Faculty of Computers, Informatics and Microelectronics
Technical University of Moldova

REPORT

Laboratory work #1
Embedded Systems

Performed by:
st. gr. FAF – 141, Postica Denis

Verified by:
Senior Lecturer, Andrei Bragarenco

Chișinău 2017

Topic:

I/O port programming in AVR.

Objectives:

Introduction in AVR microcontroller programming.
Connection of standard I/O libraries to the interface.

Task:

Develop an application which would allow reading, writing using standard libraries (printf, scanf). Program application such that each second it would print the value of a counter (it counts seconds till program was started).

Overview:

In the literature discussing microprocessors, we often see the term embedded system. Microprocessors and microcontrollers are widely used in embedded system products. An embedded system is controlled by its own internal microprocessor (or microcontroller) as opposed to an external controller.

Typically, in an embedded system, the microcontroller's ROM is burned with a purpose for specific functions needed for the system. A printer is an example of an embedded system because the processor inside it performs one task only; namely, getting the data and printing it. Contrast this with a Pentium-based PC (or any x86 PC), which can be used for any number of applications such as word processor, print server, bank teller terminal, video game player, network server, or Internet terminal. A PC can also load and run software for a variety of applications.

Of course, the reason a PC can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM and lets the CPU run it. In an embedded system, typically only one application software is burned into ROM. An x86 PC contains or is connected to various embedded products such as the keyboard, printer, modem, disk controller, sound card, CD-ROM driver, mouse, and so on. Each one of these peripherals has a microcontroller inside it that performs only one task.

For example, inside every mouse a microcontroller performs the task of finding the mouse's position and sending it to the PC.

A brief history of the AVR microcontroller

The basic architecture of AVR was designed by two students of Norwegian Institute of Technology (NTH), Alf-Egil Bogen and Vegard Wollan, and then was bought and developed by Atmel in 1996.

There are many kinds of AVR microcontroller with different properties. Except for AVR32, which is a 32-bit microcontroller, AVRs are all 8-bit microprocessors, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8 bits has to be broken into 8-bit pieces to be processed by the CPU. One of the problems with the AYR microcontrollers is that they are not all 100% compatible in terms of software when going from one family to another family. To run programs written for the ATtiny25 on a ATmega64, we must recompile the program and possibly change some register locations before loading it into the ATmega64. AVRs are generally classified into four broad groups: Mega, Tiny, Special purpose, and Classic.

Mega AVR (ATmegaxxxx)

These are powerful microcontrollers with more than 120 instructions and lots of different peripheral capabilities, which can be used in different designs. See Table 1-3. Some of their characteristics are as follows:

- Program memory: 4K to 256K bytes
- Package: 28 to 100 pins
- Extensive peripheral set
- Extended instruction set: They have rich instruction sets.

Table 1-3: Some Members of the ATmega Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32, PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44, PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44, PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64, MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100, CBGA

Notes:
1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (general-purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the register space.
3. All the above chips have USART for serial data transfer.

UART

The UART (Universal Asynchronous Receiver/Transmitter), is a large scale integrated circuit which contains all the software programming necessary to fully control the serial port of a PC.

A UART chip is an electronic circuit that transmits and receives data through the serial port. It converts bytes into serial bits for transmission, and vice versa. It also generates and strips the start and stop bits appended to each character.

Once a connection is established, the UART then reconstructs a byte of data based on the receiving line. If parity is in use, the UART will also check the parity for correctness and strip the parity bit off the byte being passed to the computer.

SingleByte FlFO Buffers

The UART speed in most PCs, until recently, used 8250 or 16450 UARTS. These are singlebyte FIFO (FirstIn, FirstOut) buffers. Below is a table of how fast the computer would have to respond to the different rates. The column "Characters", is how often a character arrives at the listed speeds.

COM port (bps)	Characters
2400	4 milliseconds
9600	1 millisecond
19200	520 microseconds
38400	260 microseconds

If the computer can not retrieve the data at the above speeds, then it will be overwritten. If this occurs, then that character will be lost. The UART does not only handle serial communications, it also takes care of other tasks, like disk interrupts, keyboard interrupts, screen refresh cycles, and many other items that involve timing with your system.

So, as you can see, the UART is doing a lot of other things that can limit how much incoming data it can handle. If the system can not keep up, this is when you see block errors and UART overrun errors. If you receive these errors, it's time to start looking into a buffered UART, such as a 16550.

Tools and technologies:

Atmel Studio

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging Atmel® SMART ARM®-based and Atmel AVR® microcontroller (MCU) applications. Studio 7 supports all AVR and Atmel SMART MCUs. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to Atmel debuggers and development kits.

