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 în programarea microcontrolerelor si implementarea comunicării seriale UART”

A efectuat st. gr. FAF-141: Oxana Dunav

A verificat: Andrei Bragarenco

Chişinău 2016

Topic

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

Purpose

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

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

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.

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces),[7] 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 specialised 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).

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.

➔ Microcontroller

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.

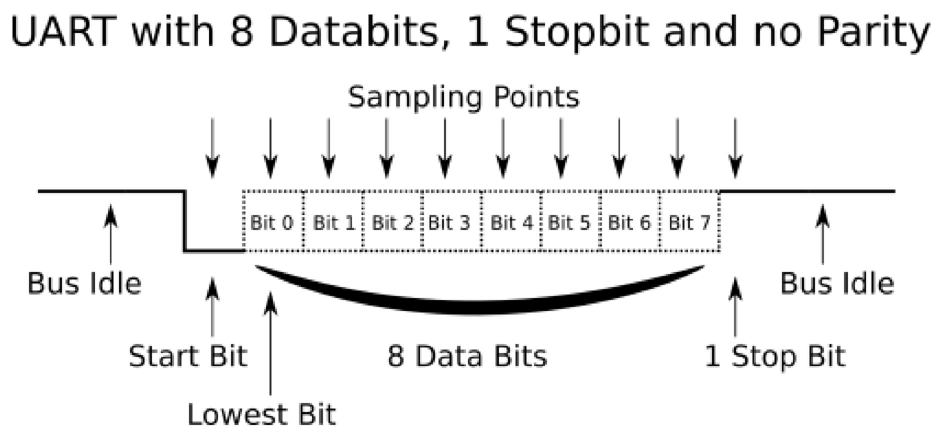
Program Counter (also known as Instruction Pointer)

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 5 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.

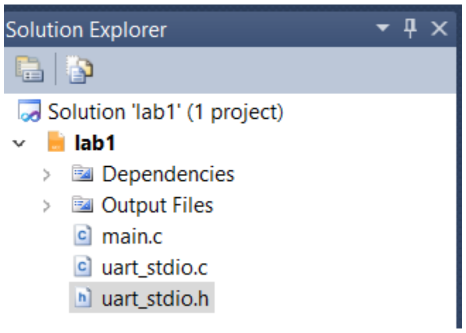
Proteus let's use simulate our hardware before creating it. It's very useful tool especially for beginners. It makes virtual "hardware" which will work like real one.

In my laboratory work I used Proteus 8 Professional.

Solution

First of all, in order to use UART we need to write a driver which will know how to interact with peripheral device.

