Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

at Embedded Systems

Laboratory Work #1

Topic: Introduction to MCU. Serial interfacing using UART – Universal Asynchronous Receiver/Transmitter

Performed by: Diana ARTIOM

Verified by: Andrei BRAGARENCO

Chisinau 2016

Topic:

Introduction to MCU. Serial interfacing using UART – Universal Asynchronous Receiver/Transmitter

Objectives:

- Get in touch with what MCU means
- Study UART and understand basic concept
- Write and execute the program for 8 bit ATMega32 MCU using PROTEUS simulator

Task:

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

Overview:

Embedded systems

An **embedded system** is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular function. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with programming interfaces, and embedded systems programming is a specialized occupation.

Microcontrollers

A **microcontroller** (or **MCU**) is a small computer on a single integrated circuit containing a processor core, 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 typically 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.

MCU Architecture

A basic CPU architecture is depicted in Figure 2.1. It consists of the data path, which executes instructions, and of the control unit, which basically tells the data path what to do.

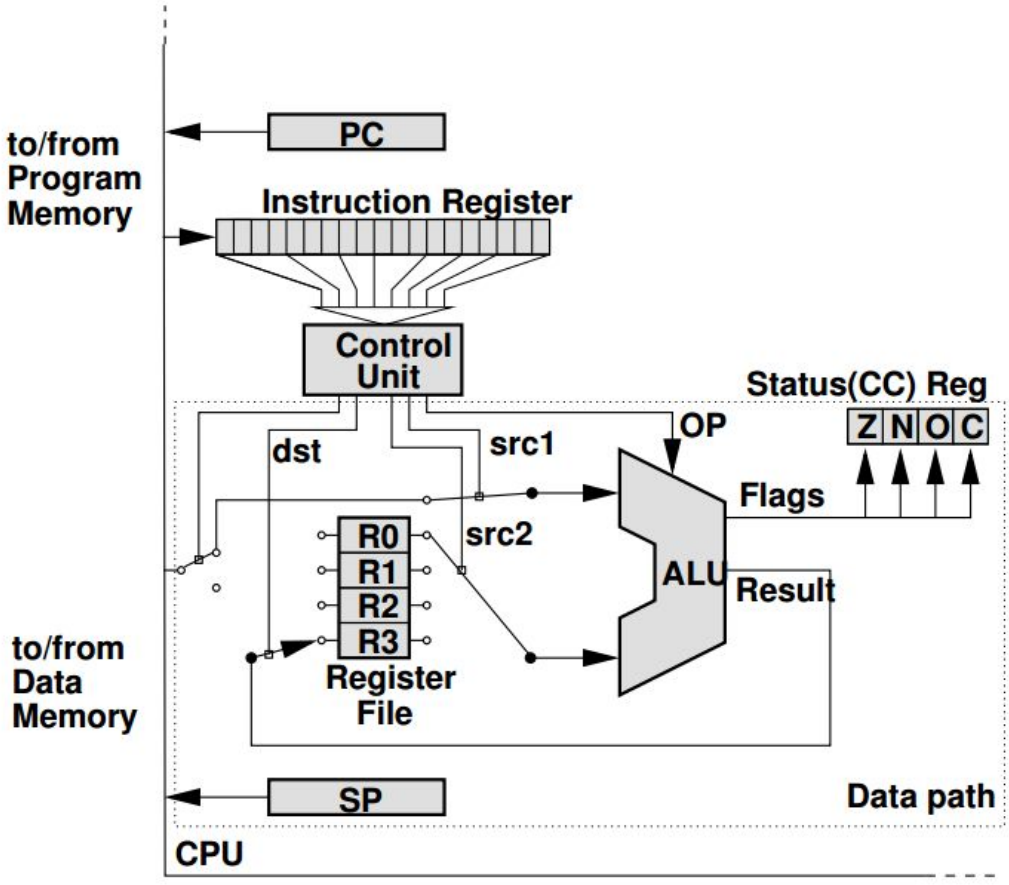


Figure 1: Representation MCU Architecture

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.

