Microcontroladores II

Escuela de Ingenierías

IUSH Institución Universitaria SALAZAR Y HERRERA

SER PERSONA DESDE EL SABER Y EL HACER

Conozcamonos

Juan Esteban Giraldo Hoyos
Ingeniero Electrónico
Magíster en Gestión de Ciencia, Tecnología e Innovación
Director de I+D+i en Helo
CEO en arqueros.co
Email:
juan.giraldoho@comunidad.iush.edu.co

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Contenido temático y cronograma del curso

No. De sesión y Fecha	Actividad de Trabajo directo	Horas TD	Actividad de Trabajo independiente	Horas TI	Actividad evaluativa	% de Evaluación
1 (05/04/2024)	Exposición por parte del docente de los temas: - Polling vs Interrupciones - Interrupciones de software y hardware en sistemas embebidos - Modos de bajo consumo	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	10%
2 (12/04/2024)	Exposición por parte del docente de los temas: -Interfaces de software para comunicación con sistemas embebidos usan python y la comunicación serial	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	10%
3 (19/04/2024)	Exposición por parte del docente de los temas: - Diseño de interfaces gráficas interactivas - Programación de interfaces gráficas interactivas	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	10%
4 (26/04/2024)	Exposición por parte del docente de los temas: - ¿Qué son sistemas operativos en tiempo real y su estructura? - Introducción a sistema operativo para microntroladores FREERTOS	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	20%
5 (03/05/2024)	Exposición por parte del docente de los temas: - Interacción con FREERTOS	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	
6 (10/05/2024)	Exposición por parte del docente de los temas: Aplicaciones de sistemas embedidos: - Servidor web para consulta - Comunicación por radiofrecuencia	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	15%
7 (17/05/2024)	Exposición por parte del docente de los temas: Aplicaciones de sistemas embedidos: - Sistema de control de acceso	6	Videos de YouTube / Lecturas sugeridas / Ejercicios de repaso y simulación	12	Actividad práctica	15%
8 (24/05/2024)	Presentación de Proyectos finales del curso	6	Implementación del proyecto final del curso	12	Actividad práctica	20%

