



SLIDE 1



I2C

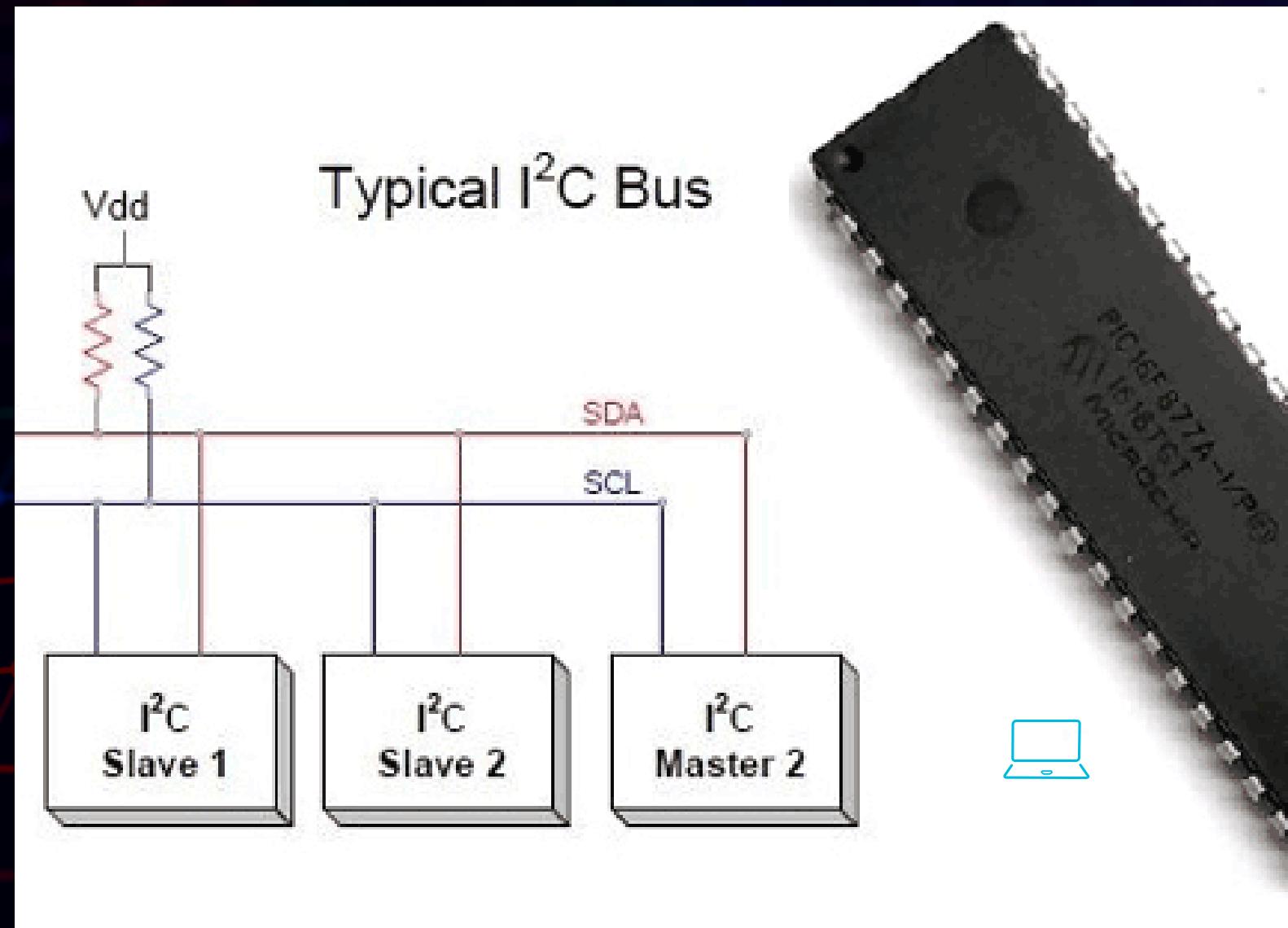
Simpson y Reyes Brian Carl & Ojeda Serafio Hector



¿QUÉ ES?

SLIDE 2

I²C



I²C (Inter-Integrated Circuit) es un protocolo de comunicación serial que permite conectar varios dispositivos entre sí usando solo dos cables:

- SDA (Serial Data) → Transporta los datos.
- SCL (Serial Clock) → Lleva la señal de reloj que sincroniza la comunicación.

Un solo "maestro" puede controlar varios "esclavos", o incluso varios maestros pueden compartir la misma línea (con manejo avanzado).

- Sensores (como acelerómetros, brújulas o termómetros digitales).
- Pantallas OLED o LCD.
- Memorias EEPROM.
- Convertidores ADC/DAC.
- Microcontroladores comunicándose entre sí (como justo lo que hicimos con los tres tipos Arduino, ESP32-C6, Attiny85).



¿QUÉ HICIMOS?

SLIDE 3

Configuramos dos de cada uno de los microcontroladores (uno como maestro y otro como esclavo) para comunicarse por I₂C.

El maestro envió 1 Byte de datos al esclavo y este devolvió el tiempo que tardó en recibirllos.

Ajustamos la configuración con pines de SDA SCL y las diferentes configuraciones para poder ver el modo serial, ademas resolviendo problemas de conexión y dirección.

Finalmente, medimos los tiempos de ida y vuelta en milisegundos ó microsegundos para analizar la velocidad de la comunicación.

