

# Lab Sheet for Week 7

Lab Instructor: RSP

## Lab 1: 4-Bit LED Control

### Objectives

- Use the **Wokwi Simulator for AVR** to edit and simulate AVR assembly code.
- Use the built-in **virtual 8-channel logic analyzer** to analyze and visualize the waveform of the selected I/O signals.

### Lab Procedure

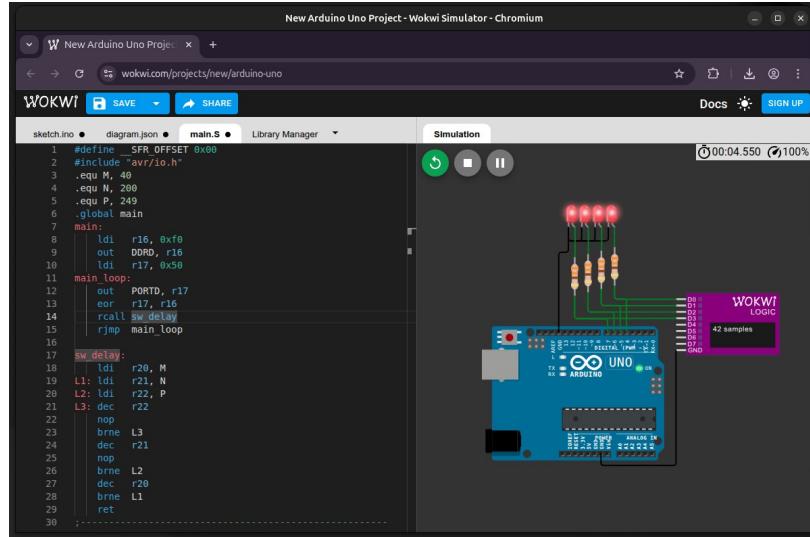
In this lab, the provided AVR assembly code (in **Code Listing 1**) shows how to blink four LEDs which are connected to the **PD4**, **PD5**, **PD6** and **PD7** of the **ATmega328P**.

1. Create a new project in **Wokwi Simulator** (<https://wokwi.com/projects/new/arduino-uno>) as shown in **Figure 1**.
  - 1.1 Select the **Arduino Uno** as the target board.
  - 1.2 In the tab “**diagram.json**”, replace the default contents with the JSON provided in **Code Listing 2**.
  - 1.3 Click on the “**Library Manager > New file...**”, add a new source-code file named “**main.S**” and insert the provided AVR assembly code.
2. Simulate the code and analyze the waveform file (named “**wokwi-logic.vcd**”) automatically generated by the virtual logic analyzer.
  - 2.1 Use a waveform visualization tool such as **surfer** or **GTKwave** to visualize the waveform.
  - 2.2 Determine the period of the output signals for each LED.
3. Rewrite the **AVR assembly code** for the same circuit diagram so that the four LEDs display a **4-bit Gray code sequence**, with a delay between two consecutive codes.
  - 3.1 Verify your code using the Wokwi simulator.
  - 3.2 Capture the resulting waveform for inclusion in the lab report.
4. Build the **.hex** file from the AVR assembly source code and test the firmware file.
  - 4.1 Install the **avr-gcc toolchain** and **avrdude** for your OS.
  - 4.2 Use the following commands to compile and generate the output file (**main.S → main.hex**).

```
avr-gcc -mmcu=atmega328p -o main.elf main.S
avr-objcopy -O ihex -R .eeprom main.elf main.hex
```
  - 4.3 Use **avrdude** to upload the firmware file (.hex) to the Arduino board.
  - 4.4 Use a digital oscilloscope or a logic analyzer to analyze the output signals.

4.5 Capture the resulting waveform for inclusion in the lab report.

4.6 During the lab session, demonstrate your work to the lab instructor.



**Figure 1:** Wokwi Simulator & Circuit Diagram for Lab 1

(Logic analyzer channels D0–D3 are connected to PD4–PD7.)

