Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

RAPORT

la Programarea aplicațiilor incorporate și independente de platformă

Lucrare de laborator Nr.1

Tema:”Introducerea in programarea MicroControloarelor si implementarea comunicareii seriale UART - Transmitator/Receptor universal asincron”

A efectuat st. gr. FAF-141: Nicu Maxian

A verificat: Andrei Bragarenco

Chişinău 2016

Topic

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

Scope

- Gain basic knowledge about Micro Controller Unit
- Programming MCU in ANSI C
- Study of UART serial communication
- Creating PCB in Proteus
- Executing written program for 8 bit **ATMega32 MCU** on created schema.

Task

Write a C program and schematics for **Micro Controller Unit** (MCU) using **Universal asynchronous receiver/transmitter**. For writing program, use ANSI-C Programming Language with **AVR Compiler** and for schematics use **Proteus**, which allow us to simulate real example.

Domain

Embedded systems

There's no perfect answer to that question, since every answer will have some exceptions. However let us declare that an embedded system is one that uses one or more microcomputers (that is, small to very, very small computers), running custom dedicated programs and connected to specialized hardware, to perform a dedicated set of functions. This can be contrasted with a general-purpose computer such as the familiar desktop or notebook, which are not designed to run only one dedicated program with one specialized set of hardware. It's not a perfect definition, but it's a start.

Some examples of embedded systems are:

- Alarm / security system
- Automobile cruise control
- Heating / air conditioning thermostat
- Microwave oven
- Anti-skid braking controller
- Traffic light controller
- Vending machine
- Gas pump
- Handheld Sudoku game
- Irrigation system controller
- Singing wall fish (or this gift season's equivalent)
- Multicopter
- Oscilloscope
- Mars Rover

Micro Controller

A microcontroller (or MCU, short for microcontroller unit) 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.

When you write a program for your microcontroller you are really writing a program that is executed by the MC CPU (central processing unit)

In the simplest sense, a CPU is that part of the microcontroller that executes instructions. It does this in a series of steps:

- Fetch an instruction from the "next instruction" memory location pointer
- Execute that instruction
- Advance the "next instruction" pointer accordingly Every computer program is just a repetitive execution of this sequence.

CPU REGISTERS

Any CPU will have a set of onboard registers, which can be viewed as very fast memory locations. These registers will either be addressed, or they will be addressed by a few bits in the instruction operation code (op code).

Stack Pointer

The stack pointer is an address register which points to a section of memory that is used for the CPU hardware stack. The hardware stack is the stack that is used by the hardware for subroutine calls and returns, and for interrupt calls and returns. It is also possible for the user program to use the same stack, pointed to by the stack pointer, for saving and restoring other data.

Program Counter (also known as Instruction Pointer)

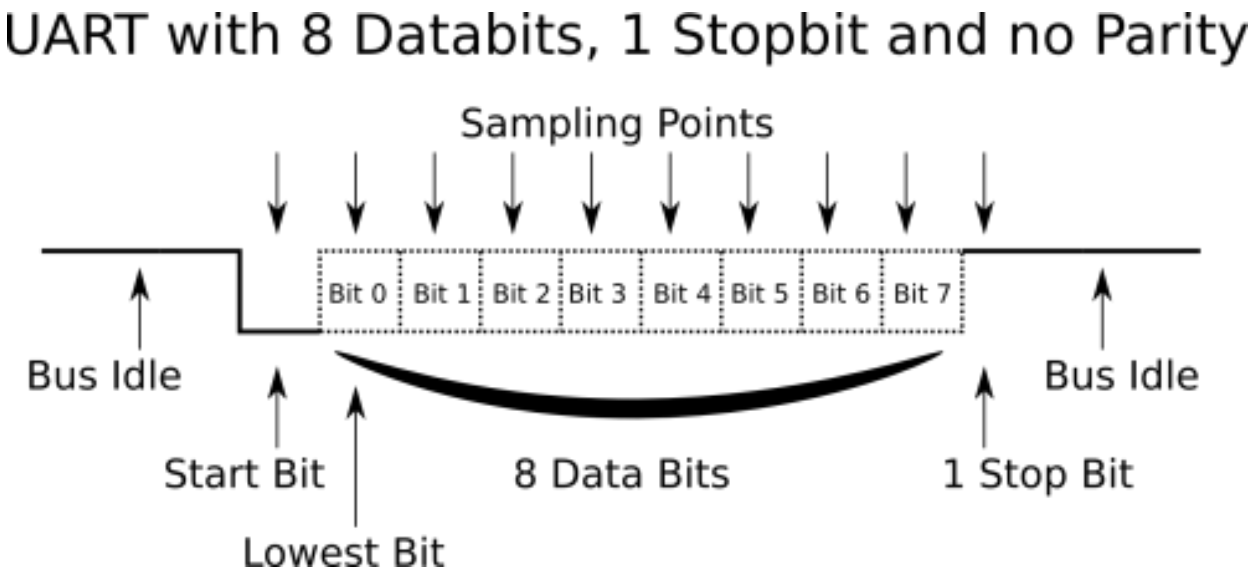
This is the address register that points to the current (or next) instruction to be executed. It may be accessed by special instructions, or it may be a standard address regiser or even a full general-purpose register. It will automatically advance to point to the next instruction in a

