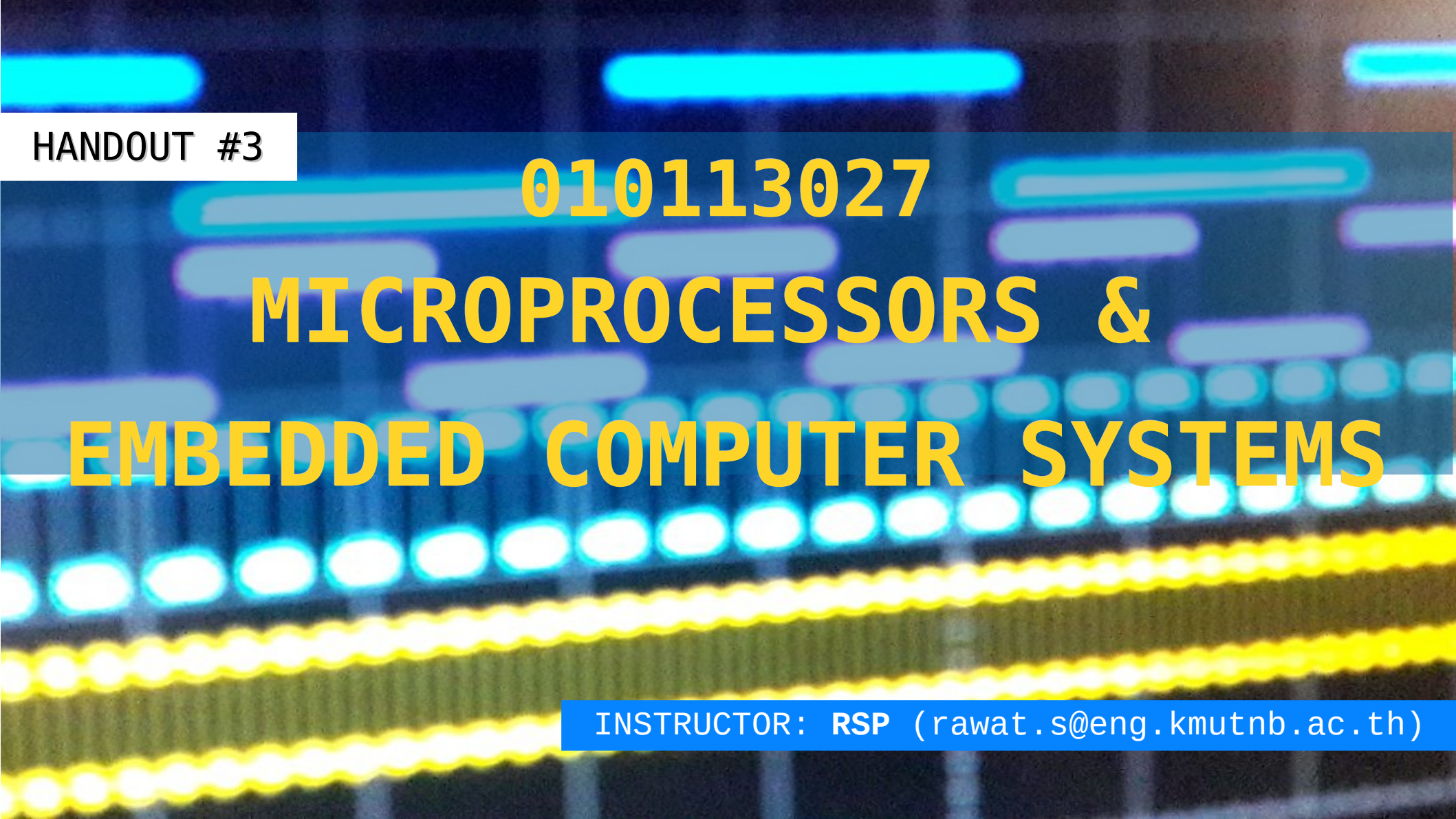


HANDOUT #3

010113027
MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

INSTRUCTOR: **RSP** (rawat.s@eng.kmutnb.ac.th)

Key Topics

- Examples of Practical (Hands-on) Learning Activities
- Arduino Board Exploration and Comparisons
- Software Exploration: Arduino IDE & Arduino API
- Low-Level (Bare-Metal) C Programming (ATmega328P)
- Virtual MCU / Arduino Programming using Wokwi Simulator
- Atmel Studio / Microchip Studio IDE: AVR C & Assembly
- Microchip MPLAB-X IDE + XC8 Compiler
- CLI Tools: AVR-GCC + AVRDUDE under Windows / Ubuntu
- Methods for Flashing Arduino Bootloader and Firmware File (.HEX)
- Using USBasp / Arduino-ISP / Arduino Board as AVR programmer

Practical Learning Activities

⚙ Arduino IDE & Arduino Core

- Install the **Arduino IDE 2.x**.
- Install, update and manage **Arduino Cores** using **Boards Manager**.
- Write and test **Arduino sketches** for **Arduino Uno / Nano** using the **Arduino API**.
- Learn and use **basic Arduino functions** such as pinMode(), digitalWrite(), digitalRead(), analogRead(), and functions for Serial, interrupts, delay functions, Wire (I2C), SPI, etc.

Practical Learning Activities

⚙ AVR-GCC Toolchain (CLI)

- Install the **AVR-GCC toolchain** on Windows 10/11 (using **Microchip AVR toolchain**) or Ubuntu.
- Learn **CLI tools** included in the AVR toolchain:
  - avr-gcc – compile C code
  - avr-objcopy – convert ELF to HEX
  - avr-size – check binary size
  - avr-objdump – disassemble
  - avr-ar – manage libraries
  - avr-as – assemble
  - make – build automation
- Compile example code into firmware (.hex or .elf).

CLI = Command Line Interface

Practical Learning Activities

⚙ Low-Level AVR Programming

- **Study bare-metal AVR C examples:**
  - Direct register access to I/O Ports (`DDRx`, `PORTx`, `PINx` registers)
  - Timers / Counters
  - External and pin-change interrupts
  - UART, SPI, I2C (TWI)
- **Study AVR assembly:**
  - AVR instruction set
  - AVR assembly examples

⚙ Uploading Firmware

- Upload firmware to Uno / Nano:
  - Software: avrdude
  - Hardware programmers: **USBasp** or **Arduino-as-ISP**
- Flash .hex files generated by GCC.

Practical Learning Activities

Professional AVR Development Tools

- Install and use **Microchip Studio (AVR)** or **MPLAB X IDE**.
- Create new projects for ATmega328P (Uno/Nano MCU).
- Compile and debug AVR projects.
- Program hardware using Arduino-as-ISP.

Arduino Bootloader

- Reflash or restore the Arduino Optiboot bootloader for Uno / Nano.
- Understand fuses and clock configuration:

  lfuse, hfuse, efuse
- Use avrdude or Microchip Studio to burn the bootloader.

AVR Virtual Prototyping

- Use online simulators like **Wokwi** to test AVR code without hardware.
- Simulate / debug the code using the AVR simulator and also virtual logic analyzer & waveforms

In-Class Exercise: Explore Arduino Boards

Objective:

- Students will **explore various Arduino boards** to identify their MCUs or MPUs, CPU architectures, on-chip peripherals, and other key characteristics.
- Students will work in small groups (3–4 per group). Each group will be assigned a list of Arduino boards to explore.
- For each board, students will research and record information about the on-board MCU / MPU.
- Note: Some Arduino boards may have more than one MCU / MPU.

In-Class Exercise: Explore Arduino Boards

On-board MCU / MPU Models

1) The main MCU / MPU:

- Number of CPU cores
- ISA / Architecture
- Clock speed
- On-chip memory (type and capacity)
- On-chip peripherals

2) The main MPU (if available):

- Number of CPU cores
- ISA / Architecture
- Clock speed

Additional Tasks:

- Identify if the board has single-core or multi-core MCU/MPU.
- Note differences between 8-bit, 16-bit, and 32-bit MCUs.
- Identify any special features (wireless, FPU, DSP, security features).

In-Class Exercise: Explore Arduino Boards

List 1

- Arduino Nano 33 BLE
- Arduino Portenta H7
- Arduino Uno R3
- Arduino GIGA R1 WiFi
- Arduino Due

List 2

- Arduino Portenta X8
- Arduino Uno R4 Minima
- Arduino Nano Every
- Arduino Nicla Vision
- Arduino MEGA 2560

List 3

- Arduino Uno Q
- Arduino Portenta C33
- Arduino Nano ESP32
- Arduino Nano 33 BLE
- Arduino Uno R3

List 4

- Arduino Nicla Sense ME
- Arduino MEGA 2560
- Arduino Nano Every
- Arduino Portenta H7
- Arduino Uno R4 WiFi

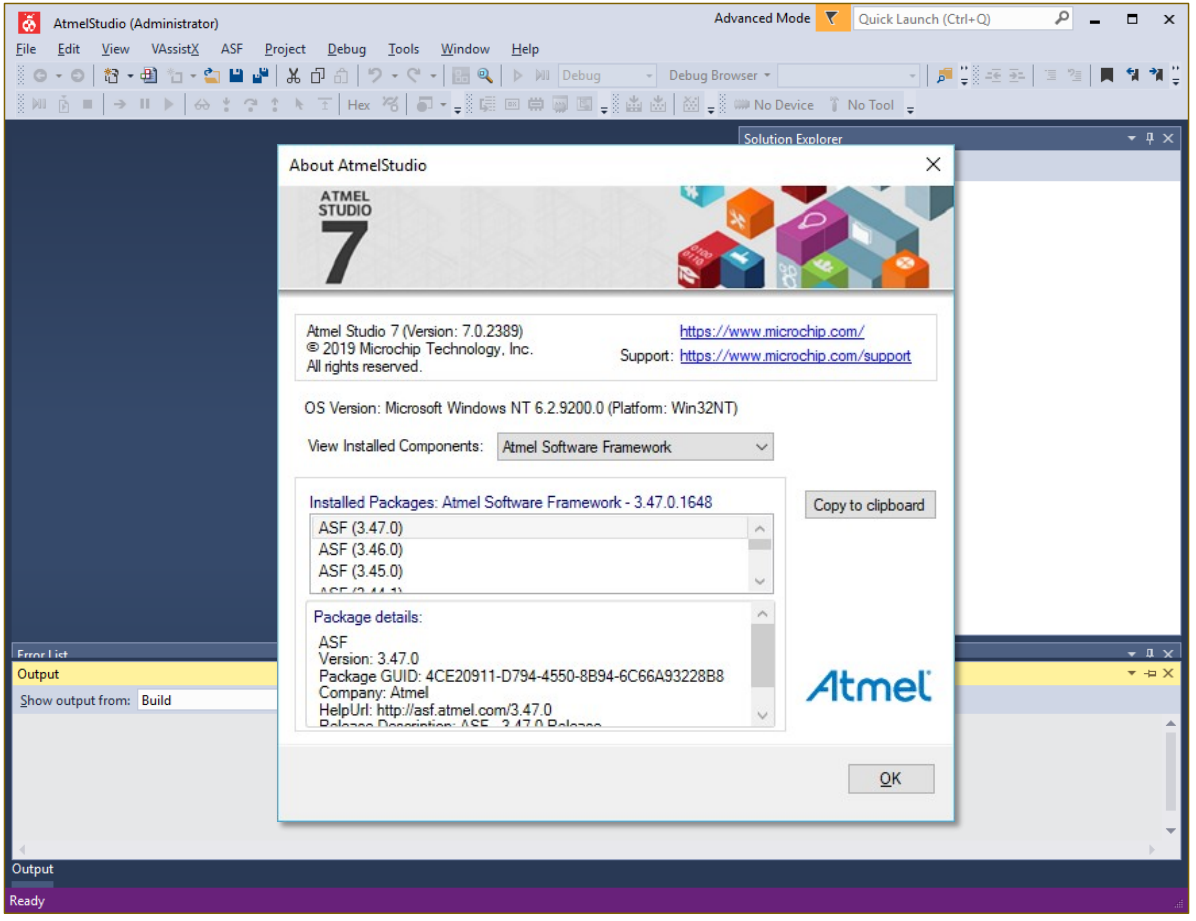
List 5

- Arduino Nano ESP32
- Arduino Uno Q
- Arduino Portenta X8
- Arduino GIGA R1 WiFi
- Arduino Portenta C33

List 6

- Arduino Nano RP2040 Connect
- Arduino Nano 33 BLE
- Arduino MEGA 2560
- Arduino Uno R3
- Arduino Nicla Vision

Legacy Software: Atmel Studio 7



- **Atmel Studio 7** (available for Windows only) was renamed **Microchip Studio for AVR and SAM Devices**.
- Microchip does not recommend using the Atmel Studio IDE anymore; instead, they recommend **MPLAB X IDE**.
- The latest version of Microchip Studio is **7.0.2594** (released on 20 June 2022).

- https://www.microchip.com/en-us/tools-resources/develop/microchip-studio#Downloads
- https://www.microchip.com/en-us/tools-resources/archives/avr-sam-mcus

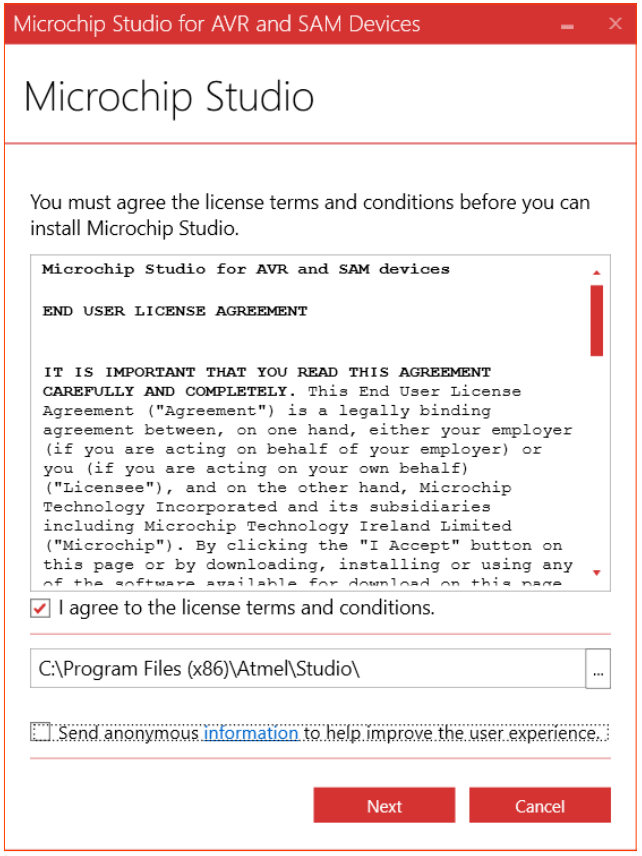
Microchip Studio (2022)

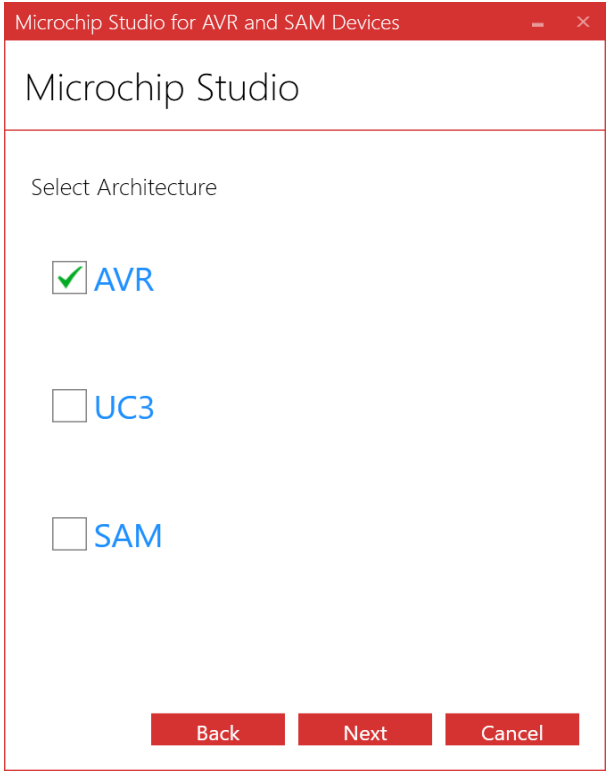
Microchip Studio for AVR (8-bit) and SAM (32-bit) Devices

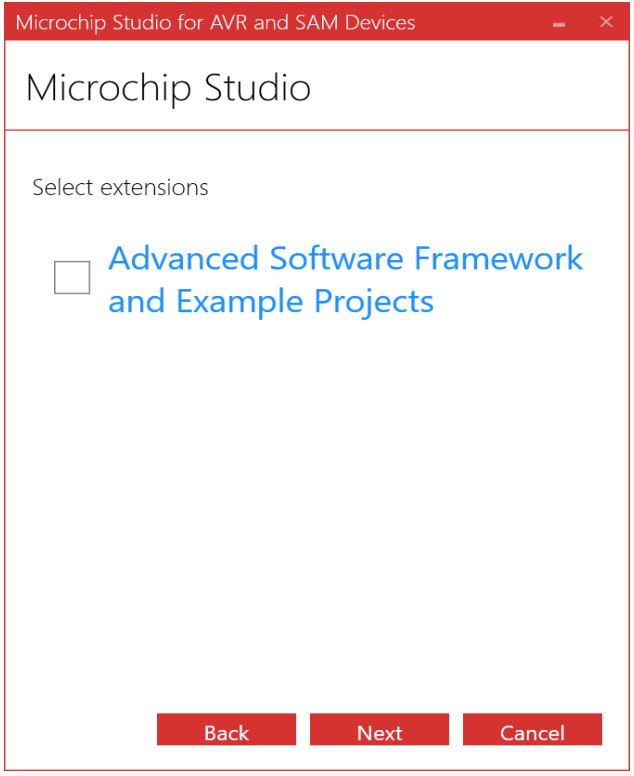
- **IDE (Integrated Development Environment)**
  - available for Windows platforms only
- **C/C++ compiler support:**
  - **AVR GCC Toolchain** (GCC 5.4.0, Binutils 2.26, avr-libc 2.0.0, gdb 7.8)
  - **MPLAB XC8 Compiler** with AVR devices support
  - **Arm GCC Toolchain** (32-bit Arm Cortex-M)
- **AVR Macro assembler**
- **Cycle correct simulator** with advanced debug functionality

Download: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio

Microchip Studio v7.0.2594 (2022)

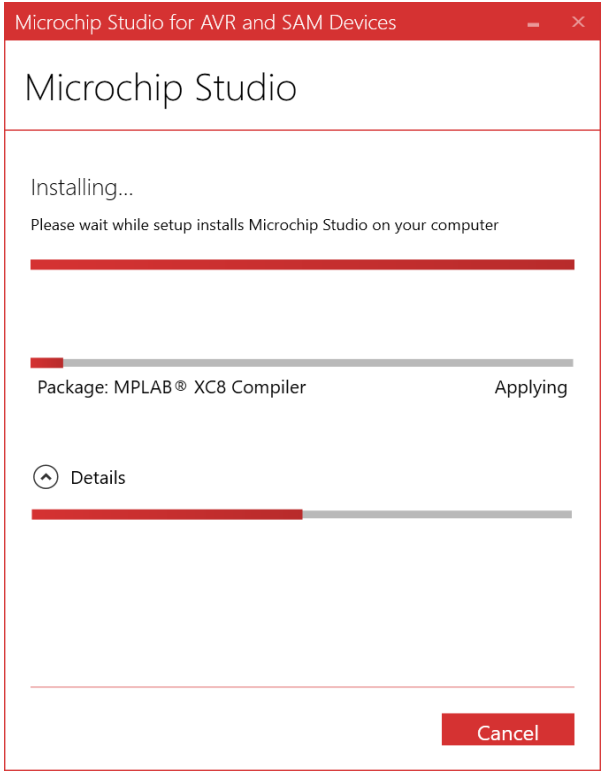


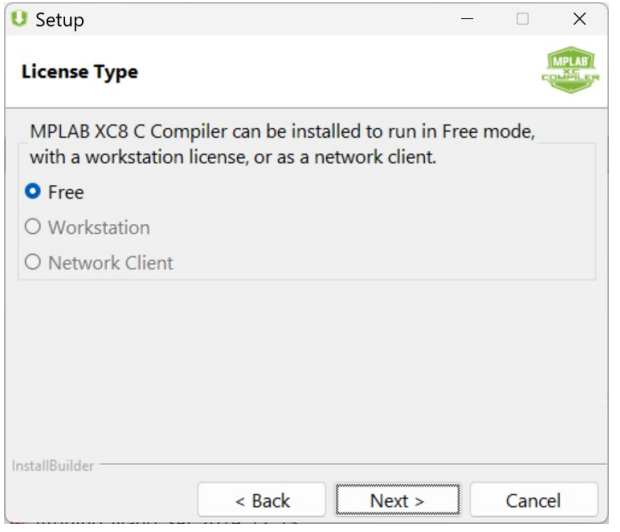




To reduce hard-disk usage during the Microchip Studio installation, install only **AVR support** and **skip UC3 and SAM**. We also don't need **ASF**, so it can be omitted as well.

