Media Engineering and Technology Faculty
German University in Cairo

Virtual Arduino

Bachelor Thesis

Author:	Ahmed Sabbah
Supervisors:	Assoc. Prof. Georg Jung

Submission Date: 14 May, 2013

Media Engineering and Technology Faculty
German University in Cairo

Virtual Arduino

Bachelor Thesis

Author:	Ahmed Sabbah
Supervisors:	Assoc. Prof. Georg Jung

Submission Date:	14 May, 2013

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Ahmed Sabbah
14 May, 2013

Abstract

The use of microcontrollers is becoming more popular year by year. Nearly all devices now have microcontrollers. They even invented a soccer ball with a built-in microchip. Microcontroller boards are widely used for educational purposes in universities. they are considered an efficient way to learn and practice embedded systems programming, as it provides hands-on experience to the user. However, many university students do not have access to these boards as they are expensive. The challenge nowadays is to decrease this cost. we provide a real time simulator for a popular microcontroller board, the Arduino Uno. This simulator provides an experience as close as possible to the actual board. Using this simulator, the user can write Arduino code, compile and upload it to a simulated Arduino Uno board. He can also implement circuit simulators, add virtual hardware components and connect them to the virtual board. The simulator also provides visualization for the output that the Arduino board produces on the circuits and hardware components. Through this application, the user has the opportunity to learn most of the phases of working with microcontrollers.

Contents

Chapter 1

Introduction

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals[4]. Arduino Uno is based on the ATmega328 microcontroller. It is a popular and easy to work with microcontroller board. We provide a simulation for Arduino Uno, ATmega328 microcontroller and the experience of working with them.

Motivation

In educational context, microcontroller boards are considered a practical way for learning and practising embedded systems programming. These boards provide the user with the opportunity to learn to program and implement hardware circuits. Cost is a problem that stands against the growing use of microcontrollers mainly in educational context. This problem results in the unavailability of these boards in many universities, thus a lot of students do not have access to them. Making a simulator for these board would be a solution for this problem. Several simulators for Arduino have been implemented, but none of them can replace the the real life experience of the actual board and hardware components.

Aim of the project

The aim of this project is to provide a real time hardware simulator for a commonly used microcontroller board (Arduino Uno). This simulator is as close as possible to real experience where it covers all aspects of an embedded system. It gives the user the ability to work with hardware components, hardware interfacing, board setting, code compilation and uploading. The user will be able to write, compile and upload Arduino code using the Arduino IDE. He will be able to implement the circuitry and hardware components virtually and connect them to Arduino Uno virtual board. He can choose from a scalable library of the most common hardware components. The output of the code and hardware connections is reflected on the board and circuitry. By providing these aspects, users would not need to buy the actual board or any hardware components.

Chapter 2

Background

This chapter presents all the background information needed to work on this thesis. It also describes all the modules needed for implementing this project. The following figure display the three main modules that describe the architecture of the project.

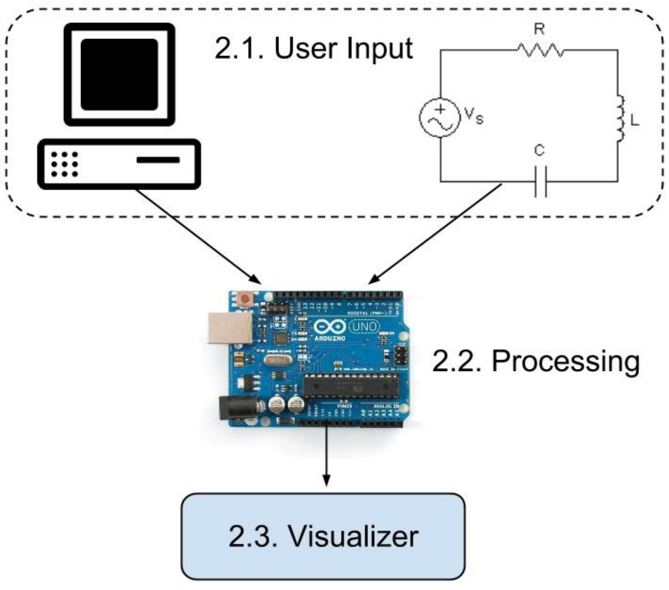


Figure 2.1: Virtual Arduino Architecture

2.1 User Input Modules

This is the first level of the program architecture. It represents all forms of input the user can give. This input can be in the form of Arduino code or circuitry.