The *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 microcontroller 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);
```

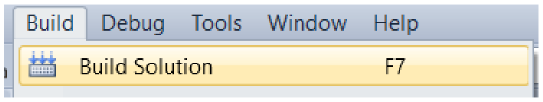
Function for printing/sending char to peripheral device.

Main Program

The code 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…

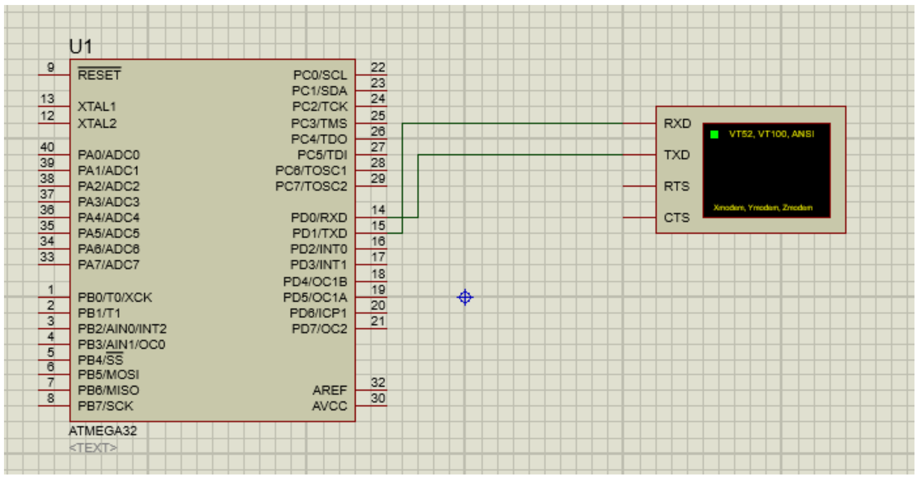
Sleep is done with avr/delay.h library, which has method procedure _delay_ms(duration) defined
After code implementation, we should now Build Hex which will be written to MCU ROM.



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.

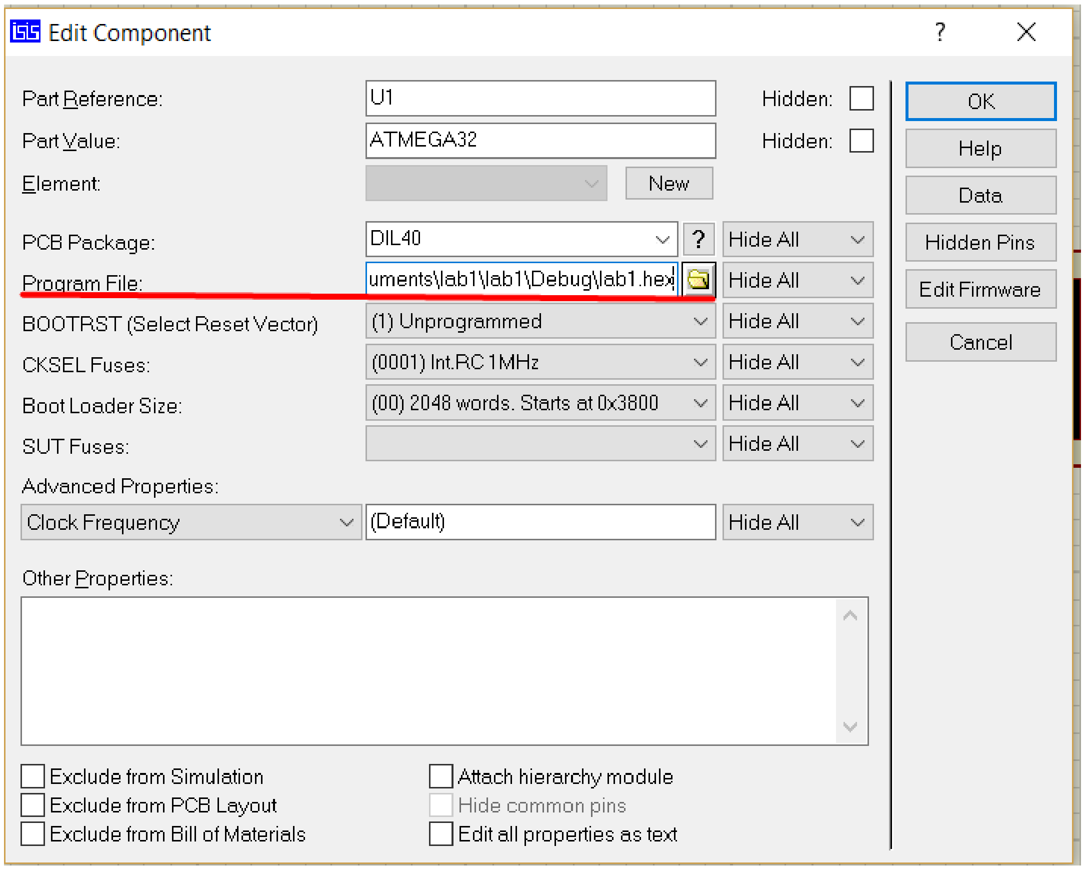


We need to make sure that our MCU is connected to Virtual Terminal. Because we use only data transmission on one direction (OUTPUT) we need to make sure that our MC **Tx** is connected to Peripheral **Rx**.

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.

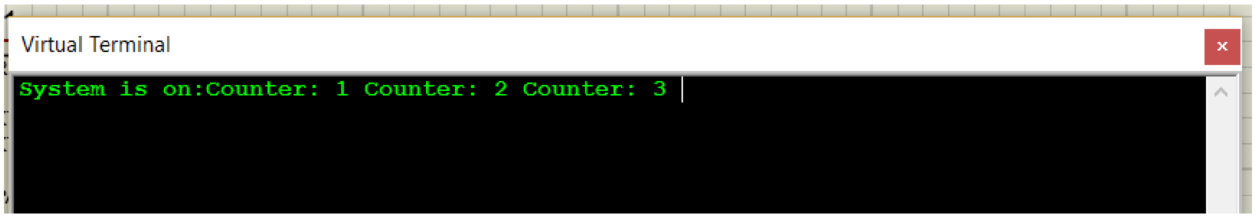


Part Reference: U1
Part Value: ATMEGA32
Element:
PCB Package: DIL40
Program File: uments\lab1\lab1\Debug\lab1.hex
BOOTRST (Select Reset Vector) (1) Unprogrammed
CKSEL Fuses: (0001) Int.RC 1MHz
Boot Loader Size: (00) 2048 words. Starts at 0x3800
SUT Fuses:
Clock Frequency (Default)

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