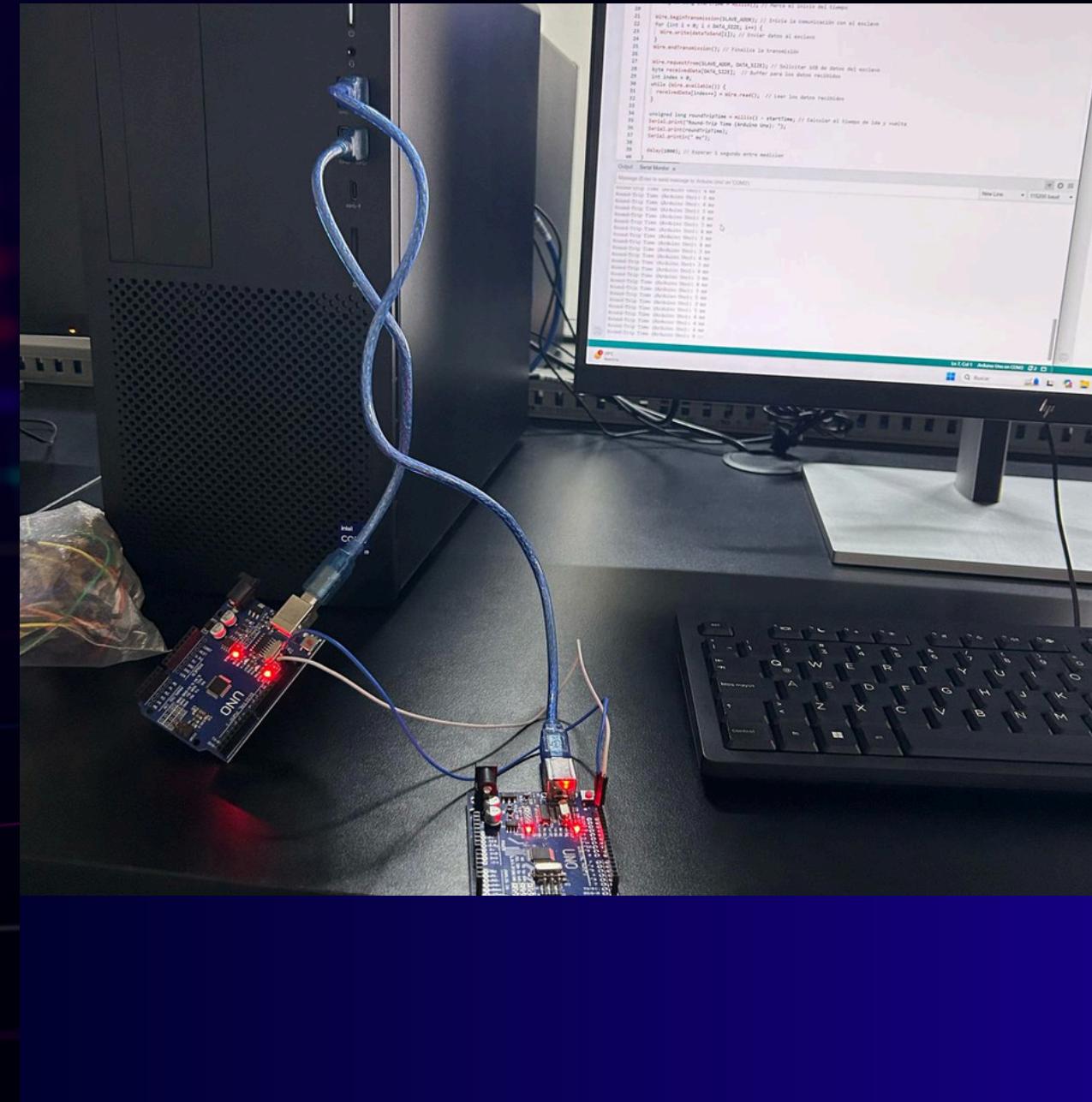




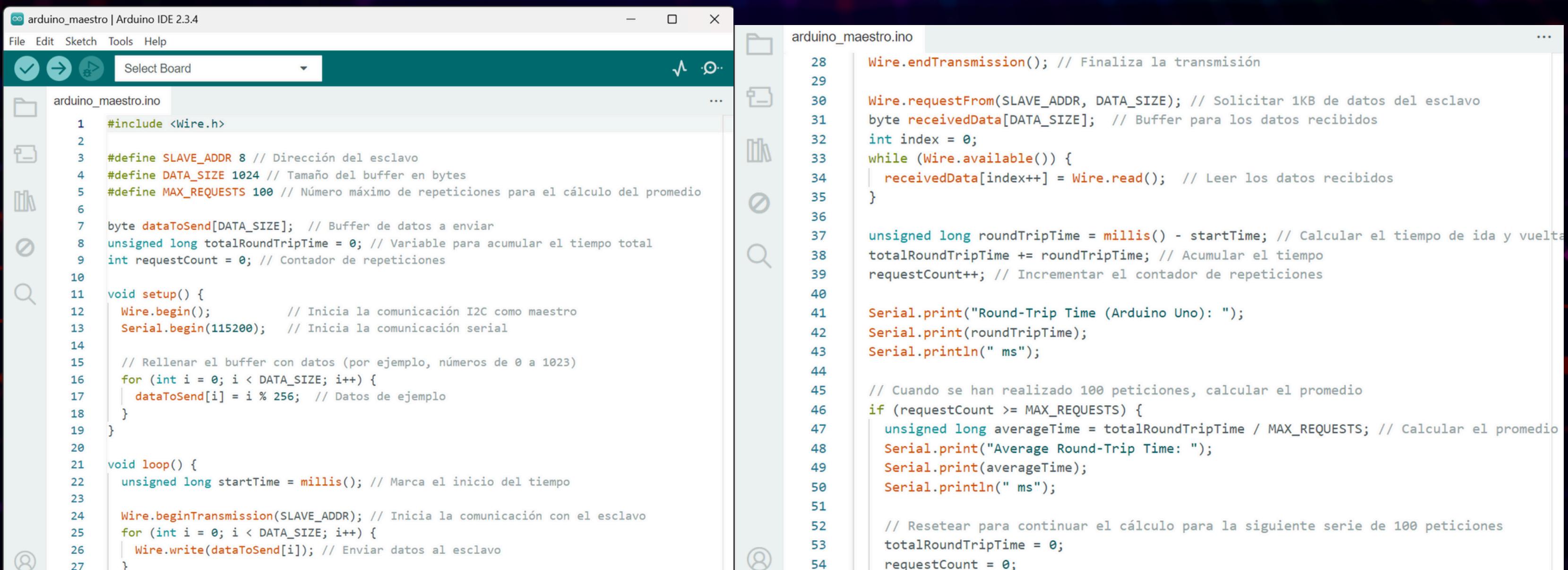
ARDUINO

En esta práctica, configuramos dos Arduinos para comunicarse usando I₂C. Un Arduino Maestro envía un bloque de datos al Arduino Esclavo, que los recibe y los devuelve cuando el maestro los solicita. La comunicación se monitorea por el puerto Serial para verificar que todo funcione correctamente.

El Maestro inicia la comunicación con la dirección 0x08, llena un buffer de un byte con valores del 0 al 255 y los envía al Esclavo. Luego mide el tiempo de ida y vuelta, solicita los datos y los imprime en el Serial. El Esclavo, configurado también con la dirección 0x08, guarda los datos recibidos y los devuelve cuando el Maestro los pide, mostrando un mensaje por Serial cada vez que responde.