Microchip Studio v7.0.2594 (2022)

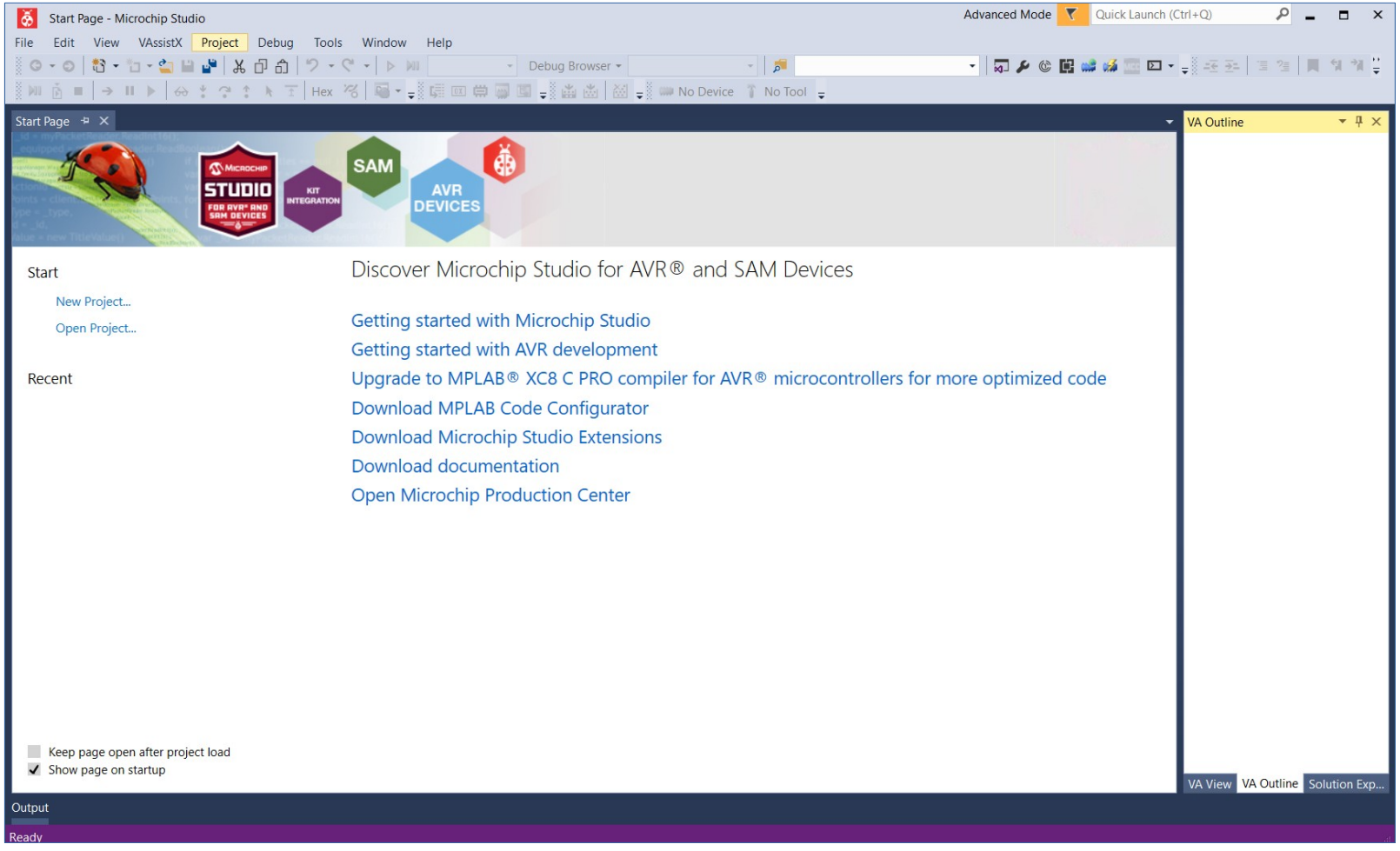




Be sure to install the **XC8 compiler** (included by default).

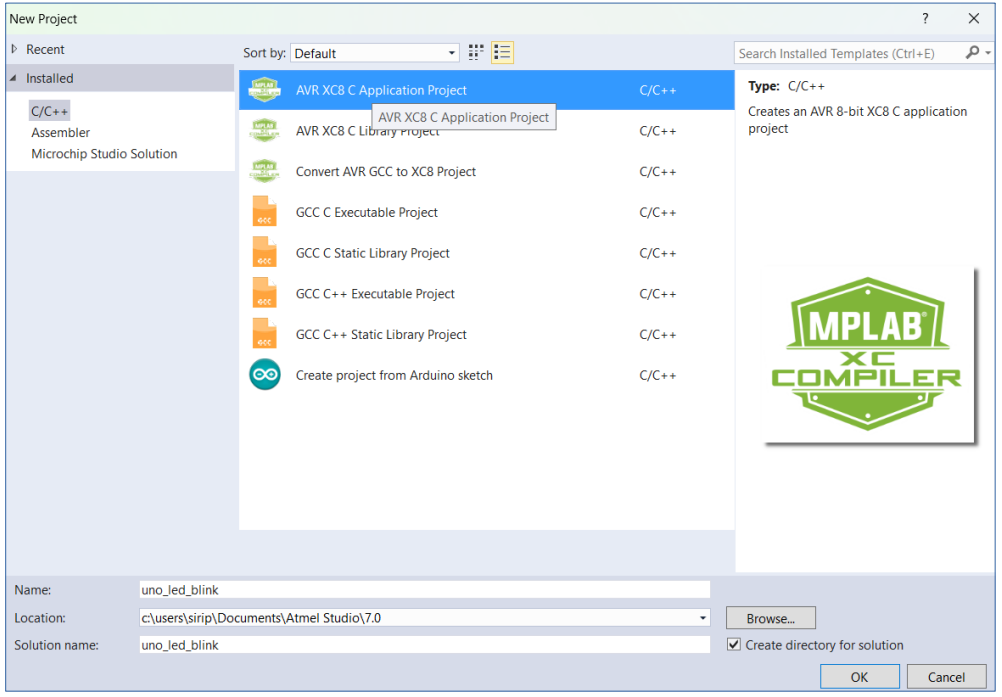
Note: Microsoft Visual Studio 2015 Shell (isolated) will be installed.

Microchip Studio v7.0.2594 (2022)

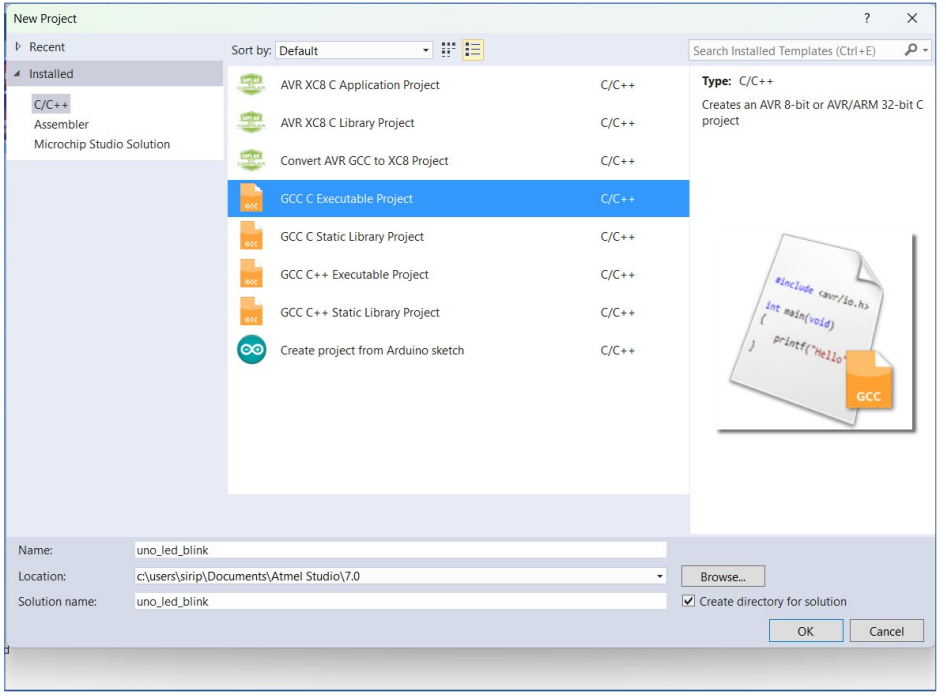


Microchip Studio v7.0.2594 (2022)

New Project Creation

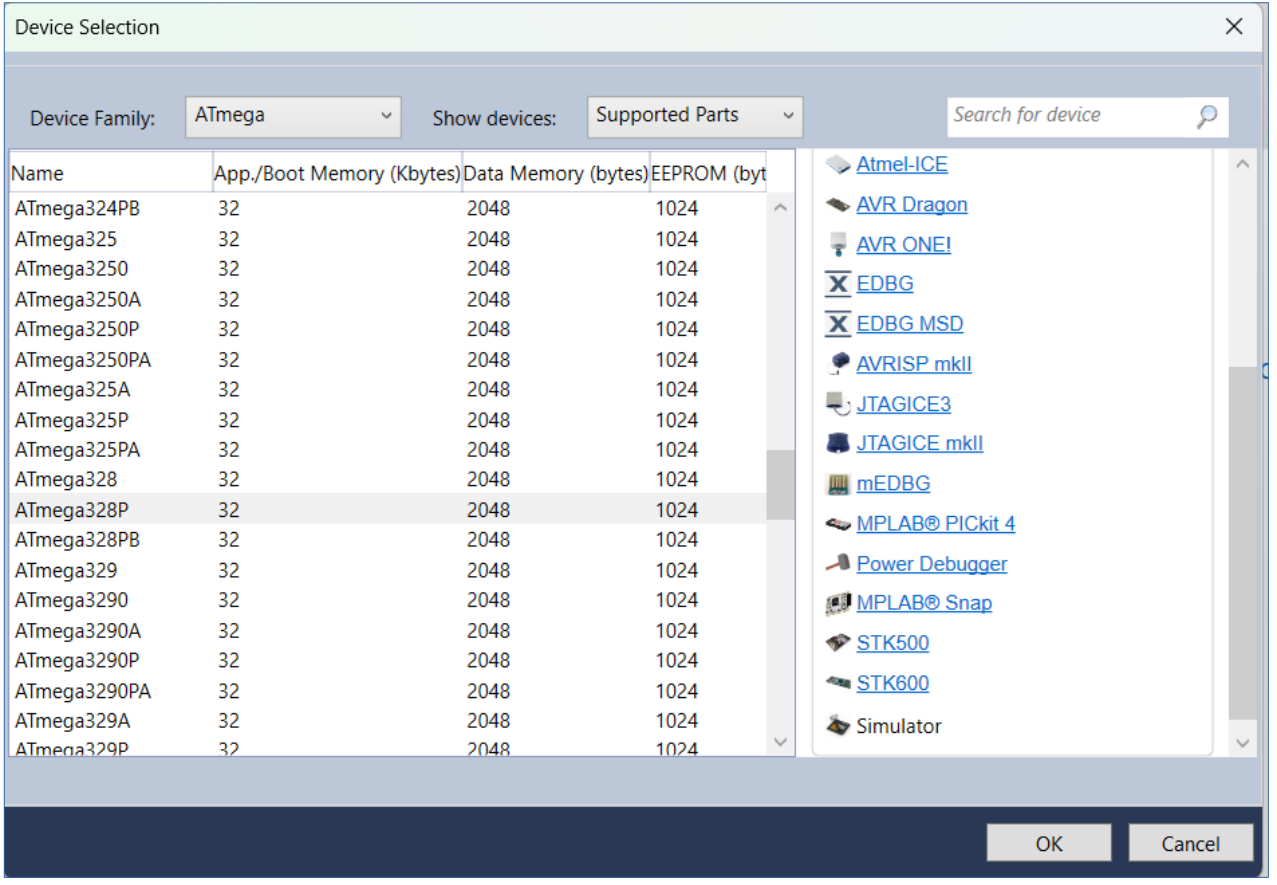


AVR XC8 C Application Project with MC8 Compiler toolchain



C Executable Project with AVR-GCC Compiler toolchain

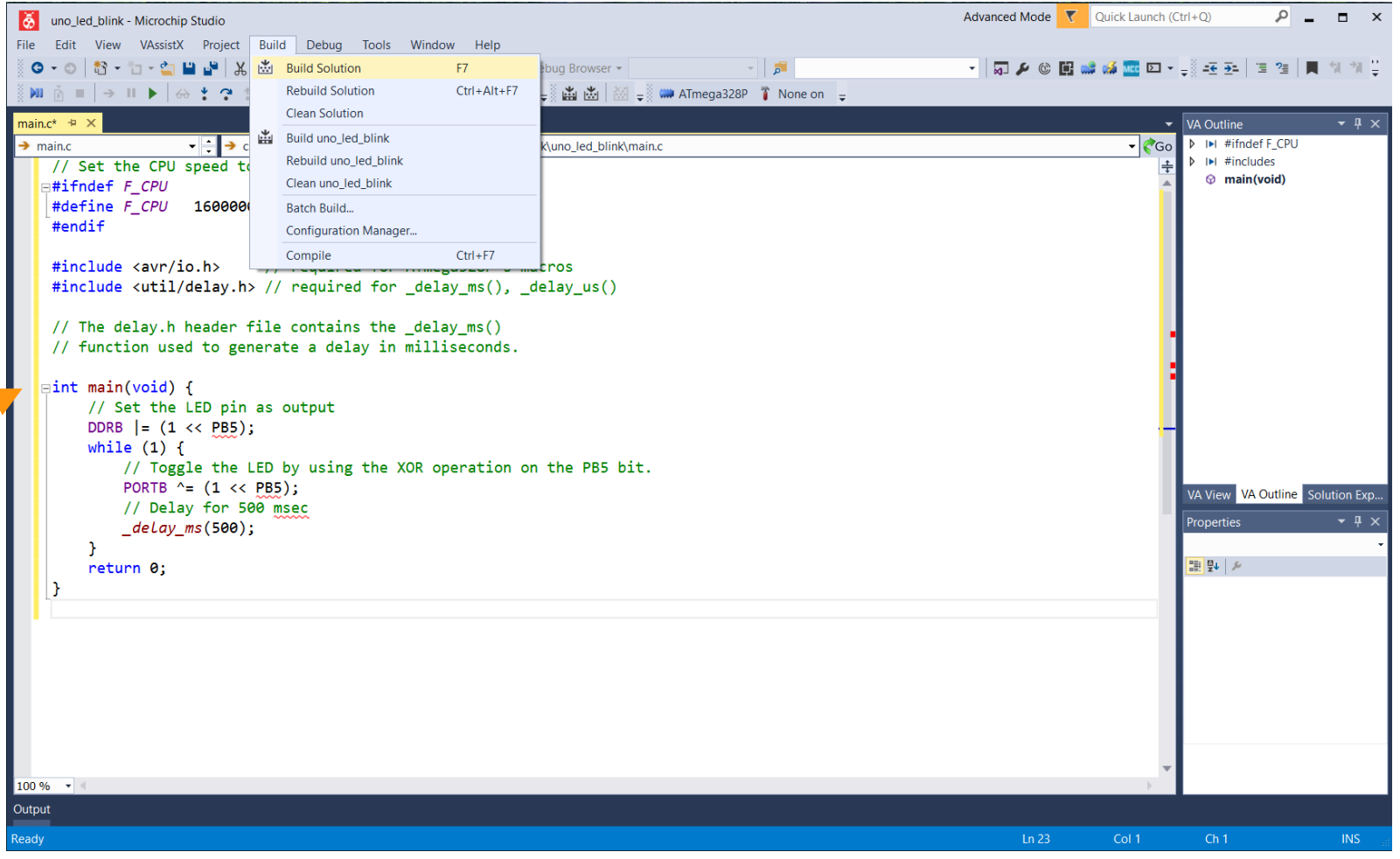
Microchip Studio v7.0.2594 (2022)



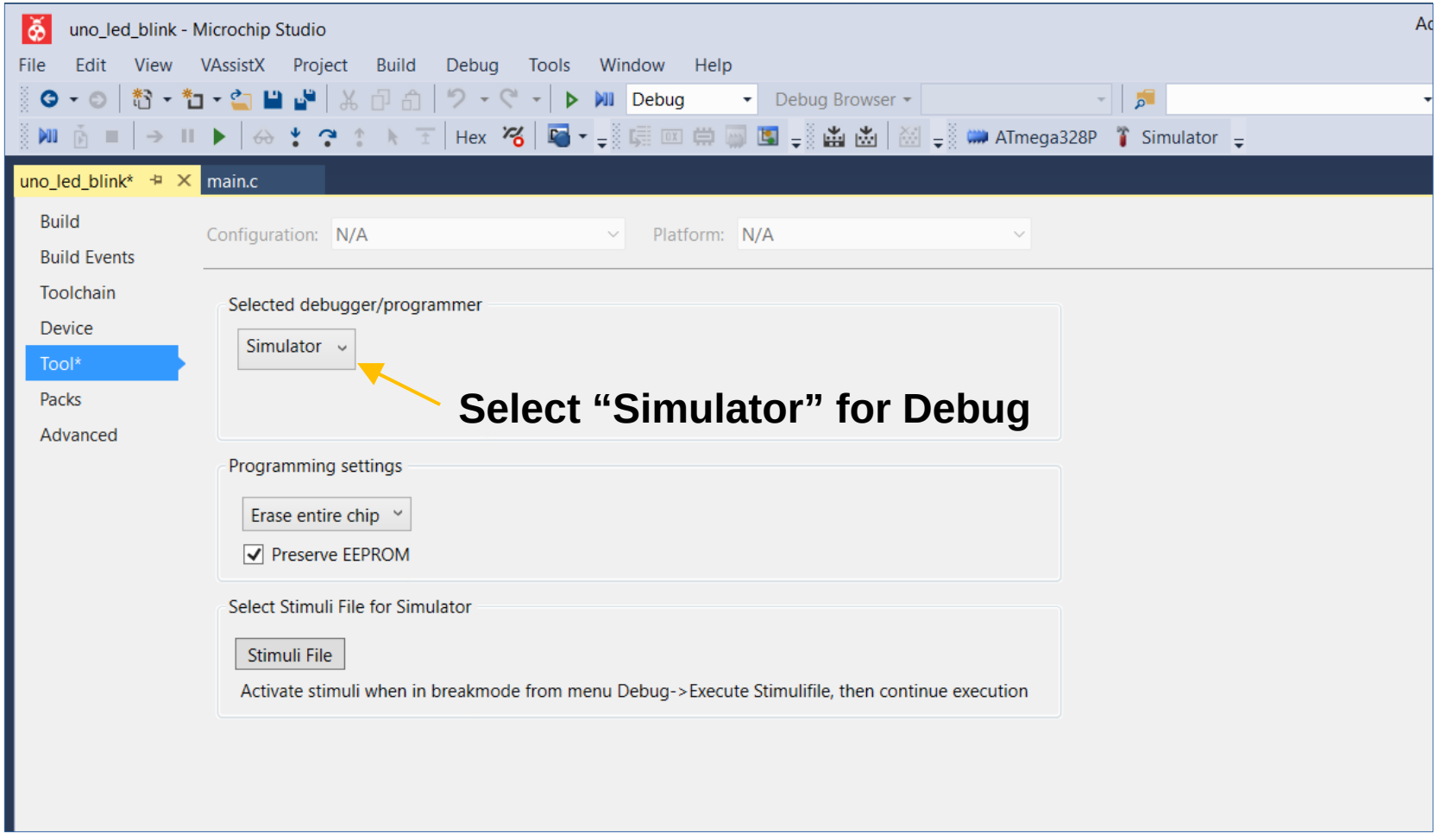
Target Device Selection: ATmega328P

Microchip Studio v7.0.2594 (2022)

Sample C Code (main.c)



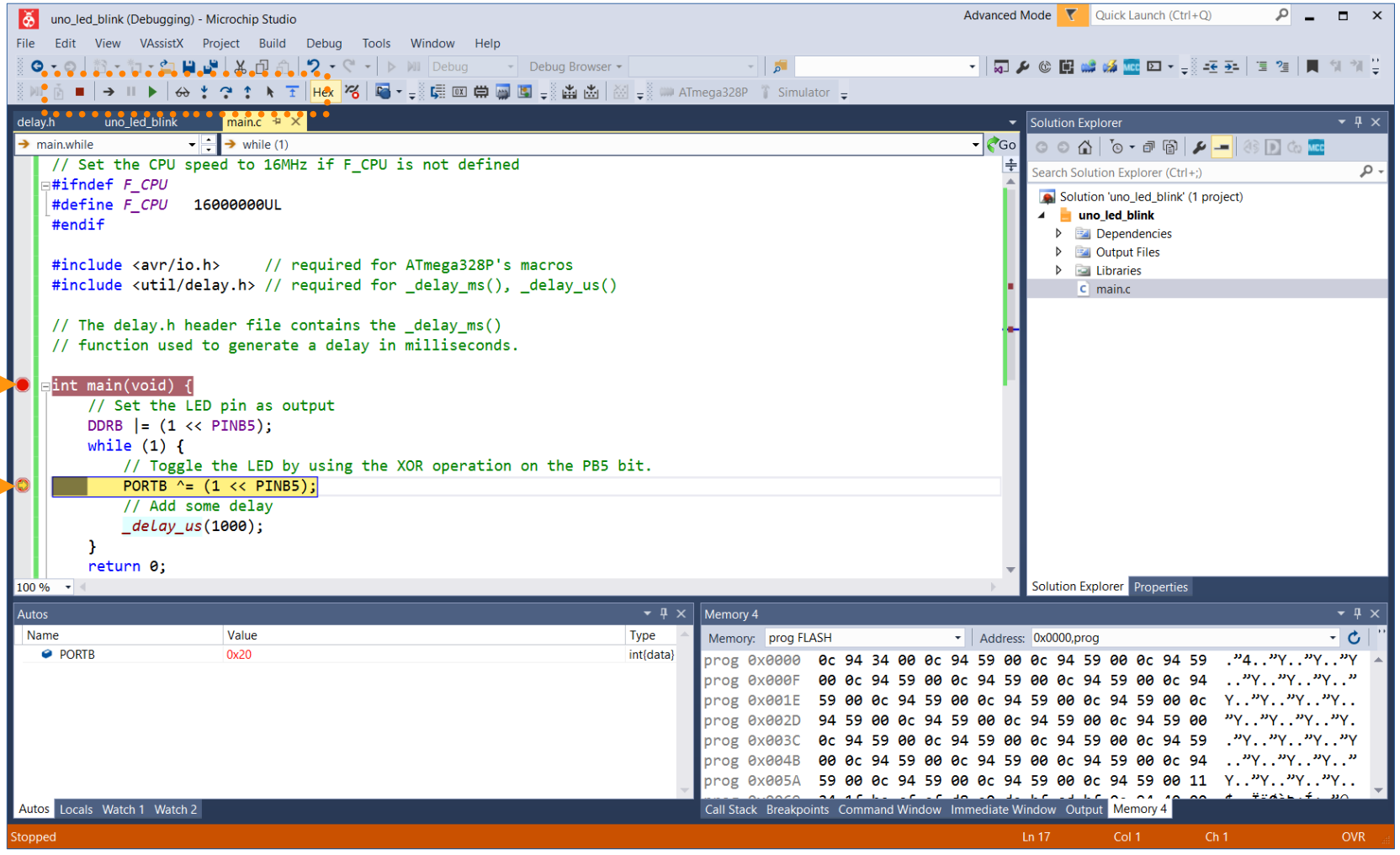
Microchip Studio v7.0.2594 (2022)



