

You:

Explica que es el protocolo UART Y COMO FUNCIONA, EN PARTICULAR USA COMO EJEMPLO LA COMUNICACIÓN ENTRE UN ARDUINO UNO Y UNA PC VIA COMUNICACIÓN SERIAL A 115200 BAUDIOS

DeepSeek:

El protocolo **UART** (Universal Asynchronous Receiver/Transmitter) es un sistema de comunicación serial asíncrona ampliamente utilizado para transferir datos entre dispositivos digitales. A diferencia de protocolos síncronos, no utiliza una señal de reloj compartida, sino que ambos dispositivos deben acordar previamente la velocidad de comunicación (**baud rate**) y el formato de los datos. A continuación, te explico cómo funciona, usando como ejemplo la comunicación entre un **Arduino Uno** y una **PC** a **115200 baudios**.

Partes clave del protocolo UART

1. **Transmisión asíncrona:** No hay señal de reloj; se sincroniza usando los bits de inicio y parada.
 2. **Dos líneas de comunicación:**
 - **TX (Transmit):** Envía datos desde el Arduino hacia la PC.
 - **RX (Receive):** Recibe datos en el Arduino desde la PC.
 3. **Frame de datos:** Cada bloque de datos enviado sigue una estructura específica:
 - **Bit de inicio:** Siempre en nivel bajo (0).
 - **Datos:** 5 a 9 bits (generalmente 8 bits para un byte).
 - **Bit de paridad (opcional):** Para detección de errores (no usado en el ejemplo).
 - **Bits de parada:** 1 o 2 bits en nivel alto (1).
-

Funcionamiento paso a paso (Ejemplo Arduino-PC a 115200 baudios)

1. Configuración inicial

- El Arduino Uno y la PC acuerdan una velocidad de **115200 baudios** (≈ 115200 bits por segundo).
- Cada bit dura: $1 / 115200 \approx 8.68 \mu s$.
- Se usan **8 bits de datos, sin paridad, 1 bit de parada** (configuración típica: 8N1).

2. Envío de datos desde Arduino a PC

- **Ejemplo:** Enviar el carácter A (código ASCII 65 o binario 01000001).
 - **Paso 1:** La línea TX está en **reposo (alto, 1)**.
 - **Paso 2:** Arduino envía el **bit de inicio (0)** durante $8.68 \mu s$.
 - **Paso 3:** Envía los **8 bits de datos** (LSB primero: $1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0$).
 - **Paso 4:** Envía el **bit de parada (1)** durante $8.68 \mu s$.
 - **Paso 5:** La línea vuelve a reposo (1) hasta el próximo byte.

3. Recepción en la PC

- La PC detecta el **flanco de bajada** del bit de inicio y espera **1.5 veces el tiempo de bit** (para muestrear en el centro de cada bit).
- Muestrea la línea cada **$8.68 \mu s$** para leer los bits de datos.
- Reconstruye el byte y verifica el bit de parada.

4. Comunicación bidireccional

- La PC puede responder al Arduino usando la misma lógica a través de la línea **RX** del Arduino.
-

Implementación en Arduino Uno

Código de ejemplo

cpp

Copy

Download

```
void setup() {
  Serial.begin(115200); // Configura comunicación a 115200 baudios
}

void loop() {
  Serial.println("Hola PC!"); // Envía datos a la PC
  delay(1000);

  if (Serial.available()) { // Si la PC envía datos
    char dato = Serial.read(); // Lee el byte recibido
    Serial.print("Recibido: ");
    Serial.println(dato);
  }
}
```

Conexión física

- El Arduino Uno tiene un chip **ATmega16U2** que convierte la señal UART (TTL) a **USB**.
- **TX del Arduino** → Conectado al **RX de la PC** (vía USB).
- **RX del Arduino** → Conectado al **TX de la PC** (vía USB).

Consideraciones clave

- **Sincronización:** Ambos dispositivos deben usar el mismo **baud rate**. Un error del 5% suele ser tolerable.
- **Niveles lógicos:** Arduino usa **5V** para representar 1 y 0. La PC usa niveles USB (3.3V), pero el conversor USB-serial se encarga de la adaptación.
- **Ruido:** No hay detección de errores integrada (a menos que se use el bit de paridad).

Ventajas y desventajas

- **Ventajas:** Simple, ampliamente compatible, bajo costo.
- **Desventajas:** Sin control de flujo nativo, susceptible a ruido si los baud rates no coinciden.

