Instituto Politécnico Nacional

Escuela Superior de Cómputo

Practica 5.- Comunicación UART entre Dos Microcontroladores

Sistemas en chip
Profesora: Ana Luz Barrales Lopez

Luciano Hernández Jonathan
Rodriguez Morales Iñaki
Lara Sarmientos Erick
Ortiz Chavez Martín Cipriano

8 de noviembre de 2024

1 Introducción

UART, o Universal Asynchronous Receiver-Transmitter, es un protocolo de comunicación serial que permite transmitir y recibir datos entre dos dispositivos. Se utiliza ampliamente en sistemas embebidos y microcontroladores (como Arduino) debido a su simplicidad y eficiencia en la transmisión de datos de corta distancia.

En esta tarea, aprenderá a establecer una comunicación en serie entre dos microcontroladores mediante UART. Un microcontrolador enviará un mensaje y el otro lo recibirá y lo mostrará o realizará una acción en consecuencia.

1.1 Objetivos

- **Configurar la Comunicación UART entre los Microcontroladores.**
- **Implementar un Sistema de Transferencia de Datos Simples.**
- **Desarrollar un Modelo de Comunicación Semidúplex.**
- **Establecer una Comunicación Dúplex Completa**

1.2 Materiales

Los materiales utilizados en la práctica son los siguientes:

- 2 Arduino Uno
- Arduino IDE
- 8 leds
- 8 resistencias de 330 ohms
- Diaplay 7 segmentos (cátodono común)
- Cable para conexiones

2 Desarrollo de la Práctica

2.1 Simplex

En este tipo de transferencia de datos mediante UART, se enviará unicamente datos en una sola direción, de un arduino a otro.

2.2 Explicación de las conexiones

Puerto Serial: Conecta el pin TX del Arduino al dispositivo receptor de datos para enviar números generados aleatoriamente.

Display de 7 segmentos: Los segmentos (a-g) del display están conectados a los pines digitales 2, 3, 4, 5, 6, 7 y 8 del Arduino. Cada pin controla un segmento específico del display para mostrar los números recibidos por el puerto serial.

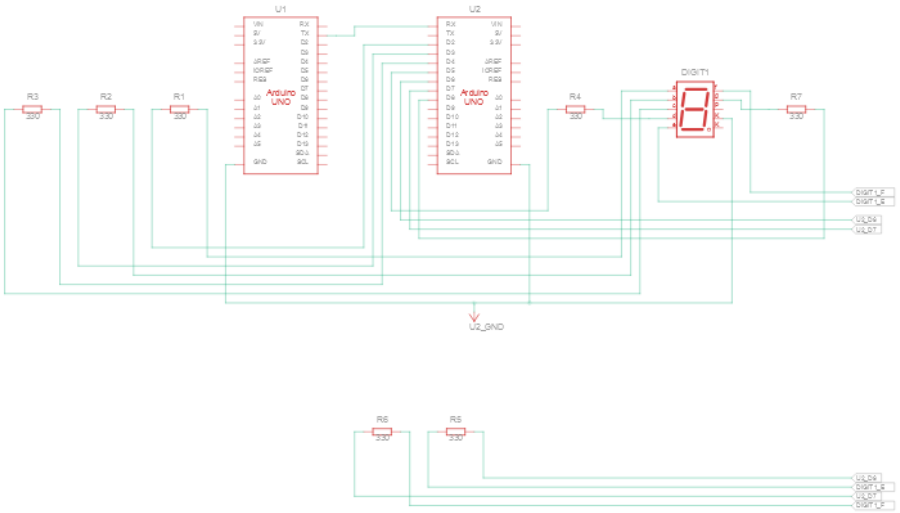

Figure 1: Diagrama del circuito: UART simplex

2.3 Desarrollo del primer código

2.3.1 Configuración inicial (void setup):

Se inicializa la comunicación serial con una tasa de transferencia de 9600 baudios, lo que permite el envío de datos hacia otros dispositivos a través del puerto UART.

```
void setup() {
  Serial.begin(9600);  // Inicializa la comunicaci n UART
}
```

2.3.2 Bucle Principal de Envío de Datos

Se genera un número aleatorio entre 0 y 9. Se envía este número al puerto serial utilizando Serial.write, enviándolo como un byte. Se espera 1 segundo antes de repetir el proceso, lo que establece una frecuencia de envío de un número por segundo.

```
void loop() {
  int numberToSend = random(0, 10);    // Genera un n mero aleatorio
   entre 0 y 9
  Serial.write(numberToSend);          // Env a el n mero
  delay(1000);                         // Espera 1 segundo antes de
   enviar otro n mero
}
```

2.4 Desarrollo del segundo código

2.4.1 Definición de pines y mapeo de segmentos

segmentPins: Arreglo que define qué pines del Arduino están conectados a cada segmento del display. Cada índice representa un segmento de 'a' a 'g', correspondiendo a los pines digitales del 2 al 8.

