Faculty of Computers, Informatics and Microelectronics
Ministerul Educației al Republicii Moldova
Universitatea Tehnică a Moldovei

PROGRAMAREA APLICATIILOR INCORPORATE SI INDEPENDENTE DE PLATFORMA

Lucrare de laborator #1

Introducerea in programarea MicroControloarelor si implementarea comunicareii seriale UART - Transmitator/Receptor universal asincron"

A efectuat:
Mereuta Alex

A verificat:
Andrei Bragarenco

2016

1 Topic

Introduction to Micro Controller Unit programming. Implementing serial communication over UART – Universal Asynchronous Receiver/Transmitter.

2 Objectives

- Programming an MCU in ANSI C
- Studying UART serial comunicator
- Create a PCB in Proteus
- Write a program for 8 bit ATMega32 MCU

3 Task

Write a program using UART that will print to the virtual terminal the value of a variable every second. Simulate the wirtten program of a scheme build with Proteus.

4 Domain

4.1 Embedded systems

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPU's with integrated memory or peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

4.2 Microcontrollers

A microcontroller (or MCU for microcontroller unit) is a small computer on a single integrated circuit. In modern terminology, it is a System on a chip or SoC. A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals. Program memory in the form of Ferroelectric RAM, NOR flash or OTP ROM is also often included on chip, as well as a small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications consisting of various discrete chips.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

Some microcontrollers may use four-bit words and operate at frequencies as low as 4 kHz, for low power consumption (single-digit milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

4.3 Universal asynchronous receiver/transmitter

A universal asynchronous receiver/transmitter (UART /ˈjuːɑːrt/), is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling, etc.) are handled by a driver circuit external to the UART.

UARTs are commonly used in conjunction with communication standards such as TIA (formerly EIA) RS-232, RS-422 or RS-485. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers. A dual UART, or DUART, combines two UARTs into a single chip. Similarly, a quadruple UART or QUART, combines four UARTs into one package, such as the NXP 28L194. An octal UART or OCTART combines eight UARTs into one package, such as the Exar XR16L788 or the NXP SCC2698.

