

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.

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



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.

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

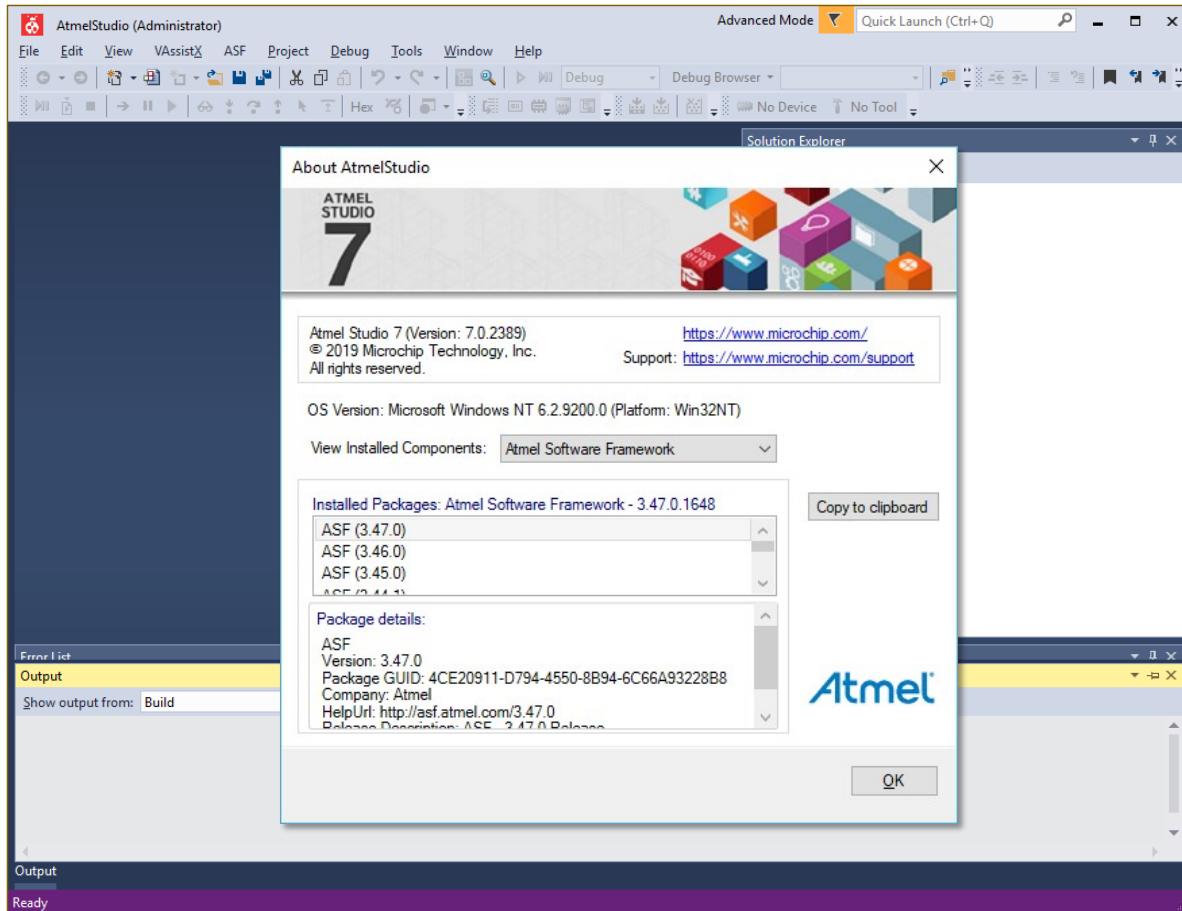
List 5

- 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).

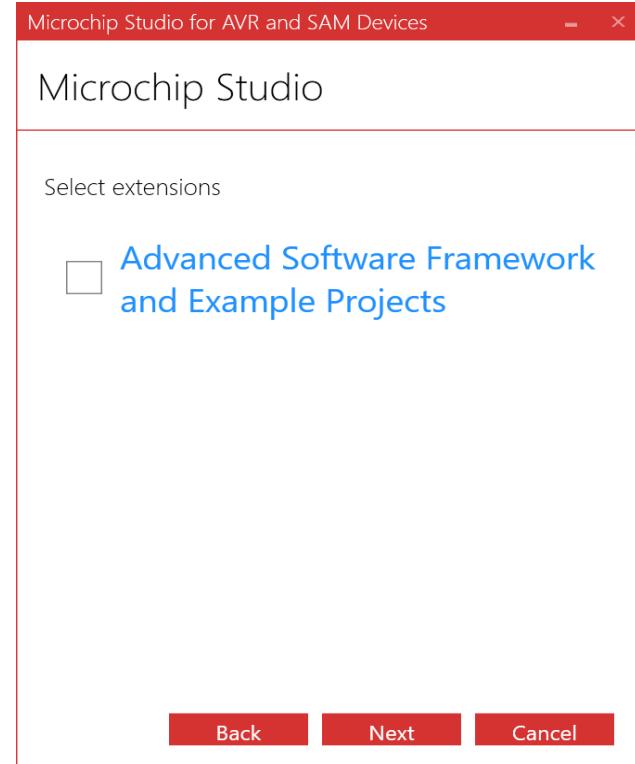
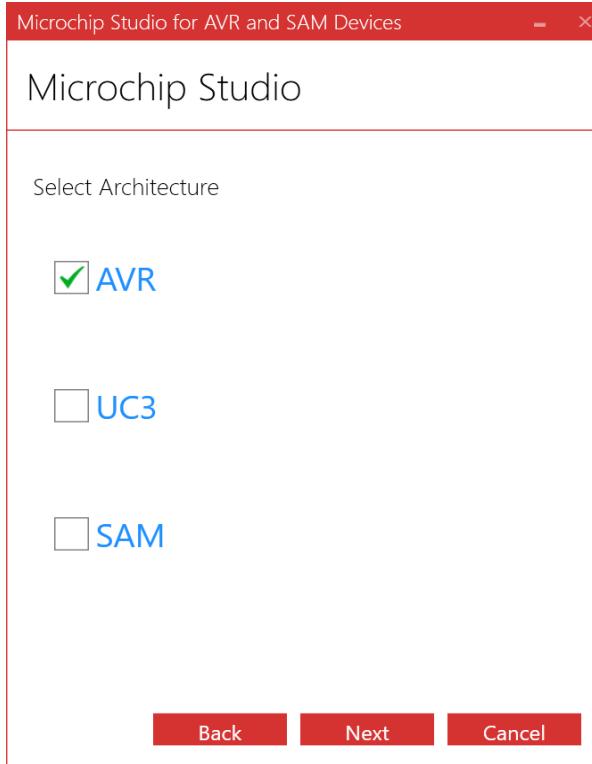
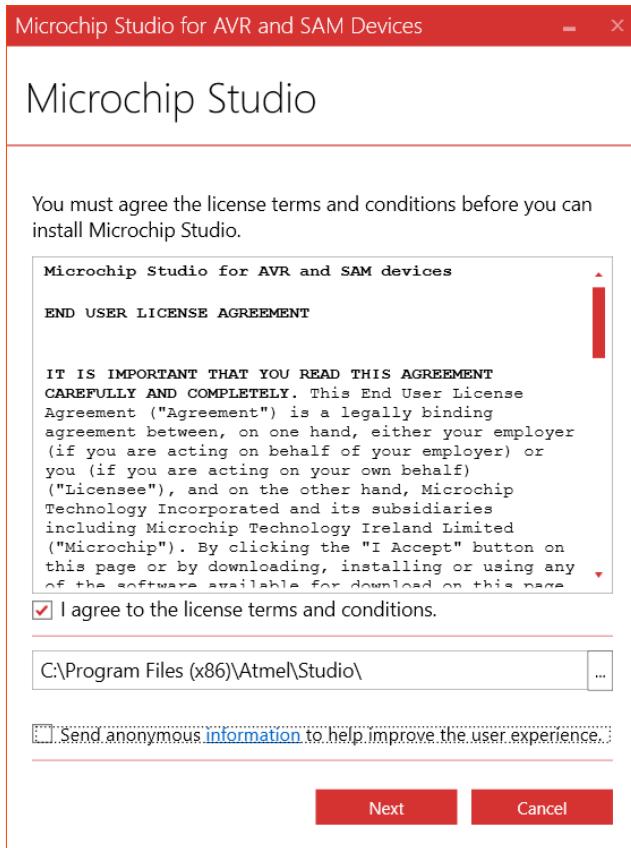
- <https://www.microchip.com/en-us/tools-resources/develop/microchip-studio#Downloads>
- <https://www.microchip.com/en-us/tools-resources/archives/avr-sam-mcus>

Microchip Studio 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

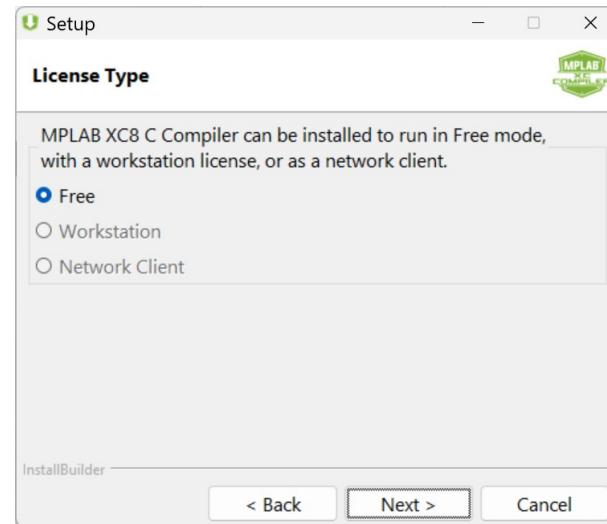
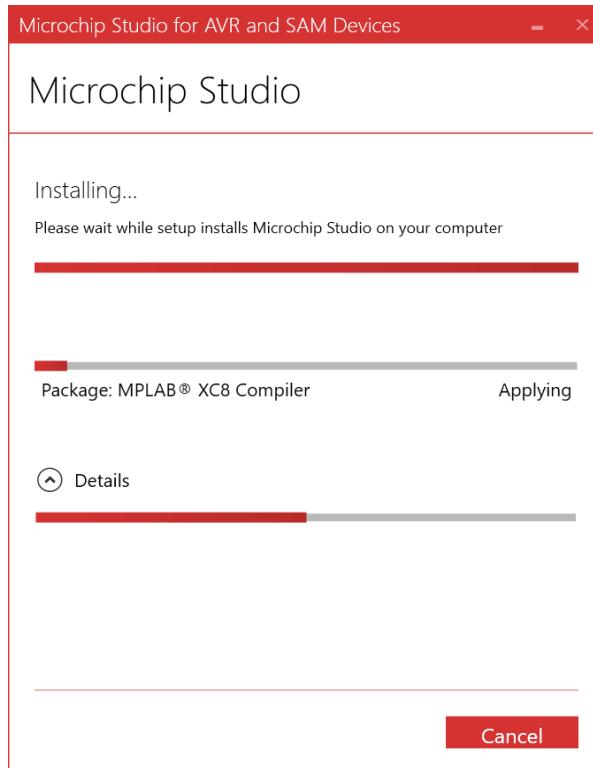
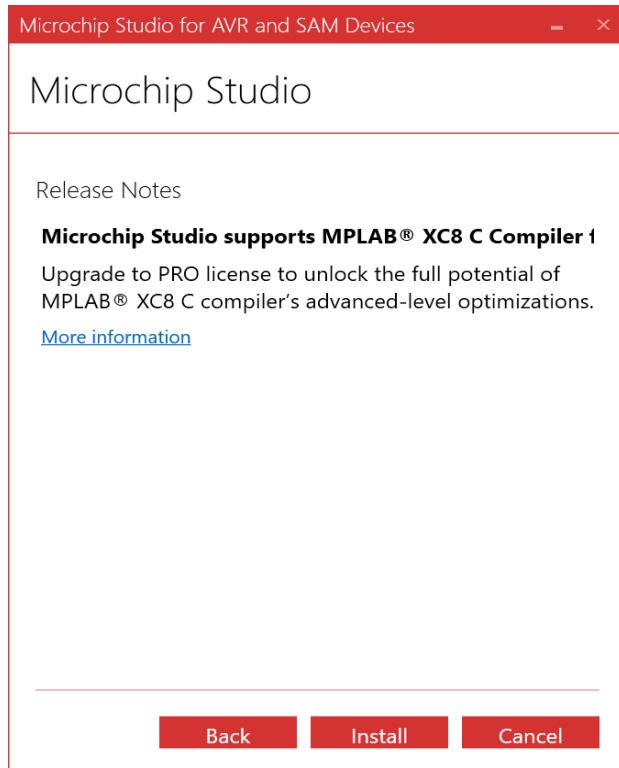
Download: <https://www.microchip.com/en-us/tools-resources/develop/microchip-studio>

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.

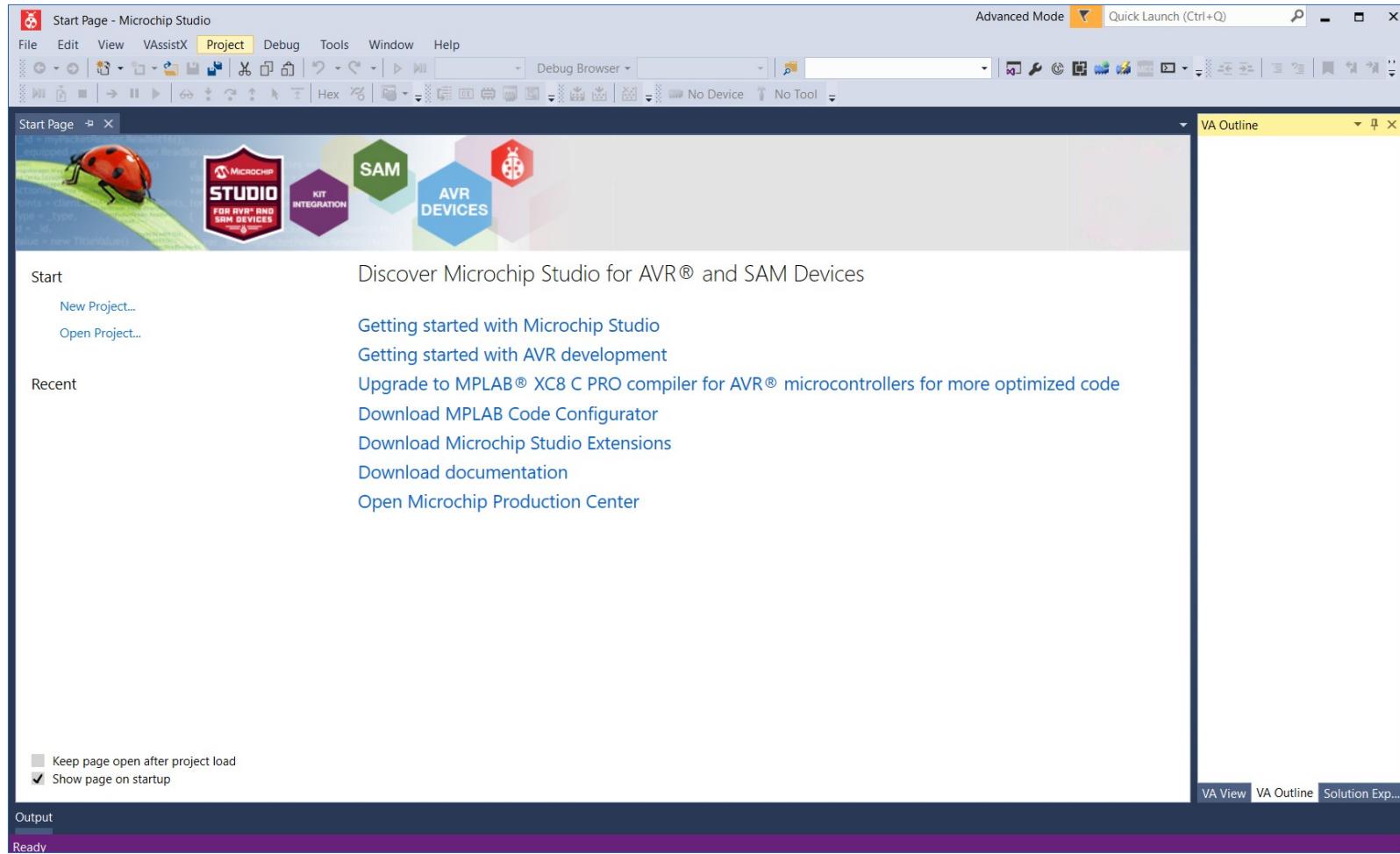
Microchip Studio v7.0.2594 (2022)