program, and will automatically be adjusted based on program jump, call and return instructions.

Universal asynchronous receiver/transmitter

A universal asynchronous receiver/transmitter , 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.

Data transmission over **UART**



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.

Used Resources

Atmel Studio (previously AVR Studio)

Atmel Studio is the integrated development platform (IDP) for developing and debugging Atmel® SMART ARM®-based and Atmel AVR® microcontroller (MCU) applications. Studio supports all AVR and Atmel SMART MCUs. The Atmel Studio IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. Although we need just AVR C Compiler, for compiling C Program in Hex, we will also use AVR IDE for development. It has some features like :

- Support for 300+ Atmel AVR and Atmel SMART ARM-based devices
- Write and debug C/C++ and assembly code with the integrated compiler
- Integrated editor with visual assist

In my laboratory work I used Atmel Studio 6 which is based on Visual Studio IDE.

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.

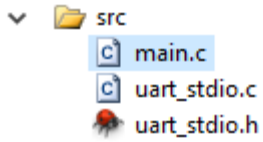
Proteus let’s use simulate our hardware before creating it. It’s very useful tool especially for beginners. It makes virtual “hardware” which will work like real one.

In my laboratory work I used Proteus Design Suite 8 as one of latest version of current software.

Solution

First of all, in order to use UART we need to write a **Driver** which will know how to interact with peripheral device. This is actually harder thing in our laboratory work.

Generally, **Project Structure** looks in this way



UART Driver

UART driver has dependencies on.

#include <stdio.h>

For defining UART asa STD stream for IO Library. Example :

```
FILE uart_istream = FDEV_SETUP_STREAM(uart_PutChar, NULL, _FDEV_SETUP_WRITE);
```

#include <avr/io.h>

It has MACRO definition for registers which makes our driver to work not only on ATMega32 but on more devices. I checked source code and it has definition for many Micro Controller CPUs.

uart_stdio.h / uart_stdio.c

Header file for UART Driver. It has only 1 procedure and 1 function.

void uart_stdio_Init(void);

This procedure initializes UART Baud frequency in order to make peripheral device to understand our signals correct.

Int uart_PutChar(char c, FILE *stream);

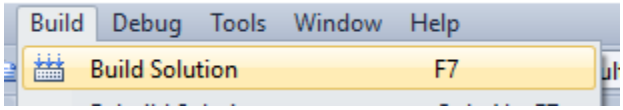
Function for printing/sending char to peripheral device.

