

# Lab Sheet for Week 8

Lab Instructor: RSP

## Lab 1: Measuring ATmega328P Sleep Mode Current Consumption

### Objectives

- Write bare-metal C code to configure the **ATmega328P** on an **Arduino Uno board** to enter **sleep mode**, with periodic **wake-up** using the **Watchdog Timer (WDT)** or an external interrupt (active-low push button).
- Use a digital multimeter to measure the current consumption during sleep mode.

### Lab Procedure

In this lab, the provided **AVR C code** (in **Code Listing 1**) shows how to place the MCU into its **lowest-power sleep mode** and wake it periodically using the **Watchdog Timer** at an interval of approximately **4 or 8 seconds**.

1. Create a new project using **Arduino IDE** and add the `main.c` code in a new tab.
  - 1.1 Select **Arduino Uno** as the target board.
  - 1.2 Copy and paste the **AVR C code** into the `main.c`.
  - 1.3 Connect the **Arduino Uno** to the host computer using a USB cable.
  - 1.4 Build and upload the generated .hex file to the target board.
2. Provide an external **DC power supply** to the board.
  - 2.1 Disconnect the **USB cable** from the **Arduino Uno**.
  - 2.2 Apply a **DC power supply of +9V to +12V** to the **VIN pin** of the **Arduino Uno** (see the pin map of the Arduino Uno board in **Figure 1**), and connect the supply ground to the **GND** of the board.
3. Current consumption measurement.
  - 3.1 Use a **DMM** for current measurement, and select the **µA or mA range** as appropriate.
  - 3.2 Measure the current flow from the external power supply to the **VIN** pin of the **Arduino** board.
  - 3.3 Observe and record both the lowest and highest current values indicated by the **DMM** and record the following measurements:
    - Peak current (during **LED ON** and active state): \_\_\_\_\_ mA.
    - Sleep current (during sleep mode, **LED OFF**): \_\_\_\_\_ µA or mA.

## Notes:

- The **Arduino Uno** board has a **power-on LED** and a **USB-to-serial circuit** that also draws current continuously and contributes to the measured sleep current of the board.
- Figure 2** shows the schematic of the **Arduino Uno** board, and **Figure 3** shows the 5V linear voltage regulator circuit powered from the **VIN** pin and the power-on LED circuit.

## ⚠️ CAUTIONS

- Ensure **correct polarity** when connecting **external DC power supply**.
- Set **DMM** to appropriate current range before connecting to circuit.
- Never connect power supply while **DMM** is in voltage mode.

4. Revise the C code to wake up the **AVR MCU** using a **push button** configured as an **external interrupt**.
- 4.1 Choose the **PD2 (INT0)** pin and connect it to a push button (active-low) on the breadboard. The INT0 interrupt should trigger on a falling edge.
- 4.2 After a power reset or system reset, the MCU waits for about 1 second, and then enters sleep mode. When the push button is pressed, the MCU wakes up, turns on the onboard LED, and disables sleep mode.