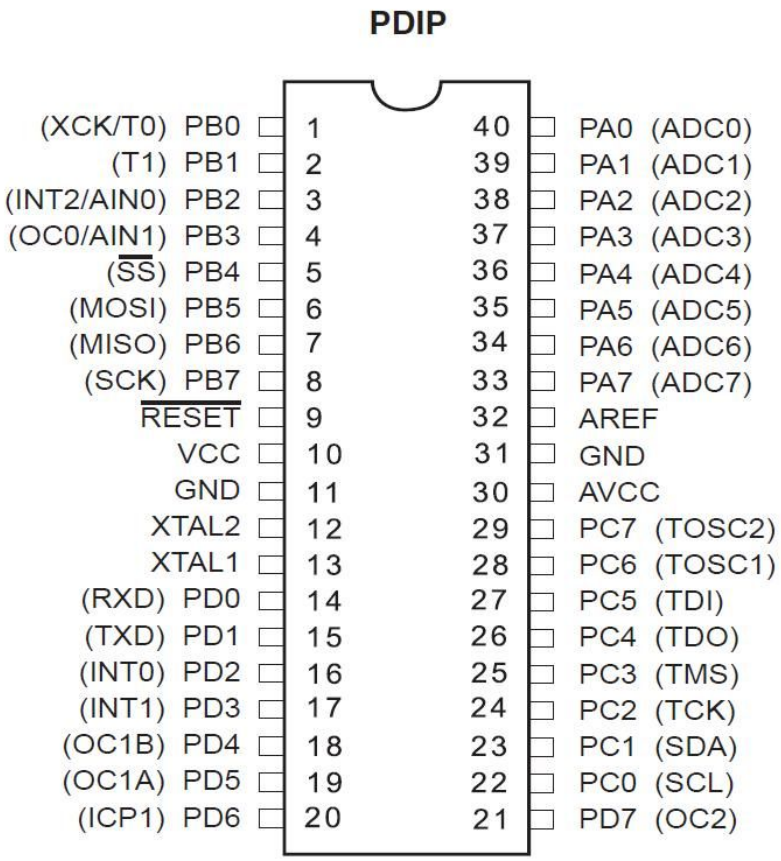


Figure 2: Representation of ATMega32 MCU

UART - Universal Asynchronous Receiver/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 interrupts 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

Tools and Technologies used:

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.

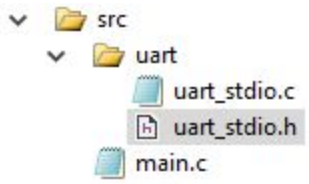
Proteus

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 wrote a driver to actually interact with the virtual terminal, which is, by means, the essence of the laboratory work.

But, before proceeding to explain which is what, I will first include here the project structure of the:



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.