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

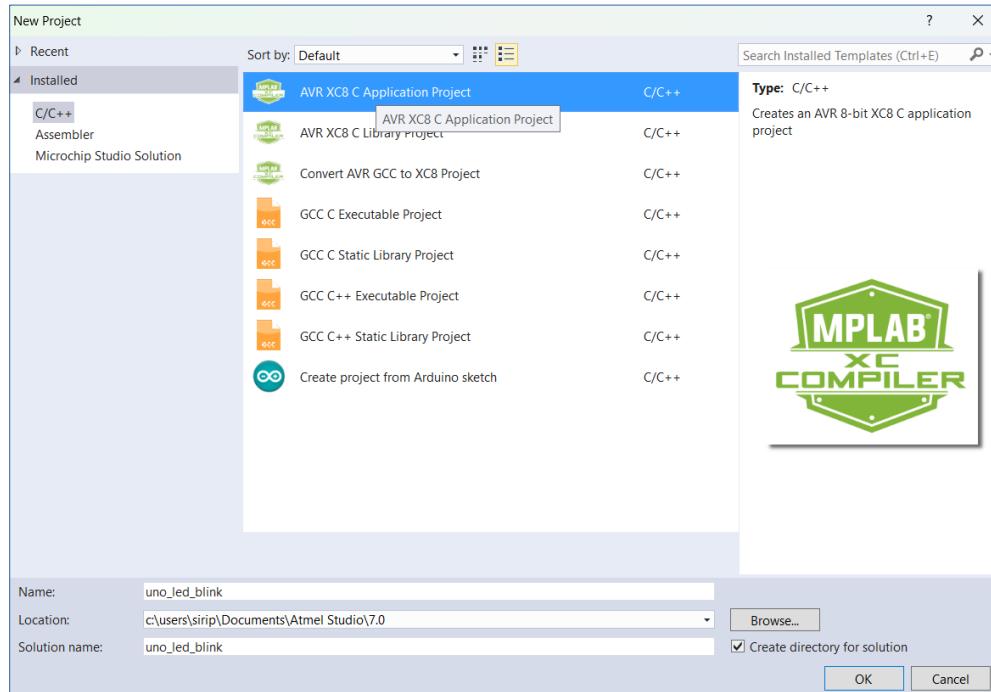
Note: Microsoft Visual Studio 2015 Shell (isolated) will be installed.

Microchip Studio v7.0.2594 (2022)

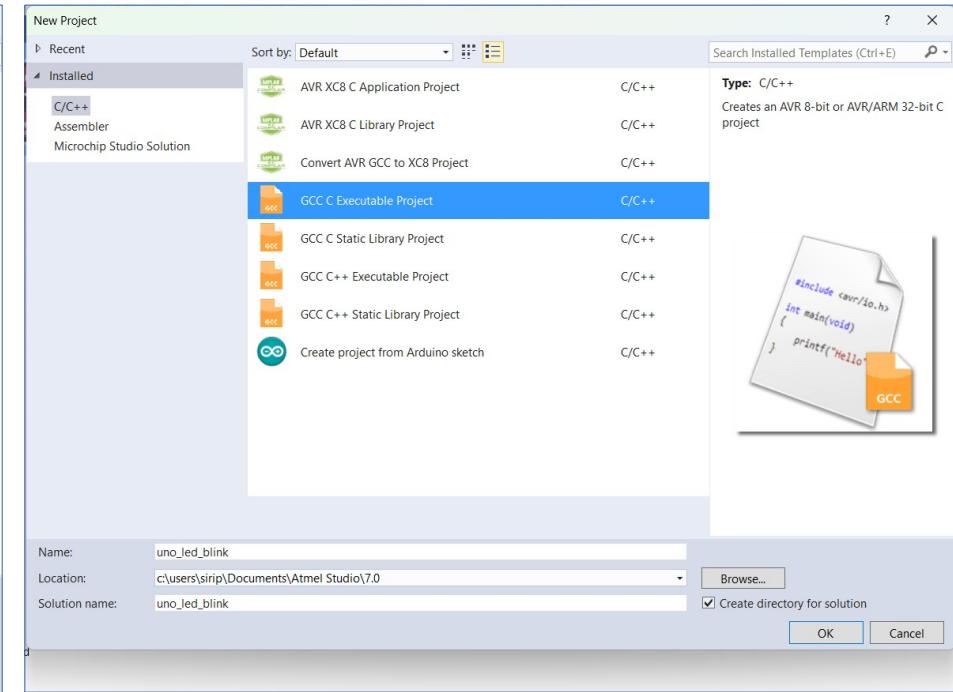


Microchip Studio v7.0.2594 (2022)

New Project Creation

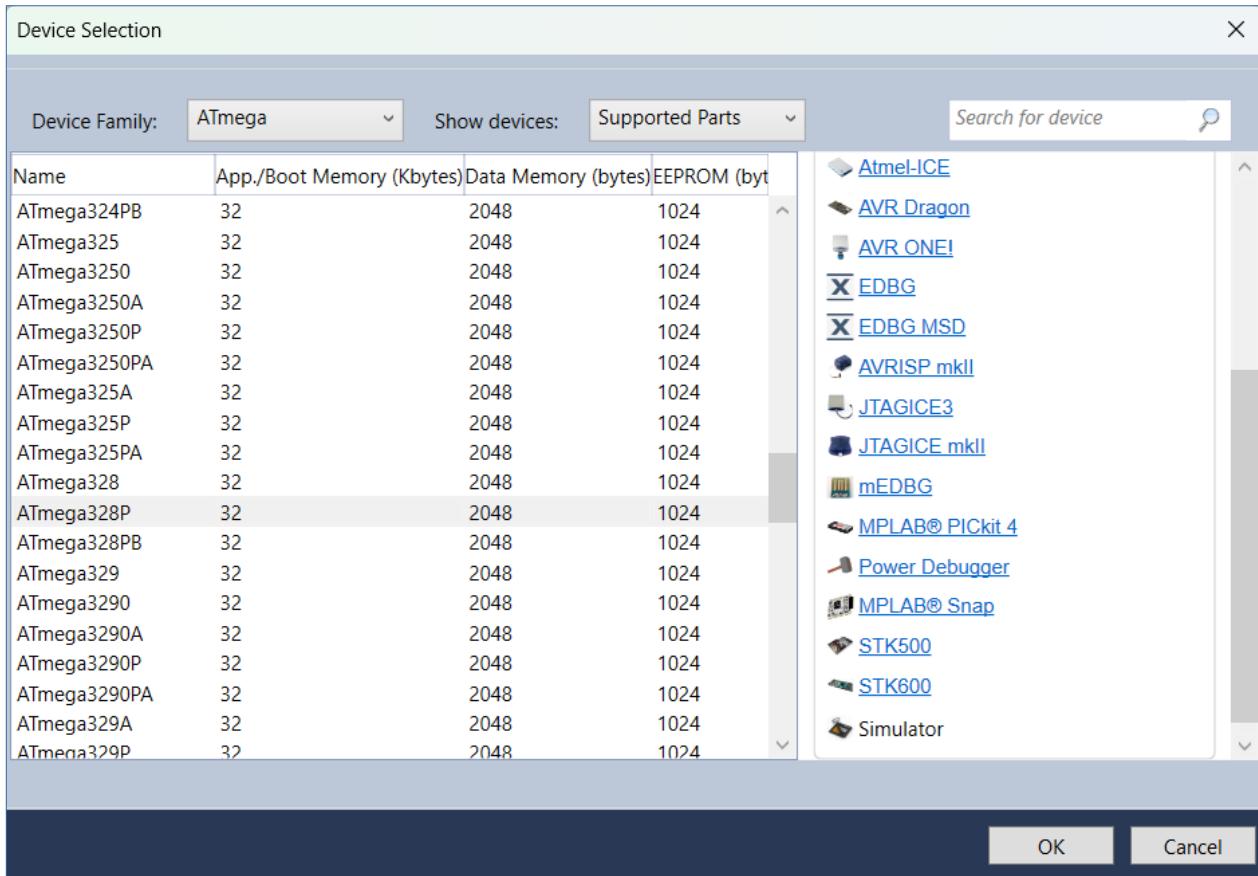


**AVR XC8 C Application Project
with MC8 Compiler toolchain**



**C Executable Project with
AVR-GCC Compiler toolchain**

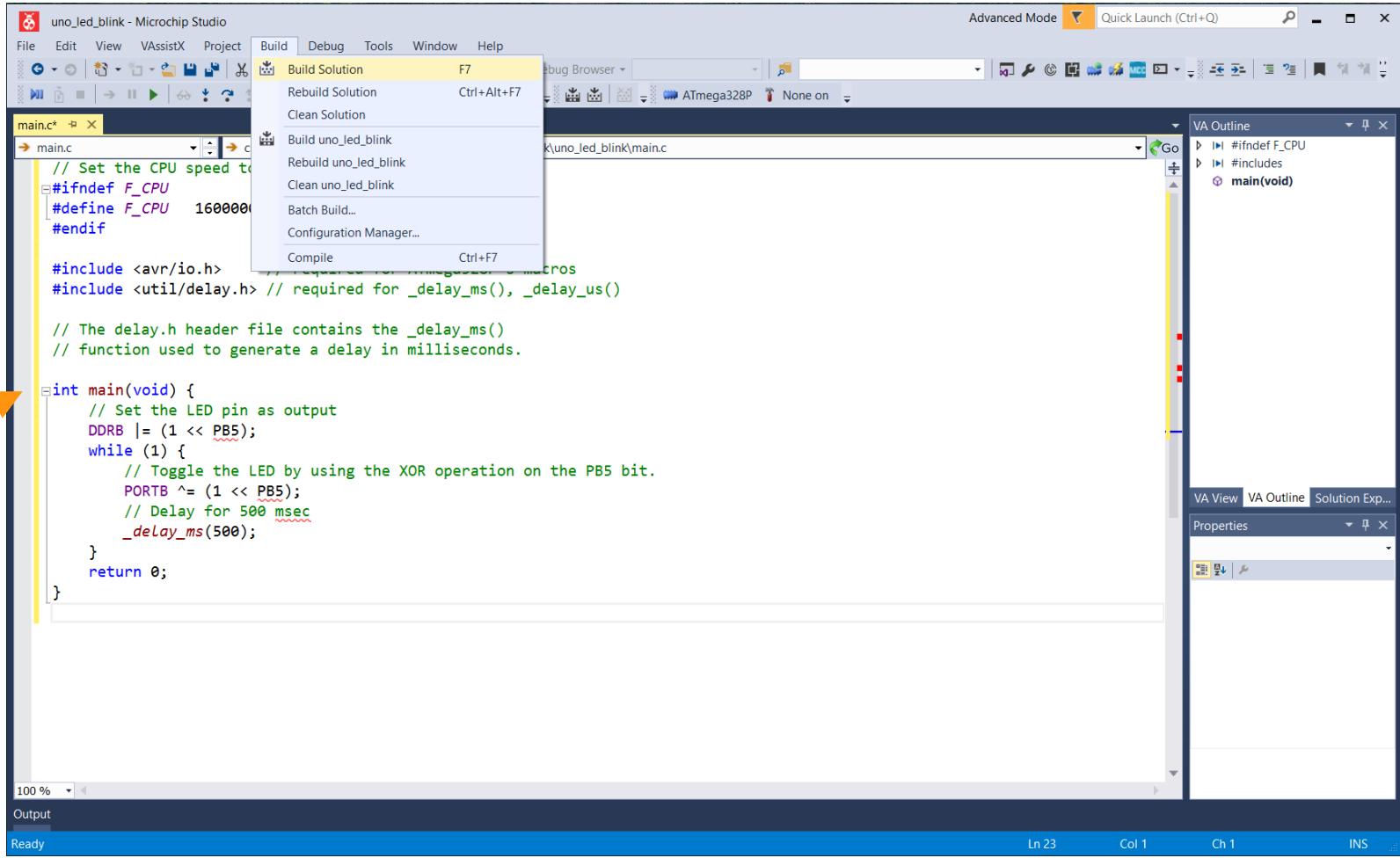
Microchip Studio v7.0.2594 (2022)



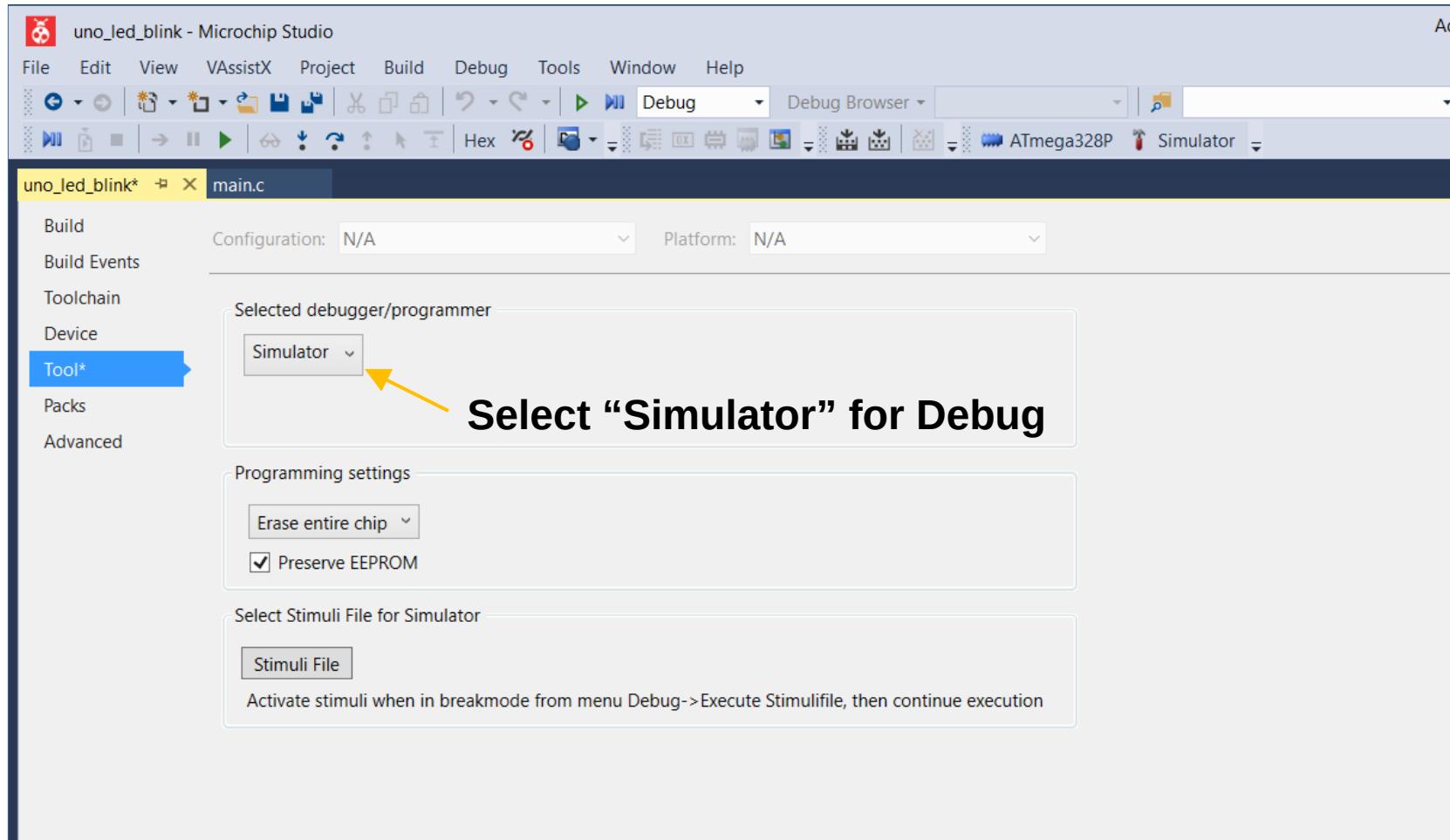
Target Device Selection: **ATmega328P**

Microchip Studio v7.0.2594 (2022)

Sample
C Code
(main.c)