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"
.equ M, 40
.equ N, 200
.equ P, 249
.global main
main:
    ldi r16, 0xf0
    out DDRD, r16
    ldi r17, 0x50
main_loop:
    out PORTD, r17
    eor r17, r16
    rcall sw_delay
    rjmp main_loop
sw_delay:
    ldi r20, M
L1: ldi r21, N
L2: ldi r22, P
L3: dec r22
    nop
    brne L3
    dec r21
    nop
    brne L2
    dec r20
    brne L1
    ret
```

**Code Listing 1:** AVR assembly code of main.S for Lab 1

```
{
  "version": 1,
  "author": "Anonymous",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-uno", "id": "uno",
      "top": 50, "left": 25, "attrs": {} },
    { "type": "wokwi-resistor", "id": "r2",
      "top": -30, "left": 135, "rotate": 90,
      "attrs": { "value": "330" } },
    { "type": "wokwi-resistor", "id": "r3",
      "top": -35, "left": 155, "rotate": 90,
      "attrs": { "value": "330" } },
    { "type": "wokwi-resistor", "id": "r4",
      "top": -40, "left": 175, "rotate": 90,
      "attrs": { "value": "330" } },
    { "type": "wokwi-resistor", "id": "r5",
      "top": -45, "left": 195, "rotate": 90,
      "attrs": { "value": "330" } },
    { "type": "wokwi-led", "id": "led1",
      "top": -140, "left": 200, "attrs": { "color": "red" } },
    { "type": "wokwi-led", "id": "led2",
      "top": -140, "left": 180, "attrs": { "color": "red" } },
    { "type": "wokwi-led", "id": "led3",
      "top": -140, "left": 160, "attrs": { "color": "red" } },
    { "type": "wokwi-led", "id": "led4",
      "top": -140, "left": 140, "attrs": { "color": "red" } },
    { "type": "wokwi-logic-analyzer", "id": "logic1",
      "top": 0.35, "left": 316.8, "attrs": {} }
  ],
  "connections": [
    [ "uno:GND.1", "led4:C", "black", [ "v-20", "h0", "v-100", "h30" ] ],
    [ "led3:C", "led4:C", "black", [ "v20", "h-30" ] ],
    [ "led2:C", "led4:C", "black", [ "v20", "h-60" ] ],
    [ "led1:C", "led4:C", "black", [ "v20", "h-90" ] ],
    [ "uno:7", "r2:2", "green", [ "v-20", "h0" ] ],
    [ "uno:6", "r3:2", "green", [ "v-30", "h-5" ] ],
    [ "uno:5", "r4:2", "green", [ "v-40", "h-5" ] ],
    [ "uno:4", "r5:2", "green", [ "v-50", "h-5" ] ],
    [ "r5:1", "led1:A", "green", [ "v0" ] ],
    [ "r4:1", "led2:A", "green", [ "h0" ] ],
    [ "r3:1", "led3:A", "green", [ "h0" ] ],
    [ "r2:1", "led4:A", "green", [ "h0" ] ],
    [ "uno:GND.3", "logic1:GND", "black", [ "v17.7", "h112.8" ] ],
    [ "logic1:D0", "r5:2", "green", [ "h0" ] ],
    [ "logic1:D1", "r4:2", "green", [ "h0" ] ],
    [ "logic1:D2", "r3:2", "green", [ "h0" ] ],
    [ "logic1:D3", "r2:2", "green", [ "h0" ] ]
  ],
  "dependencies": {}
}
```

**Code Listing 2:** JSON code for the circuit diagram (diagram.json)

## Post-Experiment Questions

1. Analyze the assembly code in **Code Listing 1** and determine the number of clock cycles between two consecutive toggle events. Use the parameters **M**, **N**, and **P** in your calculation.
2. What are the period and frequency of the output signal?
3. Which values of **M**, **N**, and **P** should be used in the code to obtain an output frequency of **10 Hz**?

## Windows Batch Script

The following script (`build_upload.bat`) is provided as an example for compiling an AVR assembly source file and uploading the generated `.hex` file to an Arduino board.

Windows Command Prompt (cmd.exe) is required, and the script requires two command-line arguments:

```
build_upload.bat <COM_PORT> <HEX_FILE>
```

where:

- `<COM_PORT>` is the serial port assigned to the Arduino board (e.g., COM3)
- `<HEX_FILE>` is the name of the HEX file to be uploaded (e.g., `main.hex`)