CODIGOS ARDUINO MAESTRO

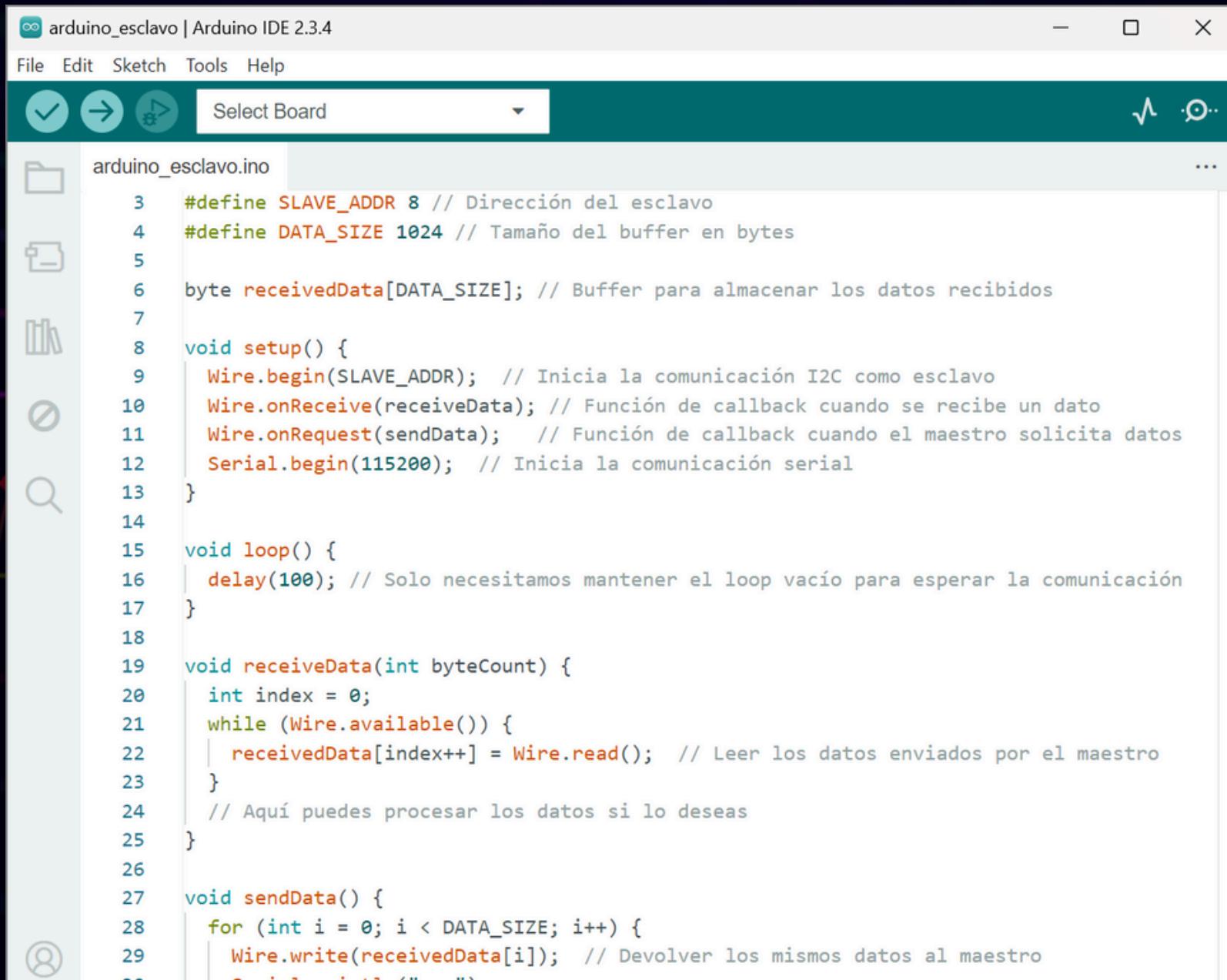


The image shows the Arduino IDE interface with the title bar "arduino_maestro | Arduino IDE 2.3.4". The left sidebar shows the file structure with "arduino_maestro.ino" selected. The main editor area displays the following C++ code:

```
arduino_maestro.ino
1 #include <Wire.h>
2
3 #define SLAVE_ADDR 8 // Dirección del esclavo
4 #define DATA_SIZE 1024 // Tamaño del buffer en bytes
5 #define MAX_REQUESTS 100 // Número máximo de repeticiones para el cálculo del promedio
6
7 byte dataToSend[DATA_SIZE]; // Buffer de datos a enviar
8 unsigned long totalRoundTripTime = 0; // Variable para acumular el tiempo total
9 int requestCount = 0; // Contador de repeticiones
10
11 void setup() {
12     Wire.begin(); // Inicia la comunicación I2C como maestro
13     Serial.begin(115200); // Inicia la comunicación serial
14
15     // Rellenar el buffer con datos (por ejemplo, números de 0 a 1023)
16     for (int i = 0; i < DATA_SIZE; i++) {
17         dataToSend[i] = i % 256; // Datos de ejemplo
18     }
19 }
20
21 void loop() {
22     unsigned long startTime = millis(); // Marca el inicio del tiempo
23
24     Wire.beginTransmission(SLAVE_ADDR); // Inicia la comunicación con el esclavo
25     for (int i = 0; i < DATA_SIZE; i++) {
26         Wire.write(dataToSend[i]); // Enviar datos al esclavo
27     }
28     Wire.endTransmission(); // Finaliza la transmisión
29
30     Wire.requestFrom(SLAVE_ADDR, DATA_SIZE); // Solicitar 1KB de datos del esclavo
31     byte receivedData[DATA_SIZE]; // Buffer para los datos recibidos
32     int index = 0;
33     while (Wire.available()) {
34         receivedData[index++] = Wire.read(); // Leer los datos recibidos
35     }
36
37     unsigned long roundTripTime = millis() - startTime; // Calcular el tiempo de ida y vuelta
38     totalRoundTripTime += roundTripTime; // Acumular el tiempo
39     requestCount++; // Incrementar el contador de repeticiones
40
41     Serial.print("Round-Trip Time (Arduino Uno): ");
42     Serial.print(roundTripTime);
43     Serial.println(" ms");
44
45     // Cuando se han realizado 100 peticiones, calcular el promedio
46     if (requestCount >= MAX_REQUESTS) {
47         unsigned long averageTime = totalRoundTripTime / MAX_REQUESTS; // Calcular el promedio
48         Serial.print("Average Round-Trip Time: ");
49         Serial.print(averageTime);
50         Serial.println(" ms");
51
52     // Resetear para continuar el cálculo para la siguiente serie de 100 peticiones
53     totalRoundTripTime = 0;
54     requestCount = 0;
```



CODIGOS ARDUINO ESCLAVO



The image shows a screenshot of the Arduino IDE 2.3.4 interface. The title bar reads "arduino_esclavo | Arduino IDE 2.3.4". The main area displays the following C++ code for an I2C slave:

```
#define SLAVE_ADDR 8 // Dirección del esclavo
#define DATA_SIZE 1024 // Tamaño del buffer en bytes

byte receivedData[DATA_SIZE]; // Buffer para almacenar los datos recibidos

void setup() {
  Wire.begin(SLAVE_ADDR); // Inicia la comunicación I2C como esclavo
  Wire.onReceive(receiveData); // Función de callback cuando se recibe un dato
  Wire.onRequest(sendData); // Función de callback cuando el maestro solicita datos
  Serial.begin(115200); // Inicia la comunicación serial
}

void loop() {
  delay(100); // Solo necesitamos mantener el loop vacío para esperar la comunicación
}

void receiveData(int byteCount) {
  int index = 0;
  while (Wire.available()) {
    receivedData[index++] = Wire.read(); // Leer los datos enviados por el maestro
  }
  // Aquí puedes procesar los datos si lo deseas
}

void sendData() {
  for (int i = 0; i < DATA_SIZE; i++) {
    Wire.write(receivedData[i]); // Devolver los mismos datos al maestro
  }
}
```