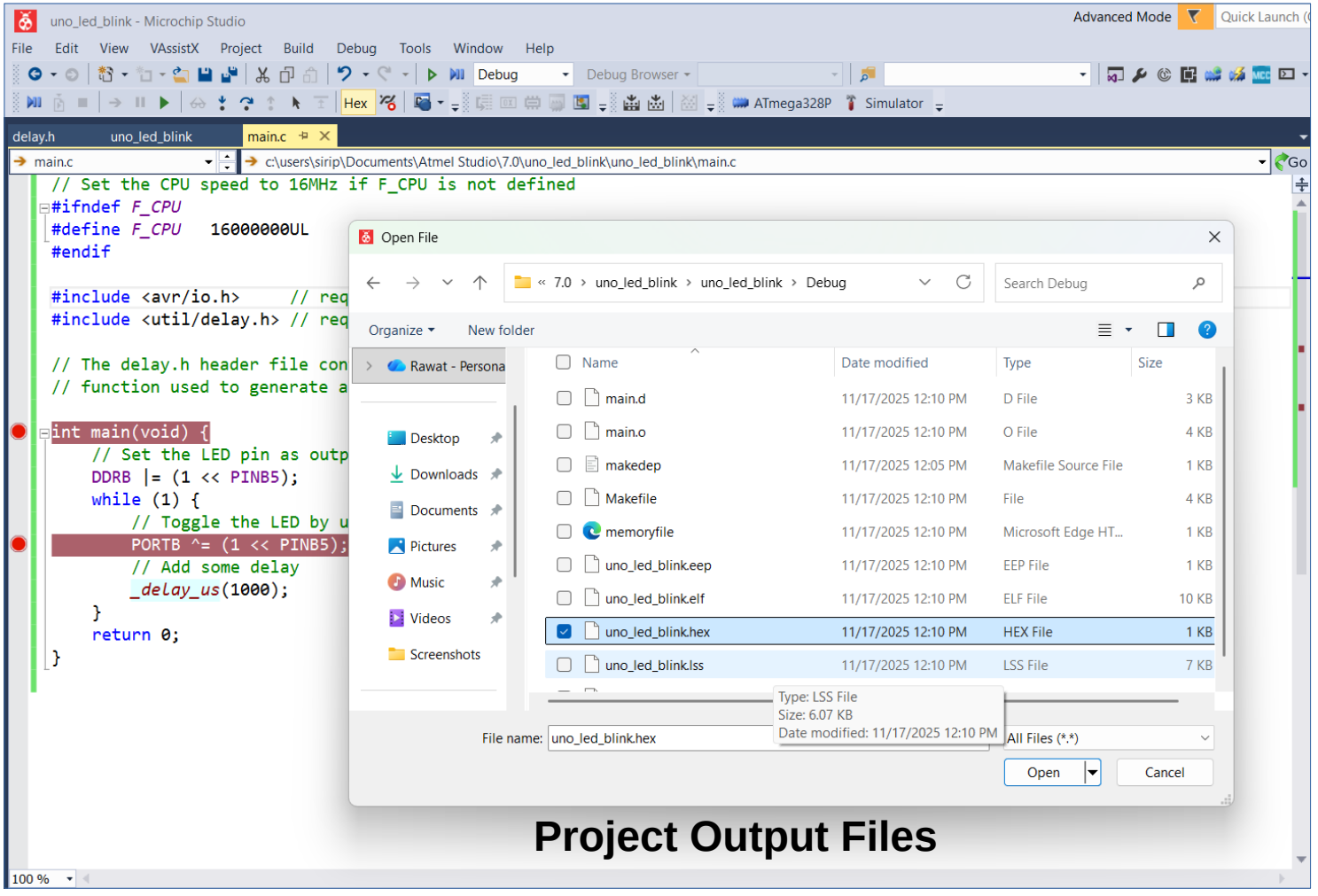
Microchip Studio v7.0.2594 (2022)

Debug Control

Break points



Microchip Studio v7.0.2594 (2022)



Project Output Files

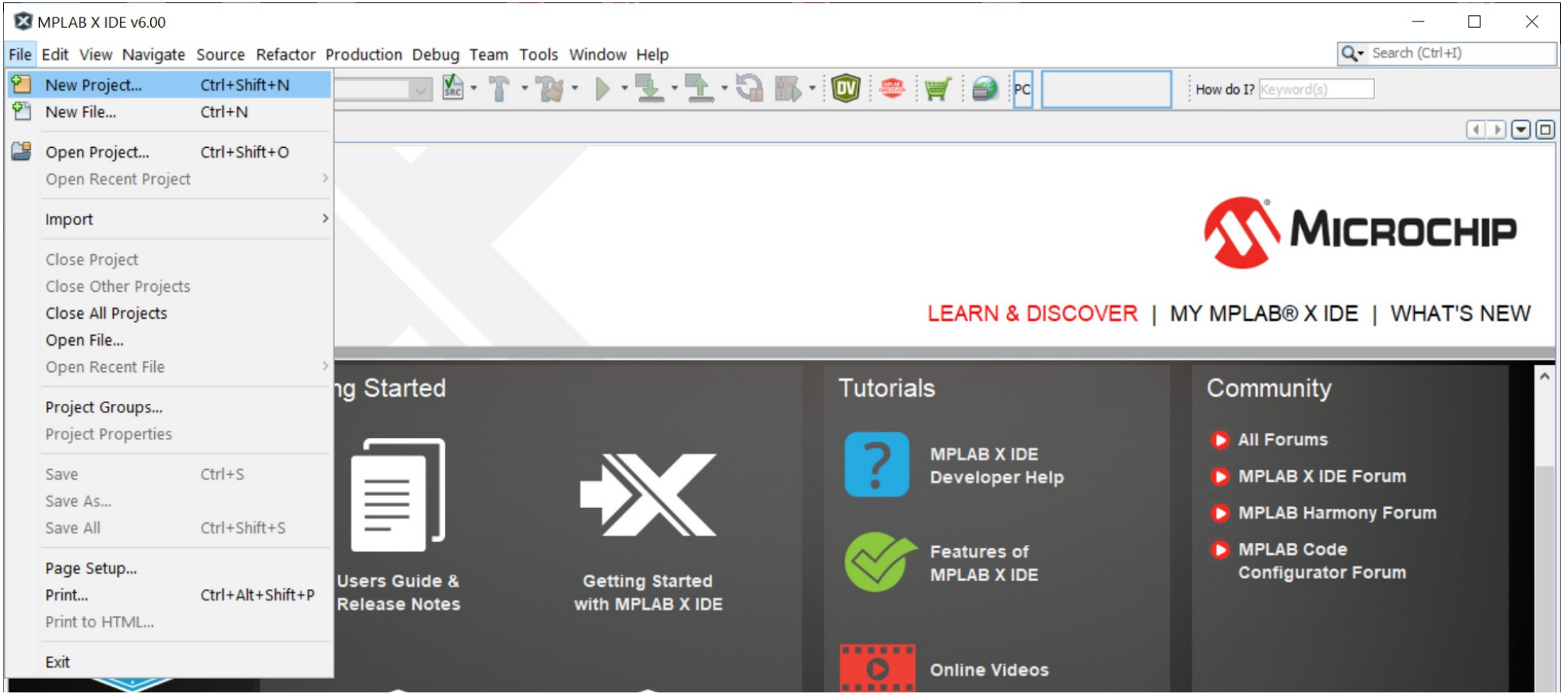
Sample C Code

```
#define F_CPU 16000000UL    // 16MHz Crystal Frequency

#include <avr/io.h>         // for SFRs: DDRB, PORTB, …
#include <util/delay.h>     // for function _delay_ms()

int main(void) {
  DDRB |= (1 << PB5);        // use PORTB5 (Arduino D13) as output
  while (1) {
    PORTB |= (1 << PB5);     // Turn LED on
    _delay_ms( 500 );        // Wait 500 msec.
    PORTB &= ~(1 << PB5);    // Turn LED off
    _delay_ms( 500 );        // Wait 500 msec.
  }
  return 0;
}
```

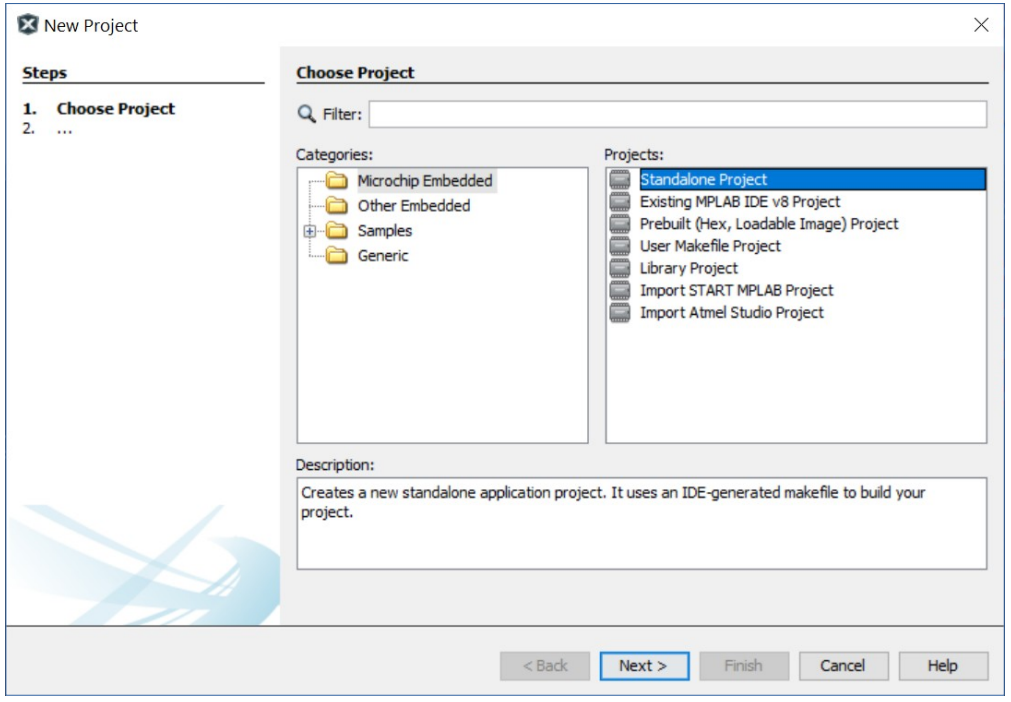
API Reference: https://www.nongnu.org/avr-libc/

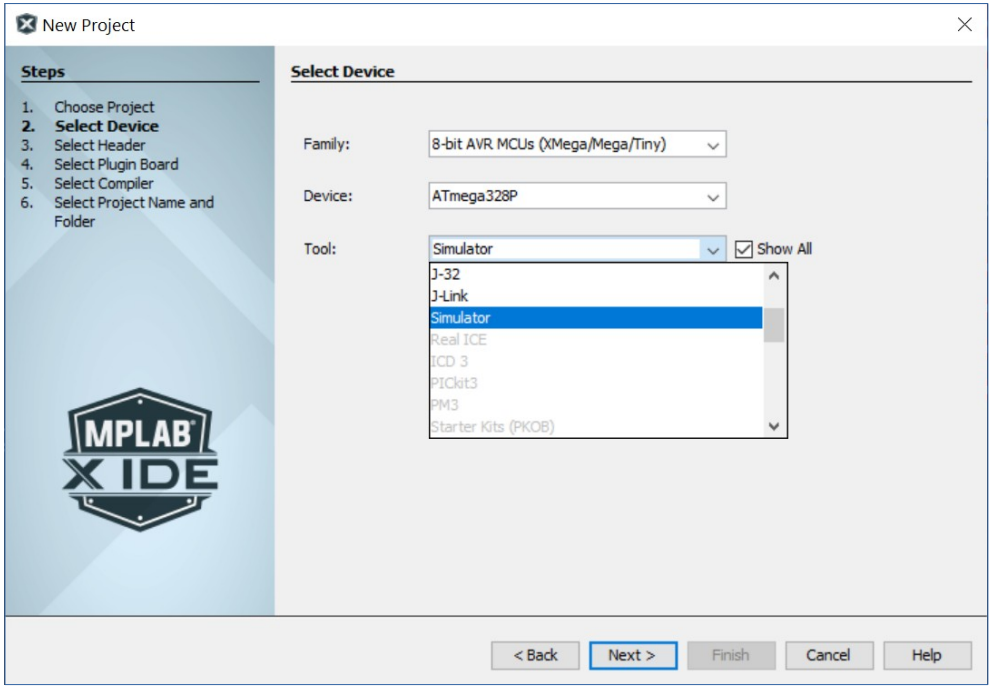
DDRB = The Port B Data Direction Register
PORTB = The Port B Data Register
PINB = The Port B Input Register

Microchip MPLABX IDE

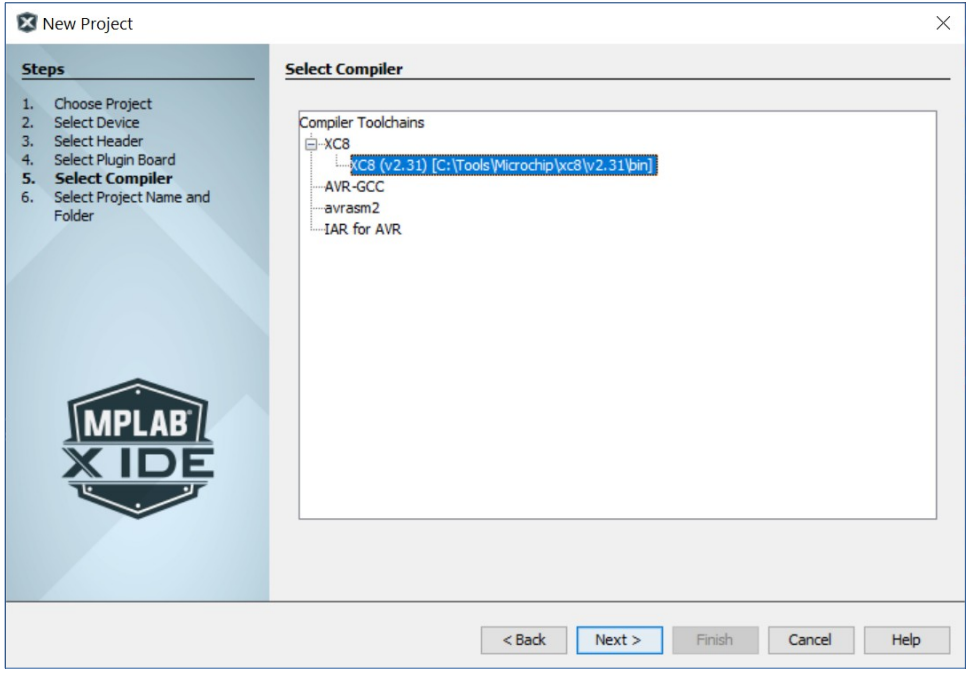


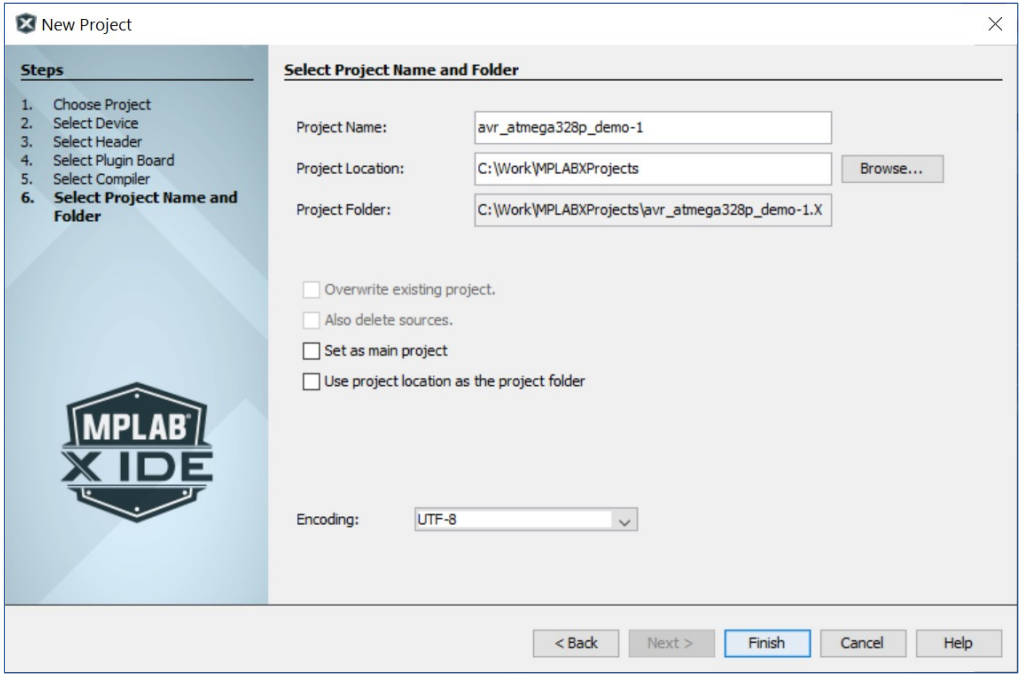
Microchip MPLABX IDE



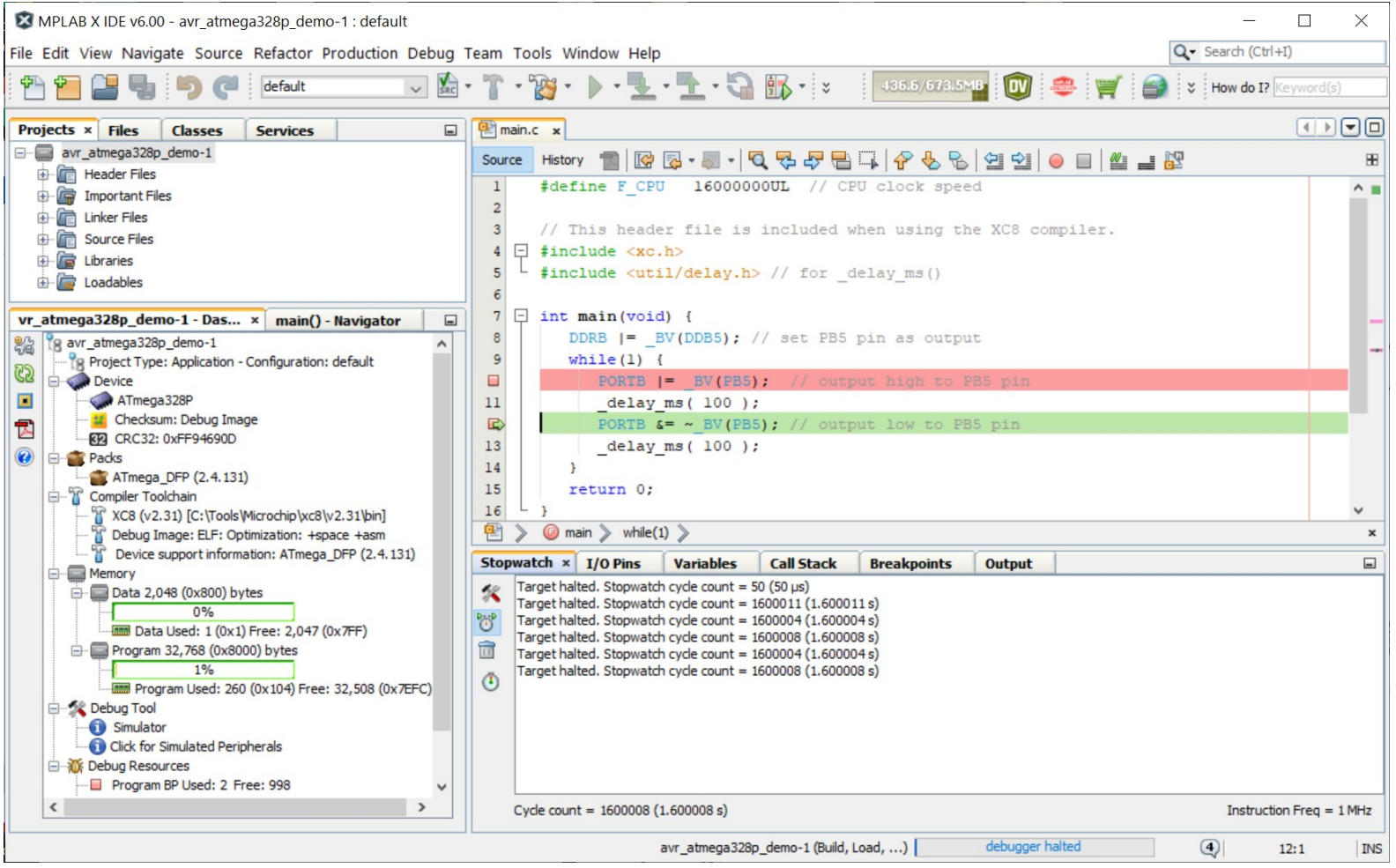


Microchip MPLABX IDE





Microchip MPLABX IDE



Source: https://iot-kmutnb.github.io/blogs/arduino/mplab-x_ide_avr/

CLI Tools: AVR-GCC and AVRDUDE

For Windows platforms:

1) Download File: avr8-gnu-toolchain-4.0.0.52-win32.any.x86_64.zip
from the following URL.

https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers

2) Download File: avrdude-v8.1-windows-x64.zip
from the following URL.