```
@echo off

set "ARDUINO_TOOLS_PATH=C:\Users\%USERNAME%\AppData\Local"
set "ARDUINO_TOOLS_PATH=%ARDUINO_TOOLS_PATH%\Arduino15\packages\arduino\tools"
set "AVR_GCC_BIN=%ARDUINO_TOOLS_PATH%\avr-gcc\7.3.0-atmel3.6.1-arduino7\bin"
set "AVRDUE=%ARDUINO_TOOLS_PATH%\avrdude\6.3.0-arduino18"

del /q main.elf main.hex

"%AVR_GCC_BIN%\avr-gcc.exe" ^
-mmcu=atmega328p -o main.elf main.S || exit /b %ERRORLEVEL%

"%AVR_GCC_BIN%\avr-objcopy.exe" ^
-O ihex -R .eprom main.elf main.hex || exit /b %ERRORLEVEL%

"%AVRDUE%\bin\avrdude.exe" ^
-C "%AVRDUE%\etc\avrdude.conf" ^
-v -patmega328p -carduino -P%1 -b115200 -D -Uflash:w:%2:i
```

## Lab 2: BCD 7-segment display

### Objectives

- Use the **Wokwi Simulator for AVR** to edit and simulate AVR assembly code.
- Use the built-in **virtual 8-channel logic analyzer** to analyze and visualize the waveform of the selected I/O signals.

### Lab Procedure

In this lab, the provided AVR assembly code (in **Code Listing 3**) targeting at the ATmega328P on the Arduino Uno board shows how to read a 4-bit digital input from a DIP switch and display the corresponding number on a 7-segment display.

1. Create a new project in **Wokwi Simulator** (<https://wokwi.com/projects/new/arduino-uno>) as shown in **Figure 2**.
  - 1.1 Select the **Arduino Uno** as the target board.
  - 1.2 In the tab “**diagram.json**”, replace the default contents with the JSON provided in **Code Listing 4**.
  - 1.3 Click on the “**Library Manager > New file...**”, add a new source-code file named “**main.S**” and insert the provided AVR assembly code.
2. Simulate the code using the Wokwi Simulator.
3. Modify the circuit diagram to use a **common-cathode 7-segment display**, and revise the assembly code accordingly. Note that the provided Wokwi diagram already uses a common-anode connection.
  - 3.1 Verify the modified code using the Wokwi Simulator.
  - 3.2 Change the DIP switch positions and observe the corresponding changes on the 7-segment display.
  - 3.3 Capture screenshots of the simulation for inclusion in the lab report.
4. **Rewrite the AVR assembly code** and use the same circuit diagram (the DIP switches are left unused), to implement the following functionality:
  - The ATmega328P **counts down from 9 to 0**, with a **period of approximately 0.1 seconds** between each count.
  - When the **push button is pressed**, reset the count value to 9.
  - When the **counter reaches 0**, make the **7-segment display blink**.
5. Use the Wokwi Simulator to test and verify your modified code.
  - 5.1 Capture screenshots of the simulation for inclusion in the lab report.
  - 5.2 During the lab session, demonstrate your complete work to the lab instructor.

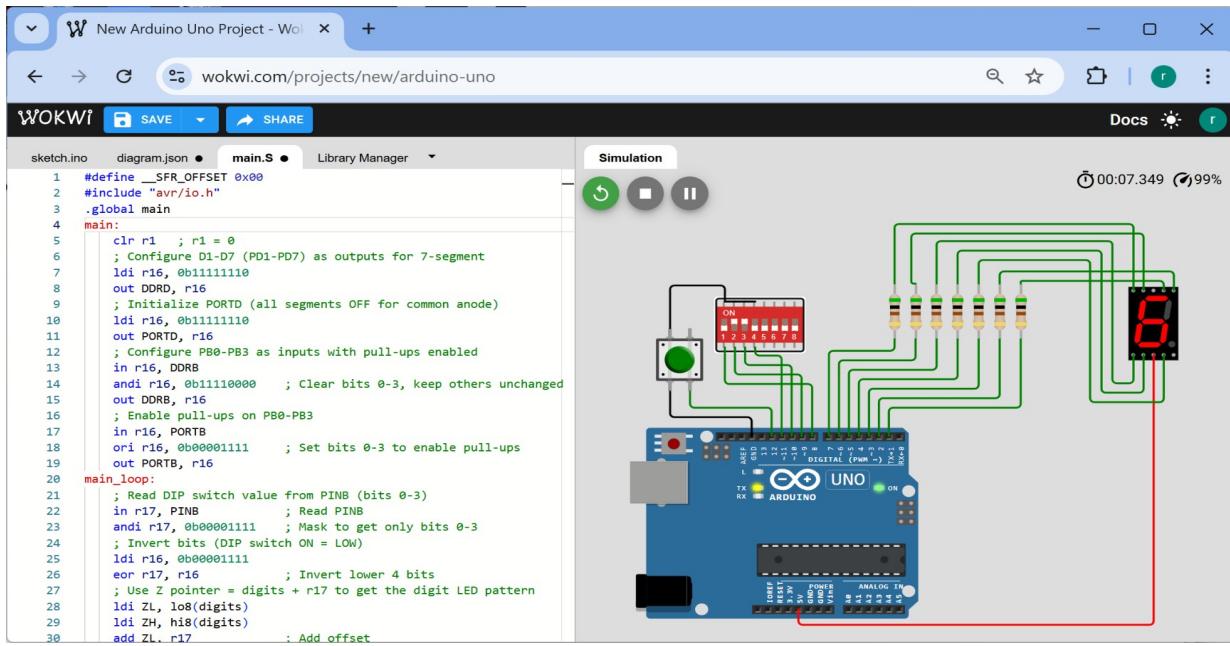


Figure 2: Wokwi Simulator & Circuit Diagram for Lab 2

