Lab Sheet for Week 3

Lab Instructor: **RSP**

Lab 1: Bare-metal C Programming for AVR using Microchip Studio IDE

Objectives

- Use Atmel Studio 7 or Microchip Studio for AVR to develop firmware for the ATmega328P MCU.
- Write bare-metal embedded C code to perform basic I/O operations on the MCU's I/O ports and pins.
- Use the built-in simulator in Atmel Studio 7 or Microchip Studio for AVR, or the Wokwi Simulator, to simulate and test the source code.
- Use AVRDUDE to upload firmware to the Arduino Uno or Nano board without overwriting the Arduino bootloader.

In this lab, we will learn how to use **Atmel Studio 7 (Microchip Studio for AVR)** to write C code for the **ATmega328P**, build a project to generate output files such as a .hex file, and perform source-level debugging using the built-in simulator.

Using the built-in AVR simulator provided by the Atmel/Microchip software, we will learn how to set breakpoints, step through the code during a simulation session, and observe the CPU status and I/O registers.

After successfully building the project, the .hex file can be uploaded to the MCU board using **AVRDUDE**, an open-source software tool. To verify the code, we will also use a digital oscilloscope to measure the output signal on the Arduino board.

Register-based AVR I/O Programming

The first lab activities focus on I/O port configuration and I/O read/write operations using register-level C programming. **Figure 1** shows the details of the three I/O registers of **Port B** as an example.

1) **DDRx** (Data Direction Register): Controls the direction of each pin of the same I/O port.
   - 1 = configure pin as output
   - 0 = configure pin as input

2) **PINx** (Port Input Register): Used to read the logic level on an input pin.
   - The input value is read from the PINx register.
   - Writing a 1 to a **PINx** bit toggles the corresponding **PORTx** output bit.

3) **PORTx** (Port Output Register): Used to set the logic level on an output pin.
   - When the pin is configured as output: Writing to **PORTx** sets the output value (HIGH or LOW).
   - When the pin is configured as input: Writing 1 to **PORTx** enables the internal pull-up resistor.

Note: When a pin is configured as an output (i.e., bit *b* of **DDRx** = 1), writing a 1 to the corresponding bit in **PINx** causes the hardware to toggle the output state of that pin.

PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Figure 1: I/O registers associated with Port B

Lab Procedure

1. Create a new C Project in **Atmel Studio / Microchip Studio.**

   1.1 Select **GCC C Executable Project** (see **Figure 2**), enter a project name, and choose a directory for the project.

   1.2 Select **ATmega328P** as the target device (see **Figure 3**) and click the “OK” button.

   1.3 Open **main.c** and edit the source code using the provided template (**Code Listing 1**).

   1.4 Compile and build the project to generate the output files (see **Figure 4**).

2. Debug the AVR program using the built-in simulator.

   2.1 Start a **debugging session** (choose **Debug → Start Debugging** and **Break** from the menu) using the simulator to begin the code execution.

   2.2 Ensure that the built-in simulator is selected as the debugger (see **Figure 5**).

   2.3 Set breakpoints on selected lines of code in **main.c** before stepping through the compiled code (see **Figure 6**).

   2.4 Observe the AVR MCU states (e.g., CPU registers, I/O registers, CPU cycle count, etc.) during code execution by opening the **Processor Status View, CPU Registers View**, **I/O View**, and **Memory View**.

   2.5 Step through the code to repeat the loop until the next breakpoint is reached.

   2.6 Open the **Disassembly View** (**Debug → Windows → Disassembly**) to see the AVR assembly code which is results from compiling the C source code. Observe the changes in the I/O register values and the CPU cycle count (see **Figure 7**).

3. Uploading the .hex file using **AVRDUDE**

3.1 Connect the **Arduino Uno board** to the host computer via a USB port.

3.2 Open **Windows PowerShell** and run the provided command lines (see **Code Listing 2**) to call the **AVRDUDE** command which uploads the .hex file of your project to the Arduino Uno board. **Figure 8** shows how the command lines are used to upload the .hex file to the target board.

3.3 Ensure that you specify or adjust the correct file paths and the **correct serial COM port number** before executing the command.

3.4 Use a digital oscilloscope to measure and capture the output signal on the **PB5** pin.

Notes:

- **AVRDUDE** is a command-line software tool used to flash, program, or upload firmware (e.g., Intel .hex files) to AVR-based boards.
- **AVRDUDE** is open-source, and prebuilt executable binaries (for Windows, macOS, and Linux) are available on its GitHub repository: https://github.com/avrdudes/avrdude/releases. If the AVRDUDE is not installed on your computer, install it first.
- If the **Arduino IDE 2.x** is already installed on the host computer, the AVRDUDE program is usually included and can be found within the Arduino installation path.

Questions

- What does it mean to “build” a project in Microchip Studio? What files are generated after building (files such as .elf, .hex and .map)?
- What is a breakpoint? Why do we set breakpoints when debugging code?
- Why is simulation valuable in early stages of embedded firmware development, even before running code on actual hardware?
- How can the CPU cycle count observed during a debug session be used to verify the correctness of the **_delay_ms()** function call? Explain how to check whether the timing is accurate.