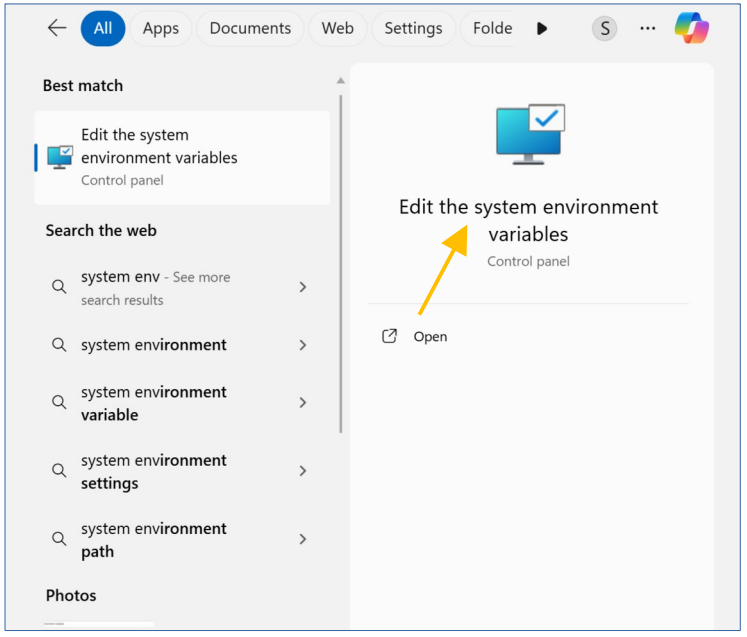
https://github.com/avrdudes/avrdude/releases

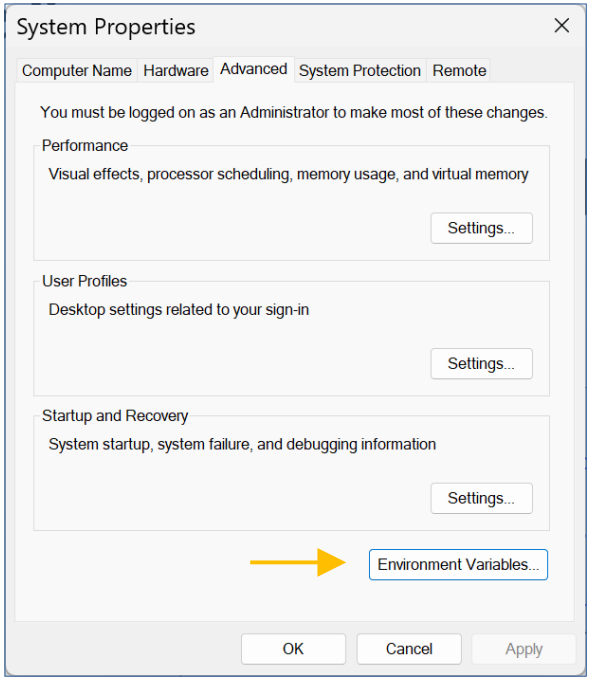
3) Unzip the archive files and move / rename the folders to following the folders

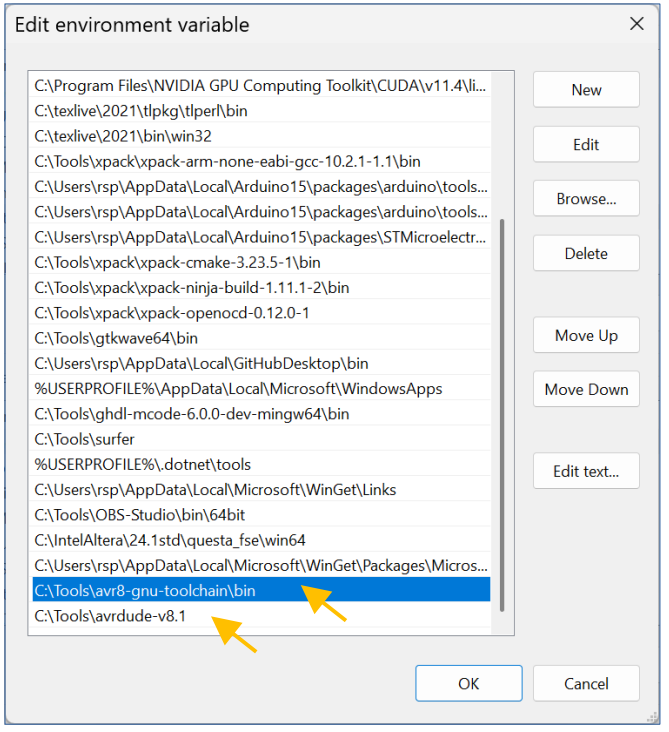
C:\Tools\avr8-gnu-toolchain\bin

C:\Tools\avrdude-v8.1

How to Add a CLI Path to PATH (Windows 11)

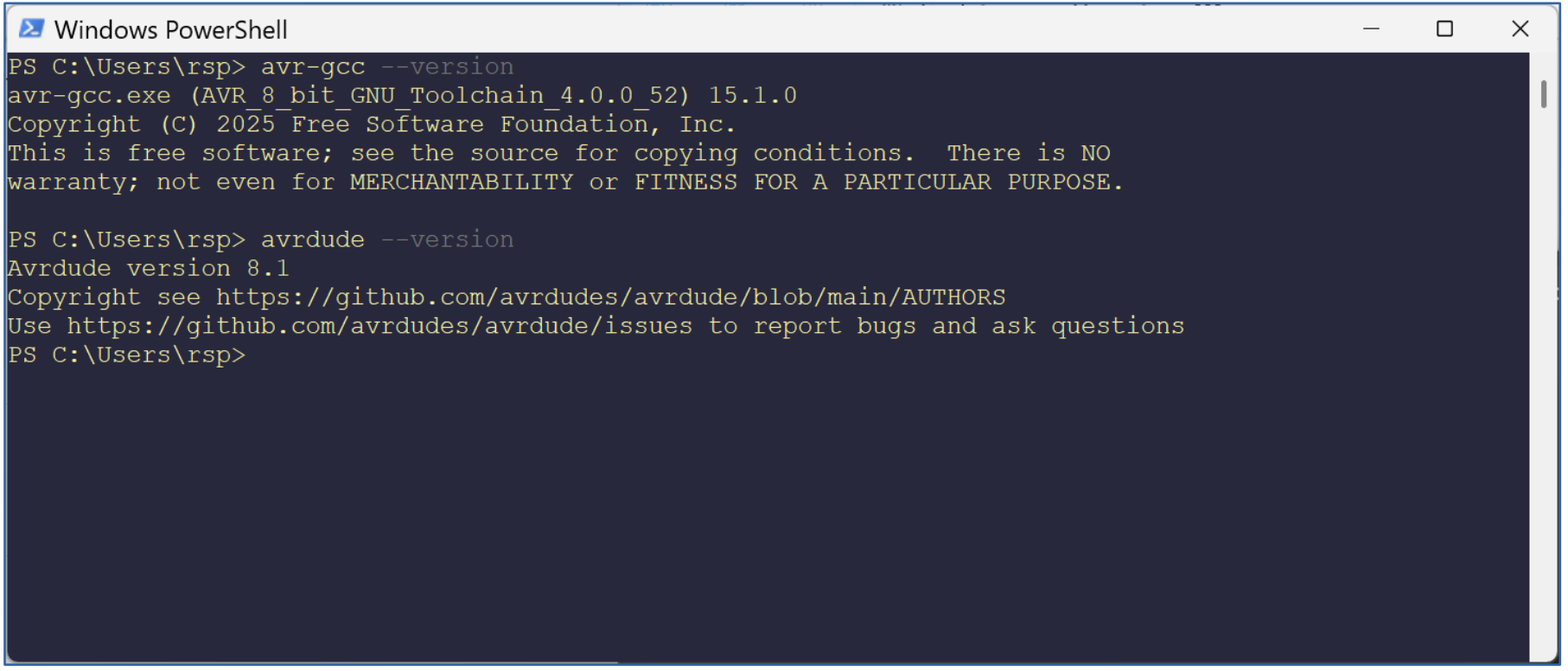






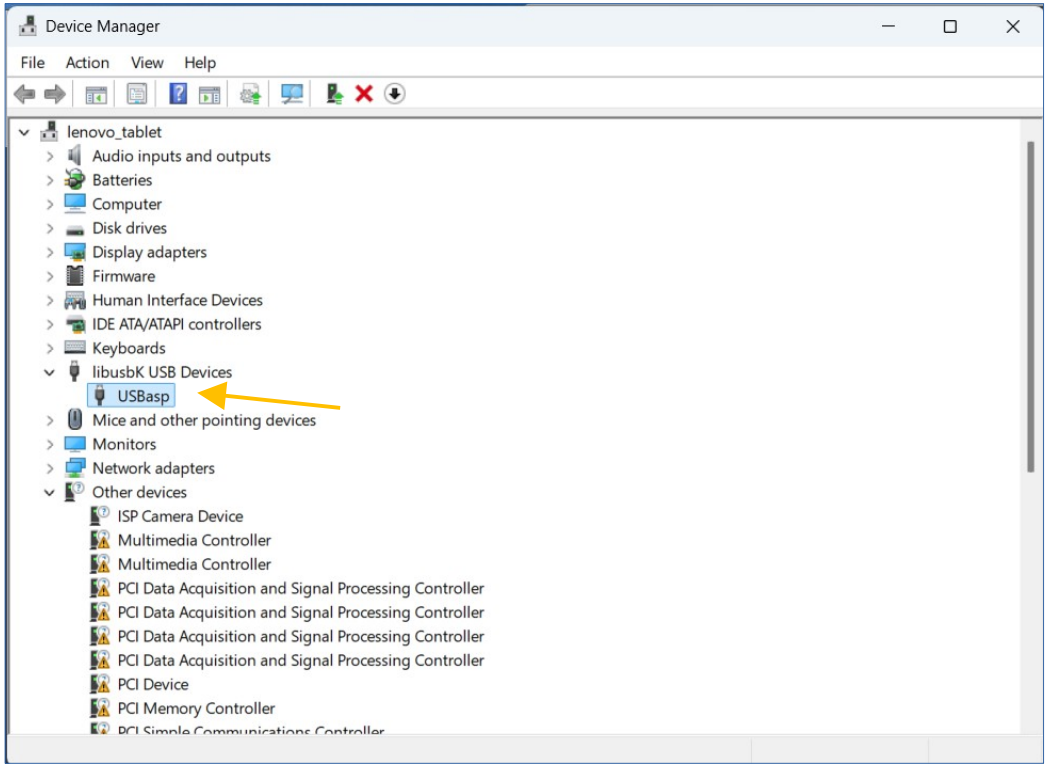
- Click "**Edit the system environment variables**".
- Under User variables, find the variable named **Path**.
- Select Path → click Edit and **add a New Path Entry**.
- Enter the full path to a new CLI tool (such as **avr-gcc** and **avrdude**).

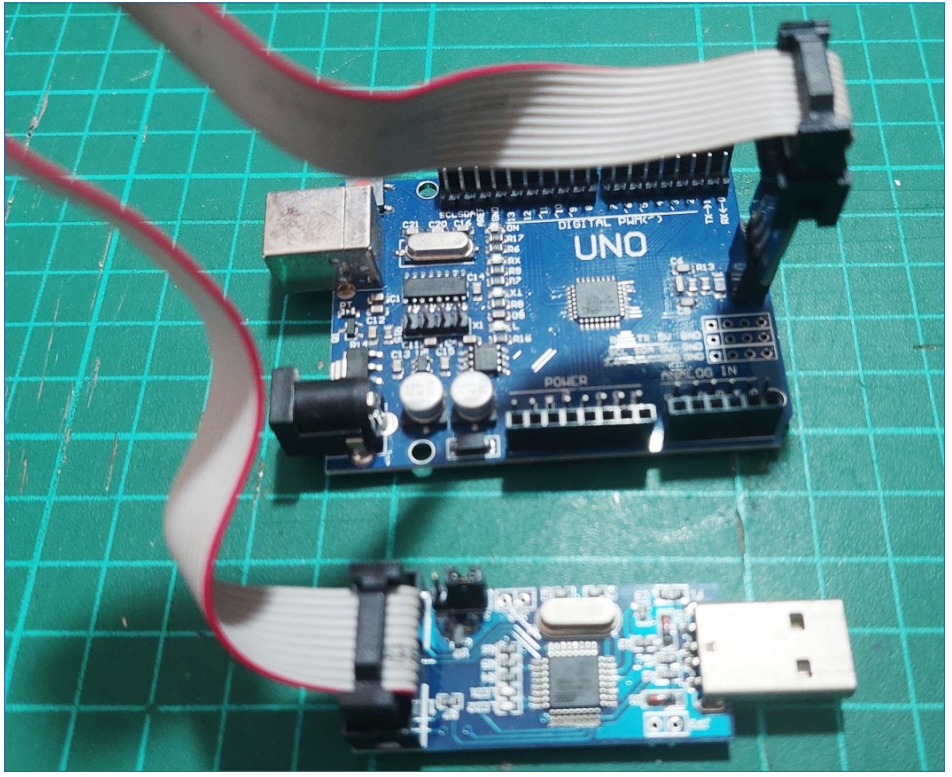
Using CLI Tools under Windows 11



Try to run the avr-gcc.exe and avrdude.exe command in Windows Powershell.

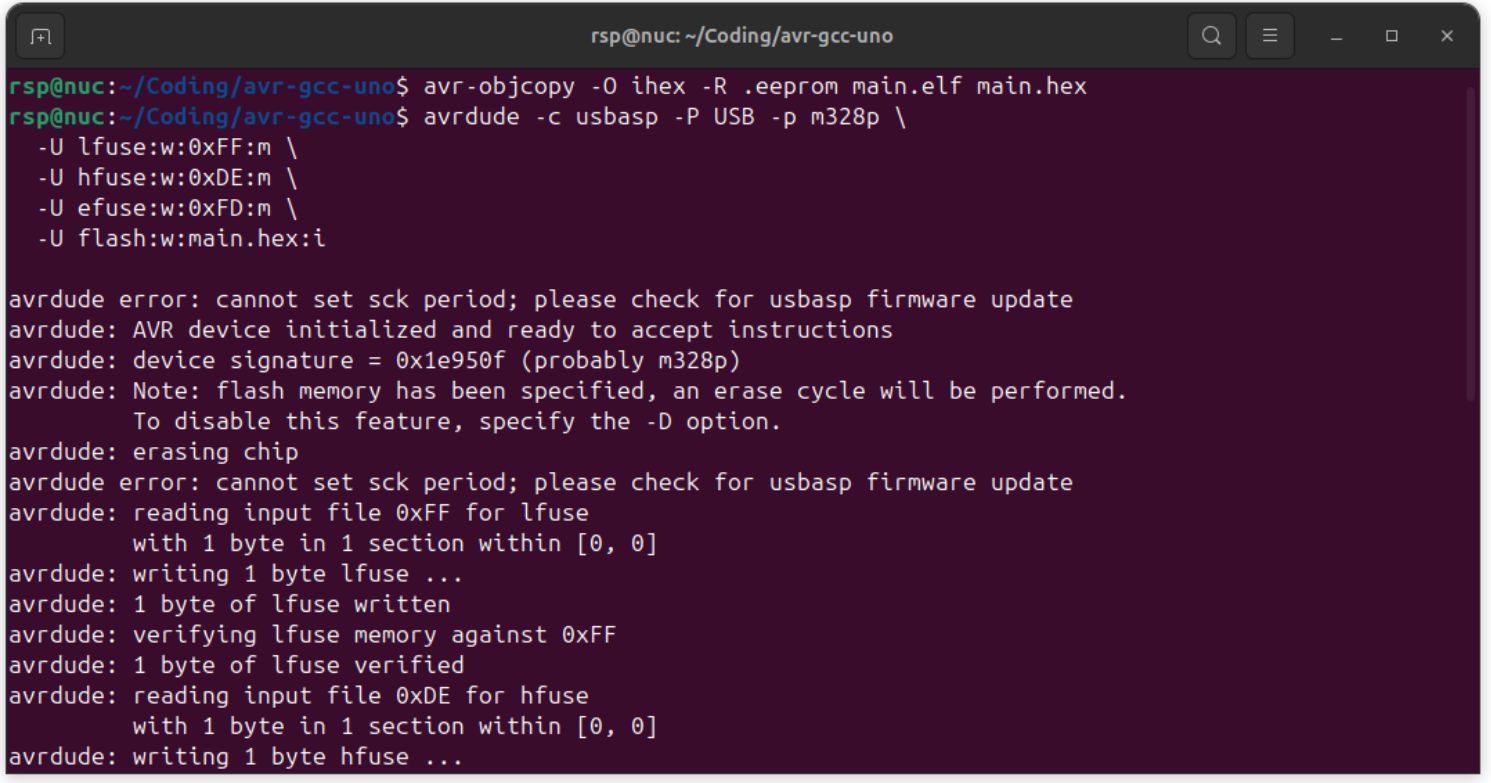
Using USBasp under Windows 11





Make sure that the USB driver for USPasp is installed correctly.

Using CLI Tools under Ubuntu 24.04




```
$ sudo apt update
$ sudo apt install gcc-avr \
      binutils-avr avr-libc
```

gcc-avr → AVR C compiler

binutils-avr → assembler, linker

avr-libc → standard C library for AVR

Applying UDEV rules for USBasp Under Linux

For **Ubuntu or other Debian-based Linux** distributions, you must add the following **UDEV rules file** to allow non-root users to access the **USBasp** device without using `sudo`.

```
$ sudo nano /etc/udev/rules.d/99-usbasp.rules
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="16c0", ATTR{idProduct}=="05dc",
GROUP="plugdev", MODE="0666"
```

A single line

Press Ctrl + O → save the file, Enter → confirm, and Ctrl + X → exit

Apply / update the udev rules.

```
$ sudo udevadm control --reload-rules && sudo udevadm trigger
```

Add the current user to the ‘plugdev’ group.

```
$ sudo groupadd plugdev
```

CLI under Ubuntu 24.04: AVR-GCC and AVRDUDE

```
# Compile the main.c source file and generate the .elf output file.
$ avr-gcc -mmcu=atmega328p -DF_CPU=16000000UL -Os -o main.elf main.c
# Convert the .elf file to an Intel HEX file.
$ avr-objcopy -O ihex -R .eeprom main.elf main.hex
# Use the avrdude tool with the USBasp programmer to flash the firmware.
$ avrdude -c usbasp -P USB -p m328p -D \
  -U lfuse:w:0xFF:m \
  -U hfuse:w:0xDE:m \
  -U efuse:w:0xFD:m \
  -U flash:w:main.hex:i
```

AVRDUDE Options

-c usbasp = USBasp programmer; Supports ISP (In-System Programming) via SPI
-P USB = Programmer is connected via USB. Detect it automatically.”
-p m328p = Specify the device part number "m328p" = ATmega328P.
-D Disable auto-erase before programming flash **to preserve the bootloader** and EEPROM. Normally, avrdude will erase entire chip, then writes flash.

AVRDUDE and ATmega328P Fuse Settings

`-U lfuse:w:0xFF:m`

Write **LOW fuse (lfuse)** with value 0xFF.

- External 16 MHz crystal
- No clock division (DIV8 disabled)
- Maximum startup time

m = "write using immediate mode"

`-U flash:w:main.hex:i`

Write to the MCU flash memory:

- flash → target program memory
- w → write mode
- main.hex → file to write
- i → Intel HEX format

`-U hfuse:w:0xDE:m`

Write **HIGH fuse (hfuse)** with value 0xDE.

- Bootloader size = 512 bytes
- Boot reset vector → jump to bootloader
- SPIEN → allow SPI programming
- EESAVE → preserve EEPROM on chip erase

This is required for the Arduino bootloader (Optiboot).

`-U efuse:w:0xFD:m`

Write **efuse (0xFD)** with value 0xFD.

- Set BOD (Brown-Out Detection) voltage level = 2.7V

Restoring Arduino Uno Bootloader

1) Download the bootloader .hex file (for OptiBoot).