```

#define __SFR_OFFSET 0x00
#include "avr/io.h"
.global main
main:
    clr r1 ; r1 = 0
    ; Configure D1-D7 (PD1-PD7) as outputs for 7-segment
    ldi r16, 0b11111110
    out DDRD, r16
    ; Initialize PORTD (all segments OFF for common anode)
    ldi r16, 0b11111110
    out PORTD, r16
    ; Configure PB0-PB3 as inputs with pull-ups enabled
    in r16, DDRB
    andi r16, 0b11110000 ; Clear bits 0-3, keep others unchanged
    out DDRB, r16
    ; Enable pull-ups on PB0-PB3
    in r16, PORTB
    ori r16, 0b00001111 ; Set bits 0-3 to enable pull-ups
    out PORTB, r16
main_loop:
    ; Read DIP switch value from PINB (bits 0-3)
    in r17, PINB ; Read PINB
    andi r17, 0b00001111 ; Mask to get only bits 0-3
    ; Invert bits (DIP switch ON = LOW)
    ldi r16, 0b00001111
    eor r17, r16 ; Invert lower 4 bits
    ; Use Z pointer = digits + r17 to get the digit LED pattern
    ldi ZL, lo8(digits)
    ldi ZH, hi8(digits)
    add ZL, r17 ; Add offset
    adc ZH, r1 ; r1 is always 0 (zero register for avr-gcc)
    ; Load digit pattern from program memory
    lpm r16, Z ; Load pattern into r16
    ; Invert for common anode and mask to D1-D7
    com r16 ; Invert all bits
    andi r16, 0xFE ; Mask to keep only bits 1-7
    out PORTD, r16 ; Output to PORTD
    rcall sw_delay ; call subroutine
    rjmp main_loop

sw_delay:
    ldi r24, lo8(40000) ; Use r25:r24 (16-bit) register pair
    ldi r25, hi8(40000)
delay_loop:
    sbiw r24, 1 ; Subtract 1 from r25:r24 [2 cycles]
    brne delay_loop ; Branch if not zero [2/1 cycles]
    ret ; [4 cycles] return from subroutine

; Digit LED patterns table (stored in program memory)
digits:
    .byte 0b01111110 ; 0
    .byte 0b00001100 ; 1
    .byte 0b10110110 ; 2
    .byte 0b10011110 ; 3
    .byte 0b11001100 ; 4
    .byte 0b11011010 ; 5
    .byte 0b11111010 ; 6
    .byte 0b00001110 ; 7
    .byte 0b11111110 ; 8
    .byte 0b11011110 ; 9
    .byte 0b11101110 ; A
    .byte 0b11111000 ; b
    .byte 0b01110010 ; C
    .byte 0b10111100 ; d
    .byte 0b11110010 ; E
    .byte 0b11100010 ; F

```

**Code Listing 3:** AVR assembly code of main.S for Lab 2