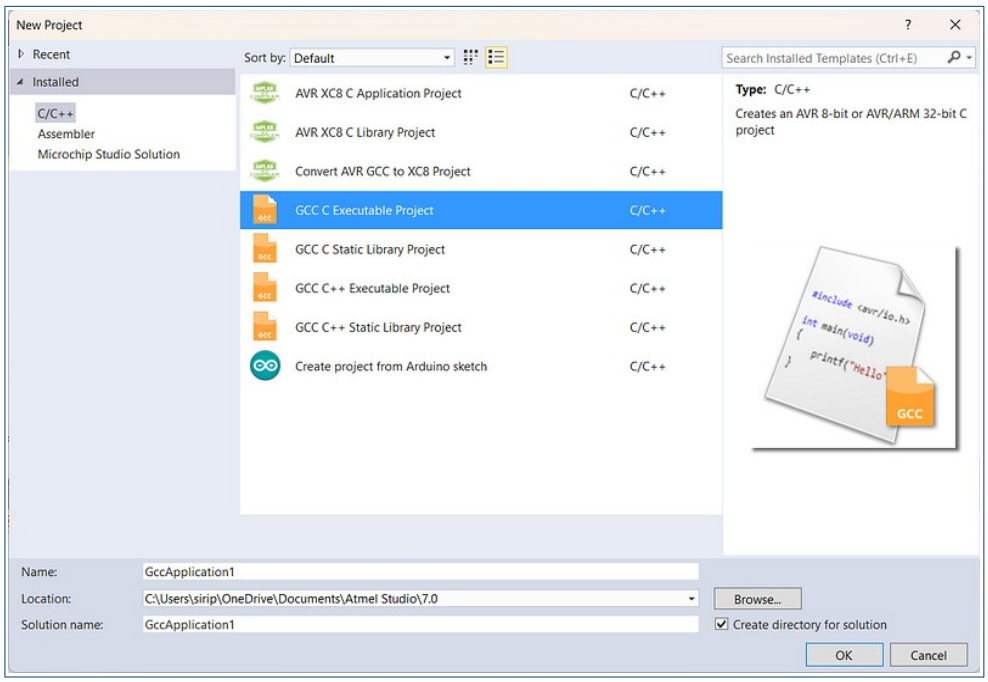

Figure 2: Project type selection in Microchip Studio for AVR

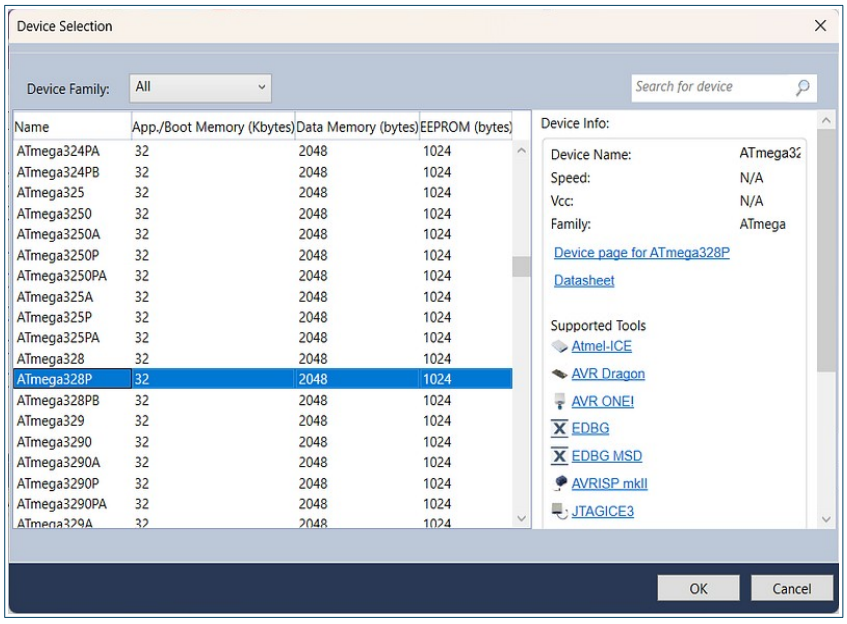

Figure 3: Device Selection (ATmega328P)