URL: https://github.com/arduino/ArduinoCore-avr/raw/refs/heads/master/bootloaders/optiboot/optiboot_atmega328.hex

2) Use the AVRDUDE command to flash the firmware (with USBasp).

```
# Set the fuse bytes using USBasp
$ avrdude -c usbasp -p m328p \
    -U lfuse:w:0xFF:m \
    -U hfuse:w:0xDE:m \
    -U efuse:w:0xFD:m

# Flash the bootloader file using USBasp
$ avrdude -c usbasp -p m328p \
    -U flash:w:optiboot_atmega328.hex:i
```

Restoring Arduino Uno Bootloader using ArduinoISP

```
$ avrdude -c avrisp -b 19200 -P /dev/ttyACM0 -p m328p

avrdude: AVR device initialized and ready to accept instructions
         device signature = 0x1e950f (probably m328p)

# Flash the Optoboot bootloader file using Arduino-ISP
$ avrdude -c avrisp -b 19200 -P /dev/ttyACM0 -p m328p \
  -U lfuse:w:0xFF:m -U hfuse:w:0xDE:m -U efuse:w:0xFD:m \
  -U flash:w:optiboot_atmega328.hex:i

# Flash the Optoboot bootloader file using Arduino-ISP
$ avrdude -c avrisp -b 19200 -P /dev/ttyUSB0 -p m328p \
  -U lfuse:w:0xFF:m -U hfuse:w:0xDE:m -U efuse:w:0xFD:m \
  -U flash:w:optiboot_atmega328.hex:i
```

Typical Serial port name under Ubuntu: `/dev/ttyUSB0` or `/dev/ttyACM0`

Wokwi Simulator

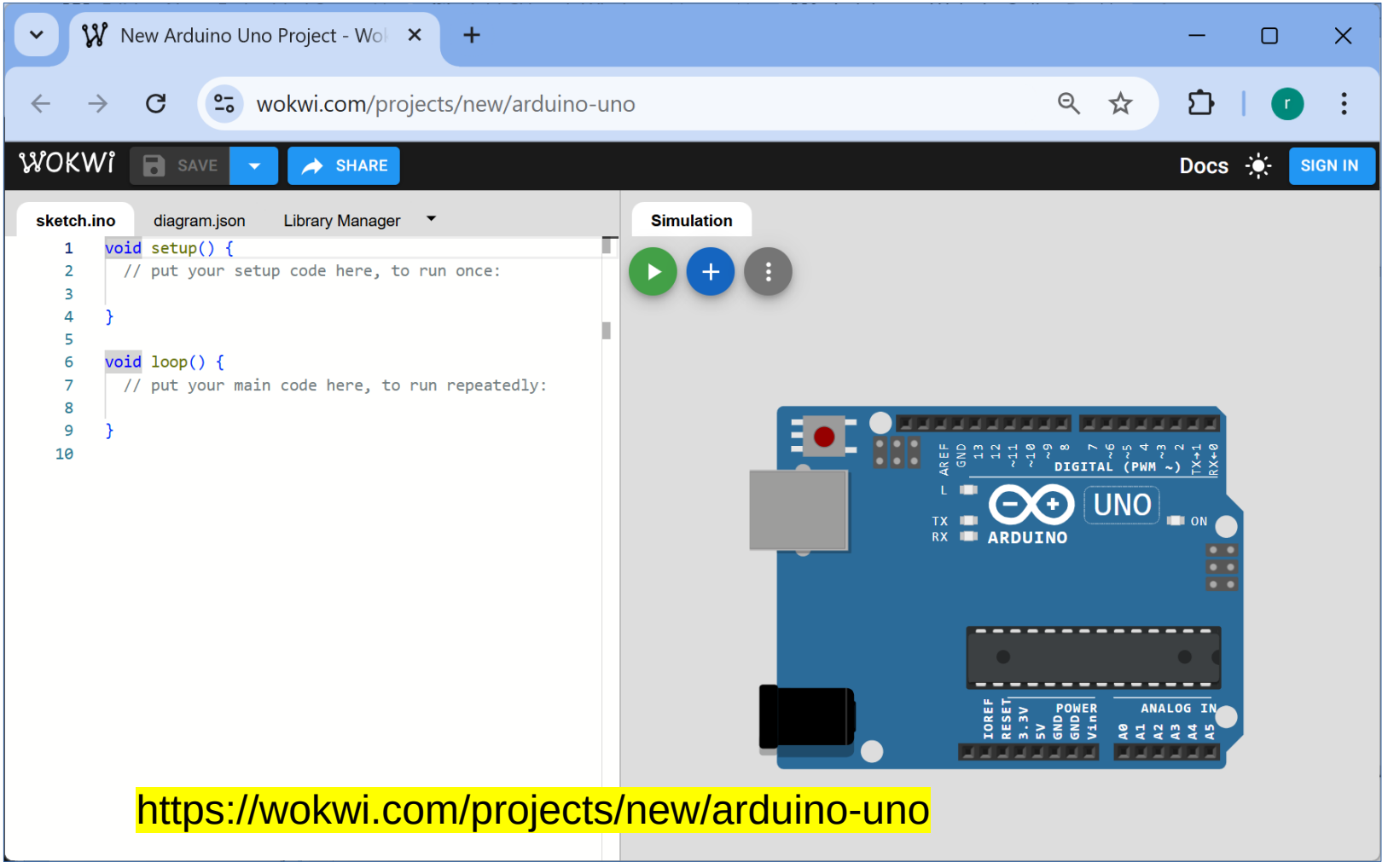
Wokwi AVR Simulator

- Uses **avr8js**, a **JavaScript instruction-set emulator for AVR**.
- Supports **C/C++ with AVR libc and AVR Assembly** code.
- Emulates the **ATmega328P** CPU at the instruction cycle level.
- Implements most AVR instructions, ALU operations, registers, stack, and interrupts.
- Executes code directly in the browser (client-side), no server needed.
- Cannot upload sketch to Arduino board.

Assembly Code (for AVR-GCC Assembler)

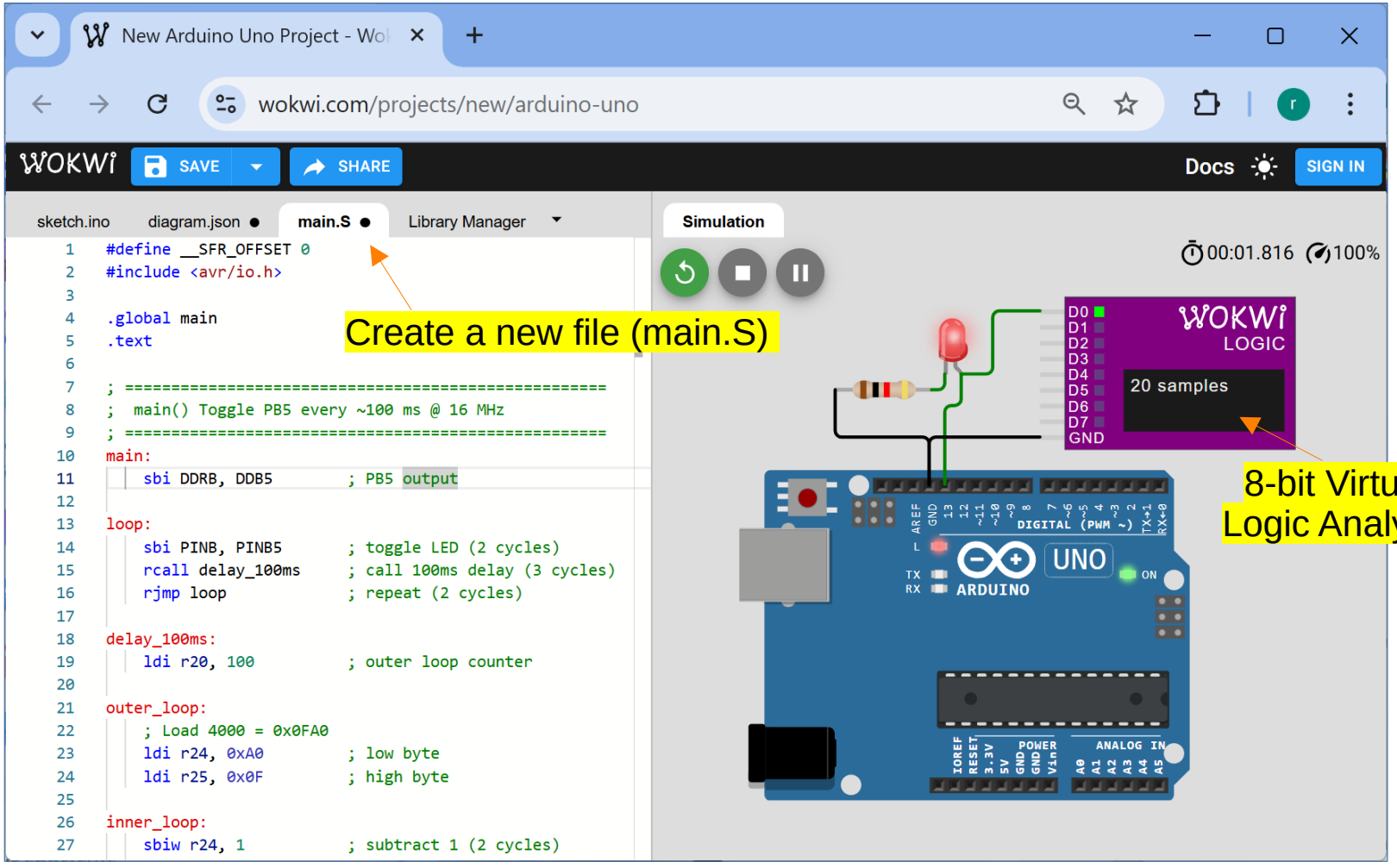
```
#define __SFR_OFFSET 0
#include <avr/io.h>

.global main
.text

; ================================================
;  main() Toggle PB5 every ~100 ms @ 16 MHz
; ================================================
main:
    sbi DDRB, DDB5        ; PB5 output

loop:
    sbi PINB, PINB5       ; toggle LED (2 cycles)
    rcall delay_100ms     ; call 100ms delay (3 cycles)
    rjmp loop             ; repeat (2 cycles)

delay_100ms:
    ldi r20, 100          ; outer loop counter

outer_loop:
    ; Load 4000 = 0x0FA0
    ldi r24, 0xA0         ; low byte
    ldi r25, 0x0F         ; high byte

inner_loop:
    sbiw r24, 1           ; subtract 1 (2 cycles)
    brne inner_loop       ; 2 cycles (taken) or 1 (last)
    dec r20               ; 1 cycle
    brne outer_loop       ; 2 cycles (taken) or 1 (last)
    ret                   ; 4 cycles
```

Wokwi Simulator: Uno Board



https://wokwi.com/projects/new/arduino-uno

Wokwi Simulator: AVR Assembly Simulation

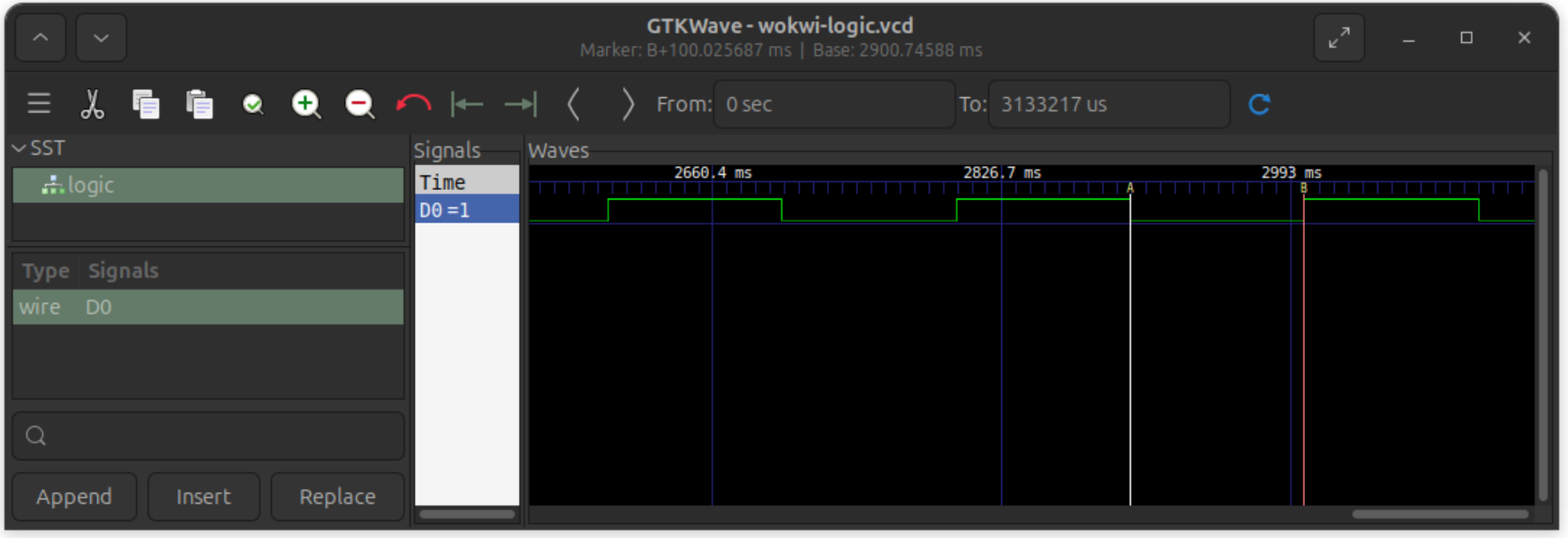



```
1   #define __SFR_OFFSET 0
2   #include <avr/io.h>
3
4   .global main
5   .text
6
7   ; ====================================================
8   ;  main() Toggle PB5 every ~100 ms @ 16 MHz
9   ; ====================================================
10  main:
11      sbi DDRB, DDB5        ; PB5 output
12
13  loop:
14      sbi PINB, PINB5       ; toggle LED (2 cycles)
15      rcall delay_100ms     ; call 100ms delay (3 cycles)
16      rjmp loop             ; repeat (2 cycles)
17
18  delay_100ms:
19      ldi r20, 100          ; outer loop counter
20
21  outer_loop:
22      ; Load 4000 = 0x0FA0
23      ldi r24, 0xA0         ; low byte
24      ldi r25, 0x0F         ; high byte
25
26  inner_loop:
27      sbiw r24, 1           ; subtract 1 (2 cycles)
```

GTKWave: VCD Waveform Viewer

GTKWave is an open-source VCD waveform viewer.

In GTKWave, open the VCD waveform file (wokwi-logic.vcd) generated by the **Wokwi simulator's virtual logic analyzer** and downloaded from the online simulator.



Toggle Interval (measured): 100.02msec

How to compile AVR Assembly Code Using avr-gcc

```
# Compile the AVR assembly file (main.S) to an ELF file.
$ avr-gcc -mmcu=atmega328p -Os -o main.elf main.S
```

`-mmcu=atmega328p` → set the target MCU (same as Arduino Uno)

`-Os` → optimize for size

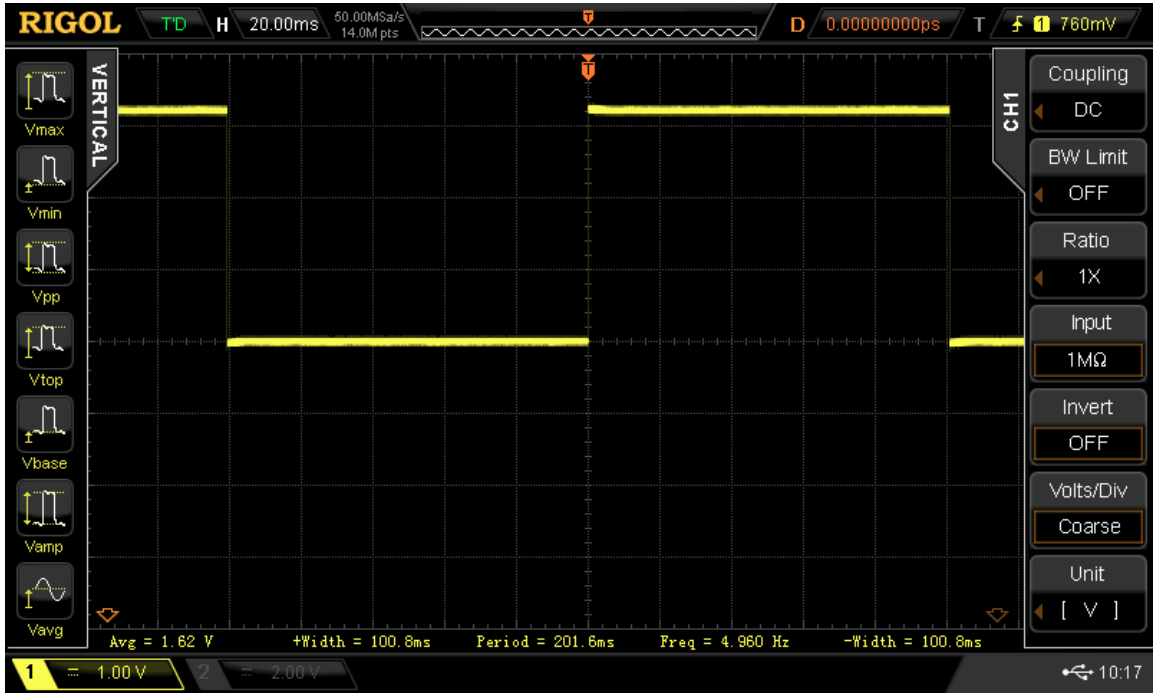
`.S` → preprocessed assembly

```
# Produce the HEX file from the ELF file.
$ avr-objcopy -O ihex -R .eeprom main.elf main.hex

# Convert ELF file to AVR assembly code listing.
$ avr-objdump -d main.elf
```

Upload HEX File to Arduino Uno

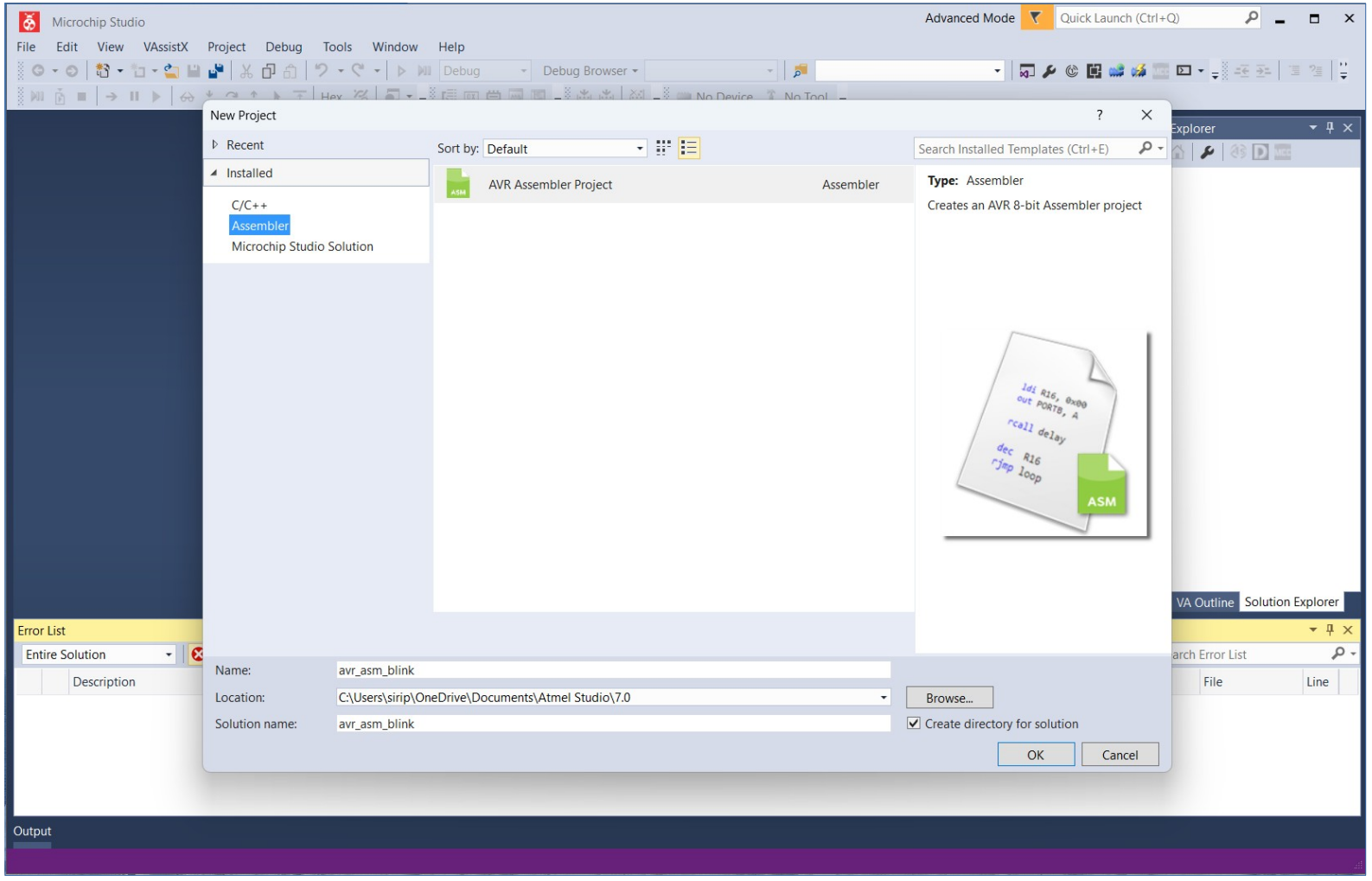
```
# Use avrdude with the USBasp programmer to flash the firmware.
# Need to overwrite the Arduino bootloader (without -D option).

$ avrdude -c usbasp -P USB -p m328p \
  -U lfuse:w:0xFF:m \
  -U hfuse:w:0xDE:m \
  -U efuse:w:0xFD:m \
  -U flash:w:main.hex:i
```

