CENG 347L Arduino IDE and GPIO

Laboratory 1

Getting Started, GPIO, and Debugging

Zackery Holloway/ Alexis Englund 2/02/25

Purpose:

1. Setting up VS Code with Platform.IO for microcontroller programming.
2. Interfacing with GPIO pins to control hardware components.
3. Utilizing debugging tools to analyze and improve code functionality.

Procedure:

The lab is divided into three key parts:

Part 1: Setting Up the Development Environment

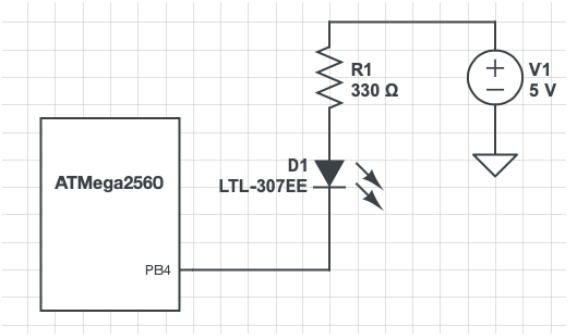
1. Install and configure VS Code with Platform.IO.
2. Import AVR I/O and delay libraries:

```
#include <avr/io.h>
#include <util/delay.h>
```

3. Use the _delay_ms(x) function to control LED blink rates.
4. Load and run Blink.cpp to verify the ability to compile and upload code to the microcontroller, ensuring the LED connected to PB7 blinks at 1Hz.
5. Modify the delay to observe changes in blink rates:
    - 2Hz (1000 ms delay)
    - 10Hz (100 ms delay)
6. Remove delay and document observations.

Part 2: Binary Counter Implementation

1. Develop a 4-bit binary counter using PORTB (PB4 to PB7).
2. Connect LEDs to the high nybble of PORTB with appropriate current-limiting resistors.
3. Understand and implement inverse logic due to the LED wiring configuration.
4. Demonstrate counting from 0 to 15 in binary, with LEDs representing the binary output.



CENG 347L Arduino IDE and GPIO

Laboratory 1

Getting Started, GPIO, and Debugging

Zackery Holloway/ Alexis Englund 2/02/25

Part 3: Debugging Techniques

1. Learn debugging using Platform.IO and Atmel ICE.
2. Set breakpoints and use step-through debugging to monitor changes in PORTB.
3. Address key debugging questions:

- Observing PORTB contents after assignments.
- Analyzing PORTB behavior during code stepping.
- Evaluating differences between single-stepping and continuous execution.

Application:

Part 1: Setting Up the Development Environment

Most parts of this lab were done already in Lab 0 and was mostly repeat.

Blink.cpp was downloaded and ran on the board with the needed include files.

Code:

```
#include <avr/io.h>
#include <util/delay.h>

int main()
{
  //setup PORTB (pin7) as an output
  DDRB |= 0x80;

  //set PORTB to a known state
  PORTB = 0xFF;
  //loop for continuous blinking
  while(true)
  {
    //toggle PORTB
    PORTB ^= 0x80;
    //wait 500 ms
    _delay_ms(500);
  }
}
```