digits: Matriz que define la configuración de los segmentos para mostrar cada dígito de 0 a 9. Cada subarreglo contiene 7 valores (1 o 0) que indican si un segmento (a-g) debe estar encendido o apagado para formar el número correspondiente en el display.

```
    const int segmentPins[7] = {2, 3, 4, 5, 6, 7, 8};
    const bool digits[10][7] = {
      {1, 1, 1, 1, 1, 1, 0},  // 0
      {0, 1, 1, 0, 0, 0, 0},  // 1
      {1, 1, 0, 1, 1, 0, 1},  // 2
      {1, 1, 1, 1, 0, 0, 1},  // 3
      {0, 1, 1, 0, 0, 1, 1},  // 4
      {1, 0, 1, 1, 0, 1, 1},  // 5
      {1, 0, 1, 1, 1, 1, 1},  // 6
      {1, 1, 1, 0, 0, 0, 0},  // 7
      {1, 1, 1, 1, 1, 1, 1},  // 8
      {1, 1, 1, 1, 0, 1, 1}   // 9
    };
```

2.4.2 Configuración inicial de pines y comunicación serial

- pinMode(segmentPins[i], OUTPUT): Configura cada pin de los segmentos como salida.
- digitalWrite(segmentPins[i], LOW): Apaga todos los segmentos del display al inicio.

```
    void setup() {
      Serial.begin(9600);  // Inicializa la comunicacion serial a 9600 baudios

      for (int i = 0; i < 7; i++) {
        pinMode(segmentPins[i], OUTPUT);     // Configura cada pin del display como salida
        digitalWrite(segmentPins[i], LOW);   // Apaga todos los segmentos inicialmente
      }
}
```

2.4.3 Bucle principal: Recepción y visualización del dígito

- Serial.available() ¿ 0: Verifica si hay datos disponibles en el buffer serial.
- Serial.read(): Lee el byte de datos (número) recibido y lo almacena en receivedNumber.
- if (receivedNumber = 0 && receivedNumber = 9): Asegura que el valor recibido esté dentro del rango (0-9). Si es asi, se llama a displayDigit para mostrar el número.

```
    void loop() {
      if (Serial.available() > 0) {               // Verifica si hay datos en el buffer serial
        int receivedNumber = Serial.read();       // Lee el n mero recibido
        if (receivedNumber >= 0 && receivedNumber <= 9) {  // Valida que el n mero est  en el rango
          displayDigit(receivedNumber);           // Llama a la funci n para mostrar el n mero en el display
        }
      }
    }
```

2.4.4 Función displayDigit: Configuración de segmentos para cada número

La función displayDigit recibe el dígito y enciende o apaga cada segmento del display de acuerdo con el mapeo digits correspondiente a ese dígito. digitalWrite activa o desactiva el segmento según el valor (1 o 0) del mapeo digits.

```
1     void displayDigit(int digit) {
2   for (int i = 0; i < 7; i++) {
3     digitalWrite(segmentPins[i], digits[digit][i]);  // Enciende o
      apaga cada segmento
4   }
5 }
```

2.5 Simulación

En la simulación que se muestra acontinuación que se realizó en el software tinkercad, se muestra el correcto funcionamiento de esta primera parte de la práctica.

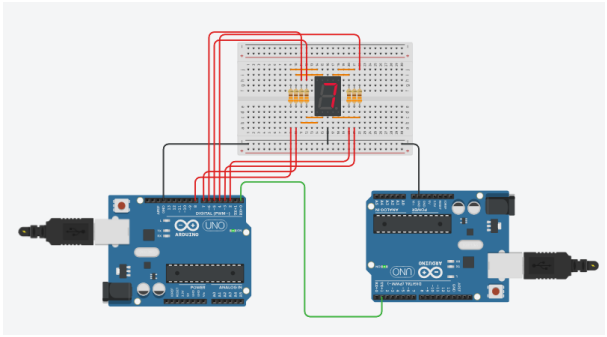

Figure 2: Funcionamiento del circuito

Primera imagen donde se puede observar la conexion que se siguió para el correcto funcionamiento de la practica donde el arduino emisor tiene incluido los datos que se enviarán al arduino emisor (número aleatorio). Finalmente el arduino receptor muestra en el display de 7 segmentos el número recibido por el arduino emisor.

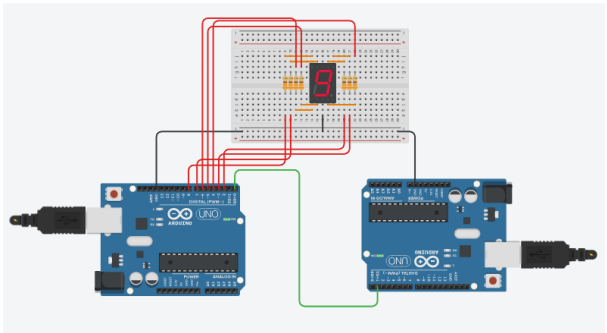

Figure 3: Funcionamiento del circuito

Se puede observar que el arduino emisor genera un nuevo número aleatorio y lo manda al arduino receptor, mostrandolo en el display de 7 segmentos.

2.6 Implementación

Una vez implementado la conexion y desarrollo del codigo dentro de la simulación, se elaboró el circuito de manera practica con componentes reales para observar su funcionamiento, a continuación, se muestran las imagenes del citcuito en funcionamiento conectado a a un arduino uno.