Horario de clase

Bloque 1: 6pm - 7:30pm

Receso: 7:30pm - 8:00pm

Bloque 2: 8:00pm - 9:30pm

Escuela de Ingenierías

IUSH Institución Universitaria SALAZAR Y HERRERA

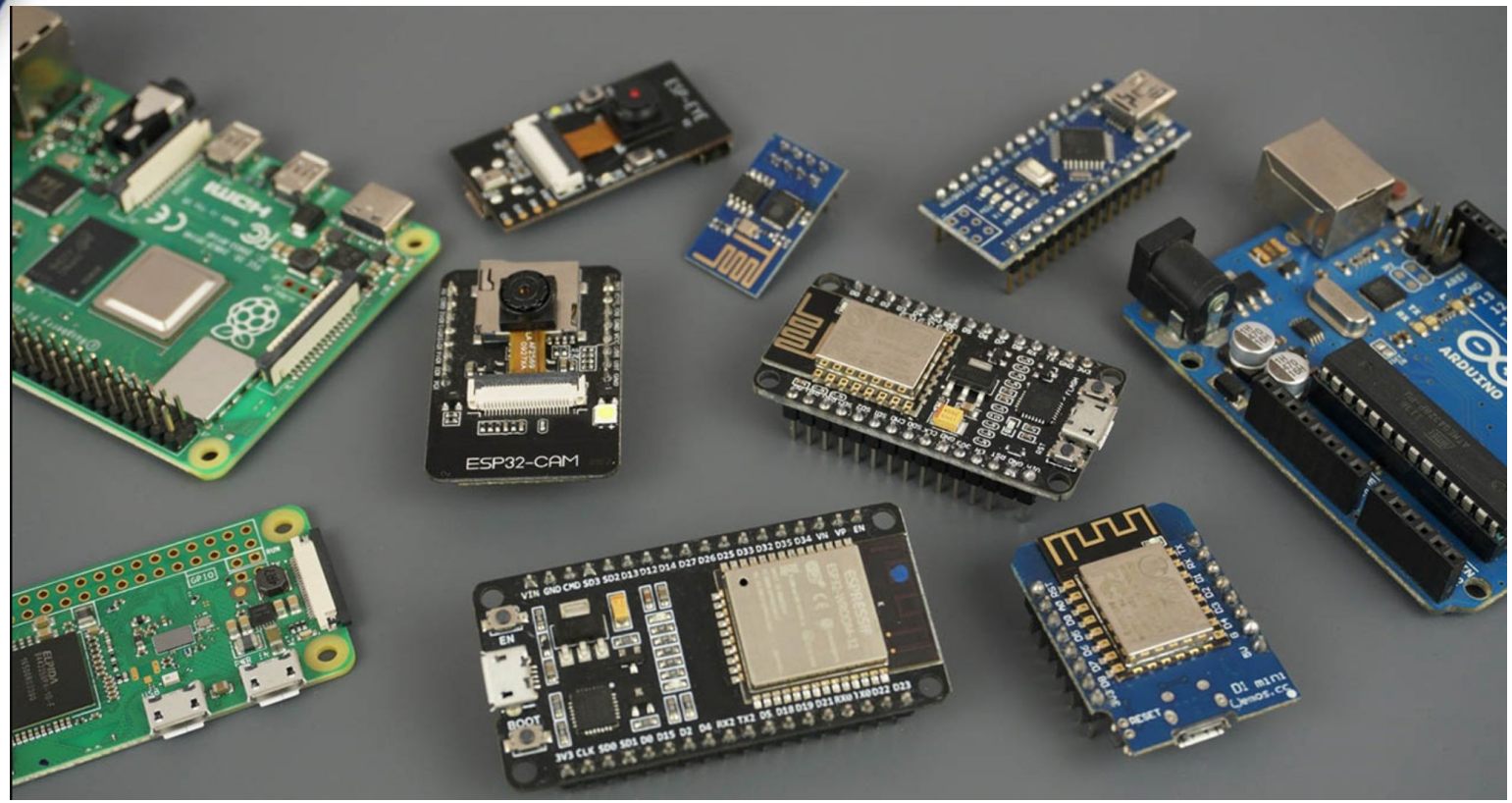
SER PERSONA DESDE EL SABER Y EL HACER

Proyectos de Aula

Los proyectos de Aula están encaminados a resolver unos retos propuestos por el docente que serán socializados la semana 3. Estos proyectos se presentan de forma grupal (máximo 3 personas) y se sustentan la última semana del curso

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II



Los microcontroladores son dispositivos compactos que integran procesador, memoria y periféricos en un solo chip, diseñados para ejecutar tareas específicas en sistemas embebidos, como controlar dispositivos electrónicos o recopilar datos. “El cerebro orquestador”

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling

Programación basada en Polling

En la programación basada en polling, un programa o proceso verifica regularmente el estado de un recurso (como un dispositivo de entrada/salida) para determinar si ha ocurrido algún evento o si se han completado ciertas operaciones. El proceso de polling es activo y continuo, ya que el programa verifica repetidamente si hay eventos nuevos.

Microcontroladores II - Polling

Programación basada en Polling - Ejemplo

POLLING1

```
// Definimos los pines
const int buttonPin = 8;  // Pin del botón
const int ledPin = 13;    // Pin del LED

// Variables para almacenar el estado anterior del botón y el estado actual
int lastButtonState = LOW;
int buttonState;

void setup() {
  // Configuramos el pin del botón como entrada
  pinMode(buttonPin, INPUT);
  // Configuramos el pin del LED como salida
  pinMode(ledPin, OUTPUT);
}
```

```
void loop() {
  // Leemos el estado actual del botón
  buttonState = digitalRead(buttonPin);

  // Verificamos si ha ocurrido un cambio en el estado del botón
  if (buttonState != lastButtonState) {
    // Si ha ocurrido un cambio, actualizamos el estado del LED
    if (buttonState == HIGH) {
      // Si el botón está presionado, encendemos el LED
      digitalWrite(ledPin, HIGH);
    } else {
      // Si el botón está suelto, apagamos el LED
      digitalWrite(ledPin, LOW);
    }

    // Actualizamos el estado anterior del botón
    lastButtonState = buttonState;
  }
}
```

