Lab 1: Git version-control system, AVR tools

Learning objectives

After completing this lab you will be able to:

- Use markdown README files
- Create git repository
- Understand basic structure of C files
- Compile and download firmware to AVR device
- Use breadboard and connect electronic devices to AVR pins

The purpose of this laboratory exercise is to learn how to use the `git` versioning system, write the markdown readme file, learn the basic structure of C code, and how to use development tools to program ATmega328P microcontroller on the Arduino Uno board.

Table of contents

Components list

- Arduino Uno board, USB cable
- Breadboard
- 2 LEDs
- 2 resistors
- Jumper wires
- Logic analyzer

Pre-Lab preparation

1. If you don't have any, create a free account on GitHub.

2. For future synchronization of local folders with GitHub, download and install git. Git is free, open source, and available on Windows, Mac, and Linux platforms. Window users may also need to use the Git Bash application (installed automatically with git) for command line operations.

3. (Optional) Download and install SimulIDE electronic circuit simulator.

4. (Optional) If you have option to use Arduino Uno board and logic analyzer, also download and install Saleae Logic 1.

Part 1: GitHub

GitHub serves as a platform for hosting code, facilitating collaboration, and managing version control. It enables you and your collaborators to work together on projects, retain a history of all prior changes, create distinct branches, and offers a multitude of additional features.

1. In GitHub, create a new public repository titled **digital-electronics-2**. Initialize a README, C template `.gitignore`, and MIT license.

2. Use any available Git manuals, such as Markdown Guide, Basic Syntax and add the following sections to your README file.

    - Headers H1, H2, H3
    - Emphasis (*italics*, **bold**)
    - Lists (ordered, unordered)
    - Links
    - Table
    - Listing of C source code (with syntax highlighting)

3. Use your favorite file manager and run Git Bash (Windows) or Terminal (Linux) inside your home folder `Documents`.

4. With help of Git command, clone a local copy of your public repository.

    > **Important:** To avoid future problems, never use national characters (such as éščřèêö, ...) and spaces in folder- and file-names.
    >
    > **Help:** Useful git command is `git clone` - Create a local copy of remote repository. This command is executed just once; later synchronization between remote and local repositories is performed differently.
    >
    > Useful bash commands are `cd` - Change working directory. `mkdir` - Create directory. `ls` - List information about files in the current directory. `ls -a` - List information aout all files in the current directory. `pwd` - Print the name of the current working directory.

    ```
    ## Windows Git Bash or Linux:
    $ git clone https://github.com/your-github-account/digital-
    electronics-2
    $ cd digital-electronics-2/
    $ ls -a
    .gitignore  LICENSE  README.md
    ```

5. Set username and email for your repository (values will be associated with your later commits):

    ```
    $ git config user.name "your-git-user-name"
    $ git config user.email "your-email@address.com"
    ```

    You can verify that the changes were made correctly by:

```
$ git config --list
```

Part 2: Install and test AVR tools

1. Run Visual Studio Code, follow these instructions and install the PlatformIO plugin.

2. Create a new project `lab1-blink_arduino`, select `Arduino Uno` board, and change project location to your local repository folder `Documents/digital-electronics-2`. Copy/paste blink example code to your `LAB1-BLINK_ARDUINO > src > main.cpp` file.

3. IMPORTANT: Rename `LAB1-BLINK_ARDUINO > src > main.cpp` file to `main.c`, ie change the extension to `.c`.

   The final project structure should look like this:

   ```
   LAB1-BLINK_ARDUINO  // PlatfomIO project
   ├── include         // Included files
   ├── lib             // Libraries
   ├── src             // Source file(s)
   │   └── main.c
   ├── test            // No need this
   └── platformio.ini  // Project Configuration File
   ```

4. Compile and download the firmware to target ATmega328P microcontroller. Go through all the lines of code and make sure you understand their function. Change the delay duration and observe the behavior of on-board LED.