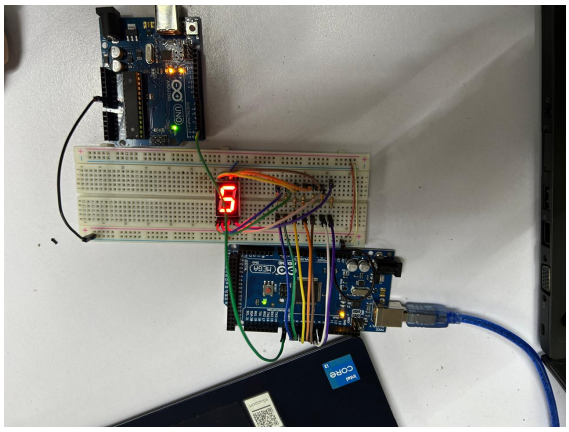

Figure 4: Funcionamiento del cricuito

Primera fotografía donde se puede observar la conexion que se siguió patra el correcto funcionamiento de la practica. Además de mostrar de manera correcta el numero en el display desde el arduino receptor.

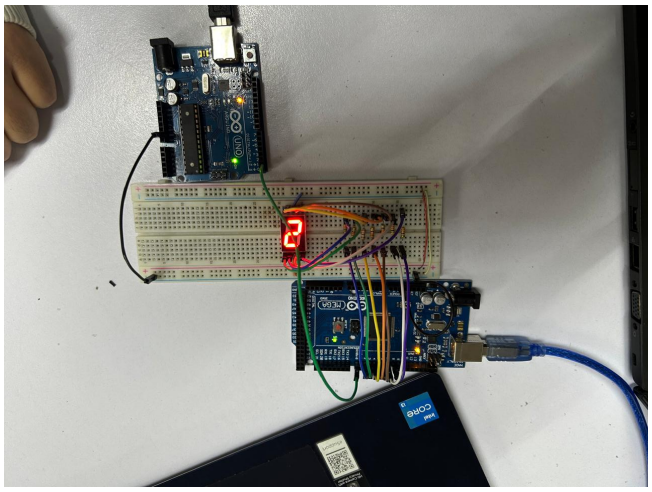

Figure 5: Funcionamiento del cricuito

Segunda fotografía donde se puede observar el mensaje siendo recibido de manera correcta por la terminal de la computadora.

2.7 Semi-Duplex

En esta segunda sección de la práctica se se envían datos de manera secuencial entre los dos arduinos, en el que el arduino 1 envia un numero aleatorio y el arduino 2 los recibe y lo representa en binario meidante los leds, posteriormente, envía un 1 al arduino 1 para prender un led, después de eso, el arduino 1 vuelve a generar un nuevo número, comenzando con el proceso de nuevo.

2.8 Explicación de las conexiones

Para realizar las conexiones, conecta el pin 13 del Arduino Emisor a un LED en serie con una resistencia 330 ohms y su cátodo a GND, lo cual indica la confirmación de recepción. En el Arduino Receptor, conecta cuatro LEDs en los pines 2, 3, 4 y 5 respectivamente, cada uno con una resistencia similar, representando en binario el número recibido; el cátodo de cada LED debe ir a GND. Para la comunicación serial, conecta el pin TX del Arduino Emisor al pin RX del Receptor y el RX del Emisor al TX del Receptor, compartiendo GND entre ambos Arduinos.

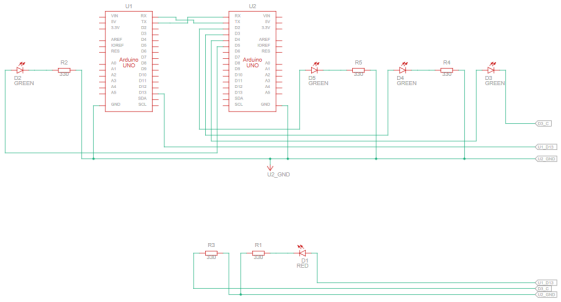

Figure 6: Diagrama de conexiones Semiduplex

2.9 Desarrollo del primer código

2.9.1 Declaración de variables

ledRecepcion indica el pin del LED que confirma la recepción, y numeroEnviado almacenará el número aleatorio que se enviará al receptor.

```
int ledRecepcion = 13;
int numeroEnviado;
```

2.9.2 Configuración inicial

Se inicia la comunicación serial a 9600 baudios, se configura el pin del LED como salida, y se usa randomSeed para inicializar la generación de números aleatorios, basada en una lectura analógica.

```
void setup() {
  Serial.begin(9600);
  pinMode(ledRecepcion, OUTPUT);
  randomSeed(analogRead(0)); // Para generar n meros aleatorios
}
```

2.9.3 Generación y envío de número aleatorio

Se genera un número aleatorio entre 0 y 15, se envía al receptor mediante Serial.write(), y se espera un segundo antes de continuar.

```
void loop() {
  numeroEnviado = random(0, 16); // Genera n mero entre 0 y 15
  Serial.write(numeroEnviado); // Env a el n mero al receptor
  delay(1000); // Pausa para asegurar la transmisi n
```