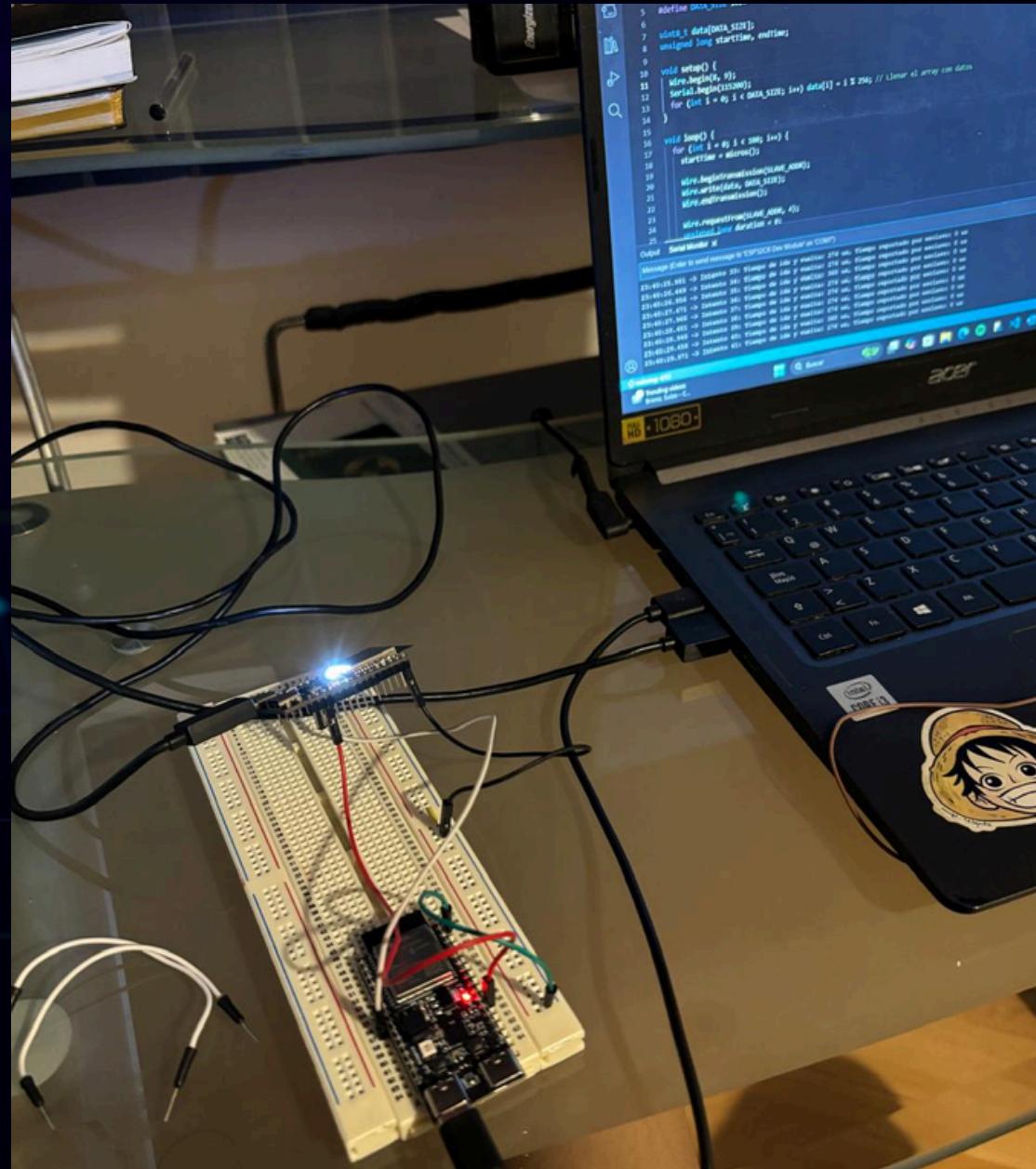
ESP32C6

SLIDE 7

En esta práctica, configuramos tambien dos ESP32C6 para comunicarse usando I2C.

Un Esp32C6 Maestro envía un bloque de datos al ESP32C6 Esclavo, que los recibe y los devuelve cuando el maestro los solicita. La comunicación se monitorea por el puerto Serial para verificar que todo funcione correctamente.

El Maestro inicia la comunicación con la dirección 0x08, llena un buffer de un byte con valores del 0 al 255 y los envía al Esclavo. Luego mide el tiempo de ida y vuelta, solicita los datos y los imprime en el Serial. El Esclavo, configurado también con la dirección 0x08, guarda los datos recibidos y los devuelve cuando el Maestro los pide, mostrando un mensaje por Serial cada vez que responde.





CODIGOS ESP32C6 MAESTRO/ESCLAVO

```
1 #include <Wire.h>
2 #define SLAVE_ADDR 0x08
3 #define DATA_SIZE 1024
4
5 uint8_t data[DATA_SIZE];
6 unsigned long startTime, endTime;
7
8 void setup() {
9     Wire.begin(8, 9);
10    Serial.begin(115200);
11    for (int i = 0; i < DATA_SIZE; i++) data[i] = i % 255; // Llenar el array con datos
12 }
13 void loop() {
14     for (int i = 0; i < 100; i++) {
15         startTime = micros();
16
17         Wire.beginTransmission(SLAVE_ADDR);
18         Wire.write(data, DATA_SIZE);
19         Wire.endTransmission();
20         Wire.requestFrom(SLAVE_ADDR, 4);
21         unsigned long duration = 0;
22
23         if (Wire.available() == 4) {
24             duration = Wire.read() | (Wire.read() << 8) | (Wire.read() << 16) | (Wire.read() << 24);
25         }
26         endTime = micros();
27         Serial.print("Intento ");
28         Serial.print(i + 1);
29         Serial.print(": Tiempo de ida y vuelta: ");
30         Serial.print(endTime - startTime);
31         Serial.print(" us");
32         delay(500);
33     }
34     while (1);
35 }
```