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Interrupciones

Programación basada en Interrupciones

En la programación basada en interrupciones, el control del programa es transferido a una rutina de manejo de interrupciones cuando ocurre un evento específico, como una solicitud de entrada/salida o una señal de temporizador. Esto permite que el procesador maneje múltiples tareas de manera eficiente, ya que puede responder a eventos externos sin tener que esperar activamente o realizar polling continuo.

Microcontroladores II - Interrupciones

Programación basada en Interrupciones - Ejemplo

INTERRUPCIONES1

```
// Definimos los pines
const int buttonPin = 2;  // Pin del botón
const int ledPin = 13;    // Pin del LED

// Variable para almacenar el estado del botón
volatile int buttonState = LOW;

void setup() {
  // Configuramos el pin del botón como entrada
  pinMode(buttonPin, INPUT);
  // Configuramos el pin del LED como salida
  pinMode(ledPin, OUTPUT);

  // Configuramos la interrupción para el pin del botón
  attachInterrupt(digitalPinToInterrupt(buttonPin), buttonInterrupt, CHANGE);
}

void loop() {
  // No hay necesidad de hacer nada en el bucle principal cuando se usan interrupciones
}

// Función de interrupción para el botón
void buttonInterrupt() {
  // Leemos el estado actual del botón
  buttonState = digitalRead(buttonPin);

  // Si el botón está presionado (estado HIGH), encendemos el LED
  // Si el botón está suelto (estado LOW), apagamos el LED
  digitalWrite(ledPin, buttonState);
}
```

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling Vs Interrupciones

Característica	Programación basada en Polling	Programación basada en Interrupciones
Eficiencia	Menos eficiente, ya que implica un uso continuo de recursos del procesador para verificar el estado de los eventos.	Más eficiente, ya que el procesador solo responde a eventos cuando ocurren, lo que minimiza el uso de recursos durante períodos de inactividad.
Latencia	Mayor latencia, ya que los eventos pueden no ser detectados inmediatamente, dependiendo de la frecuencia de polling.	Menor latencia, ya que el procesador responde a los eventos tan pronto como ocurren, lo que minimiza el tiempo de espera.
Complejidad del código	Puede resultar en código más simple y directo, ya que la lógica de control está integrada en el bucle principal del programa.	Puede ser más complejo, ya que requiere la implementación de rutinas de manejo de interrupciones y la gestión del flujo de control entre la rutina de interrupción y el programa principal.
Consumo de CPU	Mayor consumo de CPU, ya que el proceso de polling requiere que la CPU esté activamente involucrada en la verificación del estado de los eventos.	Menor consumo de CPU, ya que la CPU solo se activa cuando ocurre un evento, lo que permite un uso más eficiente de los recursos del sistema.
Escalabilidad	Puede ser menos escalable, ya que el rendimiento del sistema puede degradarse significativamente cuando se agregan más dispositivos o eventos a monitorear.	Más escalable, ya que el sistema puede manejar eficientemente un mayor número de dispositivos y eventos debido a la naturaleza reactivo de las interrupciones.

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Interrupciones

Definición de Interrupciones de Hardware y Software:

1. **Interrupciones de Hardware**:
   - Son interrupciones generadas por eventos físicos externos al microcontrolador, como un cambio de voltaje en un pin específico o la finalización de una operación de E/S.
   - Estas interrupciones son gestionadas directamente por el hardware del microcontrolador y pueden tener prioridad sobre el código en ejecución.
   - Las interrupciones de hardware pueden ser esenciales para la detección de eventos en tiempo real y para la gestión eficiente de tareas sensibles al tiempo.
2. **Interrupciones de Software**:
   - Son interrupciones generadas por instrucciones de software dentro del programa en ejecución.
   - Estas interrupciones se utilizan generalmente para realizar tareas programadas, como la generación de intervalos de tiempo regulares o la ejecución de rutinas específicas en respuesta a ciertas condiciones de estado.
   - Aunque son gestionadas por el software, pueden ser utilizadas para mejorar la eficiencia y la precisión de las operaciones del microcontrolador, como en el caso de temporizadores y contadores.

