Primera Práctica de laboratorio

1st Camila Perez Mercado
Facultad de Ingeniería Electrónica
Universidad Santo Tomás
Bogotá, Colombia
camila.perez@usantotomas.edu.co

2nd Santiago Hernández Avila
Facultad de Ingeniería Electrónica
Universidad Santo Tomás
Bogotá, Colombia
santiagohernandeza@usantotomas.edu.co

3rd Diego Alejandro Rodriguez Guzman
Facultad de Ingeniería Electrónica
Universidad Santo Tomás
Bogotá, Colombia
diegoarodriguezg@usantotomas.edu.co

I. Introducción

La comunicación entre dispositivos es un aspecto fundamental para el desarrollo de aplicaciones que requieren intercambio eficiente y confiable de datos protocolos para ellos entender como funcionan el UART, SPI e I2C van a constituir una base solida e importante de la interconexión entre microcontroladores, computadoras de placa reducida y sensores, permitiendo construir sistemas integrados capaces de realizar tareas complejas de manera coordinada.

En esta práctica se explora la comunicación entre una Raspberry Pi y un Arduino Uno, mas sin enbargo se uso una raspberry pi pico 2W y un microcontrolador ESP8266MOD. Representan dos enfoques distintos de procesamiento que al complementarse permiten ampliar las posibilidades de control y procesamiento de información

De esta forma, el laboratorio busca comprenda la relevancia de las comunicaciones digitales, desarrolle habilidades en la configuración de interfaces y adquiera una visión integral de cómo distintos dispositivos pueden interactuar de manera eficiente en un sistema

II. Marco Teórico (Punto 1)

A. Repasando Conceptos(Raspberry Pi y Arduino 1)

1) Revisar la arquitectura del arduino 1, explicar lo siguiente:

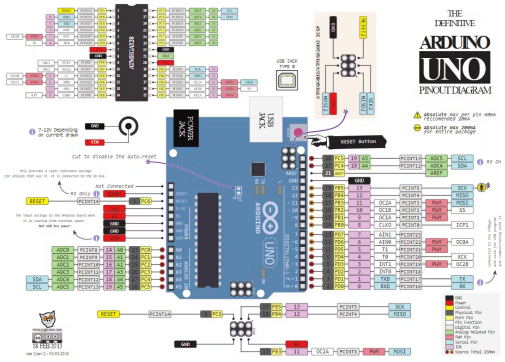

Fig. 1. Pinout Arduino Uno

¿Qué componentes principales integra un microcontrolador típico como el Atmega328P del arduino Uno y cómo interactúa?
Arquitectura del Arduino Uno (Atmega328P) componentes principales y su interacción: Un microcontrolador como el ATmega328P es un Sistema-en-un-Chip (SoC) que integra todos los componentes de una computadora básica en un solo circuito integrado:

- CPU (Unidad Central de Procesamiento): El núcleo de 8 bits que ejecuta las instrucciones del programa.
- Memoria Flash: Almacena el programa o firmware (32 KB).
- SRAM: Memoria volátil donde se almacenan las variables durante la ejecución (2 KB).
- EEPROM: Memoria no volátil para guardar datos que deben persistir después de apagarse (1 KB).
- Periféricos de Input/Output: Puertos de entrada/salida (GPIO), temporizadores/contadores, interfaz serie (UART), interfaz SPI, interfaz I²C, y un conversor analógico-digital (ADC).
- Reloj: Genera la señal de sincronización para la CPU y los periféricos.
- Interacción: La CPU lee el programa desde la memoria Flash, utiliza la SRAM para manipular datos, y controla los periféricos (como prender un LED a través de un GPIO o leer un sensor con el ADC) según las instrucciones del código. Todos los componentes se comunican a través de un bus interno sincronizado por el reloj.