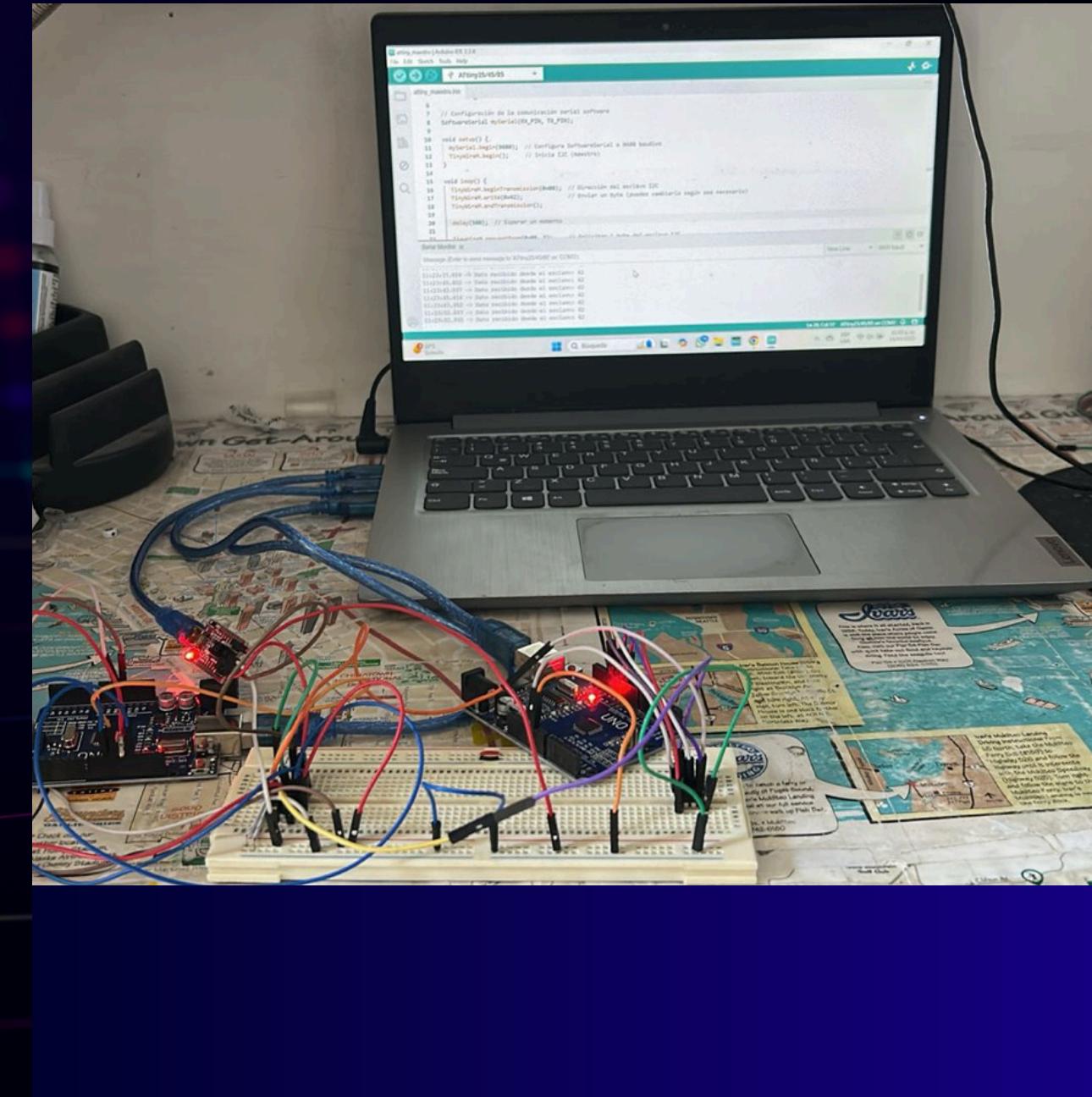
```
1 #include <Wire.h>
2 #define DATA_SIZE 1024
3
4 uint8_t buffer[DATA_SIZE];
5 volatile bool dataReceived = false;
6 unsigned long receiveTime;
7
8 void setup() {
9     Wire.begin(0x08, 8, 9);
10    Wire.onReceive(receiveEvent);
11    Wire.onRequest(requestEvent);
12    Serial.begin(115200);
13 }
14 void loop() {
15     if (dataReceived) {
16         receiveTime = micros();
17         dataReceived = false;
18     }
19     delay(100);
20
21     void receiveEvent(int bytes) {
22         int index = 0;
23         while (Wire.available() && index < DATA_SIZE) {
24             buffer[index++] = Wire.read();
25             Serial.println(" Tus");
26         }
27         dataReceived = true;
28     }
29     void requestEvent() {
30         unsigned long duration = micros() - receiveTime;
31         Wire.write((uint8_t*)&duration, sizeof(duration));
32         Serial.println(" us");
33     }
34 }
```



ATTINY85

En esta práctica, configuramos dos ATtiny85 para comunicarse entre sí mediante el protocolo I₂C, donde uno actúa como maestro y el otro como esclavo. Además, utilizamos SoftwareSerial en el maestro para enviar la información recibida a otro dispositivo, como una computadora o un microcontrolador.

El ATtiny85 maestro envía un dato (0x42) al esclavo a través de I₂C y luego solicita una respuesta. Si el esclavo responde con un dato, el maestro lo recibe y lo muestra en el puerto serial en formato hexadecimal. Todo este proceso se repite cada 500 milisegundos, asegurando una comunicación continua entre ambos microcontroladores.