Microcontroladores II - Interrupciones

Interrupciones de Hardware en Arduino Uno:

1. **External Interrupts (INT0 y INT1)**:
   - El Arduino Uno tiene dos pines específicos para interrupciones externas: **D2 (INT0)** y **D3 (INT1)**.
   - Estos pines pueden ser configurados para detectar cambios de flanco o niveles específicos de voltaje y activar una interrupción.
   - Las interrupciones externas pueden ser útiles para detectar eventos en tiempo real, como pulsos o cambios en sensores.

Interrupciones de Software en Arduino Uno:

1. **Timer Interrupts (Timers)**:
   - El ATmega328P cuenta con tres temporizadores (Timer0, Timer1 y Timer2), cada uno con capacidades de generar interrupciones.
   - Estos temporizadores pueden ser configurados para generar interrupciones en intervalos regulares, lo que es útil para realizar tareas programadas, medición del tiempo, etc.

Microcontroladores II - Interrupciones

Microcontrolador	Interrupciones de Hardware	Interrupciones de Software
Arduino Uno	External Interrupts (INT0, INT1)	Timer Interrupts (Timers)
Arduino Mega	External Interrupts (INT0 - INT3, INT4 - INT7)	Timer Interrupts (Timers)
		Pin Change Interrupts
NodeMCU (ESP8266)	External Interrupts (GPIO)	Timer Interrupts (Timers)
		Software Interrupts (ESP8266SoftwareTimerInterrupt)
ESP32	External Interrupts (GPIO)	Timer Interrupts (Timers)
		Pin Change Interrupts
		Task Scheduler Interruptions (RTOS)
		Touch Sensor Interrupts
		Software Interrupts (esp_task_wdt_isr_register)
Raspberry Pi Pico W	PIO (Programmable I/O) Interrupts	Timer Interrupts (Timers)
		Programmable Interrupts (IRQs in PIO state machine)

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling vs Interrupciones

POLLING2

```
// Definimos el pin del sensor de temperatura
const int sensorPin = A0;

// Variable para almacenar el valor de la temperatura
int temperature;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);
}

void loop() {
  // Sensamos la temperatura cada 5 segundos
  delay(5000);

  // Leemos el valor del sensor de temperatura
  temperature = analogRead(sensorPin);

  // Convertimos el valor leído a temperatura en grados Celsius
  float voltage = temperature * (5.0 / 1023.0);
  float tempCelsius = (voltage - 0.5) * 100.0;

  // Mostramos la temperatura por el puerto serial
  Serial.print("Temperatura: ");
  Serial.print(tempCelsius);
  Serial.println(" grados Celsius");
}
```

Montar un sensor de temperatura lm35 e implementar el siguiente código para sensar la temperatura cada 5 seg y mostrar por comunicación serial

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling vs Interrupciones

POLLING2

```
// Definimos el pin del sensor de temperatura
const int sensorPin = A0;

// Variable para almacenar el valor de la temperatura
int temperature;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);
}

void loop() {
  // Sensamos la temperatura cada 5 segundos
  delay(5000);

  // Leemos el valor del sensor de temperatura
  temperature = analogRead(sensorPin);

  // Convertimos el valor leído a temperatura en grados Celsius
  float voltage = temperature * (5.0 / 1023.0);
  float tempCelsius = (voltage - 0.5) * 100.0;

  // Mostramos la temperatura por el puerto serial
  Serial.print("Temperatura: ");
  Serial.print(tempCelsius);
  Serial.println(" grados Celsius");
}
```