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <stdint.h> // for integer types such as uint8_t, ...
#include <avr/io.h> // for PINx, DDRx, PORTx registers
#include <util/delay.h> // for _delay_ms()

int main(void) {
 uint8_t state = 0; // Declare a local state variable
 // Configure PB5 as output
 DDRB |= (1 << DDB5); // Set PB5 bit in DDRB to make it output

 while (1) {
  State ^= 1; // Toggle the state variable
  if (state) { // use the current state to set/clear output
     PORTB |= (1 << PORTB5);   // Output PB5 high
  } else {
     PORTB &= ~(1 << PORTB5);  // Output PB5 low
  }
  _delay_ms(1); // Add some delay
 }
 return 0;
}
```

Code Listing 1: AVR C code template for **main.c** in Lab 1

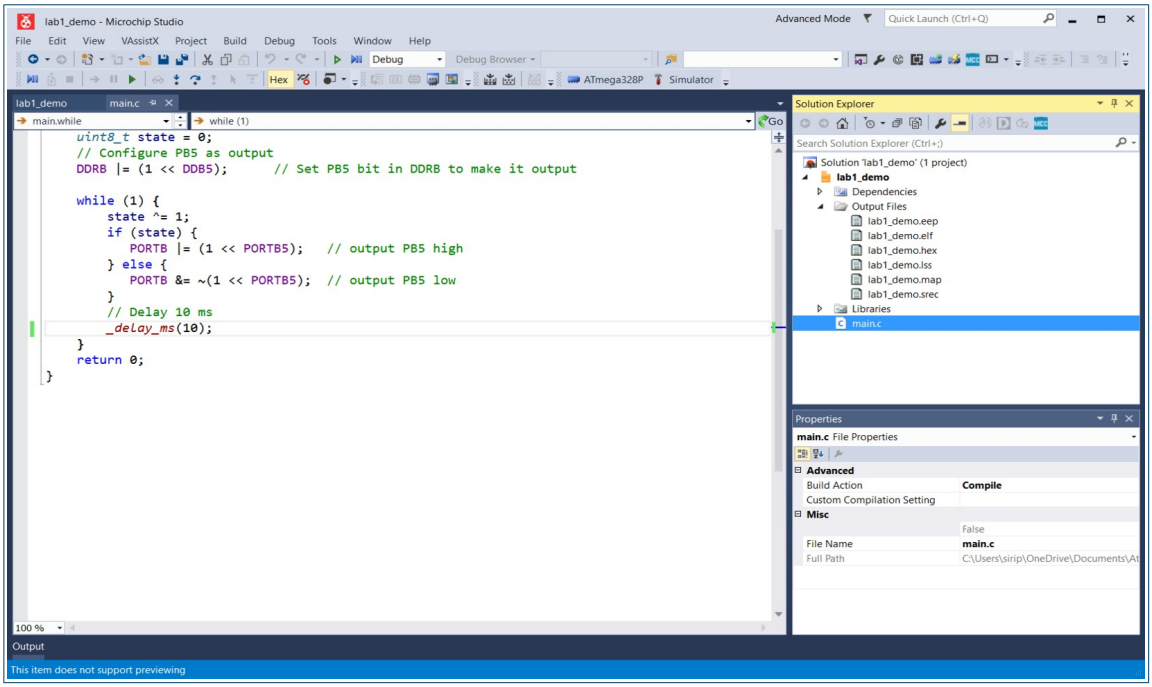

Figure 4: Output files of the project under the subdirectory named Debug

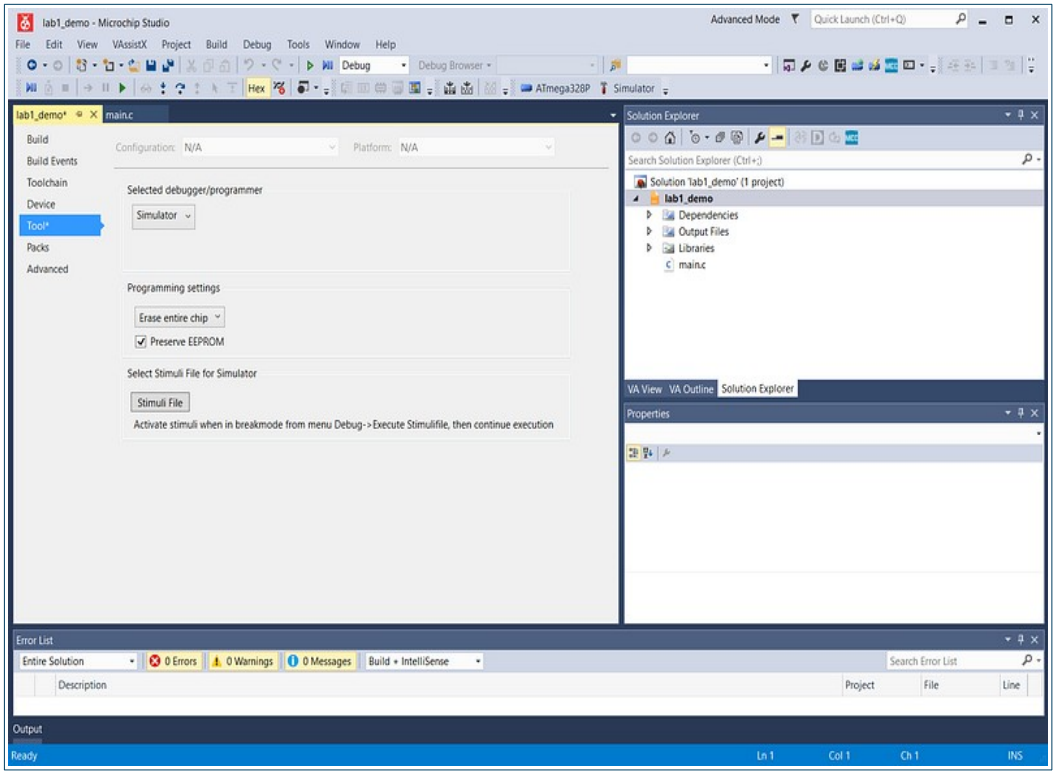

Figure 5: Selecting the built-in simulator for debugging

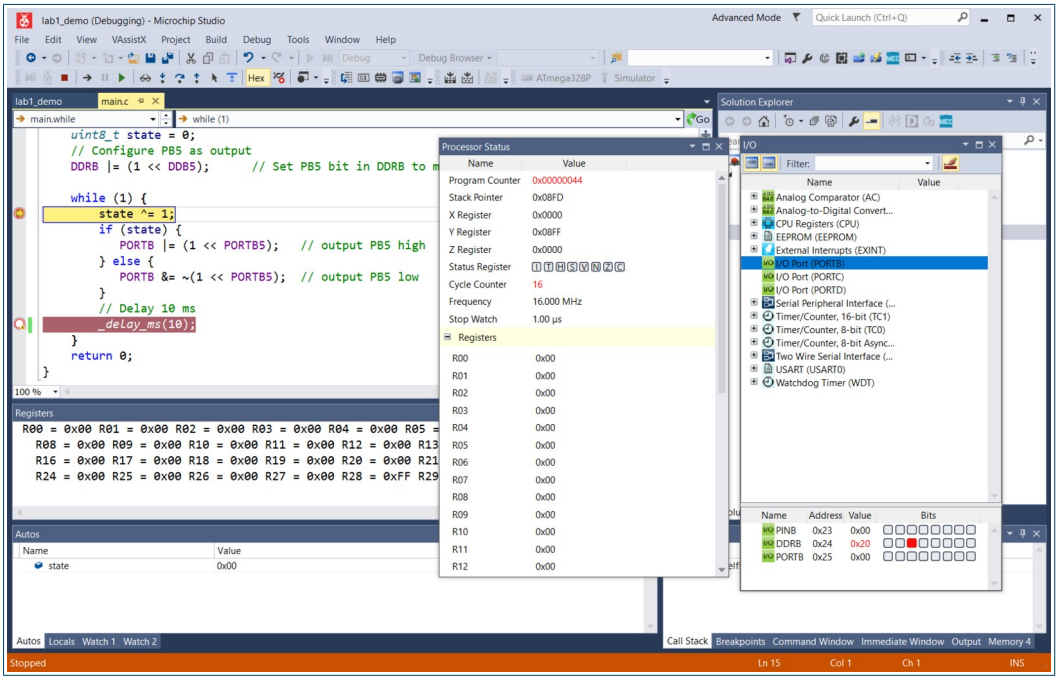

Figure 6: Breakpoint settings and simulated code execution

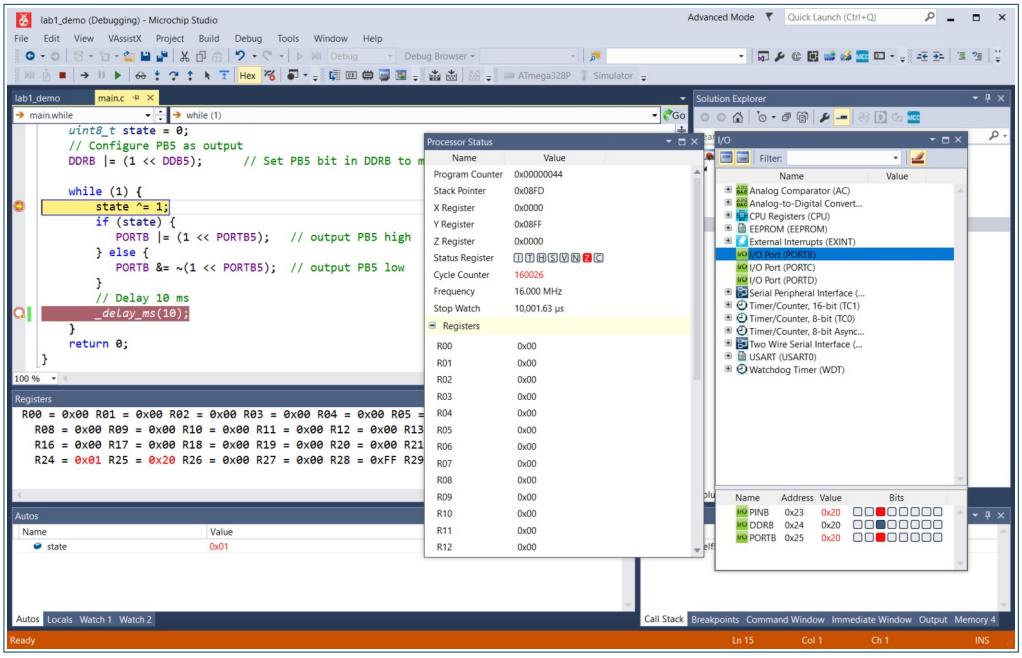

Figure 7: Stepping through the code until reaching the next breakpoint