CODIGOS ATTINY MAESTRO/ESCLAVO

attiny_maestro | Arduino IDE 2.3.4

```
File Edit Sketch Tools Help
ATtiny25/45/85

attiny_maestro.ino

1 #include <TinyWireM.h> // Librería para I2C con ATTiny85
2 #include <SoftwareSerial.h> // Librería para SoftwareSerial
3
4 #define RX_PIN 4 // Pin de recepción de datos (RX)
5 #define TX_PIN 3 // Pin de transmisión de datos (TX)
6
7 // Configuración de la comunicación serial software
8 SoftwareSerial mySerial(RX_PIN, TX_PIN);
9
10 void setup() {
11   mySerial.begin(9600); // Configura SoftwareSerial a 9600 baudios
12   TinyWireM.begin(); // Inicia I2C (maestro)
13 }
14
15 void loop() {
16   TinyWireM.beginTransmission(0x08); // Dirección del esclavo I2C
17   TinyWireM.write(0x42); // Enviar un byte (puedes cambiarlo según sea necesario)
18   TinyWireM.endTransmission();
19
20   delay(500); // Esperar un momento
21
22   TinyWireM.requestFrom(0x08, 1); // Solicitar 1 byte del esclavo I2C
23   if (TinyWireM.available()) {
24     byte receivedData = TinyWireM.read(); // Leer el byte recibido
25     mySerial.print("Datos recibidos desde el esclavo: ");
26     mySerial.println(receivedData, HEX); // Mostrar el dato recibido en formato hexadecimal
27   } else {
28     mySerial.println("Error: No se recibió respuesta.");
```

attiny_esclavo | Arduino IDE 2.3.4

```
File Edit Sketch Tools Help
ATtiny25/45/85

attiny_esclavo.ino

1 #include <TinyWireM.h>
2
3 #define DATA_SIZE 16 // Tamaño del buffer de datos (ajustado para evitar uso excesivo de RAM)
4 byte receivedData[DATA_SIZE]; // Buffer para almacenar los datos recibidos
5
6 void setup() {
7   TinyWireM.begin(); // Inicia I2C como esclavo
8 }
9
10 void loop() {
11   // Leer datos cuando el maestro envía información
12   if (TinyWireM.available()) {
13     for (int i = 0; i < DATA_SIZE && TinyWireM.available(); i++) {
14       receivedData[i] = TinyWireM.receive();
15     }
16   }
17
18   // Simulación de respuesta al maestro
19   TinyWireM.send(0x42); // Envía un byte de respuesta (ejemplo)
20 }
21
```

