

Ministerul Educatiei al Republicii Moldova  
Universitatea Tehnica a Moldovei  
Filiera Anglofona

# Report

Laboratory Nr.1

Embedded Systems

Performed By:

Daniel Ciobanu

Verified By:

Andrei Bragarenco

Chisinau 2017

**Topic :** Introduction to MCU. Serial Interfacing using USART.

**Task:** Write a basic program that will display every second the message „Hello World!“ in the virtual terminal. The simulation will be done in the on the circuit build in Proteus.

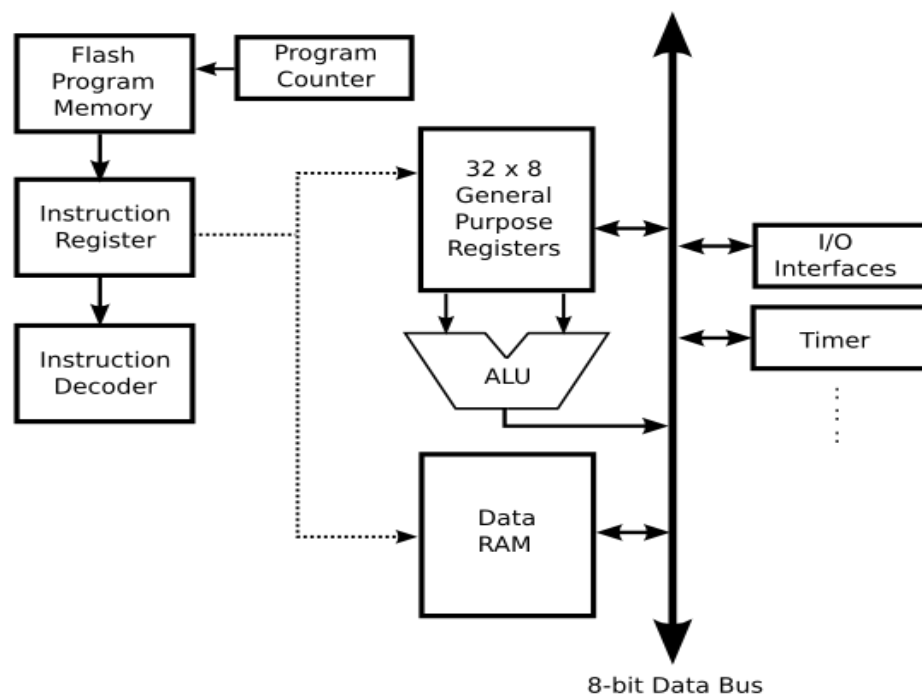
### Theory:

A **microcontroller** is a self-contained system with peripherals, memory and a processor that can be used as an embedded system. Most programmable microcontrollers that are used today are embedded in other consumer products or machinery including phones, peripherals, automobiles and household appliances for computer systems. Due to that, another name for a microcontroller is "embedded controller." Some embedded systems are more sophisticated, while others have minimal requirements for memory and programming length and a low software complexity. Input and output devices include solenoids, LCD displays, relays, switches and sensors for data like humidity, temperature or light level, amongst others.