Additionally, Atmel Studio includes Atmel Gallery, an online apps store that allows you to extend your development environment with plug-ins developed by Atmel as well as by third-party tool and embedded software vendors. Atmel Studio 7 can also able seamlessly import your Arduino sketches as C++ projects, providing a simple transition path from Makerspace to Marketplace.

Proteus

The Proteus Design Suite is a Windows application for schematic capture, simulation, and PCB layout design. It can be purchased in many configurations, depending on the size of designs being produced and the requirements for microcontroller simulation. All PCB Design products include an autorouter and basic mixed mode SPICE simulation capabilities.

Schematic capture in the Proteus Design Suite is used for both the simulation of designs and as the design phase of a PCB layout project. It is therefore a core component and is included with all product configurations.

The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analog and digital electronics connected to it. This enables it's used in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design. It also finds use in the general hobbyist community and, since no hardware is required, is convenient to use as training or teaching tool.

AVR microcontroller I/O pins

The AVR can have from 3 to 86 pins for I/O. The number of I/O pins depends on the number of pins in the package itself. The number of pins for the AVR package goes from 8 to 100 at this time. In the case of the 8-pinAT90S2323, we have 3 pins for I/O, while in the case of the 100-pinATmegal280, we can use up to 86 pins for I/O.

In the AVR family, there are many ports for I/O operations, depending on which family member you choose. Examine Figure 1 for the ATmega32 40-pin chip. A total of 32 pins are set aside for the four ports PORTA, PORTB, PORTC, and PORTD. The rest of the pins are designated as YCC, GND, XTALI, XTAL2, RESET, AREF, AGND, and AYCC.