```
$AVRDUDE_PATH = "C:\Users\$Env:USERNAME\AppData\Local\Arduino15\packages\arduino\tools\avrdude"

$AVRDUDE = "$AVRDUDE_PATH\6.3.0-arduino17\bin\avrdude.exe"

$CONF = "$AVRDUDE_PATH\6.3.0-arduino17\etc\avrdude.conf"

$HEX = "C:\users\sirip\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_demo.hex"

& $AVRDUDE -C $CONF -c arduino -P COM5 -b 115200 -p m328p -D -U flash:w:"$HEX":i
```

Code Listing 2: Command lines for using AVRDUDE to flash the Arduino Uno board

```
Windows PowerShell
PS C:\users\sirip> $AVRDUDE_PATH = "C:\Users\$Env:USERNAME\AppData\Local\Arduino15\packages\arduino\tools\avrdude"
PS C:\users\sirip> $AVRDUDE      = "$AVRDUDE_PATH\6.3.0-arduino17\bin\avrdude.exe"
PS C:\users\sirip> $CONF         = "$AVRDUDE_PATH\6.3.0-arduino17\etc\avrdude.conf"
PS C:\users\sirip> $HEX          = "C:\users\sirip\OneDrive\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_demo.hex"
PS C:\users\sirip> & $AVRDUDE -C $CONF -c arduino -P COM5 -b 115200 -p m328p -D -U flash:w:"$HEX":i

avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: reading input file "C:\users\sirip\OneDrive\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_demo.hex"
avrdude.exe: writing flash (174 bytes):

Writing | ################################################## | 100% 0.05s

avrdude.exe: 174 bytes of flash written
avrdude.exe: verifying flash memory against C:\users\sirip\OneDrive\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_demo.h
ex:
avrdude.exe: load data flash data from input file C:\users\sirip\OneDrive\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_
demo.hex:
avrdude.exe: input file C:\users\sirip\OneDrive\Documents\Atmel Studio\7.0\lab1_demo\lab1_demo\Debug\lab1_demo.hex contains 174 byte
s
avrdude.exe: reading on-chip flash data:
```

Figure 8: Running the AVRDUDE command to upload the .hex file to the Arduino Uno board

Lab 2: Pressing a Push Button to Toggle an LED

Objectives

- Use Atmel Studio 7 or Microchip Studio for AVR to develop firmware for the ATmega328P MCU.
- Write bare-metal embedded C code to perform basic I/O operations on the MCU's I/O ports and pins.
- Use the built-in simulator in Atmel Studio 7 or Microchip Studio for AVR, or the Wokwi Simulator, to simulate and test the source code.
- Use AVRDUDE to upload firmware to the Arduino Uno or Nano board without overwriting the Arduino bootloader.

In this lab we will use the following **I/O pin configuration**.

- **PD2** is used as an input with its internal pull-up resistor enabled.
- **PB5** is used as an output connected to an LED circuit (active-high).

Lab Procedure

1. Create a new C project in **Atmel Studio / Microchip Studio.**

   1.1 Select **GCC C Executable Project**, enter a project name, and choose a directory for the project.
   1.2 Select **ATmega328P** as the target device and click the "OK" button.

   1.3 Open **main.c** and edit the source code using the provided template (**Code Listing 3**).

   1.4 Compile and build the project to generate the output files.

2. Debug the AVR program using the built-in simulator.

   2.1 Start a debugging session by selecting **Debug → Start Debugging and Break** from the menu to begin code execution using the simulator.

   2.2 Set breakpoints on the desired lines of code in **main.c** before stepping through the compiled code (see **Figure 9**).

   2.3 Open the **I/O View**. Set or clear **Bit 2 of PORTD** to simulate changes in the input state during the code debug process (see **Figure 9 and 10**). Step through the code and observe the corresponding changes in **Bit 5 of Port B** and **Bit 2 of Port D**.

3. Upload the .hex file to the **Arduino Uno** using the **AVRDUDE** utility.

   3.1 Connect a push button on the breadboard to the input pin **PD2** of the AVR using an active-low configuration.

   3.2 Measure the signals on the **PD2** and **PB5** pins using a digital oscilloscope with probes **CH1** and **CH2**, respectively.

   3.3 Ensure a common ground is connected between the Arduino board, the push button circuit, and the oscilloscope.

3.4 Press and release the button several times and observe the corresponding input and output signals on the oscilloscope.

4. Revise the code in **main.c** to implement the following function.

   4.1 When the push button (active-low configuration) is pressed and then released, the LED state is toggled only once.

   4.2 Ensure the LED state changes only after the button is released, not during the press.

5. Test the revised code using the Arduino board.

   5.1 Upload the .hex file to the AVR.

   5.2 Press and release the button several times and observe the corresponding input and output signals on the oscilloscope.

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <util/delay.h>  // for _delay_ms()

int main(void) {
 // Configure PD2 as input and enable internal pull-up
 DDRD  &= ~(1 << DDD2);    // Clear PD2 bit in DDRD to make it input
 PORTD |= (1 << PORTD2);   // Enable pull-up on PD2

 // Configure PB5 as output
 DDRB |= (1 << DDB5);      // Set PB5 bit in DDRB to make it output
 while (1) {
   uint8_t value = PIND & (1 << PIND2); // Read PD2 input
   if (value) {
     PORTB &= ~(1 << PORTB5);  // PD2 high, output PB5 low
   } else {
     PORTB |= (1 << PORTB5);   // PD2 low, output PB5 high
   }
   _delay_ms(1); // Delay 1 ms
 }
 return 0;
}
```

