Ministerul Educaţiei şi Tineretului al Republicii Moldova

Universitatea Tehnică a Moldovei

Departament „Informatica aplicată”

RAPORT

Lucrarea de laborator nr.1

LA DISCIPLINA”Programarea aplicațiilor incorporate și independente de platformă ”

A efectuat:

St. gr. FAF 141 Bega Valeria

A verificat:

Lect . supp Bragarenco Andrei

Chişinău 2017

Topic: Initiating in MCU programing. Library organization. Connecting Stdio.

Objectives:

- Learn how to program a microcontroller in Atmel Studio
- Learn to build a scheme in Proteus
- Study and understand UART

Task:

Write a program that every second will print on virtual terminal, using UART, the value of a counter variable. Simulate the program on a scheme, constructed with Proteus.

Overview of the work:

1. What is an Embedded System?

An embedded system is a dedicated computer system designed for one or two specific functions. This system is embedded as a part of a complete device system that includes hardware, such as electrical and mechanical components. The embedded system is unlike the general-purpose computer, which is engineered to manage a wide range of processing tasks.

2. Microcontrollers

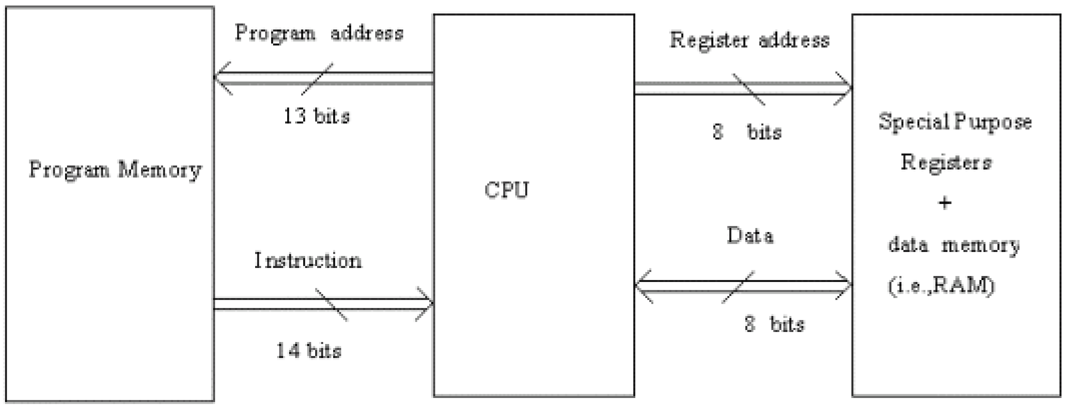
A microcontroller is a compact microcomputer designed to govern the operation of embedded systems in motor vehicles, robots, office machines, complex medical devices, mobile radio transceivers, vending machines, home appliances, and various other devices. A typical microcontroller includes a processor, memory, and peripherals.

The simplest microcontrollers facilitate the operation of the electromechanical systems found in everyday convenience items. Originally, such use was confined to large machines such as furnaces and automobile engines to optimize efficiency and performance. In recent years, microcontrollers have found their way into common items such as ovens, refrigerators, toasters, clock radios, and lawn watering systems. Microcomputers are also common in office machines such as photocopiers, scanners, fax machines, and printers.

3. MCU Architecture

Microprocessing unit is synonymous to central processing unit, CPU used in traditional computer. Microprocessor (MPU) acts as a device or a group of devices which do the following tasks.

