Sistemas con Microprocesadores

Práctica 4: Comunicaciones mediante I²C

Resumen:

Utilizar 2 placas Arduino para establecer una comunicación entre ellas mediante el bus serie I²C.

Entrega:

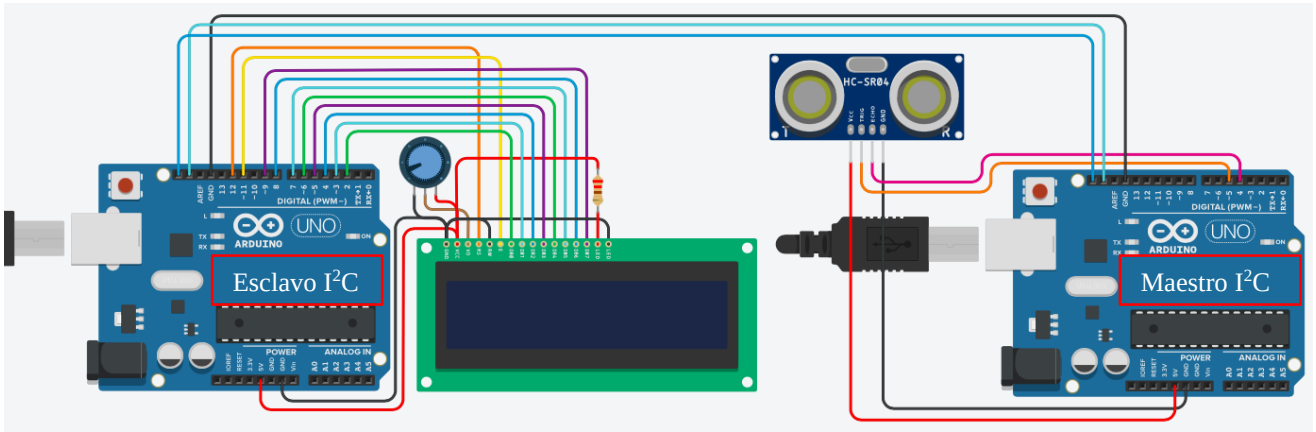
Hacer público el diseño Tinkercad.
Entregar un PDF de 1 página aproximadamente que contenga:

- Nombre y apellidos
- Dirección web del proyecto Tinkercad
- Captura de pantalla con el programa en ejecución donde aparezca la información en la pantalla LCD y código fuente.
  (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 consta de un solo apartado:

1) Realizar un diseño donde una primera placa Arduino tiene conectada una pantalla LCD (*Liquid Crystal Display*) y una segunda placa Arduino tiene conectado un módulo para medir distancia mediante ultrasonidos (HC-SR04). El primer Arduino actúa como esclavo del bus I²C (en nuestro caso recibiendo mediciones de distancia del maestro y mostrándolas en la pantalla) y el segundo como maestro (tomando mediciones y enviándolas al esclavo). Conectamos entre sí las dos placas Arduino mediante las patillas del bus de comunicaciones I²C (*Inter-Integrated Circuit*): SDA y SCL, además de la masa (GND).



EN EL ARDUINO MAESTRO: PARA MEDIR LA DISTANCIA:

Para medir la distancia podemos utilizar la siguiente función:

```
unsigned long mide_distancia(uint8_t triggerPin, uint8_t ecoPin)
{
  delayMicroseconds(2); // Por seguridad esperamos un tiempo antes de
activar la patilla
  digitalWrite(triggerPin, HIGH); // Activa el trigger 10 us
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  return pulseIn(ecoPin, HIGH, TIMEOUT_ECO); // Mide la anchura del
pulso en us
}
```

Debemos definir los pines donde hemos conectado el módulo de ultrasonidos y también el valor *timeout* para la función `pulseIn`:

```
#define PIN_DISPARADOR 5
#define PIN_ECO 4

#define TIMEOUT_ECO <anchura máxima del pulso generado por el módulo en
µs>
```

Por seguridad, calculamos la anchura máxima que puede tener el pulso que genera el módulo de ultrasonidos (macro `TIMEOUT_ECO`). Esta anchura es proporcional a la distancia que ha medido el módulo, por tanto, buscamos en su *datasheet* (HC-SR04) la distancia máxima que es capaz de medir y la pasamos a microsegundos, sabiendo que cada centímetro de distancia medida equivale a 58 microsegundos de pulso. De esta forma limitamos el tiempo que la función `pulseIn` puede quedar esperando.