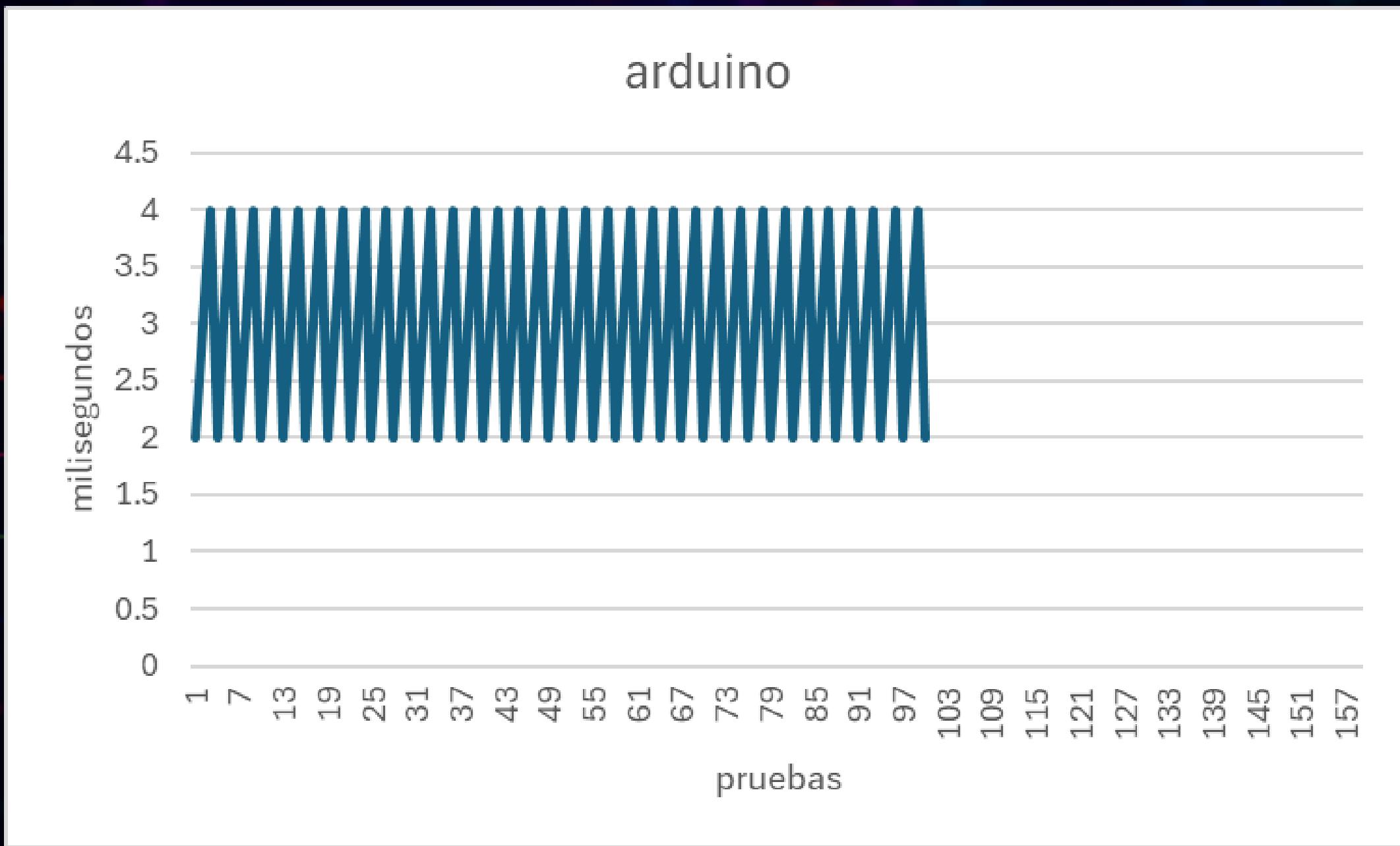


CONCLUSIONES

SLIDE 11

ARDUINO

tiene un promedio de 3ms





CONCLUSIONES

SLIDE 12

ESP32C6

Tiene un promedio de 291.44 (us)





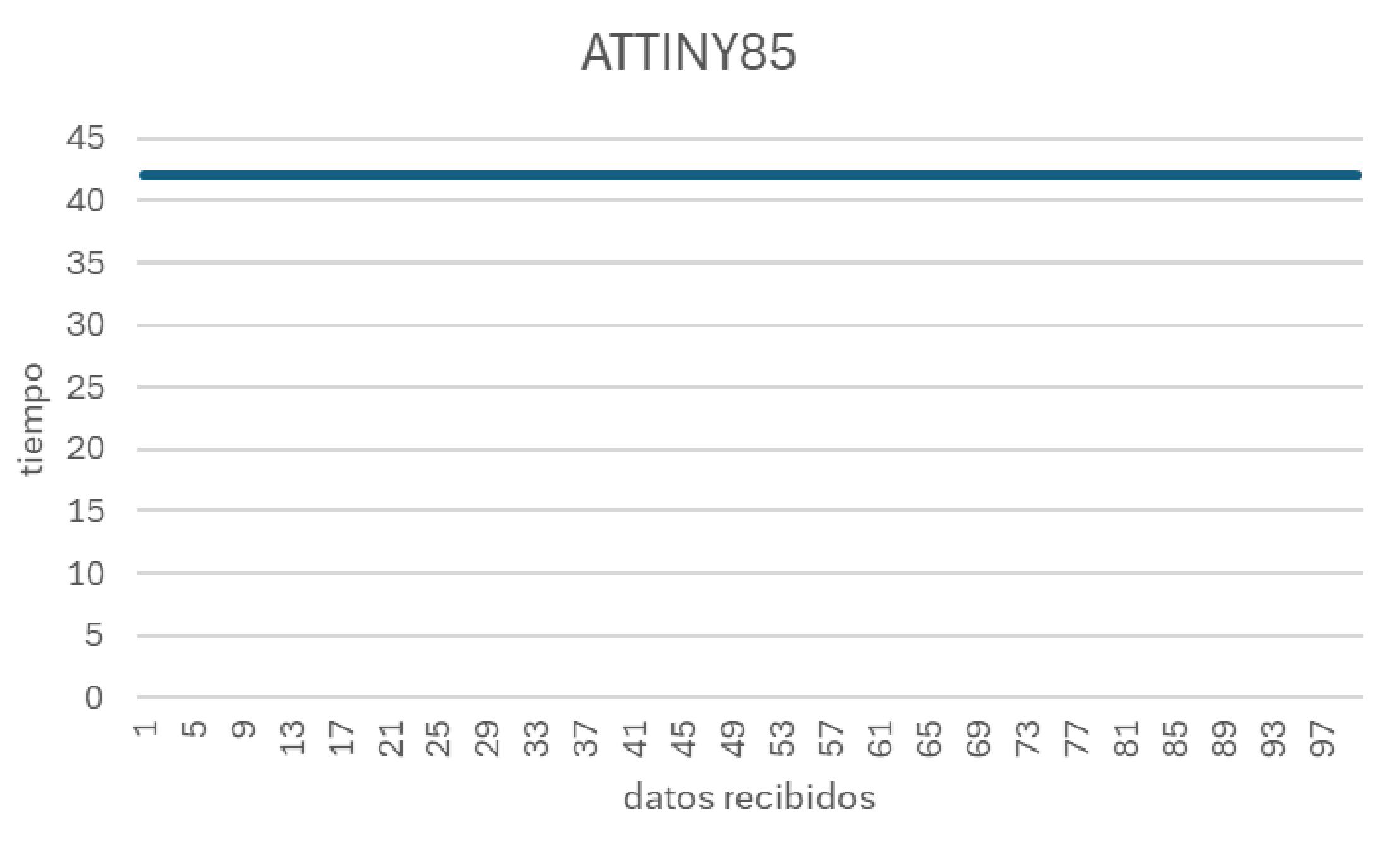
CONCLUSIONES

SLIDE 13

ATTINY

Tiene un promedio de 42ms

ATTINY85





CONCLUSIONES

SLIDE 14

	Reloj interno	I2C (prom)
ESP32C6	160 MHz	291.44 us
Arduino Uno	16MHz	3ms
Attiny85	8MHz	42ms



SLIDE 15



GRACIAS!