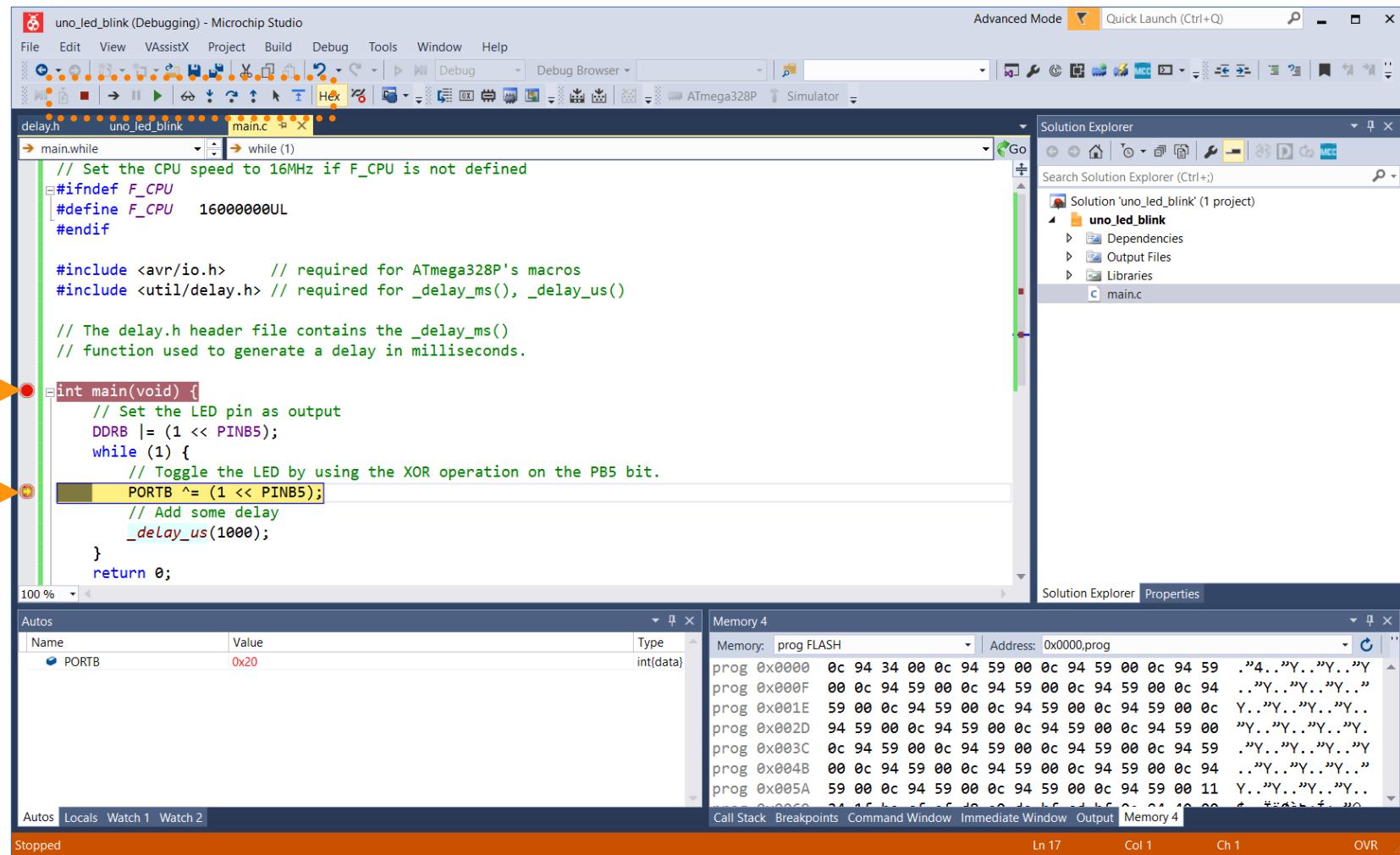


Microchip Studio v7.0.2594 (2022)

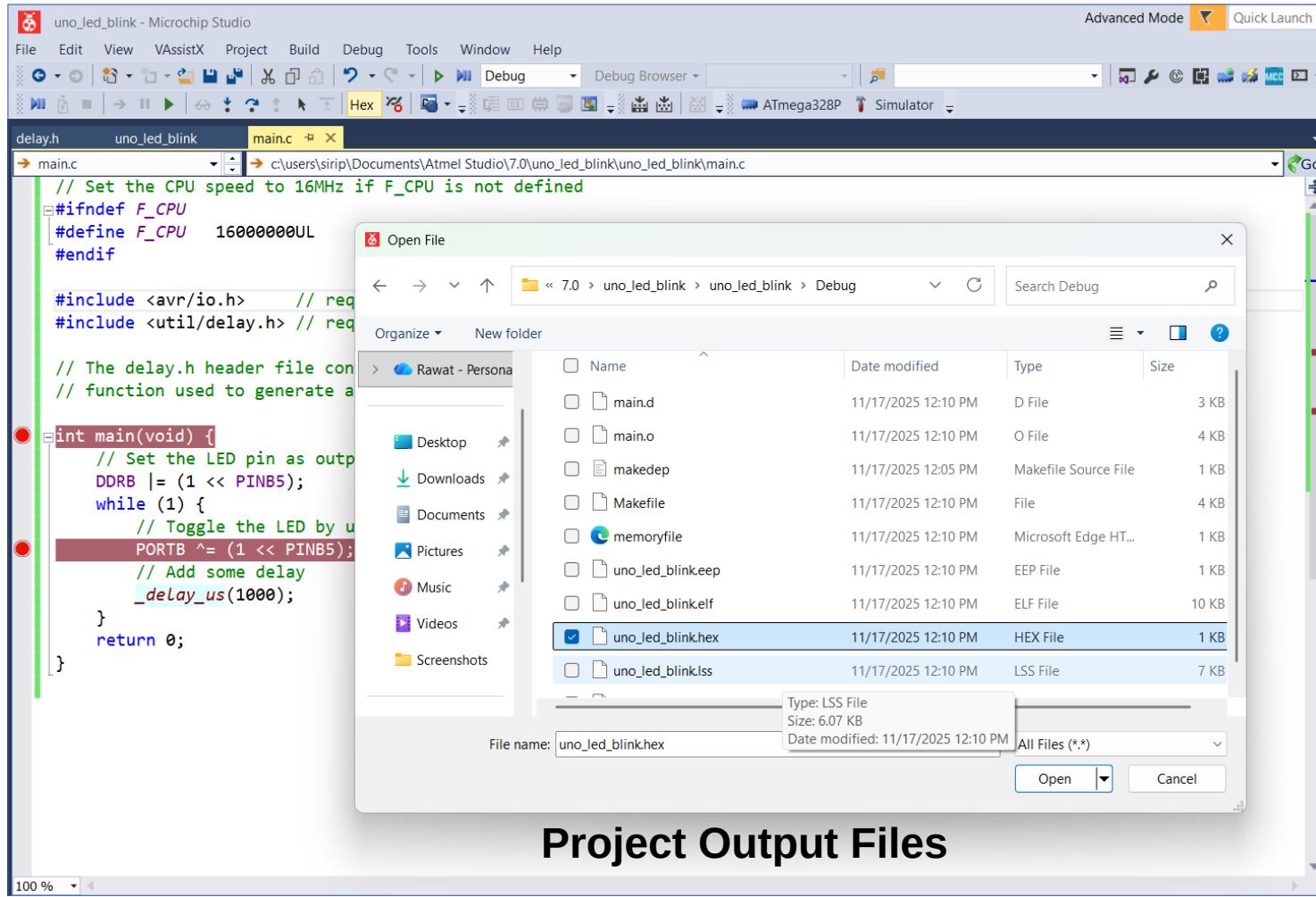


Microchip Studio v7.0.2594 (2022)

Debug Control



Microchip Studio v7.0.2594 (2022)