¿Cómo se gestionan las interrupciones en Arduino?
Las interrupciones son señales que detienen temporalmente la ejecución normal del programa para ejecutar una rutina específica (ISR - Interrupt Service Routine). El ATmega328P tiene:

- Interrupciones hardware: Generadas por eventos externos (cambio de voltaje en un pin) o por periféricos internos (como un temporizador o la finalización de una comunicación serial).
- Interrupciones software: Generadas por instrucciones del programa.
- La gestión se hace mediante un controlador de

interrupciones que prioriza las señales y salta a la dirección de memoria de la ISR correspondiente.

¿Por qué el reloj del Arduino Uno es de 16 MHz?

- Es una velocidad de reloj equilibrada que permite un buen rendimiento para la mayoría de tareas de automatización y control para las que fue diseñado el Arduino, El reloj de 16 MHz define la frecuencia a la que se ejecutan las instrucciones. Como el ATmega328P necesita varios ciclos por instrucción, el rendimiento efectivo es de alrededor de 1 instrucción por cada 62,5 ns manteniendo un consumo de energía relativamente bajo y una operación estable sin requerir componentes complejos de refrigeración.

¿Qué implicaciones tiene en la velocidad de procesamiento?

- Implicaciones en la velocidad: Define la cantidad máxima de instrucciones (ciclos de reloj) que la CPU puede ejecutar por segundo. A 16 MHz, el ATmega328P ejecuta hasta 16 millones de instrucciones por segundo. Es una velocidad suficiente para aplicaciones de control en tiempo real, pero orders de magnitud más lenta que los procesadores de una Raspberry Pi genérica.

2) Revisar la arquitectura de la Raspberry Pi y explicar lo siguiente:

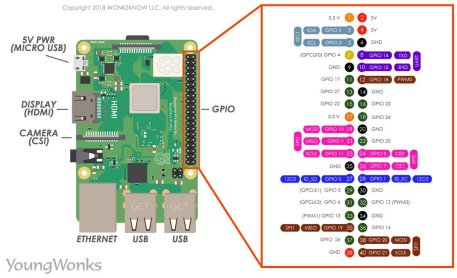

Fig. 2. Pinout Raspberry Pi 3

¿En qué se diferencia una Raspberry Pi (SoC) de un microcontrolador como el de Arduino?

- Raspberry Pi (SoC): Es una computadora de placa única. Su SoC (p.ej., Broadcom BCM2711) integra una CPU de 64 bits (ARM Cortex-A72), una GPU, memoria RAM (ej: 4 GB), y múltiples periféricos en un solo chip. Está diseñado para ejecutar sistemas operativos complejos y realizar múltiples tareas simultáneamente (multitarea).
- Arduino (Microcontrolador): Es un chip de computadora de un solo propósito. Diseñado para tareas específicas de control y lectura de sensores. Es más simple, de bajo consumo, y generalmente ejecuta un único programa (sketch) de forma repetitiva.
- ADC: Raspberry Pi vs. Arduino: A diferencia del Arduino, la mayoría de las Raspberry Pi genéricas no incluyen un ADC integrado en su SoC. Sus pines GPIO solo pueden leer señales digitales (0 o 1). Para leer señales analógicas (que varían entre 0V y 3.3V), se requiere un circuito externo, como un chip ADC (p.ej., el MCP3008), que se conecta a la Pi a través de un bus de comunicación como SPI o I²C y realiza la conversión.

¿Por qué la Raspberry Pi requiere un SO (ej: Linux) y Arduino no?

- La Pi tiene recursos de hardware avanzados que permiten ejecutar sistemas operativos multitarea (Linux).
- Arduino, al ser más limitado, ejecuta directamente el código sin necesidad de SO.

¿Qué es GPIO y cómo se gestiona en Raspberry Pi?

- GPIO (General Purpose Input/Output): Son pines que pueden configurarse programáticamente como entradas o salidas digitales para interactuar con otros circuitos.
- En Raspberry Pi se gestionan a través de bibliotecas en Linux (ej. RPi.GPIO en Python) que interactúan con el kernel para modificar registros internos del SoC.

Describe el papel del convertidor analógico-digital (ADC) en Arduino vs. Raspberry Pi: ¿Por qué la Pi requiere circuitos externos para leer señales analógicas?