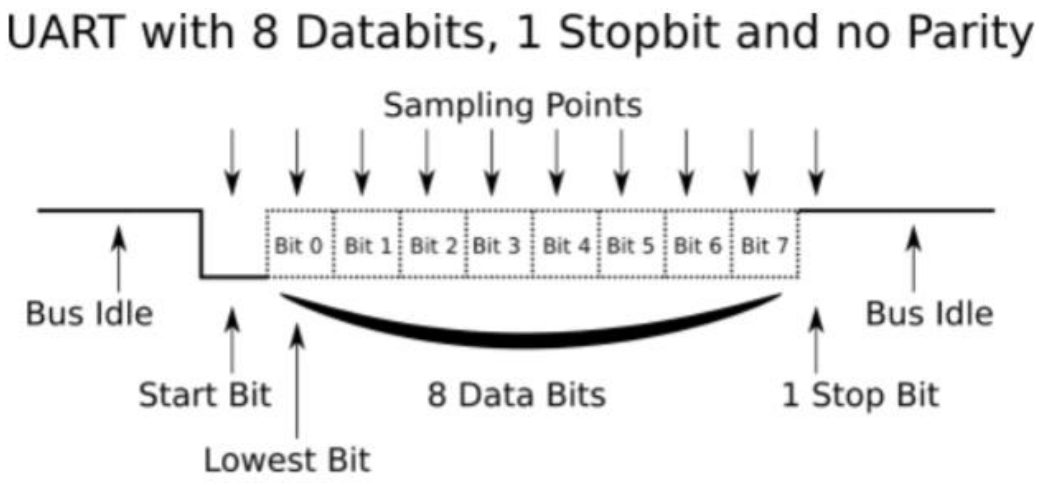


Figure 1 Data Transmison using UART

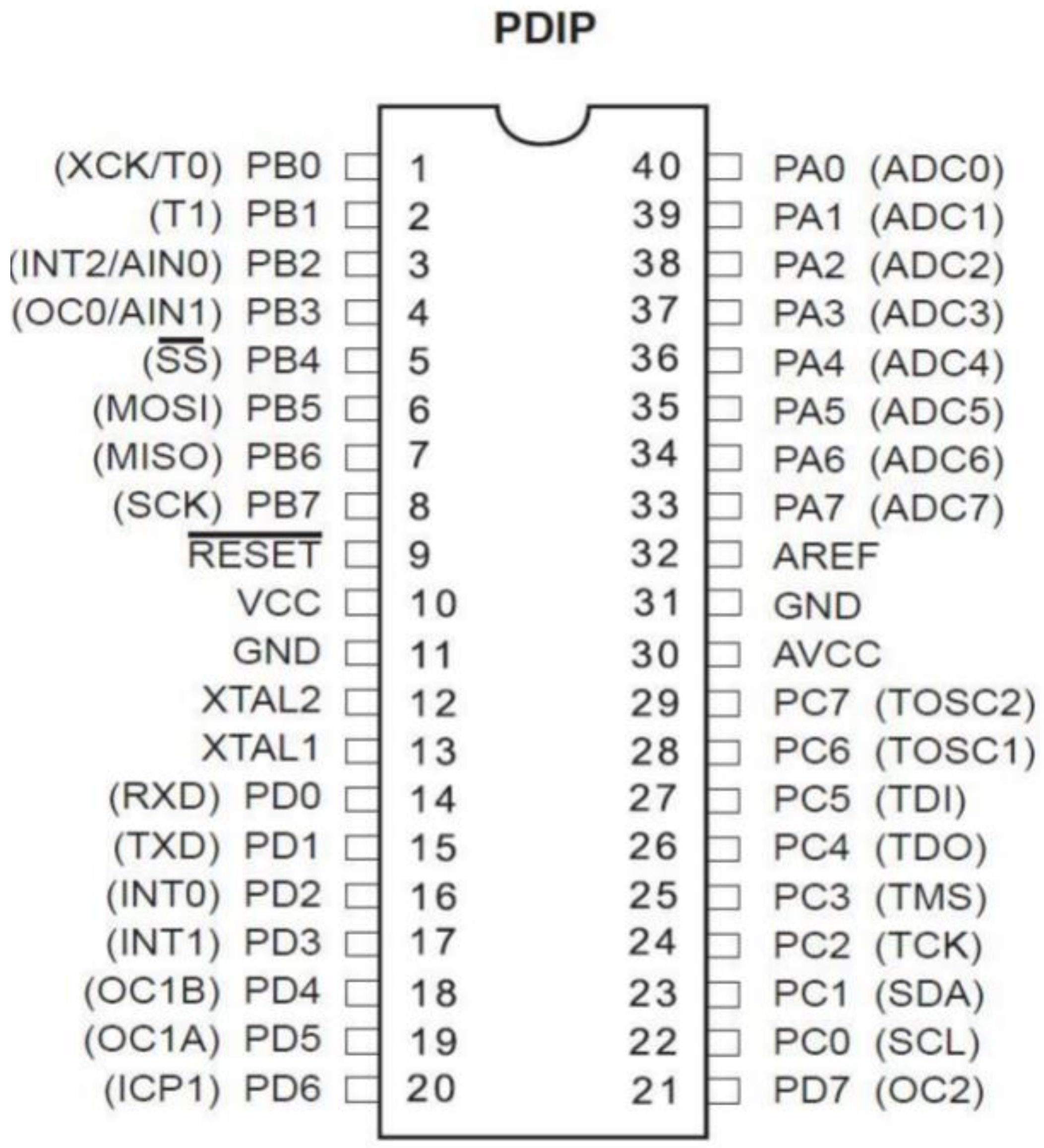
4.4 Atmel AVR

AVR is a family of microcontrollers developed by Atmel beginning in 1996. These are modified Harvard architecture 8-bit RISC single-chip microcontrollers. AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

AVR microcontrollers find many applications as embedded systems; they are also used in the popular Arduino line of open source board designs.

4.5 Atmel®AVR®ATmega32

The Atmel®AVR®ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.



5 Resources Used

5.1 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.

5.2 Proteus Design Suite

Proteus lets you create and deliver professional PCB designs like never before. With over 785 microcontroller variants ready for simulation straight from the schematic, built in STEP export and a world class shape based autorouter as standard, Proteus Design Suite delivers the complete software package for today and tomorrow's engineers. Proteus let's use simulate our hardware before creating it. It's very useful tool especially for beginners. It makes virtual "hardware" which will work like real one.

6 Solution

First thing we need to do is write a driver which will know how to interact with a peripheral device. Turns out this was the most difficult part of the laboratory.

6.1 UART Driver

In UART driver implementation I used the following dependencies:
#include <stdio.h> - used for defining UART as STD stream for IO library;
#include <avr/io.h>- header file including the appropriate IO definitions for the device that has been specified by the -mmcu= compiler command-line switch.

6.2 uart_studio.h

It is the header file for the written UART driver. It contains the specific includes and the functions prototypes:

```
void uart_Stdio_Init(void);
int uart_PutChar(char c, FILE *stream);
```

6.3 uart_studio.c

It is the file where the implementation functions for the written UART driver are written.

6.4 main.c

This is the entry point of the program. It works in the following way:

1) Declares the global variable for counting:
   `int count = 0;`
2) Initializes UART Driver
   `uart_Stdio_Init();`
3) Enters the infinite while loop:
   With a frequency of 1000 ms (_delay_ms(1000);)
   i) Increments the counter:
      `count = count + 1;`
   ii) Prints the counter on the screen:
      `printf("%d\n",count);`

The _delay_ms() function is retrieved from the <avr/delay.h> library.

7 Schematics

For our laboratory work we need only simple ATMega32 MCU and peripheral UART device, which in our case is virtual terminal.

