An Arduino Simulator for Practical Embedded Programming Teaching

Paulo F. Gonçalves
ISEC
Instituto Politécnico de Coimbra
Coimbra, Portugal
a21171940@alunos.isec.pt

João Durães
ISEC
Instituto Politécnico de Coimbra
Coimbra, Portugal
jduraes@isec.pt

***Abstract*—Embedded systems are currently very relevant in many crucial aspects of modern society. Common daily-use objects are becoming smart, increasingly using micro controllers and embedded systems, and the IoT has amplified this tendency. In this context, the need for tools for training professionals for this type of systems is very relevant.**

Many courses address teaching programing for embedded systems using the Arduino platform. This platform offers many advantages but has inherent and unavoidable costs: component setup time, flash wear-out due to the many writes involved, wire connection issues and other electronic details not relevant for introduction to programing and so on. We propose the use of a simulator that completely replaces the need for the physical device in classes while keeping the typical development unchanged.

This paper presents our Arduino simulator platform that we developed for educational purposes, including several components commonly used. We discuss its structure and technological options, and performance aspects. We show how it can be used in an educational context with virtually no changes in the strategy in the classes of introductory embedded programming. We also present the advantages and some use cases that are made possible using our simulator that would be otherwise difficult using the physical device.

***Index Terms*—Arduino, teaching, embedded programming**

I. Introduction

Programming for embedded systems using common platforms such as Arduino [1] is becoming a relevant and increasingly common scenario. Several factors contribute to this situation, and chief among them are the *relevance of embedded systems*, and the *relevance of early programming skills*.

Relevance of embedded systems: Today's society is increasingly becoming dependent on pervasive and ubiquitous computing. Many common objects are now required to be "smart". Many examples can be given, from the typical and now commonplace sensor for doors and automated window blinds, to such things as wearable devices, or the now very relevant devices for climate conditions tracking for agriculture and climate change assessment. In other words, embedded systems are gaining a significant momentum. With this momentum comes the important aspect of training professionals for programming for embedded systems and programming at general.

Relevance of early programming skills: Computer programming has been recognized as an important skill for the population at general. Instead of being seen as a particular skill for a specific set of professionals, it is now seen as an important aspect of literacy for the common citizen [2], [3], and most important, it is also recognized as a tool to promote early cognitive development. To address this, many elementary and middle schools are now including subjects of programming using Arduino [4]. This platform is more affordable than a common desktop computer and can also be used for simple robotic-themed lessons to capture the student attention and devotion.

Teaching programming using Arduino as the computing platform is then an increasingly common practice. However, there are some drawbacks:

- *Setup procedures*: Setting-up an Arduino with smal components attached (even simple ones as LEDs) is error-prone and may introduce some disorder to the classroom and pose a barrier to the acceptance of the technology from students, in particular.
- *Time*: Setup time (and tear-down time) is not negligible, consuming valuable lesson time.
- *Skill impedance*: Typical Arduino setups tend to use small electronic components, such as LEDs, variable resistors, etc. This introduces a skill mismatch where students may also be requires to know some aspects about microelectronics, and this is not a typical or easy skill for young students and forms a barrier (impedance) to learning the intended skill which is programming.
- *Cost*: Although one Arduino setup is relatively cheap, one classroom will require many such setups and the total cost can become rapidly high, posing problems to smaller schools.
- *Wear-down*: The many writes to the Arduino flash may bring an early fail to the equipment, causing an increased cost.

To address the above mentioned issues we propose the use of a Arduino simulator entirely implemented in software. We intend to address all the above issues in the following manner:

- Concerning the *setup procedures* and *required time*, the simulator shall be readily available for use immediately after its launch, requiring no extra actions besides a minimal once-per-installation setup (explained further on).

- The *skill impedance* issue will be solved as no real electronic components will be involved, and circuit setup is made using the mouse to connect virtual lines to virtual components or processor pins.
- *Cost* will not be an issue as the simulator will use the common computers already available at most, if not all, schools. The simulator does not requires heavy processing power and can be hosted in medium-end desktop or servers. The *wear-down* issue will also be non existent, as there is no Arduino hardware involved.