- Arduino Uno tiene un ADC de 10 bits integrado que permite leer señales analógicas directamente.
- La Raspberry Pi no tiene ADC integrado, por lo que necesita un chip externo (ej. MCP3008) para convertir señales analógicas en digitales.

B. Repasando Conceptos (Raspberry Pi pico 2w y ESP8266MOD)

Aunque la práctica original se planteó utilizando una Raspberry Pi y un Arduino Uno, en nuestro caso se emplearon un microcontrolador ESP8266MOD y una Raspberry Pi Pico 2W para realizar los puntos de el laboratorio, con esta adaptación no solo se permitió realizar la comunicación entre dispositivos otros componentes, sino que también se aprendio de diferentes arquitecturas de microcontroladores y sus capacidades.

ESP8266MOD

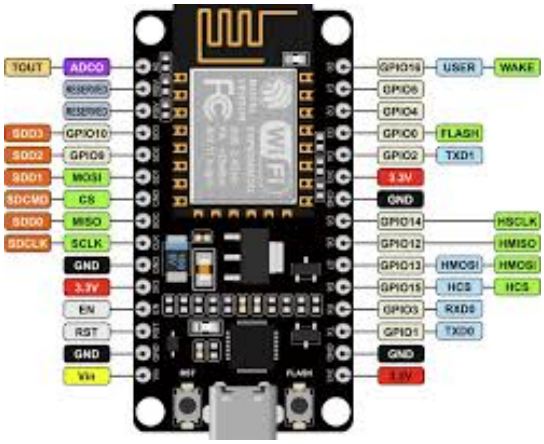

Fig. 3. Pinout ESP8266MOD

- Integra una CPU L106 a 80/160 MHz (más rápida que los 16 MHz del ATmega328P).
- Posee memoria SRAM y Flash externa para el programa.
- Incluye Wi-Fi integrado, lo cual amplía sus aplicaciones en IoT.
- Tiene GPIOs, UART, I2C, SPI y ADC de 10 bits, similar al Arduino, pero con más potencia de cómputo y conectividad.
- Las interrupciones funcionan de forma parecida: vectores de interrupción redirigen el flujo a rutinas específicas

A diferencia del Arduino Uno, incorpora un procesador más rápido, mayor capacidad de memoria y conectividad Wi-Fi integrada, lo cual lo convierte en una herramienta poderosa para aplicaciones de Internet de las Cosas.

Raspberry Pi Pico 2W

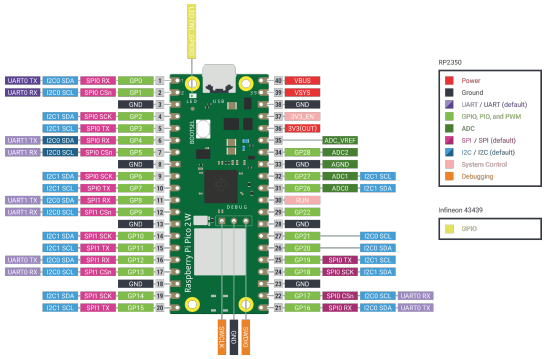

Fig. 4. Pinout Raspberry Pi Pico 2W

la Raspberry Pi Pico 2W se basa en el microcontrolador RP2040 de doble núcleo, Es un microcontrolador dual-core de 32-bit (ARM Cortex-M0+). que integra:

- Está basada en el RP2040, un microcontrolador (no un SoC como la Pi tradicional).
- Tiene doble núcleo ARM Cortex-M0+ a 133 MHz, memoria SRAM de 264 KB y Flash externa.
- Incluye pines GPIO programables, UART, SPI, I2C y ADC de 12 bits integrados.
- A diferencia de la Raspberry Pi con Linux, la Pico 2W no requiere sistema operativo completo, se programa directamente en C/C++ o MicroPython.
- Incorpora conectividad Wi-Fi en la versión 2W, lo que la hace más versátil en IoT.
- ADC: El RP2040 sí incluye un ADC integrado de 12 bits de resolución (4096 valores) .Esto es una ventaja sobre las Raspberry Pi genéricas y es comparable e incluso superior al ADC del Arduino Uno.