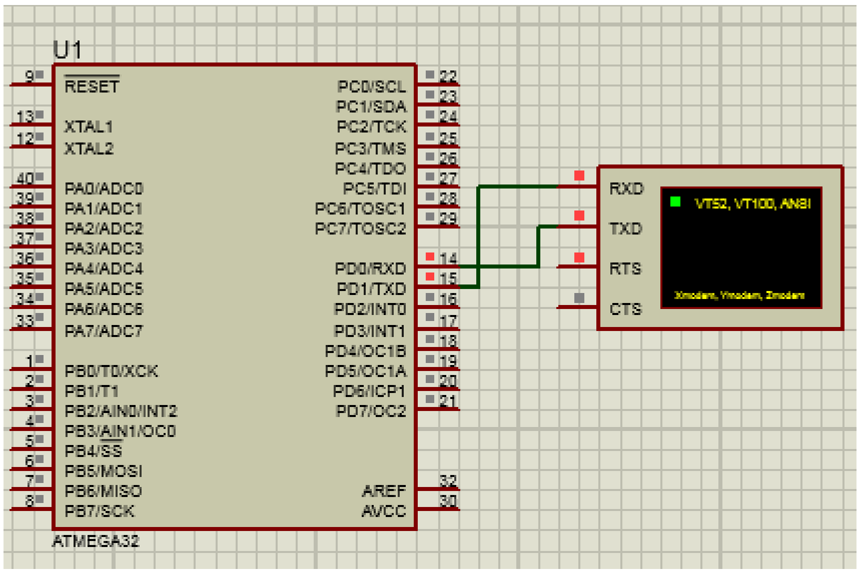
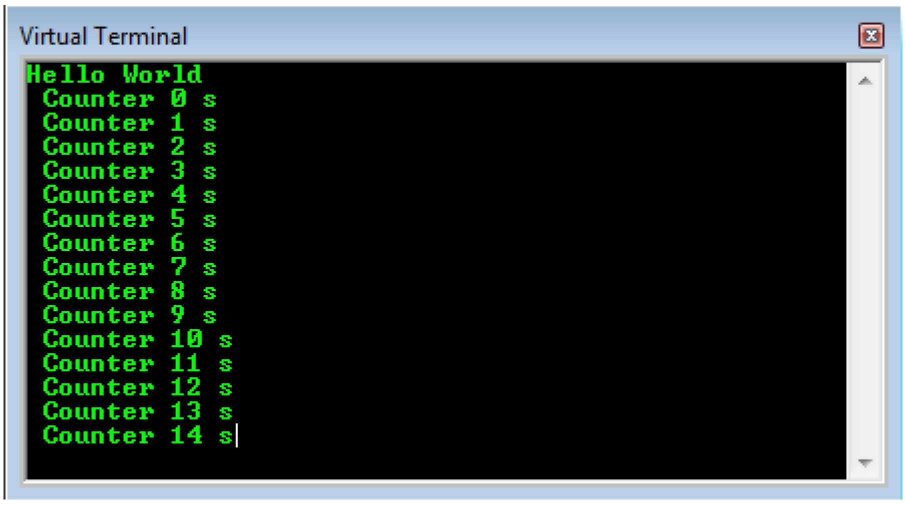


Figure 2 Our Scheme



```
Hello World
 Counter 0 s
 Counter 1 s
 Counter 2 s
 Counter 3 s
 Counter 4 s
 Counter 5 s
 Counter 6 s
 Counter 7 s
 Counter 8 s
 Counter 9 s
 Counter 10 s
 Counter 11 s
 Counter 12 s
 Counter 13 s
 Counter 14 s
```

Figure 3 Our result in the virtual terminal

8 Conculsion

This laboratory gave us the basic concepts about MCU programming in C and the construction of a PCB in Proteus. We also learned how to connect these 2 together to make them work. We have developed a program which uses UART for implementing a simple counter. This laboratory was something like an introduction to ES. I have to say it was pretty interesting and developed I field where I have had no previous knowledge. It was also quite interesting because there are a huge number of things around us that use MCU and now we know how they work.

9 Appendix

9.1 main.c

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


int count = 0;


void main() {
   uart_Stdio_Init();


   while(1){
       count = count + 1;
       printf("%d\n",count);
       _delay_ms(1000);
   }
}
```

In uart folder :

9.2 uart_stdio.h

```
#ifndef _UART_STDIO_H
#define _UART_STDIO_H
```

```
#define UART_BAUD 9600
           #define F_CPU 1000000UL


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


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


#endif
```

9.3 uart_stdio.c

```
#include "uart_stdio.h"


  FILE my_stream = FDEV_SETUP_STREAM(uart_PutChar, NULL,
_FDEV_SETUP_WRITE);


  void uart_Stdio_Init(void) {


      stdout = &my_stream;


#if F_CPU < 2000000UL && defined(U2X)
      UCSRA = _BV(U2X);             /* improve baud rate error
(2x clock) */
      UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;
#else
      UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;
#endif
              UCSRB = _BV(TXEN) | _BV(RXEN); /* enable
transmitter and receiver registers*/
  }


  int uart_PutChar(char c, FILE *stream) {


      if (c == '\n')
```

```
        uart_PutChar('\r', stream);

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

    return 0;
}
```