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;
}
```

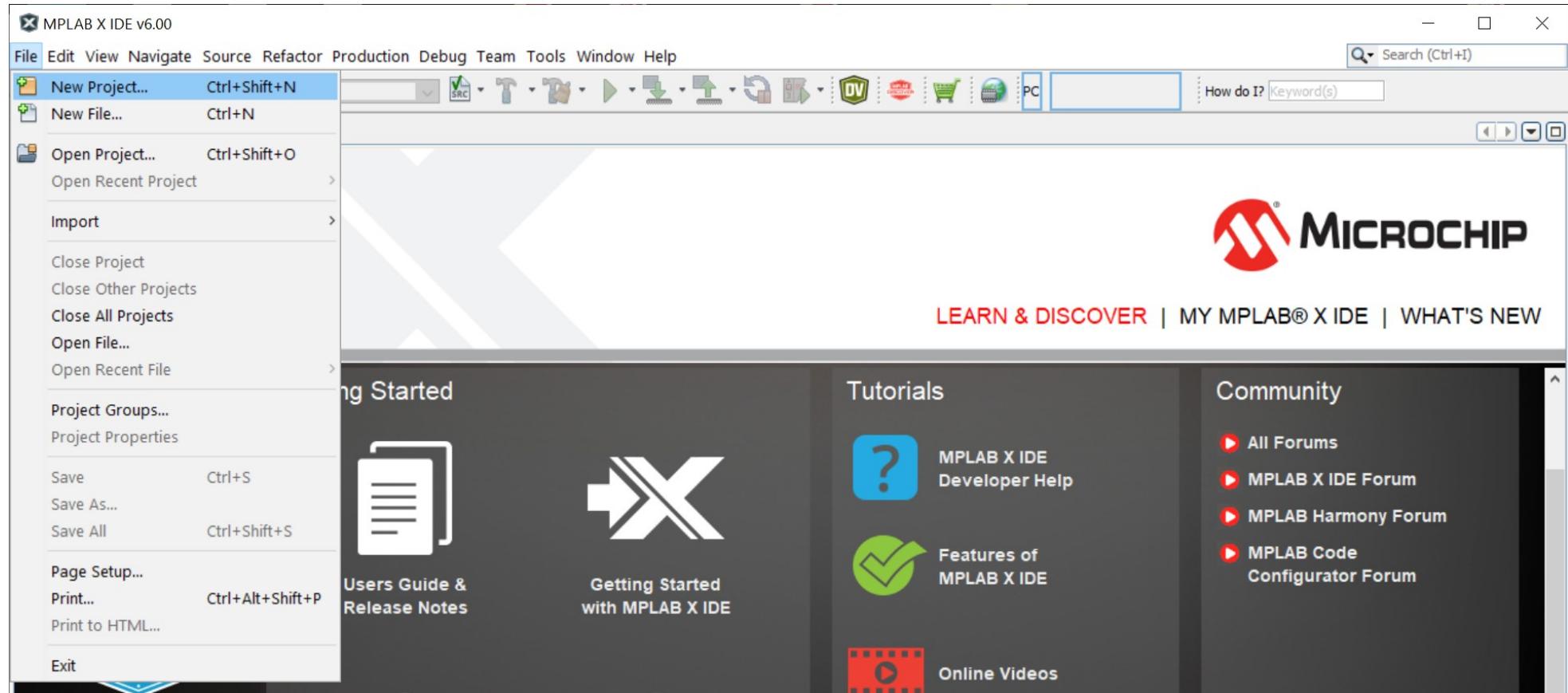
API Reference: <https://www.nongnu.org/avr-libc/>

DDRB = The Port B Data Direction Register

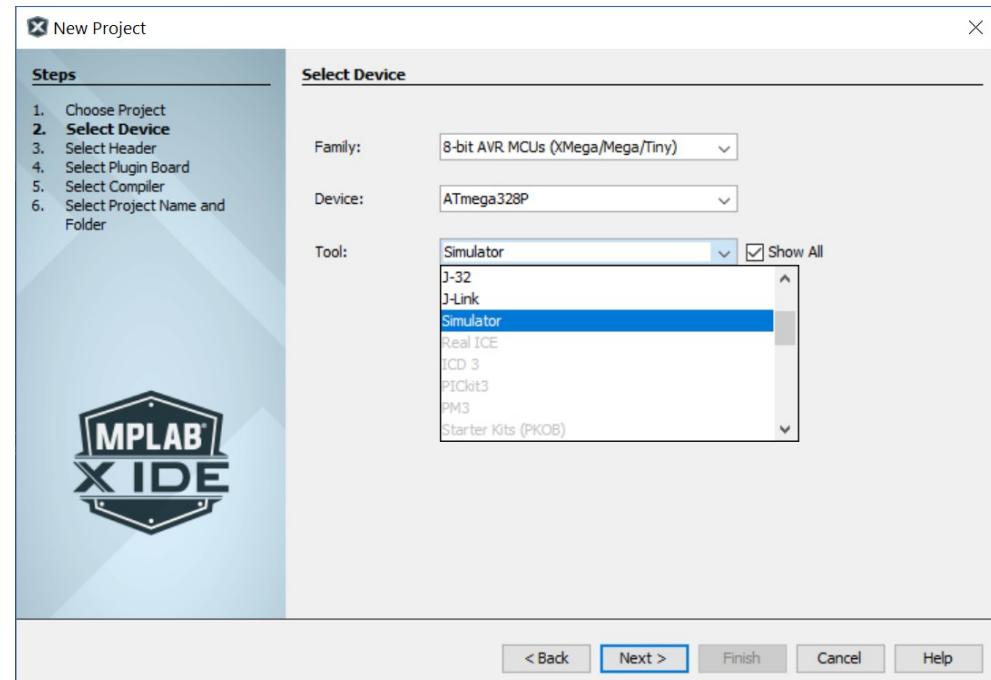
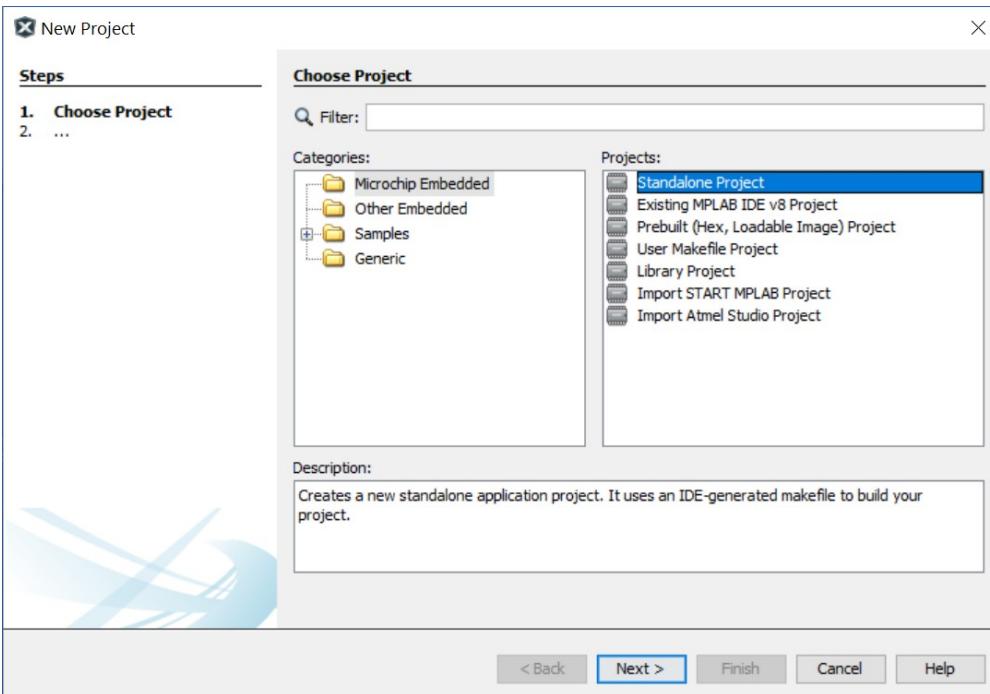
PORTB = The Port B Data Register

PINB = The Port B Input Register

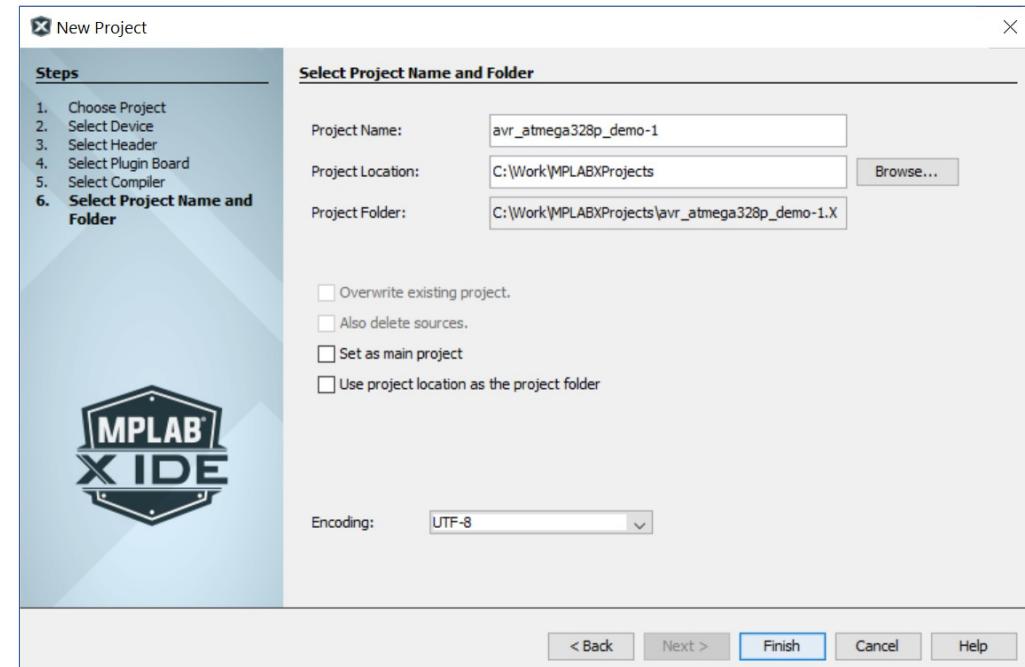
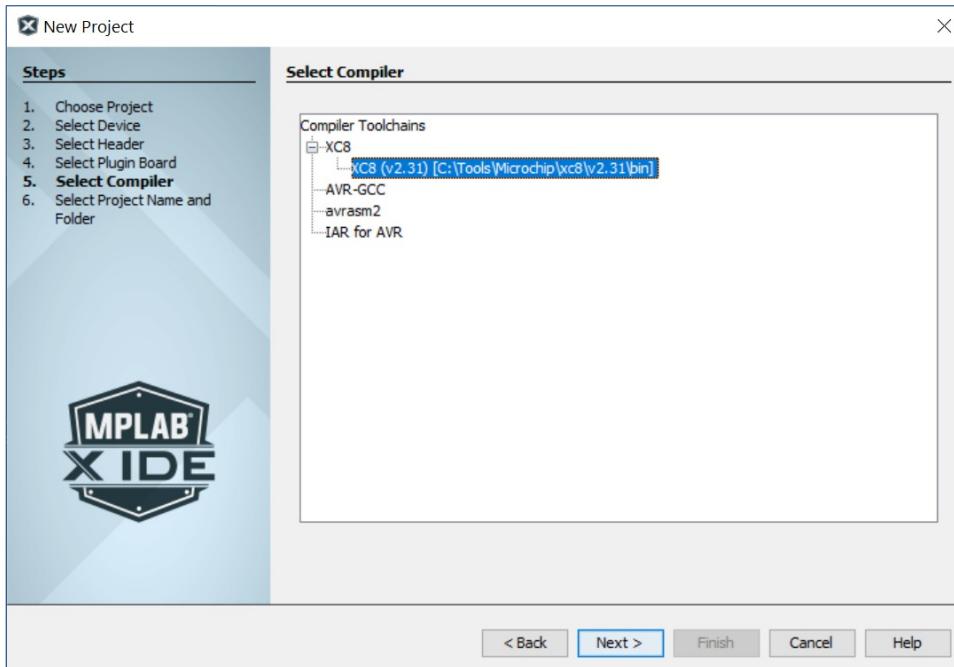
Microchip MPLABX IDE



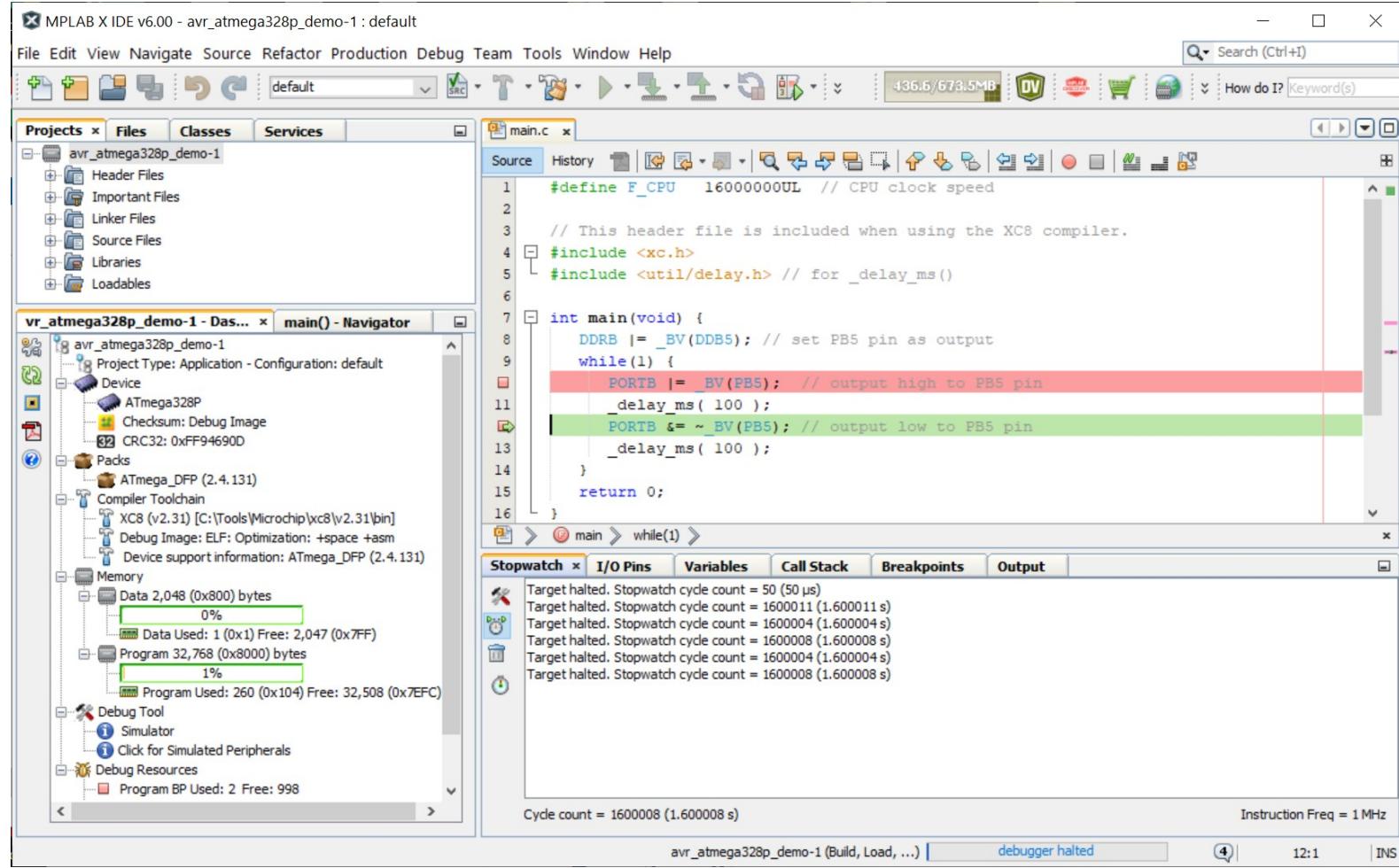
Microchip MPLABX IDE



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.

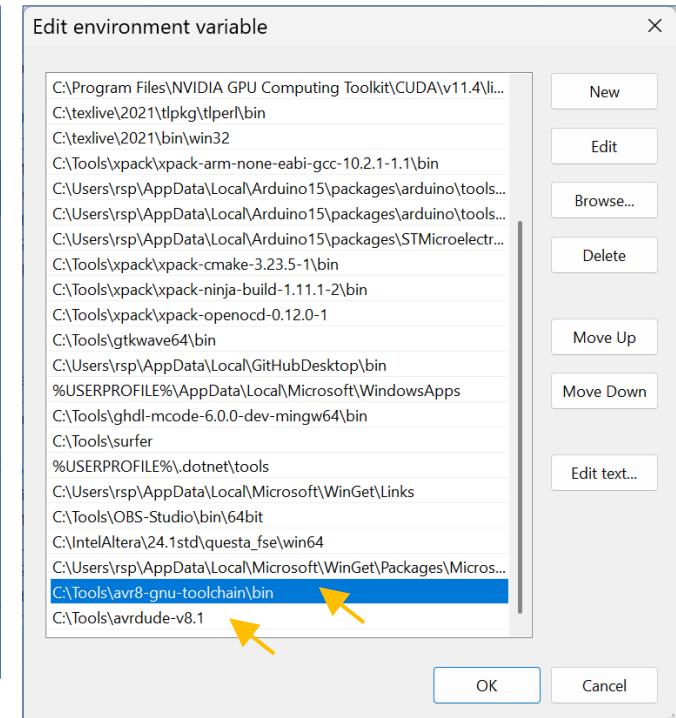
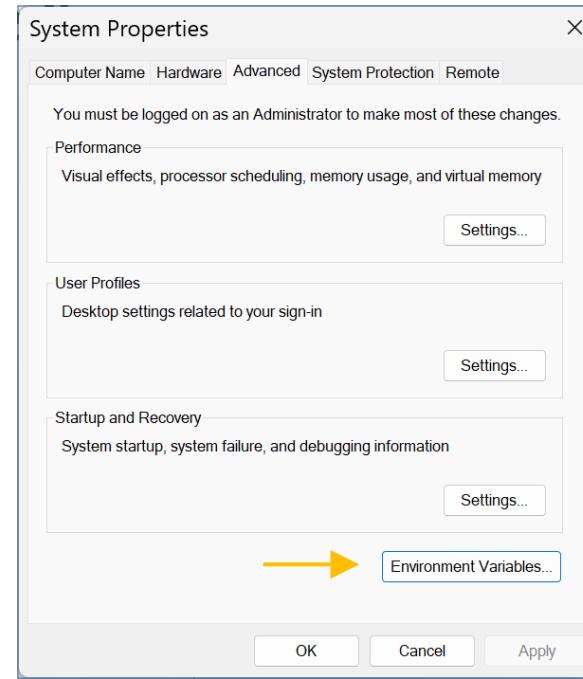
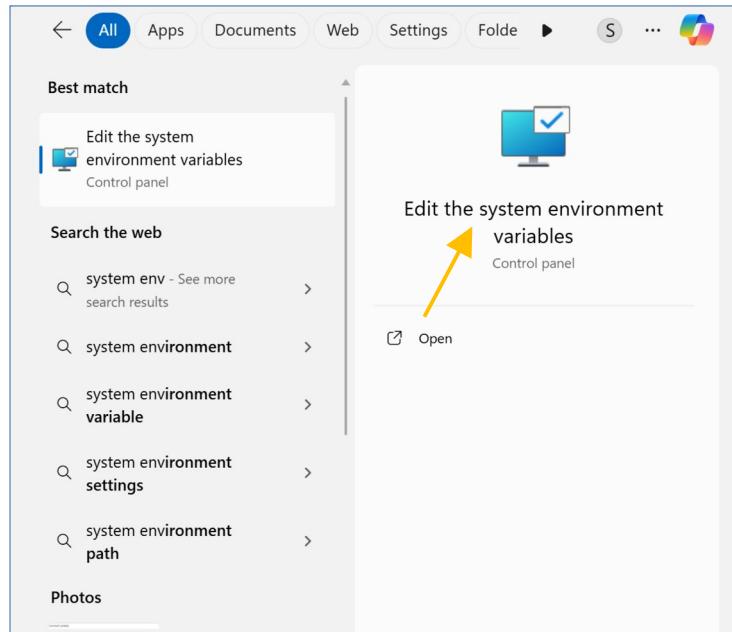
<https://github.com/avrdudes/avrdude/releases>

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

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

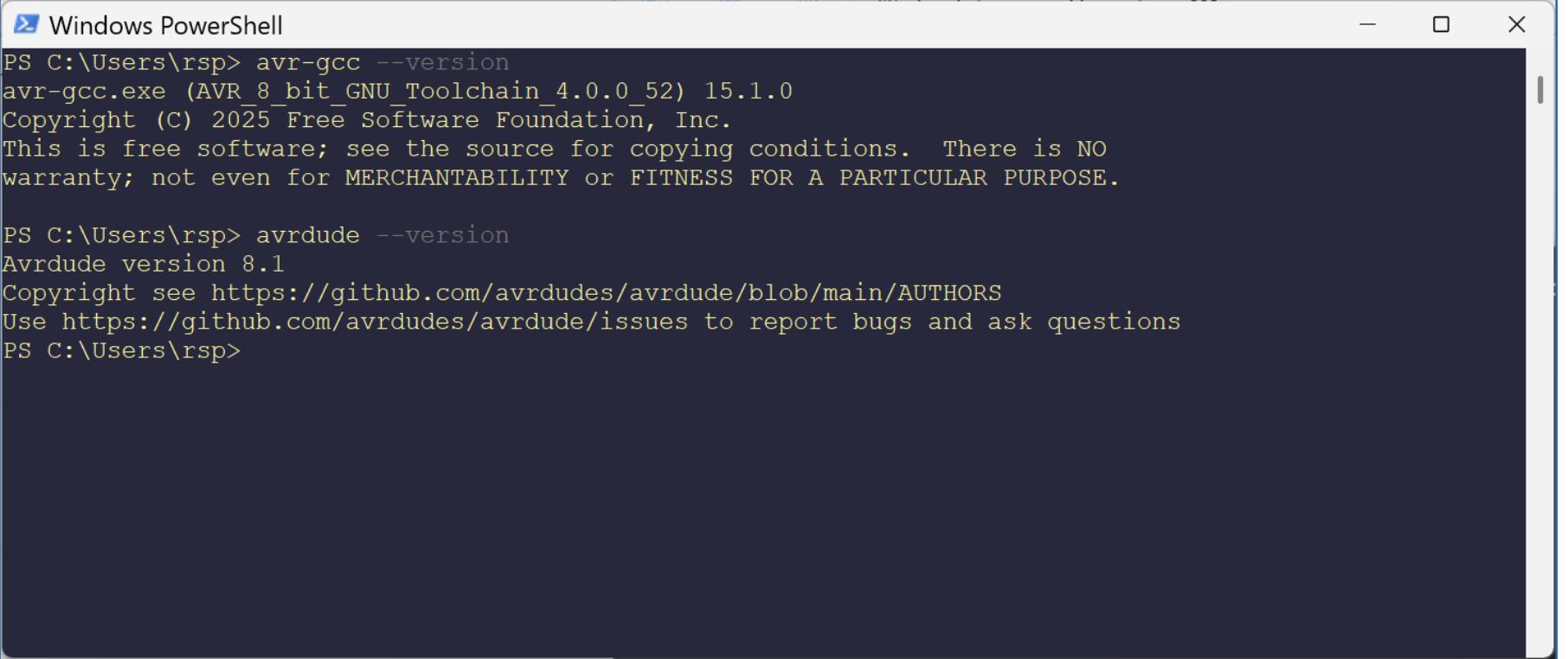
`C:\Tools\avrdude-v8.1`

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



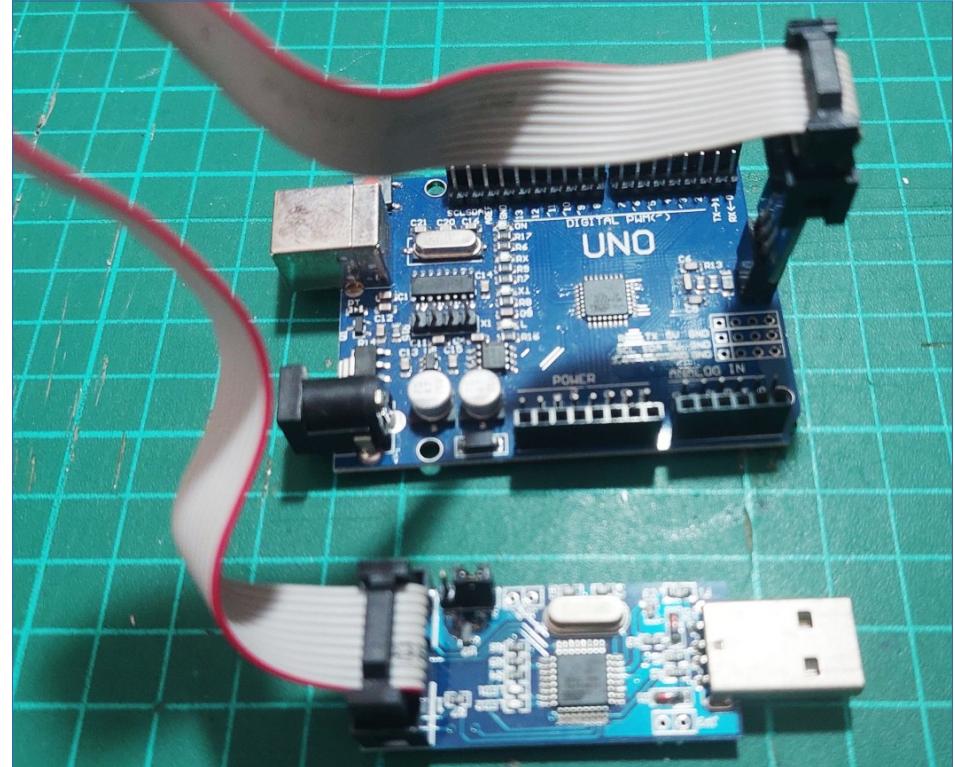
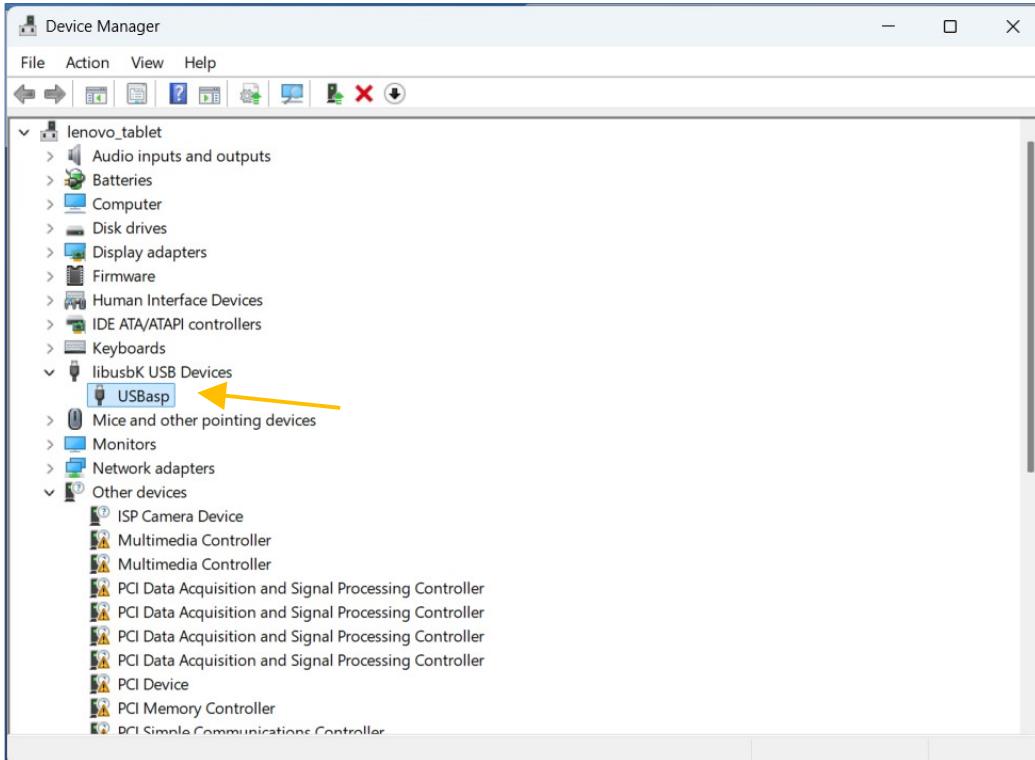
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the command "avr-gcc --version" being run, which outputs the version information for the AVR-GCC toolchain. It also shows the command "avrdude --version" being run, which outputs the version information for the Avrdude tool. The PowerShell window has a dark blue background and white text.