Code Listing 2: AVR C code template for **main.c** in Lab 2

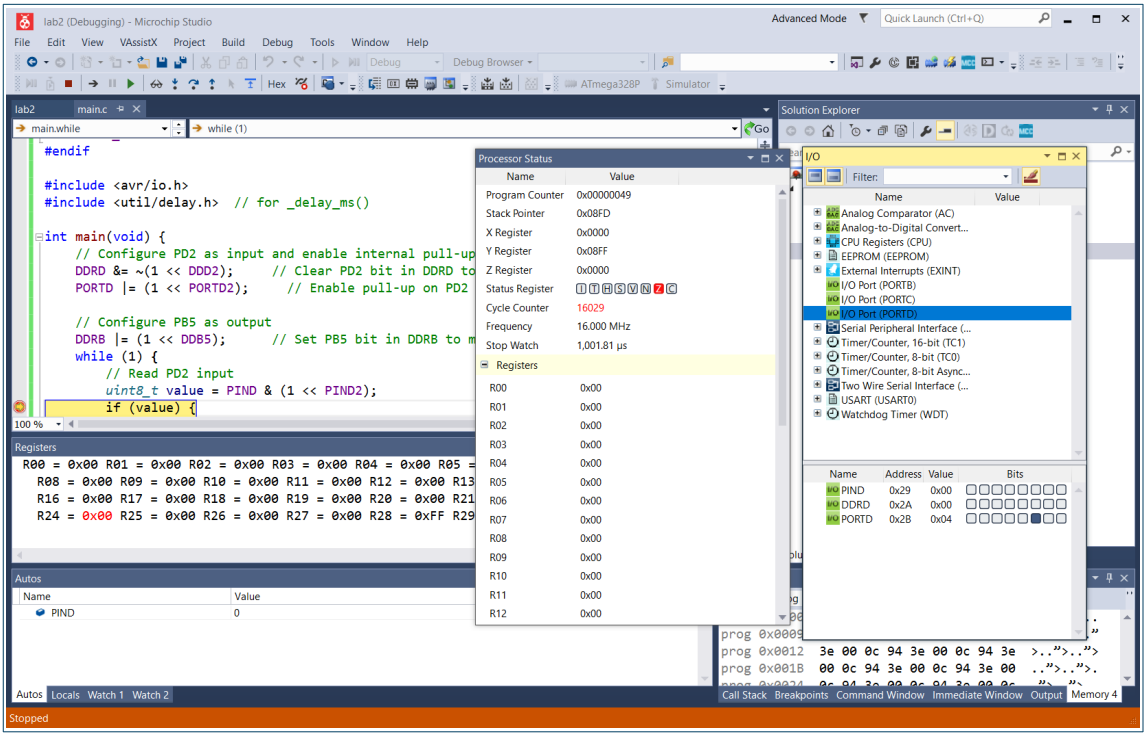

Figure 9: Debug Session (Observing I/O Port D)