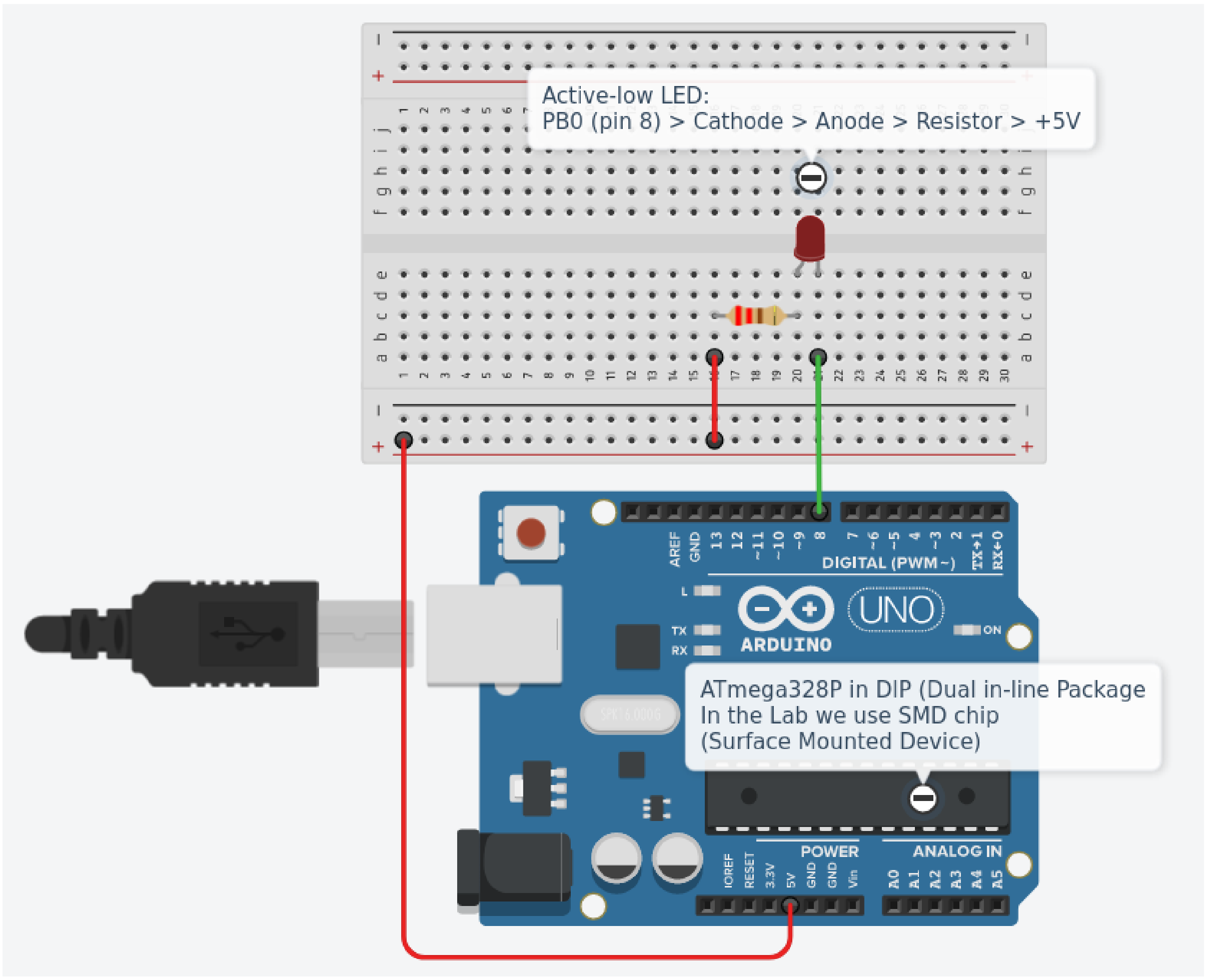
   - See Arduino Uno pinout
   - See Arduino Docs for GPIO / Pin Management

5. Use breadboard, wires, resistor, and a second LED. Connect it to a GPIO pin PB0 in active-low way and modify your code to blink both LEDs.

