isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

Um simulador de Arduino

Autor

Paulo Alexandre Figueiredo Gonçalves

Orientador

João António Pereira Almeida Durães

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, julho, 2020

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Um simulador de Arduino

Relatório de Trabalho de Projeto para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

Paulo Alexandre Figueiredo Gonçalves

Orientador

João António Pereira Almeida Durães

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, julho, 2020

Esta dissertação é dedicada à minha esposa
pela paciência que teve nestes tempos
e aos meus pais que com certeza ficarão
muito orgulhosos por esta conquista

Abstract

The learning of embedded systems and microcontrollers in particular, is becoming very important in schools, both in high school and higher education, among others, due to the growth of Internet of Things (IoT). In this context, the need for tools for the training of professionals for this type of systems is very relevant. The Arduino Platform, given its simplicity, is increasingly being used as a central component in this area. However, there are several aspects to consider that can be seen as disadvantages for use in the classroom, such as maintenance of equipment, wear down of devices or the usual misuse by those who are learning, among others. To mitigate these negative aspects, an Arduino simulator is presented, specifically designed for the teaching area. With this work, we propose to improve the usability, cost and efficiency of the class, allowing for improvements and even new forms of use and benefits for learning. We designed, implemented and tested the simulator in a real environment, and concluded that it has a very positive impact on the efficiency of class time without any observed negative impact on the other aspects.

Keywords: Arduino, AVR, Education, Java, Simulator, Virtualization

Resumo

A aprendizagem de sistemas embebidos e microcontroladores em particular, está a tornar-se muito importante nas escolas, tanto no ensino secundário como no ensino superior, entre outros, devido ao crescimento da *Internet of Things* (IoT). Neste contexto, a necessidade de ferramentas para a formação de profissionais para este tipo de sistemas é muito relevante. A Plataforma Arduino, dada a sua simplicidade, está ser cada vez mais usada como um componente central nesta área. No entanto existem vários aspectos a considerar que podem ser vistos como desvantagens para uma utilização na sala de aula, como por exemplo a manutenção dos equipamentos, o desgaste dos dispositivos ou a habitual má utilização por parte de quem está a aprender, entre outros. Para mitigar estes aspectos negativos apresentamos um simulador de Arduino pensado especificamente para a área do ensino. Com este trabalho, propomos melhorar a usabilidade, o custo e a eficiência da aula, permitindo melhorias e até novas formas de uso e benefícios para a aprendizagem. Desenhámos, implementámos e testámos o simulador em ambiente real, e concluímos que tem um impacto muito positivo na eficiência do tempo da aula sem qualquer impacto negativo observado nos restantes aspectos.

Palavras-chave: Arduino, AVR, Educação, Java, Simulador, Virtualização

Agradecimentos

Gostaria de agradecer em primeiro lugar ao meu orientador, João Durães, por todo o apoio, orientação e motivação que me deu e principalmente pela paixão que demonstrou por este projecto desde o primeiro dia.

Ao professor João Sá da Escola Secundária Avelar Brotero, à professora Anabela Coelho do Agrupamento de Escolas de Pombal e aos seus alunos pela possibilidade de usar as suas aulas para testar e validar a utilização do simulador desenvolvido.

Ao Fikalab, e em especial ao Gonçalo Silva, pela extraordinária ajuda que deu na divulgação do projecto.

Ao José Silva, meu chefe e amigo, a quem posso agradecer o incentivo para ir completar mais um ciclo de estudos da minha vida académica.

Finalmente quero agradecer também a todos os meus colegas de Mestrado com quem partilhei cerca de dois anos da minha vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de listagens de código

Definições e Acrónimos

Acrónimos

ADC Analog-to-Digital Converter. 13, 51, 71, *Definição:* ADC

AJAX Asynchronous JavaScript and XML. 57, *Definição:* AJAX

ALU Arithmetic Logic Unit. 11, *Definição:* ALU

API Application Programming Interface. 13–16, 36, 50–52, 54, 56, 59, 71, *Definição:* API

ARM Advanced RISC Machine. 28, *Definição:* ARM

ASCII American Standard Code for Information Interchange. 67, *Definição:* ASCII

CPU Central Processing Unit. 5, 21, 32, 34, 43, 51, 82, *Definição:* CPU

CSS Cascading Style Sheets. 55, 62, *Definição:* CSS

DNS Domain Name System. 20, 57, *Definição:* DNS

DOM Document Object Model. 55, 62, *Definição:* DOM

EEPROM Electrically Erasable Programmable Read-Only Memory. 11, 45, *Definição:* EEPROM

ELF Executable and Linkable Format. 43, 52, 55, 57, 68, 73, *Definição:* ELF

GDB GNU Debugger. 15, 16, *Definição:* GDB

GPIO General-purpose input/output. 11, 51, 57, *Definição:* GPIO

HTML HyperText Markup Language. 41, 55, 62, *Definição:* HTML

HTTP Hypertext Transfer Protocol. 17, 54, 55, 57, 59, *Definição:* HTTP

I/O Input/Output. 12, 24, 45, 51, *Definição:* I/O

I²C Inter-Integrated Circuit. 12, 51, *Definição:* I²C

ICE In Circuit Emulation. 27, *Definição:* ICE

IDE Integrated Development Environment. 1, 5, 6, 13–19, 41, 43, 45, 52, 54, 57, 61, 62, 65, 70, 73, 74, 83, 86, 89, *Definição:* IDE

IoT Internet of things. 1, *Definição:* IoT

IP Internet Protocol. 20, 54, *Definição:* IP

ISA Instruction Set Architecture. 18, 27, 28, 30, 32–34, 38, 42, 50, 75, *Definição:* ISA

J2EE Java 2 Platform Enterprise Edition. 17, *Definição:* J2EE

JAR Java Archive. 21, *Definição:* JAR

JDBC Java Database Connectivity. 56, *Definição:* JDBC

JIT Just-in-Time Compilation. 24, 38, *Definição:* JIT

JRE Java Runtime Environment. *Definição:* JRE

JSON JavaScript Object Notation. 59, *Definição:* JSON

JSP Java Server Pages. 55, 56, 62, *Definição:* JSP

JVM Java Virtual Machine. 37–39, 54, *Definição:* JVM

LCD Liquid-Crystal Display. 13, 14, 77, *Definição:* LCD

LED Light-emitting diode. 1–3, 6, 71–73, 83, *Definição:* LED

MIPS Microprocessor without Interlocked Pipeline Stages. 28, *Definição:* MIPS

NASA National Aeronautics and Space Administration. 83, *Definição:* NASA

NASA-TLX NASA Task Load Index. 83–85, B1, *Definição:* NASA-TLX

NAT Network Address Translation. 54, *Definição:* NAT

PDF Portable Document Format. 34, 42, *Definição:* PDF

PowerPC Performance Optimization With Enhanced RISC – Performance Computing. 28, *Definição:* PowerPC

PWM Pulse-width modulation. 12, 51, 52, 78, *Definição:* PWM

SHA Secure Hash Algorithm. 62, *Definição:* SHA

SO Sistema Operativo. 15, 22–26, *Definição:* SO

SPARC Scalable Processor Architecture. 28, *Definição:* SPARC

SPI Serial Peripheral Interface. 12, 51, *Definição:* SPI

SRAM Static Random Access Memory. 12, 18, 21, 32, 42, 43, 45, 48, 50, 60, 67, 73, 74, *Definição:* SRAM

SVG Scalable Vector Graphics. 62, 63, *Definição:* SVG

TCP/IP Transmission Control Protocol/Internet Protocol. 17, 54, *Definição:* TCP/IP

URL Uniform Resource Locator. 57, 61, 62, 65, 70, *Definição:* URL

USART Universal synchronous and asynchronous receiver-transmitter. 12, 51, *Definição:* USART

USB Universal Serial Bus. 9, 13, 14, 27, *Definição:* USB

VLIW Very-Long Instruction Word. 28, *Definição:* VLIW

VMM Integrated Development Environment. 24–26, 28, *Definição:* VMM

WAR Web Application Archive. 21, *Definição:* WAR

Definições

ADC *Analog-to-Digital Converter* (Conversor Analógico-Digital) é um sistema que converte um sinal analógico, como um som capturado por um microfone ou luz capturada por um sensor de uma câmara, num sinal digital. 13

AJAX *Asynchronous JavaScript and XML (Extensible Markup Language)* é um conjunto de técnicas de desenvolvimento da Web que usa várias tecnologias no lado do cliente para criar aplicações Web assíncronas. 57

ALU *Arithmetic Logic Unit* (Unidade Lógica e Aritmética) é um circuito electrónico digital combinacional que executa operações aritméticas e lógicas em números binários inteiros. Uma ALU é um componente fundamental de muitos tipos de circuitos de computação, incluindo a CPU dos computadores. 11

API *Application Programming Interface* é uma interface ou protocolo de comunicação entre diferentes partes de um programa de computador, destinado a simplificar a implementação e a manutenção de software. 13

ARM *Advanced RISC Machine* é uma família de arquitecturas RISC (*Reduced instruction set computer*) desenvolvida pela empresa britânica ARM Holdings. 28

ASCII *American Standard Code for Information Interchange* é um código binário que codifica um conjunto de 128 sinais: 95 sinais gráficos (letras do alfabeto latino, sinais de pontuação e sinais matemáticos) e 33 sinais de controlo, usando 7 bits para representar todos os seus símbolos. 67

AVR é uma família de microcontroladores que usam uma arquitectura RISC (*Reduced instruction set computer*) de 8 bits Harvard modificada. 6, 9–11, 14, 15, 17, 32–36, 40–42, 75

CPU *Central Processing Unit* (Unidade Central de Processamento) é um circuito electrónico de um computador que executa instruções que compõem um programa. A CPU executa operações aritméticas básicas, lógicas, de controle e de entrada/saída (E/S) especificadas pelas instruções. 5

CSS *Cascading Style Sheets* é uma linguagem de folha de estilos usada para descrever a apresentação de um documento escrito numa linguagem como HTML. 55

DNS *Domain Name System* é um sistema hierárquico e descentralizado usado para gestão de nomes de máquinas ligadas à Internet. 20

DOM *Document Object Model* é uma norma multiplataforma e independente da linguagem de programação que trata um documento XML (*Extensible Markup Language*) ou HTML como uma estrutura em árvore, em que cada nó é um objecto que representa uma parte do documento. 55

DWARF é um formato de dados padrão usado guardar informação de debug. 55, 57

EEPROM *Electrically Erasable Programmable Read-Only Memory* é um tipo de memória não volátil normalmente usada em microcontroladores. 11

ELF *Executable and Linkable Format* é um tipo de ficheiro padrão para armazenar dados executáveis. 43

FLASH é um tipo de memória electrónica não volátil que pode ser apagada e reprogramada electronicamente. 2, 11, 14, 18, 21, 32, 40, 42, 43, 45, 47, 48, 50, 57, 60, 61, 67, 68, 73, 74

GDB *GNU Debugger* é um *debugger* portável que pode ser usado para depuração em sistemas *Unix-like* e que suporta várias linguagens de programação. 15

GPIO *General-purpose input/output* são portas programáveis de entrada e saída de dados usadas como interface com periféricos em microcontroladores e microprocessadores. 11

HIGH representa um valor lógico alto ou 1 (um) em circuitos electrónicos digitais. A tensão ou intervalo de tensões a que corresponde depende da tecnologia com que é implementada a electrónica. 52

HTML *HyperText Markup Language* é a linguagem de formatação padrão para documentos projectados para serem exibidos num navegador da Internet. 41

HTTP *Hypertext Transfer Protocol* é um protocolo de comunicação utilizado em sistemas distribuídos. É a base para a comunicação de dados na Internet. 17

I/O *Input/Output* (Entrada/Saída) é um termo utilizado para indicar a entrada e saída de dados em que entrada representa sinais ou dados recebidos pelo sistema e saída representa sinais ou dados enviados. 12

I²C *Inter-Integrated Circuit* é um barramento série síncrono, multi-master, multi-slave, usado para ligar em curtas distâncias periféricos de baixa velocidade a microcontroladores. 12

ICE *In Circuit Emulation* é a utilização de um dispositivo de hardware usado para depurar o software de um sistema embebido. Funciona usando um processador com a capacidade adicional de dar suporte a operações de depuração, assim como a de executar a função principal do sistema. 27

IDE *Integrated Development Environment* (Ambiente Integrado de Desenvolvimento) é uma aplicação que fornece vários recursos abrangentes aos programadores para o desenvolvimento de software. 1

Intel HEX é um formato de ficheiro que transporta dados binários encapsulados num formato de texto simples. É normalmente usado para programar micro-controladores e outros tipos de lógica programável. 43, 73

IoT *Internet of Things* (Internet das Coisas) é um termo utilizado para definir um sistema de computadores e outras máquinas digitais ligados em rede e que não necessitam de interacção humana. 1

IP *Internet Protocol* é um protocolo de comunicação usado entre todos os dispositivos ligados em rede para encaminhamento de dados na Internet. 20

ISA *Instruction Set Architecture* é um modelo abstracto de um computador que define quais são as operações que um processador, microprocessador, micro-controlador, CPU ou outros periféricos programáveis suporta, e portanto, que fornece ou disponibiliza ao programador. 18

J2EE *Java 2 Platform Enterprise Edition* é um conjunto de especificações que estende a Java Platform Standard Edition (Java SE) com especificações para recursos empresariais, como computação distribuída e serviços direccionados à web. 17

JAR *Java Archive* é um formato de ficheiro normalmente usado para agregar classes Java, metadados e recursos associados (texto, imagens, etc.) num ficheiro para distribuição. 21

JDBC *Java Database Connectivity* é uma API para a linguagem de programação Java que define como um cliente pode aceder a uma base de dados. 56

JIT *Just-in-Time Compilation* ou compilação/tradução dinâmica, é a compilação de um programa em tempo de execução, usando uma abordagem diferente da compilação anterior à execução. Geralmente, consiste em transformar o código em código de máquina, que é então executado directamente, mas também se pode referir a tradução para outros formatos. 24

JSON *JavaScript Object Notation* é um formato de ficheiro para transferência de dados em texto consistindo em pares atributo/valor e listas. Apesar de ser derivado do Javascript não depende de nenhuma linguagem de programação. 59

JSP *Java Server Pages* é uma colecção de tecnologias que ajuda os programadores a criar páginas da Web geradas dinamicamente com base em HTML, XML (*Extensible Markup Language*) ou outros tipos de documentos. 55

JVM *Java Virtual Machine* (Máquina Virtual Java) é uma máquina virtual que permite que um computador execute programas Java, bem como programas criados em outras linguagens de programação que também são compiladas para *bytecode* Java. 37

LCD *Liquid-Crystal Display* é um monitor de ecrã plano ou outro dispositivo óptico modulado electronicamente que usa as propriedades de modulação de luz dos cristais líquidos combinados com polarizadores. 13

LED *Light-emitting diode* (Díodo emissor de luz) é um semicondutor que emite luz quando é atravessado por uma corrente eléctrica. 1

LOW representa um valor lógico baixo ou 0 (zero) em circuitos electrónicos digitais. A tensão ou intervalo de tensões a que corresponde depende da tecnologia com que é implementada a electrónica. 52

MIPS *Microprocessor without Interlocked Pipeline Stages* é uma arquitectura de microprocessadores RISC (*Reduced instruction set computer*) desenvolvida pela MIPS Computer Systems. 28

NASA *National Aeronautics and Space Administration* (Administração Nacional da Aeronáutica e Espaço) é uma agência do Governo dos Estados Unidos da América responsável pela pesquisa e desenvolvimento de tecnologias e programas de exploração espacial. 83

NASA-TLX *NASA Task Load Index* é uma ferramenta de avaliação multidimensional, capaz de lidar com subjectividade e amplamente usada que classifica a carga de trabalho percepcionada para avaliar a eficácia ou outros aspectos do desempenho de uma tarefa, sistema ou equipa. 83

NAT *Network Address Translation* é uma técnica que consiste em reescrever os endereços IP de origem de um pacote que passam por um router ou firewall de maneira que um computador de uma rede interna tenha acesso ao exterior ou à Internet. 54

no-op *no operation* (nenhuma operação) é uma instrução ou operação que não tem nenhum efeito significativo. 42

PDF *Portable Document Format* é um formato de ficheiro, desenvolvido pela Adobe Systems em 1993, para representar documentos de maneira independente das aplicações, do hardware e do sistema operativo usado para o criar. 34

PowerPC *Performance Optimization With Enhanced RISC – Performance Computing* é uma arquitectura de computadores RISC (*Reduced instruction set computer*) desenvolvida por uma aliança entre a Apple, a IBM (International Business Machines Corporation) e a Motorola. 28

PWM *Pulse-width modulation* é um método de reduzir a potência fornecida por um sinal eléctrico, mudando rapidamente de estado entre ligado e desligado gerando um aparente sinal analógico a partir de um sinal puramente digital. 12

SHA *Secure Hash Algorithm* é uma família de funções de dispersão criptográfica considerada praticamente impossível de inverter. 62

SO Sistema Operativo é um conjunto de software que gere o hardware e recursos de software de um computador além de fornecer serviços comuns aos programas que nele executam. 15

SPARC *Scalable Processor Architecture* é uma arquitectura de computadores RISC (*Reduced instruction set computer*) *open-source* desenvolvida pela Sun Microsystems e pela Fujitsu. 28

SPI *Serial Peripheral Interface* é uma especificação de porta série síncrona usada para comunicações de curta distância, normalmente em sistemas embebidos. 12

SRAM *Static Random Access Memory* é um tipo de memória de acesso aleatório que usa *flip-flops* para armazenar dados. O termo *static* diferencia a SRAM da DRAM (*Dinamic Random Access Memory*) que tem de ser periodicamente refrescada. 12

SVG *Scalable Vector Graphics* é uma linguagem XML (*Extensible Markup Language*) para descrever de forma vectorial desenhos e gráficos bidimensionais. 62

TCP/IP *Transmission Control Protocol/Internet Protocol* é um conjunto de protocolos de comunicação entre computadores em rede usado na Internet. 17

URL *Uniform Resource Locator* é uma referência a um endereço de rede no qual se encontra algum recurso informático e que define o protocolo a usar para acesso e a localização desse mesmo recurso. 57

USART *Universal synchronous and asynchronous receiver-transmitter* é um tipo de porta série que pode ser programada para comunicar de modo assíncrono ou síncrono. 12

USB *Universal Serial Bus* é um padrão de indústria que estabelece especificações para cabos, conectores, e protocolos de comunicação para ligação, comunicação e fornecimento de energia entre computadores pessoais e dispositivos periféricos. 9

VLIW *Very-Long Instruction Word* refere-se a arquitecturas de conjuntos de instruções projectadas para explorar a execução de instruções em paralelo. Enquanto as unidades centrais de processamento convencionais (CPU) permitem principalmente que os programas especifiquem instruções para executar apenas em sequência, um processador VLIW permite que os programas especifiquem explicitamente instruções para executar em paralelo. Esse design destina-se a

permitir um maior desempenho sem a complexidade inerente a outras abordagens. 28

VMM *Virtual Machine Monitor* ou *hypervisor* é um software, firmware ou hardware que cria e executa máquinas virtuais. Um computador em que um VMM executa uma ou mais máquinas virtuais é chamado de *host*, e cada máquina virtual é chamada de *guest*. 24

WAR *Web Application Archive* é um tipo de ficheiro usado para distribuir uma colecção de ficheiros JAR, Java Server Pages, Java Servlets, classes Java, ficheiros XML (*Extensible Markup Language*), bibliotecas de *tags*, páginas HTML e outros recursos que juntos constituem uma aplicação web. 21

x86 é uma arquitectura de computadores CISC (*Complex instruction set computer*) desenvolvida pela Intel. 25, 28

Capítulo 1

Introdução

O ensino de programação com microcontroladores está a ficar cada vez mais comum no ensino secundário e superior, mesmo em cursos que não são focados especificamente na área da electrónica. Vários factores promovem este cenário: existe uma iniciativa nacional para promover uma literacia digital precoce, que agora é vista como um aspecto importante de cidadania que deve ser promovido o mais cedo possível (Prensky, 2008; Vee, 2013), solicitando por cursos relacionados à programação desde o ensino secundário ou até mais cedo (por exemplo usando *scratch* (Direção-Geral da Educação, 2020)). A crescente expansão da *Internet of things* (IoT), que usa todos os tipos de microcontroladores, o custo reduzido dos mesmos como plataforma de programação em comparação aos computadores tradicionais e o uso de microcontroladores como ferramenta de programação que permitem resultados imediatamente visíveis e palpáveis (por exemplo, *Light-emitting diode* (LED)s a piscar, controlo de motores etc.) promovem o interesse e o envolvimento dos alunos.

Uma das plataformas mais frequentemente usadas e melhor adaptadas ao cenário de ensino com microcontroladores é a Plataforma Arduino (Arduino SA, 2018), pois combina a simplicidade, ainda que engenhosa, do hardware com a facilidade de um ambiente de desenvolvimento integrado projectado especificamente para pessoas sem conhecimento significativo sobre microcontroladores (Barragán, Banzi Associate Professor & Crampton Smith Director, 2004) e tem a vantagem adicional de ser uma plataforma *open source*, o que significa que não há custos de licenciamento. Não é de surpreender que muitas escolas de ensino secundário e superior agora incluam assuntos de programação usando o Arduino (Agatolio & Moro, 2017; Sarik & Kymissis, 2010; Jamieson, 2011).

Um simulador de Arduino na sala de aula pode introduzir vantagens no entanto o seu uso ainda não é comum. Não encontrámos bibliografia que referencie que sejam utilizados simuladores de Arduino completos ou pelo menos utilizáveis no contexto de ensino em Portugal. Propomos contribuir para esse cenário, desenvolvendo e disponibilizando gratuitamente um simulador que possa ser usado em sala de aula, permitindo o desenvolvimento e teste de programas e pequenos circuitos de maneira semelhante à plataforma real de hardware e compatível com o uso do mesmo *Integrated Development Environment* (IDE), mantendo os mesmos procedimentos aos quais a pessoa que desenvolve o projecto está habituada.

1.1 Motivação

Os Arduinos são um componente essencial dos recursos computacionais presentes na sala de aula, pois desempenham várias funções ao mesmo tempo, de entre as quais se destacam: fornece o poder computacional necessário para executar pequenos projectos experimentais, permite aulas e exercícios de programação pura e permite um primeiro contacto com dispositivos electrónicos.

Apesar das suas muitas características positivas, o uso da Plataforma Arduino na sala de aula tem vários aspectos que podem diminuir a eficiência do esforço educacional (por exemplo, eficiência de custo e tempo na sala de aula) e podem até impedir o uso de algumas formas de ensino (por exemplo, ensino à distância). Estes aspectos vêem-se como uma oportunidade de melhoria, introduzindo o uso de um *Arduino virtual*, ou seja, um simulador de Arduino que mantém todas as vantagens do hardware real, diminui ou remove os aspectos que minimizam o desempenho da aula e abre novas oportunidades e formas de ensino não presentes no verdadeiro Arduino.

Dadas as suas vantagens, a tendência para usar este dispositivo na sala de aula provavelmente continuará no futuro próximo e possivelmente até aumentará. Embora o cenário de uso actual do Arduino na sala de aula seja um processo bem conhecido e bem gerido, há aspectos que podem ser aprimorados e oportunidades que podem ser exploradas para melhorar ainda mais o processo de ensino.

1.1.1 Limitações do uso do hardware real

Embora tenha muitas vantagens, existem algumas limitações ao usar o Arduino na sala de aula: desgaste do dispositivo, custo, bases de electrónica, destreza manual, problemas na montagem, eficiência na gestão de tempo da aula.

- **Desgaste do dispositivo** e **custo**: Tentativa e erro é o método mais comum usado pelos alunos, o que leva a um grande número de reprogramação dos dispositivos. O número médio de gravações da memória FLASH de um Arduino antes da falha é de cerca de 10.000 vezes (Atmel, 2018). Esse número é facilmente alcançado em apenas dois anos de uso em sala de aula. Se considerarmos o cenário comum de uma escola com 6 turmas interagindo com 1 sala equipada com Arduinos, com 5 exercícios por aula, 10 tentativas por exercício (baixa estimativa) e 15 aulas por ano, teremos cerca de 4500 gravações por ano. Isso fará com que a escola tenha de substituir os seus Arduinos a cada dois anos.

- **Bases de electrónica**: As típicas montagens de Arduino requerem o uso de pequenos componentes electrónicos (LEDs, resistências, etc.). Isso introduz a necessidade de noções básicas de electrónica, que não são normais para jovens estudantes, principalmente se não forem da área da electrónica, e podem actuar como uma barreira para os objectivos do ensino da programação (Currie & James-Reynolds, 2017).

- **Destreza manual**: Os projectos Arduino envolvem pequenos componentes que requerem um controlo muscular fino das mãos. A grande maioria dos estudantes não terá problemas com isso. No entanto, os alunos com uma leve

incapacidade de controlo muscular terão muita dificuldade em interagir com os pequenos componentes das montagens de projectos Arduino. As escolas devem ser inclusivas e, mesmo que o número de alunos afectados seja pequeno, esse é um problema a ser considerado.

- **Problemas na montagem**: Às vezes as ligações nas *breadboards* normalmente usadas não estão em perfeitas condições e o circuito não funciona por razões fora do controlo imediato do aluno, levando a perda de tempo e frustração, que agem contra os objectivos da aula.

- **Eficiência de tempo**: Os projectos Arduino requerem algum tempo para seleccionar todos os componentes, organizá-los e configurá-los e, no final da aula, recolhê-los e guardá-los. Quando comparado com a duração típica de uma aula, isto pode levar uma quantidade considerável de tempo que não é usada como tempo real de aula.

1.1.2 Mitigações oferecidas pela simulação

Propomos mitigar estes problemas através do uso de um simulador. Os problemas serão tratados e mitigados da seguinte maneira:

- **Desgaste do dispositivo e custo**: Os componentes usados no simulador são virtuais. Não há desgaste nem necessidade de substituição periódica. Isso reduzirá imediatamente o custo operacional da manutenção de uma sala de aula equipada com Arduinos. Em vez do hardware real tudo o que é necessário é um computador normal. Normalmente, esse tipo de equipamento já existe na escola, portanto, nenhum custo será adicionado.

- **Bases de electrónica**: A necessidade de grandes bases de electrónica pode ser reduzida ou mesmo totalmente removida se o simulador remover os pequenos detalhes inevitáveis, mas não centrais para os objectivos de ensino envolvidos no hardware real. Por exemplo, os LEDs não necessitam de uma resistência extra que pode ser simplesmente removida da simulação, permitindo que os alunos se concentrem nos aspectos centrais da programação. Por outro lado, se se desejar incluir esses pequenos detalhes, basta o simulador passar a considerá-los.

- **Destreza manual** e **problemas na montagem**: Usar um simulador com interface gráfica e mover um dispositivo apontador (por exemplo, o rato) é mais fácil do que colocar componentes muito pequenos numa *breadboard*. Isso elimina alguns impedimentos para alunos com leves deficiências musculares e torna a escola mais inclusiva. Também remove os casos em que as ligações defeituosos impedem o funcionamento dos circuitos e, assim, melhoram ainda mais a eficiência do tempo e reduzem a frustração.

- **Eficiência de tempo**: Montar um projecto simulado pode ser tão simples e rápido quanto ligar um computador, executar um programa e abrir um projecto da mesma maneira que se abre um ficheiro de texto. A selecção e colocação de componentes numa *breadboard* é substituída pela selecção de componentes numa lista de itens ou num menu e arrastar os mesmos para o local

desejado. Isso é mais rápido do que gerir componentes físicos, disponibilizando mais tempo para aprendizagem.

1.1.3 Novas oportunidades

O uso de uma Plataforma Arduino simulada abre novas oportunidades que podem melhorar muito os resultados do ensino. Listamos as que são mais relevantes no imediato:

- **Ensino à distância**: Os alunos normalmente não levam o hardware para casa. Se um aluno deseja continuar e melhorar o projecto, é necessário aguardar pela próxima aula para fazer isso. Um simulador pode resolver essa limitação, permitindo que o aluno use o software do simulador em casa, instalando no seu próprio computador ou ligando-se remotamente a um servidor na escola.
- **Remoção da restrição de tempo**: Se um projecto for grande o suficiente, talvez não seja possível concluí-lo numa só aula. A montagem do projecto é simplesmente desmontada para disponibilizar os componentes para a próxima turma. Um projecto simulado é apenas um conjunto de informação que pode ser armazenada num ficheiro e reaberta posteriormente, continuando o mesmo projecto. Isto abre uma oportunidade totalmente nova para projectos maiores.
- **Base de dados de exercícios e análise automatizada**: Um simulador pode oferecer facilmente um conjunto de exercícios predefinidos, até mesmo meio iniciados ou pré-configurados, permitindo que o aluno avance mais rapidamente para novos aspectos e não precise de refazer as etapas básicas já aprendidas. Mais importante, o simulador pode incluir um "oráculo" automatizado que pode analisar o circuito virtual e verificar a sua correcta operação, libertando o professor para ajudar outros alunos.
- **Gestão da sala de aula**: O simulador pode oferecer uma interface de gestão ao professor, permitindo que ele inspeccione remotamente o progresso de cada aluno sem ter que se aproximar fisicamente deles. As oportunidades aqui são muitas, incluindo, por exemplo, ajuda remota a estudantes (mesmo estudantes em casa que, por algum motivo, não pudessem estar presentes na escola).
- ***Debugging***. O *debug* não está disponível directamente usando o hardware real do Arduino devido ao modo como a placa de circuito impresso está implementada. No entanto, o *debug* é importante e deve ser incentivado. O simulador não compartilha essas limitações e pode oferecer os meios para depurar o código, incluindo funcionalidades avançadas, como execução passo a passo e inspecção de memória.

A exploração de todas as oportunidades exigirá funcionalidades avançadas. No entanto, um simulador pode ser preparado com a funcionalidade básica e novas funcionalidades podem ser adicionadas a qualquer momento. O simulador apresentado não implementa todas as oportunidades aqui mencionadas, mas permite o ensino à distância, a possibilidade de continuação do projecto na aula seguinte ou em casa e *debugging*. Pode ser usado para substituir o hardware real na sala de aula e permite a resolução dos exercícios mais comuns encontrados nas aulas dos cursos do ensino secundário. As restantes oportunidades ficam como sugestão de trabalho futuro.

1.2 Objectivos e Requisitos

Tendo em vista o objectivo de criar um simulador que permita a execução de simulações de uma plataforma Arduino de uma forma o mais semelhante possível a um ambiente real não virtualizado, torna-se necessário que o simulador seja capaz de lidar com o IDE normalmente usado com o Arduino. Mais concretamente, pretendemos que se possa enviar o produto da compilação do IDE directamente para o simulador.

Uma vez que pretendemos equipar laboratórios, que por norma têm vários computadores com este tipo de tecnologia, será desejável que o simulador siga uma lógica cliente-servidor, em que o simulador aja como um servidor e os clientes sejam apenas uma interface remota. Esta solução trás vantagens a nível de instalação, acessibilidade e manutenção. Neste cenário, a web surge como uma solução ideal, devendo o simulador agir como uma aplicação web acessível através de um *browser* comum. Deste modo também é possível usar postos de trabalho com menor poder computacional nos laboratórios uma vez que a simulação pode exigir bastantes recursos em termos de *Central Processing Unit* (CPU).

Também pretendemos um sistema que seja escalável. Como prevemos que as simulações sejam exigentes em termos de CPU, um sistema distribuído será vantajoso.

Tendo em vista os objectivos gerais do simulador e a forma que se prevê para o seu uso, definimos os seguintes requisitos para o simulador:

1. Deve ter a capacidade de ser implementado totalmente em software uma vez que não necessitamos de interface com hardware;

2. Deve ser portável de modo a poder ser usado em várias plataformas e sistemas operativos;

3. Deve ser modular de modo a ser simples adicionar suporte para outras instruções, processadores ou periféricos;

4. Deve ser capaz de executar um ficheiro binário original sem qualquer modificação;

5. A implementação deve ser eficaz em termos de desempenho de modo a que se consiga executar o simulador à mesma velocidade do hardware original ou o mais próximo possível;

6. Deve implementar mecanismos de segurança que não permitam que o binário execute código fora da sua *sandbox*;

7. Deve ser possível controlar a execução do processador de modo a parar ou retomar a execução do programa, executar *step-by-step* (uma instrução de cada vez) e inserir *breakpoints*;

8. Deve ser possível carregar simulações em *runtime*, ou seja, não ter de executar o simulador novamente para fazer uma nova simulação;

9. Deve ser possível programar o simulador atrás do IDE do Arduino;

10. Deve ter uma arquitectura cliente/servidor e usar uma interface web acessível através de um *browser*.

11. Deve ser possível de implementar dentro da calendarização disponível para este projecto.

1.3 Programa de trabalhos

Neste trabalho vamos apresentar uma *review* das placas Arduino e dos microcontroladores da arquitectura AVR, especificamente do Arduino Uno e do microcontrolador ATmega328P.

Vamos fazer um levantamento do estado da arte em termos de tecnologias de virtualização e emulação em geral para recolher conhecimento sobre o leque de técnicas existentes.

Para escolher a técnica de virtualização mais adequada ao simulador que pretendemos construir vamos criar uma prova de conceito de modo a testar cada uma delas.

Vamos apresentar uma proposta de arquitectura para um simulador com a modelação em software de todos os componentes do microcontrolador ATmega328P, com o código preparado para que seja fácil a implementação de outros microcontroladores da mesma família, juntamente com os componentes necessários para a simulação da placa Arduino Uno, mais uma vez com o código preparado para a fácil implementação de outras placas da família Arduino.

O simulador proposto vai ser implementado com uma interface gráfica e com a capacidade de criar circuitos virtuais com vários componentes como LEDs e botões, e a capacidade de simular o funcionamento dos mesmos.

Para manter a compatibilidade com o IDE do Arduino vamos criar um mecanismo para permitir a programação do simulador de uma forma similar à original, directamente do IDE.

Vamos realizar um estudo de caso de uso para verificar se o funcionamento do simulador é viável, se não altera a dinâmica da aula e se apresenta vantagens.

Para dar a conhecer o simulador e estimular a sua utilização vamos fazer acções de divulgação e apresentações junto de potenciais utilizadores.

1.4 Contribuições

Como contribuições deste trabalho ficam: um simulador de Arduino disponível em código fonte aberto, a publicação de três artigos científicos e várias acções de divulgação.

Artigos científicos em conferências internacionais:

- Tecnologias de Virtualização Para Simulação de Arduino[1], DOI:10.23919/CISTI.2019.8760727
- An ArduinoSimulator for Practical Embedded Programming Teaching
- An Arduino Simulator in Classroom - a Case Study[1], DOI:10.4230/OASIcs.ICPEC.2020.12

Acções de divulgação:

- Apresentação no ArduinoDay2019@IPT
- Apresentação no TIC@Portugal2019
- Apresentação de *workshop* sobre Arduino usando o simulador no *Critical Summer Camp — Internships*
- Várias apresentações no âmbito do Fikalab ISEC Challenge 2019

1.5 Estrutura do documento

A restante estrutura deste documento organiza-se da seguinte forma:

No Capítulo 2 vamos apresentar os conceitos e estado da arte. Vamos fazer uma introdução à Plataforma Arduino e apresentar outros produtos que podem ser usados para simular Arduinos.

No Capítulo 3 vamos apresentar a nossa proposta de arquitectura para o simulador que vamos construir.

No Capítulo 4 vamos apresentar um estudo sobre tecnologias de virtualização e seleccionar as que podem ser úteis para o nosso trabalho. Vamos, criando uma prova de conceito, escolher a técnica de virtualização mais adequada para usar no nosso simulador tendo em conta os objectivos a que nos propusemos.

No Capítulo 5 descrevemos a estrutura e as opções de implementação de cada um dos módulos que compõem o nosso trabalho. Apresentamos também a interface do utilizador e todas as suas funcionalidades.

No Capítulo 6 apresentamos as verificações que foram feitas ao sistema, incluindo testes unitários e funcionais, no sentido de perceber se se construiu um sistema segundo as especificações planeadas.

No Capítulo 7 vamos apresentar um caso de uso do simulador implementado num cenário real, recolhendo métricas sobre a sua utilização e apresentamos uma discussão sobre se é viável, se altera ou não a dinâmica da aula e se traz vantagens ao funcionamento das aulas.

No Capítulo 8 apresentamos as conclusões do trabalho e sugestões de trabalho futuro.

[1] com *peer review* e indexação *Scopus*

Capítulo 2

Conceitos e estado da arte

2.1 Plataforma Arduino

Arduino é uma plataforma electrónica, *open-source*, composta por uma vertente de hardware e uma vertente de software que trabalham em conjunto (Arduino SA, 2018).

Figura 2.1: Plataforma Arduino (Velleman for Makers, 2019)

2.1.1 Hardware

A vertente de hardware é uma placa de circuito impresso muito simples e compacta com um microcontrolador AVR (o modelo exacto depende da versão do Arduino), uma fonte de alimentação, uma interface série (normalmente *Universal Serial Bus* (USB)) para programação, pinos de entrada e saída para ligação a outros dispositivos electrónicos e um *bootloader* que permite que o dispositivo seja programado sem a necessidade de um programador dedicado. Tem também uma disposição física dos pinos de entrada e saída de modo a permitir o empilhamento de placas de expansão (*shields*) criadas especificamente para o Arduino. Existem os mais variados tipos de placas de expansão para Arduino como controladores de motores, placas de relés, interfaces Ethernet e WiFi, etc.

Podemos ver várias placas e *shields* Arduino na Figura 2.2 e a disposição dos componentes na placa de circuito impresso de um Arduino Uno na Figura 2.3.

ATmega328P

A placa Arduino escolhida para ser usada na prova de conceito foi a Arduino Uno que está equipada com o microcontrolador ATmega328P da Atmel. É um microcontrolador que implementa a arquitectura Harvard, normalmente usada em sistemas

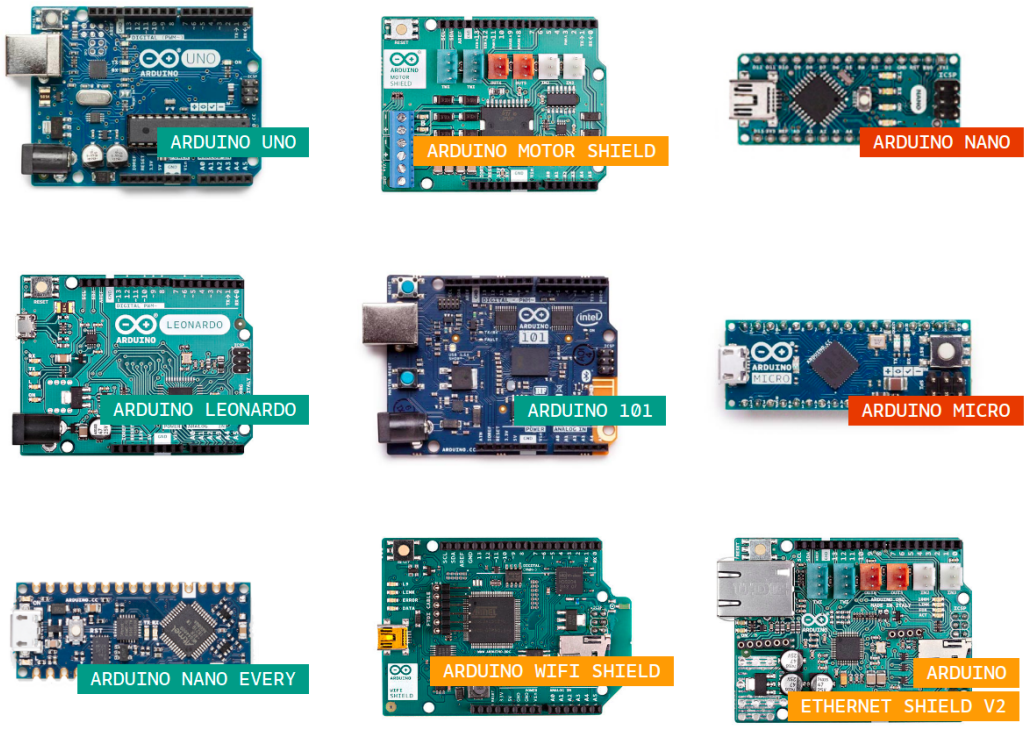


Figura 2.2: Placas e *Shields* Arduino (Arduino SA, 2018)

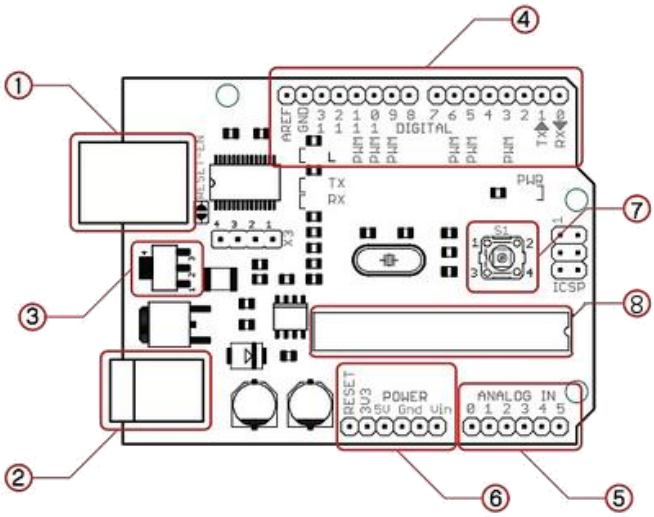


Figura 2.3: Diagrama do Arduino Uno (Adaptado de kumar C S, K.V, A, B e Appaji, 2017)

embebidos (Francillon & Castelluccia, 2008), que tem a característica de ter o endereçamento de memória e o endereçamento de instruções separados, e o *instruction set* AVR de 8 bits. A arquitectura AVR tem 131 instruções, sendo que a maioria é

executada em apenas um ciclo de relógio, e 32 registos de propósito geral todos ligados directamente à *Arithmetic Logic Unit* (ALU) permitindo o acesso aos registos num só ciclo de relógio (Figura 2.4).