2.1.1 Computer Module

In this module the user writes arduino code and compile it through the Arduino IDE. Instead of uploading the code to the actual Arduino, the user redirects the output to a virtual serial port from which the simulator reads. The code is sent in the form of a stream of bytes to the processing module.

2.1.2 Circuitry Module

The hardware part of working with Arduino is described in this module. The user can build circuits and add hardware components to it. He can also connect these circuits to the Arduino ports.

2.2 Processing Module

This is the second level of the architecture which processes the input it receives from the first module. After processing, it evaluates the voltage levels on the pins and accordingly sets the states of the hardware components. These states are then sent to the visualizer. The processing module is discussed later with more details in Chapter 3.

2.3 Output Visualizer

In this module the user will be able to visualize the output of the code on the hardware components and Arduino board.

2.3.1 Success Scenario Visualizer

This part simulates the scenario when the code is compiled and uploaded and there are no hardware failures. The user can modify the sensors input and accordingly the output changes.

2.3.2 Hardware Failure Visualizer

This is the case where failures are caused by hardware not the user. The user will be able to debug the circuits using a virtual avo-meter.

Chapter 3

Implementation

This module describes the core implementation of Arduino Uno. It is based on the ATmega328 microcontroller. The following is a block diagram that displays its components.

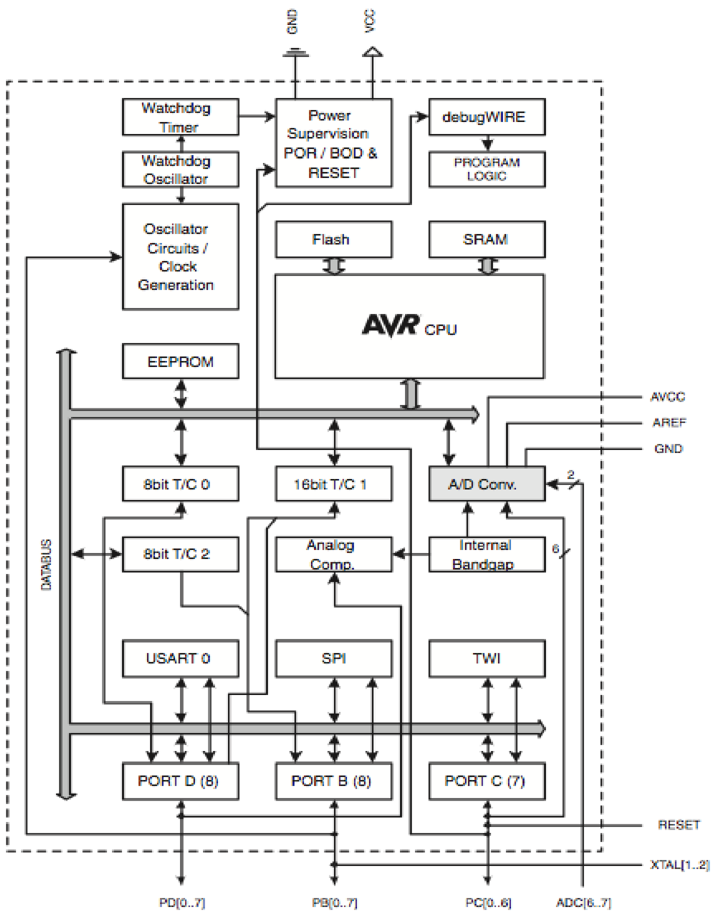


Figure 3.1: ATmega block diagram [1]

The core implementation is divided into three main sections.

3.1 Memory and registers

In ATmega328, memory is divided into 3 main parts.

3.1.1 Flash memory

It is a non-volatile read only memory of 32 KB addressed by 15-bit addresses, 0.5 KB of them are used by bootloader. It is used for storing the program.

3.1.2 SRAM (data memory)

It is a volatile memory of 2 KB. It is divided into 32 registers, 64 I/O registers, 160 external I/O registers and internal SRAM. See Figure 3.2.