The following aspects of the simulator are worth noting. The first is the **compatibility** of the simulator with the standard IDE used with Arduino. Our simulator is completely compatible with the Arduino IDE as it relies on a new board driver that is simply added and selected via the IDE menu. This board driver will transparently connect the IDE with the simulator, and from the programmer point of view, programming for the simulator and programming for a real Arduino will be the same. The second aspect is **cost** which is *none*, assuming the availability of common computers. Our intention is to make this project fully open-source, to further promote Arduino adoption and universal access to programming skills. The user interface is web-based which means that **no extra skills** will be required to use the simulator. Finally, there are **new uses** available with the simulator that are not possible with Arduino, namely *debug*, *step-by-step execution* and *save and restore of the circuit*. This will be explained further in the paper.

The remainder of the paper is organized as follows: The next section presents a brief overview of the Arduino platform. The features available in the simulator, as well as a circuit example are given in section III. The architecture of our simulator is described in section IV. The current state of the simulator and directions for future are described in section V, and section VI presents some conclusions.

II. Arduino Platform

Arduino is an *open-source* electronic platform composed of a hardware component and a software component that work together [1].

The hardware side is a very simple and compact printed circuit board with an AVR [5] microcontroller (the exact model depends on the Arduino version), a power supply, a serial interface (usually USB) for programming, input and output pins for connection to other electronic devices and a bootloader that allows the device to be programmed without the need for a dedicated programming device. It also has a physical arrangement of input and output pins to allow stacking of shields created specifically for Arduino. We can see the arrangement of components on the printed circuit board in Fig. 1.

The Arduino software component includes an API that contains a number of methods for manipulating all available hardware such as input and output pins, timers, ADCs, and so on. There are also several libraries supporting specific hardware such as LCD displays, ethernet interfaces, among others. It also has an integrated development environment (IDE) that

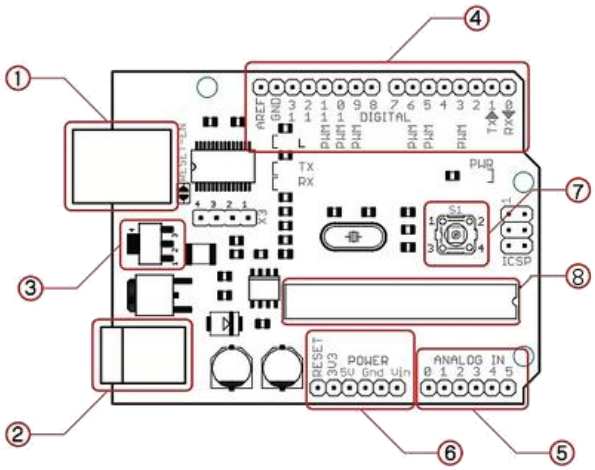

Fig. 1. Arduino Uno diagram. Source: Adapted from [6]

lets developers use an Arduino API-specific editor to write code, compile and program the device, all under the same user interface. Installing the IDE also installs all the tools required for preprocessing, compiling, assembling, linking, and uploading the executable file to the microcontroller.

A. ATmega328P

The Arduino board chosen for proof of concept of our simulator was the Arduino Uno which is equipped with Atmel's ATmega328P microcontroller. It is a microcontroller that implements Harvard architecture, commonly used in embedded systems [7], which has the characteristic of having separate memory and instruction addressing, and 8-bit AVR instruction set. The AVR architecture has 131 instructions, most of which run in just one clock cycle, and 32 general purpose registers all linked directly to the Arithmetic and Logic Unit (ALU), also allowing access to registers in a single clock cycle (Fig. 2).