Este protocolo es la base de la comunicación serial en entornos embebidos y permite interactuar con sensores, módulos y computadoras de manera eficiente.

You:

Dado un UNO R3 (que podría ser o no un arduino uno original) Explica características técnicas, como su adc de 0 a 255, voltaje de funcionamiento, sus salidas analogicas y digitales, y enumera algunos de sus perifericos. Explica brevemente que es y como se originó el framework Arduino, de este mismo que hace exactamente la función AnalogWrite

DeepSeek:

Te voy a explicar las características del Arduino Uno R3 (original o compatible), el framework Arduino y la función analogWrite().

Características técnicas del Arduino Uno R3

Microcontrolador principal

- **Modelo:** ATmega328P
- **Arquitectura:** 8-bit AVR
- **Velocidad de reloj:** 16 MHz
- **Memoria Flash:** 32 KB (0.5 KB para bootloader)
- **SRAM:** 2 KB
- **EEPROM:** 1 KB

Sistema de alimentación

- **Voltaje de operación:** 5V
- **Voltaje de entrada recomendado:** 7-12V (a través de jack DC)
- **Límite máximo de voltaje:** 20V (pero no recomendado)
- **Regulador de voltaje incorporado:** 5V y 3.3V

Entradas Analógicas (ADC)

- **Número de pines:** 6 pines (A0 a A5)
- **Resolución:** 10 bits (0-1023), no 8 bits como mencionas
 - Rango de medición: 0-5V
 - Precisión: 4.9 mV por unidad (5V/1024)
- **Características especiales:**
 - Múltiplexado entre los 6 canales
 - Tiempo de conversión: ~100 µs

Salidas Digitales

- **Número total:** 14 pines digitales (D0-D13)
- **Pines con PWM:** 6 pines (marcados con ~: 3, 5, 6, 9, 10, 11)
- **Corriente máxima por pin:** 40 mA (recomendado 20 mA)
- **Corriente total máxima:** 200 mA para todo el chip

Salidas "Analógicas" (PWM)

- **Resolución:** 8 bits (0-255)
- **Frecuencia base:** ~ 490 Hz (pines 5 y 6: ~ 980 Hz)
- **Técnica:** Modulación por Ancho de Pulso (PWM)

Comunicación

- **UART:** 1 puerto (RX=0, TX=1)
- **I2C:** 1 bus (A4=SDA, A5=SCL)
- **SPI:** 1 bus (10=SS, 11=MOSI, 12=MISO, 13=SCK)

Periféricos principales

1. **Timer/Counters:** Tres temporizadores (Timer0, Timer1, Timer2)
2. **Watchdog Timer:** Para recuperación de fallos
3. **Interrupciones externas:** 2 pines (2 y 3)
4. **Comparador analógico**
5. **Power-save modes:** Múltiples modos de bajo consumo

El Framework Arduino: Origen y Función

Origen e Historia

- **Año de creación:** 2005
- **Lugar:** Ivrea, Italia (Instituto de Diseño Interactivo Ivrea)
- **Creadores:** Massimo Banzi, David Cuartielles, Tom Igoe, David Mellis y Gianluca Martino
- **Motivación:** Crear una plataforma de prototipado electrónica accesible para estudiantes y diseñadores sin profundos

- **Motivación:** Crear una plataforma de prototipado electrónica accesible para estudiantes y diseñadores sin profundos conocimientos de electrónica

Características del Framework

- **Lenguaje basado en C/C++** simplificado
 - **API simplificada:** Funciones como `pinMode()`, `digitalWrite()`, `analogRead()`
 - **Abstracción de hardware:** Oculta registros y configuraciones complejas
 - **Entorno de desarrollo integrado (IDE):** Gratuito y de código abierto
 - **Sistema de librerías:** Extensible mediante librerías de terceros
 - **Bootloader preinstalado:** Permite programación via USB
-

¿Qué hace exactamente `analogWrite()`?