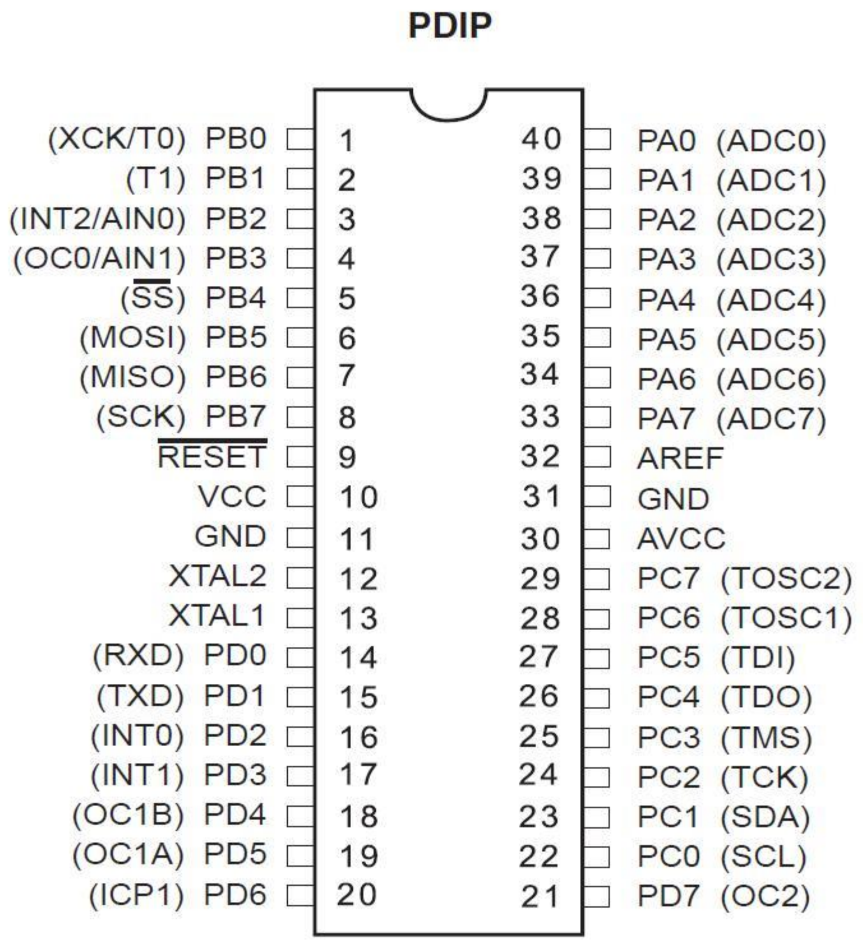
- communicate with peripherals devices
- provide timing signal
- direct data flow
- perform computer tasks as specified by the instructions in memory



4. AVR and Atmel®AVR®ATmega32

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.

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.



MCU Architecture

5. UART – Universal Asyncrhonous Reciever/Transmitter

A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. Specifically, it provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices. As part of this interface, the UART also:

- Converts the bytes it receives from the computer along parallel circuits into a single serial bit stream for outbound transmission
- On inbound transmission, converts the serial bit stream into the bytes that the computer handles
- Adds a parity bit (if it's been selected) on outbound transmissions and checks the parity of incoming bytes (if selected) and discards the parity bit
- Adds start and stop delineators on outbound and strips them from inbound transmissions
- Handles interrupt s from the keyboard and mouse (which are serial devices with special port s)
- May handle other kinds of interrupt and device management that require coordinating the computer's speed of operation with device speeds

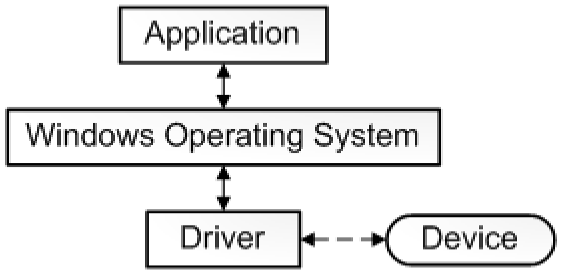
More advanced UARTs provide some amount of buffering of data so that the computer and serial devices data streams remain coordinated. The most recent UART, the 16550, has a 16-byte buffer that can get filled before the computer's processor needs to handle the data. The original UART was the 8250. If you purchase an internal modem today, it probably includes a 16550 UART (although you should ask when you buy it). According to modem manufacturer US Robotics, external modems do not include a UART. If you have an older computer, you may want to add an internal 16550 to get the most out of your external modem.