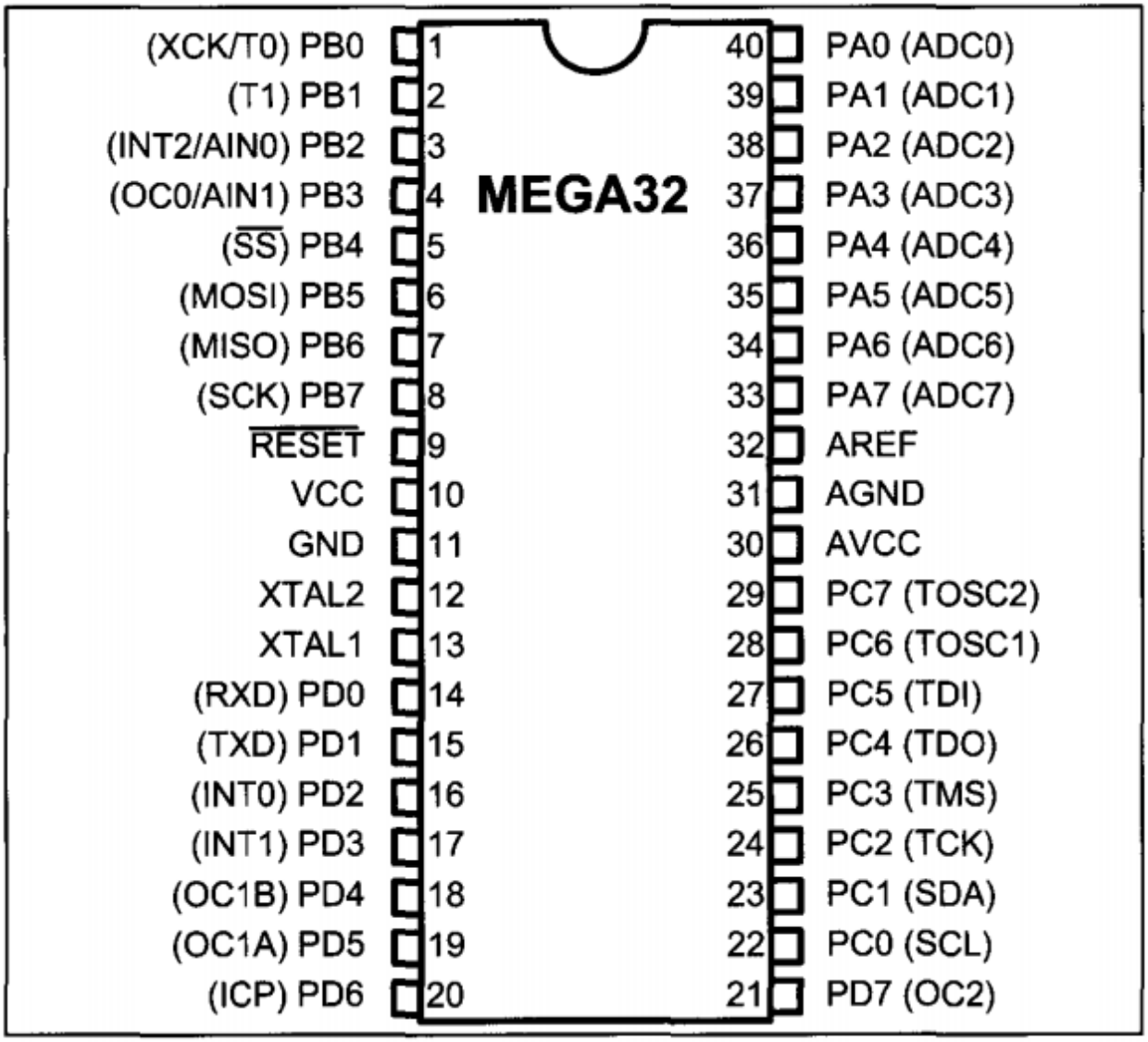

Figure 1. ATmega32 Pin Diagram.

Application development:

AVR Studio

uart_stdio.h

Header file where are declared uart_stdio functions like:

```
void uart_stdio_Init(void);
int uart_stdio_PutChar(char c, FILE *stream);
```

uart_stdio.c

The file where functions defined in header file are implemented.
First one is used to initialize uart_stdio (increases baude rate* if needed, sets UART speed and enables RX and TX).
The second one is used to output a character.

lab1.c

1. In main file initially is included uart_stdio.h to be able to use its functions. Also, library <avr/delay.h> to be able to use delay function.

2. After, the counter is defined with value 0.

3. Main starts with calling **`void uart_stdio_Init()`**, used to initialize uart_stdio.

4. Program enters the infinite loop. In the infinite loop are called sequentially:
   a. _delay_ms(1000) function, used to pause program for 1000 ms.
   b. increase the value of the counter by 1.
   c. call function printf("%d\n", count) to print the value of the counter.

Proteus ISIS

After twice building the code there is created a file with .hex extension which will be used in Proteus application scheme.
The scheme of the application can be seen in figure 2.

* The **baud rate** is the rate at which information is transferred in a communication channel. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.

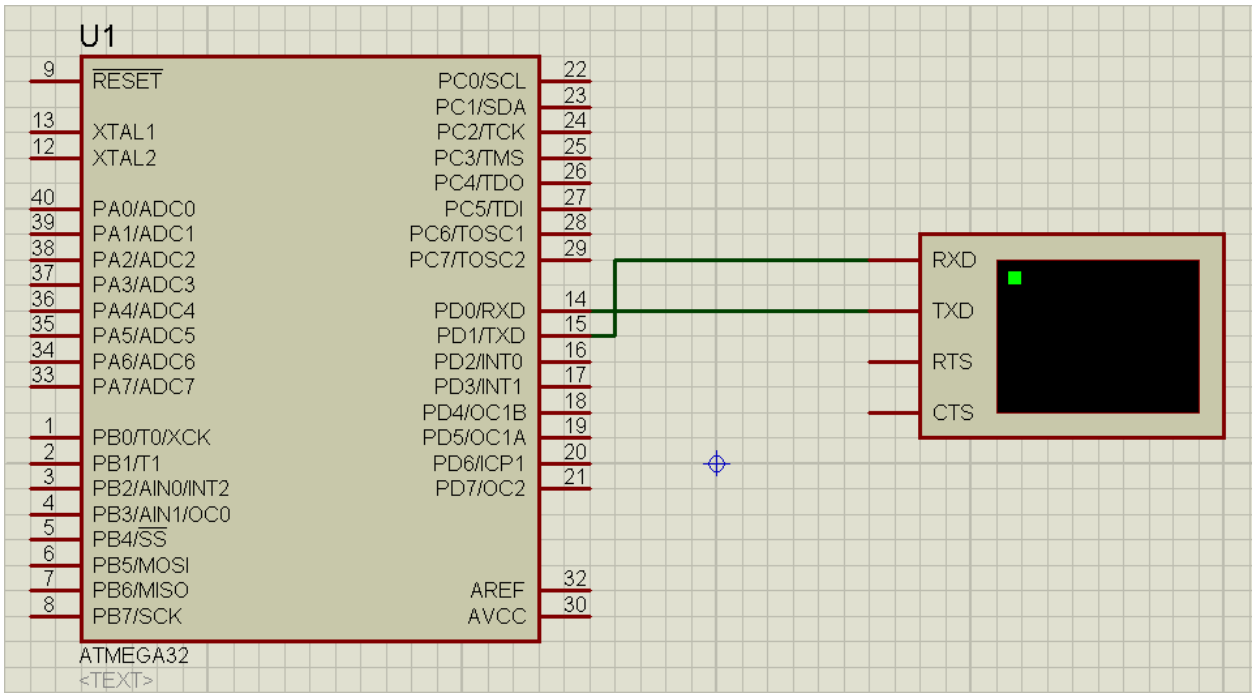

Figure 2. Scheme of the application in Proteus ISIS.