La distancia en centímetros se obtiene con:

```
unsigned long cm; // Distancia medida
cm = mide_distancia(PIN_DISPARADOR, PIN_ECO) / 58; // Función que se
llamará repetidamente para obtener distancias
```

Tinkercad nos permite simular que hay un obstáculo delante del módulo de ultrasonidos para obtener medidas de distancia. En función de la distancia entre el obstáculo y el módulo genera una anchura de pulso. Para indicarle a qué distancia se encuentra este obstáculo, durante la simulación pinchamos sobre el módulo (una vez está incluido en el diseño) y arrastramos el círculo turquesa que aparece.
También debemos configurar las patillas del microcontrolador en la función **`setup`**: la patilla donde conectamos el disparador del módulo (*trigger*) debe ser de salida y la patilla donde recibimos el pulso del módulo (*echo*) debe ser de entrada. Además, en la función **`setup`** inicializamos la patilla del

disparador a baja (`LOW`) para que quede lista para llamar a la función `mide_distancia` , ya que esta función genera un pulso en esta patilla con el cual le indica al módulo que tome una medición.

EN EL ARDUINO MAESTRO: PARA LA COMUNICACIÓN I^2C:

```
#include <Wire.h>

#define DIRECC_I2C_ESCLAVO 8
```

En la función **`setup`**:

```
  Wire.begin();
```

Para enviar la medición al esclavo:

```
    Wire.beginTransmission(DIRECC_I2C_ESCLAVO);
    Wire.write(lowByte(cm));
    Wire.write(highByte(cm));
    Wire.endTransmission();
```

En el maestro debemos realizar medidas de distancia repetidamente, y en caso de que la distancia esté entre 40 y 290 cm enviarla al esclavo. Antes de repetir este ciclo esperaremos una décima de segundo para no enviar muchas medidas por segundo al esclavo.

EN EL ARDUINO ESCLAVO: PARA LA COMUNICACIÓN I^2C:

```
#include <Wire.h>

#define DIRECC_I2C 8
```

Definimos la función que procesa los datos recibidos por I^2C:

```
void callback_recepcion_i2c(int bytes_recibidos)
{
  if(bytes_recibidos >= 2)
  {
    int cm1, cm2;

    cm1 = Wire.read();
    cm2 = Wire.read();
    if(cm1 != -1 && cm2 != -1)
    {
      uint16_t cm;
      cm = (cm2 << 8) | cm1;
      // Imprime la distancia en la pantalla
    }
    else
      // Imprime en la pantalla un mensaje indicando el error ocurrido
  }
}
```

En la función **`setup`**:

```
  Wire.begin(DIRECC_I2C);
  Wire.onReceive(callback_recepcion_i2c);
```

EN EL ARDUINO ESCLAVO: PARA CONTROLAR LA LCD:

Incluimos al principio:

```
#include <LiquidCrystal.h>

#define PIN_RS 12
#define PIN_EN 11
#define PIN_D0 2
#define PIN_D1 3
#define PIN_D2 4
#define PIN_D3 5
#define PIN_D4 6
#define PIN_D5 7
#define PIN_D6 8
#define PIN_D7 9
```

Deberemos crear un objeto global de tipo `LiquidCrystal`, que utilizaremos para configurar la pantalla LCD en la función **`setup`**, para establecer la posición del cursor en la pantalla y para imprimir mensajes de texto. Para esto consultaremos la documentación de Arduino sobre este tipo de objeto.