```
PS C:\Users\rsp> avr-gcc --version
avr-gcc.exe (AVR_8_bit_GNU_Toolchain_4.0.0_52) 15.1.0
Copyright (C) 2025 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

PS C:\Users\rsp> avrdude --version
Avrdude version 8.1
Copyright see https://github.com/avrdudes/avrdude/blob/main/AUTHORS
Use https://github.com/avrdudes/avrdude/issues to report bugs and ask questions
PS C:\Users\rsp>
```

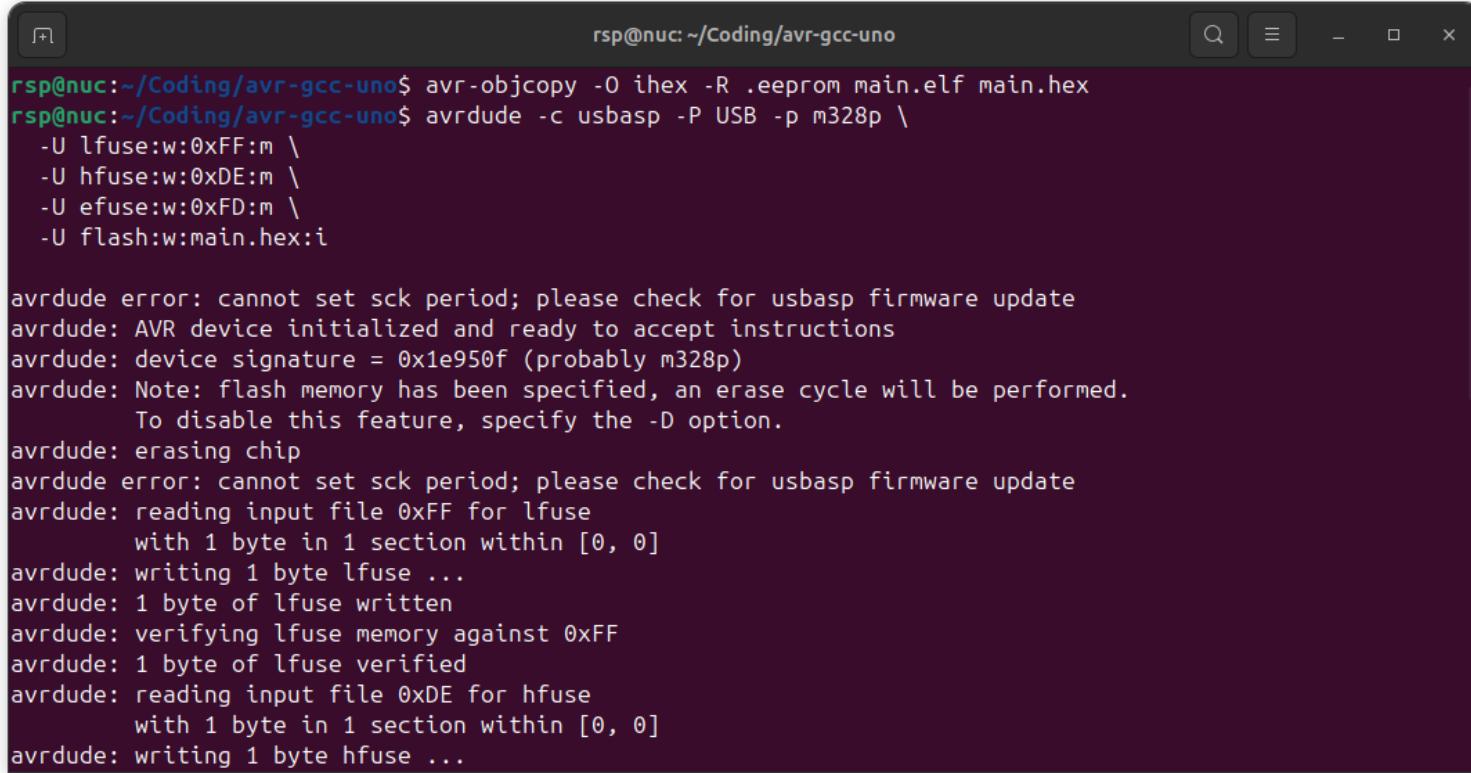
Try to run the **avr-gcc.exe** and **avrdude.exe** command in Windows Powershell.

Using USBasp under Windows 11



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

Using CLI Tools under Ubuntu 24.04



A screenshot of a terminal window titled "rsp@nuc: ~/Coding/avr-gcc-uno". The terminal displays a sequence of commands and their output:

```
rsp@nuc:~/Coding/avr-gcc-uno$ avr-objcopy -O ihex -R .eeprom main.elf main.hex
rsp@nuc:~/Coding/avr-gcc-uno$ 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

avrdude error: cannot set sck period; please check for usbasp firmware update
avrdude: AVR device initialized and ready to accept instructions
avrdude: device signature = 0x1e950f (probably m328p)
avrdude: Note: flash memory has been specified, an erase cycle will be performed.
      To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude error: cannot set sck period; please check for usbasp firmware update
avrdude: reading input file 0xFF for lfuse
      with 1 byte in 1 section within [0, 0]
avrdude: writing 1 byte lfuse ...
avrdude: 1 byte of lfuse written
avrdude: verifying lfuse memory against 0xFF
avrdude: 1 byte of lfuse verified
avrdude: reading input file 0xDE for hfuse
      with 1 byte in 1 section within [0, 0]
avrdude: writing 1 byte hfuse ...
```

```
$ 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.

AVRDUE 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 ArduinolISP

```
$ 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 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

New Arduino Uno Project - Wokwi

wokwi.com/projects/new/arduino-uno

WOKWI SAVE SHARE Docs SIGN IN

sketch.ino diagram.json Library Manager

```
1 void setup() {
2     // put your setup code here, to run once:
3 }
4
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8 }
9
10
```

Simulation

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

Wokwi Simulator: AVR Assembly Simulation

The screenshot shows the Wokwi Simulator interface for an Arduino Uno project. The top navigation bar includes tabs for 'sketch.ino', 'diagram.json', 'main.S' (selected), and 'Library Manager'. The main workspace is divided into two sections: 'Simulation' on the left and 'WOKWI LOGIC' on the right.

Simulation Section: Displays the AVR assembly code for the 'main.S' file. The code toggles an LED connected to pin PB5 every 100ms. A yellow callout box points to the file tab with the text "Create a new file (main.S)".

```
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, 0XF        ; high byte
25
26 inner_loop:
27     sbiw r24, 1         ; subtract 1 (2 cycles)
```

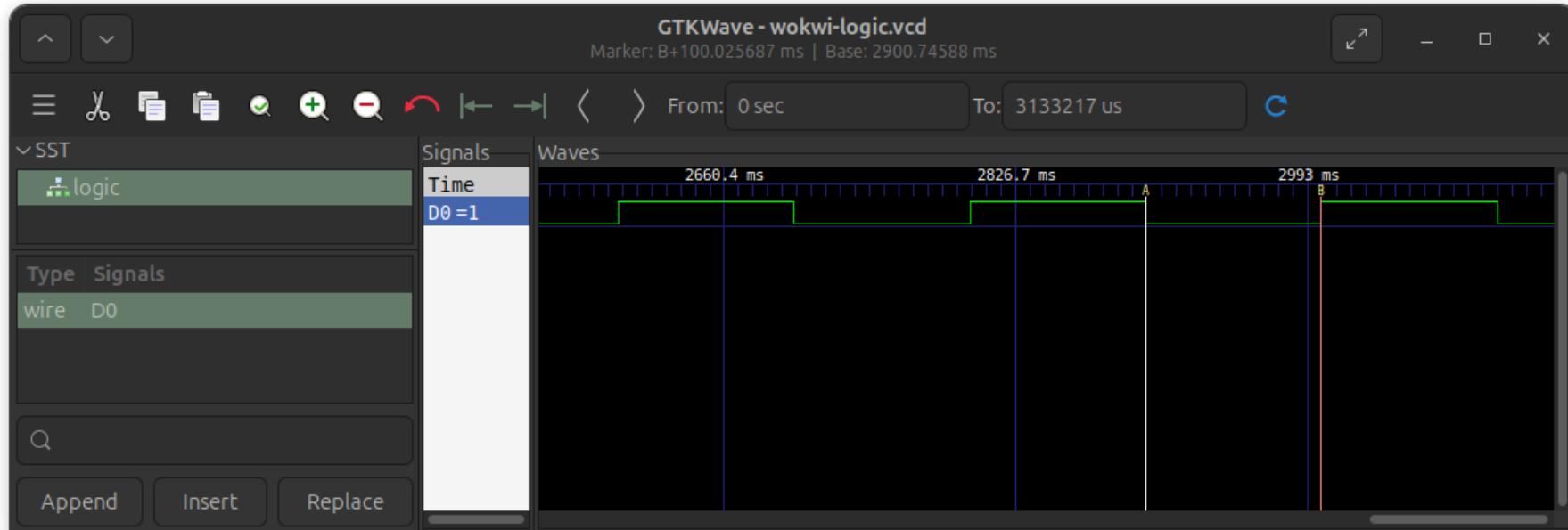
WOKWI LOGIC Section: Shows a virtual logic analyzer window titled "20 samples". It displays the digital state of pins D0 through D7 over time. A yellow callout box points to this window with the text "8-bit Virtual Logic Analyzer".

Bottom: An Arduino Uno board is shown with its pins labeled. Pin 13 is connected to the LED, and pin 11 is connected to the resistor. The board also has a red power LED and a green "ON" indicator.

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
```

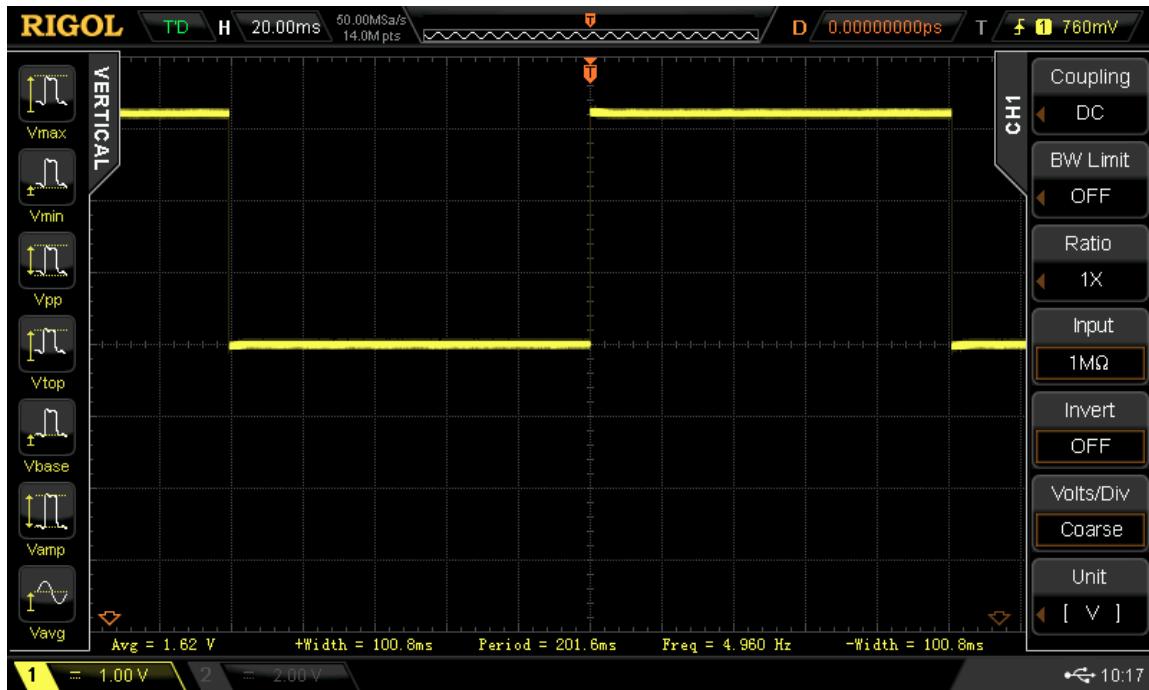
Convert ELF file to AVR assembly code listing.

```
$ avr-objdump -d main.elf
```