To be able to run the application we have to specify for ATmega32 the path to the hex file (see fig. 3).

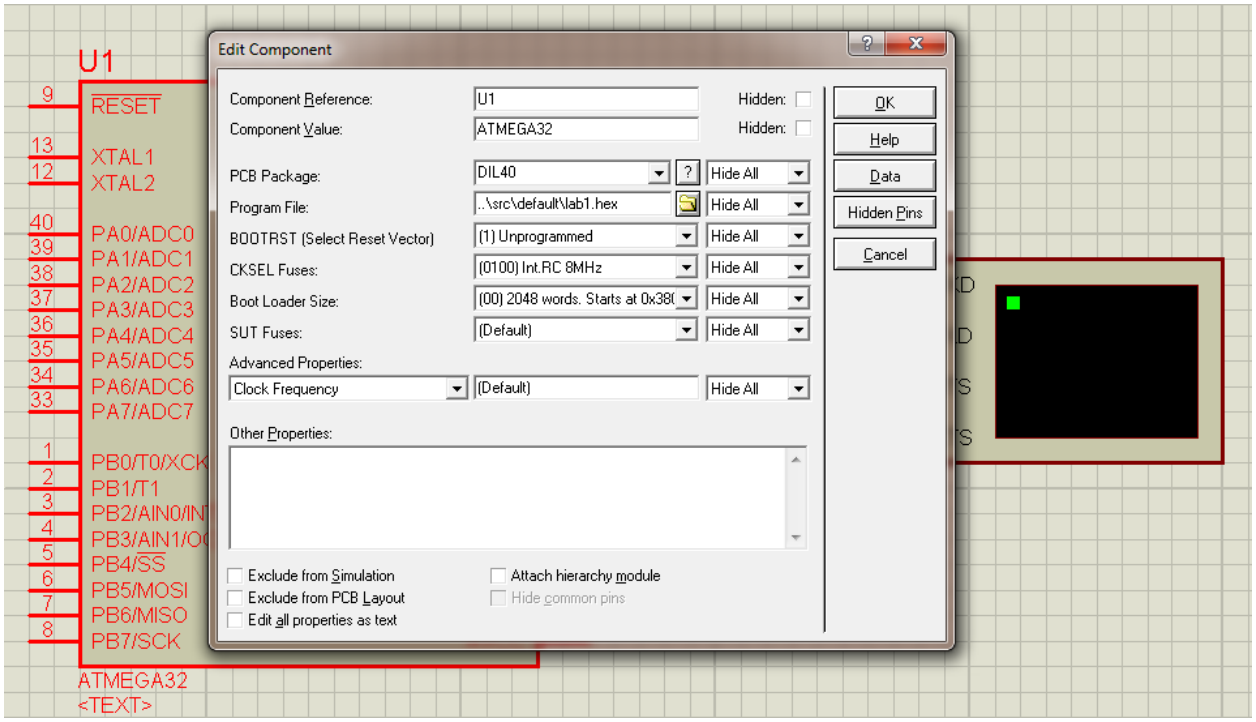

Figure 3. Path to the hex file.

When running the application we can observe that a terminal is opened and the output of counter starts (see fig. 4).