Signal Measurement with Digital Scope

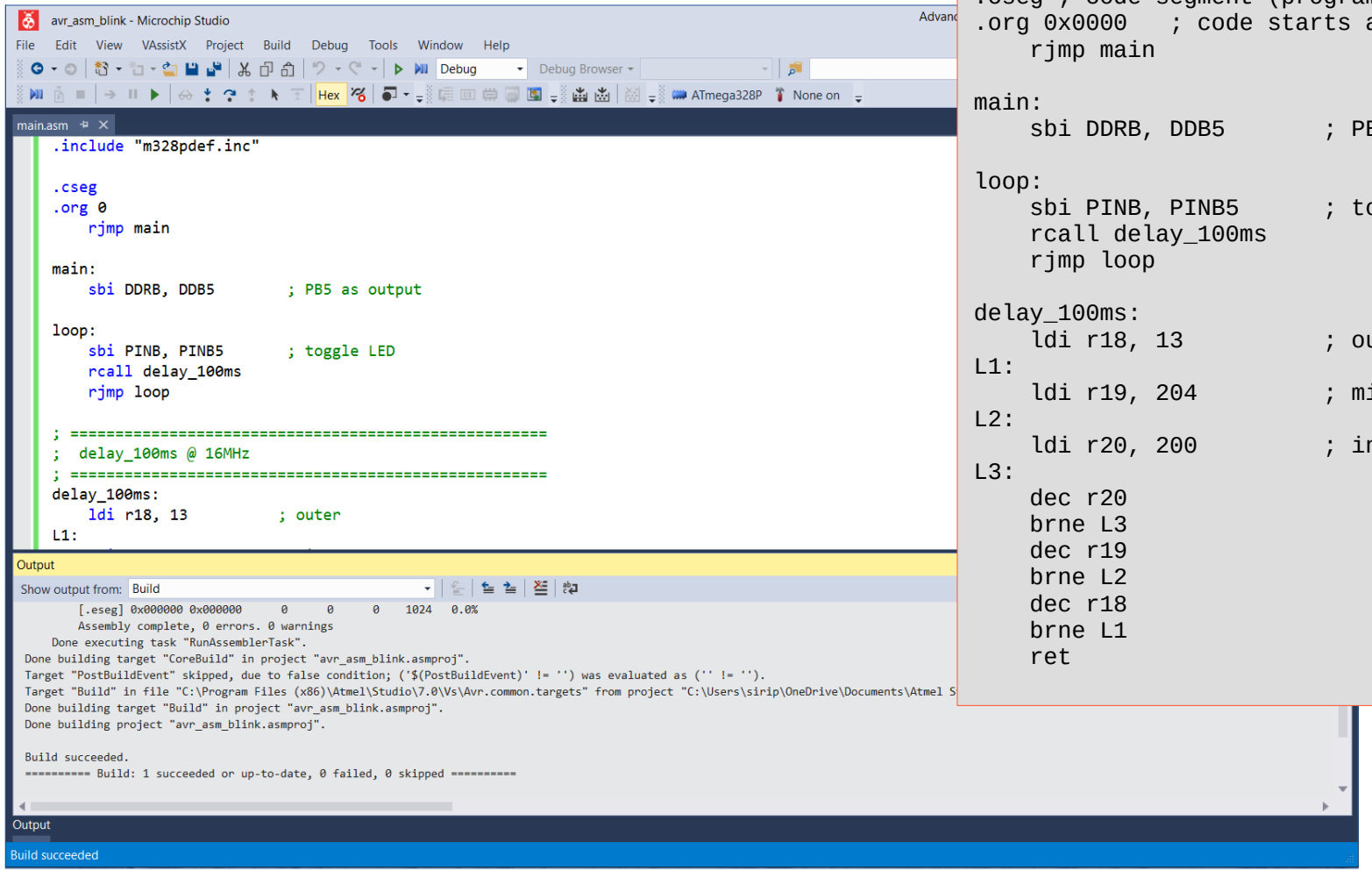


- *Do not rely solely on simulator results.*
- *Whenever possible, use a digital oscilloscope or a logic analyzer to measure, capture, and analyze the AVR's I/O waveforms on real hardware.*
- *This helps verify that the code behaves correctly under actual operating conditions.*

Microchip Studio: Create AVR Assembly Project

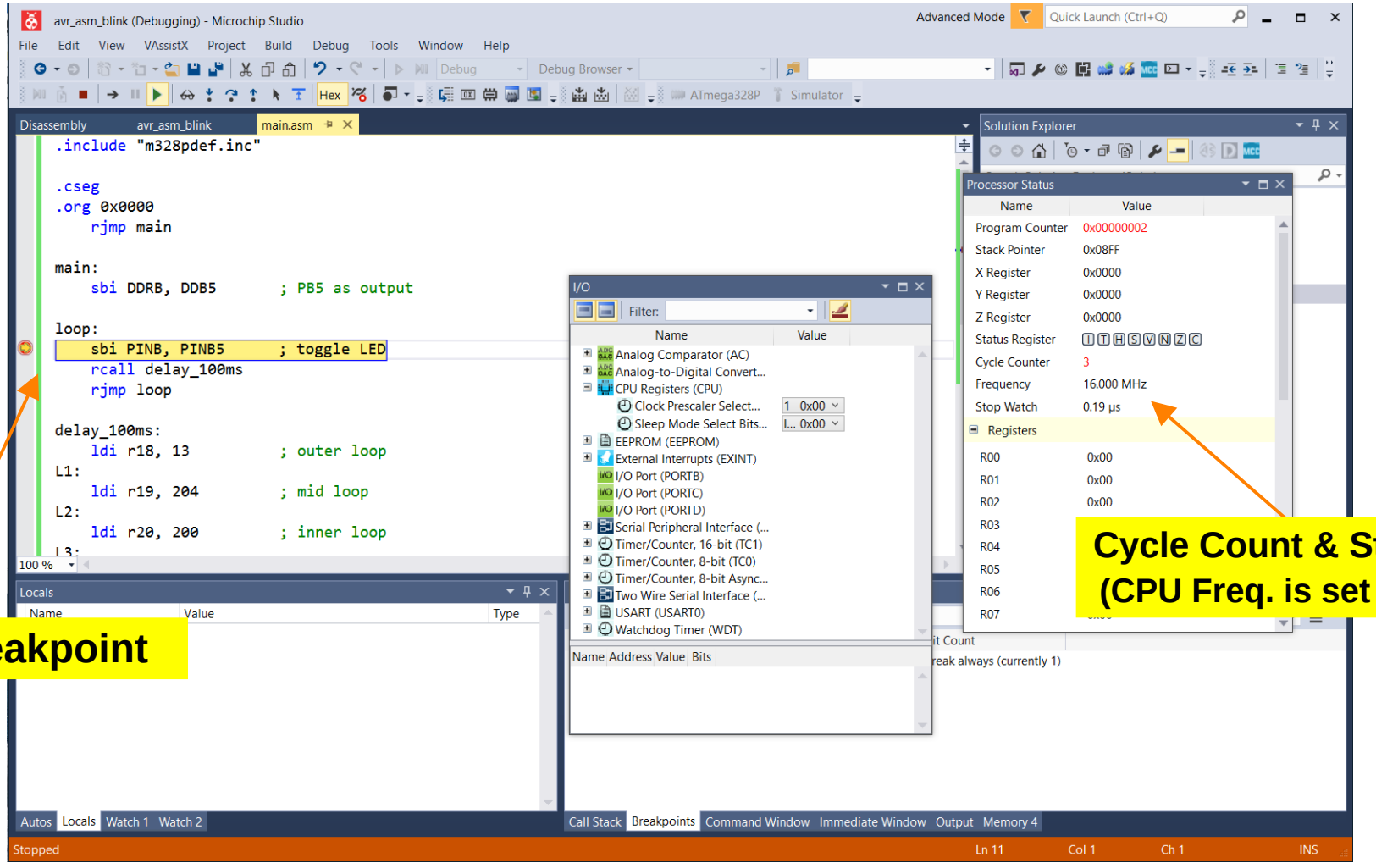


AVR Assembly

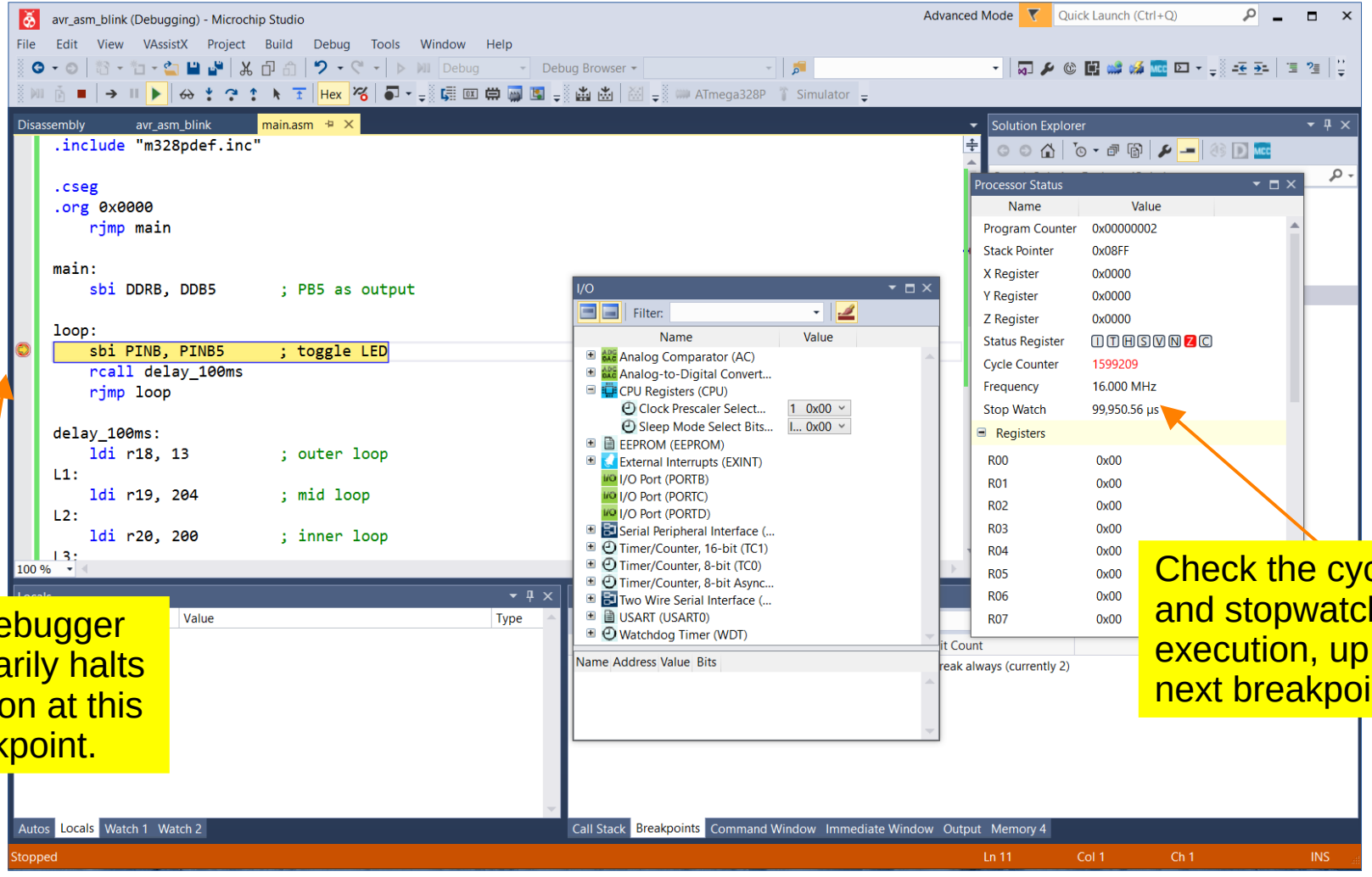



```
.include "m328pdef.inc"

.cseg ; code segment (program text)
.org 0x0000   ; code starts at 0x0000
    rjmp main

main:
    sbi DDRB, DDB5        ; PB5 as output

loop:
    sbi PINB, PINB5       ; toggle LED
    rcall delay_100ms
    rjmp loop

delay_100ms:
    ldi r18, 13           ; outer loop
L1:
    ldi r19, 204          ; mid loop
L2:
    ldi r20, 200          ; inner loop
L3:
    dec r20
    brne L3
    dec r19
    brne L2
    dec r18
    brne L1
    ret
```

AVR Simulation



AVR Simulation



The debugger temporarily halts execution at this breakpoint.

Check the cycle count and stopwatch after execution, up to the next breakpoint.

AVR-GCC Commands on Ubuntu

```
$ avr-gcc --version
avr-gcc (GCC) 14.2.0
```

Check avr-gcc version

```
$ avr-gcc -mmcu=atmega328p -DF_CPU=16000000UL \
 -Os -std=gnu11 main.c -o main.elf
```

Compile main.c (for C source code)

```
$ avr-gcc -mmcu=atmega328p -DF_CPU=16000000UL \
 -Os -nostartfiles main.s -o main.elf
```

Compile main.s (for assembly code)

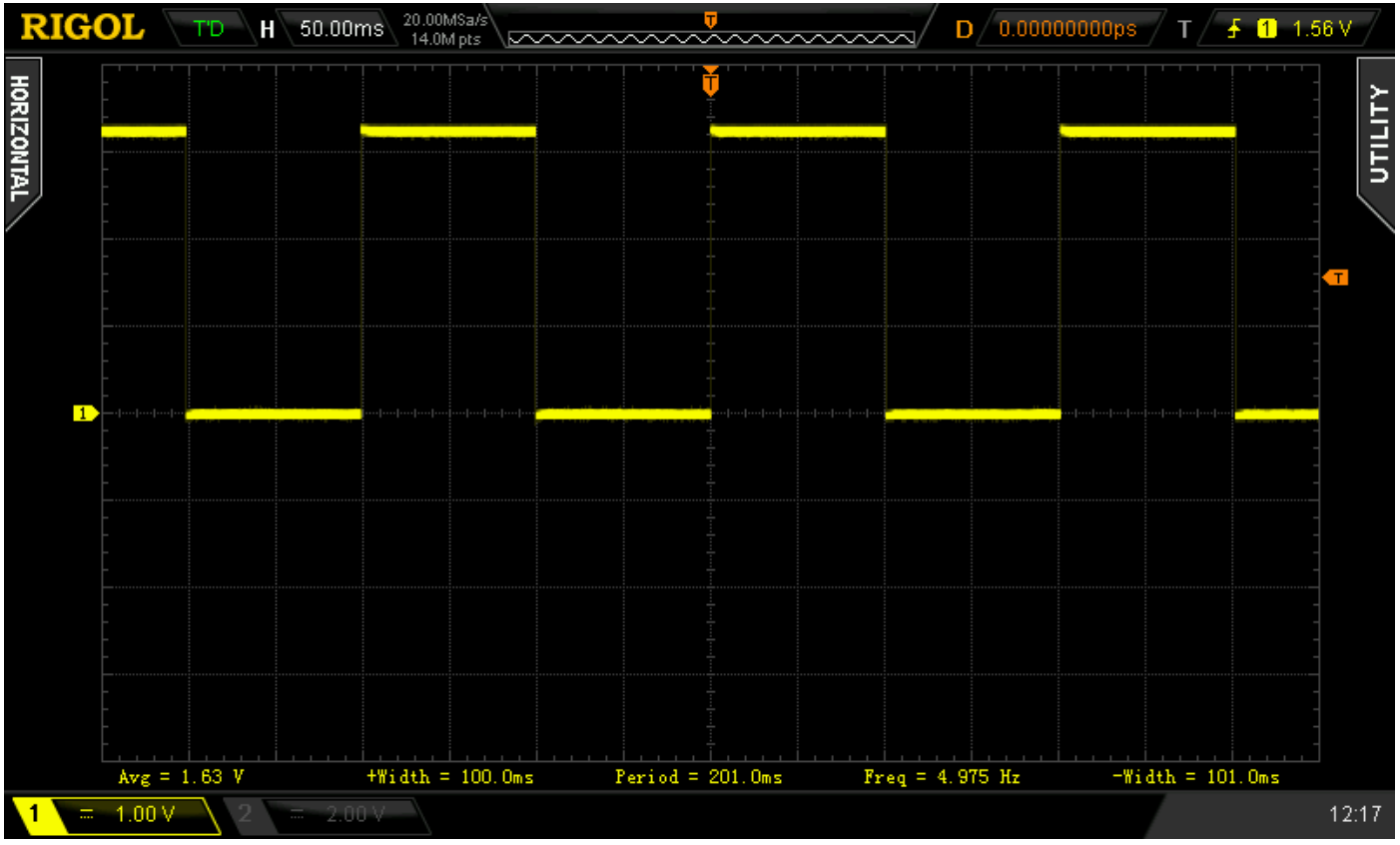
```
$ avr-objcopy -O ihex -R .eeprom main.elf main.hex
```

Convert .elf to .hex

```
$ avrdude -c arduino -p atmega328p \
  -P /dev/ttyUSB0 -b 115200 -D \
  -U flash:w:main.hex:i
```

Upload .hex to Arduino Uno or Nano (preserve the Arduino bootloader)

Signal Measurement with a Digital Oscilloscope



Toggle interval (measured): 100 msec

AVR Assembly Code: Directive Differences

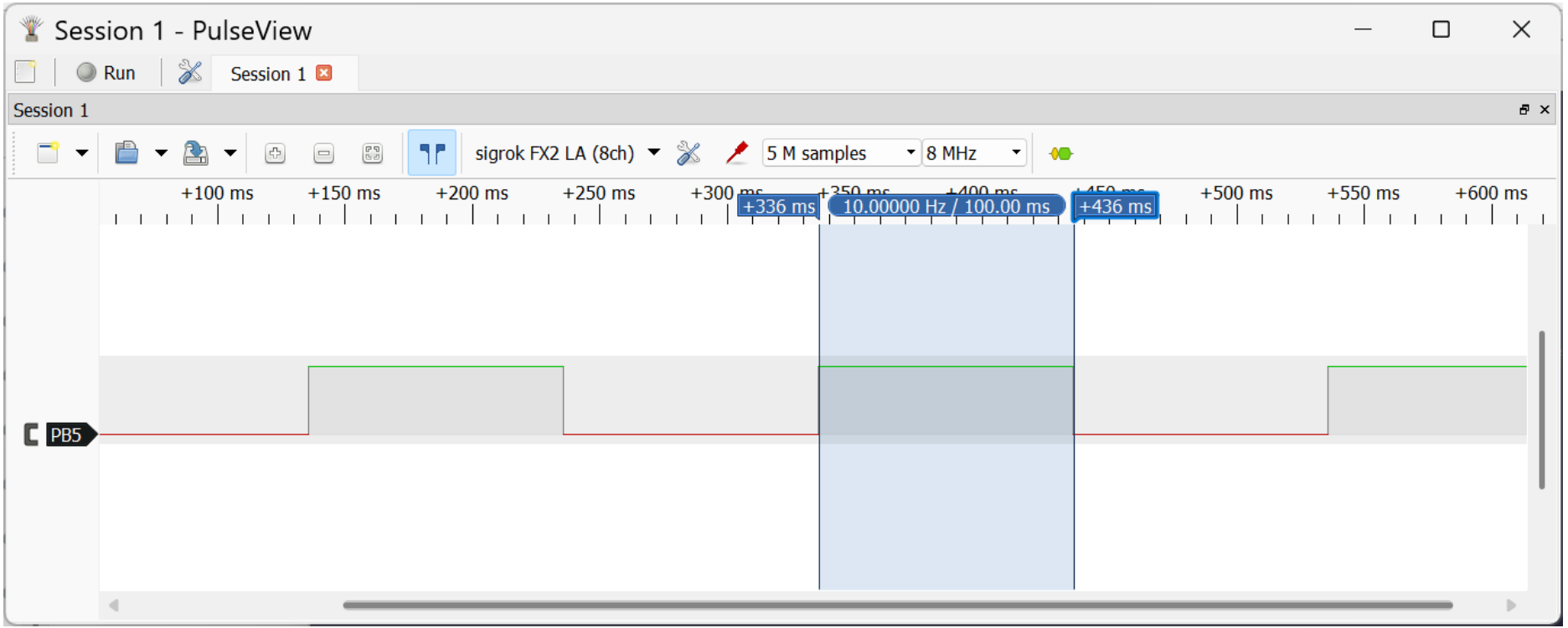
AVR-GCC

```
#define __SFR_OFFSET 0
#include <avr/io.h>

.global main
.text

; Toggle PB5 every ~100 ms @ 16 MHz
main:
    sbi DDRB, DDB5      ; PB5 output

loop:
    sbi PINB, PINB5     ; toggle LED (2 cycles)
    rcall delay_100ms   ; call 100ms delay (3 cycles)
    rjmp loop           ; repeat (2 cycles)

delay_100ms:
    ldi r20, 100        ; outer loop counter

outer_loop:
    ; Load 4000 = 0x0FA0
    ldi r24, 0xA0       ; low byte
    ldi r25, 0x0F       ; high byte

inner_loop:
    sbiw r24, 1         ; subtract 1 (2 cycles)
    brne inner_loop     ; 2 cycles (taken) or 1 (last)
    dec r20             ; 1 cycle
    brne outer_loop     ; 2 cycles (taken) or 1 (last)
    ret                 ; 4 cycles
```