Al código anterior agregar el montaje de un led y de un botón. Usando la programación por polling encender el led cuando se detecte la pulsación del botón y apagar el led cuando se suelte el botón. La temperatura sensada se debe consultar cada 5 seg igual cómo está en el código inicial. **¿Qué logramos identificar?**

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling vs Interrupciones

INTERRUPCIONES2

```
#include <TimerOne.h>

// Definimos el pin del sensor de temperatura
const int sensorPin = A0;

// Variable para almacenar el valor de la temperatura
volatile int temperature;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);

  // Configuramos Timer1 para generar una interrupción cada 5 segundos
  Timer1.initialize(5000000); // Intervalo de 5 segundos en microsegundos
  Timer1.attachInterrupt(timerInterrupt); // Asociamos la función de interrupción al temporizador
}

void loop() {
  // No hay necesidad de hacer nada en el bucle principal cuando se usan interrupciones
}

// Función de interrupción del temporizador
void timerInterrupt() {
  // Leemos el valor del sensor de temperatura
  temperature = analogRead(sensorPin);

  // Convertimos el valor leído a temperatura en grados Celsius
  float voltage = temperature * (5.0 / 1023.0);
  float tempCelsius = (voltage - 0.5) * 100.0;

  // Mostramos la temperatura por el puerto serial
  Serial.print("Temperatura: ");
  Serial.print(tempCelsius);
  Serial.println(" grados Celsius");
}
```

Implementar el siguiente código para sensar la temperatura cada 5 seg y mostrar por comunicación serial.
Usando interrupciones para el timer

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Polling vs Interrupciones

INTERRUPCIONES2

```
#include <TimerOne.h>

// Definimos el pin del sensor de temperatura
const int sensorPin = A0;

// Variable para almacenar el valor de la temperatura
volatile int temperature;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);

  // Configuramos Timer1 para generar una interrupción cada 5 segundos
  Timer1.initialize(5000000); // Intervalo de 5 segundos en microsegundos
  Timer1.attachInterrupt(timerInterrupt); // Asociamos la función de interrupción al temporizador
}

void loop() {
  // No hay necesidad de hacer nada en el bucle principal cuando se usan interrupciones
}

// Función de interrupción del temporizador
void timerInterrupt() {
  // Leemos el valor del sensor de temperatura
  temperature = analogRead(sensorPin);

  // Convertimos el valor leído a temperatura en grados Celsius
  float voltage = temperature * (5.0 / 1023.0);
  float tempCelsius = (voltage - 0.5) * 100.0;

  // Mostramos la temperatura por el puerto serial
  Serial.print("Temperatura: ");
  Serial.print(tempCelsius);
  Serial.println(" grados Celsius");
}
```

Al código anterior agregar el montaje de un led y de un botón. Usando la programación por **interrupciones** encender el led cuando se detecte la pulsación del botón y apagar el led cuando se suelte el botón. La temperatura sensada se debe consultar cada 5 seg igual cómo está en el código inicial. **¿Qué logramos identificar?**

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Modos de bajo consumo

Varios microcontroladores tienen modos de bajo consumo que pueden ser activados para reducir el consumo de energía cuando el microcontrolador no está realizando tareas activas. Uno de los modos de bajo consumo comúnmente utilizado es el modo de "Sueño Profundo" (Deep Sleep Mode), que puede ser activado para minimizar el consumo de energía mientras se espera una interrupción.

Modo de Operación	Consumo de Corriente	Funcionamiento
Activo	19-22 mA	Ejecuta el código normalmente y espera instrucciones.
Sueño Profundo	50-100 µA	Se suspende la ejecución del programa y el consumo de energía se minimiza hasta que ocurre una interrupción.