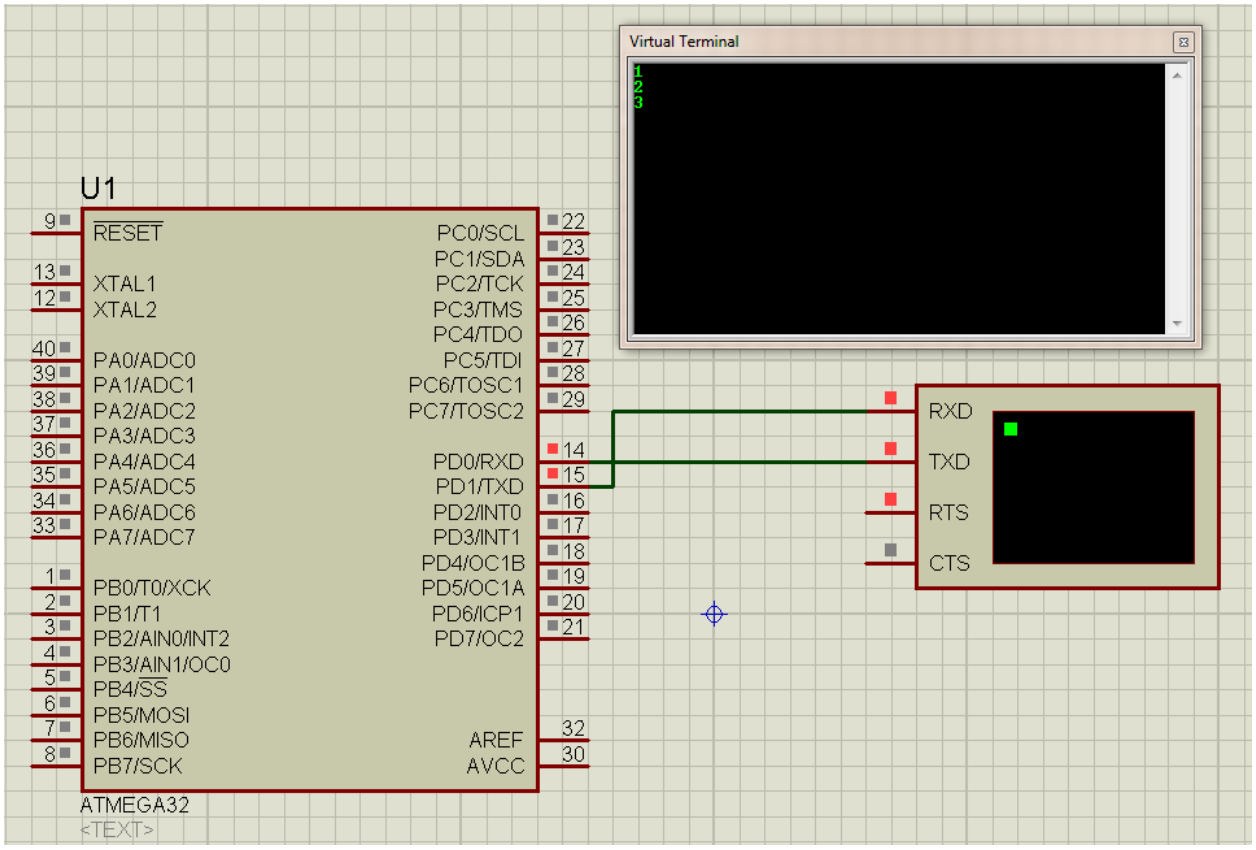

Figure 4. Application run.

Conclusions:

The laboratory work #1 represented my first experience with AVR microcontroller programming. I learned the basics of I/O programming in AVR. Also, I discovered tools like AVR Studio and Proteus, which were very useful while developing the given application.

Appendix:

uart_stdio.h

```
#ifndef _UART_STDIO_H_
#define _UART_STDIO_H

#include <stdio.h>
#include <stdint.h>
#include <stdio.h>
#include <avr/io.h>

    void uart_stdio_Init(void);
    int uart_stdio_PutChar(char c, FILE *stream);

#endif
```

uart_stdio.c

```
#include "uart_stdio.h"
#define UART_BAUD 9600

FILE uart_output = FDEV_SETUP_STREAM(uart_stdio_PutChar, NULL, _FDEV_SETUP_WRITE);

int uart_stdio_PutChar(char c, FILE *stream) {

  if (c == '\a') {                    /* checks if c is audible return character */
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')                      /* checks if c is new line */
    uart_stdio_PutChar('\r', stream); /* takes control to first position in the line */
  while(~UCSRA & (1 << UDRE));
  UDR = c;

  return 0;
}


void uart_stdio_Init(void) {
    #if F_CPU < 2000000UL && defined(U2X)
      UCSRA = _BV(U2X);                       /* improve baud rate error by using 2x clk */
      UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;/* used to set UART speed*/
    #else
      UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;/* used to set UART speed*/
    #endif
      UCSRB = _BV(TXEN) | _BV(RXEN);           /* Enable RX and TX */

      stdout = &uart_output;
}
```

lab1.c

```
#include "uart_stdio.h"
#include <avr/delay.h>

int count = 0;

void main(void) {
    uart_stdio_Init();

    while(1) {
        _delay_ms(1000);       /* pauses program for 1 second */
        count = count + 1;     /* increases the value of the counter */
        printf("%d\n", count);/* prints the value of the counter */
    }


}
```