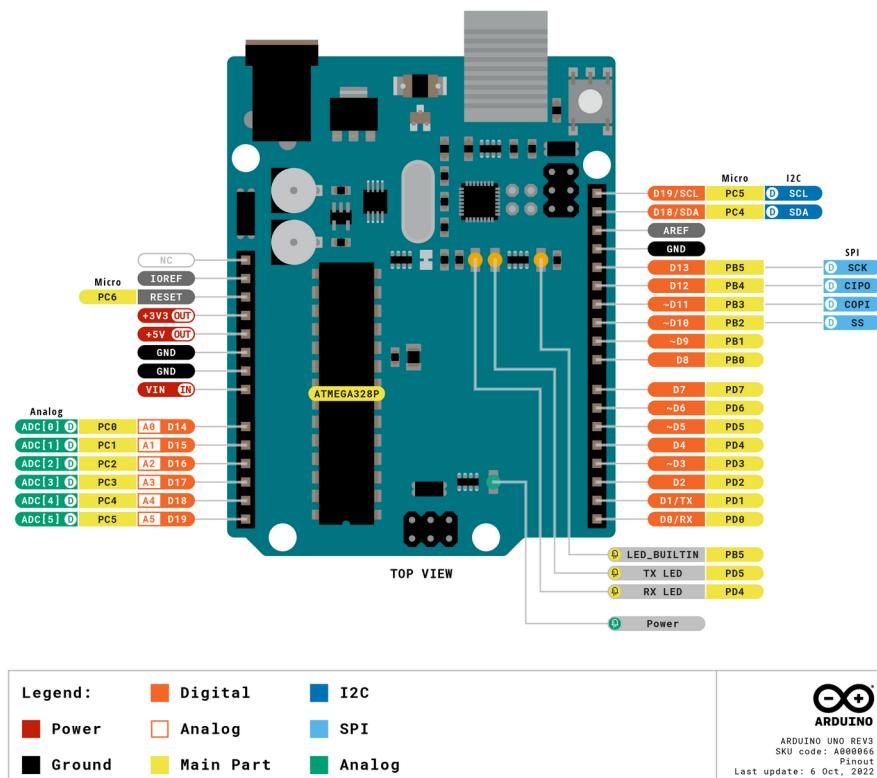
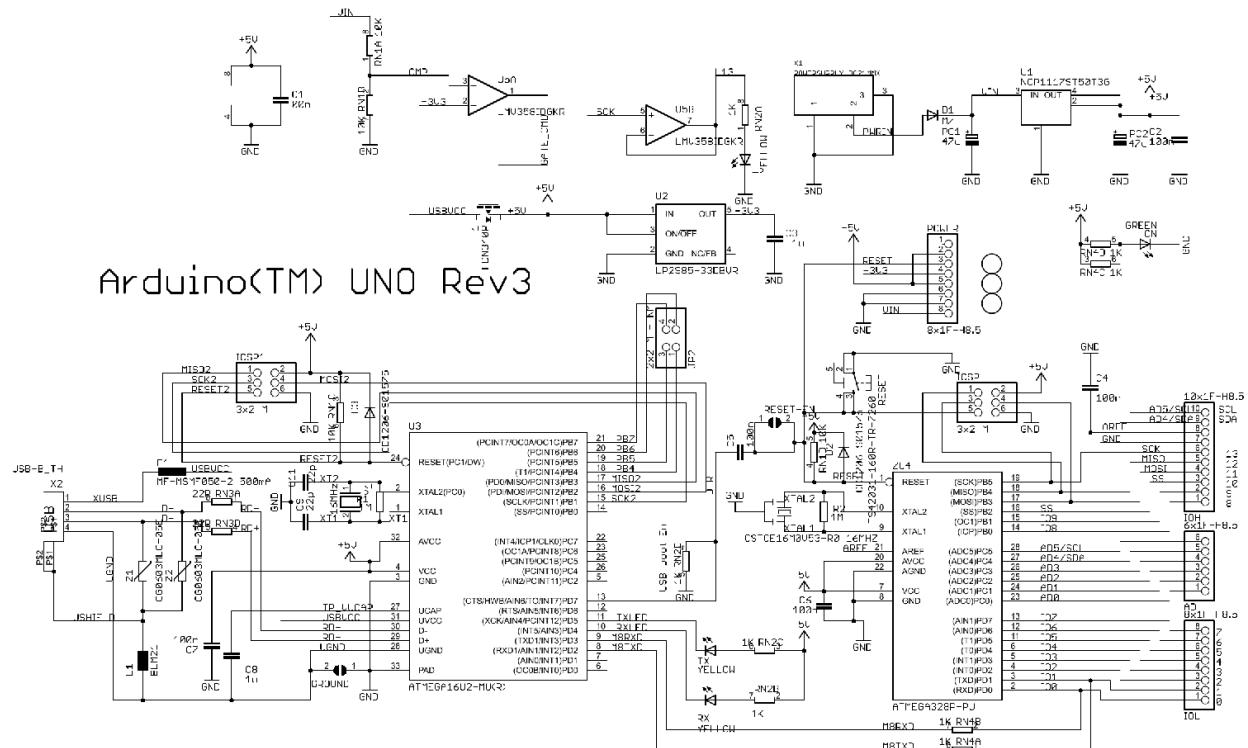