Microcontroladores II - Modos de bajo consumo

Wake-up Sources	Idle	ADC Noise Reduction	Power-save	Standby	Power-down
INT1, INT0 and Pin Change	X	X	X	X	X
TWI Address Match	X	X	X	X	X
Timer2	X	X	X		
SPM/EEPROM Ready	X	X			
A/D Converter	X	X			
Watchdog Timer	X	X	X	X	X
Other I/O	X				

Dependiendo del modo Sleep se tiene la posibilidad de despertar el microcontrolador de diferentes fuentes

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Modos de bajo consumo

BAJO_CONSUMO1

```
#include <avr/sleep.h>
#include <avr/power.h>

#define interruptPin 2

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT); //Usar led interno de la placa arduino uno
  pinMode(interruptPin, INPUT_PULLUP); //Poner el pin de interrupción en modo Pullup
  digitalWrite(LED_BUILTIN,HIGH); //Encender led
}

void loop() {
  // Este código no se ejecutará de forma repetitiva ya que el microcontrolador entra en modo de bajo consumo
  // Llamamos a Serial.println('.') una vez antes de entrar en el modo de bajo consumo
  Serial.println('.');
  sleeping();
}

void sleeping(){
    // Mensaje de depuración
  Serial.println("Entrando en el modo de bajo consumo...");
  // Esperamos unos segundos para asegurarnos de que hay suficiente tiempo para abrir el monitor serial
  delay(1000);
  digitalWrite(LED_BUILTIN,LOW);
  //set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Seleccionamos el modo de bajo consumo adecuado
  set_sleep_mode(SLEEP_MODE_PWR_SAVE); // Seleccionamos el modo de bajo consumo adecuado
  cli();
  sleep_enable();
  sleep_bod_disable();
  sei();
  sleep_cpu();
}
```

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

PRÁCTICA #1 10%

Implementar un firmware que permita sensar la temperatura de un sensor lm35 y muestre por serial la temperatura sensada. La medición de la temperatura la debe hacer cuando se genere una **interrupción en bajo** para el pin 2. El programa debe permanecer el resto del tiempo en **un modo de bajo consumo**. Ver las siguientes diapositivas para tomar el firmware de ejemplo y agregar lo que falte

Microcontroladores II - Modos de bajo consumo

BAJO_CONSUMO2

```
#include <avr/sleep.h>
#include <avr/power.h>

#define interruptPin 2

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT); //Usando led interno de Arduino Uno
  pinMode(interruptPin, INPUT_PULLUP);
  digitalWrite(LED_BUILTIN,HIGH);
}

void loop() {
  // Este código no se ejecutará ya que el microcontrolador está en modo de bajo consumo
  // Llamamos a Serial.println('.') una vez antes de entrar en el modo de bajo consumo
  Serial.println('.');
  sleeping();
}
```

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Microcontroladores II - Modos de bajo consumo

```
void sleeping(){ //Entrando a modo de bajo consumo
    // Mensaje de depuración
  Serial.println("Entrando en el modo de bajo consumo...");
  // Esperamos unos segundos para asegurarnos de que hay suficiente tiempo para abrir el monitor serial
  delay(1000);
  digitalWrite(LED_BUILTIN,LOW);
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Seleccionamos el modo de bajo consumo adecuado
  //set_sleep_mode(SLEEP_MODE_PWR_SAVE); // Seleccionamos el modo de bajo consumo adecuado
  cli();
  sleep_enable();
  sleep_bod_disable();
  sei();
  // Configuramos la interrupción para el pin del botón
  attachInterrupt(digitalPinToInterrupt(interruptPin), buttonInterrupt, LOW); //Cuando la señal llegue en BAJO
  sleep_cpu();
  digitalWrite(LED_BUILTIN,HIGH);
}

void buttonInterrupt(){ //Rutina de atención de interrupción de pin 2
  // Mensaje de depuración
  Serial.println("Entrando a interrupcion...");
  // Esperamos unos segundos para asegurarnos de que hay suficiente tiempo para abrir el monitor serial
  delay(1000);
  sleep_disable();
  detachInterrupt(digitalPinToInterrupt(interruptPin));
}
```

Escuela de Ingenierías
IUSH Institución Universitaria SALAZAR Y HERRERA
SER PERSONA DESDE EL SABER Y EL HACER

Bibliografía

- https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/
- https://www.arduino.cc/reference/en/libraries/timerinterrupt/