### Applications for Microcontrollers:

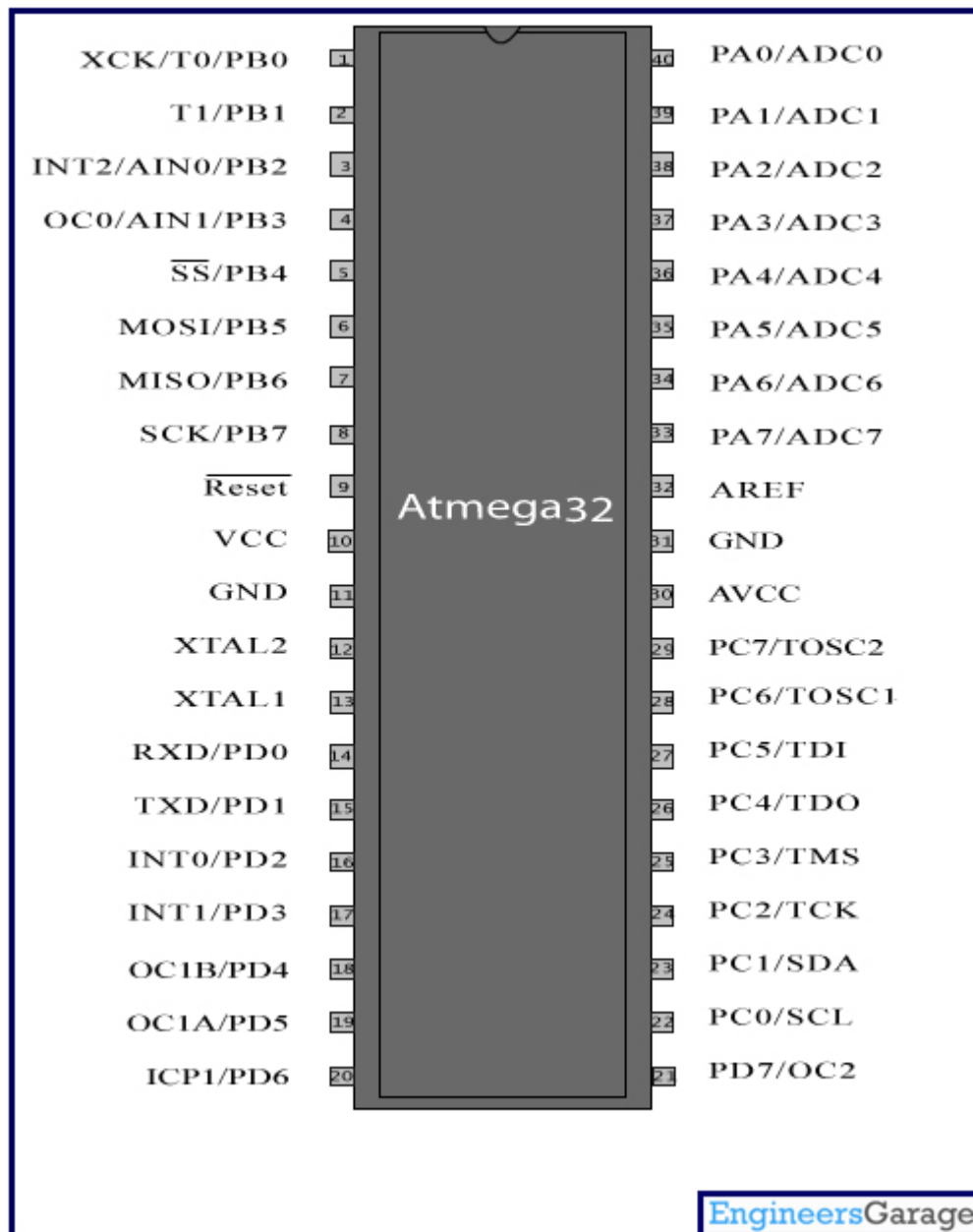
Programmable microcontrollers are designed to be used for embedded applications, unlike microprocessors that can be found in PCs. Microcontrollers are used in automatically controlled devices including power tools, toys, implantable medical devices, office machines, engine control systems, appliances, remote controls and other types of embedded systems.

### MCU Architecture:



## Atmega32 AVR microcontroller :

**ATmega32** is an 8-bit high performance microcontroller of Atmel's Mega AVR family. ATmega32 is based on enhanced RISC (Reduced Instruction Set Computing) architecture with 131 powerful instructions. Most of the instructions execute in one machine cycle. ATmega32 can work on a maximum frequency of 16MHz.



## **What is UART?**

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of a computer. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission is commonly used with modems and for non-networked communication between computers, terminals and other devices.

There are two primary forms of serial transmission: Synchronous and Asynchronous. Depending on the modes that are supported by the hardware, the name of the communication sub-system will usually include a A if it supports Asynchronous communications, and a S if it supports Synchronous communications. Both forms are described below.

Some common acronyms are:

### ***UART Universal Asynchronous Receiver/Transmitter***

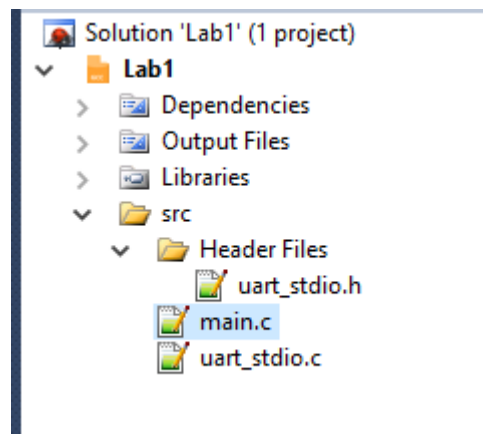
Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units.

### ***USART Universal Synchronous-Asynchronous Receiver/Transmitter***

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to “read” the next bit of the data. In most forms of serial Synchronous communication, if there is no data available at a given instant to transmit, a fill character must be sent instead so that data is always being transmitted. Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, and synchronous communication can be more costly if extra wiring and circuits are required to share a clock signal between the sender and receiver.