uart_stdio.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);
```

uart_stdio.c

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

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 `<avr/delay.h>` library.

Preparing for Proteus Simulation

After implementing the solution in terms of C code, I compiled it using the **Build Solution** option in Atmel Studio. The path for the program to run on MCU in Proteus, is the file with extension **.hex**, found in .../lab1/lab1/debug folder, with the name lab1.hex:

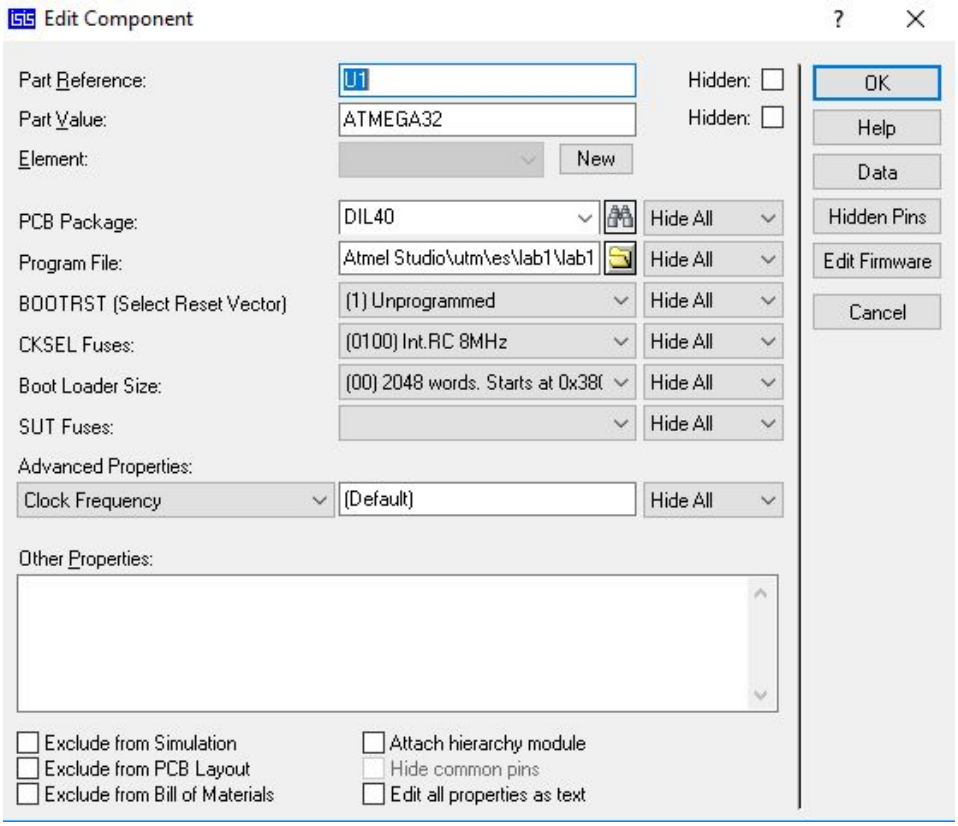


Figure 3: Setting the path to the .hex file - output of the program

Here is the scheme constructed in Proteus:

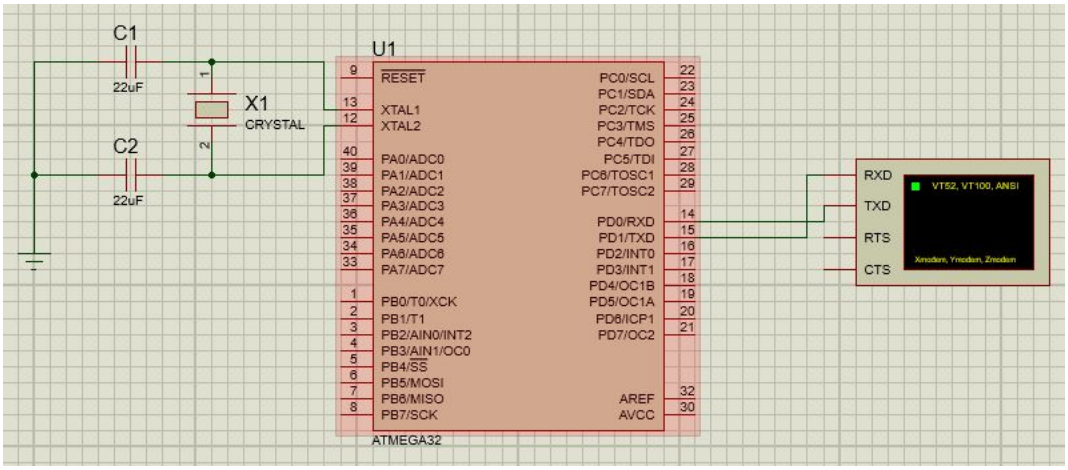


Figure 4: Construction of the scheme - Virtual Terminal Interfacing

Result:

Figure 4: Output of the program

Conclusion:

By performing this laboratory work I was introduced to the world of Microcontrollers. I learned the basic concepts of what a MCU is, how does it work, and even how it deals with communication - by working with UART. I was impressed about the use of MCU and now, after this laboratory work, I understand the importance of the microcontrollers, and their immense usage, of which I didn't even think about before.

P.S. On a positive note, being pleasantly surprised, I'm about to take the decision to write my licence paper on MCUs.

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);
   }
}
```

In uart folder :

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;
  }
```