Función real

- **NO es una verdadera salida analógica:** Genera una señal PWM (Modulación por Ancho de Pulso)
- **Principio de funcionamiento:** Ciclos rápidos de encendido/apagado que simulan un voltaje analógico

Ejemplo técnico

cpp

Copy

Download

```
analogWrite(9, 128); // 50% duty cycle en pin 9
```

Resultado eléctrico:

- Voltaje promedio: 2.5V (50% de 5V)
- Señal generada:
 - 50% del tiempo a 5V
 - 50% del tiempo a 0V
 - Frecuencia: ≈ 490 Hz

Implementación hardware

- **Basada en temporizadores:** Timer0 (pines 5,6), Timer1 (pines 9,10), Timer2 (pines 3,11)
- **Registros involucrados:**
 - TCCRnA/B: Configuran modo y preescaler
 - OCRnA/B: Establecen el ciclo de trabajo
- **Resolución fija:** 8 bits (0-255) por limitaciones hardware

Limitaciones

- **Frecuencia fija:** No configurable via software estándar
- **Interferencia:** Los pines que comparten timer afectan su frecuencia
- **Verdadero analógico:** Requiere circuito externo (filtro RC) para suavizar la señal

Alternativas para verdadera salida analógica

- **Conversor DAC externo** (ej. MCP4725)
- **Filtro paso bajo:** Convierte PWM a voltaje DC estable
- **Software avanzado:** Manipulación directa de registros para mayor control

El Arduino Uno R3 sigue siendo la placa más popular para educación y prototipado gracias a su equilibrio entre capacidades

El Arduino Uno R3 sigue siendo la placa mas popular para educacion y prototipado gracias a su equilibrio entre capacidades, facilidad de uso y comunidad de apoyo.

You:

agregale un tono mas academico a tu respuesta y citas en apa 7, puntos extras si pones los repositorios de github (?)

DeepSeek:

Protocolo UART y su implementación en Arduino Uno: Comunicación Serial a 115200 Baudios con PC

Introducción al Protocolo UART

El **Universal Asynchronous Receiver/Transmitter (UART)** es un protocolo de comunicación serial asíncrono ampliamente utilizado para transferir datos entre dispositivos digitales. A diferencia de los protocolos síncronos, el UART no emplea una señal de reloj compartida, sino que depende de la configuración coordinada de parámetros como la velocidad de transmisión (*baud rate*) y el formato de los datos para lograr la sincronización (Analog Devices, 2021; Rohde & Schwarz, s.f.). Su simplicidad y bajo costo lo hacen ideal para aplicaciones embebidas, como la comunicación entre microcontroladores y periféricos (Circuit Basics, s.f.).

Fundamentos Técnicos del UART

1. Arquitectura y Principios de Operación

- **Comunicación Asíncrona:** No requiere señal de reloj. La sincronización se logra mediante bits de inicio y parada, y un *baud rate* predefinido (Analog Devices, 2021).
- **Configuración de Tramas:** Cada trama incluye:
 - **Bit de inicio:** Transición de alto a bajo (1 → 0) para indicar el inicio de la transmisión.
 - **Datos:** 5-9 bits (generalmente 8 bits para un byte).
 - **Bit de paridad (opcional):** Verificación de errores mediante paridad par o impar.
 - **Bits de parada:** 1-2 bits en nivel alto (1) para señalar el fin de la trama (Rohde & Schwarz, s.f.).
- **Velocidad de Baud:** Define bits transmitidos por segundo (bps). Para 115200 baudios, el tiempo por bit es $\approx 8.68 \mu s$ (DeepBlue Embedded, s.f.).

2. Implementación en Hardware

- **Dos líneas de comunicación:**
 - **TX (Transmit):** Envía datos desde el transmisor.
 - **RX (Receive):** Recibe datos en el receptor.
- **Niveles lógicos:** UART utiliza niveles TTL (5V o 3.3V), a diferencia de protocolos como RS-232 ($\pm 12V$) (DeepBlue Embedded, s.f.).

3. Sincronización y Tolerancia a Errores

- La diferencia máxima permitida en el *baud rate* entre dispositivos es del 10% para evitar errores de temporización (Analog Devices, 2021).
- El bit de paridad detecta errores individuales en la transmisión, pero es limitado para errores múltiples (Circuit Basics, s.f.).

Ejemplo de Comunicación Arduino Uno-PC a 115200 Baudios

Configuración Hardware

- **Arduino Uno:** Basado en el microcontrolador ATmega328P, cuenta con un UART hardware con pines dedicados:
 - **TX (Pin 1):** Transmisión de datos a la PC.
 - **RX (Pin 0):** Recepción de datos desde la PC.
- **Conversión USB-TTL:** El Arduino Uno incluye un chip ATmega16U2 que convierte señales UART (TTL) a USB, permitiendo la comunicación con la PC mediante un puerto virtual COM (Circuito.io, 2017).

