

WYDAWNICTWO  
ATNEL

AVR Microcontrollers  
C – Programming  
Basics

Mirosław Kardaś

Szczecin 2013

*To my wife - Kasia*

The book is intended for electronics practitioners and hobbyists who want to quickly learn C-language tailored for AVR microcontrollers and how to write programs based on interesting examples. C is a high-level language with unlimited possibilities that also allows you to easily and conveniently integrate with assembler — machine language. In addition, the architecture of the AVR microcontrollers, which belong to two families, ATmega and ATtiny, has been described in this book in an accessible manner, as have their capabilities.

The material is divided into three parts: issues related to the design of microcontrollers, the basics of the language, and exercises with source code, comments and descriptions.

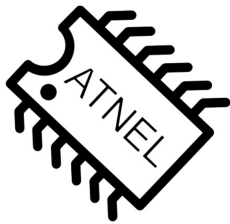
Graphic Design: Mirosław Kardaś

Cover design: Karolina Kardaś

Correction Release II: Krystyna Pawlikowska

Translated by: Michał Jezierski and Krzysztof Szczepaniak, Lingua Lab, [www.lingualab.pl](http://www.lingualab.pl).

Edited by: Brien Barnett, Lingua Lab, [www.lingualab.pl](http://www.lingualab.pl).



Wydawnictwo Atmel

70-893 Szczecin

ul. Uczniowska 5A

Tel: 91 463 56 83

Fax: 91 882 10 99

E-mail: [biuro@atmel.pl](mailto:biuro@atmel.pl)

[www.atmel.pl](http://www.atmel.pl)

Second edition revised and supplemented

ISBN 978-83-931797-3-2

© Copyright by Wydawnictwo Atmel

Szczecin 2014

All trademarks contained herein are registered trademarks or trademarks of their respective owners. The author and publisher have made every effort to provide complete and accurate information. They do not take any responsibility for its use or the possible infringement of patent rights or copyrights. The author and publisher do not assume any liability for damages resulting from the use of the information contained in the book. All rights reserved. The unauthorised reproduction in whole or in part of this publication in any form is prohibited. Making copies of or extracts from the book by photocopying or photographing, as well as copying the book or DVD to optical, magnetic or electronic media or publishing on unauthorized websites will violate the copyright of this publication.

## Table of Contents

Preface .....	10
Introduction .....	12
1 Getting Started .....	14
1.1. First Empty C Program .....	14
1.2. From the program to the CPU .....	16
1.2.1. Compilation .....	16
1.2.2. Environment .....	18
1.2.3. Hardware Programmer .....	19
1.2.4. Programming the Processor .....	21
1.2.5. The Hardware Platform .....	22
2 AVR Processors .....	25
2.1. General Information .....	25
2.2. ISP Programming .....	28
2.3. Methods for timing the processors .....	31
2.3.1. Internal oscillator .....	32
2.3.2. The external quartz-crystal resonator .....	33
2.3.3. External RC oscillator .....	33
2.3.4. External generator .....	34
2.4. Power-related topics .....	35
2.5. AVR microcontroller reset system .....	37
2.6. Internal AVR processor modules .....	37
2.6.1. FLASH, RAM, EEPROM memory .....	37
2.6.2. Interrupts .....	42
2.6.3. Hardware timers .....	45