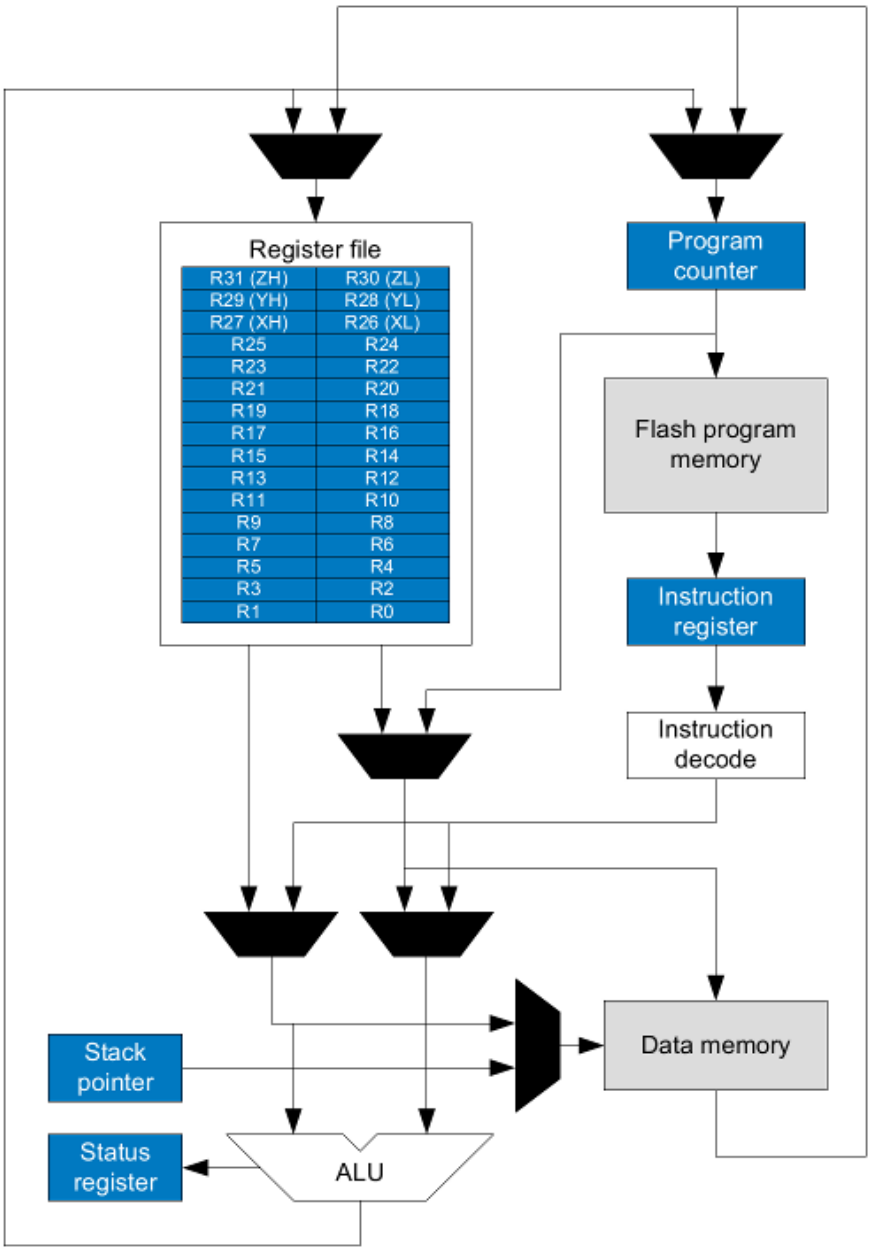


Figura 2.4: Diagrama de blocos da Arquitectura AVR (Atmel, 2018)

Este dispositivo em particular tem 32Kbytes de FLASH (memória de programa) o que permite armazenar até 16384 instruções uma vez que os dispositivos AVR têm instruções de 16 ou 32 bits. Em termos de memória RAM tem 2Kbytes e 1Kbyte de *Electrically Erasable Programmable Read-Only Memory* (EEPROM) (memória não volátil normalmente usada para guardar configurações). A velocidade de relógio pode ir até 20MHz, podendo atingir os 20MIPS. Em termos de periféricos, tem:

- 27 pinos de *General-purpose input/output* (GPIO);
- 2 temporizadores/contadores de 8 bits;

- 3 temporizadores/contadores de 16 bits;
- contador de tempo real com oscilador separado;
- 10 canais de *Pulse-width modulation* (PWM);
- 8 canais de conversores analógico/digital de 10 bits;
- 2 *Universal synchronous and asynchronous receiver-transmitter* (USART)s programáveis;
- 2 interfaces *Serial Peripheral Interface* (SPI) Master/Slave;
- 2 interfaces série Two-Wire (*Inter-Integrated Circuit* (I^2C));
- *Watchdog Timer* programável com oscilador separado;
- Comparador analógico;
- Interrupção e *Wake-Up* na mudança de pino.

Tem ainda as seguintes características:

- Oscilador interno calibrado de 8MHz;
- Fontes de interrupção externas e internas;
- 6 modos de *Sleep*;
- Sistema de detecção de falha de relógio;
- Número de série único.

O suporte para *Bootloader* permite actualizar o *firmware* a partir da própria aplicação.

Como se pode ver na Figura 2.5, a memória de dados do processador está organizada em 4 zonas, sendo a primeira a dos registos de uso genérico, a segunda a dos registos de *Input/Output* (I/O), a terceira a de registos estendidos de I/O e a quarta a *Static Random Access Memory* (SRAM).

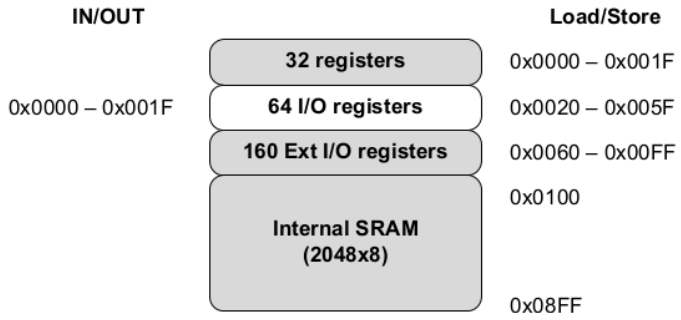


Figura 2.5: Mapa de memória do ATmega328P (Atmel, 2018)

Todos os periféricos são controlados escrevendo valores nos registos de I/O do microcontrolador não existindo instruções especiais para controlo dos mesmos.

2.1.2 Software

A vertente de software do Arduino incluí uma *Application Programming Interface* (API) que contém uma série de métodos para manipular todo o hardware disponível como pinos de entrada e saída, *timers*, *Analog-to-Digital Converter* (ADC)s, etc. Existem também várias bibliotecas de suporte a hardware específico como ecrãs *Liquid-Crystal Display* (LCD), interfaces ethernet, entre outros. Tem também um IDE (ver Figura 2.6) que permite escrever o código com um editor específico para a API do Arduino, compilar e programar o dispositivo, tudo na mesma interface. Ao instalar o IDE, são instaladas também todas as ferramentas necessárias para o pré-processamento, compilação, assemblagem, *linking* e *upload* do executável para o microcontrolador.

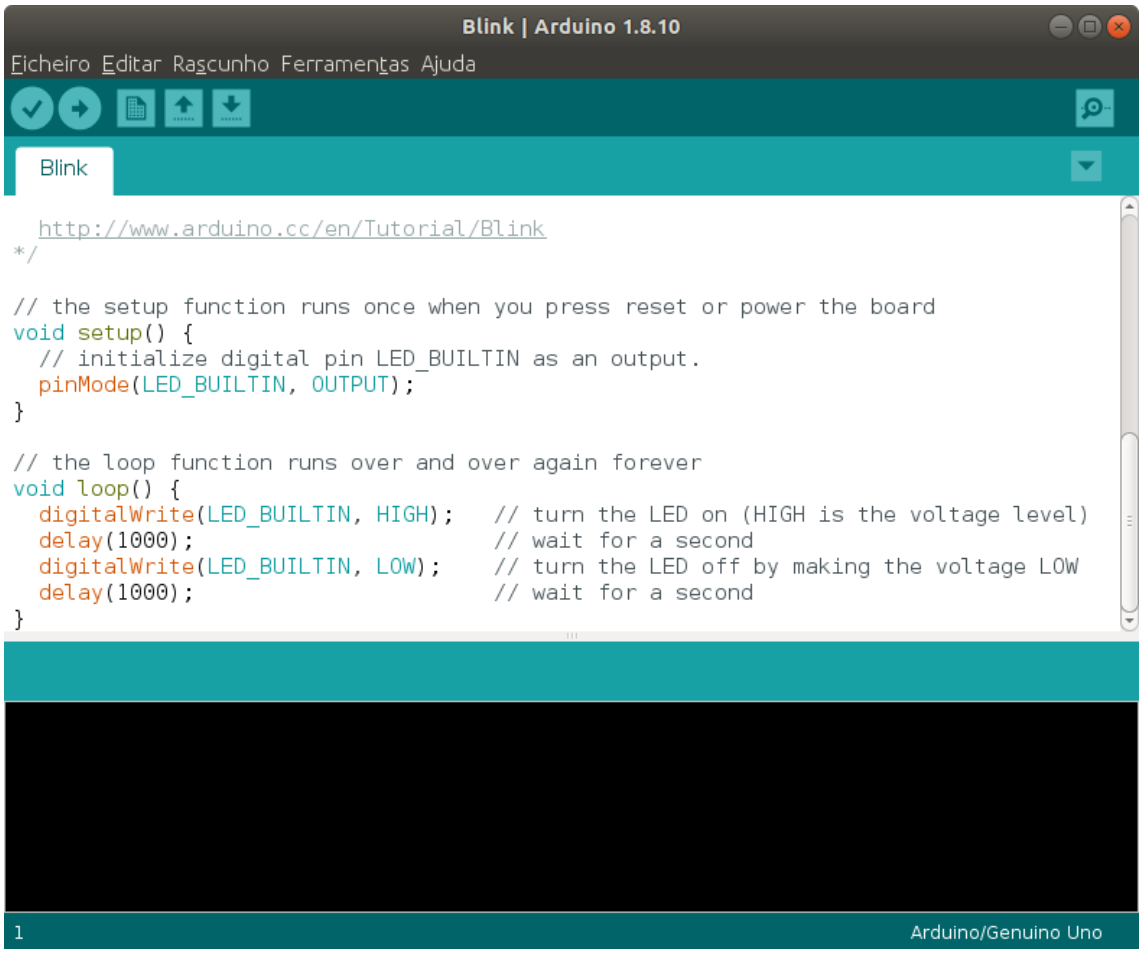


Figura 2.6: IDE Arduino

Para fazer a programação de um dispositivo Arduino basta ligar a placa ao computador através de um cabo USB, carregar num botão na interface do IDE e aguardar a mensagem de gravação concluída. É um processo bastante simples em comparação com a programação tradicional de microcontroladores em que é necessário usar um programador (um dispositivo físico que se liga entre o computador e o microcontrolador) e é necessária a utilização de um software específico para carregar o binário, muitas vezes com múltiplas opções de configuração nem sempre fáceis de compreender por parte do utilizador. Esta facilidade de programação é uma das razões para a proliferação dos Arduinos.

Nos bastidores é usada uma ferramenta de linha de comandos que comunica com o microcontrolador através de uma porta série/USB. A comunicação do lado do microcontrolador é garantida pelo *bootloader*, um software que está programado numa zona reservada da FLASH e que é o primeiro a ser executado no arranque do dispositivo. Se houver comunicação num determinado formato presente na porta série nos primeiros instantes do arranque, o dispositivo entra em modo de programação, caso contrário passa o controlo a outra zona da FLASH onde está o programa normal do Arduino. No caso de entrar em modo de programação recebe pela porta série o novo programa e escreve-o na zona dedicada ao programa normal na FLASH fazendo de seguida um *reset*.

2.2 Soluções de simulação existentes

Existem vários simuladores disponíveis que podem ser usados para executar código escrito para Arduino. Existem também ferramentas específicas para depurar código destas plataformas.

- **Proteus**: é um programa principalmente usado para desenho de placas de circuito impresso mas tem funcionalidades de simulação incluindo a execução de código em microcontroladores AVR.
- **Vitronics Simulator for Arduino**: é um simulador especificamente para Arduino desenvolvido em Pascal para sistemas operativos Windows. Não permite criar circuitos mas apenas observar os valores dos pinos.
- **VBB4Arduino**: é também um simulador específico para Arduino que apresenta uma breadboard virtual onde se podem criar circuitos com vários componentes.
- **123D Circuits**: é um produto que foi descontinuado e as suas funcionalidades foram integradas no **Tinkercad**, ambos produtos da empresa Autodesk. No módulo Circuits do Tinkercad é possível criar pequenos circuitos com Arduinos e executar simulações.
- **ArduinoDebugger**: é um simulador de Arduino com capacidade de desenho de circuitos muito limitada e em que é necessário compilar o código Arduino juntamente com o código do simulador para executar uma simulação.
- **CodeBlocks Arduino IDE**: é um IDE para Arduino que tem em desenvolvimento um simulador a nível de API para Arduino.
- **Simuino**: é um simulador ao nível dos pinos do Arduino, não permite criar circuitos. Tem uma interface em modo de texto e outra web.
- **Emulino**: é uma aplicação com uma única janela com um circuito fixo (um LCD ligado ao Arduino) e que necessita de ser compilada para mudar o código a simular.
- **Atmel Studio 7**: é uma ferramenta de *debug* para dispositivos reais. Permite fazer *debug* em microcontroladores AVR mas não simular circuitos.

- **Emulare**: permite simular Arduinos com o microcontrolador ATmega328P e criar circuitos virtuais mas para fazer debug é necessário usar a ferramenta *GNU Debugger* (GDB).

- **SimAVR**: é uma ferramenta de linha de comandos que permite simular vários microcontroladores AVR e escrever as suas saídas para um ficheiro para futura análise. Também permite fazer *debug* através do GDB.

Analisámos algumas características, relacionadas com os objectivos definidos para o simulador de Arduino e com o tipo de utilização que é possível fazer dos simuladores.

A primeira característica é se o produto é gratuito ou não. Para utilização em aulas e querendo reduzir o custo é uma característica importante.

Se é um produto *open-source* ou não também é importante. Um produto *open-source* dá a possibilidade de poder ser adaptado às nossas necessidades o que pode ser interessante em determinados cenários como por exemplo autenticação integrada com um directório existente.

Outra característica é se é *cross-platform*. Apesar de haver o domínio dos Sistema Operativo (SO)s da Microsoft não queremos limitar os utilizadores nesse aspecto.

A utilização de uma interface web facilita a manutenção do software nos postos de trabalho pelo que também é considerada.

A compatibilidade a nível binário também é importante devido à capacidade de executar código máquina aumentando o leque de funcionalidades que se podem simular.

Um software mantido é importante. Queremos um produto que não esteja estagnado no tempo, que evolua e principalmente que tenha correcção de *bugs*.

A funcionalidade de *debug* é uma mais valia dando a possibilidade ao utilizador de mais facilmente encontrar a razão de problemas no mau funcionamento do seu código, entre outras.

Consideramos a compatibilidade com o IDE do Arduino uma característica fundamental para quem está a aprender não exigindo uma mudança de contexto tão radical na passagem do simulador para o hardware real.

O resultado da nossa análise é apresentado na Tabela 2.1. A característica que consideramos mais importante é a compatibilidade a nível binário. Isto é importante e necessário para manter o mecanismo de carregamento igual ao usado no hardware real. Podemos verificar que mais de metade dos simuladores existentes apenas são compatíveis com a API. Isto quer dizer que só simulam um número fixo de funcionalidades básicas do Arduino e que não executam código binário para a arquitectura AVR. Consideramos isto uma limitação pois muito provavelmente não permitirá executar código das muitas bibliotecas existentes para Arduino.

Relativamente à compatibilidade com o IDE do Arduino, nenhum permitem programar o simulador directamente a partir do IDE. Consideramos que isto é importante numa situação de ensino pois mais tarde os alunos vão programar dispositivos reais e não queremos que o ambiente seja muito diferente daquele em que aprenderam. O objectivo no ensino deve ser ensinar com as ferramentas que os futuros profissionais vão encontrar no seu local de trabalho.

Dos compatíveis a nível binário, nenhum tem Interface Web, outro dos requisitos que consideramos importante para a facilidade de manutenção do laboratório.

Tabela 2.1: Comparação de simuladores de Arduino

Software	Gratuito	Open-source	Cross-platform	Interface Web	Compatível a nível binário	Mantido	Permite Debug	Compatível com IDE Arduino
Proteus (https://www.labcenter.com/)	Não	Não	Sim	Não	Binário	Sim	Sim	Não
Virtronics Simulator for Arduino (https://virtronics.com.au/Simulator-for-Arduino.html)	Não	Não	Não	Não	API	Não	Sim	Não
VBB4Arduino (http://www.virtualbreadboard.com/)	Não	Não	Não	Não	API	Sim	Sim	Não
123D Circuits (https://123d.circuits.io/)	Sim	Não	Sim	Sim	API	Não	–	Não
Tinkercad (https://www.tinkercad.com/)	Sim	Não	Sim	Sim	API	Sim	Sim	Não
ArduinoDebugger (https://github.com/Paulware/ArduinoDebugger)	Sim	Sim	Não	Não	API	Não	Sim	Não
CodeBlocks Arduino IDE (http://arduinodev.com/codeblocks/)	Sim	Sim	Não	Não	API	Sim	–	Não
Simuino (http://web.simuino.com/home-1)	Sim	Sim	Sim	Sim	API	Não	Sim	Não
Emulino (https://github.com/ghewgill/emulino)	Sim	Sim	Sim	Não	Binário	Não	Não	Não
Atmel Studio 7 (https://www.microchip.com/mplab/avr-support/atmel-studio-7)	Sim	Não	Sim	Não	Binário	Sim	Sim[1]	Não
Emulare (http://emulare.sourceforge.net/)	Sim	Sim	Sim	Não	Binário	Não	Sim[2]	Não
SimAVR (https://github.com/buserror/simavr)	Sim	Sim	Sim	Não	Binário	Sim	Sim[2]	Não

Apesar de muitos dos que têm compatibilidade binária permitirem *debug*, todos excepto um necessitam do auxilio de uma ferramenta externa o que também aumenta a complexidade do *setup*, e o que não necessita de ferramenta externa não está disponível gratuitamente. No caso do Atmel Studio 7, é necessário hardware adicional, por exemplo o Atmel-ICE (Atmel, 2020) cujo *kit* mais barato custa cerca de 100€[3]. Além disso é necessário fazer alterações no hardware do Arduino para permitir a utilização do protocolo debugWire (ScienceProg, 2007), que é um protocolo fechado[4], do microcontrolador ATmega328P (no caso do Arduino Uno usado neste trabalho). A utilização deste *debugger* também implica a remoção do *bootloader* do dispositivo o que faz com que não possa voltar a ser programado no IDE do Arduino antes da sua reposição. No caso do Emulare e do SimAVR é necessário usar a ferramenta GDB (The GNU Project, 2020) para fazer *debug* ao software.

[1] com hardware extra

[2] com debugger externo

[3] Valor calculado através de uma pesquisa no ebay.

[4] Embora a especificação do protocolo não seja pública existe informação sobre o mesmo na Internet (http://www.ruemohr.org/docs/debugwire.html).

Capítulo 3

Proposta e Arquitectura

Para que o sistema seja portável de modo a poder ser usado em várias plataformas e sistemas operativos optámos por usar a linguagem de programação Java. Esta linguagem, além de garantir a portabilidade, também tem a vantagem das especificações *Java 2 Platform Enterprise Edition* (J2EE) que definem características como computação distribuída, web services e servidores aplicacionais. Estas especificações vão de encontro aos objectivos de implementar uma aplicação web e uma aplicação escalável. A programação orientada a objectos, o polimorfismo e a herança do Java também é a ideal para permitir de um modo bastante simples a extensão do software para outras instruções processadores ou periféricos.

Os critérios de eficácia de desempenho, mecanismos de segurança (*sandbox*), controlo da simulação e execução da simulação em *runtime* são os que mais vão influenciar a escolha da técnica usada para implementar a virtualização do microcontrolador (ver Capítulo 4).

Decidimos usar um ambiente Web para baixar os custos de instalação, manutenção e acessibilidade. Este tipo de ambiente também permite usar postos de trabalho com menos poder computacional uma vez que estes computadores usados pelos alunos serão apenas responsáveis pela interface com o utilizador e simulação do circuito enquanto a simulação do microcontrolador e execução das instruções é da responsabilidade do servidor que tem mais recursos disponíveis.

Em termos de acessibilidade, a possibilidade de aceder ao simulador via *Hypertext Transfer Protocol* (HTTP), permite que os alunos possam trabalhar no sistema em qualquer lugar onde tenham Internet sem a exigência de configurações fora do normal como poderia ter de acontecer se se tratasse de uma aplicação cliente/servidor tradicional com a exigência de utilização de portas *Transmission Control Protocol/Internet Protocol* (TCP/IP) próprias.

O simulador está organizado como um típico sistema cliente/servidor web. O servidor opera os mecanismos e a lógica da simulação e pode servir múltiplas simulações independentes ao mesmo tempo, dependendo esse número apenas da sua capacidade de processamento. O cliente trata da interacção com o utilizador e com o IDE do Arduino. A interface do utilizador é baseada em tecnologia web comum e é servida ao utilizador através de um *browser-web* comum como o Firefox ou o Google Chrome.

A Figura 3.1 mostra os módulos que compõem o simulador, que são:

- **Microcontroller simulator**: Neste módulo são implementadas todas as características do microcontrolador, nomeadamente o *Instruction Set* AVR, os

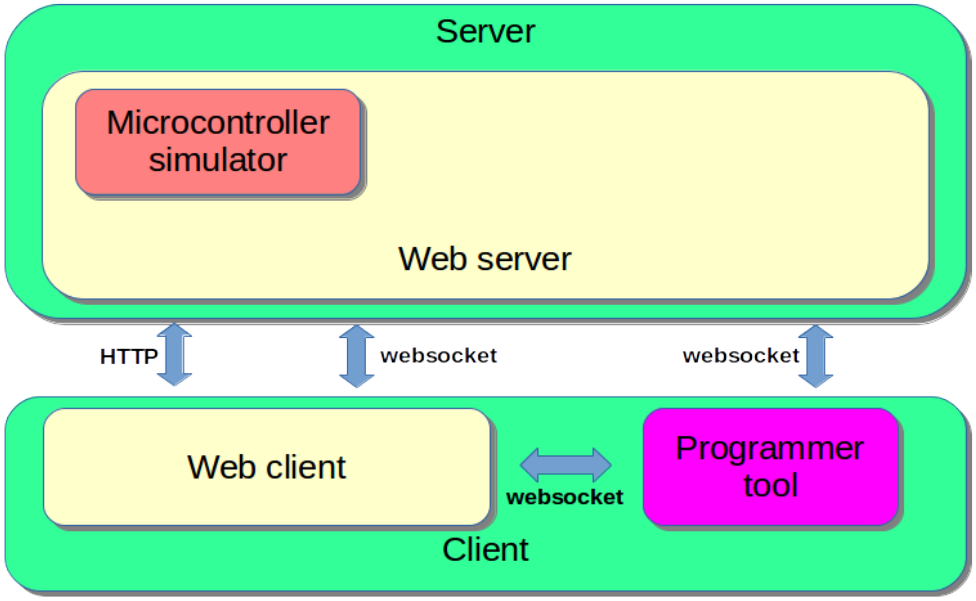


Figura 3.1: Arquitectura

periféricos do microcontrolador, a FLASH e a SRAM. É este módulo que executa o código do microcontrolador, expõe métodos para alterar o valor dos pinos e lança eventos quanto os estado dos mesmos é alterado internamente. Preparámos este modulo de modo a ser fácil de implementar novos processadores podendo o código do *Instruction Set Architecture* (ISA), FLASH, SRAM e periféricos ser utilizado em processadores diferentes do ATmega328P que foi implementado.

Este módulo corresponde a um projecto separado e as suas funcionalidades podem ser usadas noutras aplicações.

- **Web server**: Este módulo faz toda a gestão de utilizadores na aplicação, a manutenção das instâncias de simulação e a ligação entre a simulação e o cliente e o programador correcto. Também é da responsabilidade deste módulo guardar os projectos criados pelos utilizadores e todos os dados que dizem respeito a esses projectos.

- **Web client**: O cliente web é onde o utilizador pode criar um projecto para simular. O utilizador tem disponível uma área de desenho onde pode adicionar um Arduino e vários componentes electrónicos e fazer ligações entre os mesmos. Todas as funcionalidades do simulador podem ser acedidas usando o cliente: o utilizador pode dar inicio à simulação, pausá-la, fazer *step-by-step*, analisar as memórias FLASH e SRAM, adicionar ou remover *breakpoints*, etc.

  Também é da responsabilidade do cliente tentar ligar-se periodicamente à ferramenta de programação (*plugin* instalado no IDE do Arduino) e transferir para este os dados necessários para ele se poder ligar directamente ao servidor.

- **Programmer tool**: Este módulo é um software que é instalado no IDE Arduino e que substitui o programa que efectua a programação do dispositivo (avrdude). Neste caso em vez de programar um dispositivo real é enviado o

binário para o servidor web para que este possa executar a simulação. Está incluído no gestor de placa que é instalado no IDE do Arduino.

3.1 Placa Arduino para ferramenta de programação

As alternativas que se apresentam como possíveis para carregar o executável do Arduino no simulador são: fazer um simples upload de ficheiros na interface web ou criar uma nova placa (*board*) (Arduino SA, 2019), um *plugin* para o IDE Arduino, que permite expandir o IDE para novos tipos de hardware. Utilizar o upload de ficheiros implica que o utilizador tem de procurar o ficheiro do executável no seu computador, que pode não ser fácil de identificar para uma pessoa menos experiente na utilização de compiladores para Arduinos. Também modifica o método usado na utilização de um Arduino real que é fazer o carregamento directamente através da interface do IDE Arduino. A criação da placa define um novo tipo de Arduino no Gestor de Placas, o "**Arduino Uno Simulator**", e o utilizador apenas tem de seleccionar num menu do IDE qual o dispositivo que quer usar, se o real, se o simulador.

Esta nova placa tem de ser instalada no IDE Arduino mas é extremamente fácil de fazer e só é necessário fazê-lo uma vez em cada posto de trabalho.

A placa estende a placa Arduino Uno existente, permitindo manter o mesmo compilador, definindo apenas uma nova ferramenta de programação.

3.2 Simulação do circuito electrónico

A simulação da placa Arduino é feita no servidor mas a simulação do circuito electrónico ligado à placa é feita no cliente. Optámos por esta abordagem porque a simulação da electrónica é mais básica e pode ser facilmente feita em JavaScript no cliente (*browser*). Isto também permite que o desenvolvimento dos componentes electrónicos e da simulação electrónica em geral seja construída num projecto separado o que facilita o desenvolvimento. Deste modo a comunicação entre o cliente e o servidor, em termos de sinais electrónicos, fica limitada aos pinos da placa Arduino o que também torna a ligação das duas componentes do sistema mais fácil.

Em alternativa teria de ser passada muito mais informação entre o servidor e o cliente. Por exemplo, cada componente electrónico teria de ter um estado e todas as alterações desse estado implicariam tráfego entre os dois módulos. Com o aumento da complexidade dos circuitos vai aumentando esta necessidade. Com simulação da electrónica no cliente e a limitação de comunicações de estado aos pinos do microcontrolador o aumento da complexidade do circuito não aumenta o tráfego entre cliente e servidor.

3.3 Escalabilidade

Não faz parte dos objectivos uma implementação em larga escala em que exista apenas uma instalação para servir centenas ou milhares de simulações, de qualquer modo fica feita uma análise de como escalar o sistema com um único ponto de acesso, ou seja, um único portal onde se possam ligar todos os clientes.

Analisámos as seguintes hipóteses para permitir escalabilidade em grande escala:

- *Domain Name System* (DNS) *round robin* (Wikipedia contributors, 2020a): é uma técnica de balanceamento de carga implementada através da resolução de nomes, em que a cada pedido é fornecido um endereço *Internet Protocol* (IP) diferente de entre uma lista de endereços disponíveis, direccionando deste modo os clientes para os vários servidores diferentes. Com esta solução temos de acautelar a sincronização das sessões, mas os servidores aplicacionais normalmente têm mecanismos para isso, embora no nosso caso a sincronização não seja suficiente porque temos a simulação a executar num servidor específico. Teríamos de garantir também que a ferramenta de programação se liga à máquina onde está a correr a simulação o que pode ser bastante difícil de conseguir uma vez que é o servidor de DNS que faz o balanceamento dos IP's e não teríamos controlo sobre essa lógica. Também existe um outro problema bastante sério que é a utilização de recursos por parte do servidor. Quem escolhe o servidor a que o cliente vai ficar ligado é o servidor de DNS e este não tem como saber qual o servidor que tem mais disponibilidade para correr a simulação.

- Proxy com *sticky session* (Wikipedia contributors, 2020b; Metawerx, 2008): é também uma técnica de balanceamento de carga mas em que toda a comunicação é feita com um único servidor proxy que depois distribui a carga por vários servidores aplicacionais. O proxy mantém no seu estado interno o servidor aplicacional que atribuiu a cada cliente, reencaminhando o cliente para o mesmo servidor aplicacional em pedidos subsequentes. Deste modo evita-se o problema do cliente web se ligar a servidores diferentes em cada *request* mas a ferramenta de programação não tem como saber qual é o servidor. O cliente web faz uso de um *cookie* de sessão para que o proxy encaminhe os pedidos subsequentes para o mesmo servidor aplicacional mas por norma é vedado o acesso a esses *cookies* ao ambiente Javascript pelo que não seria possível encaminhar esse dado para a ferramenta de programação. Também aqui existe o problema da utilização de recursos por parte do servidor. Neste caso é o proxy que escolhe o servidor e tal como anteriormente não sabe a disponibilidade de cada um.

- Correr o simulador num serviço externo: com esta abordagem o processamento é feito num servidor externo (simulador) e pode haver uma orquestração entre o servidor web e vários desses serviços de modo a que o servidor web controla que servidores externos já estão ocupados ou não. Esta abordagem é semelhante ao funcionamento do Selenium Grid (Selenium, 2020) onde temos o *hub* e os *nodes* como apresentado na Figura 3.2.

  - O simulador quando arranca regista-se no servidor web e dá a sua disponibilidade de carga (quantas simulações consegue correr ao mesmo tempo baseado no número de *cores* ou fazendo um teste de carga correndo uma simulação de teste).
  - O servidor web faz de ponte de comunicação entre o cliente web/ferramenta de programação e o simulador.
  - O servidor web deve ter a possibilidade de mudar a simulação para outro simulador pois podemos pausar uma simulação e o simulador ficar disponível para outra simulação que entretanto começa e quando fizermos

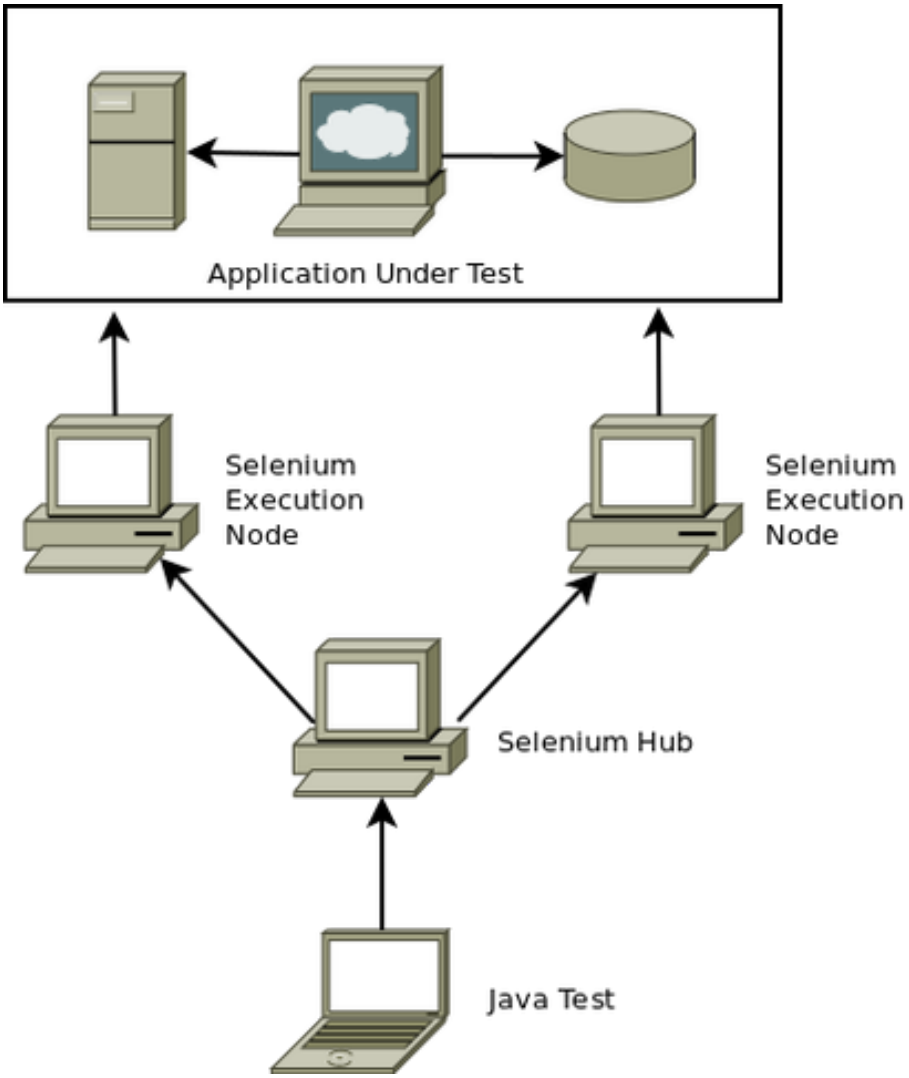


Figura 3.2: Arquitectura do *Selenium Grid* (harsha2selenium, 2015)

> resumir na primeira já não existe capacidade para a executar. Nesse caso o estado da simulação deve ser movido para outro simulador disponível. Será necessário fazer a cópia da memória FLASH, da memória SRAM e de todo o resto do estado do CPU.

A última hipótese parece bastante viável, resolvendo as limitações das duas primeiras, e fica como proposta de trabalho futuro para a evolução do projecto do simulador.

Também é possível fazer uma instalação local à máquina do utilizador e correr o servidor do simulador apenas para si. Esta solução permite ao utilizador não depender de uma instalação de terceiros como da sua escola ou de um servidor de livre acesso, mas o modo de instalação não é muito amigável para quem quer *apenas* usar o simulador pois implica instalar um *servlet container*, como por exemplo o Apache Tomcat (The Apache Software Foundation, 2020), e adicionar e configurar o ficheiro *Web Application Archive* (WAR) da aplicação no *servlet container*. Para tornar um *deploy* deste género mais fácil é possível criar um ficheiro *Java Archive* (JAR) executável com um *servlet container* que tenha a possibilidade de ser embebido, como o Eclipse Jetty (Eclipse Foundation, 2020). Isto permite executar o

servidor do simulador com um simples duplo clique num ficheiro em qualquer SO moderno que tenha o Java instalado. Esta funcionalidade não será implementada e também fica proposta como trabalho futuro.

Em caso de necessidade de aumentar a disponibilidade do sistema sem mudar a arquitectura implementada podemos aumentar o número de instalações de servidores aplicacionais e usar uma base de dados comum, uma situação usada com frequência na industria (Anicas, 2014). Esta solução implica criar um endereço web diferente para cada servidor aplicacional e fazer a distribuição dos utilizadores manualmente, ou seja, indicar a cada utilizador a que servidor se deve ligar, para evitar sobrecarregar um único servidor.

Capítulo 4

Tecnologias de virtualização

A origem do termo máquina virtual vem dos anos 60 e início dos anos 70, altura em que surgiram as primeiras máquinas virtuais de sistema (as que permitem executar todo o sistema operativo num ambiente virtualizado) (Smith & Nair, 2005). Nessa altura os computadores eram máquinas caras e de difícil acesso o que levava a que vários utilizadores partilhassem a mesma máquina. Com a tecnologia de máquinas virtuais permitia-se que os vários utilizadores pudessem utilizar os recursos disponíveis ao mesmo tempo. Virtualização pode então ser definida como uma tecnologia que coloca uma camada de software entre o hardware e o software e/ou o SO que vai executar no ambiente emulado (Sahoo, Mohapatra & Lath, 2010), permitindo deste modo abstrair o hardware. As máquinas virtuais permitem aumentar a interoperabilidade do software, a robustez dos sistemas, a versatilidade das plataformas e melhorar a sua utilização (Smith & Nair, 2005; Uhlig et al., 2005).

Com o desenrolar dos anos e com a diminuição do custo do hardware, o computador pessoal passou a estar omnipresente no dia a dia e a virtualização passou um pouco para segundo plano, voltando a estar em voga a partir de 2010, como uma maneira de melhorar a segurança, a confiabilidade e a disponibilidade do sistema, reduzir custos e proporcionar maior flexibilidade (Sahoo et al., 2010).

Hoje em dia, no caso de máquinas virtuais de sistema, a utilização mais corrente é o isolamento de vários sistemas a executar paralelamente no mesmo hardware (Smith & Nair, 2005). Se existir um problema de segurança num *guest* (a máquina que está a ser virtualizada) ou se o SO sofrer uma falha catastrófica, o software que correr noutros *guests* não será afectado. Diferentes utilizadores, com necessidades de SOs diferentes, também podem facilmente partilhar um servidor virtualizado, dado que a virtualização permite criar um ambiente independente e específico para cada um. As actualizações de SOs e software também podem ser testadas em máquinas virtuais, num sistema clonado a partir do original (de produção), mantendo este em funcionamento contínuo e testando as actualizações num ambiente *sandbox* sem riscos (Uhlig et al., 2005).

Embora estes benefícios sejam normalmente mais utilizados em servidores de alto desempenho, os novos sistemas de virtualização, ao alcance de qualquer um, têm introduzido a virtualização numa mais ampla gama de sistemas de servidor e cliente (Uhlig et al., 2005).

Outra utilização para a virtualização é a preservação de objectos digitais. Há muitos arquivos digitais que guardam documentos em formatos que vão deixando de ser suportados por novas versões de aplicações e até mesmo de novas versões de

SOs. Neste caso a virtualização é uma importante ferramenta para manter a compatibilidade necessária para a execução do software imprescindível para visualizar esses documentos digitais (Van Der Hoeven, Lohman & Verdegem, 2007).

4.1 Métodos de virtualização

Para resolver o problema da conversão de software que tinha sido feito para sistemas mais antigos, foi introduzido nas máquinas IBM System/360 uma técnica a que foi dado o nome de *emulation* (emulação) (Tucker, 1965). Compreendia uma conjugação de software e hardware o que fazia com que fosse cerca de 5 a 10 vezes mais rápido do que uma solução implementada puramente em software. Por exemplo, o *fetch* (recolha da próxima instrução a executar da memória) e o *decoding* (descodificação e identificação) da instrução é feito em hardware e depois é chamada uma rotina em software para executar a instrução.