Upload HEX File to Arduino Uno

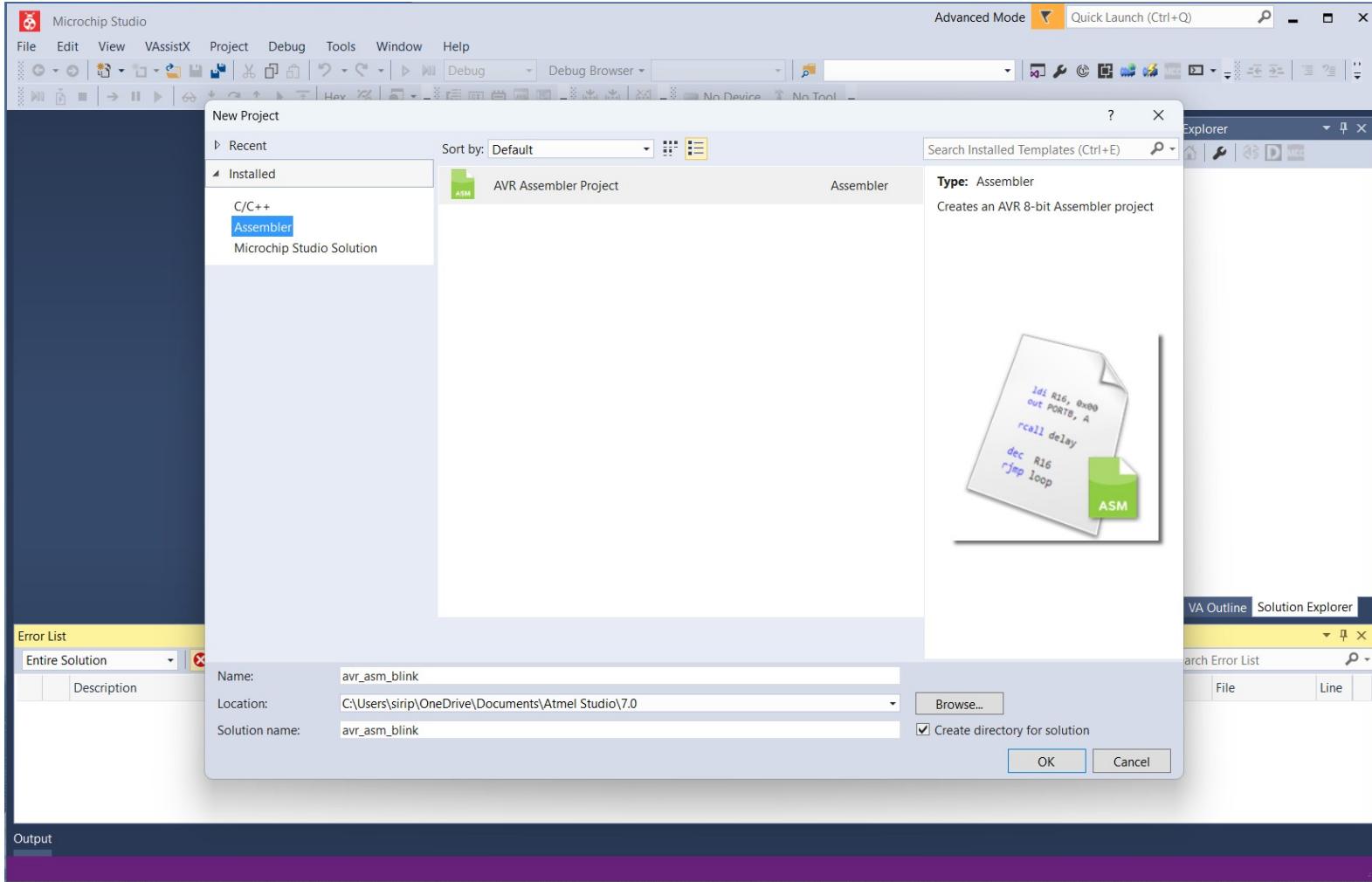
```
# Use avrdude with the USBasp programmer to flash the firmware.  
# Need to overwrite the Arduino bootloader (without -D option).  
  
$ avrdude -c usbasp -P USB -p m328p \  
  -U lfuse:w:0xFF:m \  
  -U hfuse:w:0xDE:m \  
  -U efuse:w:0xFD:m \  
  -U flash:w:main.hex:i
```

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

The screenshot shows the Microchip Studio interface with the project "avr_asm_blink" open. The assembly code is displayed in the main editor window, and the build output is shown in the "Output" tab at the bottom.

```
.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 @ 16MHz
; =====
delay_100ms:
    ldi r18, 13          ; outer
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

    .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
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
```

The assembly code implements a simple LED blink sequence. It initializes port B pin 5 (PB5) as an output. The main loop toggles the LED by reading the state of port B pin 5 and then calling a delay subroutine. The delay subroutine consists of three nested loops using registers r18, r19, and r20. The outer loop (L1) uses a value of 13, the middle loop (L2) uses 204, and the inner loop (L3) uses 200. The build output shows the assembly code was successfully generated without errors or warnings.

AVR Simulation

The screenshot shows the Microchip Studio interface during AVR simulation. The assembly code in the main.asm file includes a main loop that toggles an LED connected to PB5. The I/O register viewer shows various port pins and registers. The processor status window displays the program counter at 0x00000002, stack pointer at 0x08FF, and a cycle counter of 3. A yellow callout highlights the cycle count and stop watch information, noting that the CPU frequency is set to 16MHz.

Set Breakpoint

Processor Status

Name	Value
Program Counter	0x00000002
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	0x0000
Cycle Counter	3
Frequency	16.000 MHz
Stop Watch	0.19 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00

Cycle Count & Stop Watch
(CPU Freq. is set to 16MHz)

Call Stack Breakpoints Command Window Immediate Window Output Memory 4

Ln 11 Col 1 Ch 1 INS

45

AVR Simulation

The debugger temporarily halts execution at this breakpoint.

Disassembly main.asm

```
.include "m328pdef.inc"

.cseg
.org 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:
```

I/O

Name	Value
Analog Comparator (AC)	0x00
Analog-to-Digital Convert...	0x00
CPU Registers (CPU)	0x00
Clock Prescaler Select...	0x00
Sleep Mode Select Bits...	0x00
EEPROM (EEPROM)	0x00
External Interrupts (EXINT)	0x00
I/O Port (PORTB)	0x00
I/O Port (PORTC)	0x00
I/O Port (PORTD)	0x00
Serial Peripheral Interface (...)	0x00
Timer/Counter, 16-bit (TC1)	0x00
Timer/Counter, 8-bit (T0)	0x00
Timer/Counter, 8-bit Async...	0x00
Two Wire Serial Interface (...)	0x00
USART (USART0)	0x00
Watchdog Timer (WDT)	0x00

Processor Status

Name	Value
Program Counter	0x00000002
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	0x0000
Cycle Counter	1599209
Frequency	16.000 MHz
Stop Watch	99,950.56 µs

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)

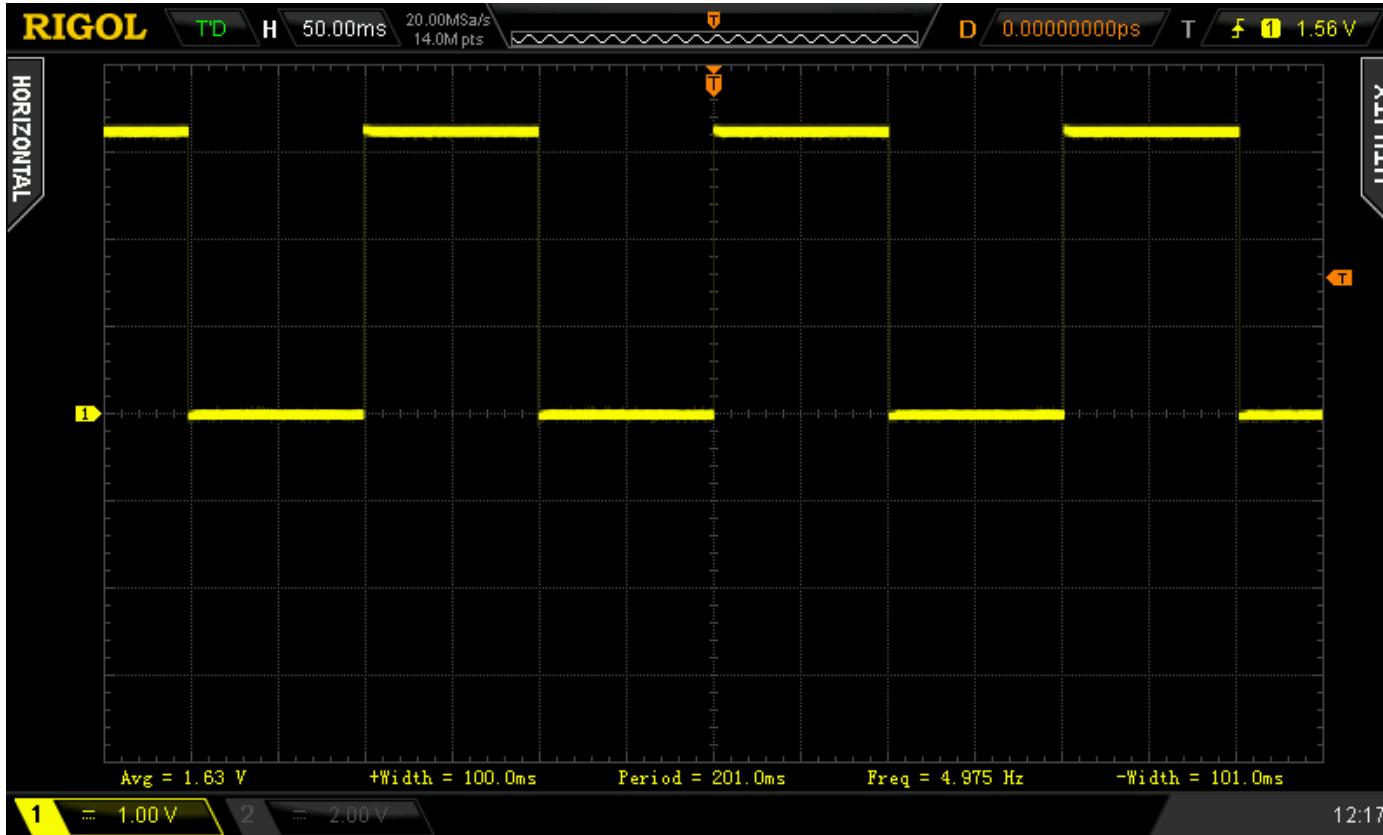
```
$ avr-objcopy -O ihex -R .eeprom main.elf main.hex
```

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, 0XF             ; 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

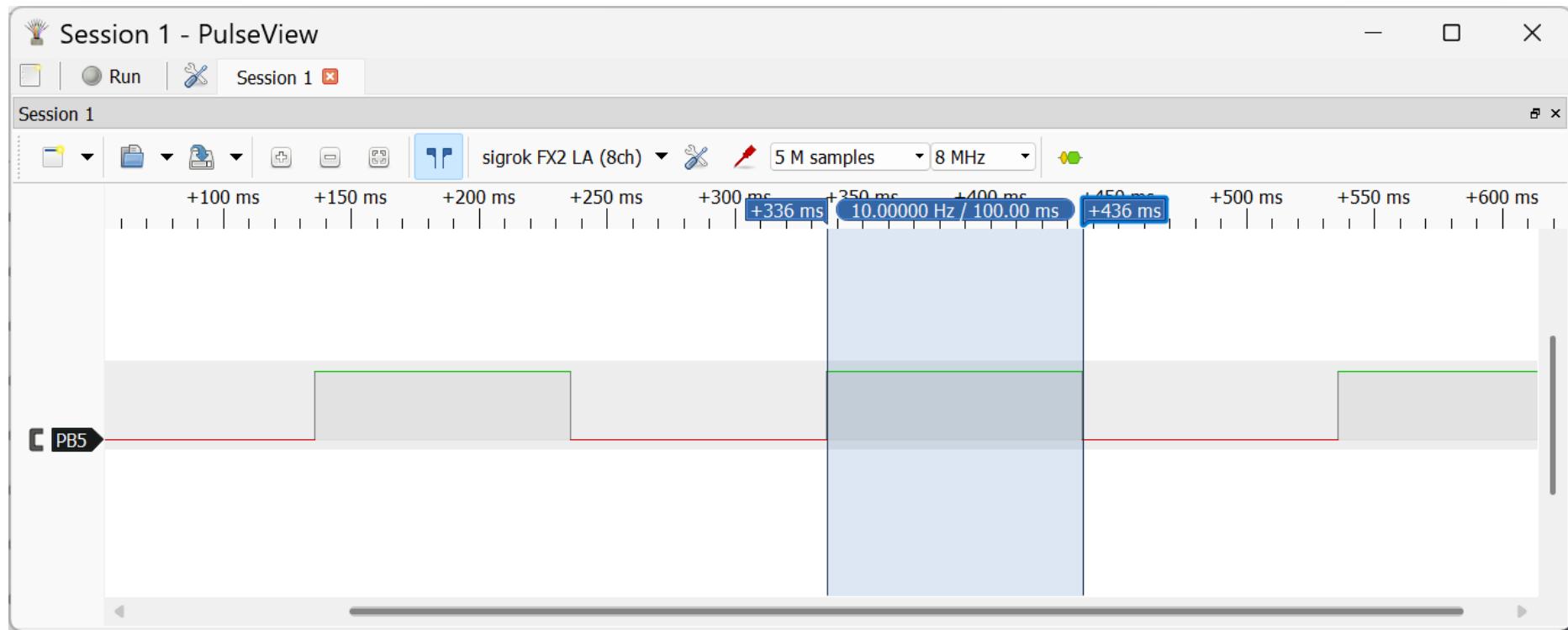
loop:
    sbi PINB, PINB5      ; toggle LED
    rcall delay_100ms    ; ~100 ms delay
    rjmp loop            ; repeat

delay_100ms:
    ldi r20, 100          ; outer loop count

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

The screenshot shows the VS Code Marketplace interface. On the left, the sidebar lists various extensions, with the 'avr helper' extension highlighted by an orange arrow. The main area displays the 'Extension: AVR Helper' details page. The AVR Helper extension, developed by Alex079, has 15,955 installs and a perfect 5-star rating. Its description states: "Helper extension to simplify code compilation and flashing for AVR MCUs." It requires 'avr-gcc' and 'avrduude' to be installed. The 'How it works' section explains that the extension provides a visual way to automate routine build and flash tasks. A 'SUGGESTED ACTIONS' sidebar on the right includes 'Build Workspace' and 'Show Config' buttons, along with an 'Add Context...' button and a 'Describe what to build next' input field.

EXTENSIONS: MARKETPLACE

avr helper

AVR Helper 15K ⭐ 5
Helper extension to simplify code com...
Alex079 [Install](#)

AVR Utils 2K ⭐ 5
This extension will help you to comp...
Agani Daniel S. [Install](#)

avro-idl 87K ⭐ 4
Avro IDL Syntax Highlighting
Street Side Software [Install](#)

Avrotize 735
This extension provides easy access t...
Clemens Vasters [Install](#)

Data Preview 726K ⭐ 4
Data Preview ?? extension for import...
Random Fractals Inc. [Install](#)

Cisac Formats Highlighter 11
Syntax highlighting for Cisac formats
Spanish Point Technologies [Install](#)

RRPG 1K
RPGLE & RPG400 syntax highlighting
SpineNetSystems [Install](#)

CHAT

AVR Helper

AVR Helper 15,955 ⭐ 5
Helper extension to simplify code com...
[Install](#) Auto Update

DETAILS FEATURES CHangelog

AVR Helper Extension

Helper extension to simplify code compilation and flashing for AVR MCUs.

This extension allows building and flashing executable code for AVR from C/C++ source files. It needs `avr-gcc` and `avrduude` installed. It uses `C/C++` extension to provide language support.

How it works

The goal of the extension is to provide a visual way of automating routine build and flash tasks.

The AVR Helper extension acts as a bridge between user...