2.6.3.1.	Basic modes of operation of timers.....	48
2.6.3.1.1	The normal counter mode .....	48
2.6.3.1.2	CTC mode – one of the most important ones.....	50
2.6.3.1.3	PWM mode .....	52
2.6.4.	ADC .....	57
2.6.5.	Analogue comparator module .....	59
2.6.6.	UART/USART module (or RS232) .....	60
2.6.7.	SPI module.....	61
2.6.8.	TWI (or I2C) module .....	62
2.6.9.	Watchdog.....	62
2.6.10.	Power-saving modes.....	63
2.6.11.	FUSE BITS (AVR configuration settings).....	64
2.6.12.	LOCK BITS (AVR security) .....	65
2.6.13.	Bootloader – incredible capabilities.....	66
3.	Basics of C.....	69
3.1.	General Topics.....	69
3.1.1.	Comments .....	69
3.1.2.	Definition and Declaration.....	70
3.1.3.	Boolean Expressions (Conditions) .....	71
3.2.	The Most Important Instructions.....	72
3.2.1.	If, Else Conditional Clause .....	72
3.2.2.	While Loop .....	75
3.2.3.	Do... While Loop .....	76
3.2.4.	For Loop .....	77
3.2.5.	Break Statement.....	79
3.2.6.	Switch Statement .....	79
3.2.7.	Continue Statement .....	82

3.2.8.	Curly Braces.....	82
3.2.9.	Goto Statement .....	83
3.3.	Types.....	84
3.3.1.	Classification of C Types.....	86
3.3.1.1.	Derived Types .....	89
3.3.1.2.	Variables Scope.....	91
3.3.1.3.	Void Type.....	93
3.3.1.4.	Qualifier Const .....	94
3.3.1.5.	Volatile Qualifier .....	95
3.3.1.6.	Register Declaration.....	97
3.3.1.7.	Typedef Declaration.....	97
3.3.1.8.	Enumeration Types .....	99
3.3.2.	Constants in C.....	102
3.3.2.1.	Integer Constants.....	102
3.3.2.2.	Floating Constants .....	103
3.3.2.3.	Character Constants.....	104
3.3.2.4.	Text Constants, Strings.....	105
3.4.	Operators .....	107
3.4.1.	Arithmetic.....	107
3.4.1.1.	Modulus, or %.....	107
3.4.1.2.	Increment and Decrement ++ -- .....	109
3.4.1.3.	Assignment Operator = .....	111
3.4.2.	Logical Operators.....	111
3.4.2.1.	Relational Operators.....	111
3.4.2.2.	Logical Sum    and Logical Product &&.....	113
3.4.2.3.	Negation - Exclamation Mark !.....	114
3.4.2.4.	Bitwise Operators .....	114

3.4.3.	Other Assignment Operators .....	121
3.4.4.	Address Operator & .....	122
3.4.5.	Conditional Expression ? : .....	123
3.4.6.	Sizeof( ) Operator .....	124
3.4.7.	Precedence of Operators .....	125
3.4.8.	Cast Operators.....	127
3.5.	Functions*** .....	128
3.5.1.	The Function Result - How does it Work? .....	132
3.5.2.	Stack - Harnessing the 'Monster' .....	134
3.5.3.	Passing Arguments by Value .....	136
3.5.4.	Inline Functions .....	139
3.5.5.	Scope of Names .....	146
3.5.5.1.	Global Scope .....	146
3.5.5.2.	Local Scope and Automatic Variables.....	147
3.5.5.3.	Static Variables and Functions .....	148
3.5.6.	Functions in Different Project Files .....	150
3.6.	Preprocessor .....	158
3.6.1.	#define Directive .....	158
3.6.2.	Macro Definitions .....	160
3.6.3.	#undef Directive .....	162
3.6.4.	## Operator - Concatenation of Names .....	162
3.6.5.	# Operator - Conversion to String .....	162
3.6.6.	Conditional Compilation Directives.....	163
3.6.7.	#ifdef and #ifndef Directives.....	166
3.6.8.	#error and Other Directives .....	167
3.6.9.	#include Directive .....	167
3.7.	Arrays.....	168