This particular device has 32Kbytes of FLASH (program memory) which allows storing up to 16384 instructions as the instructions of AVR devices are 16 or 32 bit long. In terms of RAM it has 2Kbytes and 1Kbyte of EEPROM (nonvolatile memory) normally used to store configurations. The clock speed can be up to 20MHz (the Arduino Uno uses a 16MHz clock) and it can reach 20MIPS. In terms of peripherals, it has:

- 27 I/O pins;
- 2 8-bit timer/counters;
- 3 16-bit timer/counters;
- Real-time counter with separate oscillator;
- 10 PWM channels;
- 8 10-bit ADC channels;
- 2 programmable serial USARTs;

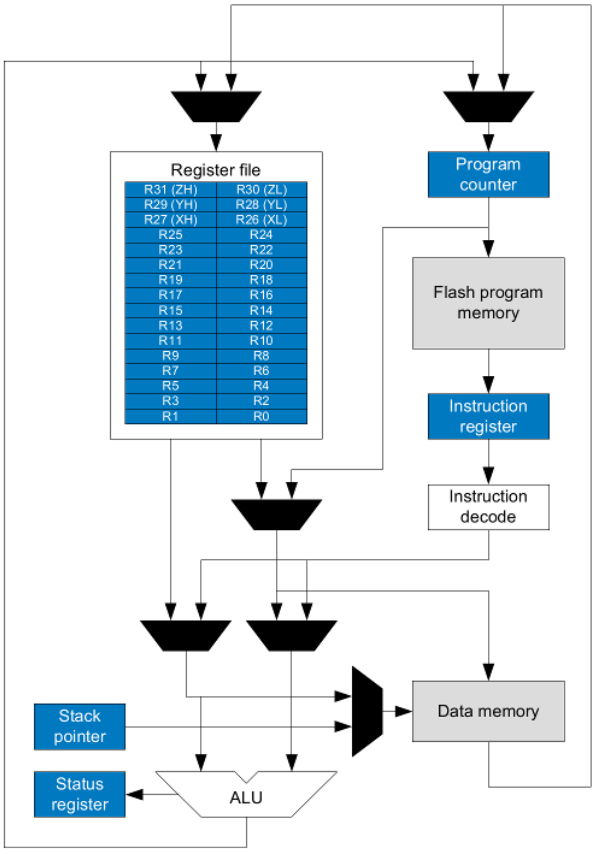

Fig. 2. AVR architecture block diagram. Source [8]

- 2 Master/Slave SPI serial interfaces;
- 2 Two-Wire serial interfaces (I^2C);
- Programmable Watchdog Timer with separate on-chip oscillator;
- One On-chip analog comparator;
- Interrupt and wake-up on pin change.

It has also the following features:

- One internal 8 MHz calibrated oscillator;
- External and internal interrupt sources;
- 6 sleep modes;
- Clock failure detection mechanism;
- Individual serial number to represent a unique ID.

The processor data memory is organized into 4 zones, the first being the General Purpose working registers, the second being the I/O registers, the third being extended I/O registers and the fourth the SRAM (Fig. 3).

The Arduino board also comes with a bootloader that allows updating the firmware from within the application itself.

There are no special instructions to control the peripherals; instead, the peripherals are controlled by writing specific values to the microcontroller I/O registers.

III. Simulator Features

Our simulator currently replicates practically all common used features of the Arduino platform, and also includes the

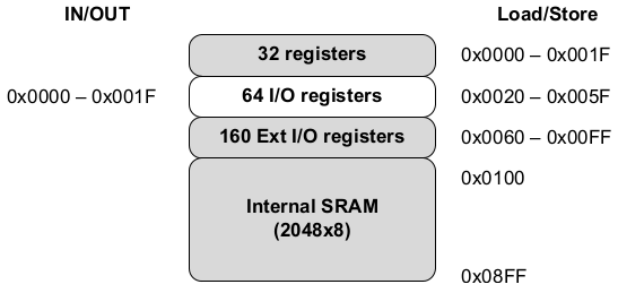

Fig. 3. ATmega328P memory map. Source [8]

ability to simulate several components to build virtual circuits to use with the simulated processor. We start by describing the user interface to give a very high-level idea of the simulator use and capabilities, and then proceed to describe the components and features available.

The user interface has 3 distinct areas as shown in Fig. 4 that also shows a typical circuit. It has a taskbar at the top of the interface where the user can perform tasks such as opening and saving projects, starting, stopping and step-by-step the Arduino microcontroller, using serial monitor, viewing FLASH and SRAM content and opening a window with the source code created in the Arduino IDE where you can place breakpoints and see the execution location if using step-by-step functionality. There is also the possibility to undo and redo actions.