SUGGESTED ACTIONS

[Build Workspace](#) [Show Config](#)

Add Context...
Describe what to build next
Agent Pick Model

× 0 △ 0

(•) Go Live

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:

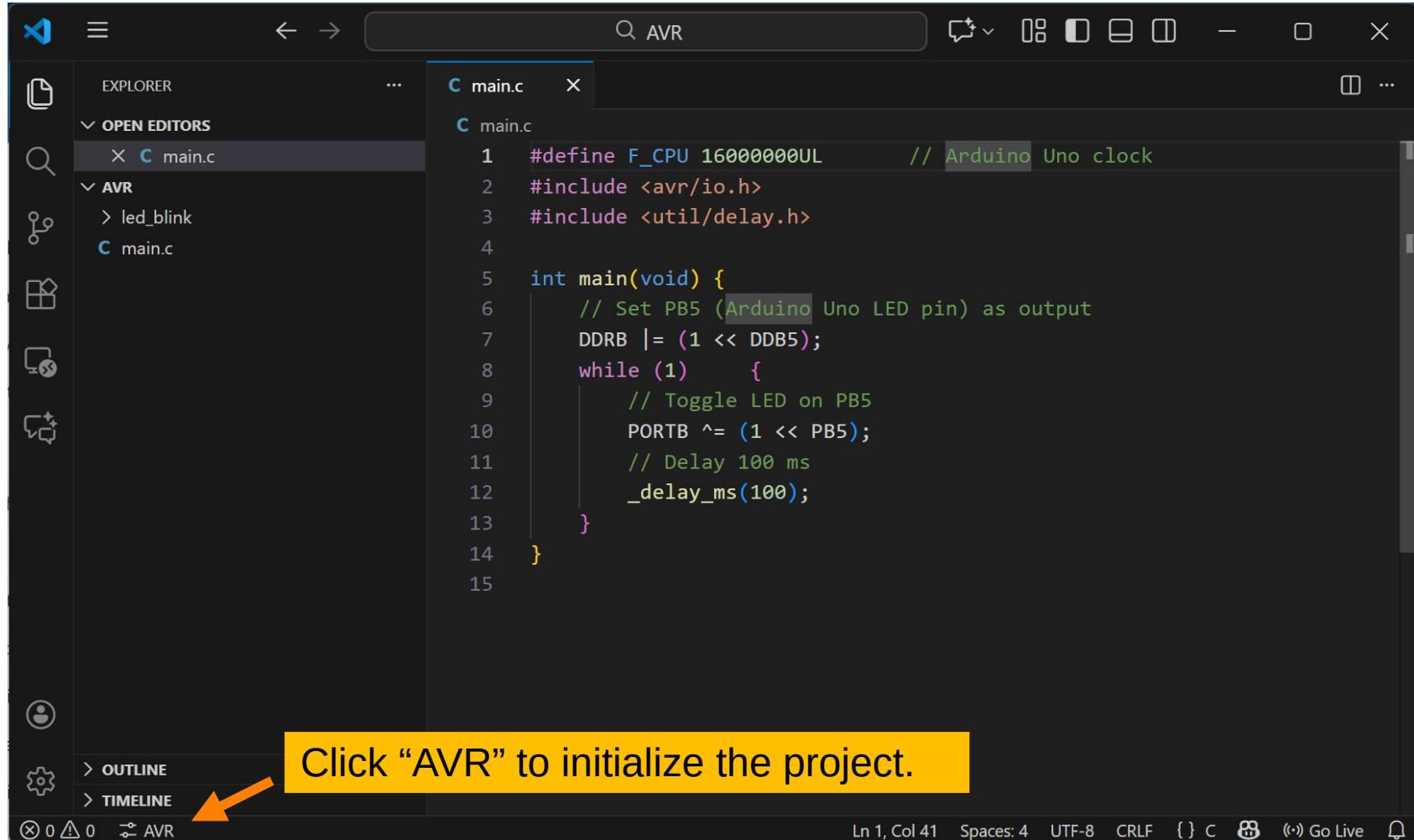
- Open folder for your project
- Create your AVR source file (e.g. main.c)

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)

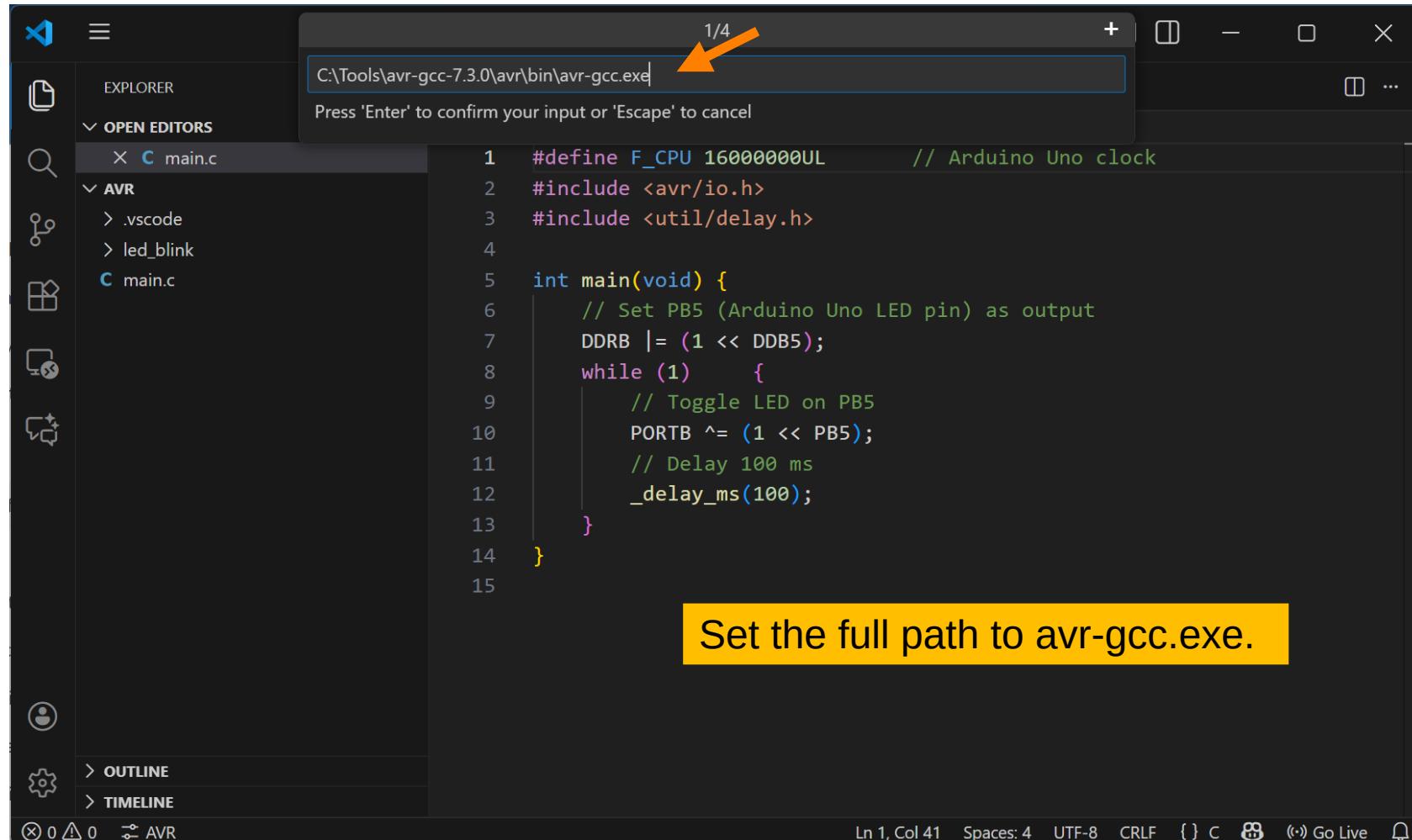


The screenshot shows the Visual Studio Code interface with the AVR Helper extension installed. The left sidebar has an 'AVR' section expanded, showing a project named 'led_blink' with files 'main.c' and 'main.c'. The main editor window displays the 'main.c' file content:

```
1 #define F_CPU 16000000UL      // Arduino Uno clock
2 #include <avr/io.h>
3 #include <util/delay.h>
4
5 int main(void) {
6     // Set PB5 (Arduino Uno LED pin) as output
7     DDRB |= (1 << DDB5);
8     while (1) {
9         // Toggle LED on PB5
10        PORTB ^= (1 << PB5);
11        // Delay 100 ms
12        _delay_ms(100);
13    }
14 }
15
```

A yellow callout box with the text "Click ‘AVR’ to initialize the project." points to the 'AVR' button in the bottom-left corner of the interface.

VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)

EXPLORER

OPEN EDITORS

C:\Tools\avrdude-v8.1\avrdude.exe

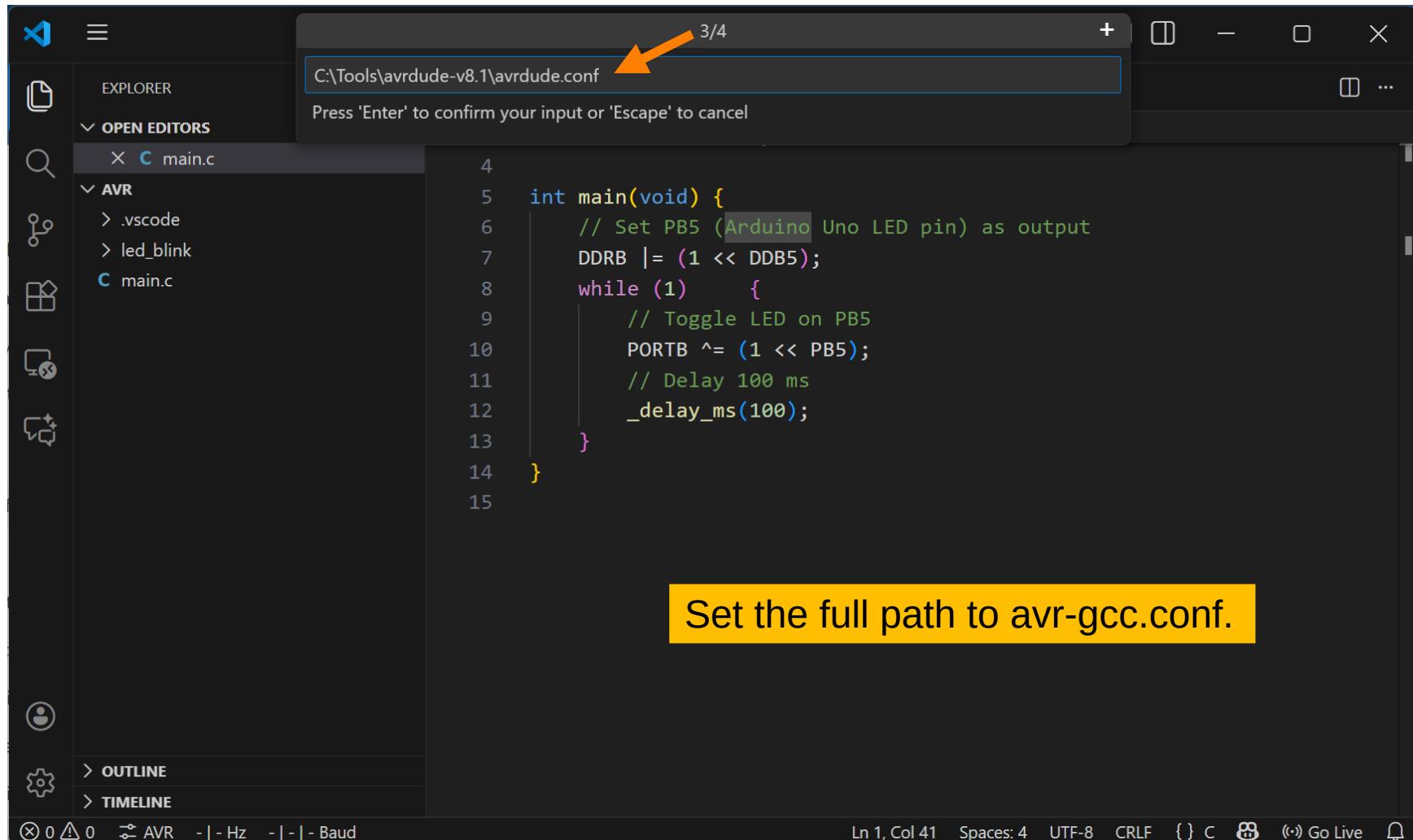
Press 'Enter' to confirm your input or 'Escape' to cancel

```
1 #define F_CPU 16000000UL // Arduino Uno clock
2 #include <avr/io.h>
3 #include <util/delay.h>
4
5 int main(void) {
6     // Set PB5 (Arduino Uno LED pin) as output
7     DDRB |= (1 << DDB5);
8     while (1) {
9         // Toggle LED on PB5
10        PORTB ^= (1 << PB5);
11        // Delay 100 ms
12        _delay_ms(100);
13    }
14 }
15
```

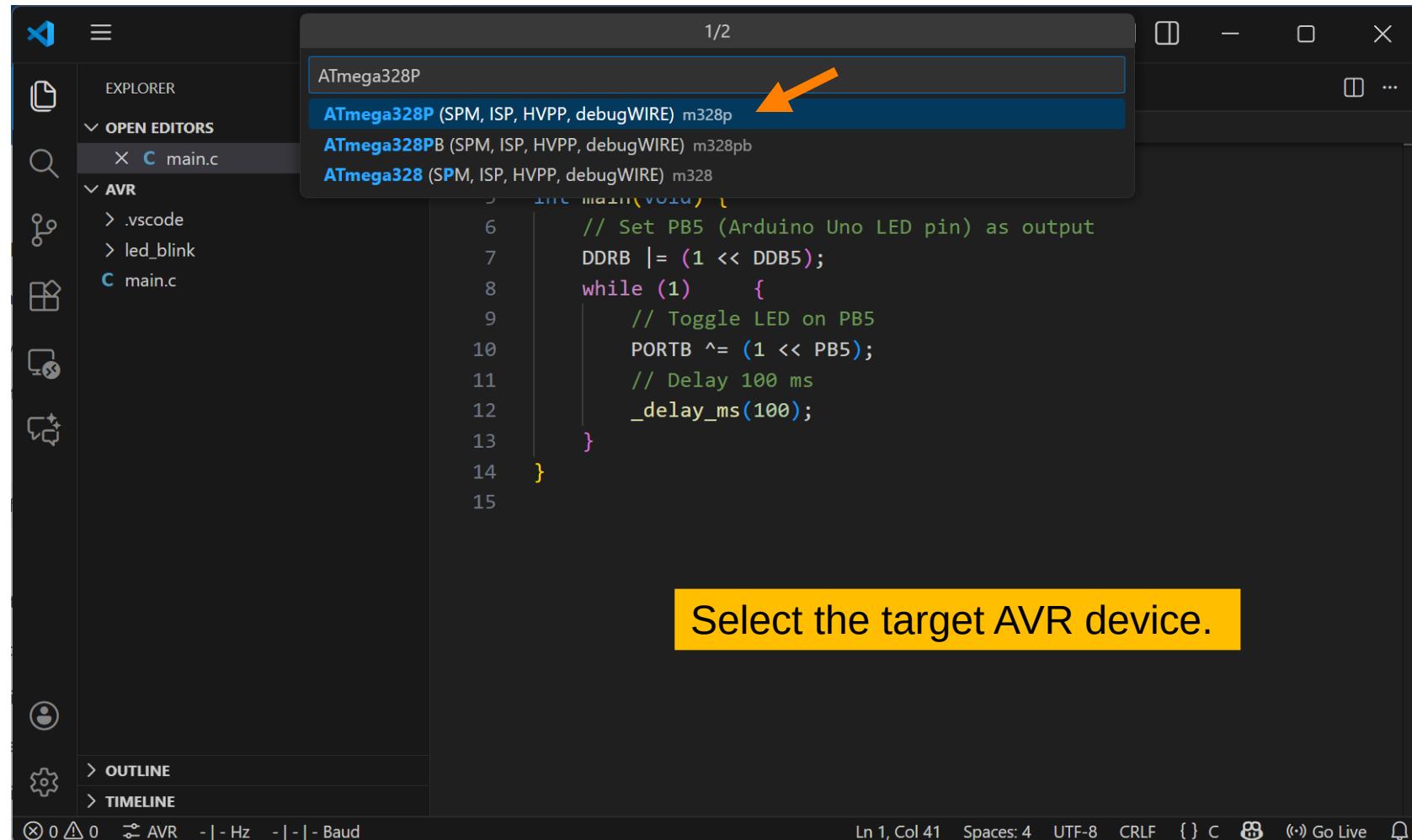
Set the full path to avrdude.exe.