2.9.4 Recepción de confirmación y encendido de LED

Si se recibe una confirmación (valor 1) desde el receptor, el LED en el pin 13 se enciende por un segundo. Luego se apaga, y tras una pausa, el ciclo se reinicia.

```
      if (Serial.available() > 0) {
        int confirmacion = Serial.read();
        if (confirmacion == 1) {
          digitalWrite(ledRecepcion, HIGH);
          delay(1000); // LED encendido por 1 segundo
          digitalWrite(ledRecepcion, LOW);
        }
      }

      delay(1000);
    }
```

2.10 Desarrollo del segundo código

2.10.1 Declaración de pines de LEDs

Se define un arreglo con los pines de los LEDs que representarán el número recibido en formato binario.

```
    int leds[4] = {2, 3, 4, 5}; // Pines para representar en binario
```

2.10.2 Configuración inicial

Se inicia la comunicación serial a 9600 baudios y se configuran los pines de los LEDs como salida.

```
    void setup() {
      Serial.begin(9600);
      for (int i = 0; i < 4; i++) {
        pinMode(leds[i], OUTPUT); // Configura los LEDs como salida
      }
    }
```

2.10.3 Recepción de número

Verifica si hay datos disponibles en el buffer serial. Si los hay, lee el número enviado por el emisor.

```
    void loop() {
      if (Serial.available() > 0) {
        int numeroRecibido = Serial.read(); // Lee el n mero enviado
    por el emisor
```

2.10.4 Visualización del número en binario

Despliega el número recibido en binario utilizando los cuatro LEDs, manteniendo esta visualización por un segundo.

```
        for (int i = 0; i < 4; i++) {
          digitalWrite(leds[i], (numeroRecibido >> i) & 0x01); //
    Ajusta cada LED
```

```
        }
        delay(1000); // Mantiene el n mero mostrado durante 1 segundo
```

2.10.5 Apagado de LEDs y Confirmación de recepción

Después de un segundo, apaga los LEDs y hace una breve pausa antes de enviar la confirmación. Envía un valor de confirmación (1) al Arduino emisor para indicar que el número ha sido recibido y mostrado en los LEDs.

```
        for (int i = 0; i < 4; i++) {
          digitalWrite(leds[i], LOW);
        }

        delay(1000); // Pausa breve antes de enviar la confirmaci n
        Serial.write(1); // Env a confirmaci n de recepci n al
    emisor
     }
}
```

2.11 Simulación

Posteiormente al desarrollo de los códigos implementados y haciendo las conexiones correspondientes, se muestra el correcto funcionamiento del sistema dentro del simulador de tinkercad.

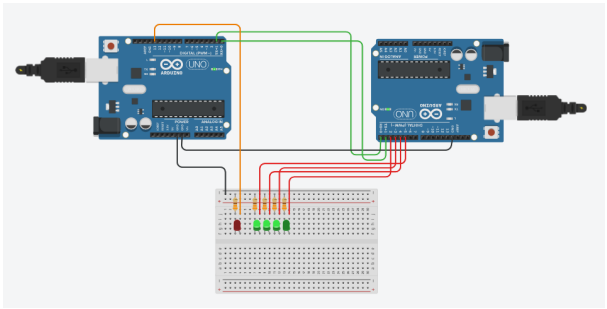

Figure 7: Arduino receptor resiviendo y mostrandop número aleatorio

En esta primera imagen, se puede observar como el arduino receptor recibe de manera correcta el número aleatorio generado por el arduino emisor y mostrandolos de manera binaria mediantes los leds.

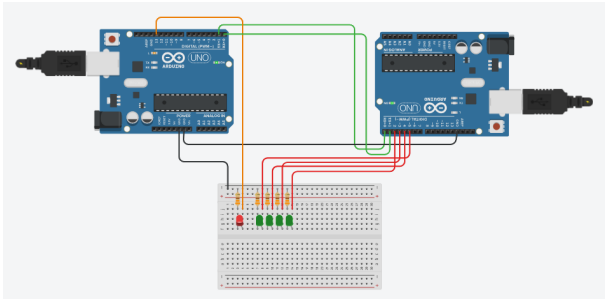

Figure 8: Arduino emisor recibiendo dato de confirmación

Finalmente en esta seguna imagen podemos observar como despues de mostrar el numero, apaga los leds, y envía dato de confirmación de manera correcta al arduino emisor, prendiendo un led separado que significa que recibio a su vez el dato para generar un nuevo número aleatorio para enviarselo al arduino receptor.

2.12 Implementación

Posteriormente que se simulara de manera exitosa la práctica se prosiguió a su elaboración mediante componentes reales.