Também foram criados interpretadores, que são ferramentas que correm no *host* (a máquina que está a executar a simulação) para imitar o comportamento de uma aplicação de uma outra arquitectura (Reshadi, Mishra & Dutt, 1986). Na interpretação o programa é guardado na memória do simulador tal como seria guardado no dispositivo original. O software que está a executar a simulação recolhe a próxima instrução a executar da memória (*fetch*), de seguida descodifica o *opcode* (o valor que representa a operação a executar) para identificar a instrução, ou seja, saber o que fazer, e de seguida executa a instrução para simular os efeitos que ela tem no estado interno do dispositivo simulado (Mills, Ahalt & Fowler, 1991). Apesar da facilidade de implementação, os interpretadores têm sérios problemas de desempenho devido ao custo de realizar o *fetch*, a descodificação e execução. Ainda assim quase todos os simuladores comerciais são interpretativos (Zhu & Gajski, 1995).

A simulação compilada é uma técnica similar à interpretação mas a descodificação das instruções é feita em tempo de compilação (do próprio simulador) e não durante o tempo de execução, o que aumenta o desempenho da simulação (Reshadi et al., 1986). A fase de *fetch* também é eliminada colocando o código na sequência correcta no programa e usando técnicas especiais para lidar com os saltos que serão detalhadas na Secção 4.6.

Existe também a técnica *Just-in-Time Compilation* (JIT) ou *dynamic compilation* que consiste na transformação em tempo de execução das instruções de uma determinada arquitectura em outra (a do hardware) trazendo deste modo as vantagem de simulação compilada e interpretação (Aycock, 2003).

Nas máquinas virtuais de sistema existe o conceito de *Virtual Machine Monitor* (VMM), que é um *layer* de software que tem como objectivo arbitrar o acesso aos recursos de hardware como o processador, a memória e o espaço de endereçamento de I/O (Uhlig et al., 2005). A virtualização com VMM permite-nos ter:

- **Isolamento do sistema**: podem ser feitas instalações de várias aplicações em SOs diferentes protegendo desse modo os sistemas contra vulnerabilidades que possam existir numa das aplicações. Também protege contra falhas numa aplicação que possa trazer todo o SO a uma situação de instabilidade.

- **Consolidação de sistemas**: podem-se juntar várias instalações de diferentes SOs num só hardware optimizando os recursos físicos.

- **Migração de máquinas virtuais**: podem existir sistemas independentes do hardware onde estão a correr o que permite fazer *upgrade* de hardware de um modo muito simples. Também permite migrar máquinas virtuais para outro hardware no caso do actual estar a ficar muito sobrecarregado.

4.2 Desafios de virtualização em IA-32 e Itanium

A virtualização levanta questões importantes de desempenho e de segurança que podem ser bastante optimizadas se existirem instruções específicas para virtualização no processador *host*. No caso concreto da arquitectura IA-32 (x86) e Itanium, é importante proteger as estruturas de dados do VMM (para que o *guest* não lhes aceda e altere estado vital para o mesmo) (Uhlig et al., 2005). Esta protecção é bastante difícil de conseguir se não existirem instruções específicas na arquitectura para virtualização.

Antes de existirem instruções específicas para virtualização, para ultrapassar os desafios impostos por essas arquitecturas, os projectistas de VMMs criaram soluções criativas para modificar o software *guest* de modo a que este execute correctamente sob o controlo de um VMM (Uhlig et al., 2005). Uma solução passa por os virtualizadores executarem o SO, e as aplicações, todos no mesmo nível de privilégio (Ring 3) ou pode-se usar *ring compression* (Uhlig et al., 2005), ou seja mudar o SO *guest* para outro Ring que não o 0. No entanto, os sistemas operativos esperam operar nesse modo e não funcionam adequadamente noutro, sendo necessário alterar o código fonte dos sistemas operativos ou fazer alterações nos binários para que a virtualização funcione correctamente. No caso de se fazer a mudança do código fonte do SO para permitir a virtualização chama-se paravirtualização (Barham et al., 2003) (ver Figura 4.1). No entanto, esta solução exige acesso ao código fonte do SO, o que nem sempre é possível.

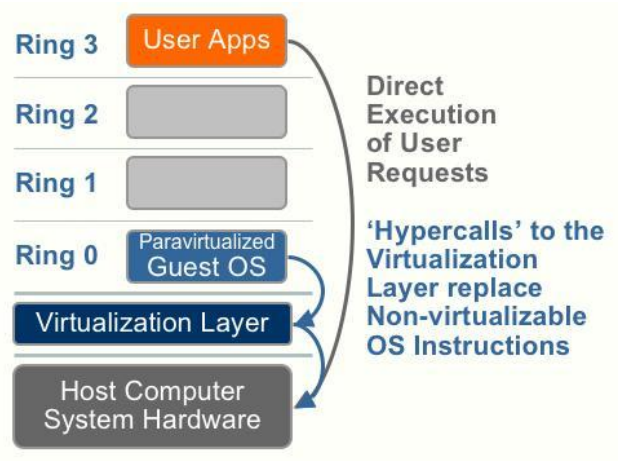


Figura 4.1: Paravirtualização (Iqbal, 2009)

No caso de se modificar os binários do sistema operativo e executar o mesmo no Ring 1 em vez do habitual Ring 0, e executar o VMM no Ring 0 (ver Figura 4.2), dá-se o nome de *binary translation*. Esta solução traz uma penalização ao nível do desempenho se comparado com a paravirtualização mas permite o suporte de um número mais alargado de SOs.

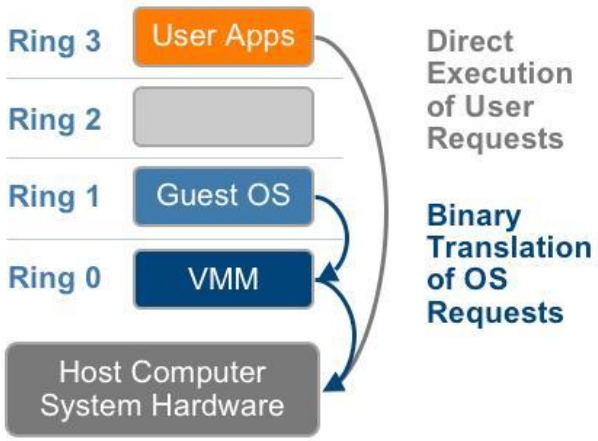


Figura 4.2: *Binary translation* (Iqbal, 2009)

A Intel introduziu uma tecnologia nos seus processadores, o VT-x nos IA-32 e VT-i nos Itanium, para ajudar os VMMs no seu trabalho. A AMD também introduziu logo de seguida uma tecnologia equivalente nos seus processadores, o AMD-V. O VT-x veio trazer um nível abaixo dos 4 *rings* normais chamado VMX root (onde corre o VMM) - que tem uma estrutura similar ao nível de cima (o VMX non-root) com 4 *rings* - e que permite ao VMM controlar as instruções que são executadas no VMX non-root. Isto permite a um SO correr onde espera sem qualquer modificação mas ao mesmo tempo ser controlado pelo VMM (ver Figura 4.3).

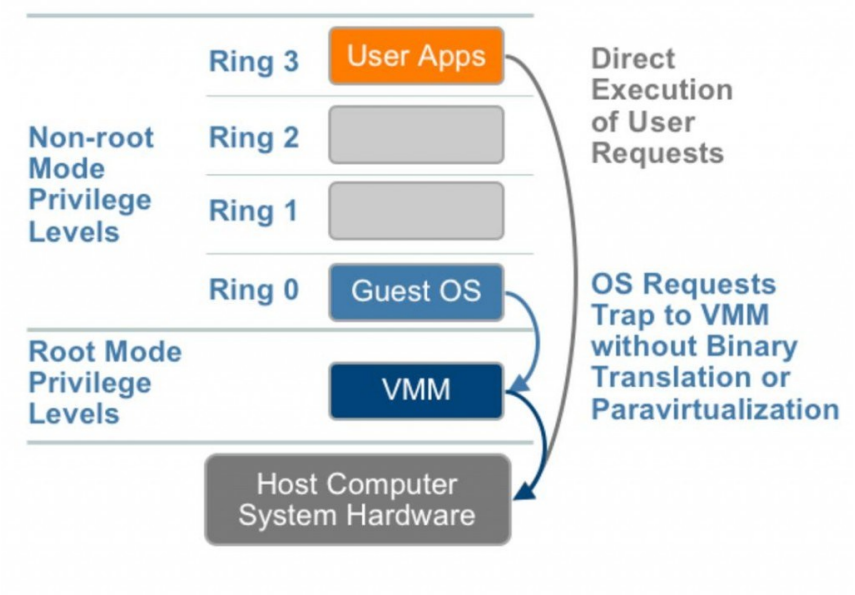


Figura 4.3: Virtualização assistida por hardware (Iqbal, 2009)

4.3 Emulação e Simulação

Simulação consiste em modelar o estado interno do que se está a simular, ou seja, todas as funcionalidades internas devem ser implementadas no simulador. Uma

simulação pode executar mais rápido (ou mais lento) do que o que se está a simular, por exemplo, uma simulação da expansão do universo.

Emulação consiste em imitar as saídas observáveis de modo a serem iguais ao que se está a emular. O estado interno do mecanismo de emulação não tem de ser igual ao que está a ser emulado. O objectivo de um bom emulador é substituir o objecto emulado. Um cenário comum de emulação é o da emulação de máquinas de jogos de vídeo que permite executar código binário original e realmente jogar o jogo (Kudlugi, Hassoun, Selvidge & Pryor, 2001). Normalmente um emulador para algo que se liga a outros componentes electrónicos (como um processador) tem uma interface em hardware para poder ser ligada onde normalmente estaria ligada aos restantes componentes (Tucker, 1965).

O produto que desenvolvemos neste trabalho tem características de ambas as definições. No nosso caso optámos por usar o termo simulação para definir o tipo de produto que construímos. O nosso simulador implementa algumas funcionalidades internas do sistema mas não todas, mas não cumpre uma das características normalmente encontradas na emulação que é a possibilidade da substituição do dispositivo original no circuito. Nunca será possível substituir um Arduino verdadeiro num circuito onde esteja aplicado pelo nosso trabalho.

4.3.1 *In Circuit Emulation*

In Circuit Emulation (ICE) é uma técnica que combina software e hardware para substituir um microprocessador, microcontrolador, memória, geradores de relógio e dispositivos de entrada e saída por um outro dispositivo controlado por software. Deste modo é possível manter controlo do sistema através de um processo de *debug*, recolher e alterar dados e executar programas (Dolinskii, Zisel'Man & Fedortsov, 1999). Normalmente fazem parte do ambiente de desenvolvimento de microprocessadores ou microcontroladores uma vez que dão suporte aos *designers* no desenvolvimento e manutenção do hardware e software dos sistemas alvo (Huang, Kao, Chen, Juan & Lu, 2002).

Os ICEs são compostos por um módulo de hardware que faz a interface com o resto do circuito, que por sua vez está ligado, normalmente por um cabo série ou USB, a um computador que corre um software que controla as suas entradas e saídas.

4.4 Classificação de tipos de Virtualização

Para melhor entendermos as diferenças entre os vários métodos, Smith e Nair, 2005 propõe a seguinte classificação em termos de virtualização:

- **Process Virtual Machines**: É o tipo de virtualização orientada a processo.
  - **Multiprogrammed systems**: É o que permite o isolamento de processos nos sistemas operativos de hoje em dia. Cada processo tem o seu espaço de endereçamento próprio e o processador é partilhado com outros processos inclusive de diferentes utilizadores.
  - **Emulators**: Um emulador permite a execução de outras ISAs através de interpretação ou *dynamic binary translation*. Um exemplo é a IA-32

Execution Layer (IA-32 EL) que permite a execução de aplicações x86 em processadores Itanium.

- **High-level language VMs**: São as máquinas virtuais que foram desenhadas especificamente para correrem código portável numa ISA que não corresponde a uma plataforma real. São exemplo a máquina virtual Java da Sun Microsystems e a Common Language Infrastructure (base do .NET) da Microsoft.

- **System Virtual Machines**: As máquinas virtuais de sistema são as que permitem que todo um sistema operativo corra num sistema emulado.

  - **Classic system VMs**: São as máquinas virtuais em que o VMM corre directamente no hardware sem a necessidade de um sistema operativo (bare-metal hypervisor). É exemplo o VMWare ESXi.
  - **Hosted VMs**: São as máquinas virtuais em que o VMM corre como uma aplicação do sistema operativo instalado no hardware. É exemplo o VirtualBox.
  - **Whole-system VMs**: São as máquinas virtuais que correm ISAs que não a do hardware que está a ser usado. Neste caso o hardware tem de ser emulado. É exemplo QEMU que corre máquinas virtuais de várias arquitecturas incluindo x86, *Microprocessor without Interlocked Pipeline Stages* (MIPS), *Advanced RISC Machine* (ARM), *Performance Optimization With Enhanced RISC – Performance Computing* (PowerPC) e *Scalable Processor Architecture* (SPARC) (Bellard, 2005).
  - **Codesigned VMs**: São máquinas virtuais criadas unicamente para emular uma ISA. O seu único propósito é aumentar o desempenho e/ou aumentar a eficiência energética. Um exemplo foi o processador Transmeta Crusoe (anos 2000) em que o hardware usava *Very-Long Instruction Word* (VLIW) para simplificar o processador e consequentemente o gasto de energia, conseguindo um desempenho similar a um processador nativo x86 da mesma época.

4.5 Tecnicas de Virtualização

Dos tipos de virtualização apresentados por Smith e Nair, 2005, aquele em que se enquadra o simulador para um dispositivo como o Arduino é a Emulação. Neste caso temos uma ISA diferente entre *guest* e *host* e apenas queremos executar um processo (o firmware que preparámos para o microcontrolador). Devemos notar que aplicamos o termo "Emulação"naquilo que consideramos ser um simulador mas, como referido na Secção 4.3, consideramos que temos um pouco das duas definições no nosso trabalho.

Das técnicas disponíveis para aplicar em emulação, temos as seguintes:

- Interpretação,
- *Dynamic translation* e
- Simulação compilada.

Vamos analisar cada uma delas em detalhe e verificar o cumprimento dos requisitos definidos para o nosso simulador.

4.5.1 Interpretação

A interpretação é a técnica mais comum e mais bem estabelecida para simular arquitecturas de computadores (Mills et al., 1991). Na interpretação, o simulador tem uma estrutura de dados que representa o estado interno do processador (Gajski, 2002), tal como o *program counter* e os registos de uso geral do processador, assim como a memória RAM e memória de programa. O programa é carregado na memória do simulador tal como seria no hardware verdadeiro. Depois o simulador entra num ciclo em que obtém o próximo *opcode* a executar (***fetch***) indicado pelo *program counter* a partir da memória de programa, faz a descodificação (***decode***) do *opcode* de modo a saber que operações devem ser realizadas, salta para a zona de código que implementa a instrução (***dispatch***), e de seguida executa (***execute***) essas mesmas operações que alteram o estado interno do processador (Mills et al., 1991; Gajski, 2002). O processo pode ser visto na Figura 4.4.

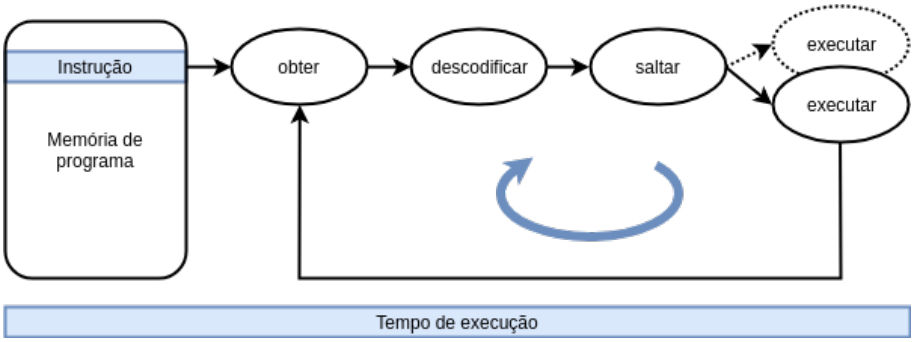


Figura 4.4: Interpretação (Adaptado de Reshadi, Mishra e Dutt, 1986)

```
for( ; ; ) {
   instruction = fetch( pc );
   opcode = decode( instruction );
   switch( opcode ) {
      ....
      case ADD:
         //execute instruction
         break;
      ....
      case JMP:
         pc = 0x125;
         break;
   }
}
```

Listagem 4.1: Ciclo de simulação interpretada

Na Listagem 4.1 podemos ver o código típico da implementação da interpretação. Temos um ciclo infinito, dentro do qual vamos buscar a instrução apontada pelo

program counter, fazemos a sua descodificação para identificar a instrução e de seguida usamos um `switch` para saltar para a parte do código que implementa a execução.

Esta é uma solução que é flexível mas relativamente lenta (Reshadi et al., 1986), pois é necessário ler o *opcode* da memória e fazer a interpretação dos *bits* que identificam a instrução a executar e os operandos da mesma.

Comparando com os critérios definidos, podemos concluir que teremos um défice em termos de desempenho.

4.5.2 *Dynamic translation*

Dynamic translation ou compilação dinâmica consiste em fazer, em tempo de execução, uma conversão entre as instruções do ISA que se está a simular para o ISA do *host* (a máquina que está a executar a simulação) (Bellard, 2005).

A conversão de instruções pode ser feita directamente de uma instrução do *guest* para uma ou mais instruções equivalentes do *host* (ver Figura 4.5) ou pode ser feita de um bloco do *guest* para outro bloco do *host* (Aycock, 2003). Quando se trata de converter blocos existe a necessidade de identificar blocos que pertençam ao mesmo controlo de fluxo do programa (Cmelik & Keppel, 1995), isto é, o bloco a converter tem de ser algo que seja sempre executado sequencialmente. Por exemplo, não se pode converter um bloco de código que tem a possibilidade de uma mudança de fluxo a meio uma vez que depois de transformado, esse mesmo código não será implementado do mesmo modo e o controlo de fluxo do programa pode ficar diferente.

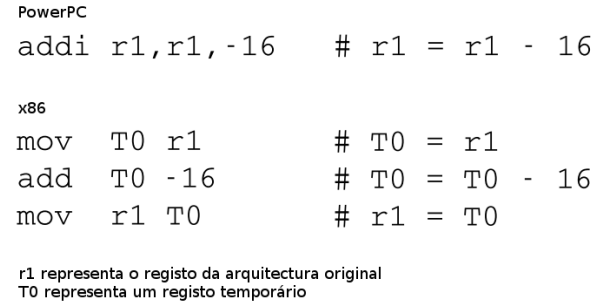


Figura 4.5: Conversão de uma instrução PowerPc em instruções x86 (Adaptado de Bellard, 2005)

Também é habitual o produto da conversão da compilação dinâmica ser guardado numa *cache* e ser reutilizada a transformação que foi feita anteriormente numa futura execução das mesmas instruções.

A compilação dinâmica é um meio para melhorar a eficácia de um programa em termos de tempo de execução e espaço (Aycock, 2003).

A compilação dinâmica é difícil de implementar e representa a mesma quantidade de trabalho do que implementar um compilador (Bellard, 2005).

Comparando com os critérios, podemos concluir que é eficaz em termos de desempenho, mas existe a possibilidade de executar código que não se pretende executar, pois estamos a gerar o código executado no *host* em tempo de execução, podendo levar a falhas de segurança. Também é extremamente difícil de implementar.

4.5.3 Simulação compilada

A simulação compilada faz a descodificação das instruções em tempo de compilação ao invés de em tempo de execução (Reshadi et al., 1986), portanto a descodificação só é feita uma vez para cada instrução (Braun, Hoffmann, Nohl & Meyr, 2001) e antes da execução.

É gerado código fonte com o código da execução das instruções de forma sequencial de acordo com o valor do *program counter*. O código que implementa uma determinada instrução vai aparecer repetido tantas vezes quantas a instrução for usada na aplicação original. Podemos ver na Figura 4.6 os passos que ocorrem em tempo de compilação e em tempo de execução.

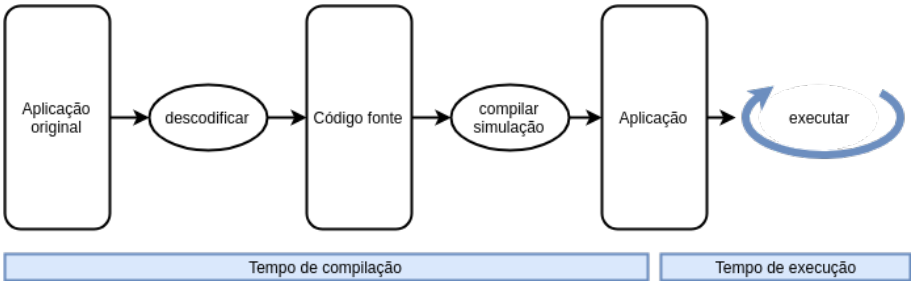


Figura 4.6: Simulação compilada (Adaptado de Reshadi, Mishra e Dutt, 1986)

Tal como podemos ver na Listagem 4.2, também a simulação compilada tem um ciclo infinito e um `switch`, tal como a interpretação, mas o `switch` selecciona directamente o código da simulação baseado no conteúdo do *program counter* actual (Leupers, Elste & Landwehr, 1999).

```
for( ; ; ) {
   switch( pc ) {
      case 0x00:
         //execute instruction
      case 0x01:
         //execute instruction
      ....
      case 0x57:
         //execute instruction
         if( is_to_jump ) {
            pc = 0x61;
            break;
         }
      case 0x58:
         //execute instruction
      ....
   }
}
```

Listagem 4.2: Ciclo de simulação compilada

Enquanto que as simulações compiladas obtêm velocidades relativamente mais altas do que a interpretação, tem a desvantagem que o programa da simulação tem

de ser refeito após cada alteração da aplicação original (Leupers et al., 1999). Além disso não permitem executar código que altera a memória de programa como é o caso quando o Arduino executa o *bootloader*, ou seja, o código deve ser estático (Reshadi et al., 1986).

Analisando os critérios definidos na Secção 1.2 e comparando os resultados obtidos, o desempenho em termos de velocidade de execução é melhor do que a interpretação. O controlo da execução é mais difícil de implementar pois teríamos de incluir código em cada `case` para testar essa situação.

4.6 Simulação do ISA AVR em Java

A solução de *Dynamic translation*, apesar de ser interessante na vertente de velocidade de execução, tem o problema da dificuldade de implementação dado que além de termos de conhecer a fundo a ISA do dispositivo que se pretende simular também é necessário conhecer a muito bem a ISA do *host* já que existe a necessidade de transformar blocos de instruções de uma arquitectura directamente na outra. Dada a complexidade da implementação desta técnica esta solução não é abordada, mas fica como uma possibilidade para quando se pretende uma abordagem em que o desempenho é o requisito mais importante.

De modo a decidir entre Interpretação e Simulação compilada, implementámos ambas as abordagens, em Java, de modo a poderem ser recolhidas estatísticas para comparação de desempenho.

Antes de entrar na implementação das técnicas em si, desenvolvemos um *decompiler* de modo a validar se o *decode* das instruções era feito de forma correcta. As instruções AVR são descodificadas e executadas de acordo com as especificações da Atmel para o *instruction set* AVR (Atmel, 2016). A forma como implementámos a descodificação das instruções é analisada com detalhe na Subsecção 4.6.1.

De seguida modulámos o estado interno do CPU, que incluí a memória SRAM, a memória FLASH, os registos internos, as *flags* do processador e alguns periféricos.

4.6.1 Descodificação de instruções

O *decode* pode ser uma tarefa bastante simples ou uma tarefa muito complicada dependendo do ISA que estivermos a interpretar. Existem ISAs em que os *bits* que identificam a instrução estão sempre na mesma posição, bastando usar uma mascara de *bits* e fazer um E lógico com o valor da instrução para obter um índice para a instrução. Noutros casos, como é o caso do AVR, esses *bits* não estão sempre na mesma posição e é necessário lógica bastante complexa para identificar a instrução. Por exemplo, no caso do ISA IA-32, e apesar de ser um *instruction set* multibyte, temos um byte que identifica imediatamente a instrução. Podemos ver na Figura 4.7 que existe um campo específico para o *opcode* da instrução. Esse valor pode ser usado num `switch` para saltar para o ponto de execução próprio para tratar essa instrução.

No caso do AVR temos a complexidade de os *bits* que identificam o *opcode* não estarem sempre na mesma localização, logo não podemos fazer um `switch` com o valor recolhido de um determinado local da instrução. Podemos verificar na Figura 4.8 que no mesmo local tanto podem estar *bits* fixos que identificam a instrução, como *bits* que representam parâmetros da mesma.

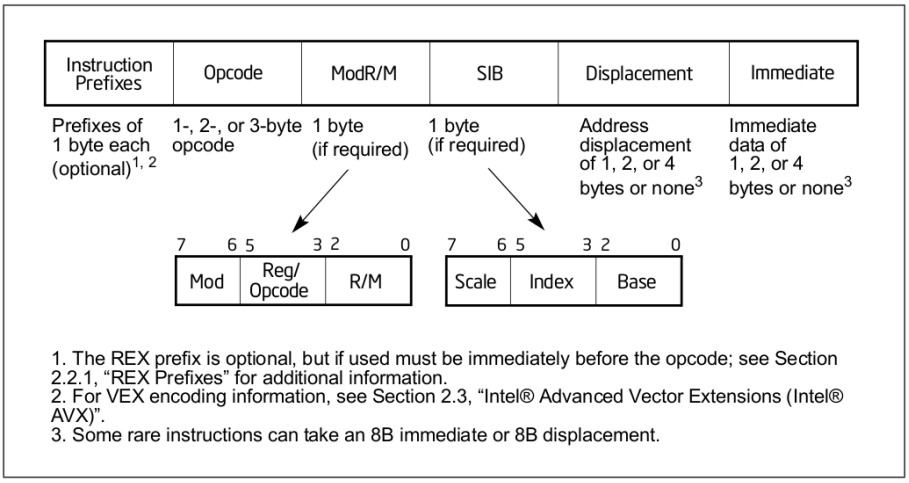


Figura 4.7: Formato das instruções Intel 64 e IA-32 (Intel Corporation, 2011)

ADC – Add with Carry

0001	11rd	dddd	rrrr

ST (STD) – Store Indirect From Register to Data Space using Index Y

10q0	qq1r	rrrr	1qqq

BRPL – Branch if Plus

1111	01kk	kkkk	k010

Figura 4.8: Formato das instruções AVR Adaptado de Atmel, 2018

Analisando o código fonte da ferramenta avr-objdump (Atmel, 2019), podemos verificar que são usados vários `switch`s encadeados (12) para fazer a descodificação da instrução para além de vários `if`s (também 12) e `else`s (7).

Uma vez que o desempenho é um requisito importante para o objectivo deste trabalho, optámos por procurar um método mais rápido para fazer a descodificação das instruções. Optámos por usar um *array* com as instruções indexado com todas as possibilidades do valor da instrução. Este método ocupa mais memória, pois o *array* contém 65535 elementos, mas o acesso é imediato, e portanto, mais eficaz em termos de velocidade. Para calcular os valores, fizemos um programa que, usando o formato da instrução (ver Figura 4.8), substitui os valores dinâmicos, representados por letras, por todas as possibilidades combinatórias de 1 e 0. Depois dos valores calculados é criado um *array* que mapeia todas as possibilidades de valores de instruções com a instrução correcta.

4.6.2 Implementação da Interpretação

Para implementar a interpretação fizemos uma classe para cada tipo de instrução. Essa classe tem o método `execute` que altera o estado interno do processador e a memória de acordo com as especificações do ISA AVR para essa instrução (Atmel, 2016). O processo de *decode* devolve um objecto do tipo da classe correcta para essa instrução. Optámos por devolver sempre o mesmo objecto e não criar um novo quando se faz o *decode*, para evitar estar sempre a criar objectos novos, que é uma operação lenta no Java. Como usamos sempre o mesmo objecto, este não pode conter estado relacionado com a instrução actual. Para resolver este problema passamos o

objecto CPU por parâmetro para o método `execute` assim como os parâmetros da instrução.

Para ajudar na criação das classes, tal como já havia acontecido para a descodificação das instruções, fizemos um programa auxiliar que cria os ficheiros das classes a partir do *Portable Document Format* (PDF) do manual do ISA do AVR (Atmel, 2016).

No executar da simulação, o programa entra num ciclo infinito, onde é feito o *fetch* da memória de programa da instrução a executar (usando o registo *program counter*), é feita a descodificação da instrução, e de seguida a execução.

4.6.3 Implementação da Simulação compilada

Para criar a simulação compilada, recorremos a mais um programa auxiliar. Esse programa faz o *fetch* e o *decode* do programa do Arduino e cria um ficheiro de código fonte Java com as instruções que depois será compilado e executado.

Primeiro faz o *fetch* das instruções de modo sequencial na memória de programa e não de acordo com o valor do *program counter*. Devemos ter isso em consideração porque neste caso queremos criar um ficheiro de código fonte, com o código da execução das instruções, na mesma sequência que estão no ficheiro binário do programa original para Arduino. Outra questão a ter em consideração é que nesta fase não estamos a executar a instrução, logo o *program counter* não é incrementado.

De seguida é feita a descodificação e colocado o código da execução da instrução no código fonte dentro de um bloco `case` tal como vimos na Subsecção 4.5.3. Para evitar repetir manualmente o código da execução da instrução, o programa lê os ficheiros de código fonte das classes criadas para a interpretação, extrai o código fonte e coloca-o no ficheiro da simulação compilada. Deste modo poupamos trabalho e ao mesmo tempo garantimos que o código da simulação interpretada e da simulação compilada é o mesmo.

Um problema que tivemos de ultrapassar foi o tamanho do método Java criado. Um método em *bytecode* não pode ter mais de 64k bytes por condicionantes do formato de ficheiro `class` do Java. Para ultrapassar esse problema criámos métodos mais pequenos, neste caso o resultante de 400 instruções AVR, que depois são chamados sequencialmente num outro método, como podemos ver na Listagem 4.3.

Este método resolve o problema mas introduz alguma penalização em termos de desempenho uma vez que todos os métodos vão ser chamados mesmo que não tenham nada para executar. Vejamos o seguinte exemplo: se o método `execute1` colocar o *program counter* a 300 e fizer um `break` (exemplo de um `JUMP`), em vez de voltar a entrar no `switch` que contém esse endereço (`execute1`), vai primeiro entrar nos métodos `execute2`, `execute3`, etc. Esses métodos, como não têm no seu `switch` uma *label* para o valor 300, vão retornar imediatamente, mas estas chamadas poderiam ser evitadas se só existisse um `switch`.

Depois a classe é compilada e executada. Quando é feita a execução do programa, já não existe a fase de *fetch* e *decode* uma vez que esse trabalho foi feito uma só vez para cada instrução na fase de compilação, portanto, no caso de haver muitos ciclos e a execução repetida da mesma zona de memória de programa, ganhamos tempo ao não ter de fazer o *fetch* e o *decode* novamente.

```
public void execute(CPU cpu) {
   for( ; ; ) {
      execute1(cpu);
      execute2(cpu);
      execute3(cpu);
      ....
   }
}

private void execute1(CPU cpu) {
   switch( cpu.getPc() ) {
      case 0: {
         ....
      }
      case 1: {
         ....
      }
   }
}

private void execute2(CPU cpu) {
   switch( cpu.getPc() ) {
      case 400: {
         ....
      }
      case 401: {
         ....
      }
   }
}
```

Listagem 4.3: Desdobramento do ciclo de simulação compilada

4.6.4 Comparação de resultados

De modo a comparar as duas técnicas necessitamos de executar o mesmo programa em ambas e ver qual demora mais tempo a executar. A questão é que programa executar. Fizemos uma pesquisa para encontrar um *benchmark* típico para a arquitectura AVR mas não encontrámos nenhum. Decidimos então usar o cálculo da sequência de Fibonacci com vários limites entre 1000 e 10 milhões, 100 vezes. Devemos ter em atenção que nesta fase não tínhamos todas as instruções AVR implementadas mas com este teste podemos exercitar ciclos (com testes lógicos), somas e forçámos a introdução da multiplicação calculando o limite da sequência através de uma multiplicação. A código é apresentado na Listagem 4.4.

O microcontrolador do Arduino nunca pára de executar instruções. Normalmente o que acontece quando não existem mais tarefas a realizar é ficar num ciclo infinito em que não faz nada. Mas para fazer os testes necessitamos de saber quando a tarefa terminou. Poderíamos contar um número de ciclos de relógio (ou seja o número de instruções executadas) e depois fazer os cálculos do tempo da execução

```
void loop() {
    for(int j=0; j<100; j++) {
        for(int i=1; i<10000; i++) {
            int n = 1000*i;

            int t1 = 0;
            int t2 = 1;
            int nextTerm = 0;

            nextTerm = t1 + t2;

            while(nextTerm <= n)
            {
                t1 = t2;
                t2 = nextTerm;
                nextTerm = t1 + t2;
            }
        }
    }
    asm volatile("sleep"::);
}
```

Listagem 4.4: Teste calculando a sequência de Fibonacci

até esse momento mas optámos por outro método. O que fizemos foi no fim da tarefa a realizar inserir no código uma instrução especial, neste caso um `sleep`, e no simulador terminamos a execução quando essa instrução é encontrada. Escolhemos o `sleep` porque o compilador AVR não gera essa instrução em código normal de Arduino (a API não usa modos de *sleep*). Temos de ser nós a pedir ao compilador para a inserir explicitamente com o código `asm volatile("sleep"::)`.

De modo a podermos comparar resultados, foi inserido no código do simulador, de ambos os testes, chamadas a funções para recolha de tempo de execução de cada instrução AVR, neste caso `System.nanoTime()`, que mede o tempo em nanosegundos. Estas chamadas implicam alguma perda de desempenho uma vez que são chamadas muito frequentemente. Apesar disso são absolutamente necessárias para podermos fazer a comparação entre as duas técnicas e afectam as duas da mesma maneira, portanto, em termos de comparação podem ser ignoradas.

Os testes foram realizados numa máquina com processador Intel® Core™ i7 Q 740 a 1.73GHz, com 4 *cores* e 8 *threads*, e 8GB de RAM. A máquina virtual Java usada foi a Java HotSpot™ 64-Bit Server, versão 1.8.0_201.

Após correr as primeiras simulações, tanto interpretada como compilada, como podemos ver na Tabela 4.1, os tempos de execução são muito díspares, e para mais, a simulação compilada mostra tempos muito piores do que a interpretada, na ordem das 15 vezes mais lenta.

Normalmente, a simulação compilada consegue velocidades de execução significativamente maior do que a interpretação (Leupers et al., 1999), que pode chegar a

Tabela 4.1: Tempos de Execução

Tipo	Instruções executadas	Tempo[1]	Velocidade
Simulação interpretada	183606001	12904ms	14.228MHz
Simulação compilada	183606001	187794ms	0.977MHz

ser na ordem dezenas ou centenas (Pees, Zivojnovic, Ropers & Meyr, 1997).

Para esclarecer estas diferenças relativamente à literatura consultada, fizemos uma análise mais profunda dos tempos de execução das instruções. Medimos o tempo de execução de cada instrução, guardámos em memória, e no final das simulações guardámos para um ficheiro para posterior análise.

Podemos confirmar no gráfico da Figura 4.9, que mostra o tempo de execução de cada 10 mil instruções, que a simulação interpretada é mais rápida do que a simulação compilada ao longo do tempo.

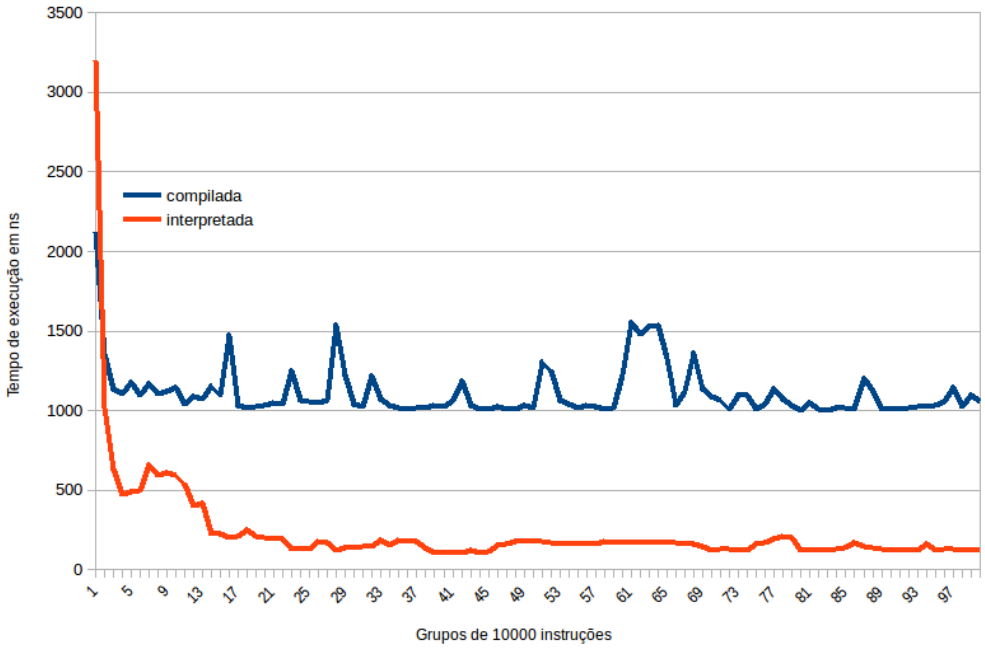


Figura 4.9: Tempo de execução em grupos de 10000 instruções

Mas a parte inicial do gráfico da Figura 4.9 merece alguma atenção pois existe um cruzamento entre a linha da simulação interpretada e a da simulação compilada. Vamos então fazer uma análise dessa parte inicial da simulação. No gráfico da Figura 4.10, focado no início da simulação, e com o tempo de execução agrupado a cada 100 instruções, podemos verificar que a simulação compilada começa por ser mais rápida do que a interpretada, mas a situação vai-se invertendo ao longo do tempo.

Depois de alguma pesquisa sobre o que poderia estar a acontecer, descobrimos que a *Java Virtual Machine* (JVM) não compila para código nativo métodos com mais de 8k bytes (Turner, 2009), e logo não compila os métodos onde é implementada a compilação simulada, que são bastante grandes. Isso pode ser confirmado

[1]Sem recolha de tempo de execução de cada instrução

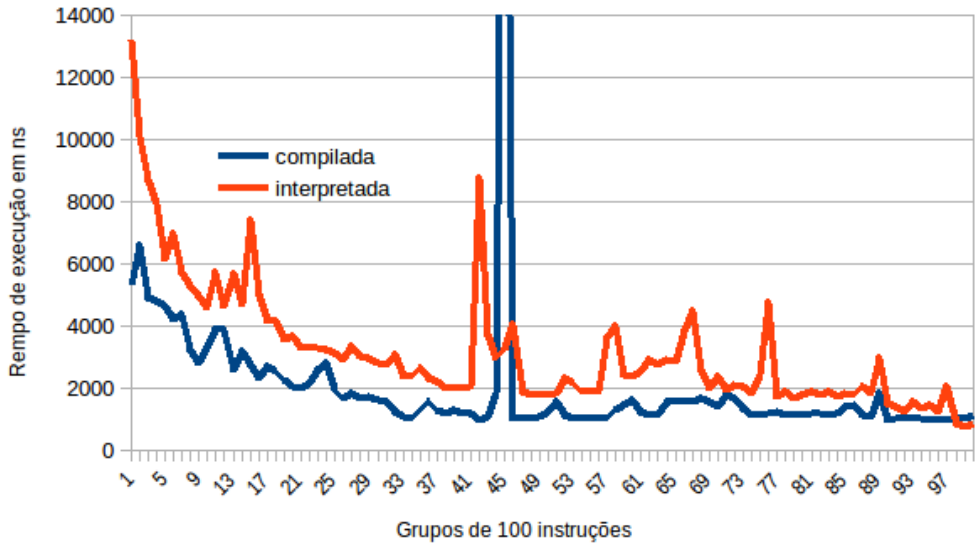


Figura 4.10: Tempo de execução em grupos de 100 instruções

executando a JVM com o parâmetro `-XX:+PrintCompilation` (Oracle, 2018) que imprime na saída padrão uma mensagem com o nome de cada método que está a ser compilado para código nativo. Desse modo verificámos que os métodos da simulação interpretada são compilados pelo JIT do Java, e os da simulação compilada não.