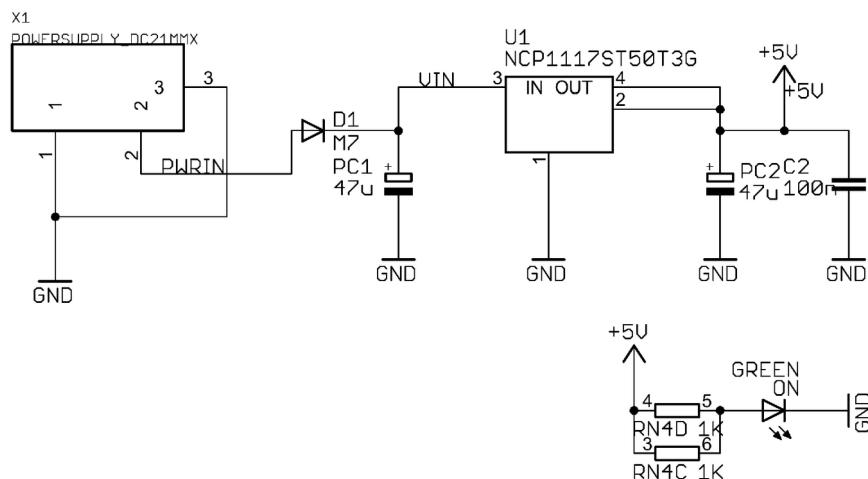
Figure 1: Arduino Uno Pinout

Source: <https://content.arduino.cc/assets/A000066-pinout.png>



**Figure 2:** Arduino Uno Rev.3 Schematic

Source: [https://www.arduino.cc/en/uploads/Main/Arduino\\_Uino\\_Rev3-schematic.pdf](https://www.arduino.cc/en/uploads/Main/Arduino_Uino_Rev3-schematic.pdf)



**Figure 3:** Linear Voltage Regulator Circuit (5V) + Power-on LED on Arduino Uno

```

#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/delay.h>

#define LED_PIN PB5

volatile uint8_t wdt_wakeup = 0;

ISR(WDT_vect) { // ISR for WDT
    wdt_wakeup = 1; // Set wake-up flag
}

static void led_init(void) {
    DDRB |= (1 << LED_PIN); // PB5 output
    PORTB &= ~(1 << LED_PIN); // LED OFF
}

static void wdt_init(void) {
    cli();
    // Clear WDRF flag
    MCUSR &= ~(1 << WDRF);
    // Enable configuration changes
    WDTCSR |= (1 << WDCE) | (1 << WDE);
    // Set timeout to ~4s and enable interrupt only
    WDTCSR = (1 << WDIE) // Watchdog interrupt enable
    WDTCSR |= (1 << WDP3); // Set WDT prescaler
    sei();
}

int main(void) {
    led_init();
    wdt_init();
    // Select lowest power sleep mode
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sei();
    while (1) {
        sleep_enable(); // Enable sleep mode
        sleep_cpu(); // Enter sleep mode
        sleep_disable(); // Disable sleep mode after wakeup
        if (wdt_wakeup) { // Check wake-up flag
            wdt_wakeup = 0; // Clear wake-up flag
            PORTB |= (1 << LED_PIN);
            _delay_ms(1000);
            PORTB &= ~(1 << LED_PIN);
            _delay_ms(1000);
        }
    }
}

```

**Code Listing 1**

## Lab 2: Sending characters as fast as possible

### Objectives

- Write **bare-metal C code** to configure the **USART0** to operate in **asynchronous mode** and transmit characters at the **highest possible baud rate**, using the **8N1** format..
- Use a **digital oscilloscope** or **USB logic analyzer** to capture and analyze the **TX** and **RX** signals.

### Lab Procedure

1. Create a new project using **Arduino IDE** and add the `main.c` code in a new tab.

1.1 Select **Arduino Uno** as the target board.

1.2 Write **AVR C code** in the `main.c` file that performs the following tasks.

- Configure the **USART0** to operate in asynchronous mode using the **8N1** format.
- Wait for incoming characters from the host computer and echo them back as fast as possible.
- Start with a baud rate of 115200 and then increase the baud rate step by step to:  
**460800, 921600, and 2000000**.

1.3 Connect the **Arduino Uno** to the host computer using a **USB** cable.

1.4 Build and upload the generated `.hex` file to the target board.

### Notes

- Enable **U2X** (Double Speed Mode) in the **UCSR0A** register when testing higher baud rates.
- Use the **USART** in interrupt-driven mode for improved responsiveness and reduced **CPU** blocking.
- The maximum achievable baud rate may be limited by the **USB-to-serial interface chip** rather than the **MCU** itself.