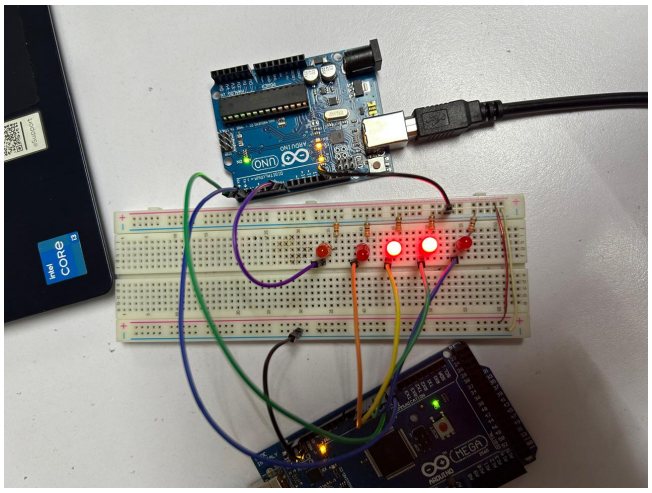

Figure 9: Arduino recibiendo y mostrando número aleatorio

En esta primer fotografía se puede observar que se llevó a cabo la conexión de pines correspondiente. Además, se observa como el arduino receptor recibe y muestra de manera correcta el número aleatorio generado y enviado por el arduino emisor.

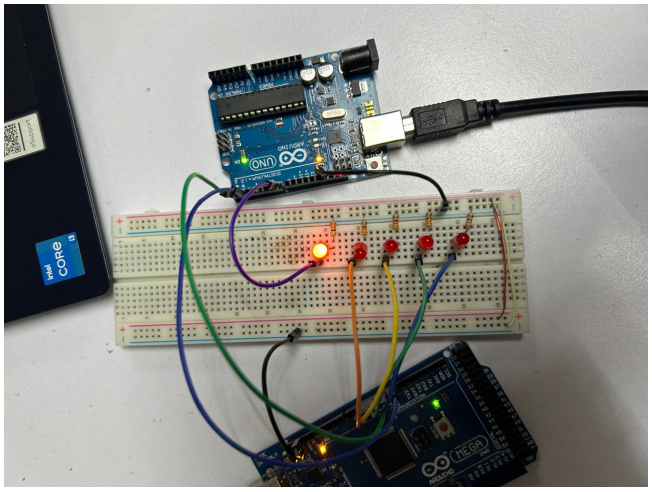

Figure 10: Arduino 2 devolviendo mensaje a arduino 1

En esta segunda imagen se puede observar que el arduino que recibió el número apaga los leds para posteriormente enviar un número 1 al arduino 1, prendiendo el led de que se recibió el dato. Generando un nuevo número aleatorio y comenzando con el ciclo de nuevo.

2.13 Duplex

Para este tipo de conexión, los arduinos deben de establecer una comunicación asincrona en ambas direcciones de manera simultanea, es decir, que en este caso, cada arduino generará un número aleatorio que enviará al otro arduino para mostrarlo nuevamente en binario mediante los leds, constantemente.

2.13.1 Explicación de las conexiones

En el lado del emisor, conecta cuatro LEDs a los pines digitales 2, 3, 4 y 5 a través de resistencias, cuyo cátodo se conecta a GND. En el receptor, se realiza una conexión similar para los LEDs a los pines 2, 3, 4 y 5, asegurándote de que cada LED tenga su resistencia. Luego, conecta el pin TX del Arduino emisor al pin RX del receptor para la transmisión de datos seriales. De igual manera se conecta el pin TX del arduino receptor al pin RX del arduino emisor, finalmente ambos arduinos deben de estar conectados a un gnd en común.

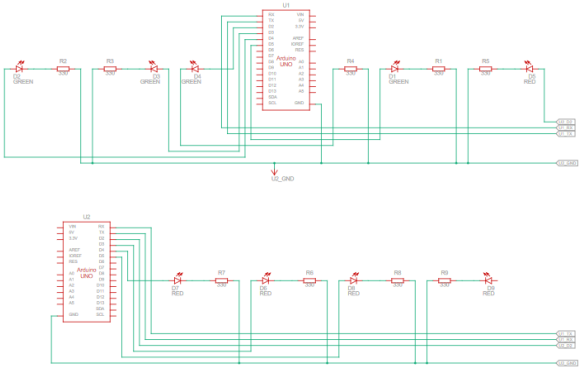

Figure 11: Conexión que se seguirá para esta parte de la practica

2.14 Desarrollo del primero código

A continuación se describe por bloques el funcionamiento general del primer código necesario

2.14.1 Definición de Pines y Configuración Inicial

En este bloque, se define un arreglo ledsEmisor que contiene los pines a los que están conectados los LEDs del emisor (2, 3, 4 y 5). La función setup() establece una comunicación serial a 9600 baudios y configura estos pines como salidas para activar o desactivar los LEDs. Además, se usa randomSeed con una lectura analógica para inicializar el generador de números aleatorios, lo que ayuda a producir una secuencia distinta en cada ejecución.

```
1     int ledsEmisor[4] = {2, 3, 4, 5}; // Pines de LEDs para el emisor
2     int numeroEmisor;
3
4     void setup() {
5       Serial.begin(9600);
6       for (int i = 0; i < 4; i++) {
```

```
7         pinMode(ledsEmisor[i], OUTPUT); // Configura los LEDs del
    emisor
8       }
9       randomSeed(analogRead(0)); // Para generar n meros aleatorios
10    }
```