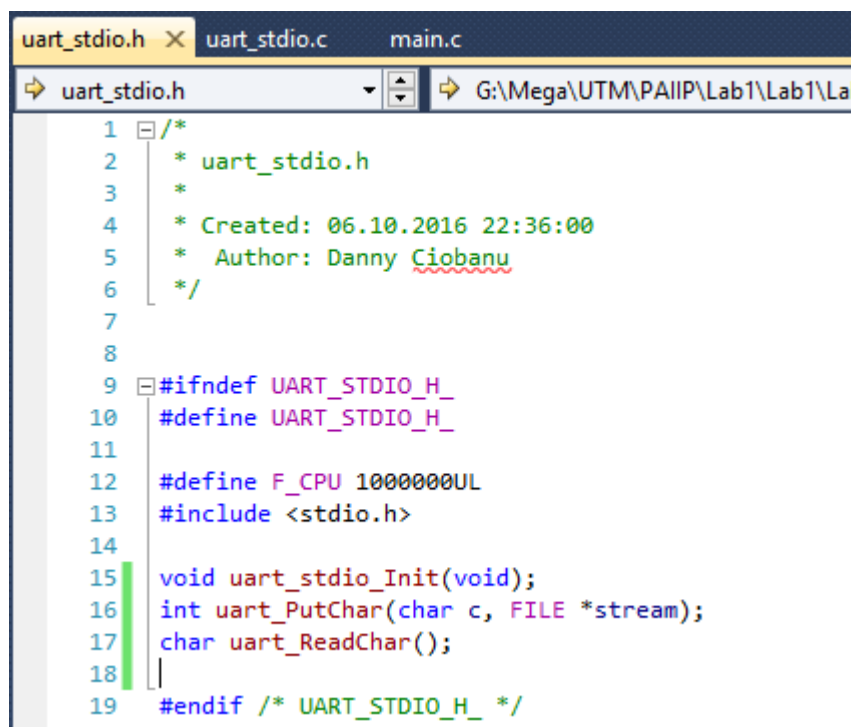
## Implementation:

The structure of the laboratory 1 is simple and looks like this:



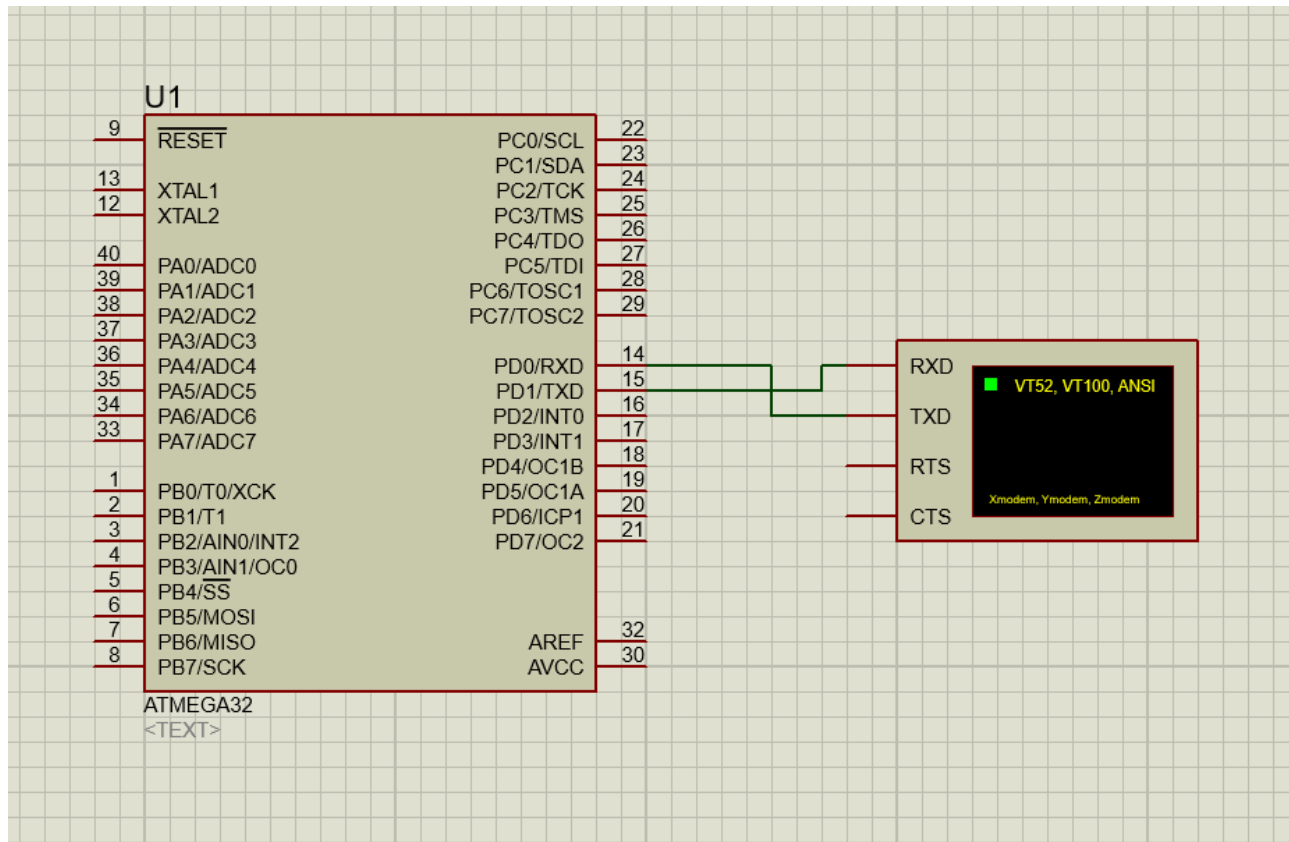
Here below i have the header of the UART driver that includes the prototypes of 3 functions:

- Initialization
- Output character
- Input character

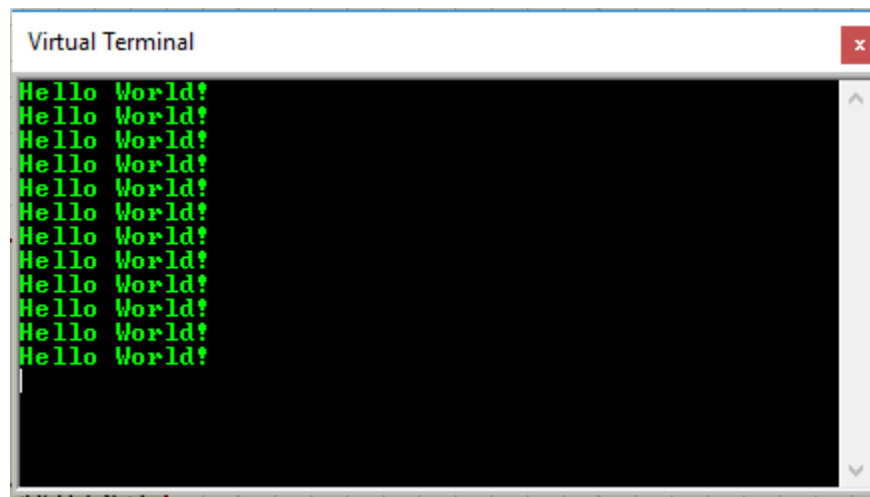
The image shows the Visual Studio Code editor with the 'uart\_stdio.h' file open. The file content is as follows:

```
1  /*
2   * uart_stdio.h
3   *
4   * Created: 06.10.2016 22:36:00
5   * Author: Danny Ciobanu
6   */
7
8
9  #ifndef UART_STDIO_H_
10 #define UART_STDIO_H_
11
12 #define F_CPU 1000000UL
13 #include <stdio.h>
14
15 void uart_stdio_Init(void);
16 int uart_PutChar(char c, FILE *stream);
17 char uart_ReadChar();
18
19 #endif /* UART_STDIO_H_ */
```

## The scheme in Proteus:



## The Result(Virtual Terminal):



## Conclusion :

This laboratory work provides us with the knowledge of the input/output drivers used by Atmega32 MCU. Also we learned the architecture and design of this MCU and gained the possibility of using more interesting feature and functions.

## Appendix:

### uart\_stdio.h

```
#ifndef UART_STDIO_H_
#define UART_STDIO_H_

#define F_CPU 1000000UL
#include <stdio.h>

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

#endif /* UART_STDIO_H_ */
```

### uart\_stdio.c

```
#include "Header Files/uart_stdio.h"

#define UART_BAUD 9600

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

FILE uart_output = FDEV_SETUP_STREAM(uart_PutChar, NULL,
_FDEV_SETUP_WRITE);
FILE uart_input = FDEV_SETUP_STREAM(NULL, uart_ReadChar,
_FDEV_SETUP_READ);

void uart_stdio_Init(void) {
    stdout = &uart_output;
    stdin = &uart_input;

    #if F_CPU < 2000000UL && defined(U2X)
        UCSRA = _BV(U2X); /* improve baud rate error by using
2x clk */
        UBRR1 = (F_CPU / (8UL * UART_BAUD)) - 1;
```

```

        #else
        UBRR1 = (F_CPU / (16UL * UART_BAUD)) - 1;
        #endif
        UCSRB = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
    }

    int uart_PutChar(char c, FILE *stream) {
        if (c == '\n')
            uart_PutChar('\r', stream);

        while (~UCSRA & (1 << UDRE));
        UDR = c;

        return 0;
    }

    char uart_ReadChar() {
        //Wait untill a data is available
        while(!(UCSRA & (1<<RXC)))
        {
            //Do nothing
        }
        //Now USART has got data from host
        //and is available is buffer

        return UDR;
    }

```

## main.c

```

#include "Header Files/uart_stdio.h"
#include <util/delay.h>

int main(void) {

    uart_stdio_Init();
    //printf("System is on");

    while(1){
        _delay_ms(500);
        printf("Hello World!\r");
    }
    return 0;
}

```