistencias con los valores: 1,0 1,2 1,5 1,8 2,2 2,7 3,3 3,9 4,7 5,6 6,8 $8{,}2 \cdot 10^n$ ohmios donde n es un número entero. Es decir, encontramos resistencias de valores nominales como: 1 ohmio, 10 ohmios, 100 ohmios, 120 ohmios… Debemos buscar el valor de resistencia nominal (R_N) que coincida con la R calculada o que esté justo por encima (no utilizaremos una resistencia con un valor por debajo para no superar la intensidad de corriente I calculada). Utilizamos este valor, R_N, en el circuito a montar con Arduino en esta práctica.

<u>Cálculo de la potencia de la resistencia</u>

Por último, calculamos cuánta potencia debe ser capaz de disipar la resistencia (en general, esto es cómo de grande debe ser la resistencia, ya que una resistencia más grande suele poder disipar más calor):
- La potencia disipada por una resistencia se calcula como la tensión que cae en ella por la intensidad que pasa por ella: P [W] = V [V] · I [A], en este caso es aproximadamente $P_R = V_R \cdot I$.
- Una vez tenemos la potencia disipada por la resistencia, buscamos una capacidad de disipación de potencia para la resistencia que esté disponible comercialmente. Los valores habituales son: 1/8 W, 1/4 W, 1/2 W, 1 W… Seleccionamos una de estas potencias nominales suficientemente alta (es decir, que esté por encima de la P_R calculada) y ya sabemos la resistencia que necesitamos. Este valor de potencia no lo necesitamos en la simulación realizada en esta práctica ya que el simulador no admite especificar este parámetro. Lo necesitaríamos si fuésemos a comprar la resistencia.