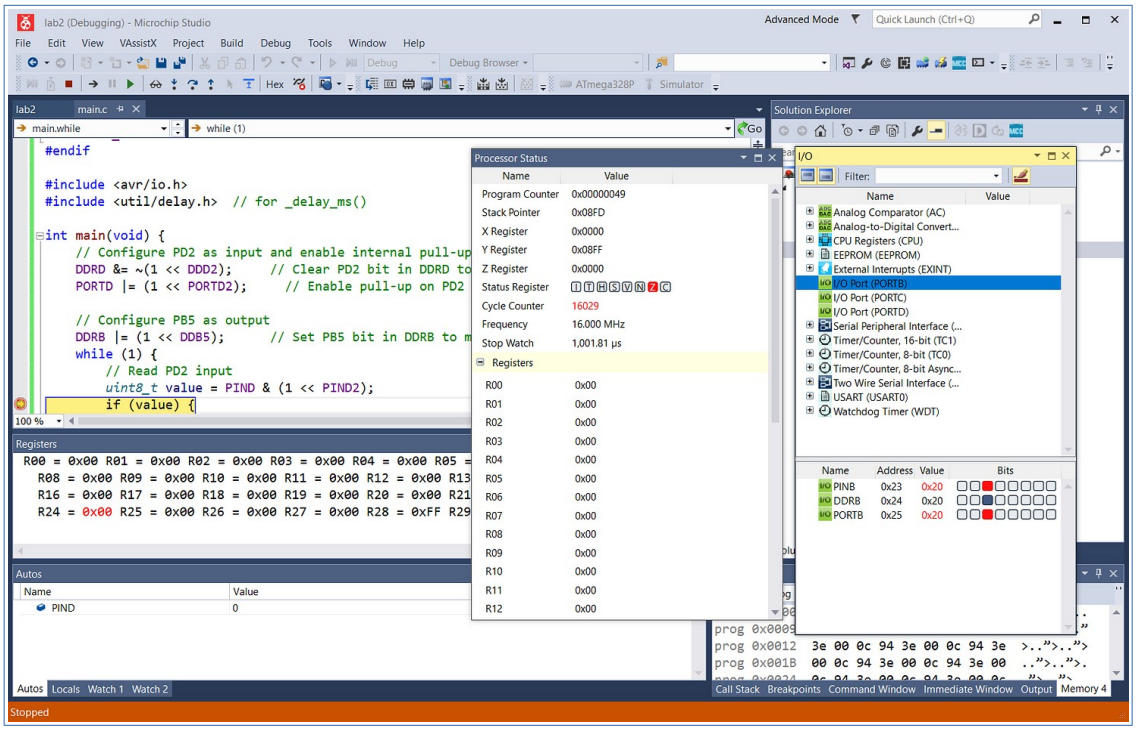

Figure 10: Debug Session (Observing I/O Port B)

Questions

1. Based on the waveforms captured on the oscilloscope, do you observe any **button bouncing effects**? If yes, explain what happens using the captured waveforms. For example, a single button press may cause the LED to toggle multiple times due to bouncing.

2. How can the code be modified to handle or eliminate the button bouncing effect? Implement your proposed solution in the program, test it on the AVR board, and capture the waveforms to verify the improvement.

Lab 3: Pulse Train Detection and Pulse Counting

Objectives

- Write C code for AVR to detect pulse trains and count pulses.
- Use a function generator and a digital oscilloscope to test and analyze the firmware on real hardware.

Lab Procedure

1. Use Atmel or Microchip Studio IDE to write C code for the **ATmega328P** that performs the following tasks:

1.1 I/O Pin Configuration

- Configure **PD2** as an input and enable its internal pull-up resistor.
- Configure **PB5** as an output connected to an LED circuit (active-high).

1.2 Pulse Detection & Pulse Counting

- Continuously read the logic level of the **PD2** input pin.
- Detect logic transitions on **PD2** and count the number of pulses.

1.3 LED Control Based on Pulse Count

- When the number of detected pulses reaches **PULSE_COUNT = 5**, turn ON the LED on **PB5** for approximately **100 milliseconds**, then turn it OFF and the pulse count variable is reset.
    - Note that using a software delay (e.g., _delay_ms()) to control the LED ON time may reduce the responsiveness of the system
- While the LED is ON, any incoming pulses on **PD2** must be ignored (pulse counting is temporarily paused).

2. Create a new AVR project

- Create a new C project in Atmel Studio / Microchip Studio.
- Write your source code in **main.c**.
- Compile and build the project to generate the output files.

3. Upload to the Arduino Uno and test the program

- Use **AVRDUDE** to upload the generated .hex file to the Arduino Uno board.
- Ensure the correct COM port, MCU type, and programmer settings are used.

4. Use a function generator to generate a pulse train

4.1 Use the **RIGOL function generator** in the lab to generate a test signal. Select the **pulse signal type** with the following settings:

- Frequency: **100 Hz**
- Duty cycle: **50% (default)**
- Amplitude (Vamp): **5V**
- Offset (Voffset): **2.5V**
- Ensure that the output signal ranges from **0V to 5V**. Use a digital oscilloscope to verify the signal.

4.2 Enable **Burst mode** and configure:

- Number of pulses (**Ncycles**): **1 to 10**
- Burst period: **500 ms**

4.3 Connect the output signal of the function generator to the AVR pins.

- Connect the pulse-train signal to **PD2**.
- Connect the function generator GND to the GND of the Uno board (**common ground**).
- Use a **suitable oscilloscope trigger mode** to capture the pulses (e.g., **Nth Edge mode** or **Timeout mode** and see **Figures 11 - 13**).

5. Verify that the Arduino board behaves as expected

- Use oscilloscope **CH1** and **CH2** to measure the signals on **PD2** and **PB5**, respectively.
- Vary the duty cycle and the number of pulses of the test signal.
- Capture the waveforms for inclusion in the lab report.