```
{
  "version": 1,
  "author": "Anonymous",
  "editor": "wokwi",
  "parts": [
    {
      "type": "wokwi-arduino-uno", "id": "uno", "top": 96, "left": 38, "attrs": {} },
    {
      "type": "wokwi-7segment", "id": "sevseg1", "top": -52.62, "left": 512, "attrs": {} },
    {
      "type": "wokwi-resistor", "id": "r1",
      "top": -32, "left": 380, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r2",
      "top": -32, "left": 360, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r3",
      "top": -32, "left": 340, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r4",
      "top": -32, "left": 320, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r5",
      "top": -32, "left": 300, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r6",
      "top": -32, "left": 280, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-resistor", "id": "r7",
      "top": -32, "left": 260, "rotate": 90,
      "attrs": { "value": "500" }
    },
    {
      "type": "wokwi-pushbutton", "id": "btn1",
      "top": 0, "left": 50, "rotate": 90,
      "attrs": { "color": "green", "xray": "1" }
    },
    {
      "type": "wokwi-dip-switch-8", "id": "sw1", "top": -40, "left": 120, "attrs": {} }
  ],
  "connections": [
    [ "uno:1", "r1:2", "green", [ "v-30", "h130" ] ],
    [ "uno:2", "r2:2", "green", [ "v-40", "h120" ] ],
    [ "uno:3", "r3:2", "green", [ "v-50", "h110" ] ],
    [ "uno:4", "r4:2", "green", [ "v-60", "h100" ] ],
    [ "uno:5", "r5:2", "green", [ "v-70", "h90" ] ],
    [ "uno:6", "r6:2", "green", [ "v-80", "h80" ] ],
    [ "uno:7", "r7:2", "green", [ "v-90", "h70" ] ],
    [ "sevseg1:COM.1", "uno:5V", "red", [ "h0", "v285", "h-180" ] ],
    [ "sevseg1:A", "r1:1", "green", [ "v-10", "h-130" ] ],
    [ "sevseg1:B", "r2:1", "green", [ "v-20", "h-160" ] ],
    [ "sevseg1:C", "r3:1", "green", [ "v50", "h-70", "v-150", "h-100" ] ],
    [ "sevseg1:D", "r4:1", "green", [ "v40", "h-40", "v-150", "h-130" ] ],
    [ "sevseg1:E", "r5:1", "green", [ "v30", "h-20", "v-150", "h-160" ] ],
    [ "sevseg1:F", "r6:1", "green", [ "v-60", "h-220.8", "v52.84" ] ],
    [ "sevseg1:G", "r7:1", "green", [ "v-70", "h-220" ] ],
    [ "uno:GND.1", "btn1:2.r", "black", [ "v-20", "h-75" ] ],
    [ "uno:8", "sw1:1a", "green", [ "v-55", "h-70" ] ],
    [ "uno:9", "sw1:2a", "green", [ "v-65", "h-50" ] ],
    [ "uno:10", "sw1:3a", "green", [ "v-75", "h-40" ] ],
    [ "uno:11", "sw1:4a", "green", [ "v-85", "h-30" ] ],
    [ "uno:12", "btn1:1.r", "green", [ "v-30", "h-130" ] ],
    [ "btn1:2.l", "sw1:1b", "black", [ "v-45", "h60" ] ],
    [ "sw1:1b", "sw1:2b", "black", [ "v0" ] ],
    [ "sw1:2b", "sw1:3b", "black", [ "v0" ] ],
    [ "sw1:3b", "sw1:4b", "black", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

**Code Listing 4:** JSON code for the circuit diagram (diagram.json)

## Lab 3: WS2812B RGB LED Control

### Objectives

- Use the **Wokwi Simulator for AVR** to edit and simulate AVR assembly code.
- Use the built-in **virtual 8-channel logic analyzer** to analyze and visualize the waveform of the selected I/O signals.

