IDE and Simulator for ATmega328P AVR

A Proposal for a Python Development Project

Written by: Dustin Goodpaster
CS 2100 – Project Proposal
02/27/2026

1. Introduction and Problem Statement

Arduino is an open-source hardware platform widely used by hobbyists, computer science students, engineers and anyone looking to gain a deeper understanding of how computers operate at a fundamental level. While the Arduino community provides their own set of tools for development on the platform, most of these are generally geared towards development in C or C++. This means compilers such as gcc generate the underlying assembly and machine code for the atmega328p microcontroller.

Even though assembly programming provides a deeper control over the processor behavior and memory management, development in pure AVR assembly is not readily accessible in the tools provided by the Arduino community.

At the University of Idaho, Dr. Wilder has developed an educational IDE and simulator called Norfair for use in CS1550: Computer Organization and Architecture. While Norfair is an excellent tool for learning AVR assembly, outside of the classroom it has limitations. Norfair intentionally limits the set of instructions which are supported by the IDE and simulator. For students like myself who have completed the course and remain interested in further development in assembly for the Arduino are limited to the set of instructions which are permitted by the software.

This absence of an instruction-level IDE and simulator which supports the full set of instructions available in AVR is the problem I aim to solve by creating my own version of Norfair.

2. Project Overview

Harmonia-AVR is a Python-based Integrated Development Environment (IDE) and instruction-level simulator for the ATmega328P microcontroller. The purpose of Harmonia-AVR is to provide:

- Full access to the AVR instruction set
- Step-by-step instruction execution
- Real-time display of register, memory, stack, and status flags values
- A lightweight and educational alternative to larger, production-oriented development environments
- Break points for debugging

Unlike traditional Arduino development environments which rely on C compilation, Harmonia-AVR focuses directly on assembly language, allowing users to observe and control execution at the machine level.

The name “Harmonia” is inspired by the Greek goddess of harmony. This reflects the project’s goal: to bring coherence between instruction-level programming, CPU state simulation, and educational exploration of microcontroller architecture.

3. Why I Chose This Project

I chose this project because of my continued interest in AVR assembly after completing CS1550: Computer Organization and Architecture. I wanted a way to continue developing for the Arduino platform using AVR assembly while benefiting from an IDE that enforces syntax rules and provides structured feedback.

Learning Python has also been one of my personal goals, as it is one of the most widely used programming languages in both industry and academia. Python is well suited for rapid prototyping and GUI development, making it an appropriate choice for building a simulation and development environment.

This project allows me to integrate multiple areas of computer science, including user interface design, parsing and lexical analysis, object-oriented programming, recursion, and computer architecture. It directly aligns with my long-term goal of specializing in embedded systems and systems-level software development.

4. Design and Implementation

Harmonia-AVR will be implemented using a modular design that separates the GUI from the parsing and CPU simulation logic. This separation ensures maintainability, clarity of design, and the ability to extend the simulator in future iterations with added design features.

The system will consist of three primary layers:

1. Graphical User Interface (GUI) Layer
2. Control and Execution Layer
3. CPU Simulation Core

4.1 Graphical User Interface Layer

The GUI will be implemented using PyQt. This layer will provide:

- A code editor for writing AVR assembly
- File loading and saving functionality
- Program controls (i.e. step, run, reset)
- A control to toggle display of register window
- A control to toggle display of stack visualization window

The GUI simply acts as a sort of interactive text editor which houses controls and gives visual display feedback for the user. Logic for execution will not live here but this layer will pass text to the execution layer and the execution layer will pass information back for the GUI to use in updating the display state of the program.

4.2 Control and Execution Layer

The control layer will serve as the intermediary piece between the GUI and the CPU simulation logic. It is responsible for the following:

- Parsing written assembly code
- Converting instructions into internal data structures

- Managing main program memory
- Handling breakpoints and step execution
- Handling program execution order

This layer ensures instruction validation and separates execution logic from visual components.

4.3 CPU Simulation Core

The CPU simulation core will model the CPU architecture used in the ATmega328P microcontroller. This includes the following:

- An array or class representing registers 0–31
- A program counter
- A status register
- A stack pointer
- A memory model representing SRAM (stack)
- Functions which emulate AVR instructions

Each instruction will modify the CPU variables to reflect the exact specifications that AVR assembly instructions would perform in the real CPU.

5. Data Flow

The data flow of Harmonia-AVR is as follows:

1. The user writes AVR assembly code in the editor.
2. The parser reads and validates the source code then converts it into the proper data structures for instruction objects.
3. Instructions are stored as program data structure.
4. The execution manager sends instructions to the CPU simulation.
5. The CPU updates registers, memory, stack, and status flags.
6. The GUI is updated to reflect the updated CPU state after each instruction is executed.

6. Timeline and Milestones

Week 1:

- Complete GUI layout and core window components
- Implement register and stack display windows

Week 2:

- Implement assembly parser for core instructions (LDI, MOV, ADD, SUB)
- Develop instruction validation logic

Week 3:

- Implement branching and stack-related instructions
- Add step execution functionality

Week 4:

- Implement run mode and break points
- Add error handling and debugging improvements

Week 5:

- Testing and debugging
- Documentation and README preparation
- Code cleanup and final submission

Links:

https://github.com/azygous123/Harmonia