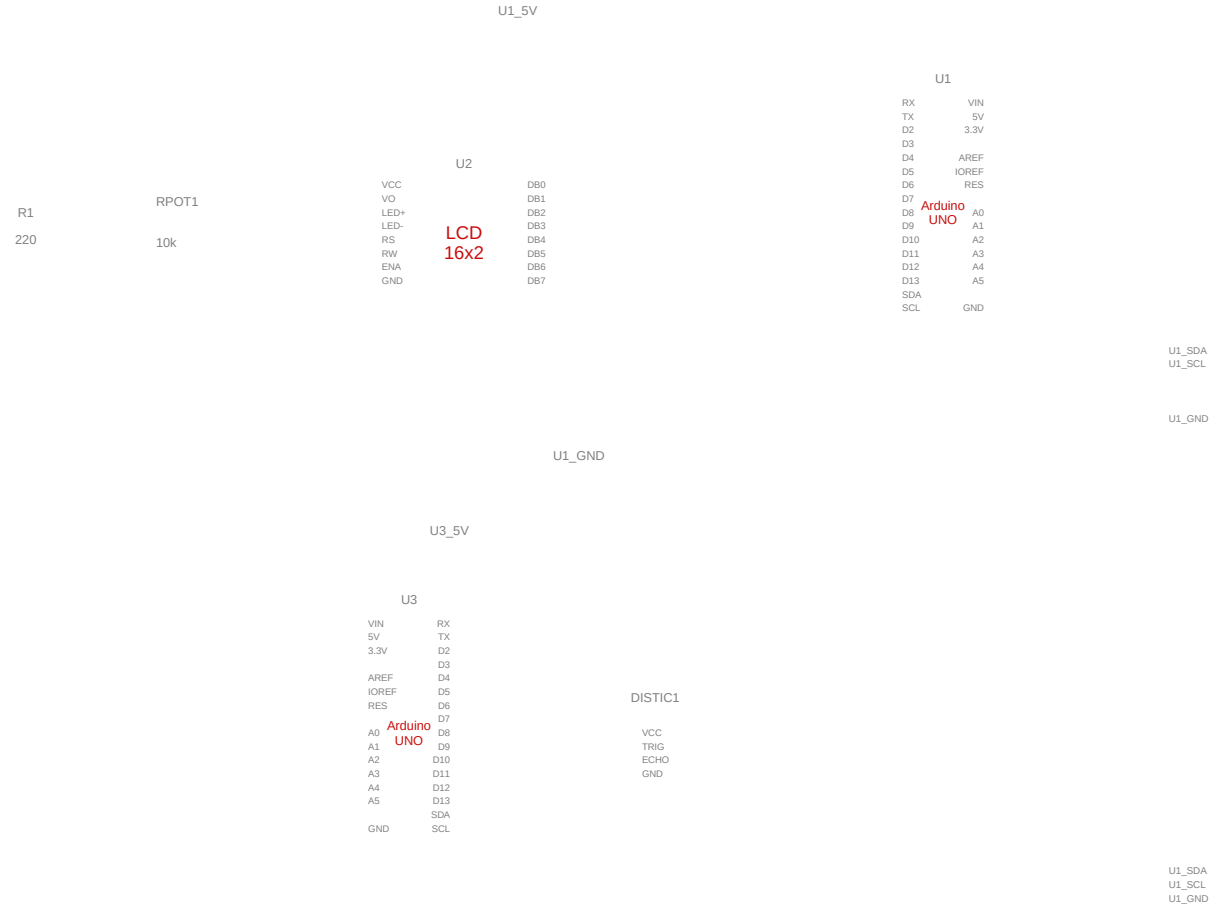
En la primera línea de la pantalla LCD debe quedar de forma permanente la fecha de realización de la práctica y las iniciales del nombre del alumno. Por ejemplo:
24/03/22 C.M.G.

En la segunda línea de la pantalla cada vez que le llegan datos el esclavo debe mostrar en la pantalla el valor recibido de la distancia seguido de las unidades de distancia usadas.

La función **`loop`** en el esclavo puede quedar vacía.

Para aprovechar eficientemente los recursos del microcontrolador procuraremos utilizar variables de tipos pequeños cuando sea posible (`uint8_t` mejor que `int` cuando no se necesiten números grandes) y evitar en la medida de lo posible el uso de números de punto flotante. Por otra parte, para evitar errores en nuestro código fuente limitaremos el uso de variables globales, declarándolas locales cuando sea posible.

Sugerencia de conexionado y valores de componentes (potenciómetro y resistencia para la LCD):



Tendremos que ajustar el valor del potenciómetro que hemos incluido en el circuito para obtener un nivel de contraste que permita ver los caracteres de la pantalla. Al iniciar la simulación la retroiluminación del LCD debe encenderse. Si no fuera así, debemos comparar las conexiones de nuestro circuito con las de este esquemático. Para ver el esquemático de nuestro circuito podemos pinchar el botón de arriba a la derecha que dice “Vista esquemática”.