### Lab Procedure

In this lab, the provided AVR assembly code (in **Code Listing 5**) demonstrates how to set the **24-bit RGB color** of a single-pixel **WS2812B-based RGB LED module**.

1. Create a new project in **Wokwi Simulator** (<https://wokwi.com/projects/new/arduino-uno>) as shown in **Figure 3**.
  - 1.1 Select the **Arduino Uno** as the target board.
  - 1.2 In the tab “**diagram.json**”, replace the default contents with the JSON code provided in **Code Listing 6**.
  - 1.3 Click on the “**Library Manager > New file...**”, add a new source-code file named “**main.S**” and insert the provided AVR assembly code.
2. Run the simulation and analyze the waveform file (named “**wokwi-logic.vcd**”) automatically generated by the virtual logic analyzer.
  - 2.1 Use a waveform visualization tool to visualize the waveform.
  - 2.2 Determine the **bit timing parameters (high and low pulse widths)** for **Bit=0** and **Bit=1** in the output waveform, and compare the results with the values specified in the **WS2812B datasheet**.
3. **Rewrite the AVR assembly code** to implement the following functionality:
  - Test color values are stored in program memory, with each color value occupying **three bytes (R, G, B)**.
  - A total of **six different color values** are defined.
  - The color values are read sequentially from program memory and applied to the RGB LED module one by one.
4. Verify your modified code using the Wokwi simulator.
5. Build the **.hex** file from the AVR assembly source code and test the firmware file on real hardware.
  - 5.1 Use a **digital oscilloscope** or a **logic analyzer** to analyze the output signal.
  - 5.2 Capture the resulting waveform for inclusion in the lab report.
  - 5.3 During the lab session, **demonstrate your complete work to the lab instructor**.

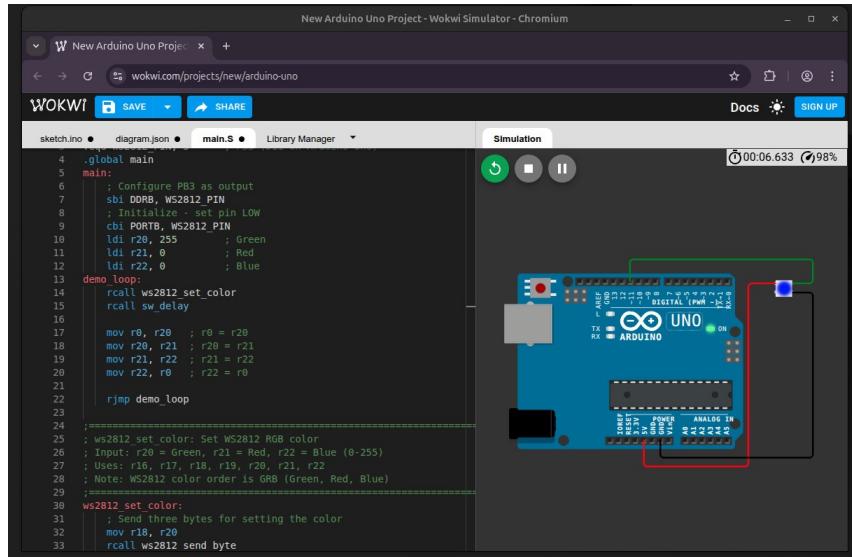


Figure 3: Wokwi Simulator & Circuit Diagram for Lab 3

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"
.equ WS2812_PIN, 3          ; PB3 (D11 on Arduino Uno)
.global main

main:
    clr r1 ; r1 = 0
    ; Configure PB3 as output
    sbi DDRB, WS2812_PIN
    ; Set data pin LOW
    cbi PORTB, WS2812_PIN
    ldi r20, 255      ; Green
    ldi r21, 0        ; Red
    ldi r22, 0        ; Blue

demo_loop:
    rcall ws2812_set_color
    rcall sw_delay

    mov r0, r20    ; r0 = r20
    mov r20, r21    ; r20 = r21
    mov r21, r22    ; r21 = r22
    mov r22, r0    ; r22 = r0

    rjmp demo_loop

;=====
; ws2812_set_color: Set WS2812 RGB color
; Input: r20 = Green, r21 = Red, r22 = Blue (0-255)
; Uses: r16, r17, r18, r19, r20, r21, r22
; Note: WS2812 color order is GRB (Green, Red, Blue)

ws2812_set_color:
    ; Send three bytes for setting the color
    mov r18, r20
    rcall ws2812_send_byte
    mov r18, r21
    rcall ws2812_send_byte
    mov r18, r22
    rcall ws2812_send_byte
    ret