On the left side there is a component palette where the user can choose components to add to the drawing area. There the user can select an Arduino, LEDs, buttons, push buttons, seven segment displays and potentiometers.

Connections are made by clicking with the mouse cursor on one connection point and dragging to another one. Colors help identify voltage potential at the connection points and at the connections themselves. Black indicates 0 volts, red a positive voltage and blue no voltage.

Although the simulator runs at a comparable speed of about 10MHz (depending on host hardware) instead of 16MHz on the Arduino board, it runs at a speed that allows feedback to the user from the circuit, such as blinking a LED, very similar to real hardware, especially if delays are used.

A. Implemented peripherals

The following peripherals have been implemented and should be enough for most microcontroller introductory exercises:

- **GPIOs**: Allows the use of regular input/output pins;
- **Timer0**: For the Arduino *delay* and *millis* functions;
- **USART0**: For the serial port. It allows the user to send and receive data through the serial monitor in the web interface (it doesn't change the pin values of TX on the board or read from RX);
- **ADC**: Allows the conversion of analogue values. There is a potentiometer that the users can use to send analogue

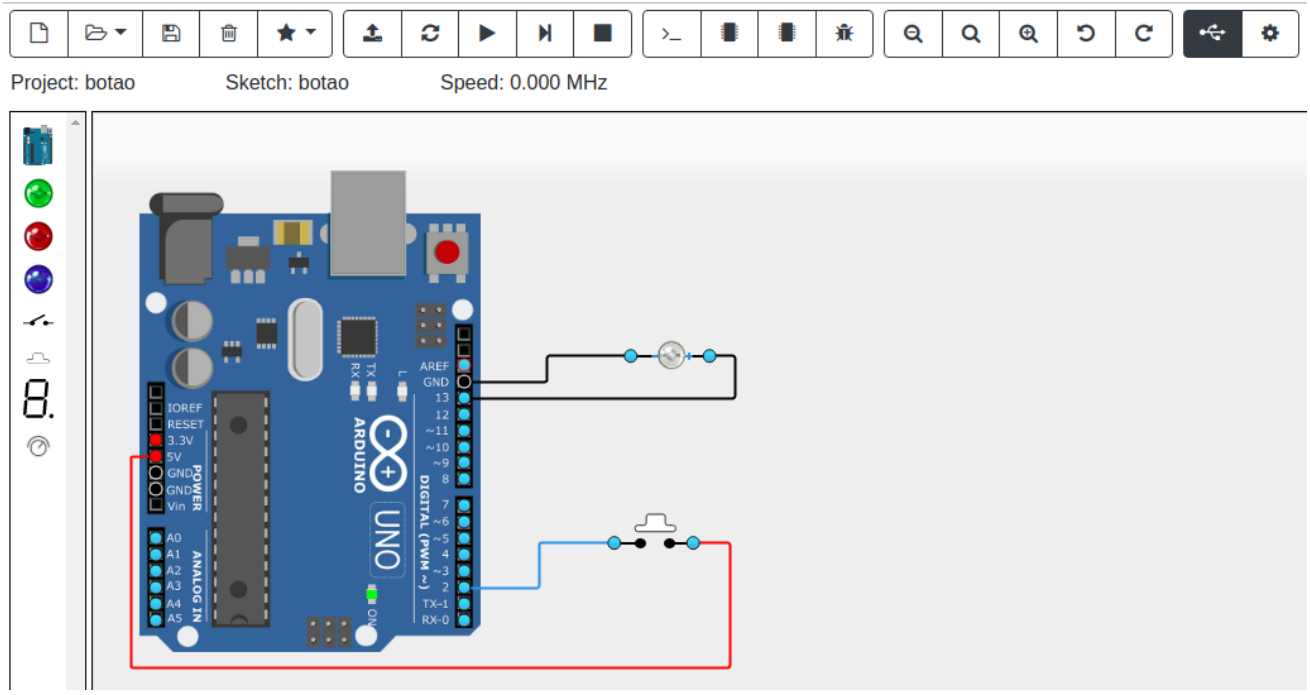

Fig. 4. Interface

values to the system. The *analog ref* pin can be set and it's value is used by the ADC.