III. DESARROLLO

Punto 1: UART Asíncrono con Bits de Paridad

- Transmitir datos con detección de errores usando paridad, visualizando éxito/error con LEDs. Adaptando los codigos para los materiales usados en los pines indicados por nosotros

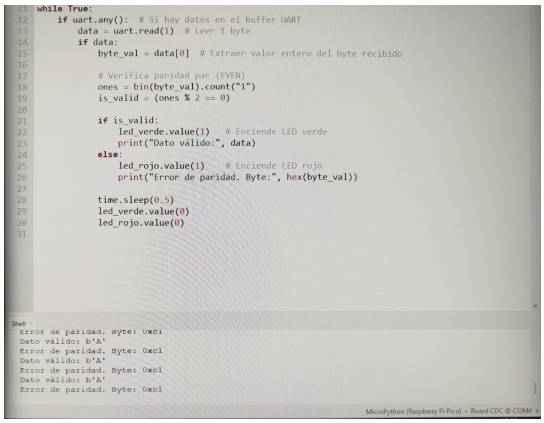

Fig. 5. Validacion en Thonny

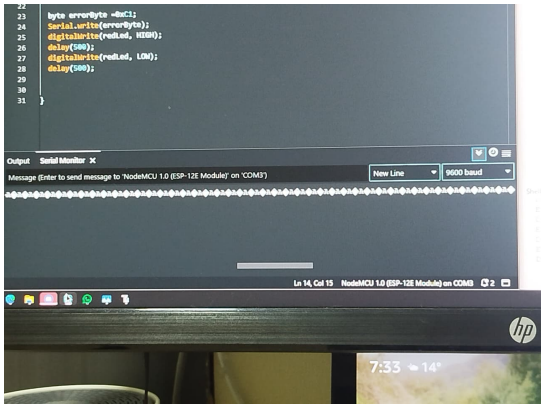

Fig. 6. Validacion en Arduino Exito/error

En este primer punto se buscaba la comunicación UART asíncrona entre la Raspberry Pi Pico 2W y el ESP8266MOD, con el objetivo de transmitir datos incluyendo un bit de paridad para la detección de errores. La señal recibida se valida mediante el encendido de LEDs, donde un LED azul representó la transmisión correcta y un LED rojo la detección de error.

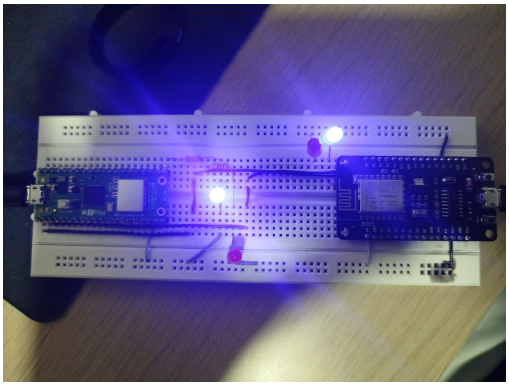

Fig. 7. Mensaje recibido correctamente

despues de que se pudieran comunicar correctamente se cargaba un prograna a la ESP simulando un error y al no ser el mensaje esperado por la raspberry se iluminaba el led rojo indicando Byte erróneo detectado

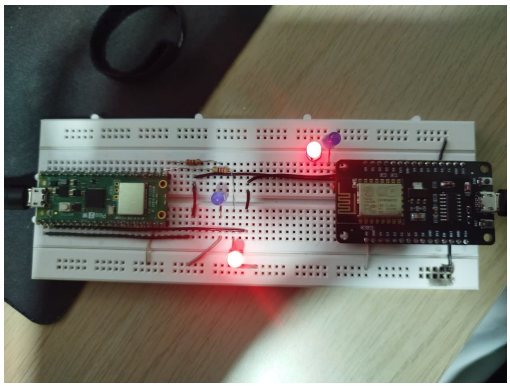

Fig. 8. Byte erroneo detectado

Apreciaciones importantes observadas:

- Si los baudios en ambos dispositivos no coinciden la comunicación falla y los datos no se reciben.
- La configuración de la paridad (par/impar) debe coincidir en ambos dispositivos; de lo contrario, el receptor marcará error en todos los envíos apreciado en la figura 5.