; Code continues on the next page...
```

```

;=====
; Uses: r24:r25, r26
sw_delay:
    ldi r26, 125          ; Outer loop (125 iterations)
delay_outer:
    ldi r24, lo8(16000)  ; Inner loop (16000 iterations)
    ldi r25, hi8(16000)
delay_inner:
    sbiw r24, 1           ; 2 cycles
    nop                  ; 1 cycle
    brne delay_inner     ; 2 cycles if taken
    dec r26               ; 1 cycle
    nop                  ; 1 cycle
    brne delay_outer     ; 2 cycles if taken
    ret                  ; return from subroutine

;=====
; ws2812_send_byte: Send one byte to WS2812
; Input: r18 = byte to send (MSB first)
; Uses: r16, r17, r18, r19

ws2812_send_byte:
    ldi r19, 8              ; 8 bits to send
send_bit_loop:
    ; Check MSB of r18
    lsl r18                 ; Shift left, MSB goes to carry flag
    brcs send_one            ; If carry set, send '1'
send_zero:
    ; Send 0: HIGH for ~400ns (6 cycles), LOW for ~850ns (14 cycles)
    sbi PORTB, WS2812_PIN   ; 2 cycles - Set pin HIGH
    nop                      ; 1 cycle
    nop                      ; 1 cycle
    nop                      ; 1 cycle
    cbi PORTB, WS2812_PIN   ; 2 cycles - Set pin LOW
    nop                      ; 1 cycle
    rjmp bit_done             ; 2 cycles
send_one:
    ; Send 1: HIGH for ~800ns (13 cycles), LOW for ~450ns (7 cycles)
    sbi PORTB, WS2812_PIN   ; 2 cycles - Set pin HIGH
    nop                      ; 1 cycle
    cbi PORTB, WS2812_PIN   ; 2 cycles - Set pin LOW
    nop                      ; 1 cycle
    nop                      ; 1 cycle
    nop                      ; 1 cycle
bit_done:
    dec r19                  ; 1 cycle
    brne send_bit_loop       ; 2 cycles if taken, 1 if not
    ret

```

**Code Listing 5:** AVR assembly code of main.S for Lab 3