Podemos tentar ultrapassar esse problema instruindo a JVM para compilar métodos para código nativo usando o parâmetro `-XX:-DontCompileHugeMethods` (atenção que o sinal menos antes do parâmetro inverte o seu significado). Fazendo uma nova análise dos tempos de execução, vemos na Figura 4.11, que os tempos de execução inicias não são bons, mas depois ficam muito similares ao da simulação interpretada. A inicialização lenta é devido à JVM começar sempre por executar o *byte code* em modo de interpretação e só depois de recolher uma série de métricas sobre o código executado, em *runtime*, é que decide se compila esse *byte code* para código nativo ou não. Na JVM Server, o compilador JIT, executa o método 10 mil vezes no modo interpretativo, para recolher informação para fazer uma compilação eficaz (Oracle, 2018), antes de compilar o método para código nativo. No caso da simulação compilada isso também acontece, mas, como podemos ver na Listagem 4.3, as funções `executeX` agregam muitas instruções, que se forem executas de forma sequencial, não implicam uma nova chamada a essa função, logo são executadas muito menos vezes do que na simulação interpretada, o que por sua vez implica que a activação do JIT demore mais tempo.

Mesmo assim não verificamos o aumento de desempenho indicado por Leupers et al., 1999 e Pees et al., 1997. Prevemos que esta situação se deva a estarmos a executar a própria simulação numa máquina virtual (a JVM), não conseguindo a optimização que seria possível obter compilando directamente para o ISA do *host*.

4.6.5 Optimizações

Já temos uma optimização na descodificação das instruções como vimos na Subsecção 4.6.1, mas ainda podemos fazer outra. Podemos fazer também a extracção dos parâmetros da instrução antes de iniciar a execução e guardar esses valores numa *ca-*

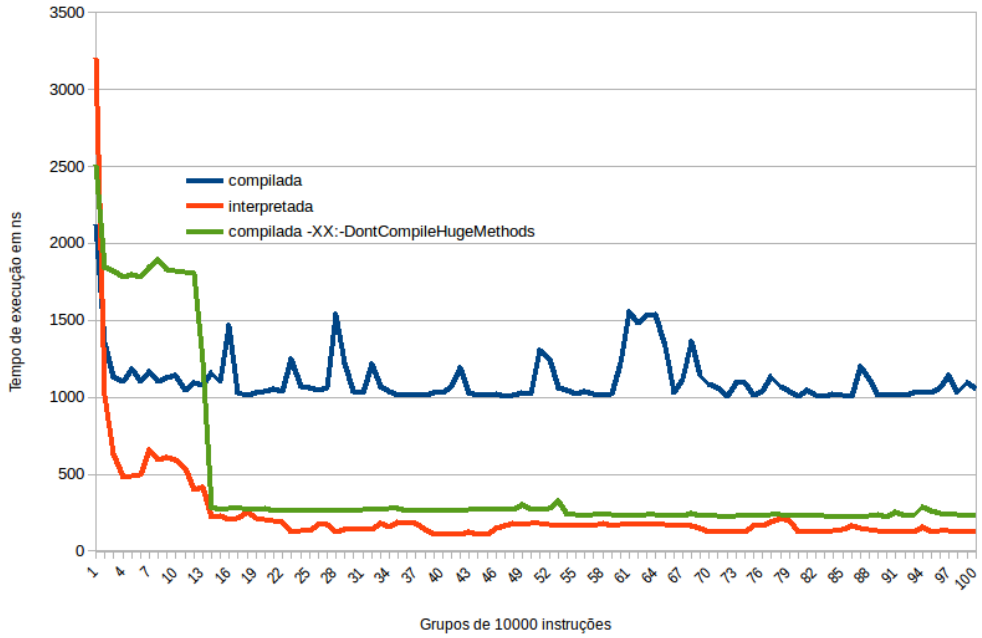


Figura 4.11: Tempo de execução com -DontCompileHugeMethods

che. No caso da memória de programa ser alterada em tempo de execução é possível refazer a *cache* no momento da escrita da memória.

Tabela 4.2: Tempos de Execução com e sem pré-decode

Tipo	Instruções executadas	Tempo[2]	Velocidade
Simulação interpretada	183606001	12904ms	14.228MHz
Simulação interpretada com pré *decode*	183606001	9945ms	18.462MHz

Podemos verificar na Tabela 4.2 que esta optimização trouxe uma melhoria significativa no tempo de execução, permitindo um aumento na velocidade de processamento de cerca de 4MHz, alcançando assim uma velocidade absoluta de mais de 18MHz, ultrapassando a de execução em tempo real do microcontrolador do Arduino, que é de 16MHz.

4.6.6 Escolha da técnica de virtualização

Quanto à melhor técnica de virtualização para implementar um simulador para a plataforma Arduino, programado em Java, escolhemos a simulação interpretada. É a técnica que dá mais flexibilidade no controlo do processador virtualizado, permite o carregamento directo do código binário a executar, não necessita de um passo intermédio de criação de código fonte Java e posterior compilação, permite a alteração do programa simulado em tempo de execução, é fácil de implementar e acaba por ter melhor desempenho do que a compilação simulada quando executada numa JVM.

Especificamente, a simulação compilada tem os seguintes problemas:

[2]Sem recolha de tempo de execução de cada instrução

- A instrução AVR SPM escreve na FLASH, o que no caso de ser executada invalidaria imediatamente o programa compilado, obrigando à geração de novo código fonte e nova compilação. Esta instrução é usada para alterar o código dinamicamente tal como aconteceria, por exemplo, no carregamento de um novo *firmware* por parte do *bootloader*.

- A instrução AVR LPM acede à memória FLASH para carregar valores constantes, portanto é necessário instanciar e inicializar essa memória mesmo que não seja necessária para a implementação da simulação compilada.

- As interrupções só ocorreriam depois de um *break* no *switch* de controlo do programa (JUMP, CALL, RET, etc) se fossem testadas no início do ciclo infinito ou então teria de fazer esse teste antes ou após a execução de cada instrução. Fazer isso implicaria uma repetição exaustiva da mesma lógica a cada instrução simulada.

- Seria muito mais difícil de implementar o *debugging*. Tal como na situação anterior não existe um sítio onde se possa fazer o controlo do código antes ou após a execução de cada instrução de modo a, por exemplo, pausar a execução do programa.

Capítulo 5

Implementação

O desenvolvimento do projecto foi feito com base em metodologias ágeis (Agile Alliance, 2020). Optámos por este tipo de metodologia por permitir a adaptação do projecto ao longo do tempo, fazendo os ajustes necessários às tarefas a desenvolver de acordo com os imprevistos que vão aparecendo. Neste tipo de projectos, em que uma só pessoa tem todos os papéis desde a gestão do projecto, à definição de requisitos, ao desenho da arquitectura, à implementação e aos testes, dá mais liberdade nas mudanças de direcção que possam ocorrer e permite aproveitar ideias de momento evitando que se percam na burocracia de um processo mais formal.

Este tipo de metodologia permite também, por exemplo, que se crie uma interface com funcionalidades básicas apenas para testar o desenvolvimento do *core* do simulador e de modo a dar um *feedback* visual ao programador. Também permite apresentar o trabalho realizado antes de estar completo como foi o caso da primeira acção de divulgação deste projecto, no ArduinoDay 2019 no Instituto Politécnico de Tomar, onde apresentámos uma versão *alpha* do simulador. Portanto dá-nos a possibilidade de ter a qualquer momento um produto funcional, o que também ajuda na motivação de quem está a desenvolver o projecto, pois ao testar constantemente o sistema é possível encontrar problemas mais cedo, evitando a frustração de trabalhar vários dias numa funcionalidade para mais tarde vir a descobrir que não cumpre os requisitos necessários e que é preciso voltar a repetir o trabalho.

Devemos fazer notar que não se seguiu nenhuma metodologia ágil em particular, mas que apenas foram aplicados alguns princípios ágeis como o foco num conjunto limitado de funcionalidades de cada vez como nas Sprints do Scrum (Scrum.org, 2020), os testes constantes como no Extreme Programming (Wells, 2013) e a abordagem incremental como no Kanban (Atlassian, 2020).

Dividimos a implementação do projecto em três subprojectos:

- Um em que implementámos a arquitectura AVR e o microcontrolador ATmega328P;
- Outro em que implementámos uma aplicação web. Neste projecto existe o código com a lógica da aplicação que executa no servidor em si e a parte que executa no *browser* do cliente incluindo todas as páginas *HyperText Markup Language* (HTML) e o código Javascript;
- E um terceiro onde implementámos a ferramenta de programação do simulador a inserir no IDE do Arduino.

5.1 Simulação do microcontrolador

Implementámos a simulação do microcontrolador ATmega328P num projecto Maven (Apache Maven, 2020) separado e sem qualquer dependência externa, sendo portanto implementada puramente em Java e completamente de raiz. Optámos por um projecto separado para que possa ser usado noutros produtos caso seja necessário.

Na Secção D.1 do Apêndice D podemos ver o diagrama de classes do projecto. A classe central é a classe `CPU`. Esta classe tem instâncias de classes que representam a SRAM (`DataMemory`), a FLASH `ProgramMemory` e uma lista de periféricos que implementam a interface `Peripheral`. Contém também uma instância da classe `InstructionDecoder` que dá acesso a todas as instruções do ISA, classes que implementam a interface `Instruction`.

Este projecto não dá apenas suporte ao microcontrolador usado no Arduino Uno, implementa o ISA AVR, e tem suporte para todas as 5 famílias existentes: AVR, AVRe, AVRxm, AVRxt e AVRrc (Atmel, 2016), embora não tenham sido implementadas algumas instruções. Não implementámos a instrução DES porque seria algo complexa de implementar e não é usada no ATmega328P. A instrução SPM também não foi implementada pois no simulador não foi usado o *bootloader* e portanto não é necessária. A instrução SLEEP também não foi implementada porque não foram implementados os restantes mecanismos para dar suporte aos vários estados de *sleep* do microcontrolador. Como não foram implementados os mecanismos de *sleep* a instrução WDR, apesar de implementada, é uma no-op.

O projecto começou com a criação de um Instruction Set Parser, um programa auxiliar que não está incluído no projecto Maven, para descodificar as instruções. Este programa faz a transformação do PDF do ISA AVR em texto simples ao qual são de seguida aplicadas *regular expressions* para extrair as informações pertinentes para a descodificação das instruções.

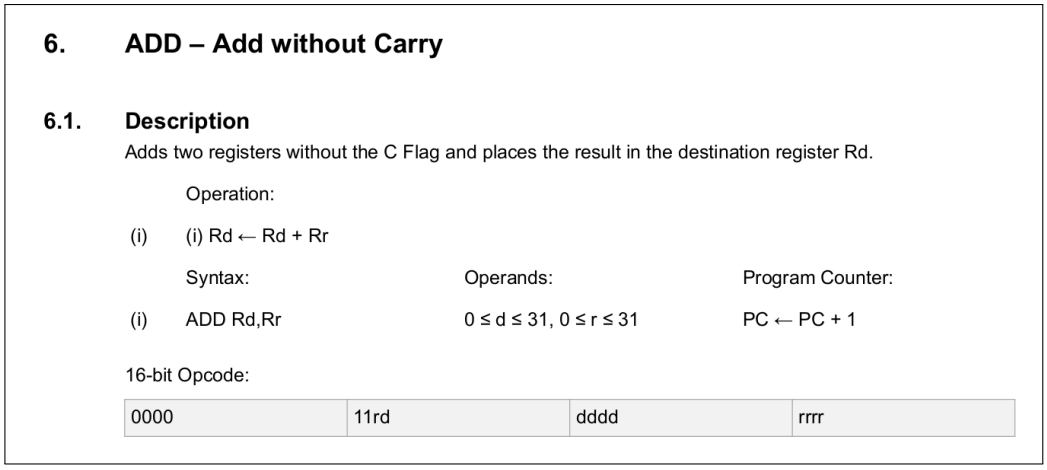
De seguida foi desenvolvido um *disassembler*. Foi a forma mais eficaz que encontrámos para ir validando o código de descodificação das instruções ao longo do trabalho.

5.1.1 Descodificação das instruções

Tal como referido na Subsecção 4.6.1, a descodificação das instruções no *Instruction Set* AVR pode ser algo desafiante. Dado o grande número de instruções existentes recorremos a programas auxiliares para ajudar a criar o código de descodificação das instruções. Desenvolvemos software para fazer o *parsing* ao ficheiro da especificação do ISA, um PDF, de modo a identificar todas as instruções existentes. Usámos a biblioteca iText (iText, 2020) para fazer o *parsing* do PDF para texto e depois foram usadas expressões regulares para identificar a instrução e, para a descodificação da instrução, o mapeamento de bits da mesma.

Um exemplo da combinação de bits do *opcode* que corresponde a cada instrução pode ser visto na Figura 5.1.

Para fazer a descodificação existe a hipótese de usar um *switch* nos bits fixos que identificam a instrução. Essa hipótese rapidamente se tornou complexa de implementar pois verificámos que não há um número de bits único onde o *opcode* é fixo. Por exemplo, na instrução ADD os primeiros 6 bits da instrução são fixos, sendo



6. ADD – Add without Carry

6.1. Description

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) (i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Figura 5.1: Excerto do PDF do *ARV Instruction Set Manual* (Atmel, 2016)

os restantes índices para os operandos mas nem todas as instruções têm esse formato. A instrução LDD tem 3 bits que representam operandos precisamente no mesmo local (os 6 primeiros bits). Por essa razão e por forma a tornar a descodificação mais rápida optámos por usar uma tabela de mapeamento com todos os valores possíveis de 16 bits (tamanho da instrução) para a respectiva instrução. Deste modo a descodificação passa ser uma rápida *lookup table* (Wikipedia contributors, 2020c).

Para auxiliar no cálculo dos valores correspondentes a cada instrução usámos o código na Listagem 5.1. Após a identificação dos bits da instrução é necessário substituir todos os bits variáveis, que são representados por letras, por todas as suas possibilidades de 1 e 0 e desse modo identificar todos os *opcodes* de 16 bits que correspondem a essa instrução. Este conjunto de valores é depois mapeado com a instrução correcta na classe `InstructionDecoder`.

Uma possibilidade para mapear os *opcodes* com a instrução correspondente seria a classe `InstructionDecoder` ter um mapeamento com o tipo de classe a instanciar e o objecto seria criado em tempo de execução. O Java é conhecido por ser lento a criar objectos (Klemm, 1999) e por isso optámos por criar apenas um objecto de cada tipo de instrução sendo devolvido pela `InstructionDecoder` sempre o mesmo objecto para cada tipo de instrução independentemente do endereço de memória onde esta se encontre, e portanto, podendo ter operandos diferentes. Deste modo o contexto de execução da instrução não pode fazer parte do objecto, pois o mesmo é usado no contexto da execução de várias instruções do mesmo tipo, pelo que os operandos terão de ser passados por parâmetro para o método `execute` da instrução.

Antes de começar a implementar as instruções tiveram de ser implementadas as estruturas de suporte para poder ser testado o código que ia sendo implementado. Numa fase mais inicial bastou implementar a memória de programa (FLASH), um carregador de ficheiros Intel HEX, um dos formatos de código compilado gerado pelo IDE do Arduino, a memória de dados (SRAM) e mais algumas estruturas do CPU.

A classe `ProgramMemory`, que implementa a FLASH, é muito simples sendo apenas um *wrapper* de um *array* com a adição de algumas validações.

Para fazer o carregamento dos vários tipos de ficheiros binários, neste caso Intel HEX e *Executable and Linkable Format* (ELF) (Tool Interface Standards Committee

```
private String[] decodeToArray(Map<String,List<String>> opCodes) {
    Map<Integer,String> map = new TreeMap<>();

    for(Map.Entry<String,List<String>> e : opCodes.entrySet()) {
        String opCode = e.getKey();
        for(String bits : e.getValue()) {
            bits = bits.replace(" ", "").substring(0, 16);
            for(String bitsExpanded : expandBits(bits)) {
                Integer code = Integer.parseUnsignedInt(bitsExpanded, 2);
                map.put(code, opCode);
            }
        }
    }

    String[] tmp = new String[0xFFFF+1];
    for(int i=0; i<=0xFFFF; i++) {
        tmp[i] = map.get(i);
    }
    return tmp;
}

private List<String> expandBits(String bits) {
    List<String> list = new ArrayList<>();
    List<String> temp = new ArrayList<>();
    temp.add(bits);
    while( !temp.isEmpty() ) {
        boolean changed = false;
        String tempStr = temp.remove(0);
        for(int i=0; i<tempStr.length(); i++) {
            char b = tempStr.charAt(i);
            //troca letras por 0 e 1 e coloca na temp
            if( b!='0' && b!='1' ) {
                changed = true;
                char[] bb = tempStr.toCharArray();
                bb[i] = '0';
                temp.add(new String(bb));
                bb[i] = '1';
                temp.add(new String(bb));
                break;
            }
        }
        //se nao fez trocas terminou; coloca na lista
        if( !changed ) {
            list.add(tempStr);
        }
    }
    return list;
}
```

Listagem 5.1: Expansão dos bits das instruções

and others, 2001), criámos a classe abstracta `ProgramMemoryLoader` que define métodos para carregar a FLASH e a EEPROM a implementar pelas classes específicas para cada tipo de ficheiro, além de servir como fábrica (padrão Fábrica (Wikipedia contributors, 2020d)) para as mesmas.

5.1.2 Implementação da SRAM

A classe `DataMemory`, que implementa a SRAM, tem alguma complexidade porque nestes processadores os registos dos periféricos estão mapeados nessa memória. Isto implica que quando houver uma alteração do valor desses registos pode existir a necessidade de passar o controlo da simulação ao código do respectivo periférico. Também existem registos que são duplos, ou seja, o valor quando se lê o registo pode não ser igual ao valor que lá se escreveu a última vez. Isto acontece, por exemplo, nos registos das portas de I/O do microcontrolador.

Para implementar estas funcionalidades existem diversas alternativas. A primeira consiste em usar *multithreading* e o sistema *notify*/*wait* do Java. Com este método podemos bloquear em toda a memória e esperar por uma alteração, no entanto tem o problema de depois não se saber qual foi o endereço que foi alterado. Também podemos bloquear endereço a endereço, mas se precisarmos de bloquear em vários endereços e a ordem pela qual as zonas de memória são alteradas não for previsível podemos provocar *deadlocks*. Imaginemos que queremos ser notificados de alterações no endereço A e no B: se chamarmos A.wait(), bloqueando nesse local esperando pela sua alteração, e alterarmos o B a *thread* não vai desbloquear. A segunda alternativa consiste em usar o padrão Observador (Gamma, 1995) e registar *listeners* para serem chamados aquando da alteração dos valores da SRAM. Deste modo pode ser passado o valor do endereço modificado para a *callback* o que permite registar uma única *callback* para vários registos do processador. Optámos por esta segunda alternativa.

De modo a optimizar a velocidade de execução do processador virtual a execução das *callbacks* é feita numa *thread* à parte que é criada quando o processador inicia a execução e terminada quando o processador pára a execução.

Também usámos *listeners* para registar *callbacks* nos endereços em que é necessário ler *segundos* valores dos registos duplos.

5.1.3 Implementação das instruções

Para a implementação das instruções criámos uma classe Java com um *template* (ver Listagem 5.2), que foi depois duplicada, substituindo o nome da classe e o texto devolvido pelo método `getName` pelo da nova instrução. Este processo foi mais uma vez automatizado recorrendo a um programa auxiliar para evitar copiar e substituir manualmente o texto na enorme quantidade de classes existentes, evitando deste modo erros de introdução manual e dando a possibilidade de repetir o processo em caso de necessidade.

Para testar a correcta descodificação fizemos a desassemblagem de um programa criado no IDE do Arduino com o avr-objdump (Atmel, 2019). Este programa dá uma listagem de texto com as mnemónicas das instruções de máquina a partir de um ficheiro objecto. Carregámos o mesmo programa no processador virtual e chamámos a função `getASM` de cada instrução após a descodificação da mesma. Comparando

```
public class TEMPLATE extends BaseInstruction {

    @Override
    public void execute(CPU cpu, int op1, int op2) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public String getASM(CPU cpu, int op1, int op2) throws Exception {
        if( cpu.getIncInDisassemble() ) {
            cpu.incPc();
        }
        return getName();
    }

    @Override
    public InstructionParams decode(CPU cpu, int opcode) {
        return params;
    }

    @Override
    public String getName() {
        return "TEMPLATE";
    }
}
```

Listagem 5.2: Classe template de instrução

a saída dos dois sistemas foi possível validar o correcto funcionamento das classes desenvolvidas.

O próximo passo foi a implementação da execução das instruções. Todas as instruções foram criadas inicialmente com um método `execute` sem funcionalidade e que lança uma excepção. Isto foi intencional de modo a que quando se colocasse o microcontrolador virtual a executar um programa, no momento em que fosse encontrada uma instrução não implementada, o mesmo suspendesse a execução lançando uma excepção. Deste modo foi possível carregar um binário e ir implementando as instruções, uma a uma, conforme iam sendo necessárias usando um método semelhante a *test driven development* (Beck, 2003), em que o objectivo não é passar um teste unitário mas sim executar uma instrução sem que seja lançada uma excepção. Este método permitiu-nos ir testando a implementação das instruções ao longo do desenvolvimento em vez de fazer todo o código e apenas testar no final.

Na Listagem 5.3 podemos ver o código do método `execute` da instrução ADC. Neste caso são carregados os valores de 2 registos, é feita a operação, o resultado é guardado num registo, são calculadas as novas *FLAGS* do processador e é incrementado o *program counter*.

Podemos verificar que os operandos são passados por parâmetro tal como referido na Subsecção 5.1.1.

A descodificação dos operandos é feita no método `decode`. Na Listagem 5.4

```
public void execute(CPU cpu, int rdAddr, int rrAddr) {
    DataMemory.StatusRegister status =
        cpu.getSRAM().getStatusRegisterObj();

    int rd = cpu.getSRAM().getRegister(rdAddr);
    int rr = cpu.getSRAM().getRegister(rrAddr);

    int r = rd + rr + (status.getCarry()?1:0);
    cpu.getSRAM().setRegister(rdAddr, r);

    status.setHalfCarry( bit(rd,3)&bit(rr,3) | bit(rr,3)&notbit(r,3) |
        notbit(r,3)&bit(rd,3) );
    status.setOverflow( bit(rd,7)&bit(rr,7)&notbit(r,7) |
        notbit(rd,7)&notbit(rr,7)&bit(r,7) );
    status.setNegative( bit(r,7) );
    status.setZero( (r&0xFF)==0 );
    status.setCarry( bit(rd,7)&bit(rr,7) | bit(rr,7)&notbit(r,7) |
        notbit(r,7)&bit(rd,7) );
    status.setSign( status.getNegative()!=status.getOverflow() );

    cpu.incPc();
}
```

Listagem 5.3: Método execute da instrução ADC

podemos ver como é feita a descodificação da instrução CALL. É passado por parâmetro o objecto `CPU` e o *opcode* a descodificar. O objecto `CPU` é necessário em algumas instruções em que apenas o *opcode* não é suficiente, como nas instruções de 32 bits nas quais é preciso fazer o *fetch* do *opcode* do endereço seguinte na memória.

O `decode` não é chamado dentro do método `execute` porque podemos fazer uma optimização e chamá-lo no momento em que a FLASH é carregada, fazendo a pré-descodificação das instruções para uma *cache*, tornando a execução mais rápida. Esta optimização obriga apenas ao cuidado de refazer a *cache* no momento em que for executada a instrução SPM [1] que escreve na FLASH.

Há parâmetros de instruções com sinal cujo tamanho em número de bits não corresponde a nenhum tipo de dado em Java. Por exemplo a instrução RJMP tem um parâmetro com sinal com 12 bits. Neste caso temos de usar uma variável do tipo *short* de 16 bits que é a que existe com um menor tamanho superior a 12. Deslocar o bit de sinal para o sítio certo e fazer com que o Java reconheça esse sinal não é tão simples como possa parecer à primeira vista. A melhor maneira é deslocar para a esquerda até o bit ficar no local correcto e depois dividir por $2^{n\_deslocamentos}$. Isto funciona porque o deslocamento para a esquerda equivale a uma multiplicação por 2 mas como é feita ao nível binário, não afecta o sinal do valor. Depois, ao fazer a divisão o valor é reposto mas como a divisão não é feita a nível binário o Java mantêm o sinal no local correcto e obtemos o valor pretendido.

Na interpretação do *datasheet* houve algumas dúvidas sobre a ordem da afectação

[1] A instrução SPM não foi implementada neste projecto

```
public InstructionParams decode(CPU cpu, int opcode) {
    //CALL 1001 010k kkkk 111k
    //      kkkk kkkk kkkk kkkk
    int opcode1 = opcode;
    int opcode2 = cpu.fetchNext();

    params.op1 = (opcode1>>3)&0x3E | opcode1&0x1;
    params.op1 = (params.op1<<16)| opcode2;
    return params;
}
```

Listagem 5.4: Método decode da instrução CALL

de *flags* na instrução *Arithmetic Shift Right* (ASR) pois uma *flag* dependia de outra e era necessário saber qual calcular primeiro. Para tirar a dúvida fizemos um teste no hardware real para ver o efeito sobre as *flags*. O código do teste feito pode ser visto na Listagem 5.5.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  char value;
  asm volatile("LDI R20, 1" ::);
  asm volatile("ASR R20" ::);
  asm volatile("in %0, %1" : "=r" (value) : "I" (_SFR_IO_ADDR(SREG)));
  Serial.println(value, HEX);
  delay(1000);
}
```

Listagem 5.5: Código Arduino para testar a instrução ASR

Com o teste feito, e analisando o resultado, foi possível concluir que a *flag* Overflow (V) deve ser calculada antes da *flag* Signal (S).

5.1.4 A classe CPU

A classe abstracta `CPU` tem a responsabilidade de ligar todas as outras classes que implementam o microcontrolador. Esta classe tem uma instância da `DataMemory`, que implementa a SRAM, uma instância da `ProgramMemory` que implementa a FLASH, uma instância da classe `InstructionDecoder` que faz o mapeamento entre um *opcode* e uma classe que implementa uma instrução, tem o controlo do *program counter* e ainda uma lista de periféricos.

O método `execute` (ver Listagem 5.6) da classe `CPU` começa por chamar o *clock* de todos os periféricos, de seguida verifica se existem interrupções pendentes, o que pode alterar o fluxo da execução, e só depois é que executa a instrução apontada

pelo *program counter*.

```
public synchronized void execute() throws Exception {
    clock();

    if( interrupt &&
        getSRAM().getStatusRegisterObj().getGlobalInterruptEnable() ) {
        int previousPC = getPc();
        if( checkInterrupt() ) {
            if( getPCSize()==16 ) {
                getSRAM().setStackW(getSRAM().getStackPointer(), previousPC);
                getSRAM().setStackPointer(getSRAM().getStackPointer()-2);
            } else {
                getSRAM().setStack3(getSRAM().getStackPointer(), previousPC);
                getSRAM().setStackPointer(getSRAM().getStackPointer()-3);
            }
            getSRAM().getStatusRegisterObj().setGlobalInterruptEnable(false);
        } else {
            interrupt = false;
        }
    }

    ....

        int instructionValue = getFLASH().get(getPc());
        Instruction instruction;

        if( instructionValue==BREAK_OPCODE ) {
            instruction = breakpoints.get(getPc());
            if( instruction==null ) {
                instruction = new BREAK();
                breakpoints.put(getPc(), (BREAK)instruction);
            }
        } else {
            instruction =
                instructionDecoder.getInstruction(instructionValue);
        }
        InstructionParams params = getFLASH().getInstructionParams(getPc());
        instruction.execute(this, params.op1, params.op2);
    }
}
```

Listagem 5.6: Extracto do método execute da classe CPU

As situações em que a instrução é um *breakpoint* são tratadas de um modo especial neste local. Como o *opcode* do BREAK é sempre o mesmo, no caso de ter sido adicionado um *breakpoint* em *runtime*, é necessário saber qual a instrução que ocupava o seu lugar anteriormente. Então não é possível usar apenas o mapeamento entre o *opcode* e a instrução uma vez que o objecto da classe `BREAK` guarda a instância do objecto correspondente à instrução que substituiu. Por isso, e apenas

neste caso, tem de ser consultado um mapeamento entre o endereço de memória e a instrução que é feito ao instalar um *breakpoint*. Ao instalar um *breakpoint* é substituído o *opcode* na FLASH e é criado um objecto do tipo `BREAK` que guarda a instrução original. Este objecto é mapeado com o endereço actual. Ao remover o *breakpoint* é consultado o mapeamento para saber qual a instrução original, é feita a substituição novamente na memória FLASH e removido o mapeamento. Na execução da instrução BREAK, o objecto mantém uma variável booleana que permite que a primeira vez que a instrução é executada seja lançada uma excepção para parar o processador. Na próxima vez que a instrução é executada é delegada a execução para a instrução que o *breakpoint* substituiu e é reposto o estado inicial para que da próxima vez seja novamente parado o processador.

Constatámos que podemos melhorar este método movendo o teste do tipo de instrução no ciclo de execução (`instructionValue==BREAK_OPCODE`) para a lógica da classe `BREAK` tornando o código mais rápido uma vez que o teste deixa de ser feito em cada iteração da fase de execução da simulação. Este melhoramento será feito na próxima iteração de desenvolvimento.

A classe `CPU` é abstracta com o objectivo de poder ser estendida para implementar microcontroladores específicos, como o ATmega328P, que têm os seus próprios registos, tamanhos de FLASH e SRAM diferentes, número de portas de entrada e saída diferentes e conjuntos de periféricos específicos. O construtor desta classe recebe por parâmetro objectos que permitem configurar o ISA, a FLASH e a SRAM e tem mecanismos para que cada subclasse tenha os seus próprios periféricos. Neste projecto implementámos a classe `ATmega328P` para dar suporte ao microcontrolador com o mesmo nome.

Para o funcionamento do relógio, inicialmente planeámos usar um *ScheduledExecutorService*, um mecanismo já existente na API do Java, para criar tarefas que são executadas periodicamente com um tempo de intervalo fixo. Para simular um relógio de 16MHz, a velocidade usada no Arduino Uno, seria necessário usar um tempo de intervalo de 63 nanossegundos entre cada execução da tarefa. Esta situação tornou-se inviável uma vez que apesar do Java permitir, em termos de API, *sleeps* de nanossegundos os sistemas operativos modernos não o permitem. No Linux o período de interrupção é de cerca de 1 milissegundo e no Windows entre 10 a 15 milissegundos [2]. Por esse motivo e porque uma execução sem tempos de espera (na máquina onde fizemos os testes inicias à técnica de interpretação e sem muita da lógica necessária a mecanismos internos do microcontrolador como interrupções ou eventos de alterações de registos) não era muito mais rápida do que a execução em tempo real do ATmega328P a 16MHz, optámos por não usar um *ScheduledExecutorService* ou qualquer outro mecanismo para controlar o tempo de execução do microcontrolador. Mais tarde, com a implementação de mais lógica interna do microcontrolador, este passou até a executar mais lentamente do que o hardware real. Esta diferença de tempo de execução é mais importante no periférico *TIMER0* do que na execução das instruções e veremos na Subsecção 5.1.5 como ultrapassámos a situação para o *TIMER0*.

[2] https://stackoverflow.com/a/11498647/662855

5.1.5 Periféricos implementados

Implementámos os seguintes periféricos que devem ser suficientes para a maioria dos exercícios introdutórios a microcontroladores:

- **GPIOs**: Permitem a utilização normal dos pinos de I/O e as respectivas interrupções nesses pinos;
- **TIMER0**: Para a implementação das funções `delay` e `millis` da API do Arduino;
- **USART0**: Para a porta série. Permite ao utilizador enviar e receber dados pelo monitor série na interface Web (não muda os valores dos pinos TX na placa nem lê dados do pino RX);
- **ADC**: Permite a conversão de valores analógicos para digitais. Existe a possibilidade de adicionar um potenciómetro ao circuito que os utilizadores podem usar para enviar valores analógicos para o sistema. O pino *analog ref* pode ser usado para colocar um valor de referencia para ser usado pelo conversor ADC.

Relativamente ao *TIMER0* fizemos duas implementações. Uma que implementa a funcionalidade quase na sua totalidade, respeitando os valores de *prescaler* (divisor do relógio principal do sistema) excepto as configurações que dizem respeito a uma origem de relógio externa, e ligado ao *clock* do CPU tal como no hardware verdadeiro. Esta implementação não era prática para a utilização do simulador pois a velocidade de relógio do processador virtual não é a mesma do processador verdadeiro e isso tem influência no método `delay` da API do Arduino. Pretendemos que quando se programe um tempo de espera de 1 segundo num programa de Arduino esse tempo seja o mais aproximado possível à realidade no momento de correr a simulação. Por isso fizemos uma segunda implementação do *TIMER0*, numa classe chamada `Timer0Dummy`, que ignora as configurações do *prescaler* e assume que está a executar um programa de Arduino que configura sempre o *TIMER0* para gerar interrupções a cada milissegundo. Deste modo a contagem do tempo nos programas de Arduino executados no simulador têm a aparência do tempo real. Isto dá um *feedback* aos utilizadores do simulador muito aproximado ao real tendo os tempos de espera definidos pelo método `delay` da API uma aproximação muito grande ao esperado.

Decidimos não implementar periféricos que provocam variações muito rápidas nos pinos de saída do microcontrolador porque essa funcionalidade provoca um grande atraso na propagação da informação do servidor (onde a simulação está a ser executada) para o cliente (onde está a interface com o utilizador), provocando lentidão na simulação, deixando portanto a mesma de ser prática. Estes periféricos são o PWM e todas as comunicações série como USART, SPI e I^2C. Implementámos a porta USART0 mas a comunicação é redireccionada para um *listener* (usando o padrão Observador), em vez dos pinos normais do microcontrolador, permitindo ao cliente registar-se e obter os dados enviados para a porta série. Apesar disso todas as configurações da porta série são respeitadas e as interrupções também são geradas.

O periférico ADC foi implementado com todas as suas funcionalidades, incluindo a selecção de entrada, as tensões de referência internas e externa, o ajuste do resultado e a geração de interrupções. Para permitir a leitura de valores analógicos o valor apresentado nos pinos de entrada do microcontrolador simulado é um valor

decimal. Quando se trabalha com lógica binária, uma saída LOW é representada pelo valor decimal 0,0 e uma saída HIGH é representada pelo valor decimal 5,0. No caso das entradas um valor abaixo de 2,25 é considerado LOW e maior ou igual é considerado HIGH, isto porque na folha de características do ATmega328P um valor abaixo de 1,5 volts é considerado um valor LOW e acima de 3 volts é considerado HIGH, e decidimos usar no simulador o valor intermédio como fronteira. Um pino aberto (sem estar ligado a nada) num microcontrolador real tem um efeito de antena capturando os valores de tensão flutuantes no ar o que normalmente resulta numa variação constante do valor de tensão apresentado ao microcontrolador. Esse efeito também é simulado sendo que quando o valor do pino é colocado a NULL é gerado internamente um valor aleatório para essa entrada que varia a cada 100 milissegundos. Deste modo obtém-se um efeito bastante realista da utilização dos pinos de entrada, tanto no modo analógico como digital, obrigando mesmo a usar a opção `INPUT_PULLUP` da API do Arduino caso se queira usar uma entrada digital com um botão no circuito virtual tal como teria de se fazer num Arduino real.

A implementação das entradas/saídas com valores decimais também poderá permitir a simulação de PWM colocando um valor decimal intermédio (entre 0,0 e 5,0) no pino de saída. Este valor corresponderá ao *duty cycle*, que é a percentagem de tempo que o sinal está activo num determinado período, ou seja, a relação entre o tempo que o sinal está HIGH e LOW. Este método não funcionará para ligar estas saídas a servomotores simulados pois este tipo de motor não é posicionado pelo *duty cycle* mas sim pela largura do sinal HIGH podendo a frequência variar (dentro de limites) sem que a posição do servomotor seja alterada. O PWM não foi implementado deste modo e fica proposto como trabalho futuro.

5.2 Plugin do IDE Arduino

Sendo a integração com o IDE do Arduino uma das características mais importantes neste projecto, reflectimos com muito cuidado no modo de fazer a ligação entre os dois sistemas sem mudar a experiência do utilizador na utilização do IDE.

Foram levantadas duas possibilidades. A primeira foi usar uma porta série virtual. Veio a tornar-se difícil pois seria necessário desenvolver um driver de porta série para cada sistema operativo onde o IDE possa ser executado. No Linux tentámos usar ferramentas que fazem emulação de porta série como o *socat* [3] mas a porta série criada não era reconhecida pelo IDE do Arduino. Usar uma porta série virtual também implicaria implementar o *bootloader* no Arduino virtual ou o protocolo de programação STK500 (Atmel, 2003) usado pelo *avrdude*, a ferramenta de programação dos Arduinos padrão. Outra desvantagem é que não seria possível carregar ficheiros ELF e portanto perderíamos a oportunidade de fazer o mapeamento das instruções com o código fonte. Também implicaria mais dificuldades para o utilizador na instalação e configuração da ferramenta de emulação da porta série.

A segunda possibilidade foi criar uma *board* para o IDE do Arduino. As *boards* são um sistema de *plugins* que permitem expandir a utilização do IDE do Arduino para outros tipos de microcontroladores (ver Figura 5.2). Por norma é necessário o *plugin* fornecer todas as ferramentas para compilar o código fonte Arduino e para programar o dispositivo em versões para Windows, Linux e Mac OS. Depois

[3] http://www.dest-unreach.org/socat/

de estudarmos a especificação dos *plugins* foi possível perceber que se pode estender uma arquitectura existente. Desse modo não é necessário fornecer ferramentas como por exemplo o compilador, podendo definir que a ferramenta de programação não é a padrão *avrdude* mas uma outra. Na Figura 5.3 podemos ver o ficheiro `boards.txt`, parte do plugin, em que temos assinalado que o `core` e a `variant` estendem o `arduino` e que a `upload.tool` é definida como `simupload`. A ligação entre `simupload` e a linha de comandos executada é definida no ficheiro `platform.txt` do *plugin*.

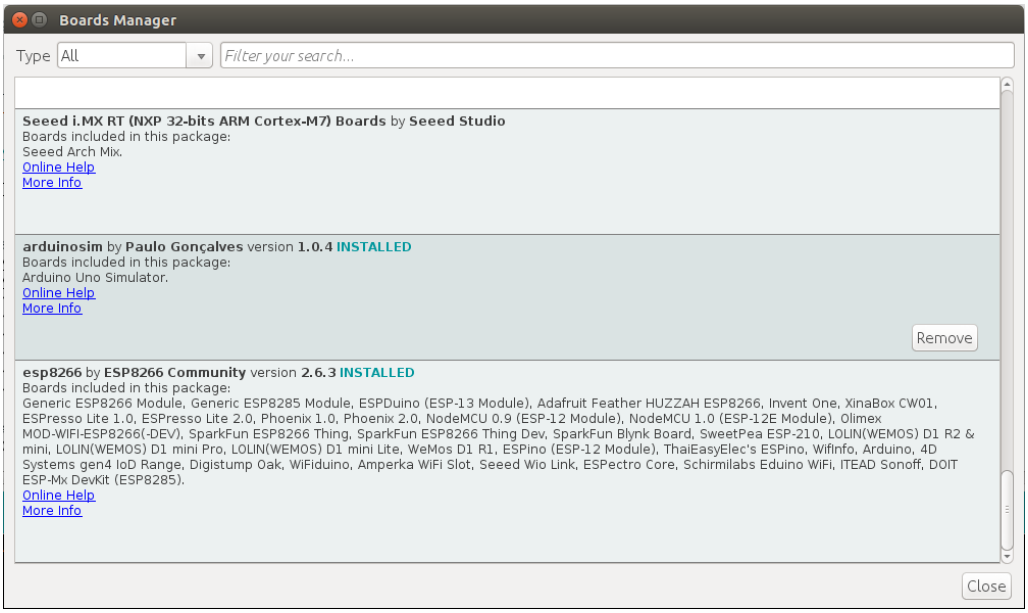


Figura 5.2: Gestão de *boards* no IDE do Arduino