Punto 2: SPI Sincrónico (Raspberry como Maestro y Arduino como Esclavo)

En este punto buscamos implementar una comunicación SPI síncrona entre dos dispositivos, donde el maestro envía comandos y el esclavo los interpreta para encender o apagar un LED. La Pico se encargó de generar la señal de reloj (SCK) y enviar los datos por la línea MOSI.

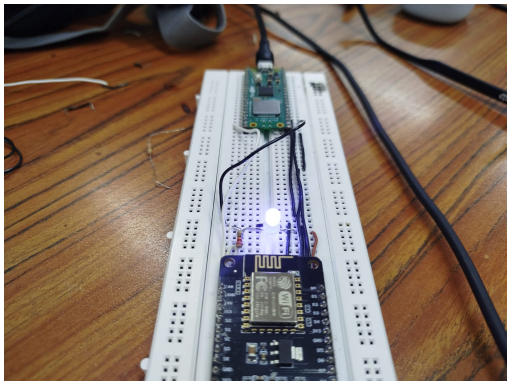

Fig. 9. ESP8266 recibe el byte encender el LED.

El ESP8266 recibió esa información a través de su pin configurado y encendió (Fig9) o apagó Fig (10) el LED conectado a un pin digital.

El protocolo SPI funciona de forma síncrona, lo que significa que los datos solo se transmiten al ritmo de la señal de

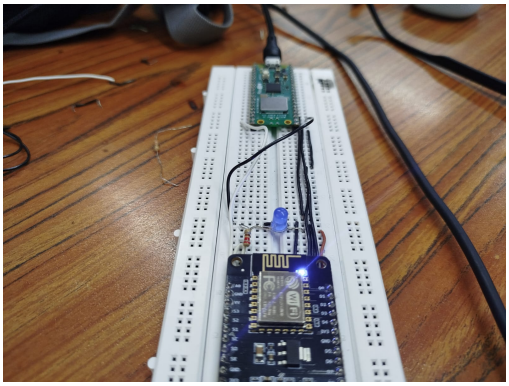

Fig. 10. ESP8266 recibe el byte apagar el LED.

reloj enviada por el maestro. Por eso, en este montaje, la Pico marca el ritmo y el ESP8266 solo responde cuando recibe la orden.

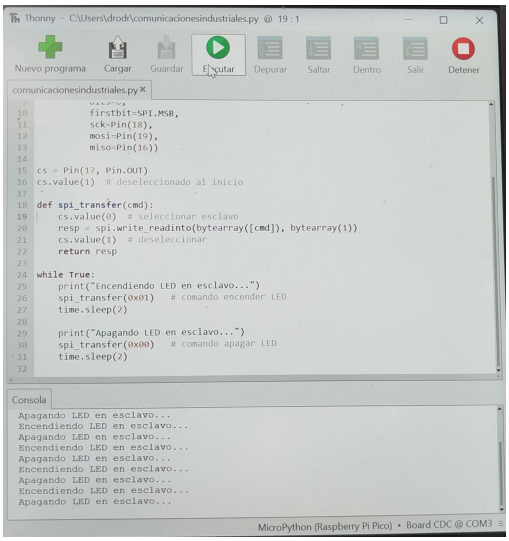

Fig. 11. Codigo Thonny

Apreciaciones importantes observadas:

- La Pico 2W trabaja con lógica de 3.3V, igual que el ESP8266, por lo que no hubo problema. Sin embargo, si uno de los dispositivos hubiera trabajado a 5V, habría sido necesario un conversor de niveles lógicos.
- Es fácil confundir las líneas MOSI y MISO. En este montaje, al no usar la comunicación de respuesta del esclavo, se priorizó la conexión MOSI (maestro → esclavo). Un error de conexión podría causar que no se encienda el LED.