2. Analyze the **TX/RX** signals.

2.1 Use a **digital oscilloscope** or **USB logic analyzer** to capture and analyze the **TX** and **RX** signals.

2.2 Capture the waveforms and if possible apply a **UART protocol analyzer** to decode the signals.

2.3 Open Serial utility software such as **Putty, Tera Term** or **Arduino Serial Monitor** and send multiple characters at once.

# Lab 3: Converting a Triangular-Wave Signal to a PWM Signal Using the Analog Comparator

## Objectives

- Write bare-metal C code to use the **internal analog comparator** of the ATmega328P to convert a triangular input signal with a DC offset into a PWM output signal.
- Use the internal analog comparator in **interrupt mode**.

## Lab Procedure

1. Create a new project using **Arduino IDE** and add the `main.c` code in a new tab.

1.1 Select **Arduino Uno** as the target board.

1.2 Connect the **Arduino Uno** to the host computer using a **USB** cable.

1.3 Write bare-metal **C code** with the following functions.

- Configure the internal analog comparator to compare two analog inputs, using the **AIN0 / PD6** and **AIN1 / PD7** pins as inputs.
- Enable the **interrupt mode** of the analog comparator. The corresponding **ISR** is used to generate a **PWM** signal.
- The **PWM** duty cycle is adjustable using a voltage divider implemented with a potentiometer.

1.4 Build and upload the generated .hex file to the target board.

2. Prepare the input signals to the **Arduino Uno** board.

2.1 Create a **voltage divider** using a **potentiometer (10kΩ)** powered from **5V**, and connect its output to the **AIN0 / PD6** pin.

2.2 Create a triangular signal using a function generator and connect it to the **AIN1 / PD7** pin:

- Waveform type: **Triangular**
- Amplitude: **2Vpp**
- DC offset: **2V**
- Frequency: **1kHz and 10kHz**

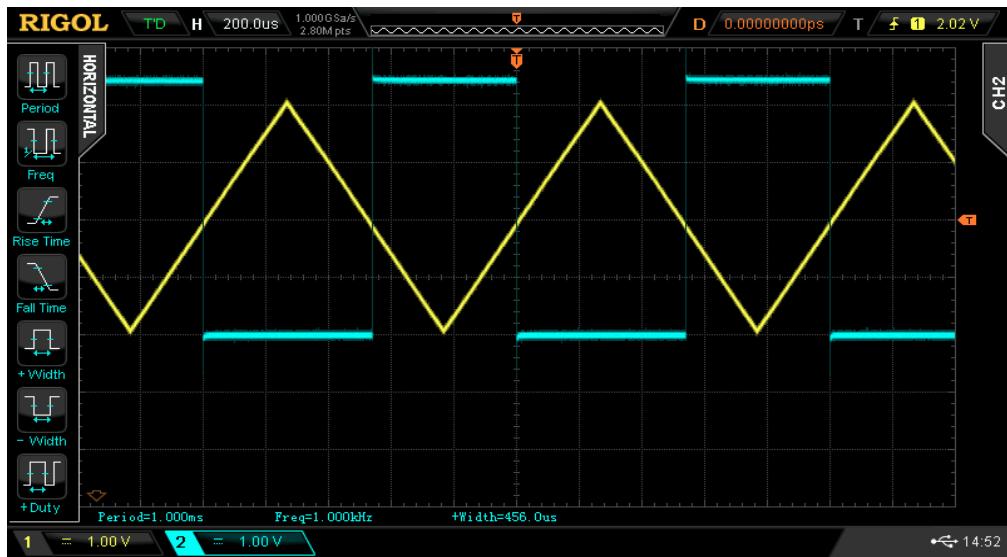
3. Use a digital oscilloscope to capture and analyze the signals.

3.1 Use the **CH1** probe to measure the input signal (the triangular signal).

3.2 Use the **CH2** probe to measure the output signal (the PWM signal).

## Notes

-  Ensure all input voltages remain within the 0–5V range.
- The PWM frequency is determined by the input triangular waveform frequency.