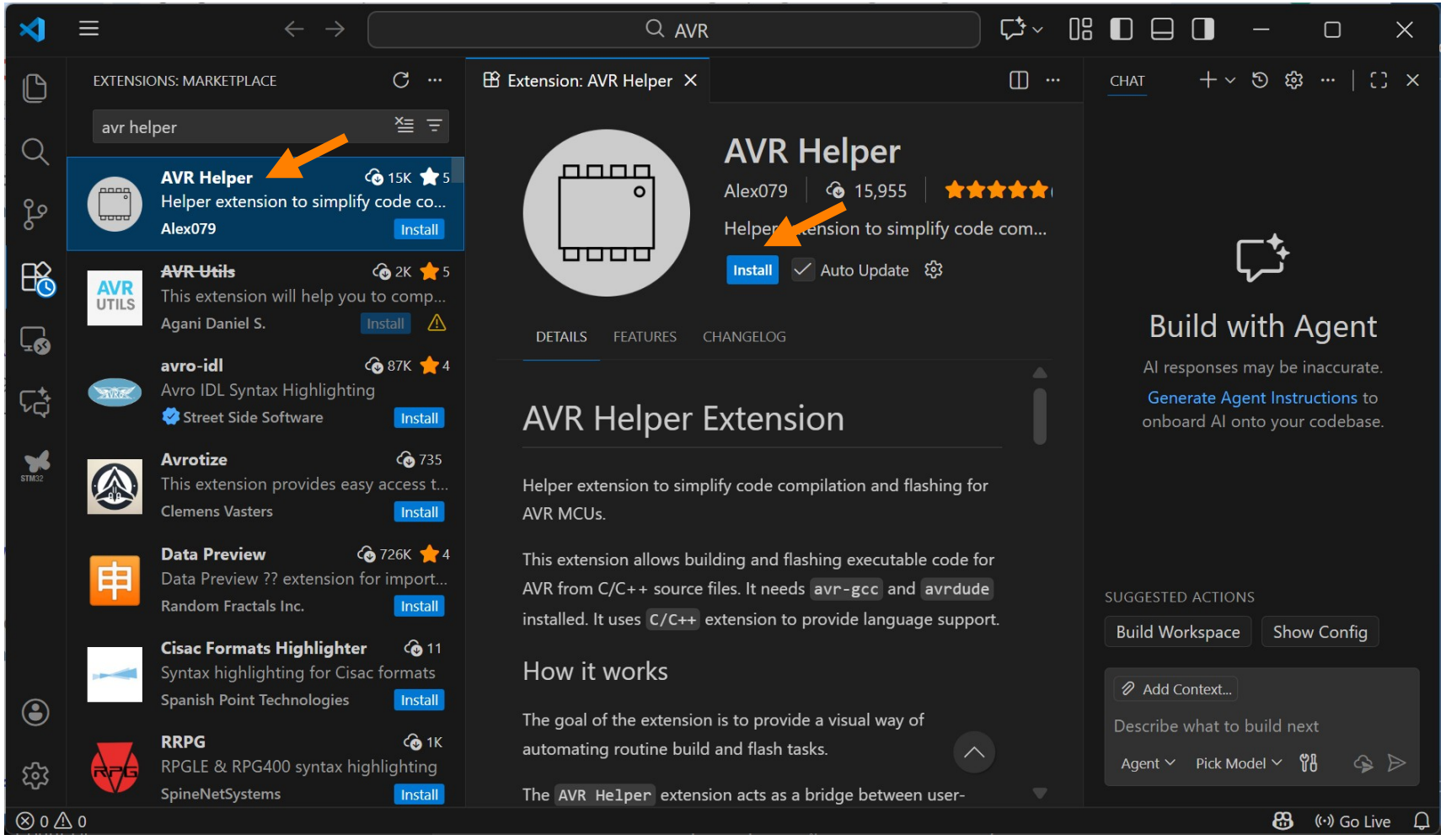
AVRASM2 / AVRA

```
.include "m328pdef.inc" ; Definitions for ATmega328P

.cseg
.org 0x0000
rjmp main

; Toggle PB5 every ~100 ms @ 16 MHz
main:
    sbi DDRB, DDB5          ; PB5 output

loop:
    sbi PINB, PINB5         ; toggle LED
    rcall delay_100ms       ; ~100 ms delay
    rjmp loop

delay_100ms:
    ldi r20, 100            ; outer loop count

outer_loop:
    ; load inner loop counter = 4000 = 0x0FA0
    ldi r24, 0xA0           ; low byte
    ldi r25, 0x0F           ; high byte

inner_loop:
    sbiw r24, 1             ; subtract 1
    brne inner_loop         ; repeat inner loop
    dec r20                 ; decrement outer loop
    brne outer_loop         ; repeat outer loop
    ret
```

Signal Measurement with a USB Logic Analyzer



VS Code IDE + AVR Helper (Extension): AVR C Programming



VS Code IDE + AVR Helper (Extension): AVR C Programming

1) Install the **toolchain** (avr-gcc, avr-libc, avrdude):
   => use **AVR-GCC for Windows**.

2) Install **VS Code + AVR Helper**

3) Create a new AVR Project in VS Code:

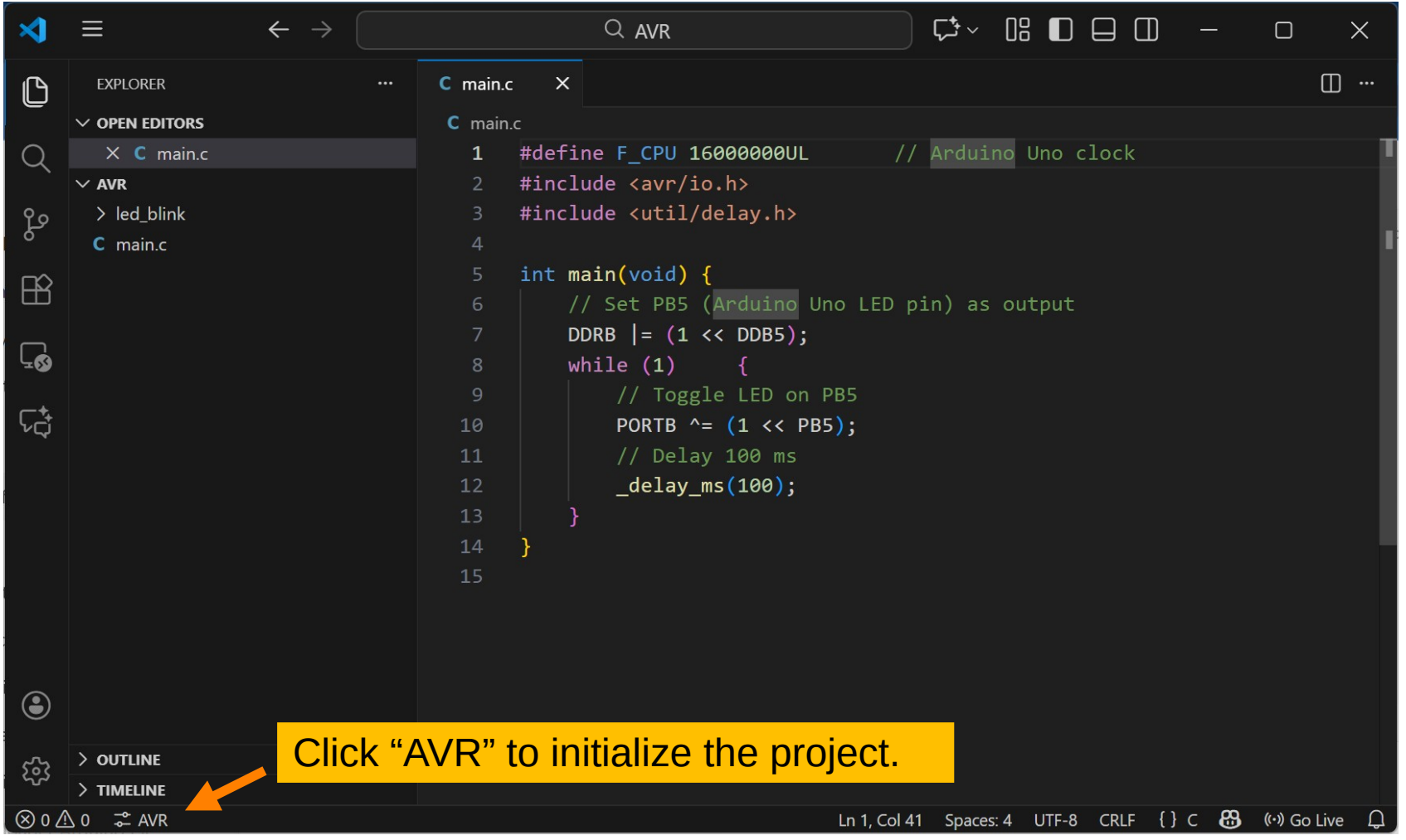
- Open folder for your project
- Create your AVR source file (e.g. main.c)

4) Press Ctrl+Shift+P and type "**AVR: Initialize Project**"

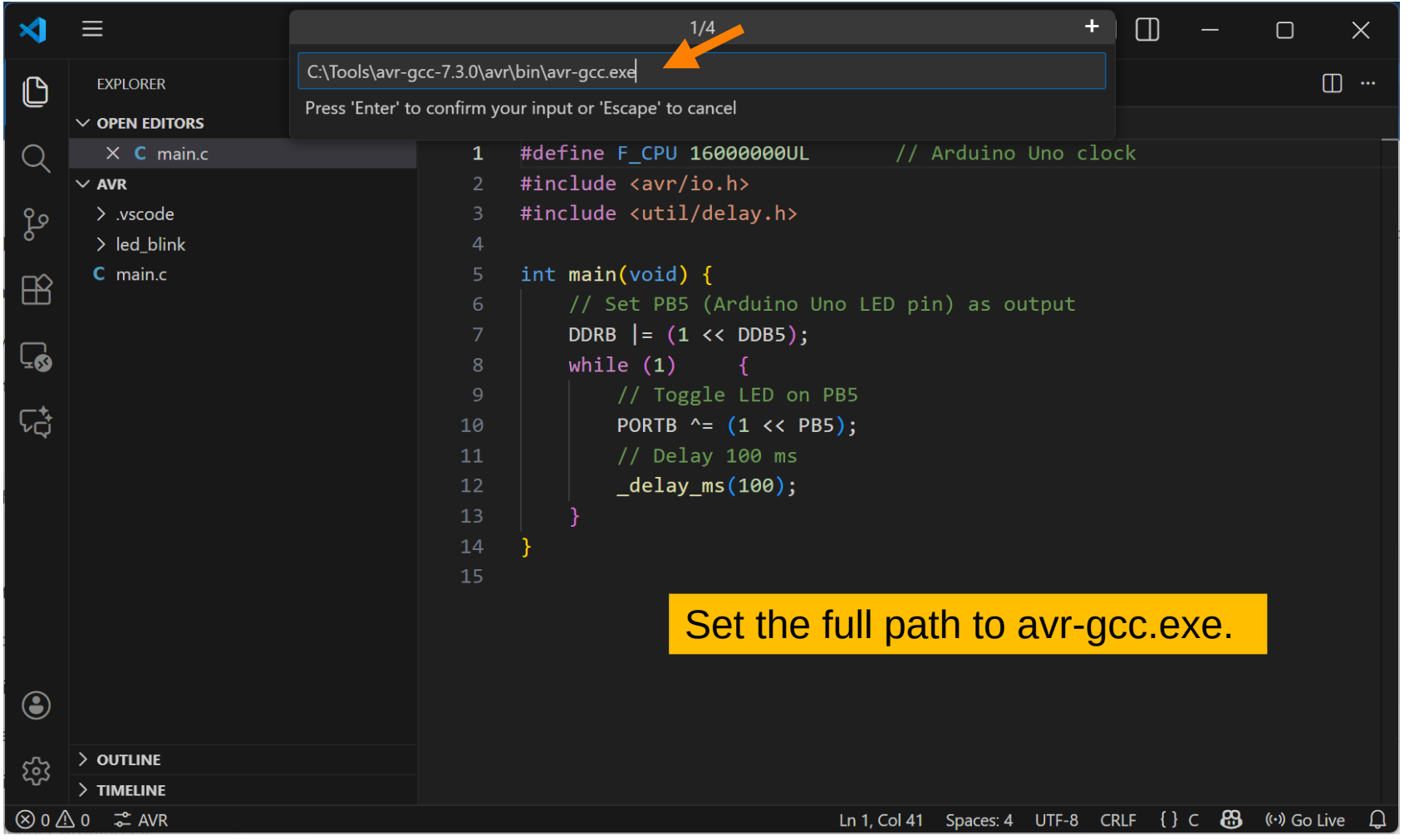
- Set full path to avr-gcc.
- Set full path to avrdude and avrdude.conf
- Select your MCU → ATmega328P
- Set CPU frequency → 16000000
- Select programmer → e.g. Arduino bootloader, Arduino as ISP or USBasp
- Configure Upload Port (port name and baudrate)

5) Edit source code, build and flash

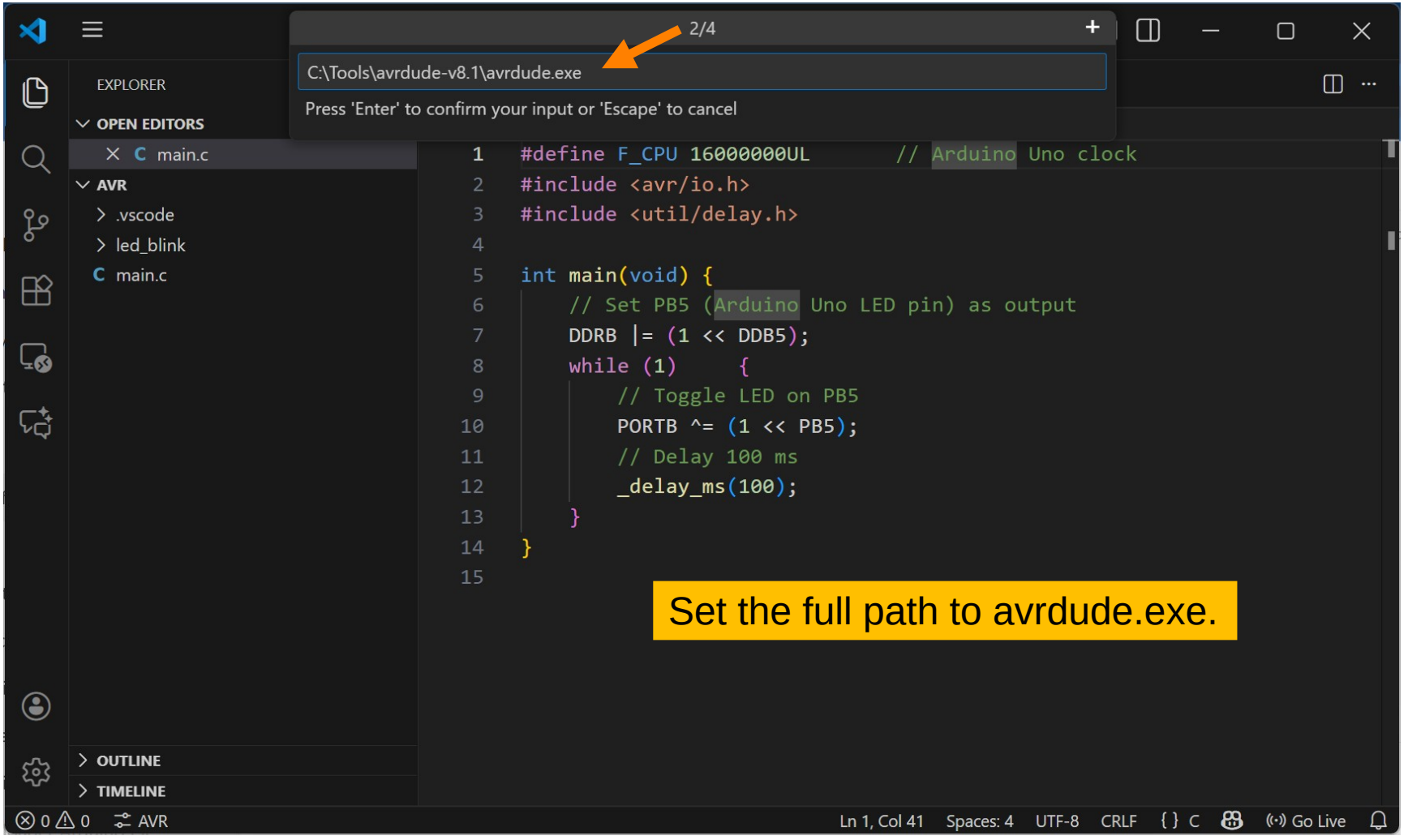
VS Code IDE + AVR Helper (Extension)



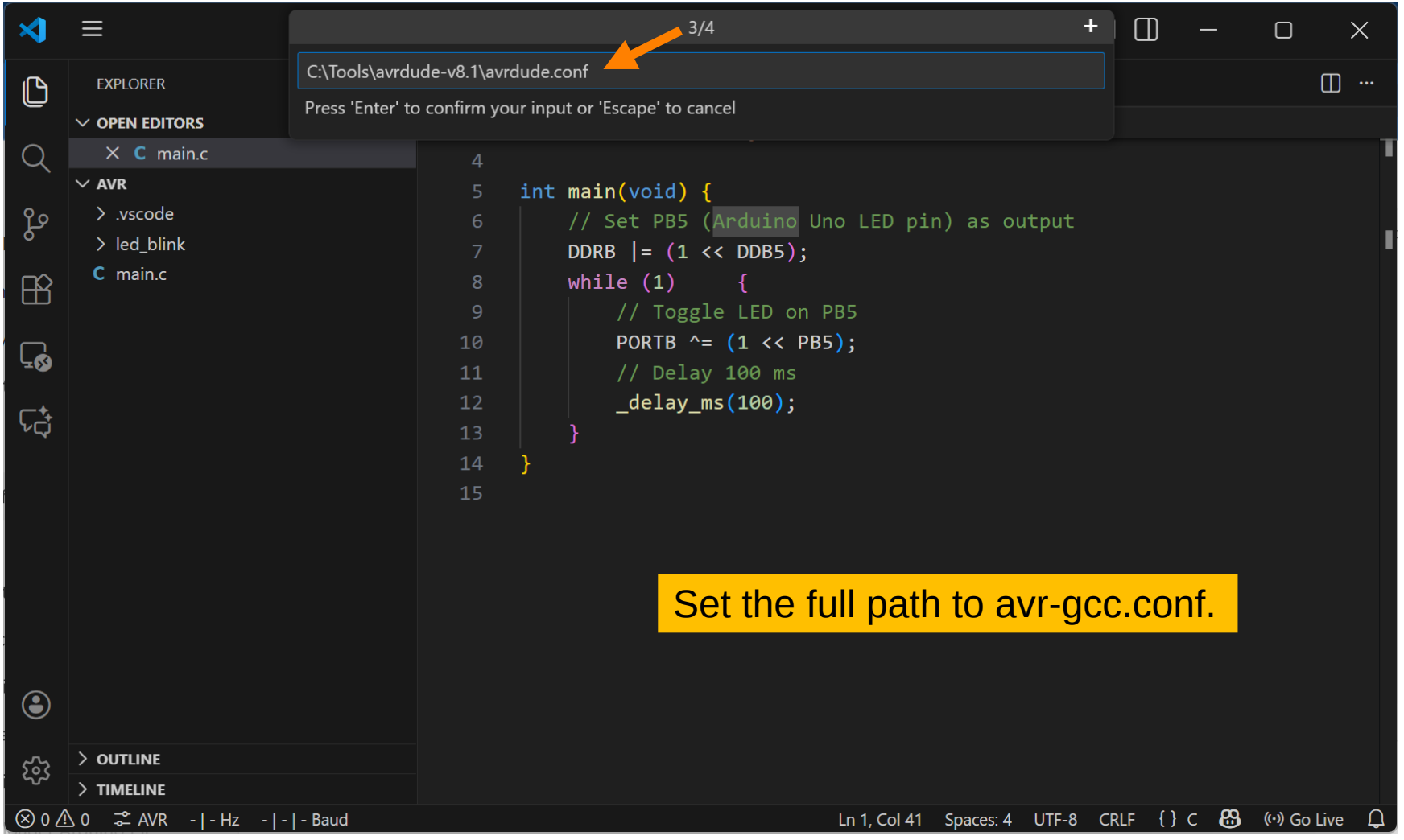
VS Code IDE + AVR Helper (Extension)