Punto 3: I2C con Múltiples Dispositivos

En este punto se implementó la comunicación I2C entre dos dispositivos, con el objetivo de leer un potenciómetro en el esclavo (ESP8266) y mostrar su valor en el maestro (Pico 2W) mediante tres LEDs conectados en binario.

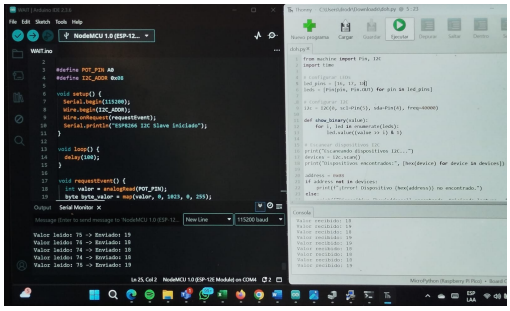

Fig. 12. Codigo Arduino y Thonny

La Pico 2W actuó como maestro, enviando las solicitudes de datos, mientras que el ESP8266 actuó como esclavo, leyendo el valor analógico del potenciómetro (conectado a su pin ADC) y enviándolo de regreso.

El valor del potenciómetro, que varía de 0 a 1023 (10 bits en el ADC), se simplificó para representarlo en 3 bits y luego mostrarse en los tres LEDs de la Pico, encendiéndose de manera binaria según el nivel detectado.

- Potenciómetro bajo.

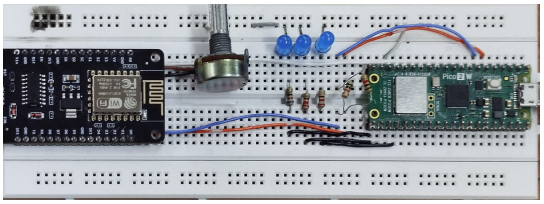

Fig. 13. LEDs muestran 000

- Potenciómetro medio.

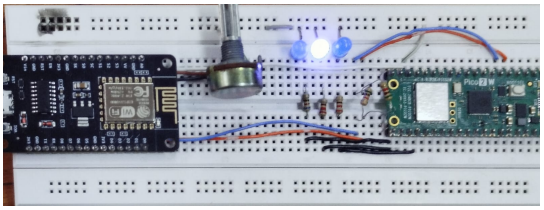

Fig. 14. LEDs muestran 010

- Potenciómetro alto.

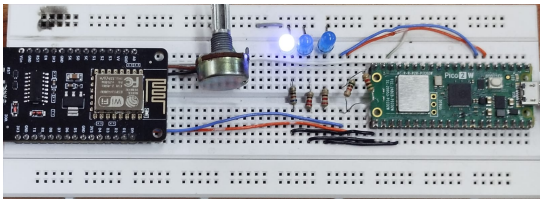

Fig. 15. LEDs muestran 111

Al mover el potenciómetro, los LEDs en la Pico cambiaban de estado de acuerdo al valor transmitido por el ESP8266 con esto se comprobó que la comunicación I2C era confiable, ya que la respuesta se actualizaba rápidamente con cada movimiento del potenciómetro y demostrando la utilidad de I2C para lectura de sensores y transmisión de datos entre dispositivos.

IV. CONLUSIONES

1) La práctica evidenció la importancia de comprender la capa física y los protocolos de comunicación serial en sistemas embebidos, ya que cada interfaz (UART, SPI, I2C) poseen una ventaja diferente.
2) El uso de la Raspberry Pi Pico 2W y el ESP8266 permitió demostrar cómo los microcontroladores modernos integran conectividad y periféricos avanzados, La correcta adaptación de los niveles de señal y pines fue esencial para lograr la comunicación sin errores.
3) Se observaron comportamientos diferenciales en cuanto a sincronización y robustez: UART, siendo asíncrono, puede presentar errores si no se configuran adecuadamente la paridad y la velocidad en ambos dispositivos; SPI mostró gran rapidez pero con mas conexiones y I2C permitió la comunicación maestro-esclavo con menos cables.