**Figure 4:** Signal waveform capture using a digital oscilloscope

(CH1: triangular input signal, CH2: PWM output signal)

# Lab 4: Using Analog Inputs to Control the Brightness of a WS2812B LED.

## Objectives

- Write bare-metal C code to configure the **ADC in single-conversion mode**.
- Use the **ADC** to read an analog input voltage and map the **ADC value** to the brightness level of a single-pixel **WS2812B RGB LED**.

## Lab Procedure

1. Create a new project using **Arduino IDE** and add the `main.c` code in a new tab.

1.1 Select **Arduino Uno** as the target board.

1.2 Connect the **Arduino Uno** to the host computer using a USB cable.

1.3 Write bare-metal **C code** with the following functions.

- Implement a C function that uses **inline AVR assembly** with cycle-accurate timing to send **24-bit RGB** data to the **WS2812B** module.
- Read the **A0** analog pin using the **ADC**, and convert the **10-bit ADC** value to an **8-bit unsigned value** to be used as the brightness level.
- Change the color of the **RGB LED** module by cycling through a list of **six different colors**, updating the color **every 1 second**.

1.4 Build and upload the generated .hex file to the target board.

2. Connect a **WS2812B** module to the **Arduino Uno** board

2.1 Use the onboard 5 V supply as the DC power source for the module.

2.2 Check that all module pins are connected correctly, paying particular attention to the power, ground, and data connections.

3. Use a **digital oscilloscope** or a **USB logic analyzer** to capture and analyze the I/O signals.

3.1 Use the **CH1** probe to measure the output signal sent to the **WS2812B** module.

3.2 Use the **CH2** probe to measure the input analog signal (voltage level).

3.3 Observe the timing characteristics of the **WS2812B** data signal and verify compliance with the datasheet.

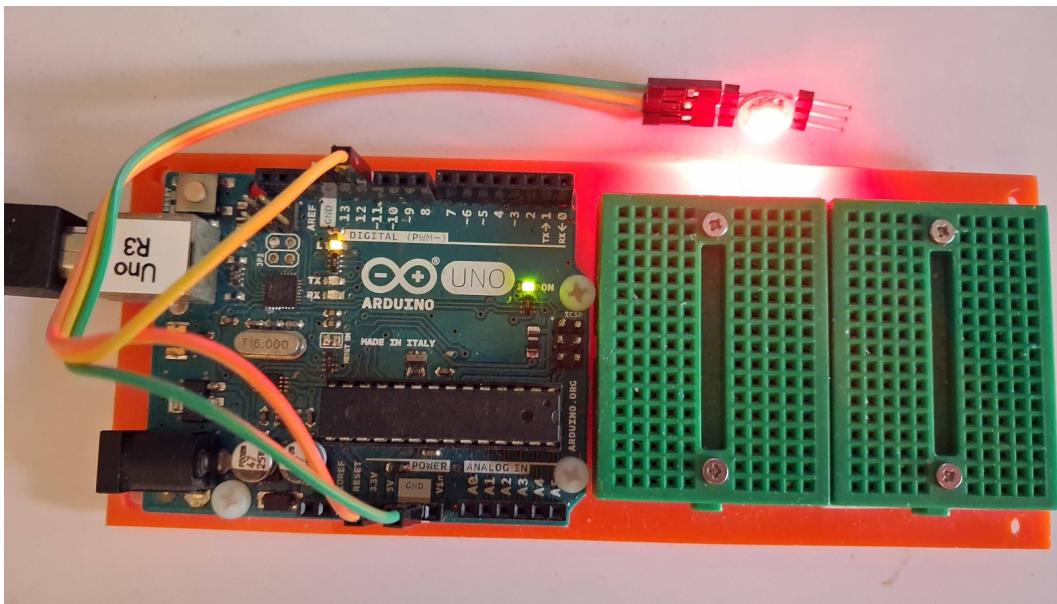
3. Revise the code to use an active-low push button to manually change the **RGB LED color** when the button is pressed.

3.1 Draw a schematic diagram of the complete system.

3.2 Test the revised code using real hardware.

3.3 Demonstrate the working system to the lab instructor.

4. Revise the code to read **three analog input channels** (for example, **A0**, **A1**, and **A2**). The corresponding **ADC values** should be reduced to **8-bit values** and used to independently set the brightness levels of the **red, green, and blue (R, G, B) color components**, respectively.



**Figure 5:** Arduino Uno + a single-pixel WS2812B RGB module