6. Add a push button circuit (active-low configuration) to provides a digital input to the **PD3** pin of the AVR.

- If the button is held pressed (logic LOW), pulse counting must be ignored. In addition, if the LED is turned ON (logic HIGH), it must be turned as quickly as possible if the button is pressed.
- Revise the source code in main.c to implement this behavior. Upload the updated firmware to the target board.
- Capture and save the waveforms (**PD2**, **PD3** and **PD5**) for inclusion in the lab report.

Questions

1. Compare and explain the differences among the trigger modes “**Edge**”, **“Nth Edge**”, and “**Timeout**” on a RIGOL oscilloscope when measuring a pulse-train signal.

2. Explain how the use of a delay function in the program can impact the responsiveness of the system in detecting input changes and updating the output.

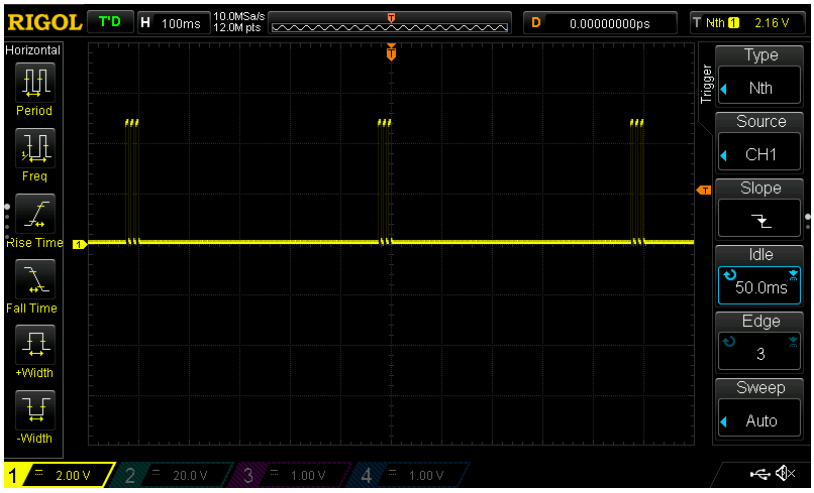

Figure 11: Waveform capture using the Nth Edge mode trigger (Ncycles = 3)

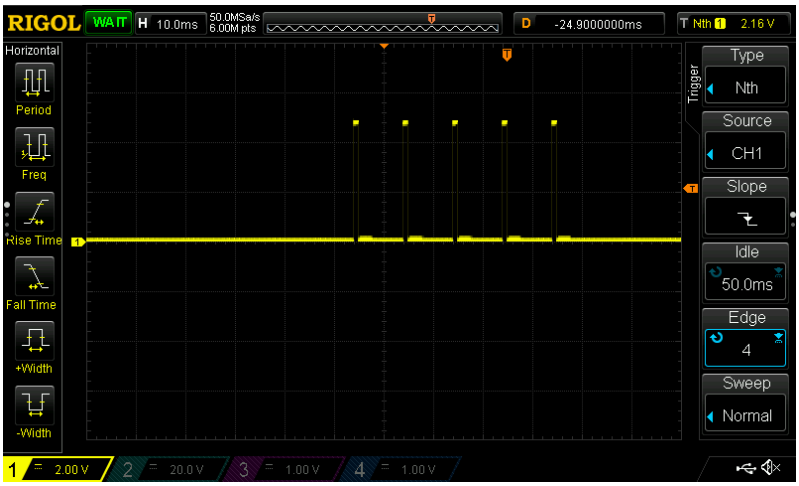

Figure 12: Waveform capture using the Nth Edge mode trigger (Ncycles = 5, Cycle Cycle = 10%)

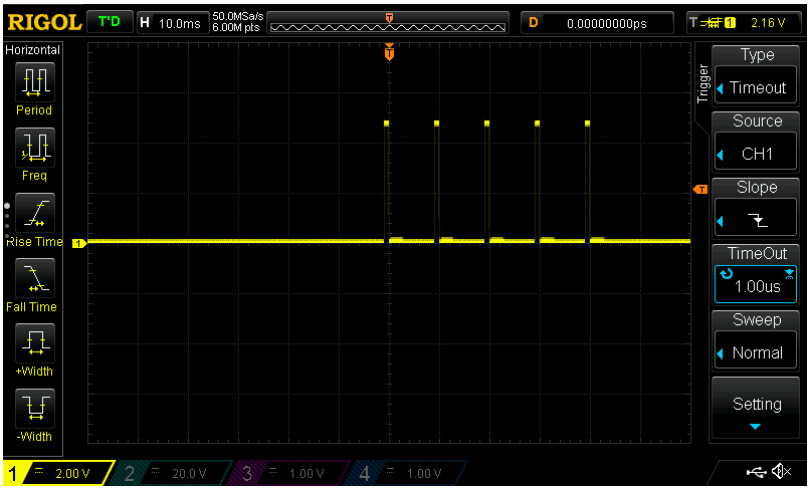

Figure 13: Waveform capture using the Timeout trigger mode (Ncycles = 5, Cycle Cycle = 10%)