2.14.2 Generación y Envío de Números Aleatorios

En este fragmento de código, el emisor genera un número aleatorio entre 1 y 15 (numeroEmisor) y lo envía al receptor mediante Serial.write(). Luego, representa este número en binario usando los LEDs conectados. Cada bit del número se desplaza y se asigna al correspondiente LED (de menor a mayor peso). Los LEDs permanecen encendidos durante un segundo, mostrando el valor binario del número.

```
1     void loop() {
2       numeroEmisor = random(1, 16); // Genera un n mero aleatorio
    entre 1 y 15
3       Serial.write(numeroEmisor); // Env a el n mero al receptor
4
5       // Muestra el n mero en binario usando los LEDs del emisor
6       for (int i = 0; i < 4; i++) {
7         digitalWrite(ledsEmisor[i], (numeroEmisor >> i) & 0x01); //
    Ajusta cada LED
8       }
9
10      delay(1000); // Mantiene los LEDs encendidos durante 1 segundo
11    }
```

2.14.3 Apagado de LEDs y Recepción de Datos del Receptor

Aquí, después de mostrar el número aleatorio, los LEDs se apagan. A continuación, el emisor espera una respuesta del receptor; si hay datos disponibles (Serial.available() ¿ 0), los lee y muestra el número recibido en binario usando los mismos LEDs. Este valor también permanece visible durante un segundo antes de apagarse.

```
1       // Apaga los LEDs despu s de mostrar el n mero
2       for (int i = 0; i < 4; i++) {
3         digitalWrite(ledsEmisor[i], LOW);
4       }
5
6       // Espera por el n mero del receptor
7       if (Serial.available() > 0) {
8         int numeroRecibido = Serial.read(); // Lee el n mero enviado
    por el receptor
9
10        // Muestra el n mero recibido en binario usando los LEDs del
    emisor
11        for (int i = 0; i < 4; i++) {
12          digitalWrite(ledsEmisor[i], (numeroRecibido >> i) & 0x01);
13        }
14
15        delay(1000); // Muestra el n mero recibido durante 1 segundo
16
17        // Apaga los LEDs del emisor
18        for (int i = 0; i < 4; i++) {
19          digitalWrite(ledsEmisor[i], LOW);
```

```
20          }
21        }
22
23        delay(1000); // Espera 1 segundo antes de enviar el siguiente
    n mero
24      }
```

2.15 Desarrollo del segundo código

2.15.1 Definición de Pines y Configuración Inicial

Este bloque configura los pines del receptor de manera similar al emisor. Se define el arreglo ledsReceptor, que contiene los pines a los que están conectados los LEDs del receptor (pines 2, 3, 4 y 5). Luego, setup() establece la comunicación serial y configura estos pines como salidas. La función randomSeed usa una lectura analógica para inicializar el generador de números aleatorios.

```
1     int ledsReceptor[4] = {2, 3, 4, 5}; // Pines de LEDs para el
    receptor
2     int numeroReceptor;
3
4     void setup() {
5       Serial.begin(9600);
6       for (int i = 0; i < 4; i++) {
7         pinMode(ledsReceptor[i], OUTPUT); // Configura los LEDs del
    receptor
8       }
9       randomSeed(analogRead(0)); // Para generar n meros aleatorios
10    }
```

2.15.2 Recepción de Datos y Visualización del Número

Este fragmento del loop() permite al receptor leer un número enviado por el emisor cuando esté disponible en el buffer serial. Luego, el número recibido se representa en binario en los LEDs, permaneciendo visible durante un segundo antes de apagarse.

```
1     void loop() {
2       // Espera el n mero enviado por el emisor
3       if (Serial.available() > 0) {
4         int numeroRecibido = Serial.read(); // Lee el n mero enviado
    por el emisor
5
6         // Muestra el n mero recibido en binario usando los LEDs del
    receptor
7         for (int i = 0; i < 4; i++) {
8           digitalWrite(ledsReceptor[i], (numeroRecibido >> i) & 0x01);
    // Ajusta cada LED
9         }
10
11        delay(1000); // Mantiene los LEDs encendidos durante 1 segundo
12
13        // Apaga los LEDs despu s de mostrar el n mero
14        for (int i = 0; i < 4; i++) {
15          digitalWrite(ledsReceptor[i], LOW);
16        }
```

2.15.3 Generación y Envío de Números Aleatorios

Aquí, el receptor genera un número aleatorio (numeroReceptor), lo envía al emisor a través de Serial.write(), y muestra el valor binario en los LEDs. Al igual que en los bloques anteriores, el número permanece visible durante un segundo antes de apagarse. Este retardo permite que tanto el emisor como el receptor mantengan sincronización en su comunicación y visualización.

```
1         numeroReceptor = random(1, 16); // Genera un n mero aleatorio
    entre 1 y 15
2         Serial.write(numeroReceptor); // Env a el n mero al emisor
3
4         // Muestra el n mero en binario usando los LEDs del receptor
5         for (int i = 0; i < 4; i++) {
6           digitalWrite(ledsReceptor[i], (numeroReceptor >> i) & 0x01);
    // Ajusta cada LED
7         }
8
9         delay(1000); // Muestra el n mero enviado durante 1 segundo
10
11        // Apaga los LEDs del receptor
12        for (int i = 0; i < 4; i++) {
13          digitalWrite(ledsReceptor[i], LOW);
14        }
15      }
16
17      delay(1000); // Espera 1 segundo antes de procesar el siguiente
    n mero
18    }
```