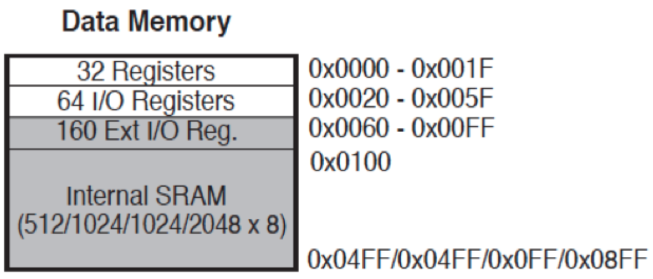
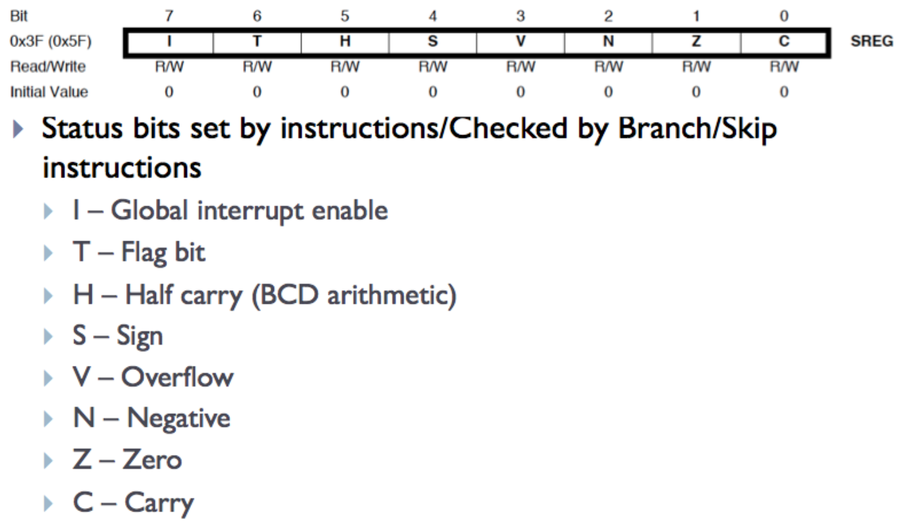


Figure 3.2: SRAM [3]

The following are special registers saved in the SRAM

Program Status Register(PSR)

It is a register for storing some status flags resulting from ALU operations. See Figure 3.3.



Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Status bits set by instructions/Checked by Branch/Skip instructions
  - I – Global interrupt enable
  - T – Flag bit
  - H – Half carry (BCD arithmetic)
  - S – Sign
  - V – Overflow
  - N – Negative
  - Z – Zero
  - C – Carry

Figure 3.3: Program Status Register [3]

Stack Pointer Register(SPR)

It is a special register in I/O space that stores the address of the last program request in a stack.

RAMPX, RAMPY, RAMPZ

Registers concatenated with the X-, Y-, and Z-registers enabling indirect addressing of the whole data space on MCUs with more than 64K bytes data space, and constant data fetch on MCUs with more than 64K bytes program space.

RAMPD

Register concatenated with the Z-register enabling direct addressing of the whole data space on MCUs with more than 64K bytes data space.

EIND

Register concatenated with the Z-register enabling indirect jump and call to the whole program space on MCUs with more than 64K words (128K bytes) program space.

3.1.3 EEPROM

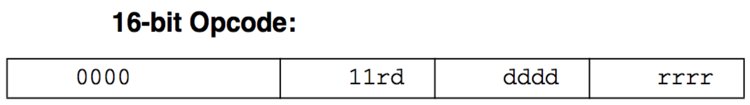
It is a long term data memory of 1 KB.

3.2 Reading program process

This is the process of receiving bytes of code and saving their values in the flash memory to be ready for execution.

3.3 Program execution

This section describes the process of executing the program saved in the flash memory. ATmega328 is based on the 8-bit AVR Instruction Set. An instruction can be either 16 or 32 bits. The application reads two bytes to form a 16 bit instruction (most significant bits first). For every instruction the opcode is matched and executed. See Figure 3.4 for the ADD instruction opcode.



16-bit Opcode:

0000	11rd	dddd	rrrr

Figure 3.4: ADD instruction opcode [2]

To determine the opcode, bitwise operations are performed on the instruction. Opcodes for all instructions and their operation are found in the AVR Instruction Set documentation. After matching with an opcode, bitwise operations are performed to extract the operands from the instruction. Then comes the next part of executing the matched instruction according to the opcode. Instruction might match with a 32 bit opcode which requires reading the next two bytes to extract the operand.

Appendix

Appendix A

Lists

List of Figures

Bibliography

[1] Atmel corporation. www.atmel.com/Images/doc8161.pdf.

[2] Atmel, avr instruction set. http://www.atmel.com/images/doc0856.pdf.

[3] University of washington computer science engineering, cse p567. http://www.cs.washington.edu/education/courses/csep567/10wi/lectures/Lecture6.pdf.

[4] Microcontroller - wikipedia. http://en.wikipedia.org/wiki/Microcontroller.