We decided not to implement peripherals that cause very fast variations in the pins because this functionality causes a long delay in the propagation of the information from the server (where the simulator is) to the client (where the user interface is), causing the simulation to be slow and therefore no longer practical. These peripherals are PWM and all serial communications like USART, SPI and I^2C. The USART0 port has been implemented but communication is redirected to the serial monitor in the web client instead of the regular pins.

B. Circuit components

There are some basic components available for the user to use in the construction of the electronic circuit:

- **LEDs**: There are 3 different colors, green, red and blue. When they are off they stay gray, when on they change to their respective color. The simulated LEDs have polarity like the real ones.
- **Button**: It's a regular button that the user can click with the mouse to open or close the circuit.
- **Push Button**: It's a button that the user can press with the mouse to close the circuit and that opens when the user releases the mouse.
- **7 Segment Display**: This is composed by LEDs arranged in a format that allows them to represent numbers. They have 8 inputs (7 for the number plus a decimal point) and a common cathode.
- **Potentiometer**: This component has 3 terminals, 2 inputs and 1 output. The output is the relative difference between the inputs. It can be used to input an analog value to the analog pins of the Arduino board.

C. Debugging

The simulator allows some debugging features not available in the real Arduino board. One of these features is the ability to pause or resume the microcontroller execution at any time by pressing a button. The practical effects are as if the clock signal on the real device is stopped while maintaining the entire internal state of the device. This allows developers to inspect the SRAM memory to see what values it contains and also to look at the FLASH memory with the decoded instructions and to know which one was last executed.

The simulator also allows developers to see which source code lines are related to which processor instructions. This is done by selecting an address in the *FLASH memory window* or a code line number in the *source code window* (when an ELF [9] file was loaded, that has information about the source files), to insert breakpoints, which when executed, automatically pauses the simulation.

When the simulation is paused, the user can execute one instruction at a time (step-by-step) and track changes to SRAM values in the *SRAM watch window* and follow the instructions that are executing in the *FLASH* and *source code watch windows*. The mapping between the executing address and the line in the source code is taken from the DWARF [10] format present in the ELF file.

D. Other features

Other noteworthy features are:

- The ability to load binaries in Intel HEX and ELF formats, with the ELF format taking advantage of the information present in the file to aid debugging.
- The ability to save projects for later use and further development, including not only the program but also the circuit.
- The perfect integration with the Arduino IDE allowing users to program the simulator as if it were a real board.
- The ability to load binary files directly in the web client interface in case users want to use another compiler not integrated in the Arduino IDE.
- The serial port monitor is where the user can view the data sent by the program running in the simulator to the USART0 device and also send data that the program can handle. It has similar usage to the Arduino IDE serial port monitor.

IV. Simulator Architecture

We decided to use a web environment in order to lower the costs of installation, maintenance and accessibility. This type of environment also allows to use workstations with less computational power as the workstations used by the developer will be responsible for just the user interface and circuit management, while the microcontroller simulation and instruction execution is hosted in a more powerful server.

In terms of accessibility, the ability to access the simulator via HTTP allows students to work on the system anywhere they have Internet without requiring unusual settings as it might have to be if it were a traditional client/server application with the requirement to use their own TCP/IP ports.

The simulator is organized as a a typical web client/server system. The server hosts the simulation logic and mechanisms and can serve multiple independent simulations at a time, depending on the computing power. The client handles all the user and IDE interaction. The user interface is based on common web technology and is served to the user through common web-browser such as firefox or chrome. Fig. 5 depicts the modules composing the simulator, which are:

- **Microcontroller simulator**: It is in this module that all the features of the microcontroller are implemented, namely the AVR Instruction Set, the microcontroller peripherals, FLASH and SRAM. It is this module that executes the microcontroller code, exposes methods for changing the value of the pins and throws events when their state is changed internally. This module is designed to be easy to implement other microcontrollers from the same Atmel family. ISA, FLASH, SRAM and peripherals code can be used on processors other than the ATmega328P that was implemented.
  This module is a separate project and its features can be used in other applications.
- **Web server**: This module manages all users in the application, maintains simulation instances, and links the simulation to it's user client and *programmer tool*. It is also the responsibility of this module to store user-created projects and all data relating to those projects.
- **Web client**: The web client is where the user can create a project to simulate. The user has a drawing area available where he can add an Arduino and various electronic components and make connections between them. All the simulator functionality can be assessed using the client: start the simulation, pause it, execute step-by-step, analyze FLASH and SRAM memory contents, add or remove breakpoints, etc.
  It is also the client's responsibility to try to periodically connect to the *programmer tool* software (board manager plugin in the Arduino IDE) and transfer to it the data necessary for it to be able to connect directly to the web server.
- **Programmer tool**: This module is a software that is installed in the Arduino IDE and replaces the device programming program (avrdude [11]). In this case instead of programming a real device the binary is sent to the web server so that it can perform the simulation. It's included in the board driver that is installed in the Arduino IDE.

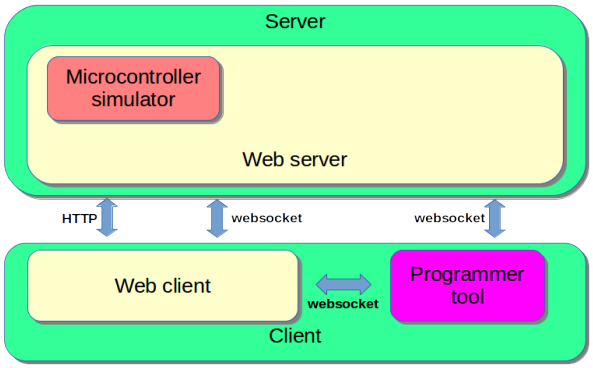

Fig. 5. Architecture

A. Virtualization technique

There are several techniques for virtualization (simulation) and we conducted a comparative study to choose the best virtualization technique to implement the simulator, specifically, to understand which one would result in the best performance (speed of simulation). Of the virtualization types presented by Smith and Nair [12], the one that best fits for a device like Arduino is Emulation. In this case we have a different ISA (Instruction Set Architecture) between *guest* and *host* and we just want to execute one process (the firmware we have prepared for the microcontroller).

From the techniques available to apply in emulation, we tested and compared the performance of one implementation of interpretation with one implementation of compiled simulation. Our results [13] suggest that, at least for this scenario, the best technique interpretation.

B. Arduino Board for programming

We could have chosen to use simple file upload to load the Arduino executable into the web application, but this has

a problem: when using a real Arduino the user loads the executable directly through the IDE interface, and we do not want to interfere nor change the developer tools and user habits.

In order to maintain this aspect of IDE usage, we decided to develop a new *Board* [14], a kind of plugin for the Arduino IDE, which allows the IDE to be expanded to new hardware. In this case this *board* defines a new type of Arduino in IDE **Board Manager**, "**Arduino Uno Simulator**". The user just has to select from an IDE menu which device he wants to use, if the real one or the simulated one.

This new *board* has to be installed in the IDE but it is extremely easy to do and only needs to be done once in each workstation.

The **board** extends the existing Arduino Uno board, allowing us to maintain the same compiler by defining only a new programmer tool.

The new *programmer tool* was written in Java because the Arduino IDE is also a Java application and has Java JRE integrated which allows us to ensure that the *programmer tool* will work on all architectures where the IDE works, otherwise a new tool would have to be built for each architecture where the IDE works.

Since the simulator is a web application, a challenge in creating the *programmer tool* was knowing where to send the binary produced by the compiler. One possibility would be to have a single server and then its address could be hardcoded in the *programmer tool*. This scenario is not desired because the processing power provided would soon be exhausted. The best way would be to have multiple server software installations, for example, each school having a server exclusively for its students. Even if that was the case (a single server) we would still have the problem of knowing what simulation each *programmer tool* upload belongs. To overcome this, an HTTP server has been deployed on the *programmer tool* itself, which the client web application can connect to as it can always be accessed on localhost, and transfers to the *programmer tool* the information it needs to load the binary on the server where the simulation is being performed. This information is the HTTP address of the server and the user session identifier of the web application.