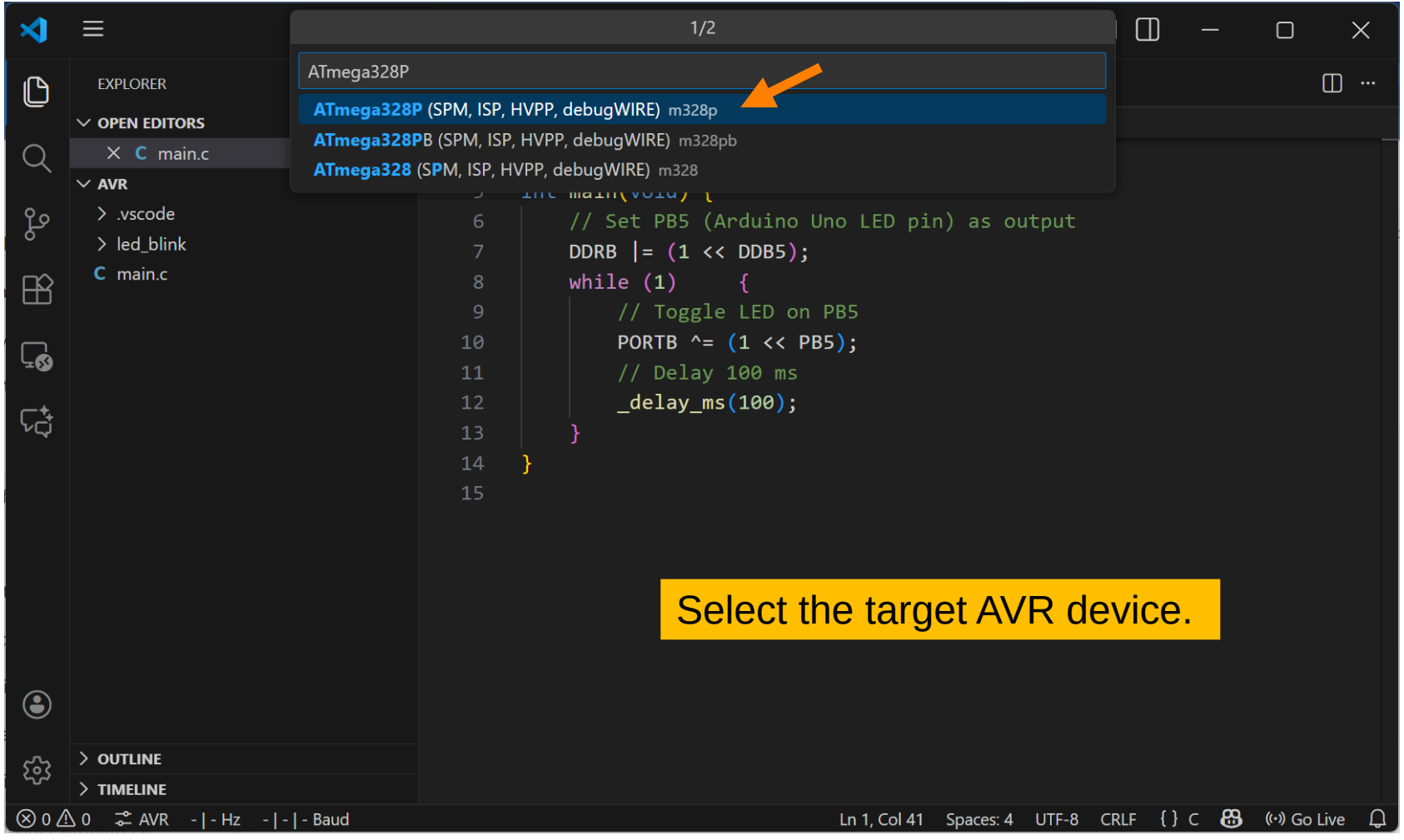
VS Code IDE + AVR Helper (Extension)



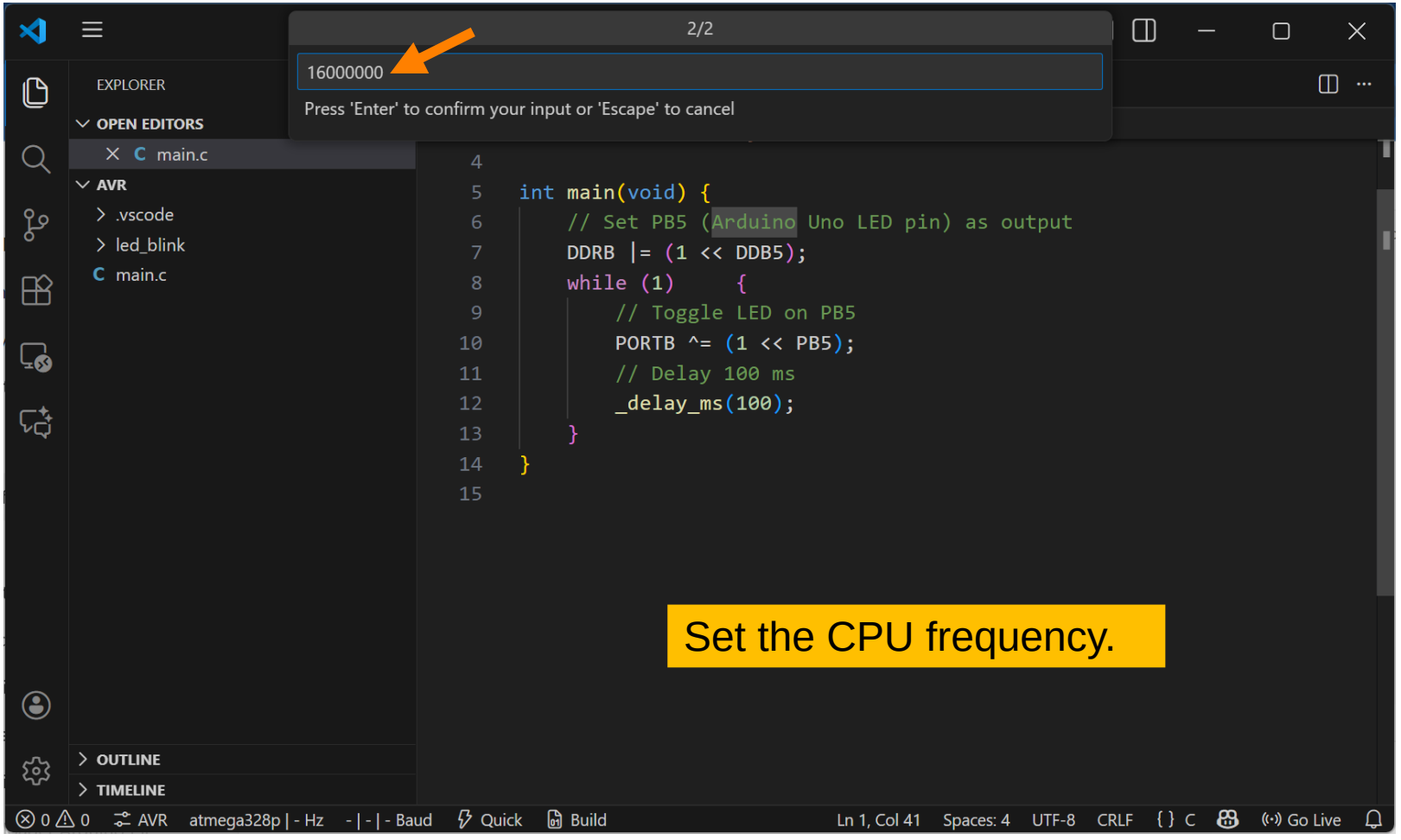
VS Code IDE + AVR Helper (Extension)



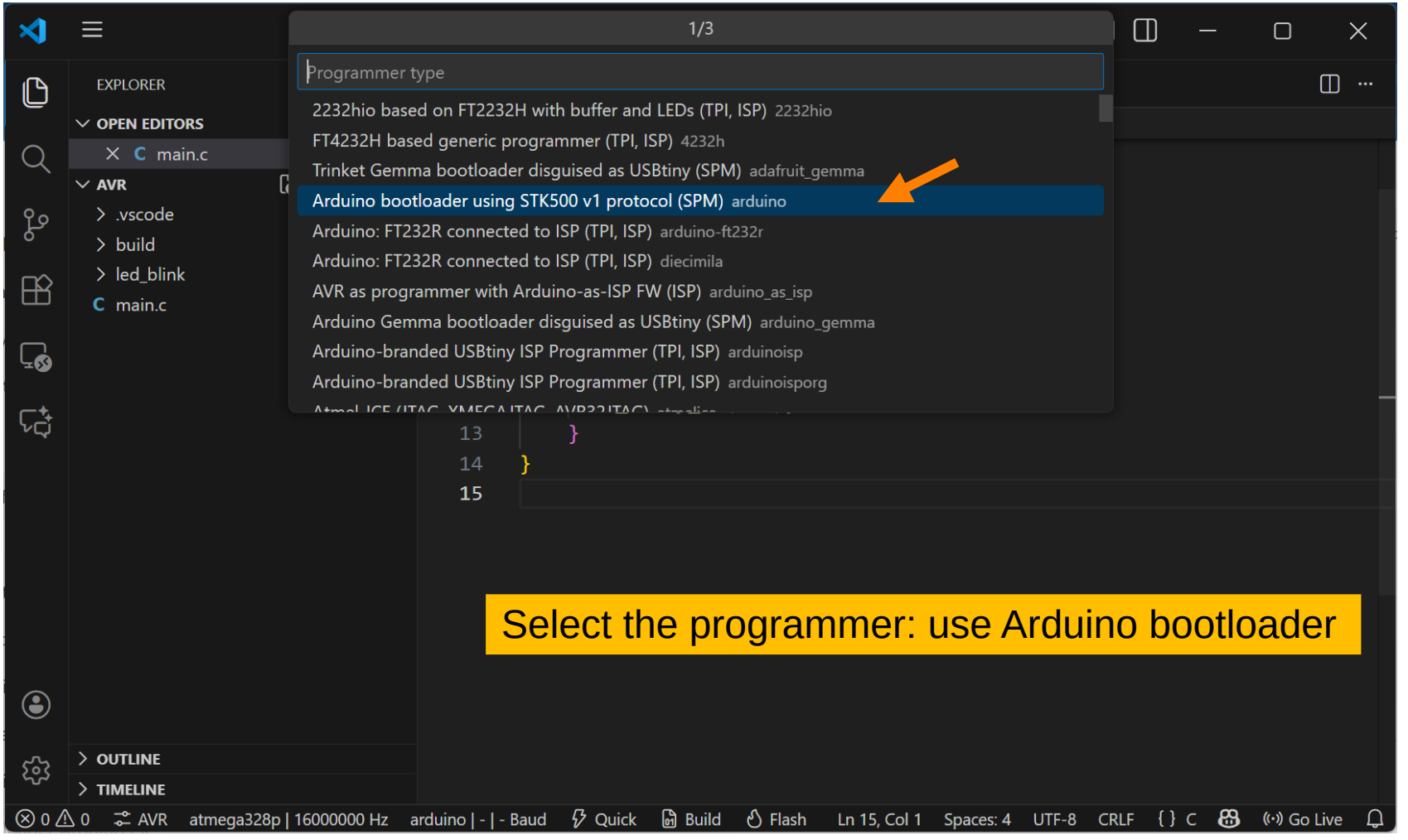
VS Code IDE + AVR Helper (Extension)



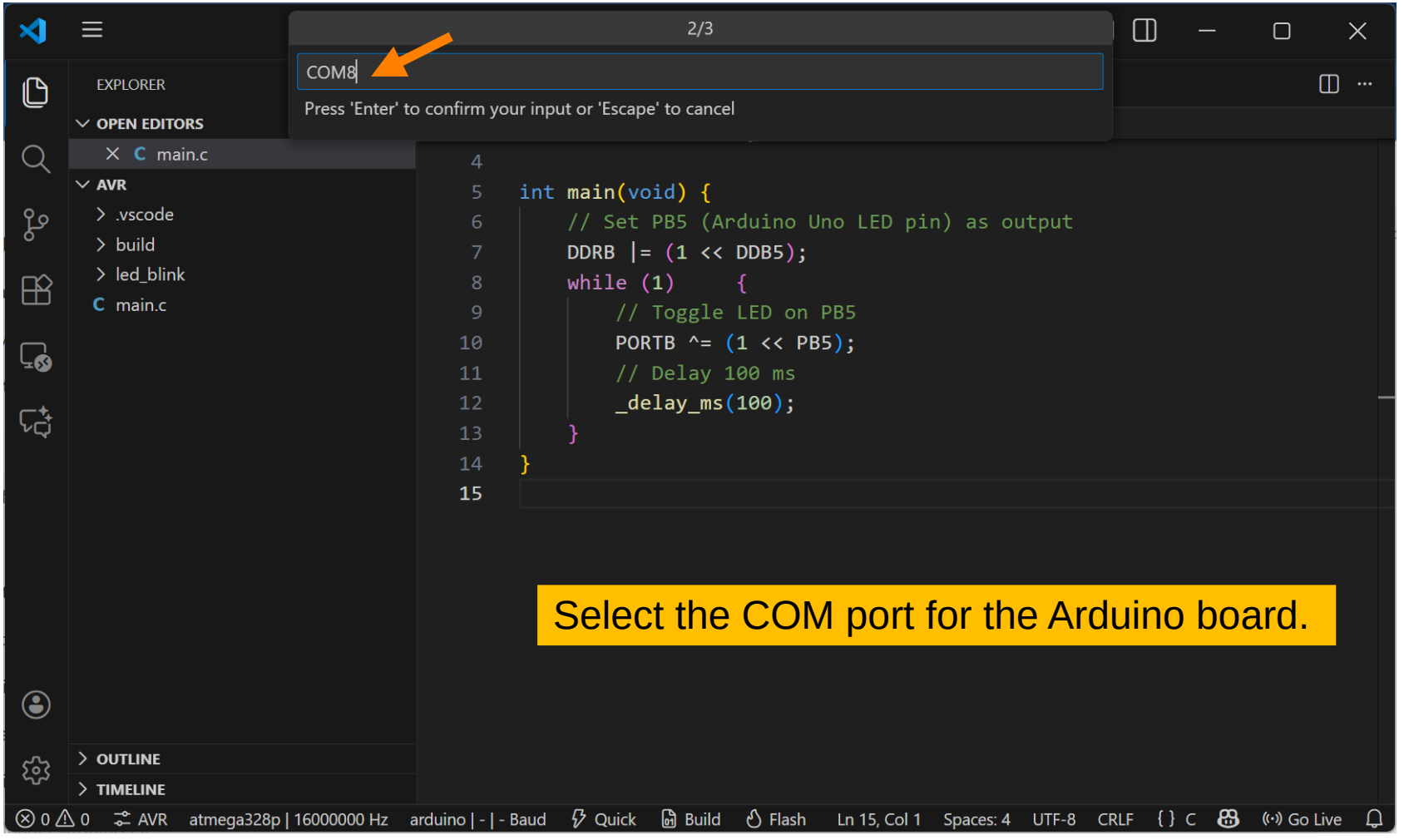
VS Code IDE + AVR Helper (Extension)



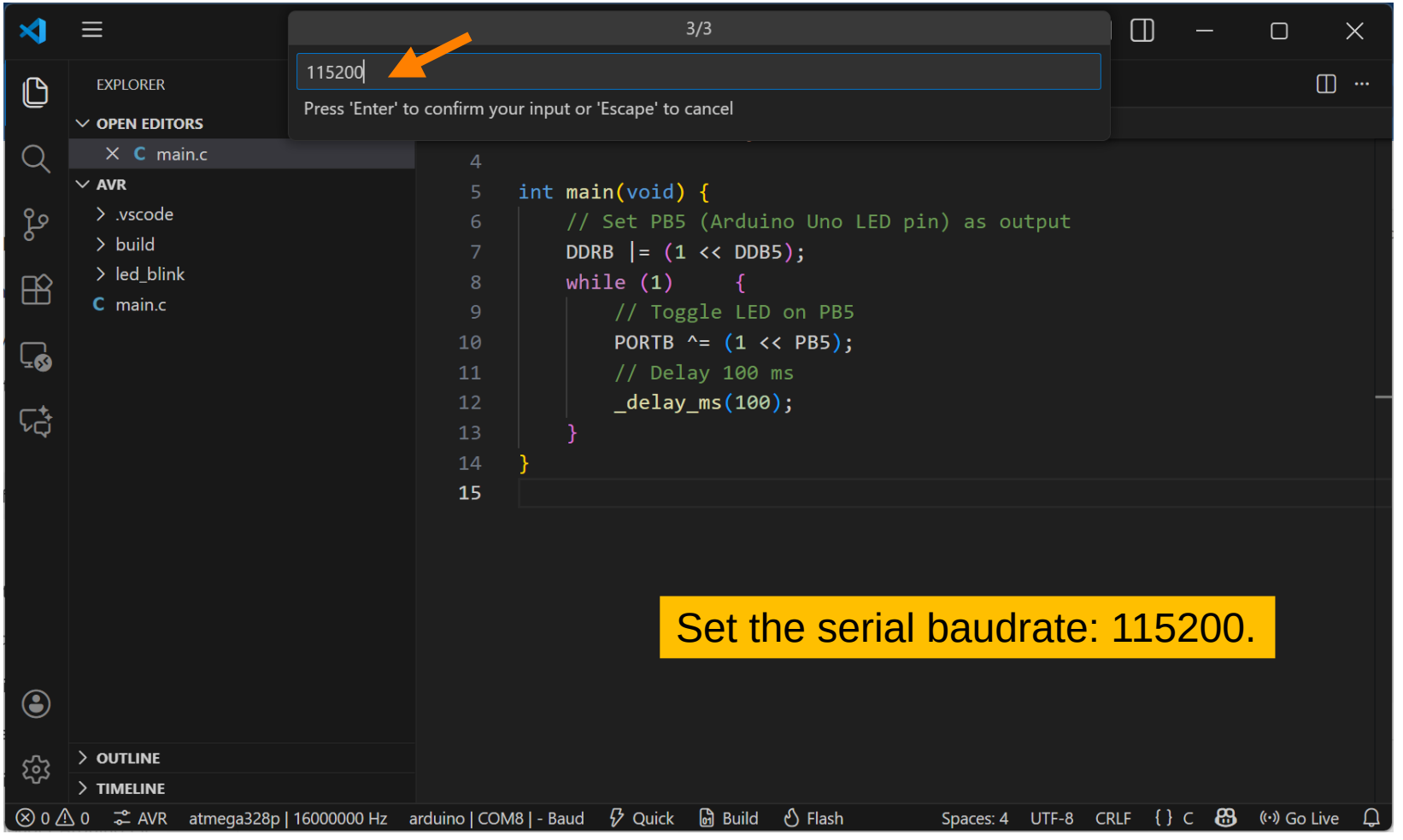
VS Code IDE + AVR Helper (Extension)



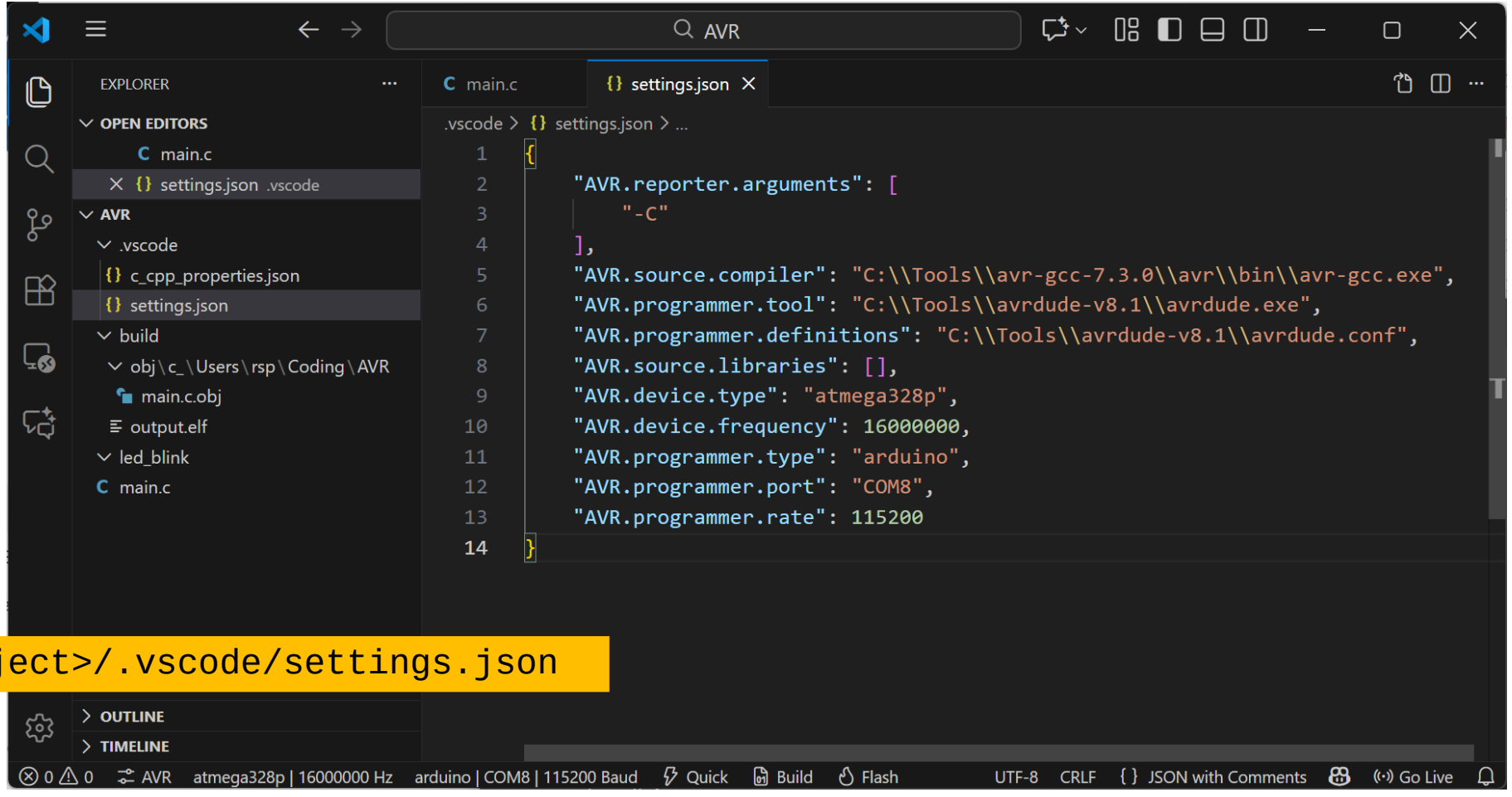
VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)



<project>/.vscode/settings.json

VS Code IDE + AVR Helper (Extension)