   - See this breadboard description or that one

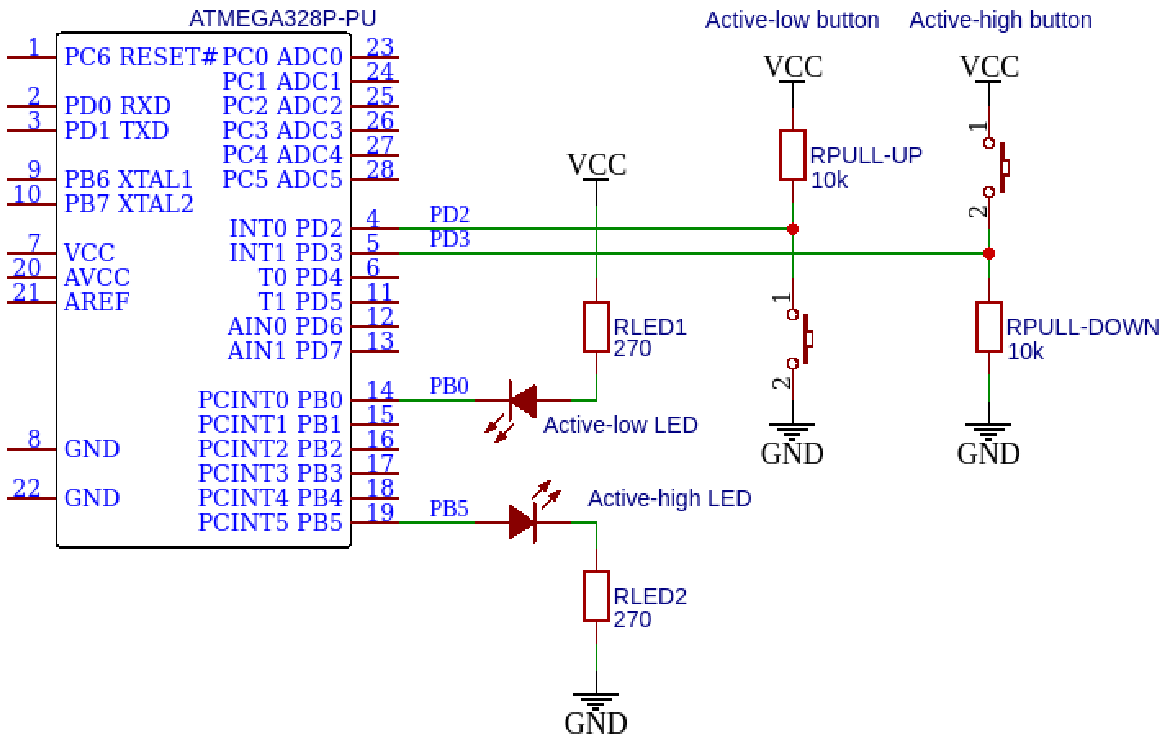
   - See LED resistor value calculation

   - Connection of external LED in active-low way:


Active-low LED:
PB0 (pin 8) > Cathode > Anode > Resistor > +5V
ATmega328P in DIP (Dual in-line Package
In the Lab we use SMD chip
(Surface Mounted Device)
DIGITAL (PWM ~)
ARDUINO
UNO
POWER
ANALOG IN

> **Note:** Picture was created by Autodesk Tinkercad.

- General connections of LEDs and push buttons in active-low and active-high way:

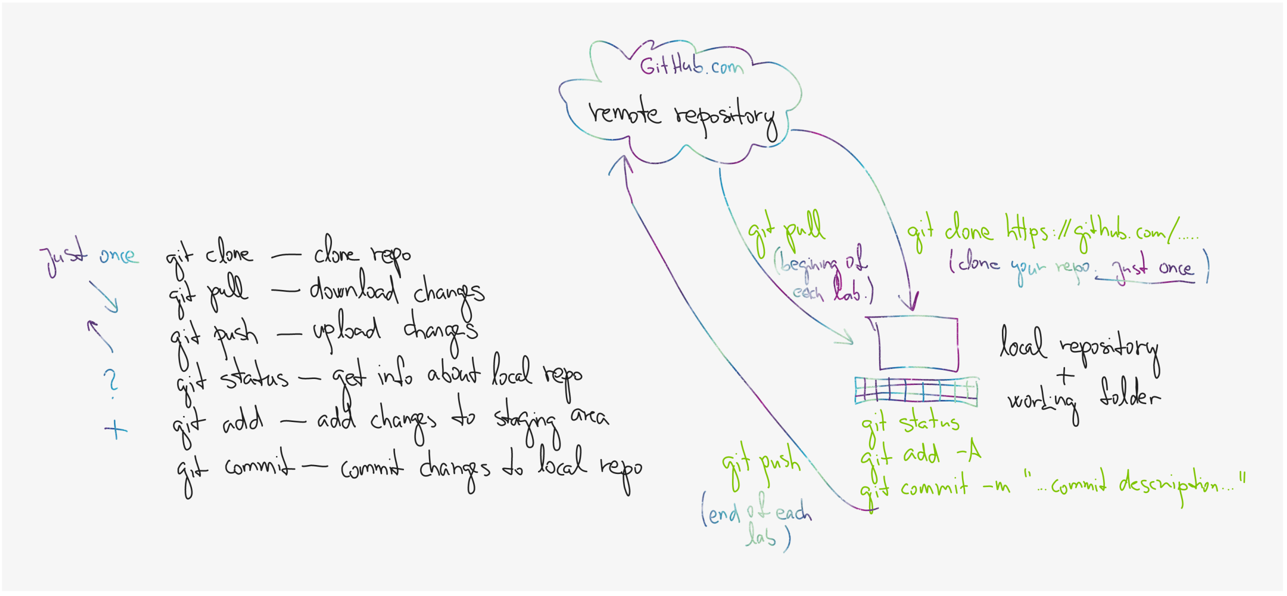


> **Note:** Picture was created by EasyEDA.

6. On breadboard, connect two LEDs and control them by a single output pin PB0. Is it possible to get all commbinations, i.e. ON+ON, ON+OFF, OFF+ON, and OFF+OFF?

7. After completing your work, ensure that you synchronize the contents of your working folder with both the local and remote repository versions. This practice guarantees that none of your changes are lost. You can achieve this by using **Source Control (Ctrl+Shift+G)** in Visual Studio Code or by utilizing Git commands to add, commit, and push all local changes to your remote repository. Check GitHub web page for changes.