C. Circuit simulation

The simulation of the Arduino board is done on the server but the simulation of the electronic circuit connected to the board is done on the client. This approach was chosen because the electronics simulation is more basic and can be easily done in JavaScript in the client (browser). This also allows the development of electronic components and electronic simulation in general to be built on a separate project which facilitates development. Thus, communication between client and server, in terms of electronic signals, is limited to the pins of the Arduino board which also makes connecting the two system components easier.

V. Features and Work in progress

Some peripherals such as EEPROM memory and some interrupt types are still being implemented.

The project is still in a validation phase and tests are planned where high school students from the Information Technology area of Avelar Brotero High School in Coimbra will be presented to the simulator in the context of their class.

The goal is for some students to be taught in the simulator while others will be part of a control group. Based on some metrics such as the solving time of the proposed exercises, evolution of student grades, student and teacher satisfaction surveys, it is intended to verify if the simulator has the capacity to be used as a pedagogic tool and if it may even have advantages over the use of real Arduino boards.

These field tests may also provide relevant suggestions that may lead to some features being reformulated or new ones introduced.

VI. Conclusions

All the basic Arduino IDE examples that use the peripherals implemented in the simulator can be run.

Our simulator replicates the functionalities present at the Arduino Uno and required for training programming for the Arduino platform. Using the simulator it is possible for students to train outside the classroom as there is no need to have the hardware with them. They can easily take homework and thus practice for more hours than class time would allow them.

The simulator also allows to Debug the created programs which is not possible with the commonly used tools (just the Arduino IDE) or without making physical modifications to the Arduino board.

At this time the number of simultaneous simulations is limited by the number of cores available on the web server since each simulation fully occupies a core. As future work we intend to separate the microcontroller simulation to a process from the web server into other servers but under remote control of the web server. This will allow us to have multiple execution nodes controlled by the same web server which will give us virtually unlimited scalability of processing power.

At this point we believe that this simulator will provide a useful teaching and training environment. We are currently starting a test in a real school and relevant results of real-use by students will be available.

References

[1] Arduino.cc. (2018) Arduino - home. [Online]. Available: https://www.arduino.cc/

[2] M. Prensky, "Programming is the new literacy," *Edutopia magazine*, 2008.

[3] A. V. and, "Understanding computer programming as a literacy," *Literacy in Composition Studies*, vol. 1, no. 2, pp. 42–64, Oct. 2013. [Online]. Available: https://doi.org/10.21623/1.1.2.4

[4] F. Agatolio and M. Moro, "A workshop to promote arduino-based robots as wide spectrum learning support tools," in *Robotics in Education*. Springer, 2017, pp. 113–125.

[5] Atmel, "AVR Instruction Set Manual," 2016.

[6] K. kumar C S, C. K.V, N. A, M. B, and I. Appaji, "Vehicle speed monitoring system using arduino and speed sensor," *IJRDO - Journal of Computer Science Engineering (ISSN: 2456-1843)*, vol. 3, no. 2, pp. 33–40, Dec. 2017. [Online]. Available: https://www.ijrdo.org/index.php/cse/article/view/396

[7] A. Francillon and C. Castelluccia, "Code injection attacks on harvard-architecture devices," in *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*. ACM Press, 2008. [Online]. Available: https://doi.org/10.1145/1455770.1455775

[8] Atmel, "ATmega328/P Datasheet," 2018.

[9] T. I. S. Committee *et al.*, "Executable and linkable format (elf)," *Specification, Unix System Laboratories*, vol. 1, no. 1, 2001.

[10] M. J. Eager, "Eager consulting," *Introduction to the DWARF debugging format*, 2012.

[11] (2019) Avrdude - avr downloader/uploader. [Online]. Available: https://www.nongnu.org/avrdude/

[12] J. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, May 2005. [Online]. Available: https://doi.org/10.1109/mc.2005.173

[13] P. F. Goncalves, J. Bernardino, and J. Duraes, "Virtualization technologies for arduino simulation," in *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.23919/cisti.2019.8760727

[14] Arduino. (2019) Arduino ide 1.5 3rd party hardware specification. [Online]. Available: https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5-3rd-party-Hardware-specification