```
unosim.name=Arduino Uno Simulator

unosim.vid.0=0x2341
unosim.pid.0=0x0043
unosim.vid.1=0x2341
unosim.pid.1=0x0001
unosim.vid.2=0x2A03
unosim.pid.2=0x0043
unosim.vid.3=0x2341
unosim.pid.3=0x0243

unosim.upload.tool=simupload
unosim.upload.protocol=arduino
unosim.upload.maximum_size=32256
unosim.upload.maximum_data_size=2048
unosim.upload.speed=115200

unosim.build.mcu=atmega328p
unosim.build.f_cpu=16000000L
unosim.build.board=AVR_UNO
unosim.build.core=arduino:arduino
unosim.build.variant=arduino:standard
```

Figura 5.3: Ficheiro boards.txt do plugin

Optámos por seguir a segunda hipótese no nosso projecto mas ainda há mais alguns desafios a considerar. Um é o tipo de binário que vamos usar para a ferra-

menta de programação: ter de fornecer um binário para cada arquitectura onde o IDE executa pode ser difícil de manter, mas o IDE é uma ferramenta Java e quando se instala traz uma JVM evitando o utilizador ter de instalar o Java à parte. Após alguns testes foi possível verificar que podemos usar a JVM instalada pelo IDE para executar as nossas ferramentas Java. Isso facilita porque só é necessário desenvolver e fornecer um único programa para todas as arquitecturas.

Um outro desafio é o mecanismo de comunicação entre a ferramenta de programação e o cliente, um navegador (*browser*) da Internet. É necessário fazer esta ligação pois temos de passar informação à ferramenta de programação de qual o endereço do servidor (porque não queremos deixar essa informação *hardcoded* no software para permitir múltiplos servidores) e também temos de passar o identificador da simulação para saber a que cliente corresponde o carregamento de um programa. Usar o endereço IP do cliente e da ferramenta de programação para relacionar os dois não funciona se existirem vários clientes por detrás de um router com *Network Address Translation* (NAT). Analisámos a possibilidade de usar WebRTC (WebRTC.org, 2020) para fazer essa comunicação, idealmente iniciando a ligação pela ferramenta de programação e não pelo cliente web pois a acção de programar o dispositivo é despoletada pelo IDE que executa a ferramenta. Tal não foi possível porque apesar do WebRTC permitir ligações ponto-a-ponto, precisa sempre de um servidor externo para mediar a ligação. Por existir a necessidade de um servidor externo ao sistema esta hipótese foi abandonada.

Chegámos à conclusão de que seria impossível ser o navegador da Internet a ter o lado servidor desta ligação e que a mesma teria de ser feita no sentido contrário, ou seja, navegador para ferramenta de programação. É simples ser o navegador a iniciar a ligação se for sobre HTTP, então implementámos um servidor web na ferramenta de programação que escuta numa porta fixa e conhecida no endereço *localhost*. Podemos usar este endereço porque o navegador da Internet e a ferramenta de programação (e o IDE do Arduino) executam sempre na mesma máquina. A desvantagem é que a ferramenta de programação está em execução apenas por breves instantes e só quando o utilizador usa o botão de programar o dispositivo no IDE. A ferramenta é executada, realiza a programação e depois termina. A aplicação cliente, a executar no navegador da Internet, não tem como saber em que momento é executada a ferramenta de programação pelo que tem de estar constantemente a tentar ligar à mesma com alguns segundos de intervalo entre tentativas. Também existe a possibilidade da porta TCP/IP que a ferramenta de programação usa para o servidor HTTP já estar a ser usada por outra aplicação. Para mitigar essa situação definimos não uma mas três portas fixas e o cliente tenta ligar a cada uma delas em sucessão até ter sucesso. Este processo de ligação não é instantâneo e pode levar a uma percepção de lentidão no envio do programa para o simulador. Para evitar este atraso a ferramenta de programação guarda, utilizando a API *Java Preferences*, os dados da última ligação feita com sucesso, e tenta usar esses mesmos dados para fazer nova programação do simulador. Em caso de insucesso, então inicia o servidor HTTP e aguarda que a aplicação web cliente lhe envie nova configuração com os dados do servidor web do simulador e o identificador da simulação onde deve ser carregado o programa do Arduino.

Este tipo de ferramentas são programas de linha de comandos e normalmente não têm interface gráfica nem pedem dados ao utilizador. Por norma todos os parâmetros são passados na linha de comandos. Neste caso a ferramenta precisa de conhecer

duas configurações que não são passadas por parâmetro e que lhe serão comunicadas pela interface HTTP. No caso de nenhum cliente se ligar a essa interface passado 10 segundos é mostrado ao utilizador uma janela onde ele pode colocar manualmente os dados necessário (ver Figura 5.4). Estes dados estão disponíveis na área de configuração do cliente web.

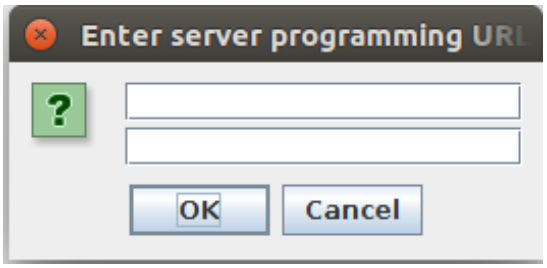


Figura 5.4: Pedido de dados pela ferramenta de programação

5.3 Aplicação Web

A aplicação web também foi implementada num projecto Maven e tem algumas dependências externas.

A primeira dependência é o projecto que faz a simulação do microcontrolador referido na Secção 5.1. Outra é código para representar as estruturas dos formatos ELF e DWARF (Eager, 2012) presente no GitHub (Kilic, 2016). Este código não foi incluído como uma dependência Maven, pois não foi publicado desse modo, e teve de ser copiado do GitHub directamente para a pasta do código fonte deste projecto. Este projecto implementa as estruturas dos tipos de ficheiro referidos e o seu carregamento a partir de um *buffer* de dados mas não tem a lógica de encadeamento das estruturas, essa parte foi programada por nós.

Para utilização do lado do cliente existe dependência das bibliotecas jQuery (The jQuery Team, 2020), Bootstrap (Bootstrap Team, 2020) e Draw2D (Herz, 2020) que por sua vez tem outras dependências. O jQuery é usado para a manipulação do *Document Object Model* (DOM) das páginas web. O Bootstrap é usado para dar estrutura às páginas web, criação de menus, barras de ferramentas, etc. A biblioteca Draw2D é uma ferramenta que permite criar diagramas em Javascript e é usada para criar o desenho do circuito no *browser* e incluí automaticamente várias funcionalidades como *zoom in* e *zoom out*, *drag* & *drop*, criação de ligações por *click* & *drag*, etc.

O projecto é uma *Java Web Application*, constituída por um conjunto de classes Java normais, *Servlets*, *Java Server Pages* (JSP)s e conteúdo estático como ficheiros HTML, *Cascading Style Sheets* (CSS), Javascript, imagens, etc. Existe também uma base de dados onde é mantida a informação sobre os utilizadores registados na aplicação e todos os dados relativos aos seus projectos. Cada projecto é um conjunto de um circuito desenhado pelo utilizador e do programa Arduino associado onde se incluí o código executável e os ficheiros de código fonte, no caso de terem sido carregados, e ainda todos os metadados como o nome dado ao projecto, o nome do projecto Arduino carregado, data de criação, etc. O esquema da base de dados é apresentado na Figura 5.5.

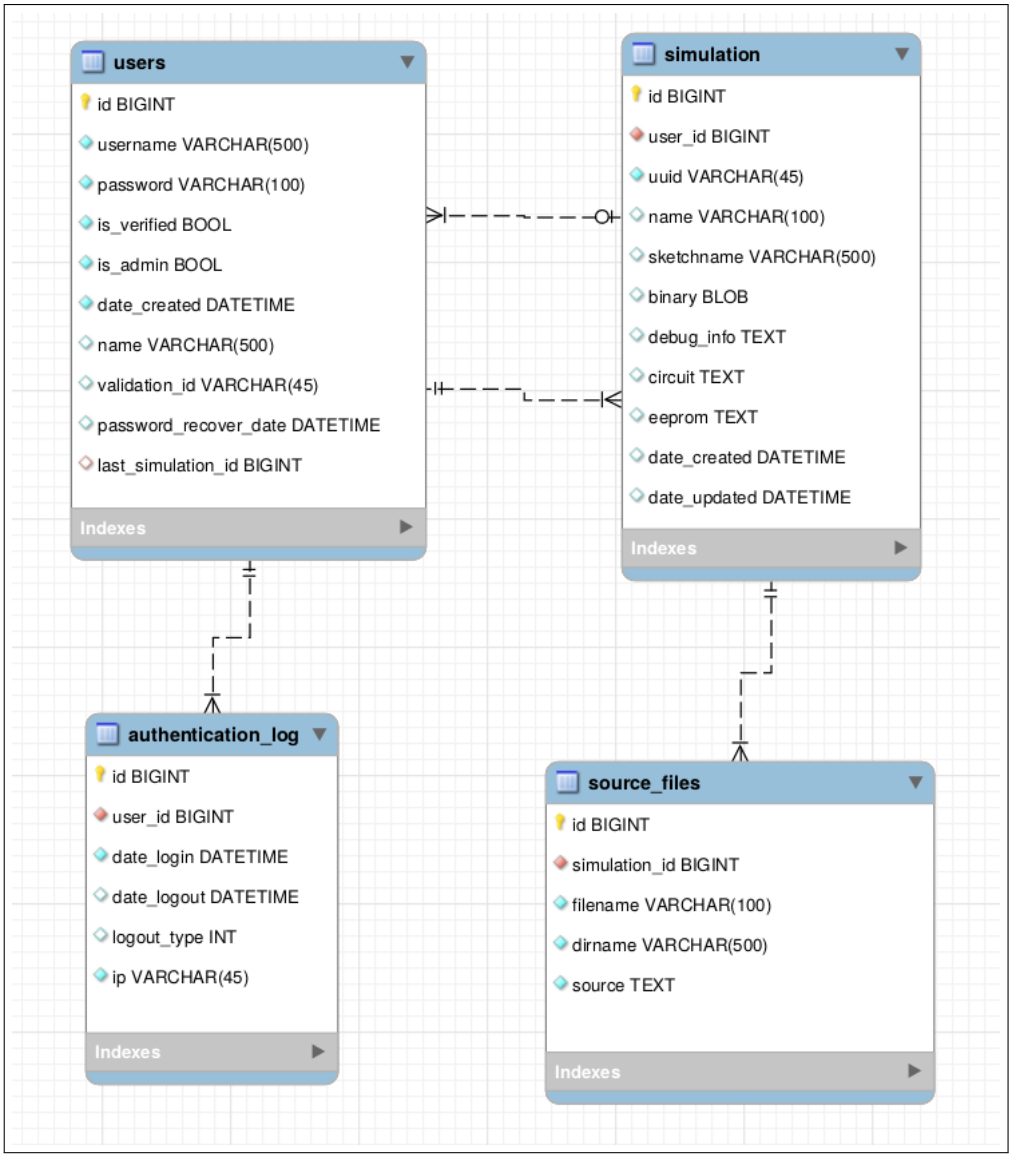


Figura 5.5: Esquema da base de dados

Para este projecto optámos por uma base de dados H2 (H2 Database, 2020) por permitir embeber o motor de base de dados na própria aplicação Java, não sendo adicionando mais um requisito externo à aplicação e evitando a necessidade de configuração adicional dos parâmetros de ligação à base de dados. H2 é um motor de base de dados programado em Java, muito rápido, *open source*, compatível com a API *Java Database Connectivity* (JDBC) e com modos de compatibilidade com muitos outros motores de base de dados como PostgreSQL, Oracle, MySQL, MS SQL Server, etc.

Alguma funcionalidade da aplicação é baseada em *Servlets* que permitem fazer o controlo de fluxo de mecanismos como o login, logout, registar utilizadores, verificar o email de registo, recuperar a password, etc. Para gerir o conteúdo apresentado aos utilizadores são usados ficheiros JSP como *templates*, localizados na directoria WEB-INF do projecto, onde estão protegidos de acesso directo por parte do cliente. As páginas em que só é permitido o acesso a utilizadores autenticados estão protegidas por um *Servlet Filter* que redirecciona o cliente para a página de login caso não

esteja devidamente autenticado.

Para a comunicação entre o cliente e o servidor na página principal do simulador são usadas *Websockets* (Mozilla Developer Network, 2019). Optou-se por esta tecnologia porque permite uma comunicação bidireccional de dados entre o cliente e o servidor sem haver a necessidade de recarregar a página. Em relação ao *Asynchronous JavaScript and XML* (AJAX) tem a vantagem de, depois de aberta a ligação, as mensagem poderem ser enviadas em qualquer sentido sem a necessidade do mecanismo de *request/response* permitindo ao servidor enviar dados sem que o cliente tenha de os pedir. Este mecanismo é de especial importância no cenário do simulador em que a simulação executa no servidor e é necessário, por exemplo, notificar o cliente de eventos como a mudança de estado de um pino de saída do microcontrolador.

De modo a manter a independência de todo o código do servidor do *Uniform Resource Locator* (URL) onde o serviço está a ser disponibilizado, ou seja, do domínio DNS, sempre que é necessário criar URLs absolutos (por exemplo nos emails enviados ou nos *links* para a ferramenta de programação ou para o *package* de instalação no IDE), estes são criados com base no URL que o cliente está a usar actualmente. Querendo descentralizar a instalação do servidor, podendo ter instalações locais a cada instituição de ensino, não ter de configurar o URL base no momento da instalação, é uma funcionalidade importante para diminuir a dificuldade da configuração e evitar erros na mesma.

A classe central no servidor web é a classe `Simulation`. É nesta classe que é mantido todo o estado de um determinado conjunto de microcontrolador/circuito. Ela cria um identificador único para a simulação, instância um objecto do tipo `ATmega328P`, adiciona *Listeners* para a porta série, para os pinos de GPIO e para as mudanças de estado do microcontrolador e tem ainda um conjunto de operações que é possível efectuar no microcontrolador como `start`, `stop`, `step`, `getDataMemory`, `getASM`, etc.

Uma das operações importantes é o carregamento de novos programas no microcontrolador virtual, na função `load` (ver Listagem 5.7). É nesta função que, com base na extensão do ficheiro carregado, é decidido que tipo de `ProgramMemoryLoader` será usado (`IntelHexProgramMemoryLoader` ou `BinaryProgramMemoryLoader`), e no caso de ser carregado um ficheiro do tipo ELF, com informação de *debug* no formato DWARF, é feita a extracção da informação que permite o mapeamento entre as linhas de código fonte e o respectivo endereço na memória FLASH. A informação recolhida também permite identificar os ficheiros de código fonte envolvidos no programa que são depois carregados para o servidor através da ferramenta de programação.

As classes `WebClientWSEndpoint` e `LoadWSEndpoint` implementam os *endpoints Websocket* respectivamente para o cliente web e para a ferramenta de programação. No caso da `WebClientWSEndpoint` existe a possibilidade de durante uma sessão HTTP ser necessário refazer a ligação. Isto pode ser necessário porque a ligação *Websocket* é mantida aberta durante toda a utilização da aplicação por parte do utilizador e pode ocorrer um erro temporário que obrigue a refazer a ligação. Como a associação de uma simulação a um utilizador da aplicação é realizada no inicio da ligação *Websocket* existe a necessidade de recuperar a simulação no caso de uma religação. O local habitual para guardar este tipo de informação é a sessão HTTP, mas

```
public List<DwLineNumberInformation.File> load(Load load) throws Exception
    {
    ...
    boolean wasRunning = cpu.isRunning();
    if( wasRunning ) {
        stop();
    }
    resetPreviousMemory();
    cpu.removeAllBreakpoints();
    if( load.getFileName().endsWith("ino.hex") ) {
        ByteBuffer buf = ByteBuffer.wrap(load.getContent());
        loadBinary = buf.duplicate();
        ProgramMemoryLoader.intelHexLoader().loadFlash(cpu.getFLASH(), buf);
    } else if( load.getFileName().endsWith(".elf") ) {
        ByteBuffer buf = ByteBuffer.wrap(load.getContent());
        buf.order(ByteOrder.LITTLE_ENDIAN);
        Elf32Context elf = new Elf32Context(buf);
        ByteBuffer txtSection = elf.getSectionBufferByName(".text");
        ByteBuffer dataSection = elf.getSectionBufferByName(".data");
        if( dataSection.remaining()>0 ) {
            ByteBuffer buffer = ByteBuffer.allocate(txtSection.remaining()
                + dataSection.remaining());
            buffer.order(txtSection.order());
            buffer.put(txtSection);
            buffer.put(dataSection);
            buffer.flip();
            txtSection = buffer;
        }
        loadBinary = txtSection.duplicate();
        ProgramMemoryLoader.binaryLoader().loadFlash(cpu.getFLASH(),
            txtSection);
        Dwarf32Context dwarf = new Dwarf32Context(elf);
        for( CompilationUnit cu : dwarf.getCompilationUnits() ) {
            Map<Integer,DwLineNumberInformation.Line> lines =
                cu.getDwLineNumberInformation().getLines();
            sourceFilesLines.addAll(lines.values());
            for(DwLineNumberInformation.Line line : lines.values()) {
                DwLineNumberInformation.File f = line.getFile();
                if( !sourceFiles.contains(f) &&
                        !f.getDirectory().startsWith(".") ) {
                    sourceFiles.add(f);
                    System.out.println(sourceFiles);
                }
            }
            sourceFilesCopy.addAll(sourceFiles);
        }
        sendSourceLines();
    } else {
        throw new Exception("invalid file type");
    }
    ...
}
```

Listagem 5.7: Extracto do método load da classe Simulation

como a API *Websocket* não tem acesso a ela, foi necessário recorrer a um mecanismo que nos desse esse acesso. A solução passou por criar uma classe que intercepta a fase de negociação da *Websocket* (classe `GetHttpSessionConfigurator`) num local onde é possível aceder à sessão HTTP e desse modo injectá-la nas configurações da *Websocket* que já é acessível na API. Podemos ver na Listagem 5.8 um excerto desse código.

```
public class GetHttpSessionConfigurator extends
        ServerEndpointConfig.Configurator {
    @Override
    public void modifyHandshake(ServerEndpointConfig config,
                                HandshakeRequest request,
                                HandshakeResponse response) {
        HttpSession httpSession = (HttpSession)request.getHttpSession();
        config.getUserProperties().put(HttpSession.class.getName(),httpSession);
    }
}
...
@ServerEndpoint(value = "/ws/client",
                configurator = GetHttpSessionConfigurator.class)
public class WebClientWSEndpoint {
    ...
    @OnOpen
    public void onOpen(Session session, EndpointConfig config)
            throws IOException {
        HttpSession httpSession = (HttpSession) config.getUserProperties()
                .get(HttpSession.class.getName());
        ...
    }
    ...
}
```

Listagem 5.8: Injecção da sessão HTTP na *Websocket*

É na sessão HTTP que fica guardada a associação entre o identificador da simulação, gerado na classe `Simulation`, e o objecto que foi instanciado.

No caso do *endpoint* `LoadWSEndpoint`, uma vez que as transacções são de curta duração não existe a necessidade de religações e o identificador da simulação é transmitido junto com os dados transferidos. Em caso de falha da ligação o procedimento de programação do simulador deverá ser repetido.

A comunicação dentro das *Websockets* é feita com mensagens no formato *JavaScript Object Notation* (JSON). O objecto JSON enviado tem sempre um atributo `cmd` que indica o tipo de mensagem. Cada tipo de mensagem pode conter outros atributos ou parâmetros. As mensagens enviadas são sempre considerados eventos, ou seja, quem envia um pedido não fica bloqueado à espera de resposta. Isto é de especial importância para não haver um bloqueio na interface com o utilizador e tem ainda a vantagem de a mesma simulação poder eventualmente ser apresentada em mais do que um cliente. Neste trabalho não é explorada essa situação mas pode-se vir a tirar partido dessa funcionalidade por exemplo para que um docente possa

acompanhar a simulação que um aluno está a fazer.

A classe `WebClientWSEndpoint` recebe as seguintes mensagens:

- **ping**: Mantém um contacto permanente entre o servidor e o cliente. Se o servidor não receber uma mensagem do cliente num determinado período de tempo pára automaticamente a execução da simulação.
- **resume**: Continua a execução de uma simulação que foi previamente parada.
- **start**: Se a simulação estava em execução pára-a. De seguida faz o *reset* ao microcontrolador e inicia a simulação.
- **stop**: Pára a execução da simulação.
- **status**: Devolve informação sobre o estado da simulação, como por exemplo se está a executar ou não ou o endereço do *program counter*.
- **speed**: Devolve a velocidade média, em MHz, a que o microcontrolador virtual está a executar.
- **step**: Executa uma instrução no microcontrolador virtual.
- **asm**: Devolve o código assembly carregado na memória FLASH do microcontrolador.
- **dataMemory**: Devolve o conteúdo da memória SRAM do microcontrolador.
- **breakpoint**: Adiciona ou remove um *breakpoint* no código.
- **usart**: Envia um carácter para a porta série do microcontrolador.
- **pinChange**: Envia uma mudança de estado num pino do microcontrolador.
- **load**: Se a simulação estava em execução pára-a. De seguida carrega no microcontrolador um novo ficheiro de código e inicia a simulação.
- **save**: Salva o projecto de simulação actual na base de dados incluindo o código e o circuito.
- **open**: Carrega um projecto de simulação guardado anteriormente.
- **delete**: Apaga o projecto de simulação actualmente carregado e cria um novo vazio.
- **new**: Cria um novo projecto de simulação vazio.

A classe `WebClientWSEndpoint` envia as seguintes mensagens:

- **exception**: Notificação no caso da ocorrência de um erro na execução da simulação.
- **pinChange**: Notificação para uma mudança de estado de um pino do microcontrolador.
- **statusChange**: Notificação quando o estado da execução da simulação muda, por exemplo quando pára.
- **speed**: Resposta com a velocidade média, em MHz, a que o microcontrolador virtual está a executar.
- **load**: Notificação quando é carregado um novo projecto.
- **asm**: Envia código assembly carregado na memória FLASH do microcontrolador para o cliente.
- **dataMemory**: Notificação com o conteúdo da memória SRAM do microcontrolador para o cliente.
- **breakpoints**: Notificação da adição ou remoção de *breakpoints*.
- **step**: Notificação após a execução de um passo com o novo endereço do *program counter*.
- **usart**: Notificação com dados da porta série para o cliente.
- **sourceFile**: Notificação com o código fonte do programa a ser simulado.

- **sourceLines**: Notificação com o mapeamento entre as linhas do código fonte e o respectivo endereço na memória de programa (FLASH).
- **loadCircuit**: Notificação para carregar um circuito na área de desenho do cliente.
- **simulationsList**: Notificação com a lista de simulações guardadas no servidor.

A classe `LoadWSEndpoint` recebe as seguintes mensagens:

- **ping**: Mensagem usada para verificar se o identificador usado no último contacto ainda se encontra válido. A ferramenta de programação só executa o servidor web para receber novos dados se o identificador anterior já não for válido.
- **load**: Se a simulação estava em execução pára-a. De seguida carrega no microcontrolador um novo ficheiro de código e inicia a simulação.
- **readFile**: Resposta ao pedido de carregamento de um ficheiro de código fonte.

A classe `LoadWSEndpoint` envia as seguintes mensagens:

- **ping**: Resposta com a validade de um identificador de simulação.
- **readFile**: Pedido para que seja enviado um ficheiro de código fonte.
- **response**: Resposta de confirmação de recepção das restantes mensagens.
- **close**: Pedido para fecho da ligação.

Usámos a classe `SimulationStorage` para abstrair o modo como são salvos os dados. Esta classe funciona como um *Data Mapper*. Usámos este padrão porque a base de dados é de baixa complexidade e não existe a perspectiva que seja alterada muitas vezes e permite mudar facilmente o tipo de armazenamento dos dados sem fazer grandes alterações no restante código.

Existem acções que devem ser executadas quando o servidor web é iniciado e quando é parado. Para executar essas acções criámos um *ServletContextListener*, um objecto que está associado ao ciclo de vida das *Servlets* e que permite executar código quando a aplicação web é iniciada e quando é parada. Implementámos essa lógica na classe `ServletListener`, que no arranque da aplicação executa duas tarefas: criar um *ExecutorService* (uma *pool* de *Threads*), para executar as simulações e criar a base de dados inicial caso não exista. Ao encerrar a aplicação web é destruído o *ExecutorService* e é encerrada a base de dados (é necessário este passo porque estamos a usar uma base de dados embebida).

A utilização de um *ExecutorService* para executar as simulações justifica-se com a necessidade de controlar o número de simulações a executar em simultâneo na mesma máquina. O *ExecutorService* é iniciado com um número máximo de *Threads*, neste caso um valor dependente do número de processadores da máquina física onde está a ser executado o simulador (num_processadores-1) ou o valor fixo 4, dependendo do que for maior. No caso de não haver *Threads* disponíveis para executar a simulação é apresentado um aviso ao utilizador e a simulação não é executada. Deste modo evitamos esgotar os recursos da máquina tornando as simulações demasiado lentas para todos os utilizadores.

Para a instalação da *board* correspondente ao simulador no IDE do Arduino é necessário criar um ficheiro com um nome e um formato especifico (Arduino SA, 2020) e disponibilizá-lo num servidor web. Este URL foi implementado como um

JSP, e não como um ficheiro estático, para permitir criar os URLs dinamicamente dependendo do nome do servidor a que o cliente acede assim como toda a outra informação presente neste ficheiro e que pode ser calculada automaticamente. Essas variáveis incluem o *Secure Hash Algorithm* (SHA) dos ficheiros, o seu tamanho e a versão. Os ficheiros da ferramenta de programação, que têm de ser disponibilizadas ao IDE no formato *tar.gz* (um tipo de ficheiro comprimido), são gerados automaticamente na fase de compilação do projecto através de um plugin Maven. Ao compilar o projecto `ArduinoSimulatorWeb`, na fase `prepare-package`, é copiado do repositório local Maven a ferramenta de programação, é criado o ficheiro *tar.gz* para cada arquitectura necessária no ficheiro *json* e é colocado na directoria correcta para futuro download. Isto torna extremamente fácil a compilação do projecto evitando a cópia manual de ficheiros existindo somente uma dependência Maven e a configuração da versão da ferramenta de programação (projecto `ArduinoSimulatorProgrammer`) que se pretende.

5.3.1 Cliente Web

O cliente web foi implementado com HTML, CSS e Javascript e *Websockets*.

Depois do carregamento inicial da página que apresenta as barras de ferramentas e a área de desenho do circuito, toda a comunicação com o servidor é feita a través de *Websockets*. Isto permite termos uma interface com um aspecto de aplicação nativa e sem qualquer recarregamento de páginas durante a utilização do simulador. Todas as acções que afectam a simulação, como iniciar ou parar a simulação, são executadas de modo assíncrono. Quer dizer que a acção sobre o botão carregado apenas envia uma mensagem para o servidor que actua de acordo com a mensagem e notifica mais tarde o cliente com a respectiva mudança de estado. Pelo contrário acções como *zoom in* ou *zoom out* ou a abertura de janelas de inspecção actuam localmente ao cliente.

A biblioteca draw2d ajuda na parte de desenho nomeadamente com as funcionalidades de *drag and drop*, *zoom*, *undo* e *redo* e ligações entre objectos. Os componentes que são acrescentados à área de desenho são todos extensões do objecto draw2d `SetFigure`, incluindo o próprio Arduino Uno. Definimos um objecto `componentProps` com a funcionalidade base comum a todos os componentes a partir do qual, estendendo `SetFigure` se gera o objecto `arduino` (ver Listagem 5.9).

```
var arduino = {};

var componentProps = {
    NAME: "arduino.Component",
    ...
};

arduino.Component = draw2d.SetFigure.extend(componentProps);
```

Listagem 5.9: Classe Arduino

Os componentes são representados no DOM do HTML como objectos *Scalable Vector Graphics* (SVG). A representação visual destes objectos pode ser criada com

caminhos SVG, primitivas 2D ou imagens. Nos componentes criados neste projecto é tirado partido de todas estas técnicas para definir a apresentação gráfica dos componentes.

Implementámos a parte de simulação da lógica dos circuitos electrónicos estendendo os objectos e tirando partido da noção de *ligação* entre os mesmos existente na biblioteca. Existem as *porta*s que são um local no componente onde se pode fazer uma ligação com outra porta de outro componente. Estas portas também têm um valor associado que pode ser atribuído com o método `setValue` e lido com o método `getValue`. Mais importante é que a atribuição de um valor numa porta gera o evento `onPortValueChanged`. Este principio é usado para propagar a electricidade virtual pelo circuito como veremos mais à frente.

Na Listagem 5.10 é possível verificar como uma mudança de valor de uma porta do componente `LED` faz mudar a imagem representativa do objecto no circuito através da alteração do *layer* que é apresentado, sendo que o *layer* `low` apresenta o `LED` em tons de cinza e o *layer* `high` apresenta o `LED` na respectiva cor. Neste caso o `LED` acende quando o cátodo tem o valor 0 (zero) e o ânodo tem um valor positivo.

```
onPortValueChanged: function (relatedPort) {
    if (this.getPort("anode").getValue() > 0 &&
        this.getPort("cathode").getValue() === 0) {
        this.layerShow("low", false);
        this.layerShow("high", true);
    } else {
        this.layerShow("low", true);
        this.layerShow("high", false);
    }
}
```

Listagem 5.10: Evento de mudança de valor de uma porta de LED

Na Listagem 5.11 podemos ver como uma porta é criada no método `init` do objecto Javascript. Neste caso a porta é criada com uma posição x e y relativa às dimensões do objecto, aceita ligações da direcção 3 que significa que os *fios* saem para a esquerda do objecto, tem uma determinada cor e um nome e o *MaxFanOut* indica que a porta só aceita uma ligação a outra porta.

```
var port = this.createPort("hybrid", new
    draw2d.layout.locator.XYRelPortLocator(-2, 57));
port.setConnectionDirection(3);
port.setBackgroundColor("#37B1DE");
port.setName("cathode"); //o polo negativo
port.setMaxFanOut(1);
```

Listagem 5.11: Criação de uma porta num componente

O método `propagateOnConnect` (ver Listagem 5.12) é usado para calcular a direcção da propagação do sinal entre duas portas. Por exemplo, quando se fecha

um botão, é necessário saber se se deve copiar o valor que está na ligação à esquerda para a ligação que está à direita ou fazer o inverso. Isto porque um botão não tem uma entrada e uma saída definida mas tem simplesmente duas ligações ou dois terminais. Para decidir que sinal propagar para onde são analisados os valores presentes nas duas portas. Se não existir sinal em nenhuma das duas (representado pelo valor `null`) não há sinal a propagar. No caso de haver sinal em ambas as portas não se faz propagação, pois ou têm o mesmo valor e não é necessário alterar o valor que já existe ou têm valores diferentes e é considerado um curto-circuito. O último caso é quando existe um sinal numa porta e não na outra. Nesse caso a que tem sinal passa a ser considerada a origem e a outra o destino para onde propagar o sinal. Colocamos então o valor na porta de destino e chamamos o método `propagate` para propagar o sinal para as outras portas ligadas a ela. Este método também devolve aquela que foi considerada a porta de saída ou destino para utilização na restante lógica da aplicação.

```
propagateOnConnect: function (p1, p2) {
    var inPort = null;
    var outPort = null;

    var v1 = p1.getValue();
    var v2 = p2.getValue();

    if (v1 === null && v2 === null) {
        return null;
    }

    if (v1 !== null && v2 !== null) {
        if (v1 !== v2) {
            console.log('curto circuito!!!!!');
        }
        return null;
    }

    if (v1 !== null) {
        inPort = p1;
        outPort = p2;
    } else {
        inPort = p2;
        outPort = p1;
    }

    outPort.setValue(inPort.getValue());
    this.propagate(outPort);

    return outPort;
}
```

Listagem 5.12: Método propagateOnConnect da classe componentProps

Na Listagem 5.13 é possível ver o código do método `propagate` que muda o sinal presente nas portas do outro lado da ligação à qual a porta actual está conectada.

Este método também muda a cor da ligação para dar uma indicação visual ao utilizador do tipo de sinal presente na mesma. Fica vermelha se tiver um sinal positivo, preta se tiver 0 volts e azul se não tiver qualquer tensão ou sinal.

```
propagate: function (port, currentCon) {
    var cons = port.getConnections();
    var _this = this;

    cons.each(function (i, con) {
        if (con === currentCon) {
            return;
        }
        if (port.getValue() === null) {
            con.setColor("#129CE4"); //not connected (blue line)
        } else if (port.getValue() === 0) {
            con.setColor("#000000"); //ground (black line)
        } else {
            con.setColor("#ff0000"); //positive (red line)
        }
        var other = con.getTarget();
        if (other === port) {
            other = con.getSource();
        }
        other.setValue(port.getValue());