```
System is on:Counter: 1 Counter: 2 Counter: 3
```

Conclusion

During this laboratory work I learned about MCU programming in C and how to create a circuit in Proteus for simulating the program for implementing a counter.

This laboratory work was very helpful as an introduction in Embedded programming and I learned a lot of new information by doing it. It was a little difficult as it is different from what we have learned before and is something new, but after doing a little research on the internet, I got to understand it better and how it works.

I think that embedded systems have an important role in our lives and it is useful to know about them.

Appendix

main.c

```c
#include "uart_stdio.h"
#include <util/delay.h>

int cnt = 0;
int main(void) {
	uart_stdio_Init();

	printf("System is on:");
	while(1){
		cnt++;
		printf("Counter: %d \n", cnt);
		_delay_ms(1000);
	}
	return 0;
}
```

uart_stdio.h

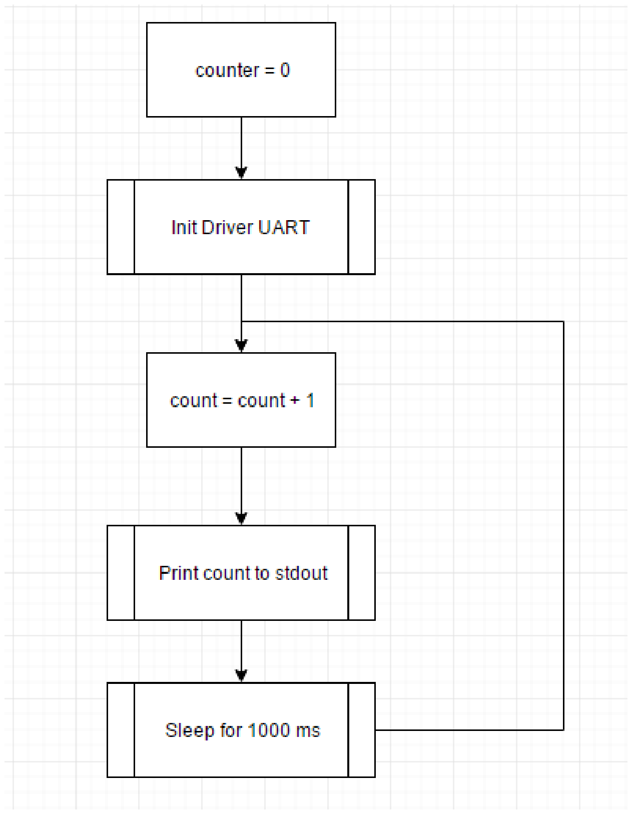
```c
#ifndef UART_STDIO_H_
#define UART_STDIO_H_

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

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

#endif /* UART_STDIO_H_ */
```

uart_stdio.c

```c
#include "uart_stdio.h"
```

```c
#define UART_BAUD  9600

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

FILE std_out =  FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_RW);

int uart_putchar(char c, FILE *stream) {
	while(~UCSRA &(1<<UDRE)); //wait while register is free
	UDR = c; //load c in register

	return 0;
}

void uart_stdio_Init(void) {

	#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 */
	stdout = &std_out;
}
```

FlowChart


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