CENG 347L Arduino IDE and GPIO

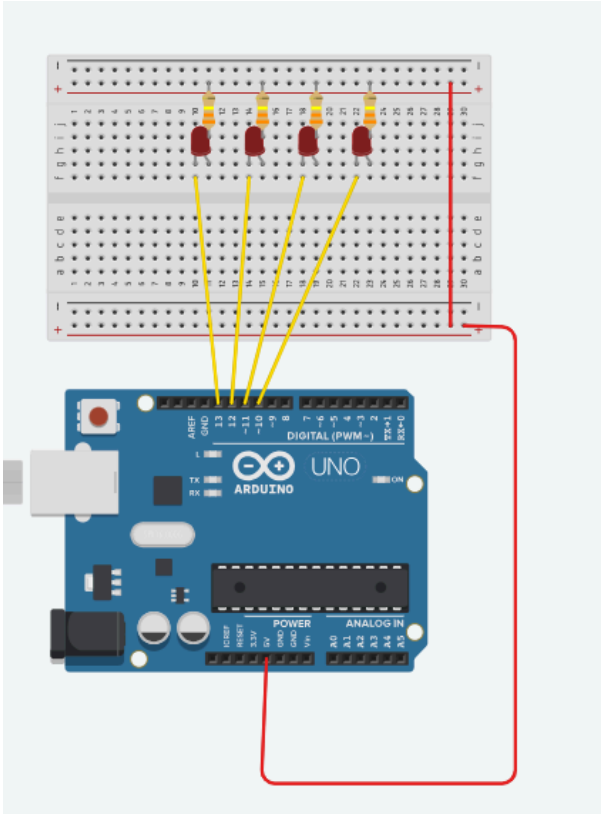
Laboratory 1

Getting Started, GPIO, and Debugging

Zackery Holloway/ Alexis Englund 2/02/25

Part 2: Binary Counter Implementation

Using the diagram provide in the lab we wired the lab to have a low signal act as a high LED. Shown below;



Blink.cpp was modified to allow for the counter.

Code:

```
#include <avr/io.h>
#include <util/delay.h>

int count = 0;

int main()
{
  DDRB |= 0xF0;      //setup PORTB (pin7-4) as an output
  PORTB &= ~0xFF;    //set PORTB to a known state

   while(true)
```

Zackery Holloway/ Alexis Englund 2/02/25

```
  {
    PORTB = (PORTB & 0x0F) | (count << 4);  //toggle PORTB
    _delay_ms(500); //wait 500 ms
    count = (count + 1) & 0x0F; //Update count
  }
}
```

Part 3: Debugging Techniques

A significant portion of the class was spent debugging due to unclear instructions. The lab write-up directed us to “download and follow the debugging tutorial provided under the resources tab on D2L,” but no such tutorial existed. Instead, we found a document titled “Debugging with Atmel ICE and PlatformIO Rev_2,” which was unhelpful as it lacked links to necessary software or documentation.

The document included a warning about setting fuse bits correctly but provided no instructions on how to do so. The TA later informed us that we needed additional software to handle this, but installing it took over 30 minutes and required more than 5 GB of storage. While waiting for the installation, we connected the debugger.

The instructions then prompted us to run the command avarice -4 -r, but doing so in any Windows terminal resulted in an error. The TA clarified that this command requires a Linux environment, though this was not mentioned in the documentation. My lab partner attempted to use her Linux device but was advised to switch to WSL instead. Unfortunately, I couldn’t run WSL on my laptop because virtualization wasn’t enabled in the BIOS, so we used my lab partner’s WSL-compatible computer.

We managed to set the Arduino’s fuse bits, but linking the debugger to WSL proved problematic. Although the command executed, the output was limited to “Defaulting JTAG bitrate to 250 kHz,” indicating that the debugger wasn’t properly interfacing with WSL. The class ended before we could complete any successful debugging.

UPDATE: We learned later that the fuse bits are disabled for jtag bits and that debugging was impossible.

CENG 347L Arduino IDE and GPIO

Laboratory 1

Getting Started, GPIO, and Debugging

Zackery Holloway/ Alexis Englund 2/02/25

Results:

Part 1: Setting Up the Development Environment

The environment was set up and the LED was blinking. TA visually confirmed this but video also attached in drop box.

Part 2: Binary Counter Implementation

The code ran successfully and counted as expected. The TA added a part not in the lab write up to count backwards. This was achieved by subtracting instead of adding. This was verified by the TA visually but also attached in drop box.

Part 3: Debugging Techniques

No progress was made due to issues listed above.

Conclusion:

This lab aimed to set up VS Code with Platform.IO for microcontroller programming, interface with GPIO pins to control hardware components, and utilize debugging tools to improve code functionality. In Part 1, we successfully configured the development environment and verified basic microcontroller functionality by running Blink.cpp. The LED blinked as expected, confirming proper setup. In Part 2, we implemented a 4-bit binary counter using PORTB (PB4–PB7) with LEDs representing binary output. The counter functioned correctly, incrementing from 0 to 15. Additionally, we extended the lab to implement a reverse counter, as suggested by the TA, which also performed as intended.

However, Part 3 presented significant challenges due to unclear instructions and missing resources. The debugging tutorial referenced in the lab materials was unavailable, and the provided documentation lacked critical details for setting up the necessary tools. Software installation delays, compatibility issues with WSL, and hardware interfacing problems prevented us from successfully completing the debugging portion. Overall, while we met the objectives related to microcontroller programming and hardware interfacing, the debugging component highlighted the need for clearer documentation and better preparation for software dependencies.