Main Program

There are no mountains and super code in this section. There is a simple code which works in following way

1. Declares global variable for counting (can be used a local one instead, no matter in our case)
2. Initializes UART Driver
3. Start infinite loop
    a. Increment counter variable
    b. Print counter to output(UART)
    c. Sleep…

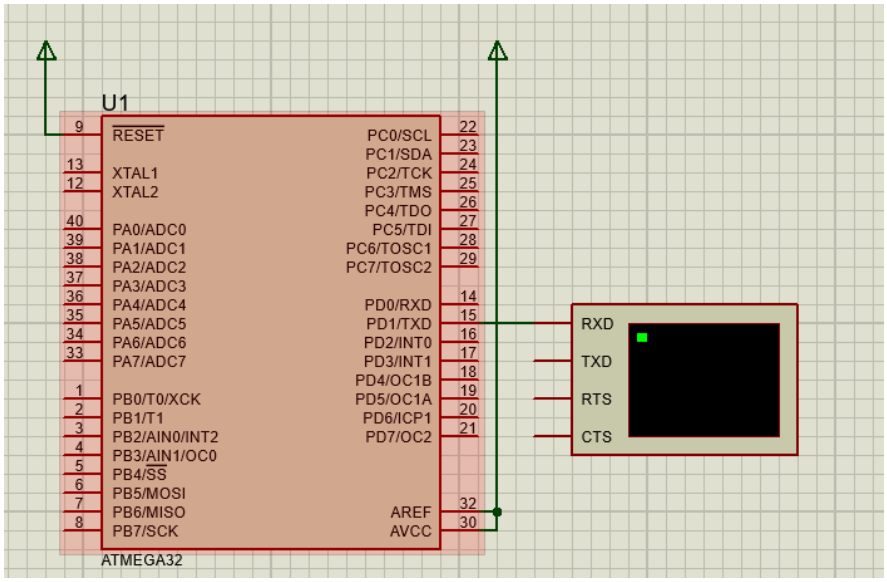
Sleep is done with **avr/delay.h** library, which has method procedure **_delay_ms(duration)** defined
After code implementation, we should now **Build Hex** which will be written to MCU ROM.



Now, as we have already HEX Program for MCU, we need to construct our PCB. For this we will use **Proteus.**