6. Input/Output

In computing, input/output or I/O (or, informally, io or IO) is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are used by a human (or other system) to communicate with a computer. For instance, a keyboard or computer mouse is an input device for a computer, while monitors and printers are output devices. Devices for communication between computers, such as modems and network cards, typically perform both input and output operations.

7. Drivers

It is challenging to give a single precise definition for the term *driver*. In the most fundamental sense, a driver is a software component that lets the operating system and a device communicate with each other. For example, suppose an application needs to read some data from a device. The application calls a function implemented by the operating system, and the operating system calls a function implemented by the driver. The driver, which was written by the same company that designed and manufactured the device, knows how to communicate with the device hardware to get the data. After the driver gets the data from the device, it returns the data to the operating system, which returns it to the application.



8. Tools used during this laboratory work:

- 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 easyto-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

Proteus is a Virtual System Modelling and circuit simulation application. The suite combines mixed mode SPICE circuit simulation, animated components and microprocessor models to facilitate co-simulation of complete microcontroller based designs. Proteus also has the ability to simulate the interaction between software running on a microcontroller and any analog or digital electronics connected to it. It simulates Input / Output ports, interrupts, timers, USARTs and all other peripherals present on each supported processor.

Solution:

In order to proceed to use UART, we had to write a driver which was going to interact with the virtual terminal.

The projects had the following structure:

UART Driver

`#include <stdio.h>` - for defining UART as STD stream for I/O library

`#include <avr/io.h>` - This header file includes the apropriate IO definitions for the device that has been specified by the -mmcu= compiler command-line switch.

uart_stdio.h

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

uart_stdio.c

Here is where the UART driver function is implemented.

main.c

- Declares the global variable for counting:

  int count = 0;

- Initializes UART Driver

  uart_Stdio_Init();

- Enters the infinite while loop:

  With a frequency of 1000 ms (_delay_ms(1000);)

  - Increments the counter:

    count = count + 1;

  - Prints the counter on the screen:

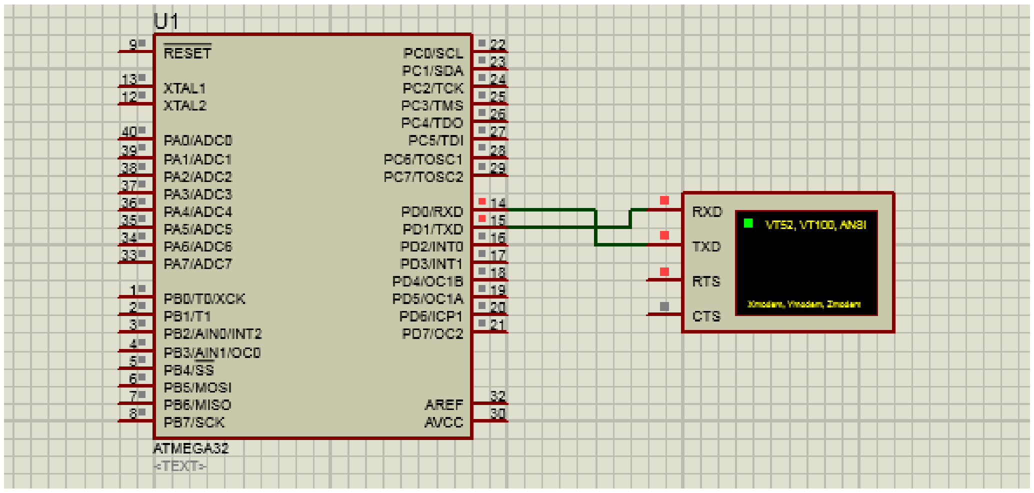
    printf("%d\n",count);