        //this allows to propagate to ports with multiple connections
        other.getConnections().each(function (i, subcon) {
            if (subcon !== con) {
                _this.propagate(other, con);
            }
        });
    });
}
```

Listagem 5.13: Método propagate da classe componentProps

5.3.2 Descrição da interface de utilizador

O cliente web é onde é feita toda a interacção com o utilizador. Na página inicial do portal, apresentada na Figura 5.6, é possível copiar o URL para configurar a placa do simulador no IDE do Arduino, registar-se, fazer login na aplicação e consultar o manual de utilização.

Para um utilizador usar o simulador, o primeiro passo a fazer é registar-se no portal web. Para isso deve indicar um *username* no formato de email, uma password e um nome. Após confirmar estes dados é-lhe enviado um email para confirmar a sua conta. Só depois de clicar num link com um identificador único, enviado no email, é que passa a ser possível efectuar login. No caso do utilizador não se recordar da password pode pedir a recuperação da mesma, inserindo o email com que se registou

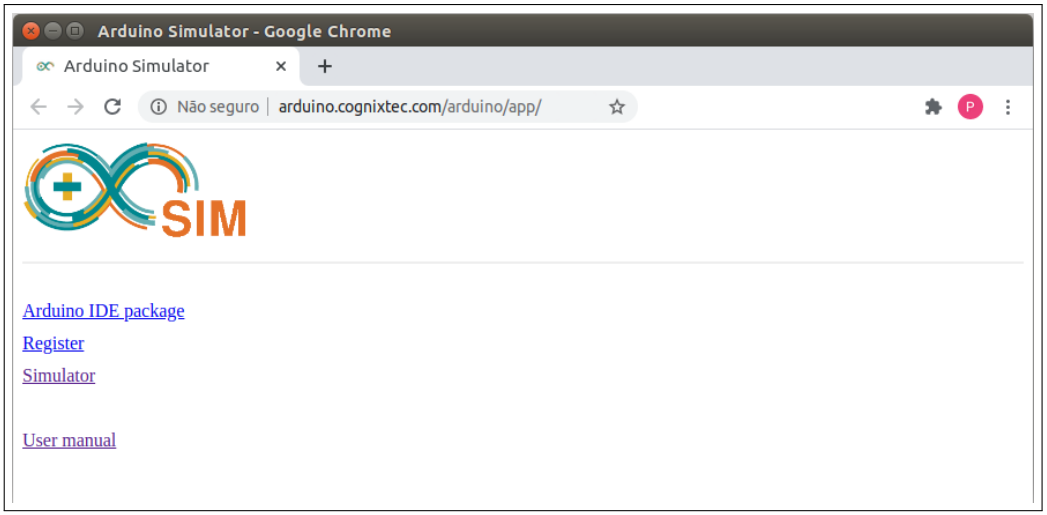


Figura 5.6: Página principal do portal do simulador

numa página, após o qual lhe é enviado um email com um link em que deve clicar levando a uma página onde pode escolher uma nova password.

A página principal da aplicação, acessível apenas a utilizadores autenticados, pode ser vista na Figura 5.7.

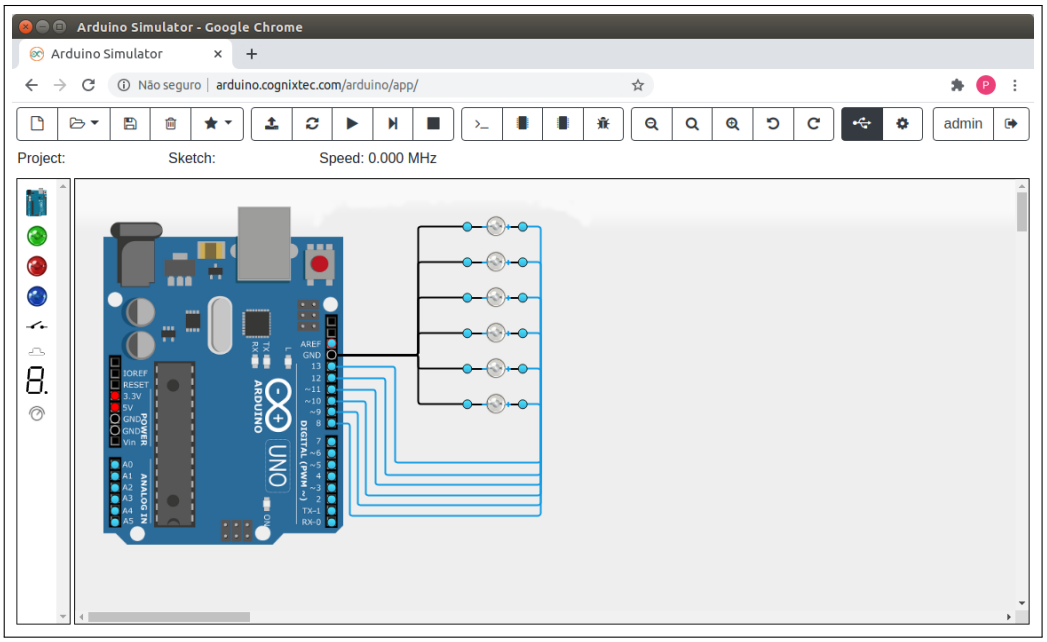


Figura 5.7: Interface principal do simulador

A barra de ferramentas na parte de cima da interface do utilizador está dividida em 6 blocos. O primeiro, na Figura 5.8, tem botões para gestão do projecto:

- Criar um novo projecto;
- Para abrir projectos previamente guardados;
- Gravar o projecto actual, pedindo um nome caso ainda não tenha;
- Apagar o projecto actual;
- Um menu com circuitos pré-definidos que podem ser carregados para a área de desenho.

Figura 5.8: Gestão do projecto

O segundo bloco, na Figura 5.9, tem botões para controlo do microcontrolador:

- Carregar um programa para Arduino (executável) directamente na interface web;
- Fazer *reset* ao microcontrolador;
- Fazer *pause* e *resume* ao microcontrolador;
- Avançar um passo na execução do microcontrolador;
- Parar a execução.

Figura 5.9: Controlo do microcontrolador

O terceiro bloco, na Figura 5.10, tem botões para abrir janelas de inspecção:

- Abrir a janela do monitor série;
- Abrir a janela do inspector da FLASH;
- Abrir a janela do inspector da SRAM;
- Abrir a janela do inspector de código fonte.

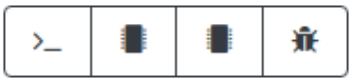


Figura 5.10: Janelas de inspecção

O monitor série (ver Figura 5.11) é uma janela onde é apresentado em *American Standard Code for Information Interchange* (ASCII) o valor enviado pelo microcontrolador para a porta série. Os valores escritos na caixa de texto no fundo da janela são enviados para o microcontrolador após ser pressionada a tela *Enter* do teclado.

Na janela de inspecção da FLASH (ver Figura 5.12) é apresentada a decompilação do código carregado no microcontrolador. À frente de algumas instruções é apresentado um comentário com a descrição da instrução ou o endereço de destino dos saltos. As instruções que fazem referência a registos do processador apresentam o nome do registo e não o seu valor numérico. Esta informação serve para ajudar o utilizador a melhor interpretar as instruções. A linha indicada a verde tracejado indica a última instrução executada e é útil para acompanhar a execução passo a passo. Ao clicar num endereço, assinalado a cor de rosa, é possível adicionar ou remover um *breakpoint*. Quando está instalado um *breakpoint* o endereço fica assinalado com o fundo laranja.

Na janela de inspecção da SRAM (ver Figura 5.13) é apresentada a memória do microcontrolador. Na zona onde estão mapeados os registos de uso geral é indicado o nome do registo, a restante memória é apresentada com o seu endereço. Os registos

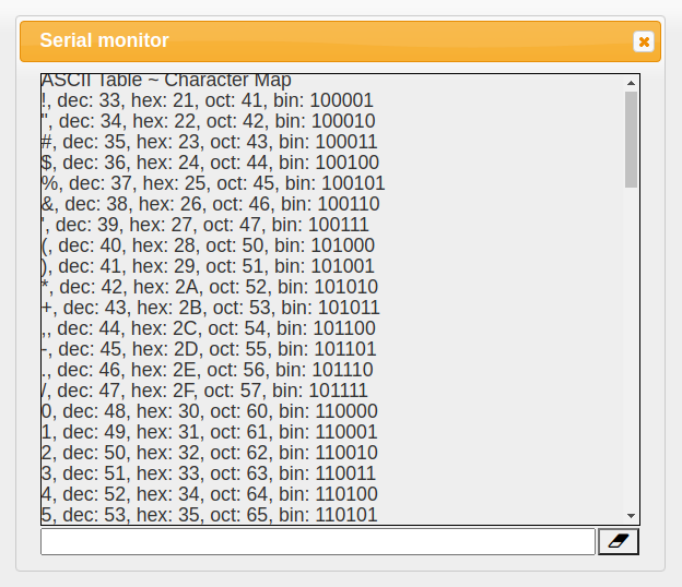


Figura 5.11: Monitor série

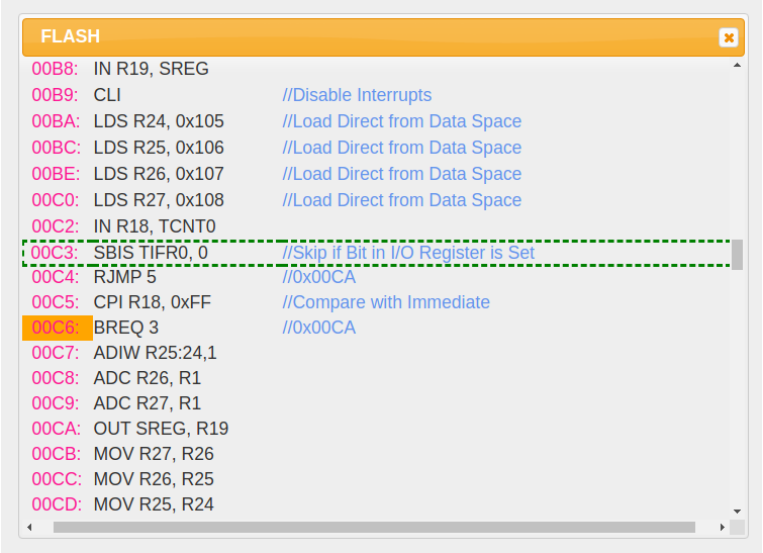


Figura 5.12: Janela de inspecção da FLASH

de endereçamento X, Y e Z de 32 bits (coincidentes com os registos R26 a R31) também estão assinalados. Quando existe uma mudança no valor de um registo esse fica assinalado durante cerca de 1 segundo com o fundo laranja. Isto ajuda o utilizador a perceber quais os registos que estão a mudar de valor.

Na janela de inspecção de código fonte apresentada na Figura 5.14 é possível ver o código fonte do programa carregado no microcontrolador (se foi carregado um ficheiro ELF através da ferramenta de programação). Esta janela apresenta os vários ficheiros que compõem o projecto Arduino e em cada ficheiro as linhas apresentadas com o fundo laranja claro têm um mapeamento directo com uma instrução num endereço da FLASH. Este mapeamento é feito pela informação de *debug* presente no ficheiro ELF. Nessas linhas é possível clicar para activar ou remover um *breakpoint*. Quando a linha fica com o fundo a laranja mais escuro indica que existe um *breakpoint* nessa linha.

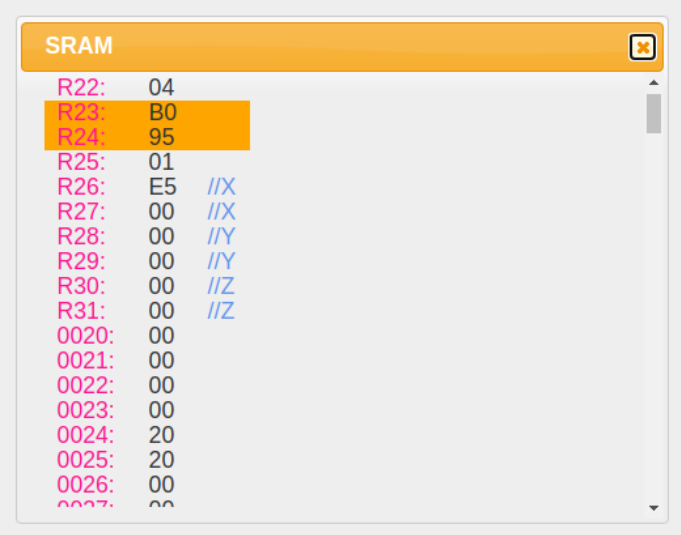


Figura 5.13: Janela de inspecção da SRAM

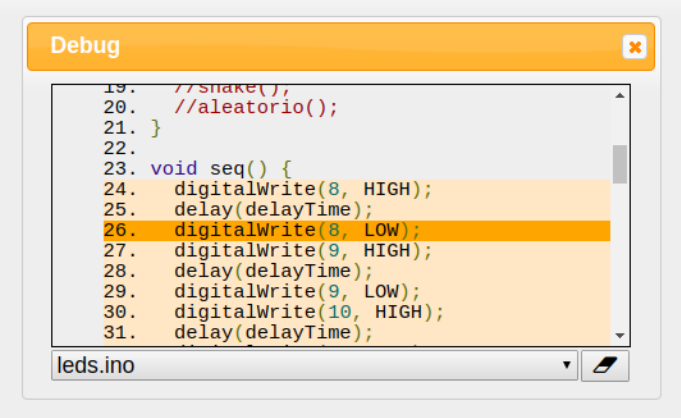


Figura 5.14: Janela de inspecção de código fonte

O quarto bloco da barra de ferramentas, na Figura 5.15, tem botões de controlo da área de desenho:

- Fazer *zoom out* à área de desenho;
- Repor o *zoom* da área de desenho;
- Fazer *zoom in* à área de desenho;
- Desfazer a última acção de alteração ao circuito;
- Refazer a última acção de alteração ao circuito.

Figura 5.15: Controlos de desenho

O quinto bloco, apresentado na Figura 5.16, tem botões para configuração:

- Activar ou desactivar a ligação à ferramenta de programação;
- Abrir janela com as configurações da simulação actual.

Figura 5.16: Controlos de configuração

A janela com as configurações da simulação actual (ver Figura 5.17) apresenta o URL para configurar a placa no IDE do Arduino além dos dados necessários para a ferramenta de programação se os quisermos fornecer manualmente.

Figura 5.17: Janela com dados de configuração

O sexto bloco da barra de ferramentas, na Figura 5.18, tem:

- O login do utilizador actual;
- Um botão para fazer logout.

Figura 5.18: Dados do utilizador e botão de logout

Logo abaixo da barra de ferramentas existe uma área onde é apresentada informação sobre o projecto actual. Podemos ver na Figura 5.19 que é indicado o nome do projecto actual, o nome do *sketch* (nome dado aos projectos no IDE do Arduino) carregado e a velocidade média de execução do simulador.

Figura 5.19: Informação do projecto actual

No lado esquerdo da interface existe uma palete de componentes que podem ser adicionados ao circuito arrastando o componente para a àrea de desenho.

O primeiro componente é um Arduino Uno. Só é permitido adicionar um Arduino ao circuito pois a aplicação só está preparada para simular um Arduino por sessão.

De seguida temos LEDs com 3 cores diferentes, verde, vermelho e azul. Na Figura 5.20 podemos ver que os LEDs podem apresentar dois estados: ligado e desligado. Na simulação do circuito assume-se que não existe a necessidade de ligar uma resistência para o LED funcionar correctamente.

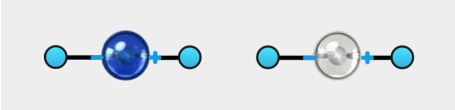


Figura 5.20: Componente LED

O próximo componente é um interruptor que pode ser ligado ou desligado com um clique do rato. Na Figura 5.21 podemos ver os dois estados possíveis para o interruptor.

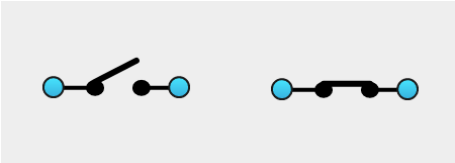


Figura 5.21: Componente interruptor

Existe também um interruptor de pressão que fica ligado quando pressionado com o rato e desliga quando é largado. Na Figura 5.22 podemos ver os dois estados possíveis para o interruptor de pressão.

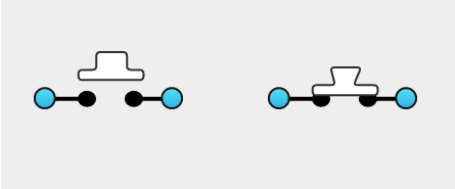


Figura 5.22: Componente interruptor de pressão

A seguir temos um *display* de 7 segmentos. É um componente destinado a apresentar um digito de um número árabe embora também seja possível representar outros símbolos sendo comum a utilização das letras A a F para ser possível representar um valor hexadecimal. Tem também um ponto decimal. Na Figura 5.23 podemos ver a atribuição dos segmentos, sendo que o segmento `a` está ligado. Cada segmento é um LED sendo que neste caso todos os LEDs que compõem o *display* têm o cátodo ligado em comum.

O último componente na palete é um potenciómetro. Este componente permite-nos testar o ADC do microcontrolador. Na Figura 5.24 podemos ver um exemplo típico de utilização. Os terminais nos extremos do potenciómetro são ligados a duas tensões diferentes, neste caso 0 volts e 5 volts, e a tensão de saída será uma tensão entre as duas da entrada, sendo que esse valor será mais próximo de uma ou outra dependendo da posição do potenciómetro (regulado com o botão verde que pode ser arrastado para um lado ou o outro com o rato). A saída deve ser ligada a uma entrada analógica do microcontrolador, neste caso a `A0` e o valor pode ser lido com a função da API do Arduino `analogRead`.

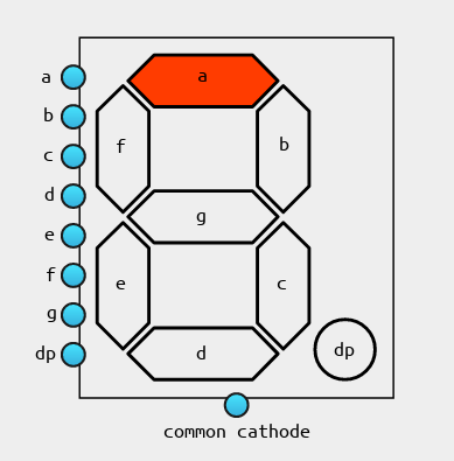


Figura 5.23: Componente *display* de 7 segmentos

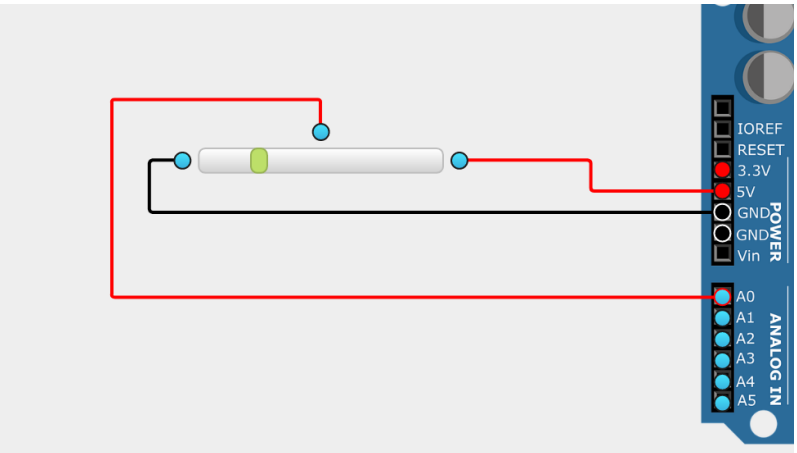


Figura 5.24: Componente potenciómetro

5.4 Características implementadas

Nesta secção vamos fazer um resumo das principais características que foram implementadas no simulador. Devemos notar que algumas são características implementadas pelo servidor e outras pelo cliente.

5.4.1 Componentes do circuito

Existem alguns componentes básicos disponíveis para o utilizador usar na construção do circuito electrónico.

- **LEDs**: Existem LEDs de três cores diferentes, verde, vermelho e azul. Quando estão desligados mantêm-se cinzentos, quando estão ligados mudam para a sua respectiva cor. Os LEDs simulados têm polaridade tal como os verdadeiros. Considera-se também que têm a adequada resistência interna pelo que não é necessário adicionar esse componente ao circuito.

- **Botão**: É um normal botão que o utilizador pode clicar com o rato para abrir ou fechar o circuito.

- **Botão de pressão**: É um botão que o utilizador pode pressionar com o rato para fechar o circuito mas que abre automaticamente quando o utilizador larga o rato.

- **Display de 7 Segmentos**: É um componente composto por LEDs dispostos numa forma que permite que os mesmos representem os números Árabes de 0 a 9. Tem 8 entradas (7 para os números mais uma para o ponto decimal) e um cátodo comum.

- **Potenciómetro**: Este componente tem três terminais, duas entradas e uma saída. A saída é a diferença relativa entre as entradas. Pode ser usado como entrada analógica para os pinos analógicos da placa Arduino.

5.4.2 *Debugging*

O simulador consegue introduzir algumas funcionalidades de *debug* não existentes na placa Arduino real. Uma destas funcionalidades é a possibilidade de pausar ou resumir a execução do microcontrolador a qualquer momento carregando num botão. O efeito prático é como se conseguíssemos parar o sinal de relógio no dispositivo real mantendo todo o estado interno do dispositivo. Isto permite-nos inspeccionar a memória SRAM para verificar que valores contém e também olhar para a memória FLASH, com as instruções descodificadas, e saber qual foi a última executada.

O simulador também permite aos utilizadores ver que linhas de código fonte estão relacionadas com que instruções. Isto é feito seleccionado um endereço na janela da memória FLASH ou um número de linha de código na janela de código fonte (quando for carregado um ficheiro no formato ELF, que contém informação sobre os ficheiros de código fonte), e inserir *breakpoints*, que quando são executados pausam automaticamente a execução da simulação.

Quando a simulação está pausada, é possível executar uma instrução de cada vez e acompanhar as alterações de valores na SRAM na janela de inspecção da SRAM e acompanhar as instruções que estão a ser executadas nas janelas de inspecção de FLASH e de código fonte.

5.4.3 Outras características

Outras características de notar são:

- A capacidade de carregar binários nos formatos Intel HEX e ELF, sendo que com o formato ELF é tirado partido da informação presente no formato para ajudar no *debug*.

- A capacidade de salvar projectos onde fica guardado o circuito criado e também os executáveis carregados.

- A perfeita integração com o IDE do Arduino permitindo programar o simulador como se tratasse de uma placa real.

- A possibilidade de carregar ficheiros binários directamente na interface do cliente web para o caso dos utilizadores querem usar outro compilador não integrado no IDE do Arduino.

- O *serial port monitor* é onde é possível visualizar os dados enviados pelo programa a correr no simulador para o dispositivo USART0 e também enviar dados que o programa pode tratar. Tem uma utilização similar ao *Serial Port Monitor* do IDE do Arduino.
- A existência de janelas de inspecção para analisar o conteúdo da FLASH e da SRAM.
- Uma aplicação cliente web evitando a necessidade de instalação e configuração de software nos clientes.
- A simulação do circuito electrónico no cliente diminuindo a necessidade de comunicação entre cliente e servidor e facilitando o desenvolvimento de novos componentes electrónicos.
- Um servidor desenvolvido em Java permitindo a sua execução em qualquer arquitectura.
- O desenvolvimento de todo o projecto em código fonte aberto, disponibilizado livremente, possibilitando a continuação do projecto com a colaboração da comunidade.

Capítulo 6

Verificação

Segundo Ryan e Wheatcraft, 2017, uma das definições de verificação indica que se trata de "um conjunto de actividades que compara um sistema ou elemento do sistema com as características necessárias. Isso pode incluir, mas não está limitado a, especificação de requisitos, descrição do projecto e o próprio sistema. A verificação garante que se construiu o sistema da maneira certa".

Para fazer a verificação do sistema, ou seja, que se construiu um sistema segundo as especificações, fizemos testes unitários e testes funcionais ao sistema.

Os testes feitos não foram exaustivos ao nível que poderiam ter sido mas há que perceber que o projecto foi realizado por uma única pessoa, não havendo tempo para ser muito minucioso neste aspecto, além de que não se consegue a independência entre o programador e o *tester* que seria necessária num ambiente de testes formal. Sendo um projecto que vai ser de código fonte aberto, prevemos não só a contribuição na evolução funcional por parte da comunidade mas também testes e correcção de *bugs*, sendo que é conhecido o beneficio do código fonte aberto para a sua correcção: *"Given enough eyeballs, all bugs are shallow"* (Raymond, 1999).

6.1 Testes unitários

A realização de testes unitários cingiu-se ao projecto `ArduinoSimulator` e às classes que implementam as instruções do ISA AVR, nomeadamente ao método `execute` da instrução. Esta pareceu-nos a área onde os testes unitários seriam mais úteis uma vez que a especificação da funcionalidade de cada instrução está definida de forma precisa no *datasheet* do ISA AVR.

Não criámos um plano de testes formal mas foram aplicados os princípios básicos da teoria de testes na criação dos mesmos. Por exemplo, o método `execute` de todas as instruções foi testado e tentámos criar testes que testassem todas as estruturas de controlo e todos os ramos do código (*branch coverage*) de cada instrução. Também tivemos atenção às condições de fronteira e à afectação de todas as *flags* de *status* do microcontrolador.

Fizemos 251 testes nas 78 classes das instruções que implementámos (ver Figura 6.1).

Os testes foram realizados recorrendo ao *framework* de testes para Java JUnit. Na Listagem 6.1 podemos ver um exemplo de uma classe de teste. No método anotado com `@BeforeClass`, que é executado apenas uma vez antes da execução do conjunto de testes presente na classe, é criada uma instância do `CPU` e outra da

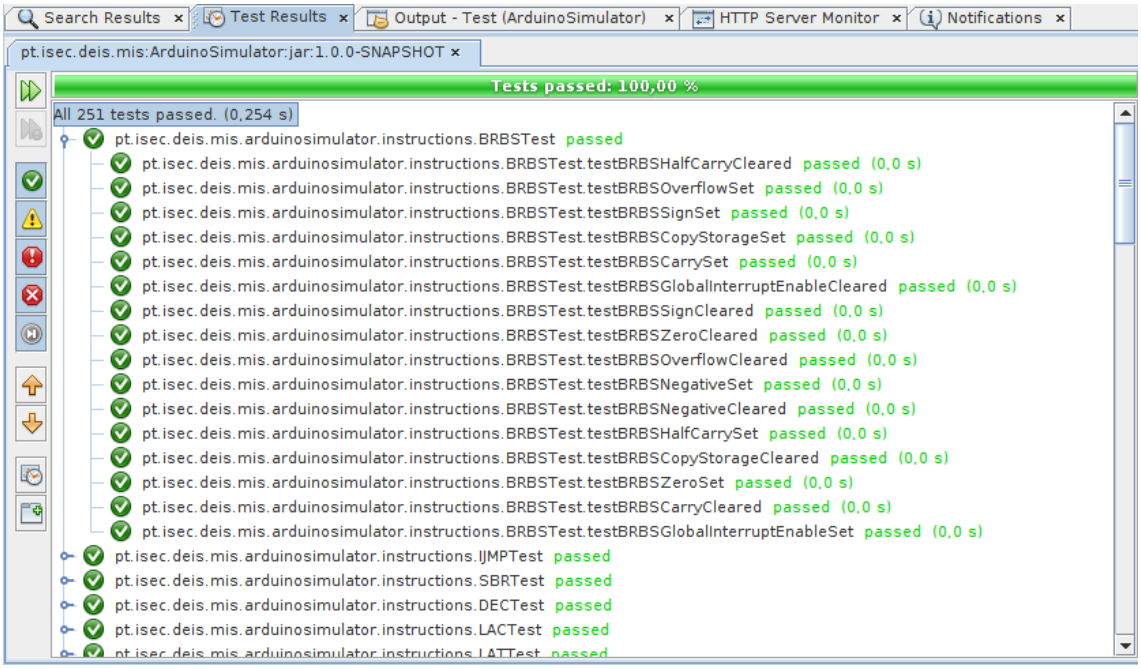


Figura 6.1: Resultados dos testes unitários

instrução a testar. No método anotado com `@Before`, que é executado antes de cada teste, é feito o *reset* ao microcontrolador para repor o estado inicial do sistema.

```
public class BRBSTest {
    static CPU cpu = null;
    static BRBS instance = null;

    @BeforeClass
    public static void setUpClass() {
        cpu = new ATmega328P();
        instance = new BRBS();
    }

    @Before
    public void setUp() {
        cpu.reset();
    }

    @Test
    public void testBRBSCarryCleared() {
        cpu.getSRAM().getStatusRegisterObj().setCarry(false);
        int pc = cpu.getPc()+1;
        //execute
        instance.execute(cpu, 0, 10);
        //assert result
        assertEquals(pc, cpu.getPc());
    }
    ...
}
```

Listagem 6.1: Exemplo de teste JUnit

Nos métodos anotados com `@Test`, é realizado o teste, começando por colocar as condições iniciais especificas deste teste, executando o método a testar e depois validando o resultado esperado.

6.1.1 Erros encontrados com testes unitários

Com a execução dos testes unitários encontrámos alguns erros que foram mais tarde corrigidos. Portanto a realização dos testes foi útil para o projecto cumprindo o seu propósito de encontrar erros e melhorar a qualidade do software.

Tabela 6.1: Erros encontrados com testes unitários

Classe	Erro	Observações
ADC	Ao validar se o resultado é zero não verificava só os 8 bits	Afectava todas as instruções que afectam a flag Z
ADIW	Ao validar se o resultado é zero não verificava os 16 bits	Afectava todas as instruções de 16 bits que afectam a flag Z
BST	Erro ao chamar setCopyStorage com false. A mascara estava invertida	O erro era na classe DataMemory
LDI	Não validava o endereço do registo que só pode ser entre 16 e 31	
LDS16	Não validava o endereço do registo que só pode ser entre 16 e 31 e o K que não pode ser >127	
ROR	Erro a calcular a flag N	

6.2 Testes funcionais

Ao longo do desenvolvimento fomos realizando testes funcionais, normalmente logo após a implementação da funcionalidade. Esta é uma das vantagem do desenvolvimento ágil, ter quase sempre um produto funcional e que se pode testar de imediato. Dos muitos testes funcionais feitos destacamos os apresentados de seguida.

Num dos testes detectámos um problema com a geração de interrupções. Nesse teste uma interrupção não estava a ocorrer quando esperado. Surgiu a duvida se o *datasheet* teria a informação correcta sobre os *bits* de configuração da interrupção no registo External Interrupt Control Register A (EICRA). Para verificar o estado em que o código Arduino colocava esses *bits* usámos um programa de Arduino (ver Listagem 6.2) que mostra o conteúdo desse registo num LCD como podemos ver na Figura 6.2. Com este teste foi possível verificar que a informação no *datasheet* está correcta e detectámos um erro na aplicação de uma máscara de *bits* a um outro registo relacionado com as interrupções. Em vez de ser feito um `AND` com a máscara de *bits* estava a ser feito um `OR`.

Durante a realização da experiência de campo detectámos um problema com a utilização da porta série. Quando se escrevia para a porta série num ciclo sem qualquer *delay* a interface ficava aparentemente bloqueada não respondendo atempadamente aos comandos do utilizador. Depois de feita uma análise ao problema, descobrimos que não se tratava de um erro propriamente dito mas a consequência

```
void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);

  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  ...
  lcd.setCursor(0, 0);
  lcd.print("EICRA: ");

  char *ptr;
  ptr = 0x69;  //endereco EICRA
  binario(*ptr);
  lcd.setCursor(7, 0);
  lcd.print(bin);