2.16 Simulación

Despué sde desarrollar los codigos necesarios y explicadas las conexiones para el correcto funcionamiento, se implemnetó en simulación, haciendo uso del software Tinkercad

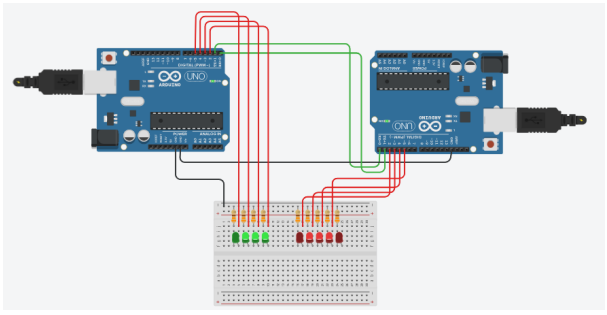

Figure 12: Ambos arduinos enviando datos simlutaneamente

En esta captura de la simulación, se puede observar como ambos arudions generan un número aleatorio, lo envían al otro arduino y este los muestra en codigo binario mediante los leds.

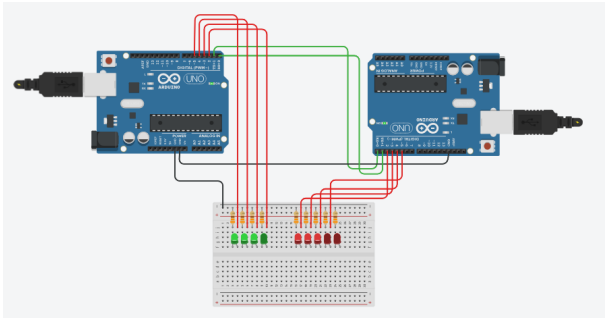

Figure 13: Envíos de datos simltaneos

En esta imagen se puede observar como eventualmente generan otro número y nuevamente lo envían para se representado mediantes los leds.

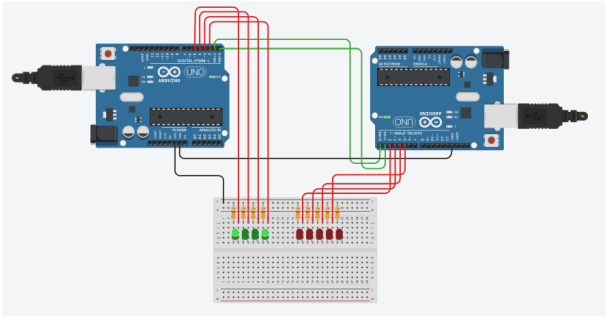

Figure 14: Caption

Finalmente, debido a que estamos usando un protocolo UART, estos no se encuentran sincronizandos, por lo que generan y envían los datos a diferentes velocidades, por lo que es común que haya ocasiones como esta en la que solo un arduino ya recibió los datos del otro arduino mientras el otro se encuentra recibiendo los datos en ese momento.

2.17 Implementación

Una vez comprobado que funciona de manera correcta en la simulación se prosiguió a implemnetar esta tercer parte de la práctica en con componentes reales como se muestra a continuación.

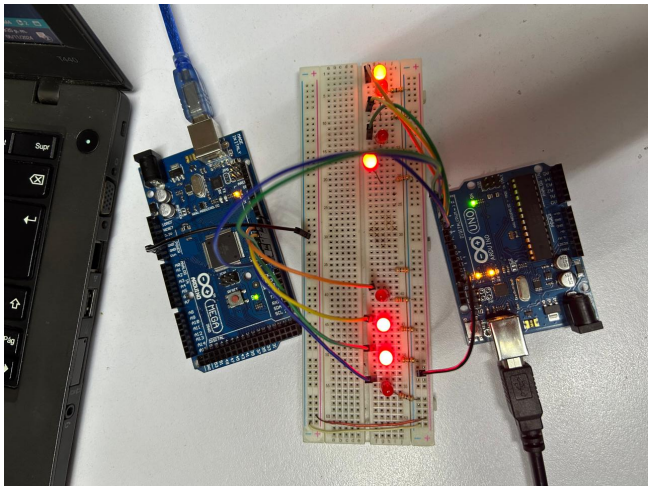

Figure 15: Envío de datos simultaneos

En esta fotografía se puede observar como ambos arduinos envían los datos simultaneamente de manera correcta, a su vez, cada arduino representandolo mediante el encendido de los leds.