```
{
  "version": 1,
  "author": "Anonymous",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-uno", "id": "uno",
      "top": 70, "left": 70, "attrs": {} },
    { "type": "wokwi-neopixel", "id": "rgb1",
      "top": 78, "left": 380, "attrs": {} }
  ],
  "connections": [
    [ "uno:11", "rgb1:DIN", "green", [ "v-25", "h210", "v5" ] ],
    [ "uno:5V", "rgb1:VDD", "red", [ "v30", "h120", "v-210", "h10" ] ],
    [ "uno:GND.3", "rgb1:VSS", "black", [ "v20", "h175", "v-180" ] ]
  ],
  "dependencies": {}
}
```

**Code Listing 6:** JSON code for the circuit diagram (diagram.json)

### Note: Using Wokwi for VS Code

- The online Wokwi Simulator is free to use; however, users **may experience delays during the build process**, which can take several seconds depending on the current server load.
- To avoid this limitation, users are encouraged to use the **Wokwi for VS Code extension** together with the **Visual Studio Code (VS Code) IDE** (<https://code.visualstudio.com/>) installed on their local computer (Windows, Linux, or macOS). For more information, please consult the following online resources:
  - <https://marketplace.visualstudio.com/items?itemName=Wokwi.wokwi-vscode>
  - <https://docs.wokwi.com/vscode/getting-started>
- When using **Wokwi for VS Code**, users must install the **AVR-GCC toolchain** locally to compile the AVR assembly source code.
  - The toolchain can be downloaded from Microchip Technology's official website:  
<https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers>
- If the **Arduino IDE** is installed, the **AVR-GCC toolchain** is already included and can be used without installing additional software.
  - The toolchain is located inside the Arduino installation directory and can be accessed from the command line.
  - Users may need to add the Arduino toolchain path to the **system PATH environment variable** in order to invoke commands such as **avr-gcc**.
- The Wokwi simulator is not responsible for compiling the program source code; therefore, the source code must be compiled manually by the user to generate a .hex file. Wokwi for VS Code is then used to simulate the virtual circuit locally by loading the generated .hex file.
- In addition, the user must add the **wokwi.toml** file, which specifies the AVR binary file(s) used for simulation.
- **Figure 5** shows the Wokwi Simulator running in the VS Code IDE during the simulation process, and **Figure 6** shows the contents of the wokwi.toml file.
- This approach enables faster build times, offline simulation, and a development workflow closer to real embedded-system projects.

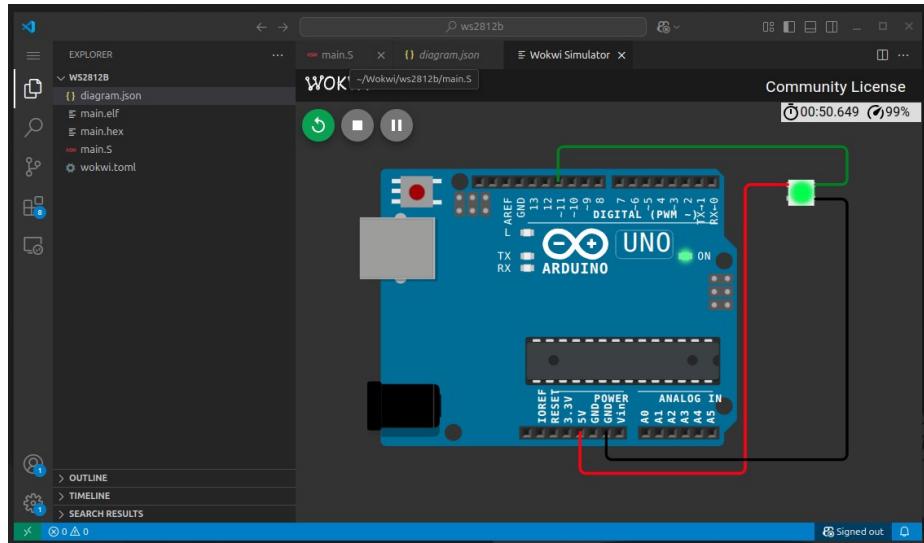


Figure 4: Wokwi for VS Code – Simulation Session

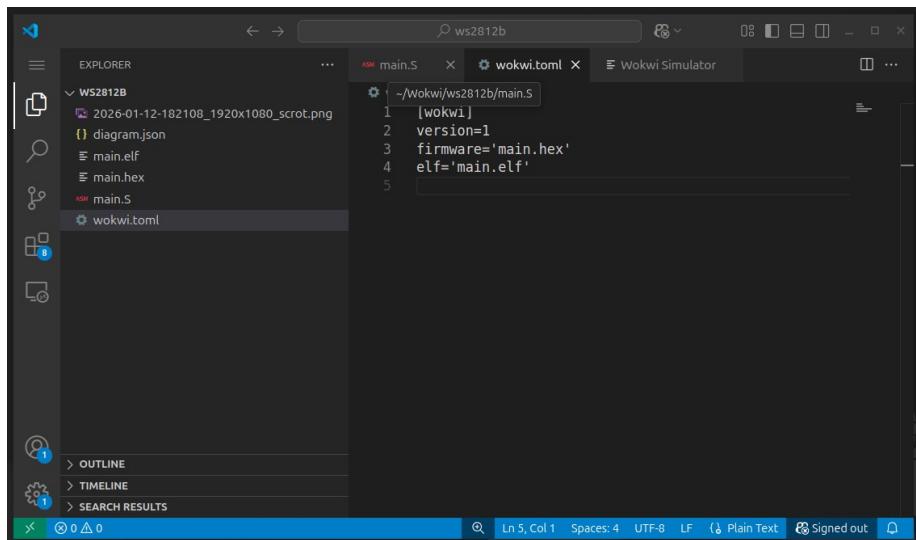


Figure 5: The contents of the wokwi.toml file