  delay(100);
}

void blink() {
  state = !state;
}

void binario(char c) {
  for(int i=0, j=7; i<8; i++, j--) {
    int mask = 1<<i;
    bin[j] = (c&mask)==mask?'1':'0';
  }
  bin[8] = 0;
}
```

Listagem 6.2: Código Arduino para testar o registo EICRA

da quantidade excessiva de mensagens enviadas para o cliente. Como o sistema de mensagens entre servidor e cliente funciona por eventos, quando o cliente carrega por exemplo no botão de pausar a simulação, apenas é enviado ao servidor o comando correspondente e só quando o servidor responde com um novo estado da simulação é que o cliente reage. Como existe uma grande quantidade de mensagens com dados da porta série no *buffer* da *Websocket*, que demoram a ser tratadas, a mensagem com o novo estado vem muito atrasada em relação à acção do utilizador. Foi precisamente por esta razão que não implementámos funcionalidades como o PWM, por gerarem demasiadas mensagens entre o servidor e o cliente. Para resolver esta situação limitámos a taxa de envio a que é possível enviar dados para a porta série. No código inicial, ao ser escrito para a porta série, o *byte* era imediatamente enviado para o cliente e marcávamos logo o *buffer* como livre para receber novos dados. Isto inicialmente evitou a necessidade de gerar uma interrupção para avisar o programa

Figura 6.2: Valor do registo EICRA num LCD

que o *buffer* de saída estava vazio. Para limitar a taxa de envio deixámos de marcar imediatamente o *buffer* como vazio, devolvendo o controlo ao programa, e depois de um pequeno atraso é marcado o *buffer* como vazio e é gerada a respectiva interrupção. Esta correcção fez com que a interface de utilizador ficasse responsiva mesmo a escrever para a porta série sem qualquer *delay* entre escritas.

Capítulo 7

Validação

Segundo Ryan e Wheatcraft, 2017, uma das definições de validação indica que "é o conjunto de actividades que garantem e dão confiança de que um sistema é capaz de atingir a utilização, as metas e os objectivos pretendidos (ou seja, atender aos requisitos das partes interessadas) no ambiente de operação pretendido. A validação garante que se construiu o sistema certo".

Para confirmarmos se o simulador pode ou não ser usado numa sala de aula de introdução à plataforma Arduino, realizámos um caso de estudo em aulas do ensino secundário. Queremos saber se a introdução do Simulador na aula é viável, se não altera a dinâmica da aula e se traz vantagens.

7.1 Descrição da experiência de campo

O caso de estudo de uso real foi realizado em duas escolas secundárias da região de Coimbra para avaliar a usabilidade e as vantagens do uso do simulador na sala de aula. O caso de estudo compreendeu 5 aulas, 3 da *Escola Secundária de Avelar Brotero* (Coimbra) e 2 do *Agrupamento de Escolas de Pombal* (Pombal, distrito de Leiria). Nas duas escolas, os alunos tinham entre 16 e 18 anos. Algumas das aulas pertenciam a cursos da área tecnológica, enquanto outras pertenciam à área de saúde. A participação na experiência foi opcional e não notámos nenhuma reserva de nenhum aluno. Em cada aula, metade dos alunos usava Arduinos reais e a outra metade usava o simulador. A Tabela 7.1 resume as características das turmas e dos alunos e como os alunos ficaram distribuídos entre a utilização do simulador e de Arduinos reais na realização dos testes.

Na Escola Secundária de Avelar Brotero, as actividades decorreram no âmbito da disciplina de Aplicações Informáticas B, durante 100 minutos em cada uma. A disciplina de Aplicações Informáticas B é uma disciplina opcional do último ano do curso, entre duas escolhidas voluntariamente pelos alunos. Aborda-se nesta disciplina uma introdução à algoritmia e à programação e temas de multimédia. Foi neste âmbito, com alunos que já programavam na linguagem C#, em que alguns deles já tinham tido contacto com a placa e com a programação em Arduino, que se desenvolveram as actividades.

No Agrupamento de Escolas de Pombal, o simulador de Arduino foi testado por dois grupos de estudantes de um curso técnico – Técnicas de Electrónica, Automação e Computadores (TEAC) – do 4º nível de qualificação do *Quadro Nacional de Qualificações*. Este grupo de teste tem uma aula de Sistemas Digitais, onde

Tabela 7.1: Caracterização dos alunos.

Escola	Turma	Área	Ano curricular	# alunos	# S[1]	# R[2]
Avelar Brotero	Turma 1	Ciências	12º	29	12	17
Avelar Brotero	Turma 2	Saúde	12º	29	12	17
Avelar Brotero	Turma 3	Mista	12º	20	12	8
Pombal	Turma 4	Electrónica	11º	11	6	5
Pombal	Turma 5	Electrónica	12º	12	7	5
			Total:	101	49	52

os alunos aprendem a linguagem de programação C e, em seguida, como trabalhar com microcontroladores Arduino (entre outros assuntos), nessas aulas, eles têm uma componente teórica e prática. Neste grupo de teste, para a 1ª turma foi o primeiro contacto com o Arduino, enquanto os outros alunos (2ª turma) já sabiam como trabalhar com ele.

O simulador foi executado no Minerva Cluster que pertence ao Laboratório de Computação de Elevado Desempenho (LaCED) hospedado no Instituto Superior de Engenharia de Coimbra (ISEC) em Coimbra. O nó de execução tinha 2 Intel Xeon E5-2695v2 CPUs (12 cores cada) @ 2.40 GHz e 192GB de memória RAM. Apesar de termos usado um *cluster* de alta performance o software podia ter sido executado num servidor normal.

7.2 Metodologia

Um dos pilares desta experiência é analisar o impacto da utilização do simulador na aula. Fazer a experiência em duplicado mudando apenas a ferramenta usada permite-nos aferir esse impacto, por isso organizámos a experiência dividindo as turmas em dois grupos tendo o cuidado de equilibrar os grupos em termos de experiência e conhecimentos tanto em programação como na Plataforma Arduino. Como os respectivos docentes são quem melhor conhece estas características dos alunos contámos com a sua ajuda para fazer esta divisão.

Um dos grupos fez um conjunto de exercícios básicos com um Arduino Uno. O outro grupo fez os mesmos exercícios mas usando o Simulador. Nas aulas da Escola Secundária de Avelar Brotero, devido ao tamanho das turmas e à falta de computadores para todos os alunos, os exercícios foram realizados em grupos de 2 alunos. Este já era o cenário habitual para essas aulas e não teve nada a ver com a experiência.

Os exercícios apresentados aos alunos consistiram em desafios de programação envolvendo circuitos simples e foram os mesmos para ambos os grupos. Estes exercícios eram os habituais para essas aulas, foram preparados pelos professores e não foram influenciados ou tiveram alguma alteração relacionada com a utilização do simulador.

Medimos a eficiência do simulador como uma ferramenta de ensino observando o tempo que os alunos levaram a resolver os exercícios, comparando a utilização do Arduino real com o simulador, o número de exercícios completados e o seu resultado

[1]Usaram o simulador

[2]Usaram um Arduino Uno real

final (correcto/incorrecto). Para avaliar a percepção dos alunos sobre a utilização do simulador usámos um questionário.

7.3 Exercícios

Foram usados três exercícios em cada teste. Estes exercícios foram definidos pelos docentes seguindo o seu habitual plano de trabalho para as aulas e não foram influenciados pela utilização do simulador. Os exercícios tinham uma dificuldade incremental. Todos os exercícios implicam tanto programar como montar um circuito electrónico. No caso de usarem um Arduino real o circuito foi montado numa *breadboard* e no caso da utilização do simulador foi desenhado no cliente do simulador. Em ambas as situações, tanto a escrita do código como a programação do dispositivo, foi feita no IDE do Arduino.

Os exercícios foram os seguintes:

1. Piscar um LED ficando um segundo aceso e um segundo apagado.
2. Fazer acender 3 LEDs em sequência, garantindo que apenas um está aceso de cada vez, e com um intervalo de meio segundo.
3. Fazer acender um conjunto de 3 LEDs de modo intermitente (todos ao mesmo tempo) apenas quando um botão de pressão ligado ao Arduino está pressionado.

Para o último exercício, uma vez que é mais complexo porque implica ler o estado de um pino de entrada e a activação da resistência de PULLUP nessa entrada, demos uma explicação teórica do funcionamento deste tipo de circuito antes da realização do exercício.

7.4 Questionários

Para fazer uma análise subjectiva das dificuldades e do esforço sentidos durante a realização dos exercícios em ambos os grupos necessitamos de recolher a opinião dos alunos pelo que optámos por fazer questionários.

Quando se pretende introduzir uma nova técnica na realização de uma tarefa, para comparar o esforço entre a técnica nova e a anterior ou mesmo para comparar duas técnicas novas, habitualmente são usados os inquéritos *NASA Task Load Index* (NASA-TLX) (Hart & Staveland, 1988). Estes questionários foram criados pelo *Human Performance Group* da *National Aeronautics and Space Administration* (NASA) para fazer a avaliação do esforço realizado no cumprimento de uma determinada tarefa e têm a dupla vantagem de ter em consideração o ponto de vista dos indivíduos e incluir aspectos subjectivos como o desconforto ou o stress.

Foram usados para medir a carga de trabalho cognitiva na introdução e leitura de dados num sistema *electronic health record* ao invés de um sistema em papel (Colligan, Potts, Finn & Sinkin, 2015); para medir a carga subjectiva da tarefa de procurar um objecto com a ajuda de um mapa 2D versus mostrar uma fotografia dos arredores do objecto e a direcção de onde está (Funk et al., 2014); para medir a carga de trabalho mental de pessoas idosas a introduzir números num sistema *touch screen* com 3 técnicas diferentes (*keypad*, botões mais e menos e caixa de

selecção) (Sani & Petrie, 2019); para medir a carga de trabalho cognitiva ao usar um sistema de reconhecimento de gestos com duas técnicas diferentes (*fixed* e *bi-level*) (Katsuragawa, Kamal, Liu, Negulescu & Lank, 2019); para medir a carga de trabalho sob condições de teste em dois sistemas diferentes de leitores de ecrã para pessoas com dificuldades de visão (Yesilada, Stevens, Harper & Goble, 2007); para medir a carga de trabalho mental ao escrever notas num dispositivo móvel num teclado com reconhecimento de escrita manual com e sem *feedback* visual (Dai, Sears & Goldman, 2009). Cerca de 7% dos casos de uso deste tipo de questionários estão relacionados com utilização de computadores (Hart, 2006).

A utilização de inquéritos que já foram validados dá mais garantias que vamos obter dados fiáveis.

Os inquéritos NASA-TLX são constituídos por 2 partes. Na primeira são apresentadas 6 escalas subjectivas para avaliar os seguintes parâmetros:

- **Esforço mental (*Mental Demand*):** Quanta actividade mental e perceptual foi necessária (por exemplo, pensar, decidir, calcular, lembrar, olhar, pesquisar etc.)? A tarefa foi fácil ou exigente, simples ou complexa, exigiu exactidão ou não?

- **Esforço físico (*Physical Demand*):** Quanta actividade física foi necessária (por exemplo, empurrar, puxar, rodar, controlar, activar etc.)? A tarefa foi fácil ou exigente, lenta ou rápida, leve ou extenuante, repousante ou trabalhosa?

- **Pressão com o tempo (*Temporal Demand*):** Quanta pressão com o tempo se sentiu devido à taxa ou ritmo em que as tarefas ou elementos da tarefa ocorreram? O ritmo era lento e descontraído ou rápido e frenético?

- **Cumprimento da tarefa (*Performance*):** Quão bem-sucedido se acha que alcançou os objectivos da tarefa definida pelo investigador (ou por si próprio)? Quão satisfeito se está com o desempenho em atingir esses objectivos?

- **Esforço (*Effort*):** Quão difícil se teve que trabalhar (mental e fisicamente) para atingir o nível de desempenho atingido?

- **Incómodo (*Frustration*):** Quão inseguro, desanimado, irritado, stressado ou aborrecido se sentiu em contradição com seguro, gratificado, satisfeito, relaxado ou complacente durante a realização da tarefa?

Em cada uma destas escalas subjectivas o individuo deve classificar numa escala de 1 a 20 o grau de dificuldade que sentiu nesse parâmetro, sendo por exemplo: 1 baixo/sem dificuldade e 20 alto/muita dificuldade. Esta primeira parte do questionário é repetida para cada vez que o individuo realiza a tarefa.

Na segunda parte é apresentado ao individuo pares das combinações de todos os parâmetros avaliados para que o individuo escolha o que considera, de entre os dois, o mais importante. Esta escolha ajuda a dar pesos aos parâmetros no momento de calcular a taxa de esforço total da tarefa. Isto implica que os pesos são diferentes entre os vários indivíduos que realizaram as tarefas. No caso dessas tarefas em análise serem substancialmente diferentes, esta segunda parte do inquérito deve ser realizada novamente e os pesos recalculados.

Para este caso optámos por simplificar o inquérito removendo o parâmetro *Effort* uma vez que neste contexto pode ser obtido individualmente pelos parâmetros

Mental Demand e *Physical Demand*. Optámos também por adaptar a escala das respostas de 1 a 20 para 1 a 6 para evitar que os alunos sentissem pressão de terem de ser demasiado precisos no momento de classificar cada um dos parâmetros da escala.

Introduzimos questões adicionais para perceber o *background* dos alunos para mais tarde podermos verificar se existe alguma possível correlação com o desempenho mostrado aquando da utilização do simulador (Tabela 7.4 na Secção 7.5).

7.5 Resultados e Discussão

Depois de uma primeira avaliação dos questionários notámos que nem todos os alunos responderam a todas as perguntas. De modo a mitigar esta situação excluímos dos inquéritos os alunos que não responderam por completo à segunda parte do NASA-TLX, 3 indivíduos a que correspondiam 4 exercícios, e os exercícios onde não foram registados os tempos de execução ou as respostas à primeira parte do inquérito NASA-TLX, que foram mais 17 exercícios. Desse modo ficámos com um total de 189 exercícios válidos, sendo que 89 foram realizados no ambiente real e 100 no simulador (ver Tabela 7.2). Comparando com outros estudos de referência da área da computação onde foi usado NASA-TLX temos mais participantes do que (Dai et al., 2009) com 75, (Yesilada et al., 2007) com 10, (Katsuragawa et al., 2019) com 36, (Sani & Petrie, 2019) com 12, (Funk et al., 2014) com 16 e (Colligan et al., 2015) com 74.

Tabela 7.2: Inquéritos validos e inválidos.

	Total	Inválidos	Válidos
Alunos	101	3	98
Exercícios	212	23	189

Deve-se notar que nem todos os alunos realizaram os 3 exercícios propostos na aula, pois só quando acabavam um exercício é que passavam para o seguinte sem ser dado um limite de tempo para realizar cada um deles. A Tabela 7.3 mostra o número de alunos que executaram cada exercício.

Na primeira aula os alunos também realizaram os exercícios todos de seguida, tendo apenas sido contabilizado o tempo total e apenas responderam uma vez à primeira parte do inquérito NASA-TLX. Contudo isto aconteceu tanto aos alunos que usaram o Arduino real como aos que usaram o simulador e continua a ser possível fazer a comparação entre ambos.

Tabela 7.3: Distribuição dos exercícios pelas turmas.

	Exercício 1		Exercício 2		Exercício 3	
Turma	Sim.	Real	Sim.	Real	Sim.	Real
1	8	13	-	-	-	-
2	12	16	12	12	4	3
3	11	8	10	8	8	2
4	5	4	5	4	5	4
5	7	5	7	5	6	5

A Figura 7.1 apresenta tanto a média como a mediana da taxa de esforço dos alunos na realização dos exercícios assim como a média e a mediana do tempo de resolução em ambos os ambientes (real e simulado).

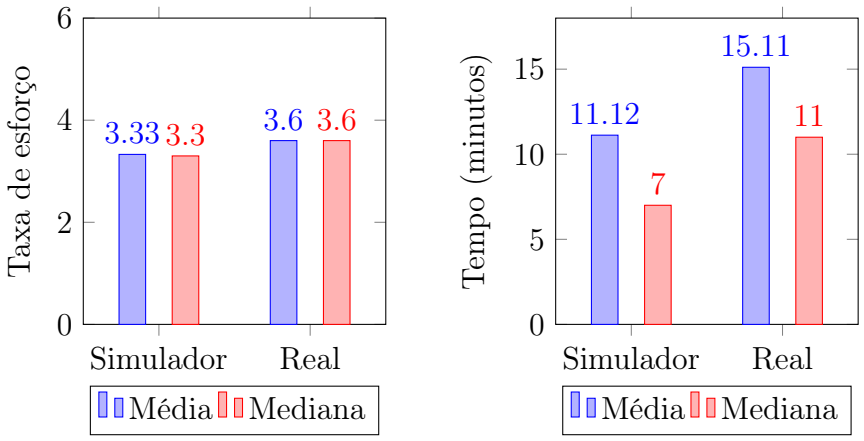


Figura 7.1: Comparação da taxa de esforço e do tempo de resolução dos exercícios.

Como podemos ver, o índice da taxa de esforço parece ser aproximadamente o mesmo para o Arduino real e para o simulador, sugerindo que o uso do simulador não interfere muito com o esforço geral experienciado pelos alunos, embora ao usar o simulador seja de cerca de 8% inferior, o que é um resultado positivo para o uso do simulador.

Em relação ao parâmetro tempo, notamos que os alunos que usam o simulador levam muito menos tempo para resolver os exercícios: 26% em média e 36% em mediana. Combinado com o menor esforço físico, isso pode indicar que o uso do simulador é mais intuitivo do que fazer ligações eléctricas numa *breadboard*. Este resultado sugere que o uso do simulador é benéfico, considerando o número de exercícios possíveis de executar durante a aula.

Em relação ao esforço físico (Figura 7.2), também observamos uma melhora significativa (um terço). Já esperávamos uma melhoria, pois é mais fácil mover o ponteiro do rato num ecrã do que manipular pequenos componentes. O facto de as melhorias serem tão significativas é um resultado muito encorajador, sugerindo que o uso do simulador afecta positivamente o processo de aprendizagem. Considerando o esforço mental e o desempenho (conclusão do exercício), os resultados são os mesmos para os dois grupos de alunos (Figura 7.2). Isso também era esperado: não havia limite de tempo, portanto, a conclusão depende principalmente do próprio exercício. O esforço mental também não deve variar muito uma vez que o IDE e o esforço de programação é o mesmo nos dois casos. Na verdade, isso está de acordo com os objectivos de não introduzir interferência no processo de desenvolvimento.

Para confirmar que os resultados obtidos não foram influenciados por experiência prévia dos alunos calculámos a correlação entre a resposta a perguntas do inquérito feito, onde questionámos sobre experiência anterior com programas de desenho, programação em geral e Arduino em particular, com a taxa de esforço e o tempo de realização dos exercícios. Fizemos o calculo da *Point Biserial Correlation* (Tate, 1954) entre a resposta "Sim" das três perguntas na Tabela 7.4 e a média do tempo de resolução dos exercícios e a média da taxa de esforço sentida durante a realização dos exercícios.

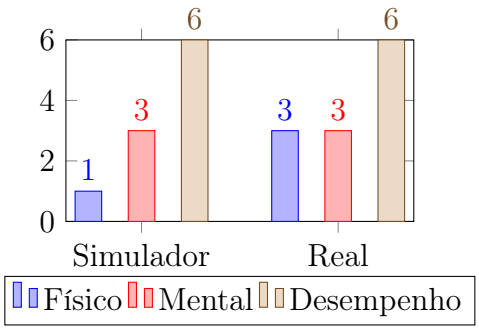


Figura 7.2: Comparação da mediana do esforço físico, esforço mental e desempenho.

No caso de experiência anterior a usar software de desenho e experiência anterior com o Arduino, a correlação é muito baixa, sugerindo que esses dois não estão relacionados com os resultados dos testes. Em relação à experiência anterior em programação, a correlação é um pouco maior, mas também não é significativa.

Tabela 7.4: Correlação entre a resposta "Sim", o tempo de resolução e a taxa de esforço.

Pergunta	Tempo	Esforço
Costuma usar programas de desenho?	-0.099	0.030
Já tinha feito programação antes deste ano letivo?	-0.131	-0.158
Já tinha feito programas para Arduino antes desta disciplina?	-0.047	-0.025

7.6 Ponto de vista dos professores

O ponto de vista dos professores é muito relevante para a nossa análise uma vez que, além dos alunos, também são parte interessada no bom funcionamento das aulas. Os professores das turmas do caso de estudo estiveram envolvidos em todas as etapas de preparação e, na sua opinião, o uso do simulador não introduziu nenhum trabalho extra de gestão da aula, e dispensar a manipulação dos componentes físicos aliviou o início e o final da aula. Existe a necessidade de uma explicação inicial aos alunos sobre a utilização do simulador, mas isso é apenas para a primeira aula em que o mesmo é utilizado. Até agora, parece que o simulador não envolve carga de trabalho extra para os professores.

7.7 Sumário

O simulador foi testado, em sala de aula, num caso de estudo em duas escolas do ensino secundário. Recolhemos métricas sobre taxa de esforço, tanto mental como físico, e de desempenho. Da análise dos resultados obtidos podemos concluir que o simulador não tem qualquer impacto negativo no funcionamento da aula e que oferece benefícios em termos de esforço físico e no tempo de execução dos exercícios.

Capítulo 8

Conclusões e Trabalho futuro

Dada a crescente utilização da Plataforma Arduino como uma ferramenta-chave de aprendizagem, é pertinente abordar os aspectos em que esse tipo de utilização podem ser melhorados. Identificámos um conjunto de aspectos nos quais o uso do Arduino em sala de aula pode beneficiar do uso de um simulador de Arduino, incluindo novas oportunidades que podem ser exploradas para benefício mútuo de professores e alunos.

8.1 Objectivos atingidos

Apresentámos o planeamento e desenvolvimento de um simulador de Arduino que, embora possa ser usado para fins gerais, é especificamente voltado para o uso no contexto educacional, pois os seus objectivos e requisitos foram baseados nas necessidades que se identificaram para uso na sala de aula. O simulador foi implementado em Java e pode ser executado nos computadores típicos normalmente encontrados nas escolas. Possui uma arquitectura cliente-servidor baseada na Web, permitindo que seja gerido centralmente e usado remotamente, possibilitando cenários de ensino à distância. Mais importante ainda, é compatível com o IDE habitual para o Arduino e não tem impacto nos procedimentos normais do programador. Deste modo podemos dizer que cumprimos com todos os objectivos elencados na Secção 1.2, embora a velocidade de execução do simulador tenha algum défice em relação ao que nos propusemos atingir inicialmente.

O código fonte do projecto está disponível livremente no GitHub[1].

8.2 Validação com caso de estudo

Testámos e validámos o uso do simulador em sala de aula num caso de estudo envolvendo duas escolas de ensino secundário com cinco turmas, utilizando os mesmos exercícios já planeados pelos professores em contexto regular. Recolhemos métricas sobre a taxa de esforço mental e físico experienciado pelos alunos e também métricas relacionadas com o desempenho, como o tempo gasto na realização dos exercícios. Concluímos que a utilização do simulador não teve impacto negativo nos alunos ou na gestão da aula e observámos uma melhora significativa na carga física de trabalho

[1] https://github.com/pafgoncalves/ArduinoSimulator

e no tempo necessário para resolver os exercícios. Essa melhoria pode ter um impacto muito positivo na eficiência do tempo da aula, possibilitando mais exercícios por aula. No que diz respeito ao ponto de vista dos professores, o *feedback* é que não foram introduzidos aspectos negativos, que a gestão da aula é mais fácil e tudo o que é necessário é uma explicação inicial aos alunos sobre a utilização do simulador.

Analisámos a correlação da experiência passada dos alunos com os resultados obtidos. Não encontrámos correlação significativa e assumimos que as melhorias de desempenho observadas estão realmente relacionadas com a utilização do simulador, sugerindo que o seu uso em sala de aula é benéfico para os objectivos de aprendizagem.

8.3 Contribuições

Com este trabalho conseguimos implementar o simulador proposto e conseguimos mostrá-lo e usá-lo em contexto de aula.

Consideramos que conseguimos sensibilizar a comunidade para o uso deste tipo de ferramentas com as várias apresentações e workshops realizados, nomeadamente em instituições de ensino como no Politécnico de Tomar e na Escola Secundária Avelar Brotero com a presença de professores de toda a zona centro do país.

Deixamos também a publicação de três artigos científicos, um focado na escolha da técnica de virtualização para implementar um simulador especificamente em Java, outro onde se apresenta o trabalho realizado com todas as suas características e funcionalidades e um terceiro onde se faz a análise de um caso de estudo da aplicação do simulador em contexto de aula.

8.4 Trabalho futuro

Mais especificamente há que trabalhar a escalabilidade do sistema implementando as propostas já feitas neste relatório. Existem outras melhorias sugeridas ao longo do relatório que também deverão ser implementadas num futuro breve.

Em termos gerais identificamos vários caminhos para trabalhos futuros: a melhoria contínua do simulador com o objectivo de implementar novas funcionalidades tais como expansão da palete de componentes, capacidade de interacção entre utilizadores (por exemplo permitir ao docente ver os circuitos dos seus alunos), lista de exercícios predefinidos, etc, de modo a aumentar o potencial do mesmo, a continuação de testes em mais cenários relacionados à sala de aula e trabalhar a divulgação para aumentar sua visibilidade e uso. Acreditamos que este simulador é uma contribuição positiva para promover uma precoce e ampliada alfabetização digital e procuraremos disseminar a utilização do simulador nas escolas.

Além do uso no ensino consideramos haver oportunidade para uso em outras áreas. Pode ser usado na indústria como ferramenta de testes a *firmware* antes da sua entrada em produção. Podem mesmo ser realizados testes unitários úteis para prevenir regressões num ciclo de desenvolvimento iterativo. A capacidade de *debug*, com a possibilidade de inserção de *breakpoints*, execução *step-by-step* e inspecção de memória, pode permitir diagnosticar mais facilmente os problemas encontrados. O simulador também pode ser expandido com características que possam simular

acontecimentos inesperados tais como falhas de hardware, possibilitando estudos de confiabilidade de sistemas baseados em Arduinos.

8.5 Outras considerações

Por razões profissionais este trabalho foi realizado sem dedicação total, o que muitas vezes levou a situações de cansaço e consequentes atrasos na realização das tarefas, mas ao mesmo tempo o tema foi muito motivante o que levou a que nunca se desistisse de chegar ao fim. Isto também influenciou a metodologia de trabalho, já que sendo um trabalho longo, obrigou a fazer escolhas em relação à aplicação de técnicas adquiridas durante a realização do mestrado, tentando aproveitar o melhor de cada uma mas sem as aplicar de forma formal e rigorosa, adaptando às necessidades e especificidades do projecto.

Desenvolvido numa área já do nosso interesse, Arduinos e microcontroladores, onde o conhecimento era superficial, aprofundamos os conceitos no tema o que acabou por ser extremamente gratificante e acima de tudo desafiante.

As acções de divulgação e apresentação do simulador realizadas e a participação em conferências também foram muito gratificantes tanto por ajudarem a dar a conhecer o projecto como pelo desenvolvimento pessoal no sentido da melhoria do discurso em público.

Sentimos que os objectivos a que nos propusemos foram claramente alcançados, tendo deixado à comunidade um produto que, apesar de ainda ter muito por onde melhorar, é uma ferramenta que está pronta a ser usada no ensino de programação para Arduino. Ao contrário de outras ferramentas é um produto completo e integrado e de fácil utilização tanto na sala de aula como em casa.

Bibliografia

Prensky, M. (2008). Programming is the new literacy. *Edutopia magazine*.

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42–64. doi:10.21623/1.1.2.4

Direção-Geral da Educação. (2020). eduscratch. Obtido 17 janeiro 2020, de http://eduscratch.dge.mec.pt/

Arduino SA. (2018). Arduino - Home. Obtido 24 novembro 2018, de https://www.arduino.cc/

Barragán, H., Banzi Associate Professor, M. & Crampton Smith Director, G. (2004). *Wiring: Prototyping Physical Interaction Design Thesis Committee*.

Agatolio, F. & Moro, M. (2017). A workshop to promote Arduino-based robots as wide spectrum learning support tools. Em *Robotics in Education* (pp. 113–125). Springer.

Sarik, J. & Kymissis, I. (2010). Lab kits using the Arduino prototyping platform. Em *2010 IEEE Frontiers in Education Conference (FIE)* (T3C–1). IEEE.

Jamieson, P. (2011). Arduino for teaching embedded systems. are computer scientists and engineering educators missing the boat? Em *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)* (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering e Applied Computing (WorldComp).

Atmel. (2018). ATmega328/P Datasheet.

Currie, E. & James-Reynolds, C. (2017). The use of physical artefacts in undergraduate computer science teaching. Em *E-Learning, E-Education, and Online Training* (pp. 119–124). Springer.

Velleman for Makers. (2019). The Ultimate Arduino Board Guide For Beginners and Experts. Obtido 1 março 2020, de https://www.vellemanformakers.com/the-ultimate-arduino-guide/

kumar C S, K., K.V, C., A, N., B, M. & Appaji, I. (2017). Vehicle speed monitoring system using Arduino and speed sensor. *IJRDO - Journal of Computer Science Engineering (ISSN: 2456-1843)*, *3*(2), 33–40. Obtido de https://www.ijrdo.org/index.php/cse/article/view/396

Francillon, A. & Castelluccia, C. (2008). Code injection attacks on harvard-architecture devices. Em *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*. doi:10.1145/1455770.1455775

Atmel. (2020). Atmel-ICE. Obtido 22 fevereiro 2020, de https://www.microchip.com/DevelopmentTools/ProductDetails/ATATMEL-ICE

ScienceProg. (2007). What is DebugWire interface. Obtido 1 março 2020, de https://scienceprog.com/what-is-debugwire-interface/

The GNU Project. (2020). GDB: The GNU Project Debugger. Obtido 22 fevereiro 2020, de https://www.gnu.org/software/gdb/

Arduino SA. (2019). Arduino IDE 1.5 3rd party Hardware specification. Obtido 24 agosto 2019, de https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5-3rd-party-Hardware-specification

Wikipedia contributors. (2020a). Round-robin DNS — Wikipedia, The Free Encyclopedia. Obtido 27 junho 2020, de https://en.wikipedia.org/wiki/Round-robin_DNS

Wikipedia contributors. (2020b). Load balancing — Wikipedia, The Free Encyclopedia. Obtido 27 junho 2020, de https://en.wikipedia.org/wiki/Load_balancing_(computing)

Metawerx. (2008). Sticky Sessions. Obtido 27 junho 2020, de http://wiki.metawerx.net/wiki/StickySessions