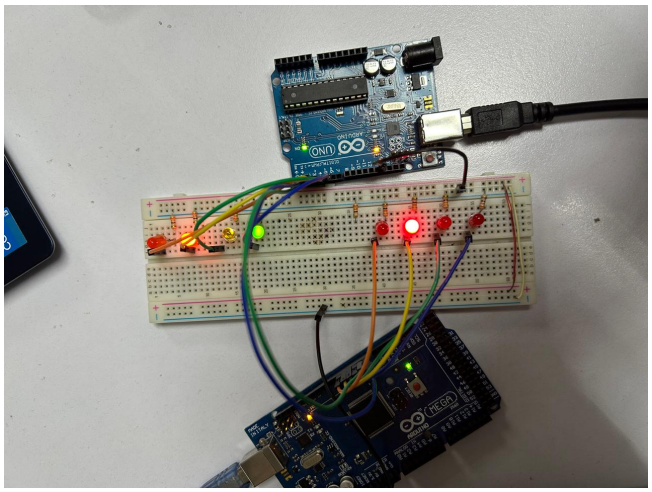

Figure 16: Envío de datos simultaneos

Después de un tiempo, ambos arduinos generan nuevos números aleatoprios y los envian mediante los pines de TX y RX, encendiendo leds diferentes que representan el nuevo número.

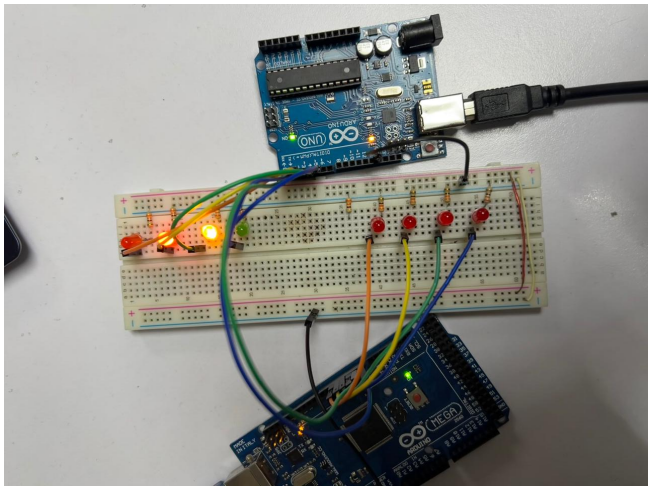

Figure 17: Caption

Por último, como se mencionó en la simulación, al estar utilizando un protocolo UART, no se encuentran sincronizados el envío de datos, por lo que hay ocasiones en los que un arduino ya esta representando los leds del número recibido, mientras que el otro se encuentra apenas recibiendo los datos.

3 Conclusiones

- **Morales Rodriguez Iñaki** La práctica demostró cómo la comunicación serie UART, a través de su protocolo asíncrono, facilita la transmisión de datos entre dispositivos. En la primera parte de la práctica, donde el Arduino emisor genera un número aleatorio y lo envía al Arduino receptor, se puso de manifiesto la eficiencia del protocolo UART para la transferencia de datos de forma simple y directa. La conexión entre los dispositivos y la correcta implementación de este protocolo son fundamentales para proyectos que requieren la transmisión de información entre microcontroladores de forma estable y sin pérdidas.

- **Luciano Hernández Jonathan** La segunda parte de la práctica enfatizó la implementación de la comunicación semiduplex, en la cual los datos son enviados en ambas direcciones, pero no simultáneamente. Este tipo de comunicación implica un control de flujo adecuado, donde el Arduino receptor no solo recibe el número enviado por el emisor, sino que también envía una señal de vuelta al emisor para continuar con el ciclo. Esta parte mostró cómo el diseño de sistemas con UART puede ser mejorado mediante el uso de señales de control como los pines de transmisión y recepción, además de las prácticas de espera activa para sincronizar ambos dispositivos.

- **Lara Sarmiento Erick** En la tercera parte, donde ambos Arduinos envían y reciben datos simultáneamente, se resalta la capacidad de UART para manejar comunicaciones bidireccionales dentro de sistemas. Aunque UART no es un protocolo que maneje comunicación full-duplex en su forma básica, el desarrollo de este tipo de sistemas permite simular la interacción de dos dispositivos de manera eficiente. Este ejercicio muestra la importancia de coordinar los tiempos de envío

y recepción en sistemas con múltiples dispositivos para evitar conflictos o pérdidas de datos.

- **Ortiz Chávez Martín Cipriano** A lo largo de la práctica, el uso de displays de 7 segmentos y LEDs permitió visualizar y controlar datos en formato binario, ayudando a comprender cómo las señales digitales pueden ser representadas visualmente. La implementación de estos periféricos, junto con la comunicación UART, mostró la flexibilidad del sistema para interactuar con el entorno físico. La representación de números binarios en LEDs no solo facilitó la visualización de los datos transmitidos, sino que también destacó la importancia de la programación para controlar el flujo de información y la activación de dispositivos electrónicos de manera secuencial y controlada.

4 Referencias

1 Arduino.cl, "Cómo configurar la comunicación UART en Arduino", Arduino.cl, 2024. [Online]. Available: https://arduino.cl/como-configurar-la-comunicacion-uart-en-arduino/. [Accessed: Nov. 4, 2024].

2 A. Quiran, "How to Setup UART Communication in Arduino", YouTube, 2024. [Online]. Available: https://acortar.link/VUb13r