3.7.1.	Multidimensional Arrays.....	172
3.7.2.	Array as Function Argument .....	172
3.7.3.	Character Arrays.....	175
3.8.	Pointers.....	180
3.9.	Structures, Unions, Bit Fields.....	195
3.9.1.	Structures.....	195
3.9.2.	Unions .....	198
3.9.3.	Combination of Structure and Union.....	200
3.9.4.	Bit Fields.....	203
4	Workshops – practical exercises .....	205
4.1	Pins & ports – setting of directions.....	205
4.2.	Preparing the processor .....	211
4.3.	Blinking LED.....	213
4.4.	Handling the micro-switch type keys.....	216
4.5.	LED multiplexing – interrupts .....	223
4.6.	LCD display (hd44780) .....	248
4.7.	PWM control (colour RGB LED).....	274
4.8.	Voltage measurement using ADC .....	290
4.8.1.	Analogue keyboard.....	304
4.8.2.	Differential voltage measurement – ammeter.....	305
4.9.	RS232/RS485 Communication .....	320
4.9.1.	Initialisation & calibration.....	320
4.9.2.	UART, interrupts & circular buffer .....	331
4.10.	Reading from – writing to the I2C bus (RTC, EEPROM) .....	346
4.10.1.	RTC – hardware I2C support .....	347
4.10.2.	Programmatic implementation of I2C.....	357
4.10.3.	EEPROM – I2C.....	362

4.11.	SPI module .....	364
4.11.1.	SPI hardware support .....	365
4.11.2.	SPI software support .....	373
4.12.	1Wire bus.....	374
4.13.	Receiving RC5 infrared codes .....	385
4.14.	Controlling DC motors.....	398
4.15.	Unipolar stepper motor.....	403
4.16.	Bipolar stepper motor.....	412
4.17.	Reading/writing the SD memory card (FAT) .....	417
4.17.1.	FatFS.....	420
4.17.2.	PetitFS .....	440
5.	Fusebits – MkAvrCalculator .....	451
5.1.	Fusebits, Lockbits .....	451
5.2.	MkAvrCalculator.....	456
6.	Bootloader.....	467
7	Projects.....	471
7.1	Infrared Remote Control .....	471
7.2.	Bluetooth module (BTM-112/222) .....	481
7.3.	Dimmer - stepless 230V power adjustment.....	489
7.4.	Introduction to Real-Time Systems.....	504
7.5.	AVR stack handling - TCP/IP.....	530
7.5.1.	Ethernet ENC28J60 network adapter.....	534
7.5.2.	HTTP server .....	537
7.5.3.	Device Driver - UDP protocol .....	547
7.6.	USBASP programmer .....	578
8.	Eclipse.....	580
Appendix 1.	New Atmel Toolchain Rules.....	591



Annex 2: A Few More Tips ..... 594

## Preface

The constantly growing interest in C programming as a result of programming for the AVR microcontroller family by ATMEL has meant a demand for all kinds of courses and manuals. So, I decided to write this book to help all those who wish to learn the ins and outs of this language from scratch. Its aim is to introduce the world of C in the simplest possible manner to those who so far have had no contact with programming and do not know what language to learn in order to effectively and quickly program the microcontrollers.

Why C? When the first microprocessors were designed software development was closely associated with the specific machine language of each microprocessor. This resulted in the need to write programs for specific devices. Assemblers, the lowest level languages, were based on mnemonics that replace the real machine language understandable to the numeric processor. To avoid the necessity of writing programs as a sequence of hexadecimal digits, such as "0x3A, 0x1B, 0x41, 0x05", which would cause the number 22 to be stored to the specified RAM memory cell, one could use mnemonics for such orders. Thus, this sequence of digits and letters could be replaced in the assembler with a more user-friendly command, such as "MOV BUFFER, 22". The assembler compiler translates the mnemonic into the sequence of numbers recognized by a specific microcontroller.

In short, the assembler is the lowest form of machine code understandable by man. Writing programs in pure assembler is certainly possible, and many years of experience allow us to achieve significant efficiency from programs written in this manner. In order to write the lowest-level programs well and efficiently, you have to spend many years learning, and still, writing larger programs becomes cumbersome, lengthy and requires time to test and verify because of all kinds of errors. In addition, a program written in a specific machine code for one processor is very difficult to move to another type. Sometimes it is not even possible and the program has to be written again from scratch. In such a situation, C is very helpful.