Selenium. (2020). Selenium Grid. Obtido 26 junho 2020, de https://www.selenium.dev/documentation/en/grid/

harsha2selenium. (2015). Selenium Grid Architecture. Obtido 2 março 2019, de https://seleniumchallenges.wordpress.com/2015/09/30/selenium-grid-architecture/

The Apache Software Foundation. (2020). Apache Tomcat. Obtido 30 junho 2020, de http://tomcat.apache.org/

Eclipse Foundation. (2020). Eclipse Jetty. Obtido 30 junho 2020, de https://www.eclipse.org/jetty/

Anicas, M. (2014). 5 Common Server Setups For Your Web Application. Obtido 26 julho 2020, de https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application#3-load-balancer-(reverse-proxy)

Smith, J. & Nair, R. (2005). The architecture of virtual machines. *Computer*, *38*(5), 32–38. doi:10.1109/mc.2005.173

Sahoo, J., Mohapatra, S. & Lath, R. (2010). Virtualization: A survey on concepts, taxonomy and associated security issues. *2nd International Conference on Computer and Network Technology, ICCNT 2010*, 222–226. doi:10.1109/ICCNT.2010.49

Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C., Anderson, A. V., ... Smith, L. (2005). Intel virtualization technology. *Computer*. doi:10.1109/MC.2005.163

Van Der Hoeven, J., Lohman, B. & Verdegem, R. (2007). *Emulation for Digital Preservation 123 Emulation for Digital Preservation in Practice: The Results The National Library of the Netherlands Tessella Support Services plc., United Kingdom*. Obtido de http://fabrice.bellard.free.fr/qemu/

Tucker, S. G. (1965). *Emulation of Large Systems*.

Reshadi, M., Mishra, P. & Dutt, N. (1986). Instruction Set Compiled Simulation: A Technique for Fast and Flexible Instruction Set Simulation, 758–763. doi:10.1109/DAC.2003.1219121

Mills, C., Ahalt, S. C. & Fowler, J. (1991). Compiled instruction set simulation. *Software: Practice and Experience*, *21*(8), 877–889. doi:10.1002/spe.4380210807

Zhu, J. & Gajski, D. D. (1995). A Retargetable , Ultra-fast Instruction Set Simulator University of California 3 A New Approach for Static Compiled Sim-. *Simulation*, 1–5.

Aycock, J. (2003). *A Brief History of Just-In-Time*.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... Warfield, A. (2003). *Xen and the Art of Virtualization*.

Iqbal, W. (2009). Service Level Agreement Driven Adaptive Resource Management for Web Applications on Heterogenous Compute Clouds. (September).

Kudlugi, M., Hassoun, S., Selvidge, C. & Pryor, D. (2001). A Transaction-Based Unified Simulation / Emulation.

Dolinskii, M. S., Zisel'Man, I. M. & Fedortsov, A. (1999). In-circuit emulators of microprocessors and microcontrollers.

Huang, I. J., Kao, C. F., Chen, H. M., Juan, C. N. & Lu, T. A. (2002). A retargetable embedded in-circuit emulation module for microprocessors. *IEEE Design and Test of Computers*. doi:10.1109/MDT.2002.1018131

Bellard, F. (2005). *QEMU, a Fast and Portable Dynamic Translator*.

Gajski, D. (2002). An ultra-fast instruction set simulator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *10*(3), 363–373. doi:10.1109/TVLSI.2002.1043339

Cmelik, B. & Keppel, D. (1995). Shade: A Fast Instruction-Set Simulator for Execution Profiling. *Fast Simulation of Computer Architectures*, 5–46. doi:10.1007/978-1-4615-2361-1_2

Braun, G., Hoffmann, A., Nohl, A. & Meyr, H. (2001). Using static scheduling techniques for the retargeting of high speed, compiled simulators for embedded processors from an abstract machine description. *International Symposium on System Synthesis (IEEE Cat. No.01EX526)*, (October), 57–62. doi:10.1109/ISSS.2001.957913

Leupers, R., Elste, J. & Landwehr, B. (1999). Generation of interpretive and compiled instruction set simulators. *Proceedings of Asp-Dac '99: Asia and South Pacific Design Automation Conference 1999*, 339–342. doi:10.1109/ASPDAC.1999.760028

Atmel. (2016). AVR Instruction Set Manual.

Intel Corporation. (2011). Intel(R) 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes. *Architecture*, (December). doi:10.1109/MAHC.2010.22

Atmel. (2019). Atmel AVR GNU Toolchain. Obtido 17 janeiro 2019, de http://distribute.atmel.no/tools/opensource/Atmel-AVR-GNU-Toolchain/

Pees, S., Zivojnovic, V., Ropers, A. & Meyr, H. (1997). Fast Simulation of the TI TMS 320C54x DSP. *Proc. Int. Conf. on Signal Processing Application and Technology (ICSPAT)*, 995–999.

Turner, A. (2009). Tuning The JVM For Unusual Uses. Obtido 21 janeiro 2019, de https://web.archive.org/web/20100529154029/http:%20//nerds-central.blogspot.com/2009/09/tuning-jvm-for-unusual-uses-have-some.html

Oracle. (2018). Java Platform, Standard Edition Tools Reference. Obtido 21 janeiro 2019, de https://docs.oracle.com/javase/8/docs/%20technotes/tools/unix/java.html

Agile Alliance. (2020). Agile 101. Obtido 27 fevereiro 2020, de https://www.agilealliance.org/agile101/

Scrum.org. (2020). Scrum.org. Obtido 26 julho 2020, de https://www.scrum.org/

Wells, D. (2013). Extreme Programming: A gentle introduction. Obtido 26 julho 2020, de http://www.extremeprogramming.org/

Atlassian. (2020). Kanban - A brief introduction. Obtido 26 julho 2020, de https://www.atlassian.com/agile/kanban

Apache Maven. (2020). Welcome to Apache Maven. Obtido 2 março 2020, de https://maven.apache.org/

iText. (2020). iText. Obtido 2 março 2020, de https://itextpdf.com/en

Wikipedia contributors. (2020c). Lookup table — Wikipedia, The Free Encyclopedia. Obtido 21 abril 2020, de https://en.wikipedia.org/wiki/Lookup_table

Klemm, R. (1999). Practical guidelines for boosting Java server performance. Em *Proceedings of the ACM 1999 conference on Java Grande* (pp. 25–34).

Tool Interface Standards Committee and others. (2001). Executable and Linkable Format (ELF). *Specification, Unix System Laboratories*, *1*(1).

Wikipedia contributors. (2020d). Factory (object-oriented programming) — Wikipedia, The Free Encyclopedia. Obtido 1 julho 2020, de https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)

Gamma, E. (1995). Richald Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley, R eading, MA*, *1*(99), 5.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Atmel. (2003). STK500 Communication Protocol.

WebRTC.org. (2020). WebRTC. Obtido 1 março 2020, de https://webrtc.org/

Eager, M. J. (2012). Eager Consulting. *Introduction to the DWARF debugging format*.

Kilic, A. (2016). Java based Elf/Dwarf parser. Obtido 5 julho 2020, de https://github.com/aykutkilic/bicirik-dwarf

The jQuery Team. (2020). jQuery. Obtido 5 julho 2020, de https://jquery.org/

Bootstrap Team. (2020). Bootstrap. Obtido 5 julho 2020, de https://getbootstrap.com/

Herz, A. (2020). Draw2D. Obtido 5 julho 2020, de http://www.draw2d.org/draw2d/

H2 Database. (2020). H2 Database Engine. Obtido 9 julho 2020, de http://h2database.com/html/main.html

Mozilla Developer Network. (2019). WebSockets. Obtido 9 julho 2020, de https://developer.mozilla.org/pt-BR/docs/WebSockets

Arduino SA. (2020). package_index.json specification. Obtido 13 julho 2020, de https://arduino.github.io/arduino-cli/package_index_json-specification/

Ryan, M. J. & Wheatcraft, L. S. (2017). On the use of the terms verification and validation. Em *INCOSE International Symposium* (Vol. 27, *1*, pp. 1277–1290). Wiley Online Library.

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, *12*(3), 23–49.

Hart, S. G. & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. Em *Advances in psychology* (Vol. 52, pp. 139–183). Elsevier.

Colligan, L., Potts, H. W., Finn, C. T. & Sinkin, R. A. (2015). Cognitive workload changes for nurses transitioning from a legacy system with paper documentation to a commercial electronic health record. *International journal of medical informatics*, *84*(7), 469–476.

Funk, M., Boldt, R., Pfleging, B., Pfeiffer, M., Henze, N. & Schmidt, A. (2014). Representing indoor location of objects on wearable computers with head-

mounted displays. Em *Proceedings of the 5th Augmented Human International Conference* (pp. 1–4).

Sani, Z. H. A. & Petrie, H. (2019). Older Adults' Number Entry Using Touchscreen and Keyboard-Mouse Computers. Em *International Visual Informatics Conference* (pp. 353–367). Springer.

Katsuragawa, K., Kamal, A., Liu, Q. F., Negulescu, M. & Lank, E. (2019). Bi-Level Thresholding: Analyzing the Effect of Repeated Errors in Gesture Input. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *9*(2-3), 1–30.

Yesilada, Y., Stevens, R., Harper, S. & Goble, C. (2007). Evaluating DANTE: Semantic transcoding for visually disabled users. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *14*(3), 14–es.

Dai, L., Sears, A. & Goldman, R. (2009). Shifting the focus from accuracy to recallability: A study of informal note-taking on mobile information technologies. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *16*(1), 1–46.

Hart, S. G. (2006). NASA-task load index (NASA-TLX); 20 years later. Em *Proceedings of the human factors and ergonomics society annual meeting* (Vol. 50, *9*, pp. 904–908). Sage Publications Sage CA: Los Angeles, CA.

Tate, R. F. (1954). Correlation between a discrete and a continuous variable. Point-biserial correlation. *The Annals of mathematical statistics*, *25*(3), 603–607.

Apêndice A

Trabalhos publicados e divulgação

A.1 ArduinoDay2019@IPT

O ArduinoDay é um evento de projecção mundial celebrado em simultâneo em várias cidades de todo o mundo.

A edição organizada pelo Instituto Politécnico de Tomar, através dos seus cursos de Engenharia Eletrotécnica e de Computadores e Engenharia Informática, foi realizada no dia 18 de Março de 2019.

A convite do Fikalab ISEC, apresentámos no evento o projecto do Simulador de Arduino, que estava ainda numa fase embrionária de desenvolvimento.

Figura A.1: Apresentação ArduinoDay@IPT

Figura A.2: Certificado de participação no ArduinoDay2019@IPT

A.2 CISTI'2019

A CISTI é um evento técnico-científico anual, que visa a apresentação e a discussão de conhecimentos, novas perspectivas, experiências e inovações no domínio dos sistemas e tecnologias de informação.

A edição de 2019, a CISTI'2019 - 14ª Conferência Ibérica de Sistemas e Tecnologias de Informação, realizou-se entre 19 e 22 de Junho de 2019, em Coimbra, Portugal.

No dia 20 de Junho de 2019 apresentámos o artigo "Tecnologias de Virtualização Para Simulação de Arduino", DOI:10.23919/CISTI.2019.8760727.

A.3 TIC@Portugal2019

TIC@Portugal é uma iniciativa da Associação EDUCOM – APTE (Associação Portuguesa de Telemática Educativa), através do seu Centro de Competência TIC (Tecnologias de Informação e Comunicação) e do seu Centro de Formação de Professores. É um evento tem como objectivo reflectir sobre as práticas do uso das TIC na Educação. Pretende ouvir os educadores e professores que no terreno usam as TIC, através da apresentação do seu trabalho, e convida especialistas a contribuírem com o que de mais recente se sabe neste domínio. É uma oportunidade para se divulgar e debater a utilização das TIC nos processos de ensino e de aprendizagem.

No dia 5 de Julho de 2019 apresentámos em Coimbra o Simulador de Arduino aos participantes da conferência.

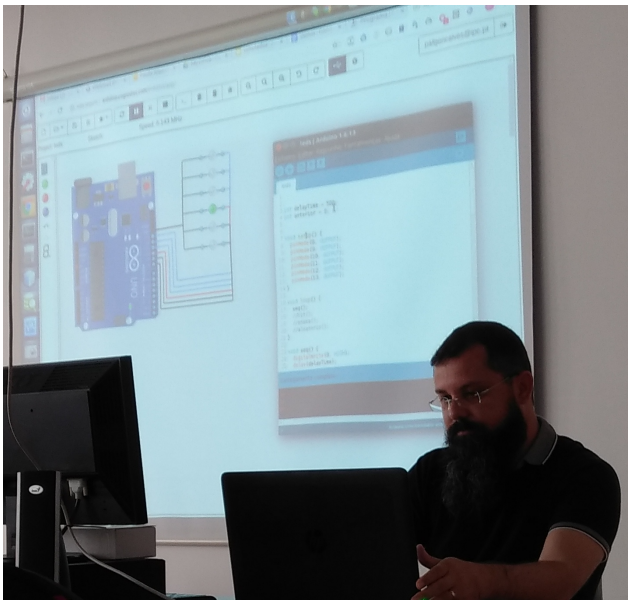


Figura A.3: Apresentação TIC@Portugal'19

educom
Centro de Formação
ccTIC

TIC@Portugal'19
XIII Encontro de Boas-práticas educativas
uma iniciativa educom

Certificado

Certifica-se que
Paulo Gonçalves

Apresentou a comunicação **Simulador de Arduino** no evento
TIC@Portugal'19 – Ambientes Digitais Potenciadores da Aprendizagem
promovida pelo **Centro de Competência TIC - Softciências** em Coimbra, na Escola Secundária de Avelar Brotero, no dia 5 de julho de 2019.

Coimbra, 5 de julho de 2019

O Coordenador do CCTIC- Softciências

(Jaime Carvalho e Silva)

ccTIC · Avelar Brotero · Minerva · educom · erte · República Portuguesa

Figura A.4: Certificado de apresentação na TIC@Portugal'19

A.4 AmiEs2019

O International Symposium on Ambient Intelligence and Embedded Systems (AmiEs) é um simpósio internacional que tem como objectivo compartilhar conhecimento e experiências nas áreas de Inteligência Ambiental, Sistemas Embebidos, Programação na Internet e Tecnologia da Informação.

No dia 13 de Setembro de 2019 apresentámos em Coimbra o artigo "An Arduino Simulator for Practical Embedded Programming Teaching".

A.5 Fikalab ISEC Challenge 2019

O Fikalab ISEC é um espaço que visa criar um ambiente para experimentação, desenvolvimento de ideias e diversão saudável dentro do ISEC. O objectivo é ligar a academia ao mundo real, tendo óptimas ideias para *coisas* e realmente construí-las, desafiando o *status quo*.

O Fikalab ISEC Challenge é um concurso que pretende premiar os melhores projectos realizados no Fikalab ISEC.

No dia 20 de Novembro de 2019, na final do Challenge de 2019, o Simulador de Arduino foi premiado com uma menção honrosa.

Figura A.5: FIKALAB ISEC Challenge 2019

No dia 30 de Julho de 2020 leccionámos um *workshop online* sobre Arduino usando o simulador no *Critical Summer Camp — Internships*.

No âmbito do Fikalab também já foram feitas outras apresentações sobre o Simulador de Arduino a colaboradores da Critical Software e a outros convidados.

Figura A.6: Menção honrosa no FIKALAB ISEC Challenge 2019

A.6 ICPEC2020

A ICPEC é uma conferência que pretende ser um espaço frequentado por professores e investigadores para discutir tópicos que promovam novas metodologias, melhores práticas, tendências, técnicas e ferramentas para melhorar o processo de ensino-aprendizagem da programação de computadores.

No dia 25 de Junho de 2020 apresentámos *online* o artigo "An Arduino Simulator in Classroom - a Case Study", DOI:10.4230/OASIcs.ICPEC.2020.12.

Figura A.7: Certificado de apresentação na ICPEC2020

Apêndice B

Protocolo da experiência

A realização da experiência que serviu de base ao caso de uso de validação deste trabalho regeu-se pelos parâmetros descritos de seguida.

Selecção dos participantes:

- Os participantes devem ser alunos do nível de escolaridade secundário;
- Os participantes devem ter idades entre os 15 e os 18 anos;
- Os participantes devem estar enquadrados em aulas práticas onde seja usada a Plataforma Arduino.

Preparação:

- Os participantes devem ser divididos em 2 grupos em que um grupo usa o simulador e o outro o hardware real;
- A divisão dos participantes pelos dois grupos deve ser feita de modo a ter grupos equilibrados em termos de experiência e conhecimentos em programação e Plataforma Arduino;
- Os participantes podem realizar os exercícios individualmente ou em grupos de dois;
- Junto com a introdução à Plataforma Arduino deve ser dada uma breve explicação do funcionamento do simulador;
- A realização da experiência deve interferir o mínimo possível com o funcionamento habitual da aula.

Realização dos exercícios:

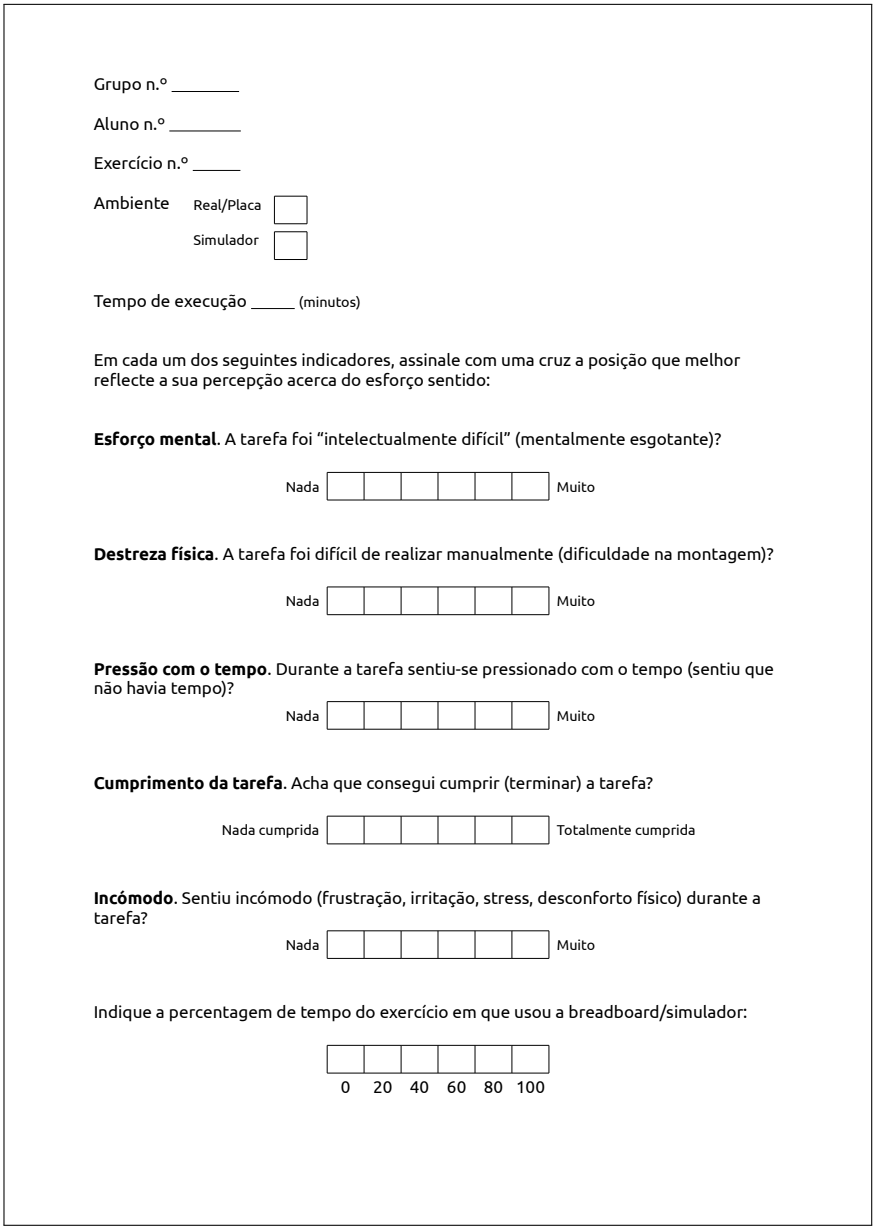
- A duração da experiência deve durar o tempo de uma aula normal (cerca de 1h30 a 2h00);
- Os exercícios a apresentar devem ser os normais para uma aula de introdução à Plataforma Arduino;
- Os exercícios são os mesmos para ambos os grupos.

Inquéritos:

- Os participantes devem responder aos inquéritos apresentados no Apêndice C;
- A resposta aos inquéritos é sempre individual;
- Os inquéritos devem ser analisados segundo a metodologia NASA-TLX.

Apêndice C

Inquéritos



Grupo n.º _____

Aluno n.º _____

Exercício n.º _____

Ambiente Real/Placa ☐

Simulador ☐

Tempo de execução _____ (minutos)

Em cada um dos seguintes indicadores, assinale com uma cruz a posição que melhor reflecte a sua percepção acerca do esforço sentido:

Esforço mental. A tarefa foi "intelectualmente difícil" (mentalmente esgotante)?

Nada ☐☐☐☐☐☐ Muito

Destreza física. A tarefa foi difícil de realizar manualmente (dificuldade na montagem)?

Nada ☐☐☐☐☐☐ Muito

Pressão com o tempo. Durante a tarefa sentiu-se pressionado com o tempo (sentiu que não havia tempo)?

Nada ☐☐☐☐☐☐ Muito

Cumprimento da tarefa. Acha que consegui cumprir (terminar) a tarefa?

Nada cumprida ☐☐☐☐☐☐ Totalmente cumprida

Incómodo. Sentiu incómodo (frustração, irritação, stress, desconforto físico) durante a tarefa?

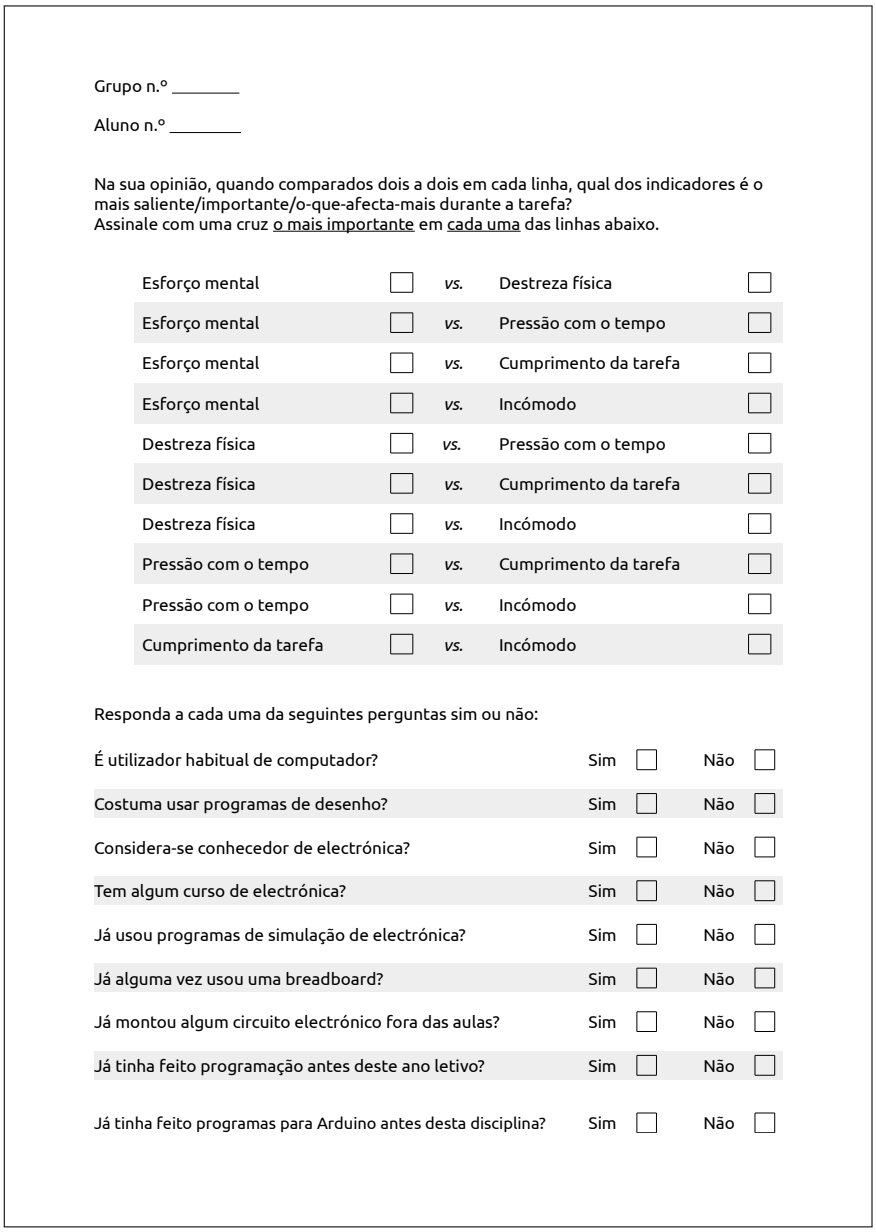
Nada ☐☐☐☐☐☐ Muito

Indique a percentagem de tempo do exercício em que usou a breadboard/simulador:

☐☐☐☐☐☐

0 20 40 60 80 100

Figura C.1: Parte I do inquérito realizado aos alunos



Grupo n.º _____

Aluno n.º _____

Na sua opinião, quando comparados dois a dois em cada linha, qual dos indicadores é o mais saliente/importante/o-que-afecta-mais durante a tarefa?
Assinale com uma cruz o mais importante em cada uma das linhas abaixo.

Esforço mental	☐	*vs.*	Destreza física	☐
Esforço mental	☐	*vs.*	Pressão com o tempo	☐
Esforço mental	☐	*vs.*	Cumprimento da tarefa	☐
Esforço mental	☐	*vs.*	Incómodo	☐
Destreza física	☐	*vs.*	Pressão com o tempo	☐
Destreza física	☐	*vs.*	Cumprimento da tarefa	☐
Destreza física	☐	*vs.*	Incómodo	☐
Pressão com o tempo	☐	*vs.*	Cumprimento da tarefa	☐
Pressão com o tempo	☐	*vs.*	Incómodo	☐
Cumprimento da tarefa	☐	*vs.*	Incómodo	☐

Responda a cada uma da seguintes perguntas sim ou não:

É utilizador habitual de computador?	Sim ☐	Não ☐
Costuma usar programas de desenho?	Sim ☐	Não ☐
Considera-se conhecedor de electrónica?	Sim ☐	Não ☐
Tem algum curso de electrónica?	Sim ☐	Não ☐
Já usou programas de simulação de electrónica?	Sim ☐	Não ☐
Já alguma vez usou uma breadboard?	Sim ☐	Não ☐
Já montou algum circuito electrónico fora das aulas?	Sim ☐	Não ☐
Já tinha feito programação antes deste ano letivo?	Sim ☐	Não ☐
Já tinha feito programas para Arduino antes desta disciplina?	Sim ☐	Não ☐

Figura C.2: Parte II do inquérito realizado aos alunos

Apêndice D

Diagramas de classes

D.1 Projecto ArduinoSimulator

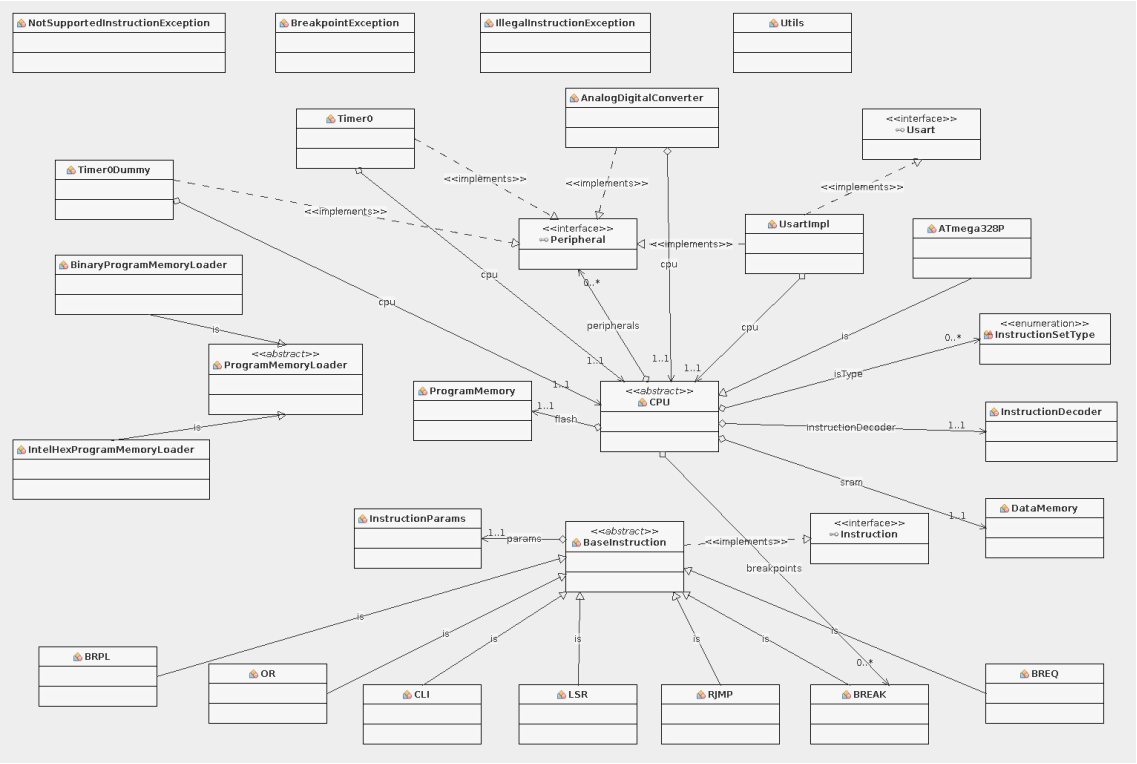


Figura D.1: Diagrama de classes simplificado do simulador do microcontrolador

[1]São apresentadas apenas 7 classes correspondentes a instruções do processador uma vez que seria impossível de apresentar a sua totalidade (123) apenas num diagrama

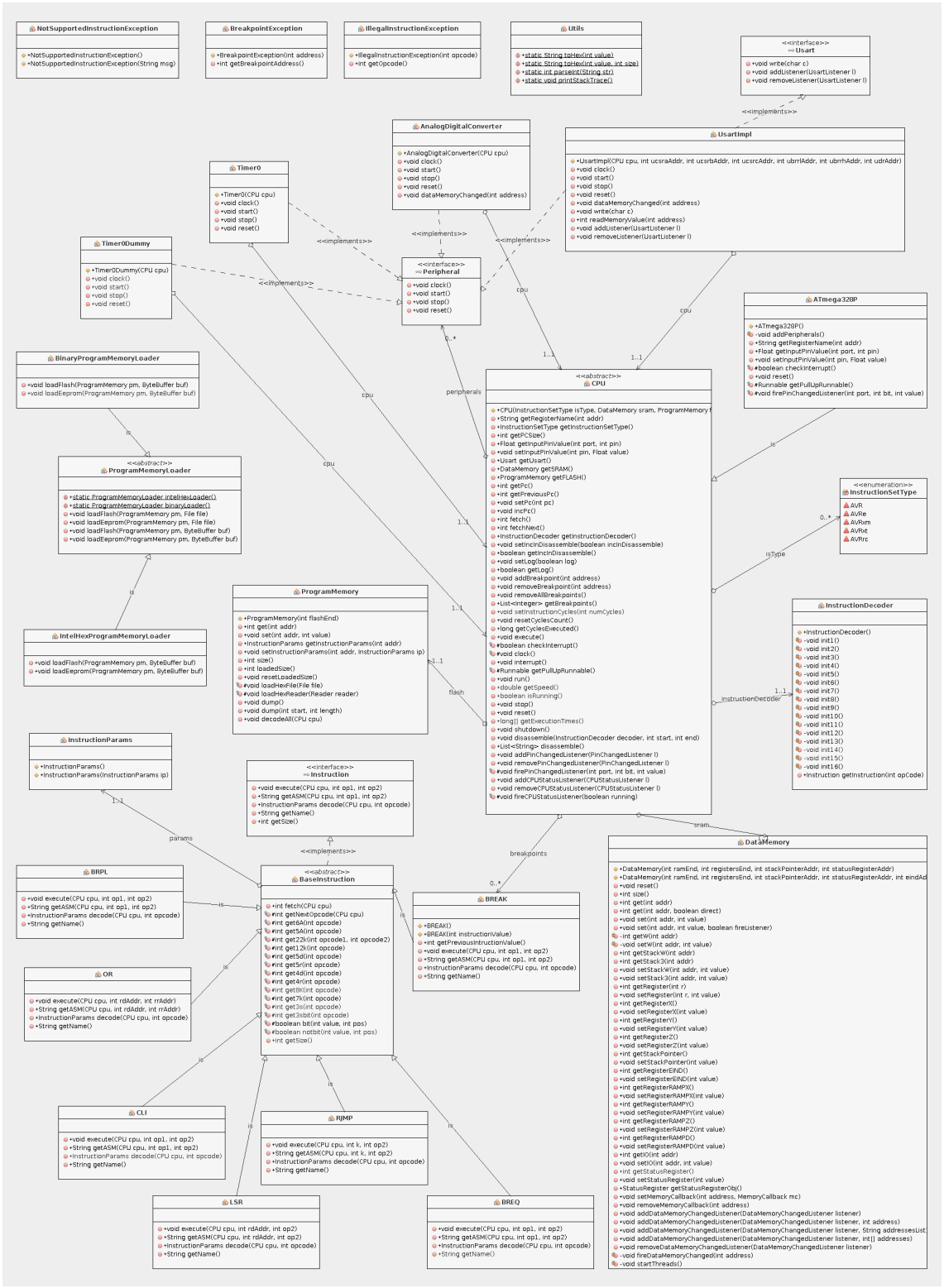


Figura D.2: Diagrama de classes do simulador do microcontrolador

[2]São apresentadas apenas 7 classes correspondentes a instruções do processador uma vez que seria impossível de apresentar a sua totalidade (123) apenas num diagrama

D.2 Projecto ArduinoSimulatorWeb

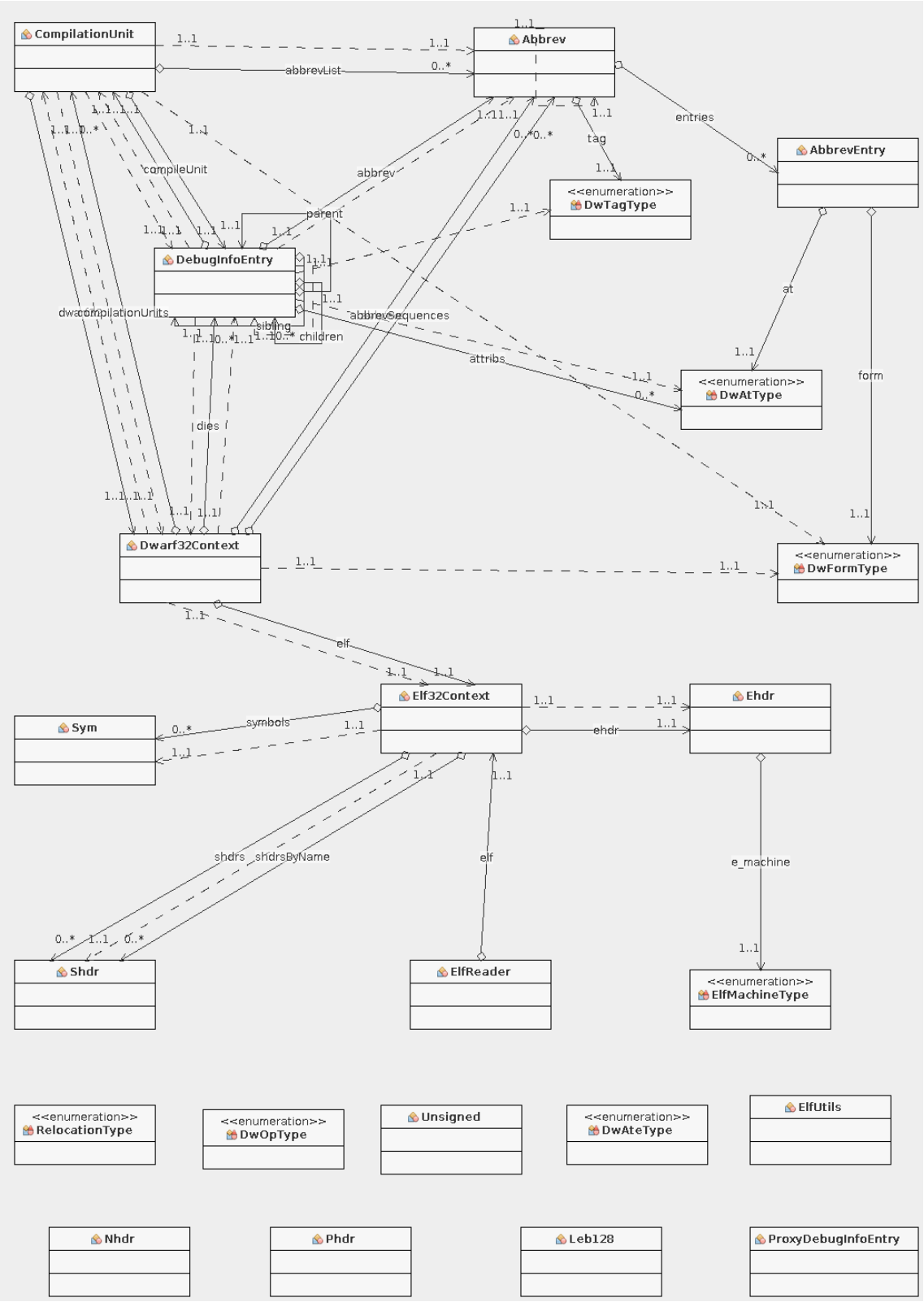


Figura D.3: Diagrama de classes simplificado das estruturas dos ficheiros ELF e DWARF

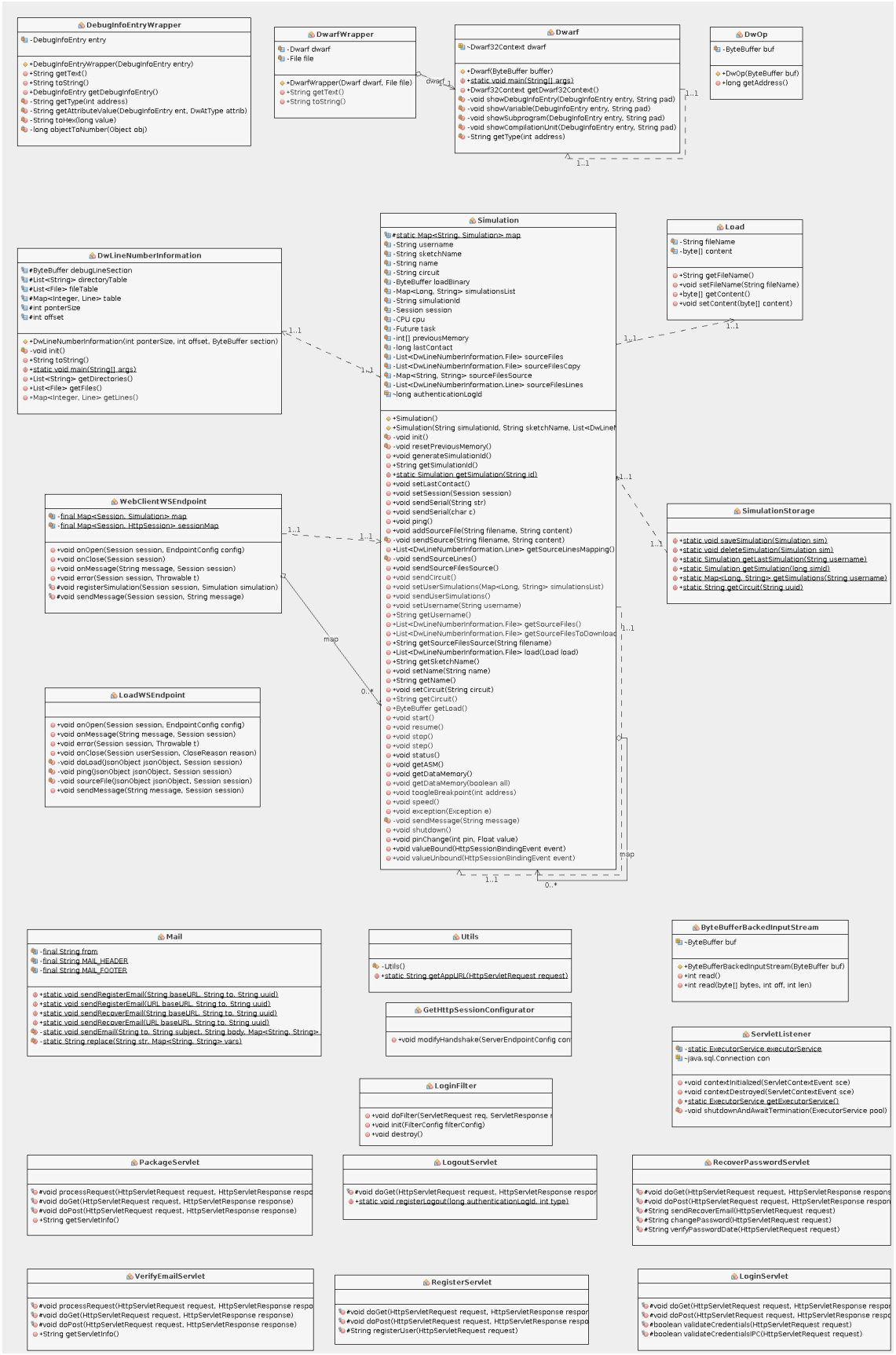


Figura D.4: Diagrama de classes da aplicação Web

D.3 Projecto ArduinoSimulatorProgrammer

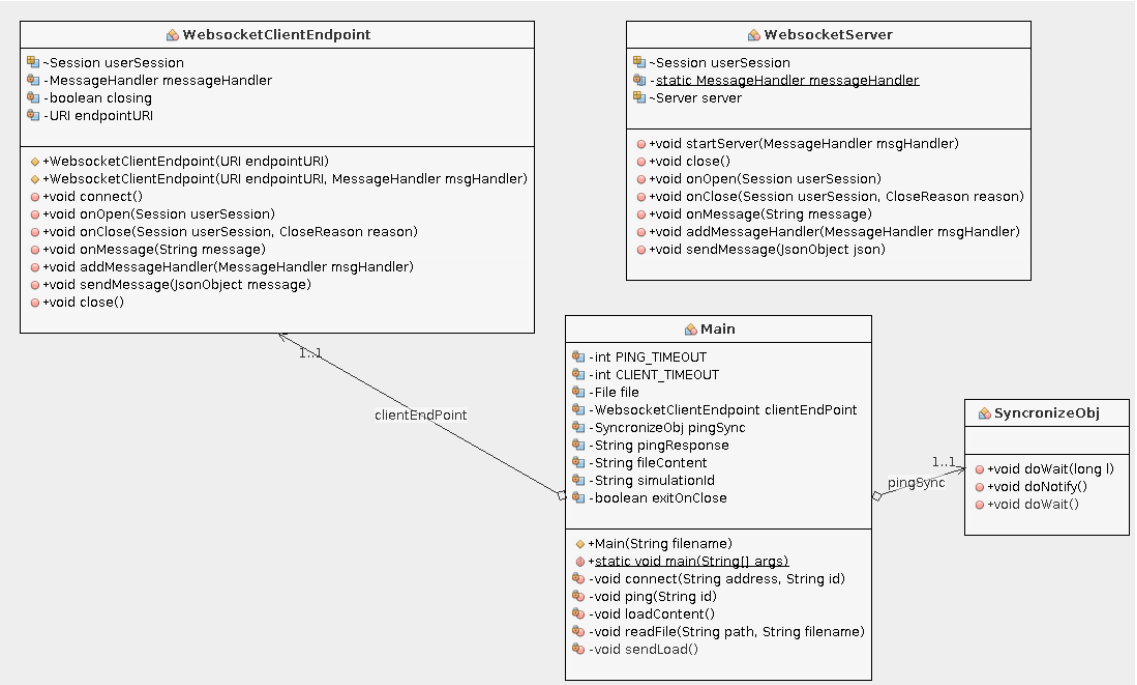


Figura D.5: Diagrama de classes da ferramenta de programação

Apêndice E

Manuais do simulador

E.1 Manual de utilização

Arduino Simulator usage instructions

In this manual we will be using the following simulator URL: `http://arduino.cognixtec.com/arduino/`. Replace in the following steps with the one you are using.

1 Install the board

The first step is to install the board in the Arduino IDE. We should go to the simulator site and copy the link “Arduino IDE package”.

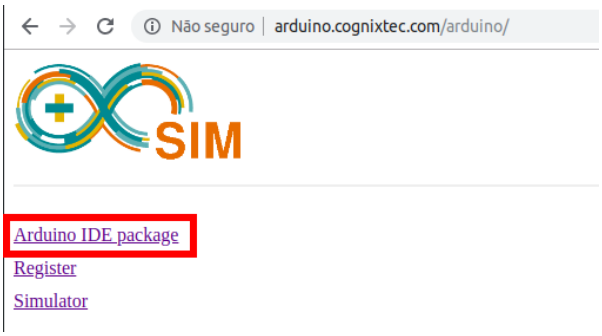


Figure 1: Arduino IDE package link

Then in the Arduino IDE menu File > Preferences we add the copied URL:

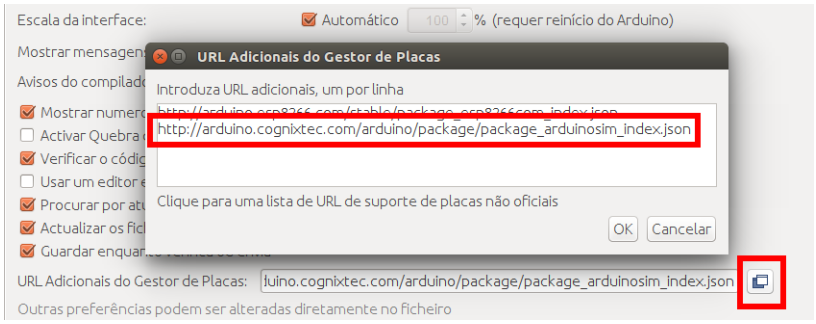


Figure 2: Arduino IDE Board Manager URL

The final step to install the board is access the menu Tools > Board: ... > Board Manager and find the **arduinosim** board.
There we press the Install button.

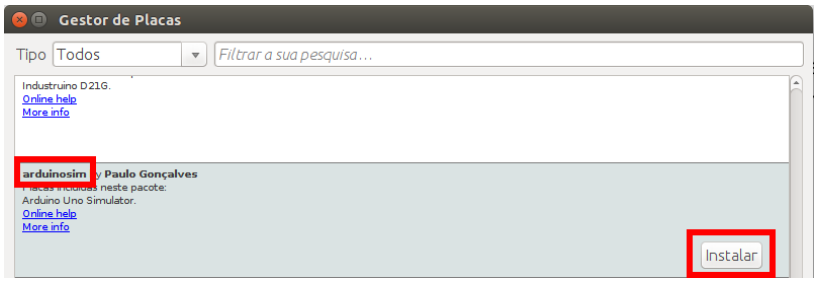


Figure 3: Arduino IDE Board Manager installation

From this moment we can select the “Arduino Uno Simulator” board.

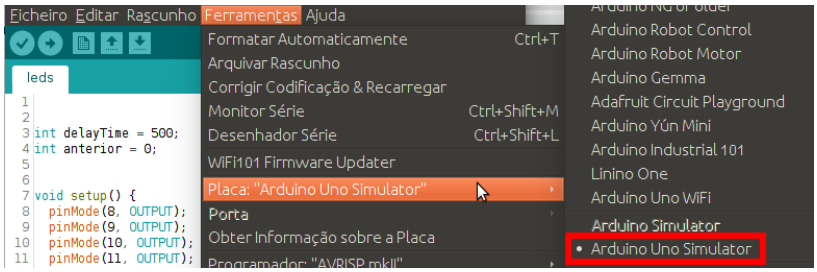


Figure 4: Arduino IDE Board Selection

2 Register

Now that the board is installed in the Arduino IDE we need to register an account in the site.

We go to the “Register” link:

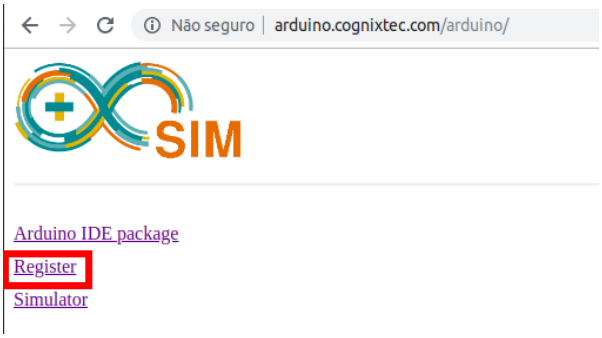


Figure 5: Register link

There we choose a username (an email), a password and we put our name.

The username must be an email that we have access because it will be sent an email to confirm the account.

Figure 6: Register

We will receive an email like the one bellow and must press the “Activate account” button.

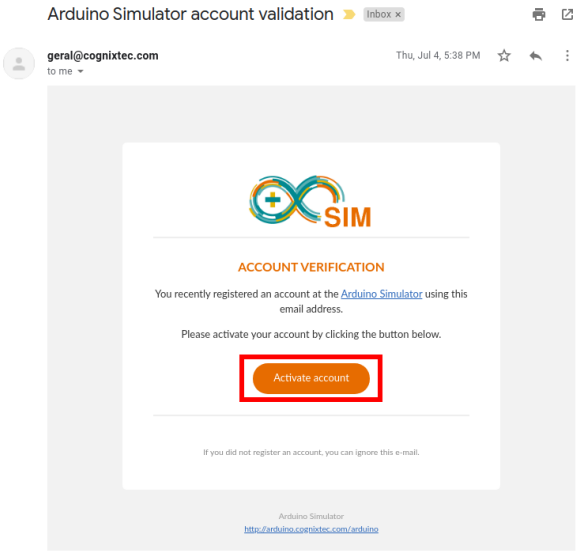


Figure 7: Account verification email

After that we can login with the email and chosen password:

Figure 8: Login

3 Simulator usage

The interface presented to us has a toolbar at the top, a component palette on the left, and a drawing area in the center. Components must be dragged from the pallet to the drawing area to be added to the circuit. Connections are made by clicking on a connection point (circles in blue, red or black) and dragging to another one. Components or connections can be selected with the mouse and deleted with the Delete key. We can only add one Arduino to the drawing area (circuit).

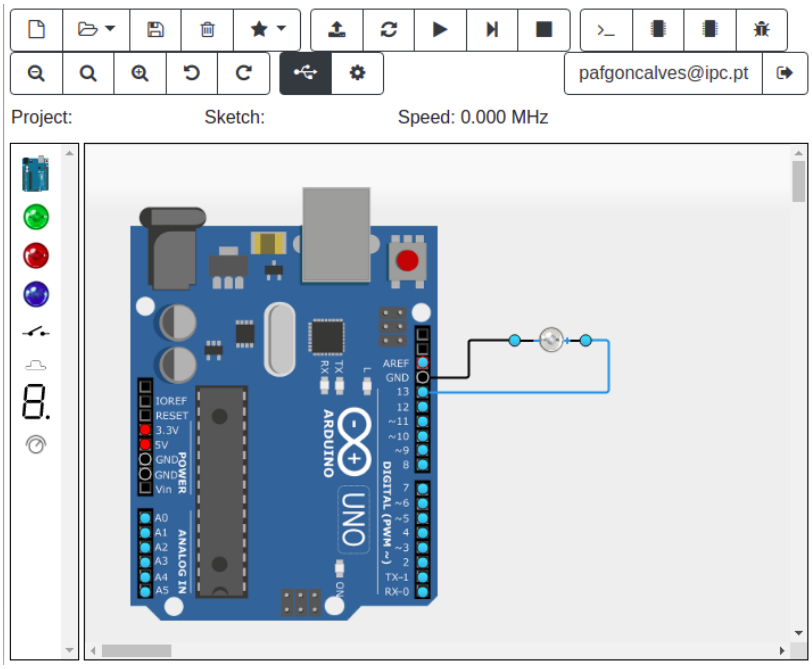


Figure 9: Simulator interface

3.1 Components

3.1.1 LEDs

Although in a real circuit there is a need to connect an LED with a resistor in series, in the simulator the LEDs are considered with an internal resistor so there are no resistors on the pallet.

The LED's have the regular polarity:

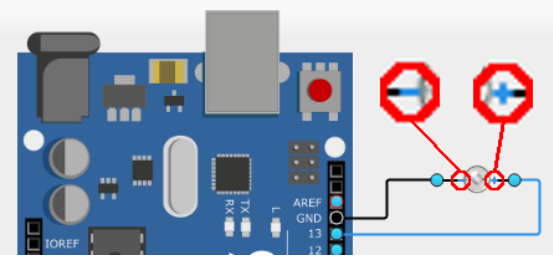


Figure 10: Simulator LEDs connection

3.1.2 Switch

The switch component can have two states, on or off. To change state click on the component.

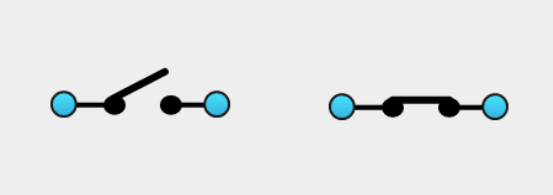


Figure 11: On-off switch

3.1.3 Push button

The push button component can also have two states, on or off. While the components is pressed with the mouse it is in the on state. For the off state simply release the mouse button.

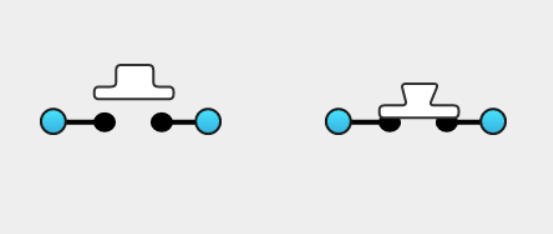


Figure 12: Push button

3.1.4 7 segment display

The 7 segment display is of common cathode. The pins are identified in the image.

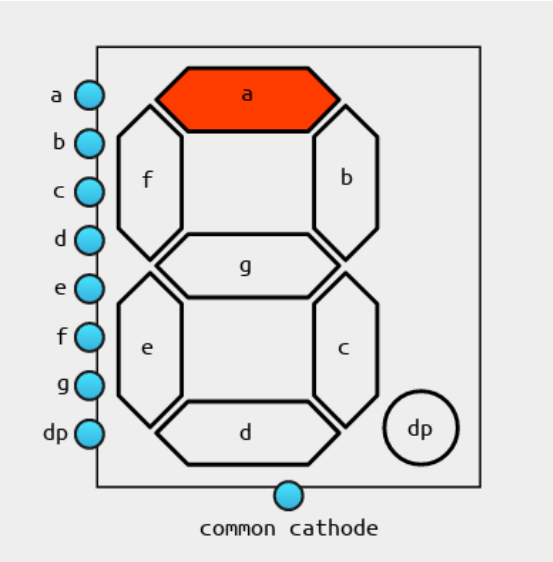


Figure 13: 7 segment display

3.1.5 Potentiometer

The potentiometer component can be used to connect to the analog inputs of the Arduino. The output is the middle connector and the green slider can be moved with the mouse.

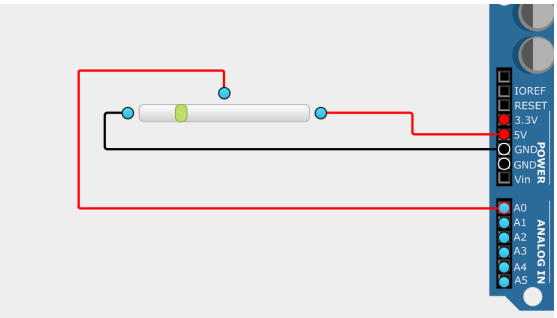


Figure 14: Potentiometer

3.2 Programming

The programming (sending of the code from the Arduino IDE to the Simulator) is done in the same way has any other Arduino. Also, the USB symbol in the Simulator interface must be active for the programming to work:

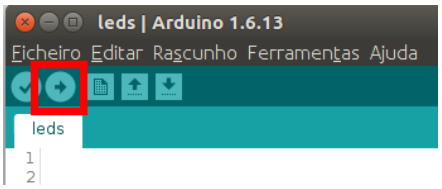


Figure 15: Arduino IDE programming button

Figure 16: Simulator programming active

It can happen that Operating System firewall gives a warning about a connection to the Internet. It's normal and must be allowed.

If there is a problem contacting the Simulator, the programmer will show a message like the one bellow. The easiest solution is to cancel and try again, but the parameters to put in the box can be found in the config area ⚙ of the Simulator interface. They are Programmer URL and Simulation ID.

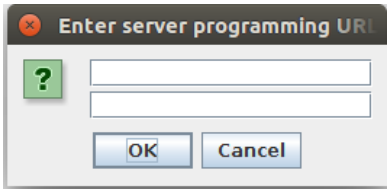


Figure 17: Programming tool parameters

Now we can use the toolbar buttons to pause, run or step-by-step the microcontroller. We can also inspect the FLASH and SRAM and add or remove breakpoints in the FLASH addresses or source code lines (in Simulator debug window). It can also be used to manage projects (create new, save, open and delete).

Figure 18: Toolbar

3.3 Project management

In the first part of the toolbar, it's possible to create a new project, open an existing one, save the current work or delete the project. When saving, if it is the first time, it will be asked a name for the project.

There is also a menu with some predefined circuits (the one with a star), that can be loaded into the drawing area.

The last button in Figure 19 allows the user to directly upload a HEX or ELF file with code for the Arduino.

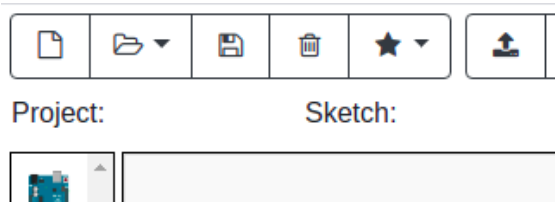


Figure 19: Project management buttons

3.4 FLASH inspection window

In the FLASH inspection window is possible to see the disassemble of the binary code loaded to the Arduino.

If using the step-by-step mode the line signalled with the green dashed line is the execution point. It's possible to click in the address (at pink) to insert or remove a breakpoint. If a breakpoint is installed the address is highlighted in orange.

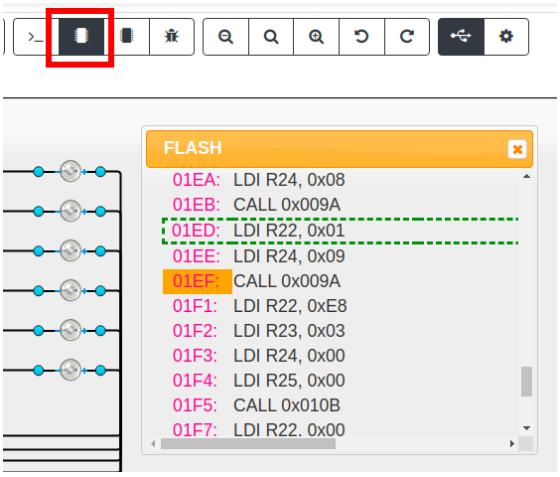


Figure 20: FLASH inspection window

3.5 SRAM inspection window

In the SRAM inspection window is possible to analyse the contents of the memory of the microcontroller. In addition to regular memory, all processor registers are mapped in memory.

When a register is written it is highlighted in orange for brief moments for the user to have an idea of the memory zones that are being manipulated.

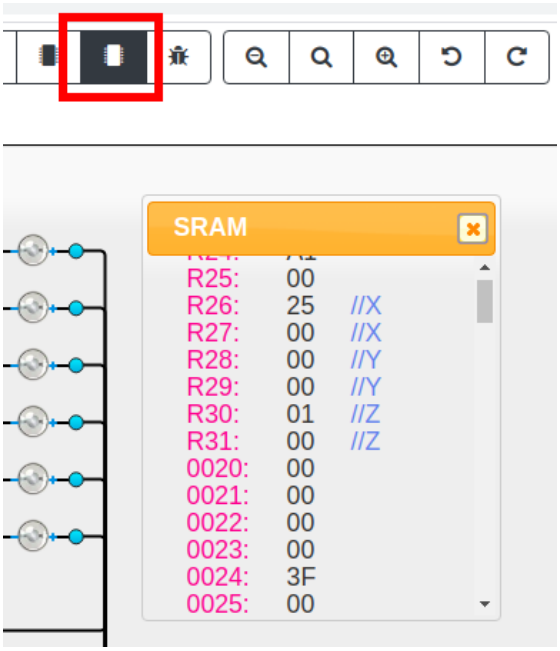


Figure 21: SRAM inspection window

3.6 Debug window

In the debug window it is possible to see the source code of the Arduino sketch if an ELF file was loaded to the simulator (the default for Arduino IDE programming).

The lines marked in yellow are the ones that correspond to some instruction in the binary program loaded in the microcontroller and can be clicked to add a breakpoint. When a breakpoint is installed the line is highlighted in orange.

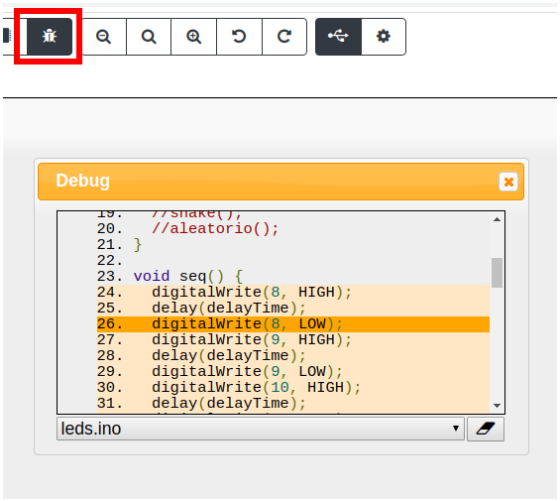


Figure 22: Debug window

3.7 Serial port monitor

The serial port 0 of the microcontroller is implemented but the input/output is redirected from the TX/RX pins to the serial port monitor. It is not possible to use these pins in the circuit for serial input/output.

It has similar functions to the Serial Monitor of the Arduino IDE. Every write to the serial port will show up in the Serial monitor of the simulator and is possible to write in the text box to send data that will be read by the microcontroller.

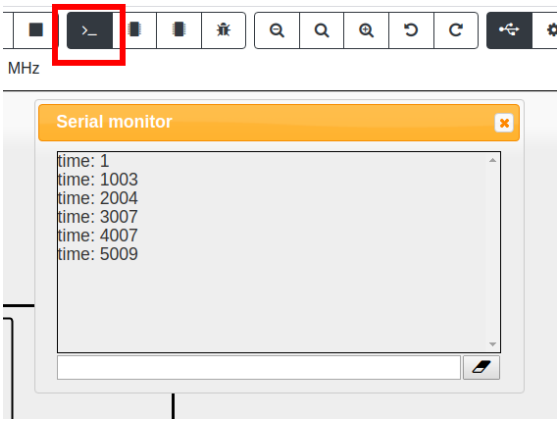


Figure 23: Serial port monitor window

4 Implemented features

The Arduino Simulator implements the following peripherals:

- **External Interrupt**
  The External Interrupts work as expected and all registers configurations are followed.
- **Pin Change Interrupt**
  The Pin Change Interrupts work as expected and all registers configurations are followed.
- **I/O Ports**
  All Port B , Port C and Port D inputs/outputs that are available on the Arduino Uno are implemented.
- **Timer 0**
  Timer 0 is implemented in a way to simulate what is expected from the Arduino API that is one interrupt every 1 ms. This allows the delay API function to work as expected and make delays work with time clock regardless of the simulator speed. Every configuration of the timer registers is ignored.
- **USART 0**
  The USART 0 is emulated and not connected to the TX/RX pins on the board. The input/output is redirected to the Serial Monitor of the simulator.

- **Analog to Digital Converter**
  The Analog to Digital Converter on the Port A (ADC0 to ADC5) is implemented. The AREF pin is also implemented.
- **Implemented Interrupts:**
  - INT 2: External Interrupt Request 0
  - INT 3: External Interrupt Request 1
  - INT 4: Pin Change Interrupt Request 0
  - INT 5: Pin Change Interrupt Request 1
  - INT 6: Pin Change Interrupt Request 2
  - INT 17: Timer/Counter0 Overflow
  - INT 19: USART Rx Complete
  - INT 20: USART Data Register Empty
  - INT 22: ADC Conversion Complete
- **Instructions**
  All AVR instructions except DES, SLEEP and SPM are implemented.

E.2 Manual de instalação

Arduino Simulator installation instructions

1 Install Java JDK

Install Java JDK from `https://www.oracle.com/java/technologies/javase-downloads.html`. Follow the instruction for your operating system.

2 Install a servlet container

We will be using Apache Tomcat as our servlet container but you can use another one.
Follow the setup guide at `https://tomcat.apache.org/tomcat-8.5-doc/setup.html` to install Apache Tomcat.

3 Deploy the simulator

You can use the Apache Tomcat manager interface at `http://server_domain_or_IP:8080/manager/html` and use the option `Select WAR file to upload`.

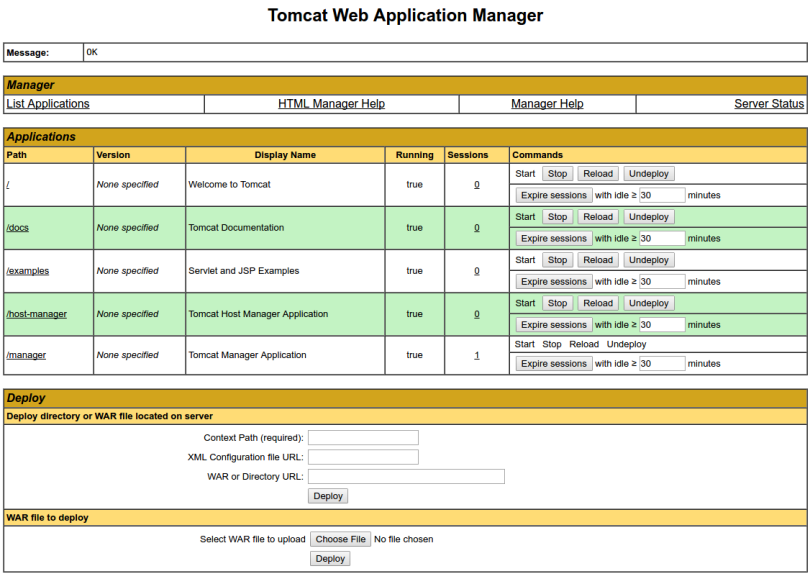


Figure 1: Apache Tomcat manager interface

To use the web interface you may need to edit the file `tomcat-users.xml` and add a user with the roles `manager-gui` and `admin-gui`.

You can also copy the WAR simulator file directly to the `webapps` folder in the Apache Tomcat installation base.
Tomcat will detect the file and automatically deploy the application.

3.1 Email settings

To register new users the applications needs to send emails.
With the default setting it is expected that a SMTP server is running on localhost that accepts connections without authentication.
To configure different email settings you should edit the file webapps/arduino/META-INF/context.xml and put your email settings in the mail/Session resource.
The available options can be consulted at `https://javaee.github.io/javamail/docs/api/overview-summary.html`.

```
<Resource name="mail/Session"
          auth="Container"
          type="javax.mail.Session"

          mail.smtp.host="smtp.example.com"
          mail.smtp.from="sender@example.com"
          mail.smtp.auth="true"
          mail.smtp.starttls.enable="true"
          mail.smtp.user="sender"
          password="password"

/>
```

Figure 2: Email resource example

4 Administer the simulator

The default username is `admin` and the default password is `123`.

There is a page at APP/admin/db.jsp, accessible only to admin user where you can execute SQL to query and update the database if needed.