Ln 1, Col 41 Spaces: 4 UTF-8 CRLF { } C ⚙️ (↻) Go Live 📲

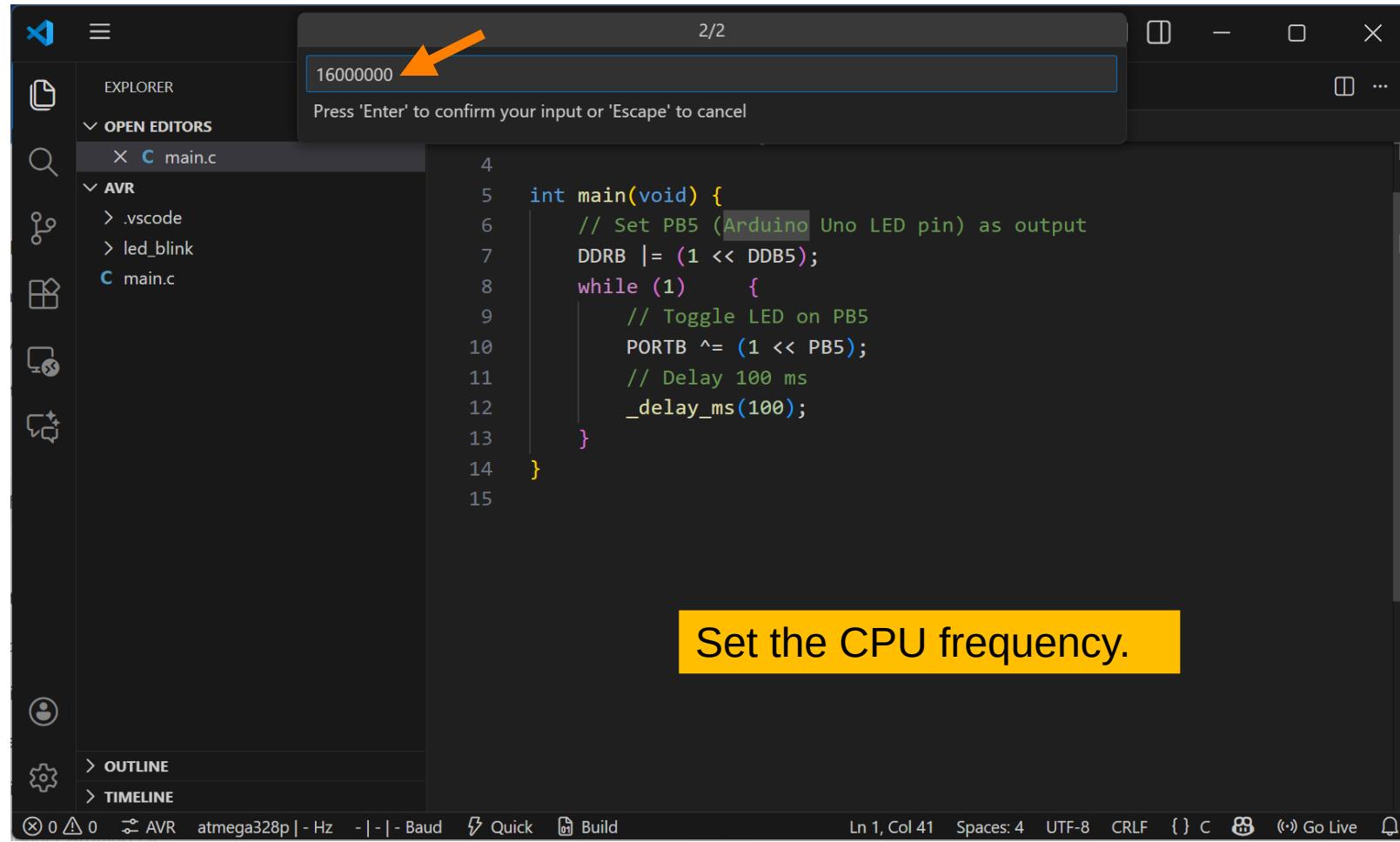
VS Code IDE + AVR Helper (Extension)



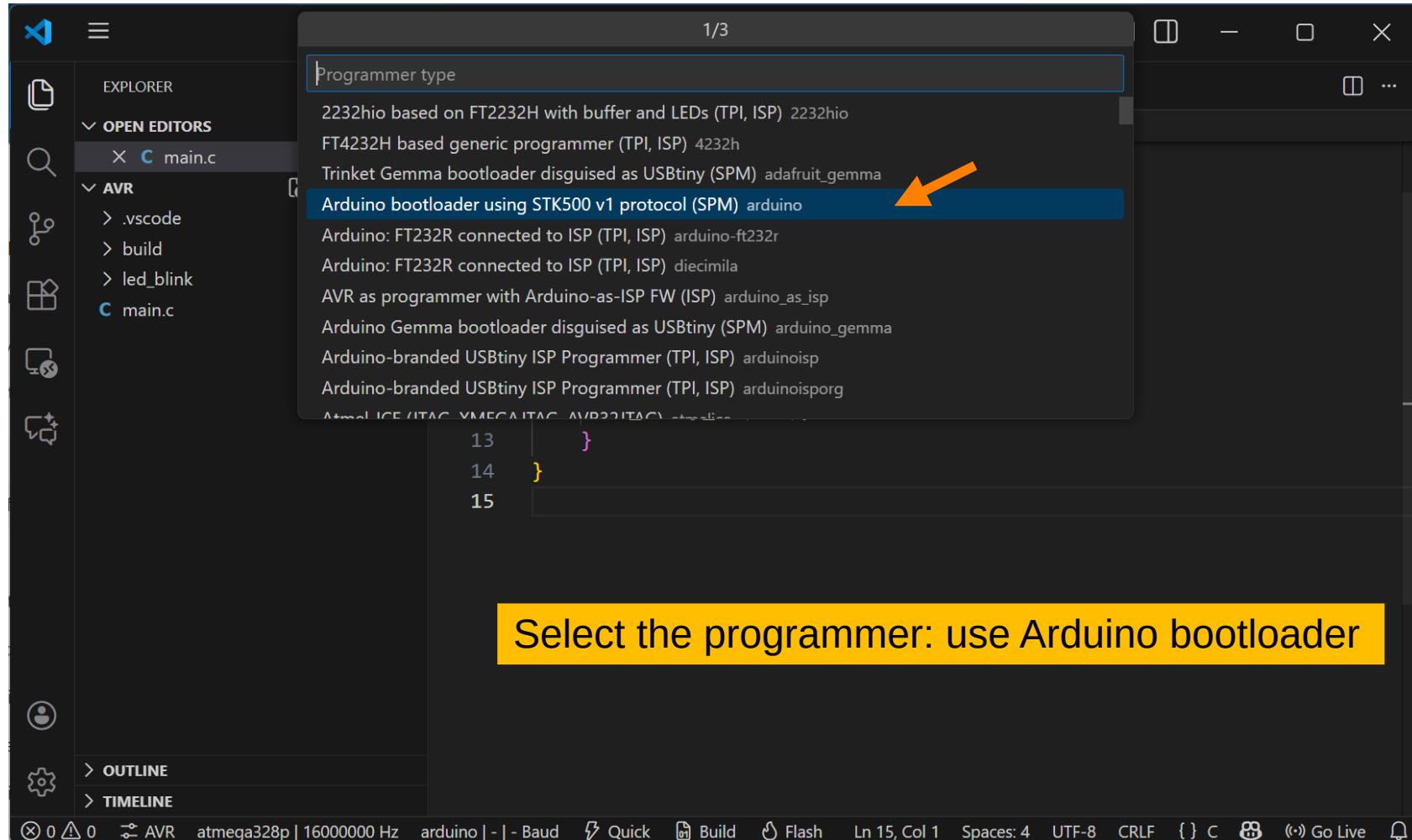
VS Code IDE + AVR Helper (Extension)



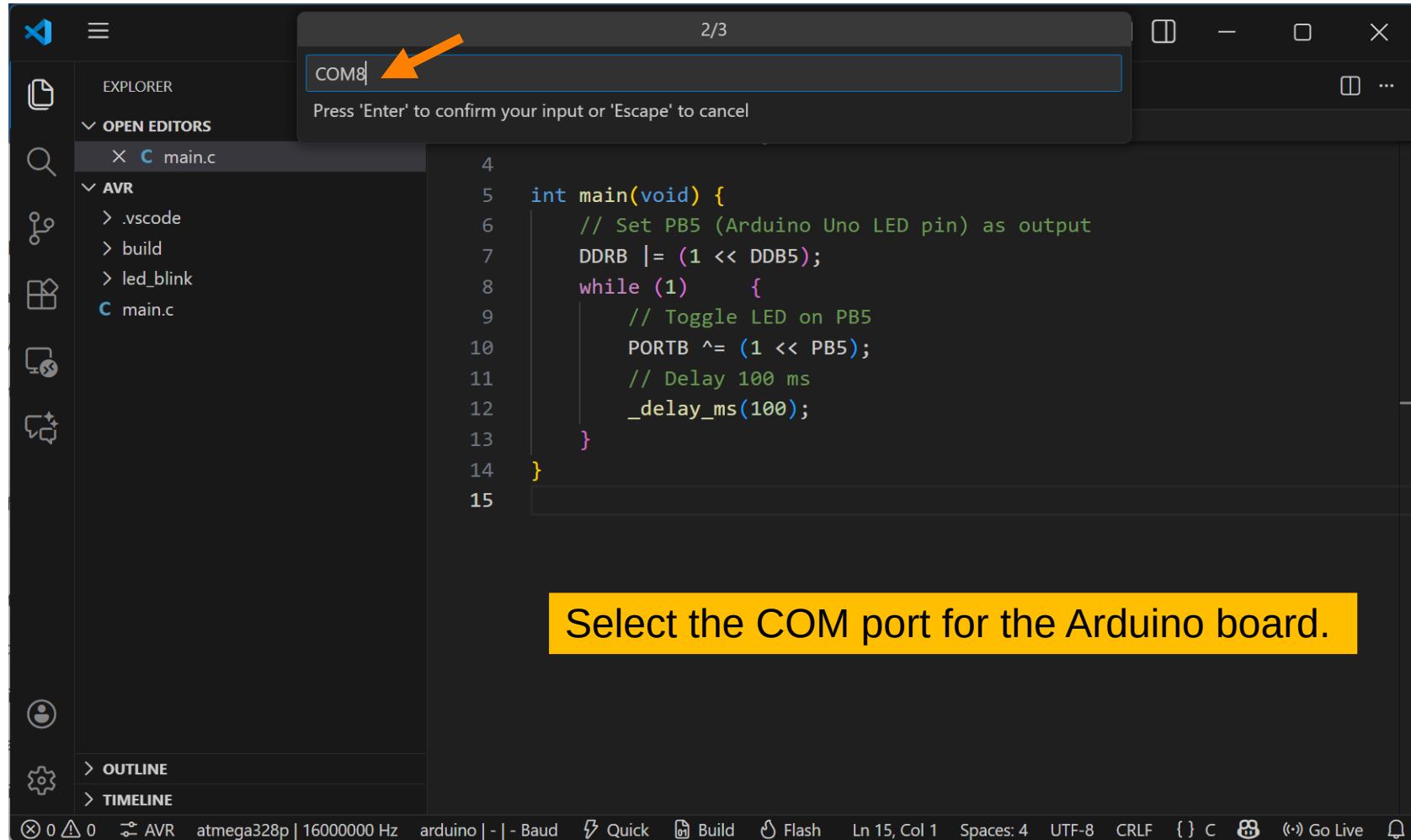
VS Code IDE + AVR Helper (Extension)



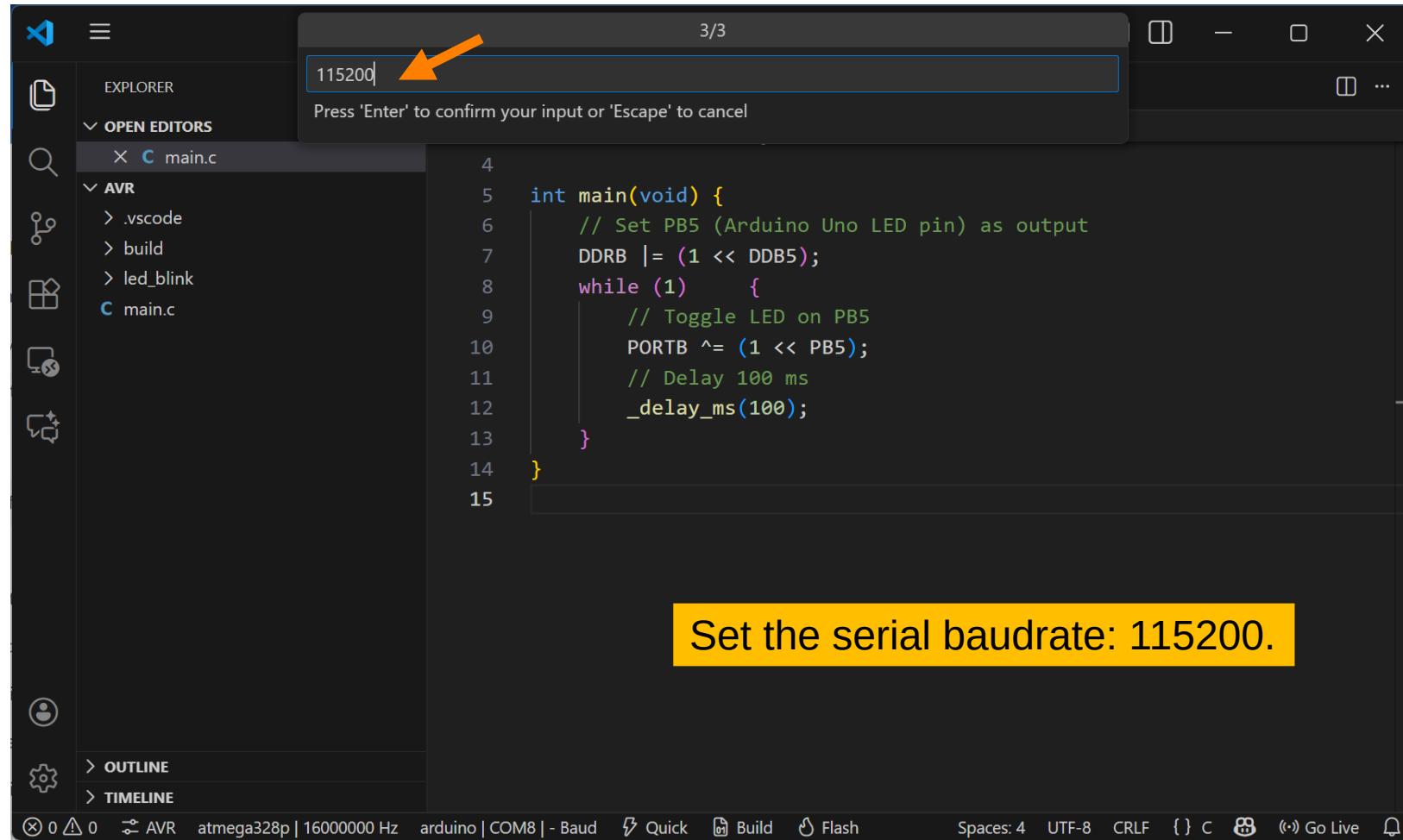
VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)



VS Code IDE + AVR Helper (Extension)

The screenshot shows the VS Code interface with the AVR Helper extension installed. The Explorer sidebar on the left lists project files: main.c, settings.json, .vscode, AVR, build, obj\c_\Users\rsp\Coding\AVR, and led_blink. The AVR folder contains .vscode, c_cpp_properties.json, and settings.json. The main editor area displays the contents of settings.json, which configures the AVR reporter arguments and compiler/programmer paths. The status bar at the bottom provides real-time build information: AVR atmega328p | 16000000 Hz, arduino | COM8 | 115200 Baud, and various build and flash icons.

```
1  {
2      "AVR.reporter.arguments": [
3          "-C"
4      ],
5      "AVR.source.compiler": "C:\\\\Tools\\\\avr-gcc-7.3.0\\\\avr\\\\bin\\\\avr-gcc.exe",
6      "AVR.programmer.tool": "C:\\\\Tools\\\\avrdude-v8.1\\\\avrdude.exe",
7      "AVR.programmerdefinitions": "C:\\\\Tools\\\\avrdude-v8.1\\\\avrdude.conf",
8      "AVR.source.libraries": [],
9      "AVR.device.type": "atmega328p",
10     "AVR.device.frequency": 16000000,
11     "AVR.programmer.type": "arduino",
12     "AVR.programmer.port": "COM8",
13     "AVR.programmer.rate": 115200
14 }
```

<project>/.vscode/settings.json

VS Code IDE + AVR Helper (Extension)