Configuración Software

1. Inicialización del Serial:

cpp

Copy

Download

```
void setup() {  
  Serial.begin(115200); // Configura UART a 115200 bps  
}
```

- El *baud rate* debe coincidir en Arduino y la PC (DeepBlue Embedded, s.f.).

2. Transmisión de Datos:

- Ejemplo: Envío del carácter A (ASCII 65, binario 01000001):
 - **Bit de inicio (0) → 8 bits de datos (LSB first: 1 0 0 0 0 0 1 0) → Bit de parada (1).**
 - Duración total: ≈86.8 µs por trama (10 bits incluidos inicio/parada).

3. Recepción en PC:

- La PC detecta el flanco de bajada del bit de inicio y muestrea la línea cada 8.68 µs.
- Software como el Monitor Serial de Arduino IDE decodifica los datos.

Ejemplo de Código

cpp

Copy

Download

```
void setup() {  
  Serial.begin(115200); // Inicializa UART a 115200 bps  
}  
  
void loop() {  
  Serial.println("Hola PC"); // Envía datos  
  if (Serial.available()) {  
    char dato = Serial.read(); // Lee datos entrantes  
    Serial.print("Recibido: ");  
    Serial.println(dato);  
  }  
}
```

Consideraciones Técnicas y Mejores Prácticas

1. Precisión del Baud Rate:

- Para 115200 bps, cristales de 16 MHz en Arduino garantizan error mínimo (<0.1%) (Analog Devices, 2021).

2. Buffer de Datos:

- El UART del ATmega328P tiene buffers de hardware de 64 bytes para recepción/transmisión, evitando pérdida de datos a alta velocidad (DeepBlue Embedded, s.f.).

3. Protocolos Personalizados:

- Se pueden implementar tramas con encabezados, trailers y CRC para mejorar la seguridad y confiabilidad (Analog Devices, 2021).

4. Limitaciones:

- **Distancia:** UART es susceptible a ruido en distancias largas. Soluciones como RS-485 extienden el rango.
 - **Velocidad:** Máximo típico de 1-2 Mbps en microcontroladores (Rohde & Schwarz, s.f.).
-

Repositorios de GitHub Relevantess

1. ArduinoCore-avr:

- Implementación de librerías UART para Arduino Uno.

- URL: <https://github.com/arduino/ArduinoCore-avr>

2. SoftwareSerial:

- Emulación de UART por software para añadir puertos seriales adicionales.
- URL: <https://github.com/arduino-libraries/SoftwareSerial>

3. Proyecto de Ejemplo:

- Comunicación Arduino-PC con visualización de datos.
- URL: <https://github.com/arduino/arduino-examples>

Conclusión

El protocolo UART es un pilar en la comunicación serial asíncrona, destacando por su simplicidad y eficiencia en aplicaciones embebidas. Su implementación en Arduino Uno, combinada con la conversión USB-TTL, permite una integración seamless con PCs, facilitando prototipado y depuración. La configuración a 115200 baudios optimiza la velocidad para la mayoría de aplicaciones, aunque requiere atención a la sincronización y gestión de errores. Para proyectos avanzados, la extensión mediante librerías de software o protocolos personalizados amplía su utilidad.

Referencias

Analog Devices. (2021). *UART: A Hardware Communication Protocol*. Analog Dialogue.

<https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html>

Circuit Basics. (s.f.). *Basics of UART Communication*. <https://www.circuitbasics.com/basics-uart-communication/>

Circuito.io. (2017). *Arduino Uno Pinout Guide*. <https://www.circuito.io/blog/arduino-uno-pinout/>

DeepBlue Embedded. (s.f.). *Arduino UART Example & Tutorial*. <https://deepbluembedded.com/arduino-uart-example-tutorial/>

Rohde & Schwarz. (s.f.). *Understanding UART*. https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html

Este análisis sintetiza los principios técnicos del UART y su aplicación práctica en Arduino, proporcionando una base académica para el desarrollo de proyectos de comunicación serial. Para detalles avanzados, se recomienda consultar los repositorios y referencias citadas.

You:

Dado un • Cooler FAN brushless 80x80x25mm 12V DC de 0,16 v genérico de origen chino, guiado por un pwm de un arduino uno (0-255) mosfet mediante, cuyo máximo voltaje es 12 v. Explica con detalle: -funcionamiento -relacion voltaje pwm/caudal/rpm -Todo lo que consideres relevante con citas en apa 7

You:

falta colocar lo del funcionamiento del software y periodos de muestreo :v