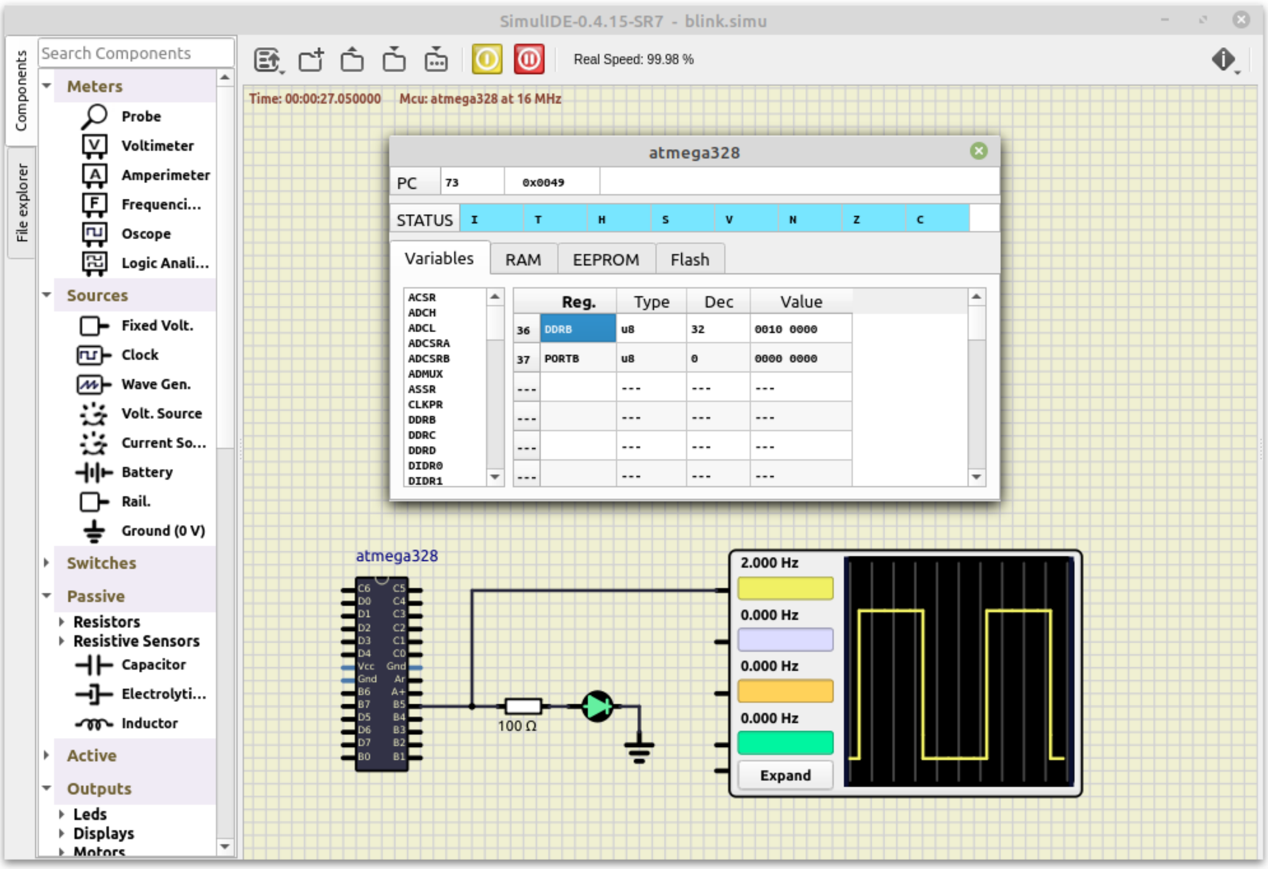
> **Help:** Useful git commands are `git status` - Get state of working directory and staging area. `git add` - Add new and modified files to the staging area. `git commit` - Record changes to the local repository. `git push` - Push changes to remote repository. `git pull` - Update local repository and working folder. Note that, a brief description of useful git commands can be found here and detailed description of all commands is here.
>
> ```
> ## Windows Git Bash or Linux:
> $ git status
> $ git add -A
> $ git status
> $ git commit -m "Creating lab1-blink program"
> $ git status
> $ git push
> $ git status
> ```


GitHub.com
remote repository
git pull (beginning of each lab.)
git clone https://github.com/.....
(clone your repo, just once)
local repository + working folder
git status
git add -A
git commit -m "...commit description..."
git push (end of each lab)
just once
?
+
git clone — clone repo
git pull — download changes
git push — upload changes
git status — get info about local repo
git add — add changes to staging area
git commit — commit changes to local repo

(Optional) Part 3: SimulIDE electronic circuit simulator

1. Run SimulIDE, use online tutorials, and create a circuit with ATmega328 AVR microcontroller.

2. All circuit and control elements are available in the **Components** tab. Use the following components: ATmega328 (**Micro > AVR > atmega > atmega328**), resistor (**Passive > Resistors > Resistor**), LED (**Outputs > Leds > Led**), and GND (**Sources > Ground (0 V)**) and connect them as shown.


SimulIDE-0.4.15-SR7 - blink.simu

Real Speed: 99.98 %

Time: 00:00:27.050000 Mcu: atmega328 at 16 MHz

atmega328

PC 73 0x0049

STATUS I T H S V N Z C

Variables RAM EEPROM Flash

	Reg.	Type	Dec	Value
36	DDRB	u8	32	0010 0000
37	PORTB	u8	0	0000 0000
---		---	---	---
---		---	---	---
---		---	---	---
---		---	---	---

3. Right-click on the ATmega package and select **Load firmware**. In your project folder, find the `*.hex` file that was created by the previous compilation process.

4. Right-click on the ATmega package and select **Open Mcu Monitor** to view internal registers and memory contents. Select **Variable** folder. In the **Reg.** column, type `DDRB` on the first line and `PORTB` on the second.

5. Click to **Power Circuit** button at the top of the window to simulate the project and monitor the LED status and register values. The simulation can be paused with the **Pause Simulation** button and stopped by pressing the **Power Circuit** button again.