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

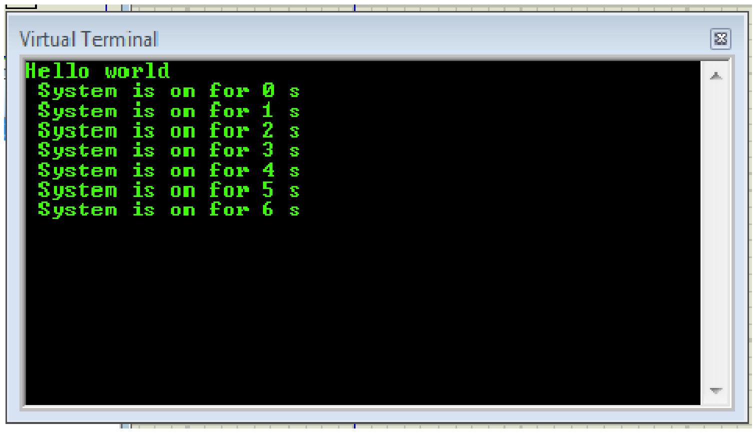
The next step is **Proteus Simulation:**

After the implementation of the solution, we compile it with the Build Solution in Atmel Studio and give the right path, where the file with the.hex extension is located.

This is how the scheme looks in Proteus:



This is how the simulation result looks like:



```
Hello world
 System is on for 0 s
 System is on for 1 s
 System is on for 2 s
 System is on for 3 s
 System is on for 4 s
 System is on for 5 s
 System is on for 6 s
```

Conclusion:

First of all, this laboratory work served as a good reminder of lots of information I learned in previous years, but forgot. We had a litle revision regarding of how to work with libaries, how to create a driver and of course, how to write code in C. I also would like to mention that during the laboratory work, the teacher corrected us lots of times on our intendations, and this also served as a good practice of how to write code.

Secondly, for the first time I got to learn about Proteus and AVR Studio as an introduction to Embedded Systems. I created a circuit in Proteus to simulate the 'counter program'.

Lastly, I would like to mention that the laboratory work was quite difficult to write, because its very different from what we do, but with a little internet research, I managed to do everything.

Appendix:

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);
   }
}
```

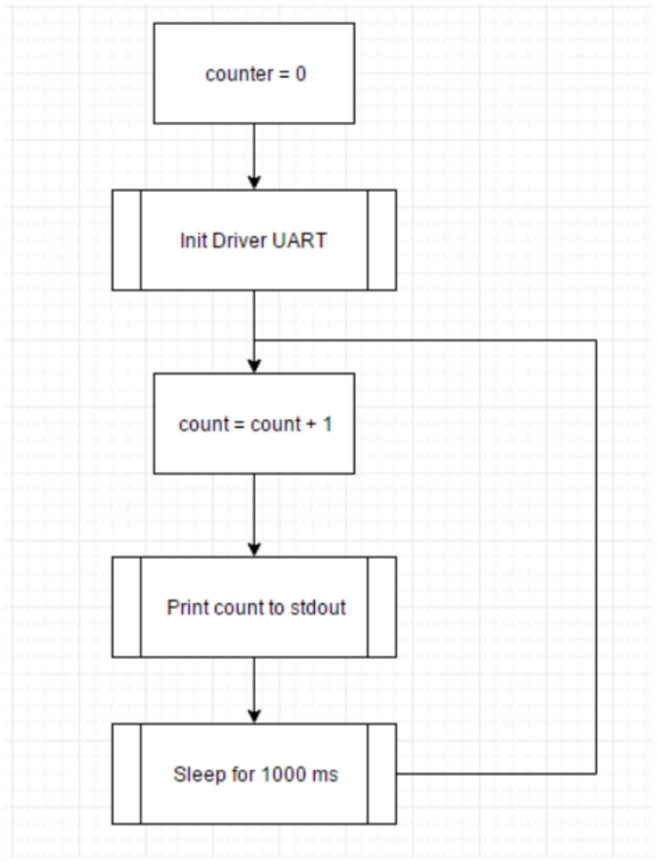
uart_stdio.h

```
#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
```

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;
  }
```

Flowchart:


counter = 0
Init Driver UART
count = count + 1
Print count to stdout
Sleep for 1000 ms