It is a general-purpose language that can be used for any microcontroller supported by a C compiler. Nowadays, there are almost no processors that cannot be programmed in C. It is even normal that manufacturers do not provide assemblers for their products, offering only the C compiler instead. With C you can quickly and easily move between various families of microcontrollers; write and test programs much faster, more

effectively and efficiently; and, create code that is much easier to learn, understand and remember.

## Introduction

Ever since I learned C, I was charmed by its capabilities, simplicity and programming logic. Today, I notice a specific approach taken by many people who, after the first attempts to self-study C, are quickly discouraged because of the supposed complexity of the language's rules. Meanwhile, the real reason is often a lack of literature describing C language based on practical examples that let you solve a large number of problems on the fly. There are a few books available on C, especially in terms of programming AVR microcontrollers, written for people who do not yet have experience with programming in general. I gained a lot of experience that went into writing this book while giving courses on AVR GCC language for AVR processors. Thus, one of the goals of this publication is to raise interest in this extremely pleasant and easy language among people who are at a crossroads and must choose which way go to be able to effectively and easily program a whole family of AVR microcontrollers.

Since practical learning of C is difficult in isolation from hardware, which in our particular case are the AVR microcontrollers, it is therefore necessary to explain the principles of operation of those processors. Most examples in this book refer to the ATmega series, but I will try to show that since we will use C the programming of the ATtiny microcontroller series is practically identical. The only differences are some limitations resulting from hardware capabilities.

With the above assumptions, this book is addressed to a wide range of readers who are looking for any information on these topics. I will try to use simple, sometimes colloquial language to discuss a more complicated issue. At first glance, the structure of the book may seem a bit chaotic because it does not describe the issues sequentially and systematically in isolation from each other. There is no introduction of the AVR microcontroller family, a description of pure C language, or chapters that deal separately with programming environments, programmers or how to physically upload programs to the microcontroller. I wrote the book in the form of a course in which I present material to participants during classes—lectures on theory are interspersed with practice—or workshops in which everybody can learn how to write and test their own programs under the supervision of the instructor. This allowed me to make smooth transitions from subject to subject. So although the material may not be in separate sections, in a logical order I try to give the information as soon as you should be able to

assimilate it. In a sense, this was proven in practice. Think of this book as a good guide helpful in crossing the jungle of digital electronics and programming.

# 1 Getting Started

At the beginning I would like to teach you how to write code for a new program in C, regardless of the target processor, system or compiler used. First, you always need to create the main program function. In C, we use functions all the time and there is no doubt that this is a huge advantage compared to other programming languages.

## 1.1. First Empty C Program

This will be a very simple but versatile program, which incidentally is idle all the time, although when the processor is idle it does not mean that it is not busy. In simple words, one can say that the program is composed of one idle process occupying 100% of the microcontroller's time. Let's see how the body of this empty program looks.

Symbolically, you can write it as follows:

```
main()  
{  
}
```

It is not possible to compile such a program. It is only a function backbone in C. Because of this generic layout, the compiler recognizes a function. You can see the name of the function, in this case `main`, as indicated by the two parentheses `()` that follow the name. The curly braces `{}` mark the beginning and end of the area in which the function code is written.

### REMEMBER

If anywhere in the code you encounter notation like `fun()`, this means that you see a function called `fun`, and there can be any name, for example: `max()`, `get()`, etc.

The compiler cannot perform a complete analysis of such code because something is still missing. Each function definition, without exception, must be more specific: the type of arguments that are passed to it, and the type of result that it returns. C also provides a special type called `void`, which means that the type of function arguments may be undefined. `Void`, as it turns out, also simply means "empty type" when declaring variables or pointers.