Schematics

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

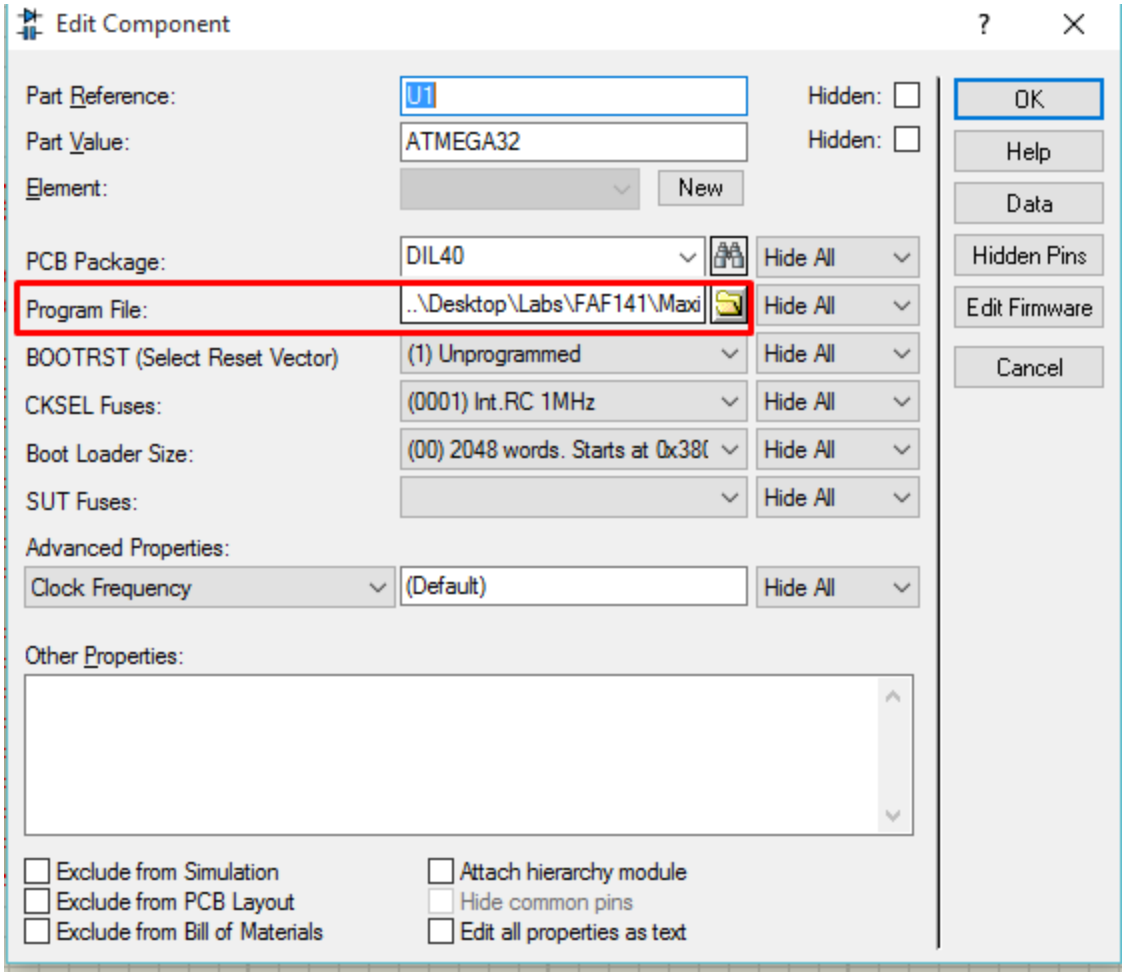


We need to make sure that our MCU is connected to Virtual Terminal. Because we use only data transmission on one direction (OUTPUT) we need to make sure that our MC **Tx** is connected to Peripheral **Rx**.

MCU is transmitter and peripheral is receiver. No vice versa connection because we don't need it in our laboratory work.

Attaching Program to our virtual MCU

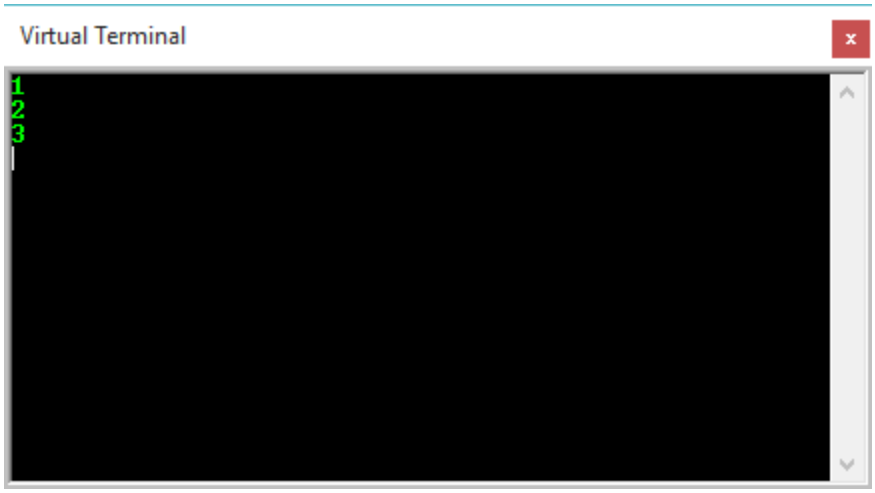
In order to attach HEX to MCU, edit ATMega32 component and find following parameter.



This field will ask us to select HEX file, which can be found in generated output from **Atmel Studio**.

So, actually we **completed** all steps. Finally we can run simulation, to see result.

Simulation Result



Conclusion

This laboratory work gave us a basic concepts about MCU Programming in C and constructing own Printed circuit board (PCB) in Proteus. We have developed simple program which uses UART for implementing simple counter. This laboratory work was actually introduction for us in Embedded System word, starting with ANSI-C and ending with Writing Generated HEX to MCU ROM. Generally speaking, it was a good and inspiring intro into this world and I really want to study and gain more knowledge about Embedded System and MCU programming

After this laboratory work we understand how important is role of Embedded System in our life. Nowadays we have a lot of devices which uses MCU and programs “installed” on them(also called firmware), to do some work (ex. Microwave, Alarm …).

Appendix

Main.c

```
#include <avr/io.h>
#include <avr/delay.h>
#include "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 uart_istream = FDEV_SETUP_STREAM(uart_PutChar, NULL, _FDEV_SETUP_WRITE);

void uart_stdio_Init(void)
{
	stdout = &uart_istream;

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

FlowChart