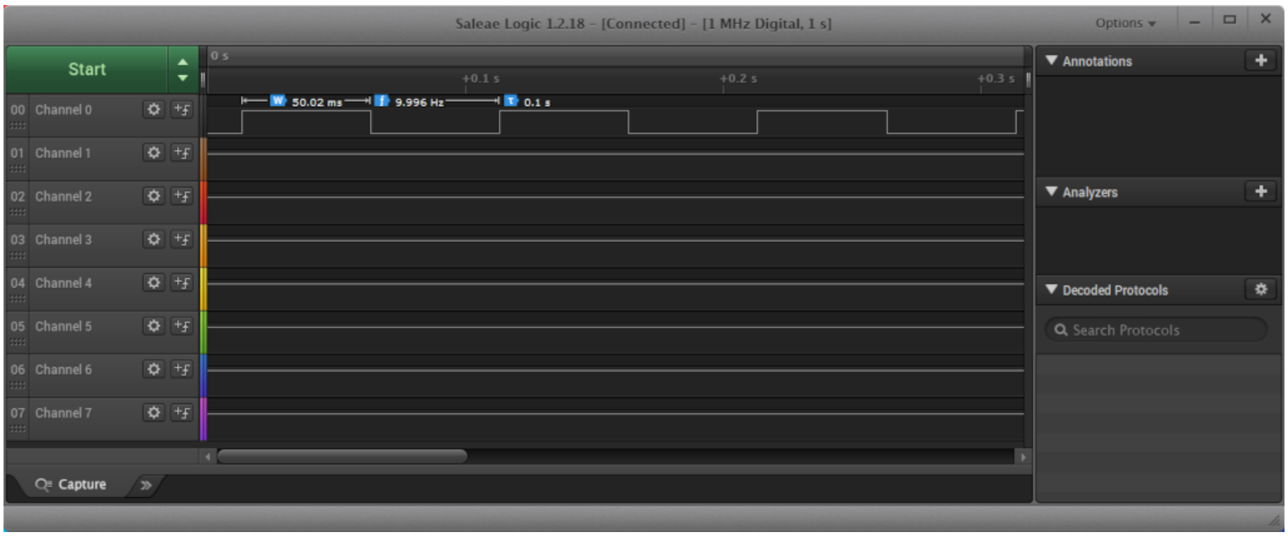
6. You can connect a probe (**Meters > Probe**), an oscilloscope (**Meters > Oscope**), a voltmeter (**Meters > Voltimeter**), or a frequency meter (**Meters > Frequencimeter**) to output B5 and observe the signal.

7. Properties of individual components can be found/changed by right-clicking on the component and selecting **Properties**.

(Optional) Part 4: Logic analyzer

1. Run Saleae Logic software, use wire and connect Channel 0 to Arduino board pin 13 (pin PB5 is connected here), and verify the duration of delay function.

2. To start sampling, press the green button with two arrows, set the sampling rate to 1 MS/s and the recording time to 1 second. Click the Start button.



(Optional) Experiments on your own

1. Install the AVR development tools on your computer.

2. Modify the code from `lab1-blink_arduino` example and build an application that will repeatedly trasnmit the string `PARIS` on a LED in the Morse code. Choose the duration of "dot" and "dash" so that they are visible during the simulation and/or implementation. Note that the proper Morse code timing is explained here.

3. Simulate the Morse code application in SimulIDE.

4. Draw a schematic of Morse code application, i.e. connection of AVR device, two LEDs (one in active-high, second in active-low way), two resistors, and supply voltage. The image can be drawn on a computer or by hand. Always name all components, their values and pin names!

5. Finish all experiments, upload them to your GitHub repository, and submit the project link via BUT e-learning. The deadline for submitting the assignment is the day prior to the next lab session, which is one week from now.

References

1. MIT license

2. Markdown Guide, Basic Syntax

3. GitHub, Inc. Mastering Markdown

4. Tomas Fryza. Useful Git commands

5. Joshua Hibbert. Git Commands

6. Stephen C. Phillips. Morse Code Timing

7. Science Buddies. How to Use a Breadboard for Electronics and Circuits