Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

Embedded Systems

Laboratory Work #1*: Introduction to MCU. Serial interfacing using UART – Universal Asynchronous Receiver/Transmitter*

Performed by: Ana-Maria Brinza

Verified by: Andrei Bragarenco

Chisinau 2016

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

Objectives:

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

Task: Write a program that will print on the virtual terminal, using UART, the value of a counter variable. Simulate the program on a scheme in Proteus.

General Information:

Microcontrollers

A microcontroller is a self-contained system with **peripherals, memory and a processor** that can be used as an embedded system. Most programmable microcontrollers that are used today are embedded in other consumer products or machinery including phones, peripherals, automobiles and household appliances for computer systems. Due to that, another name for a microcontroller is "embedded controller." Some embedded systems are more sophisticated, while others have minimal requirements for memory and programming length and a low software complexity. Input and output devices include solenoids, LCD displays, relays, switches and sensors for data like humidity, temperature or light level, amongst others.

Types of Microcontrollers

There are several different kinds of programmable microcontrollers at Future Electronics. We stock many of the most common types categorized by several parameters including Bits, Flash size, RAM size, number of input/output lines, packaging type, supply voltage and speed. Our parametric filters will allow you to refine your search results according to the required specifications.

Programmable microcontrollers contain general purpose input/output pins. The number of these pins varies depending on the microcontroller. They can be configured to an input or an output state by software. When configured to an input state, these pins can be used to read external signals or sensors. When they are configured to the output state, they can drive external devices like LED displays and motors.

MCU Architecture

The architecture of a microcontroller depends on the application it is built for. For example, some designs include usage of more than one RAM, ROM and I/O functionality integrated into the package.

The architecture of a typical microcontroller is complex and may include the following:

1. A CPU, ranging from simple 4-bit to complex 64-bit processers.
2. Peripherals such as timers, event counters and watchdog.
3. RAM (volatile memory) for data storage. The data is stored in the form of registers, and the general-purpose registers store information that interacts with the arithmetic logical unit (ALU).
4. ROM, EPROM, EEPROM or flash memory for program and operating parameter storage.
5. Programming capabilities.
6. Serial input/output such as serial ports.
7. A clock generator for resonator, quartz timing crystal or RC circuit.
8. Analog-to-digital convertors.
9. Serial ports.
10. Data bus to carry information.

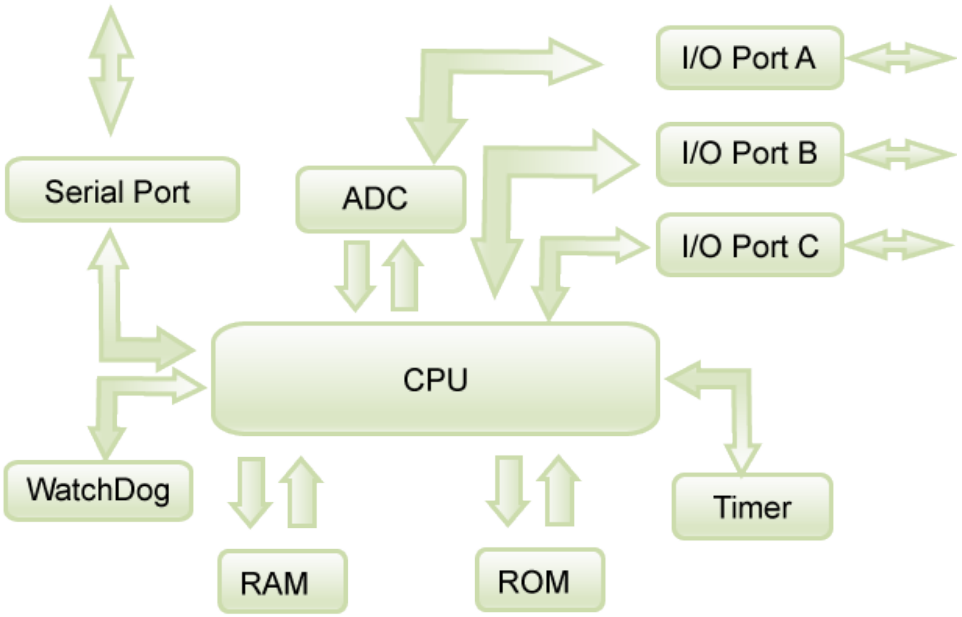

Figure 1: General architecture of MCU

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.

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.

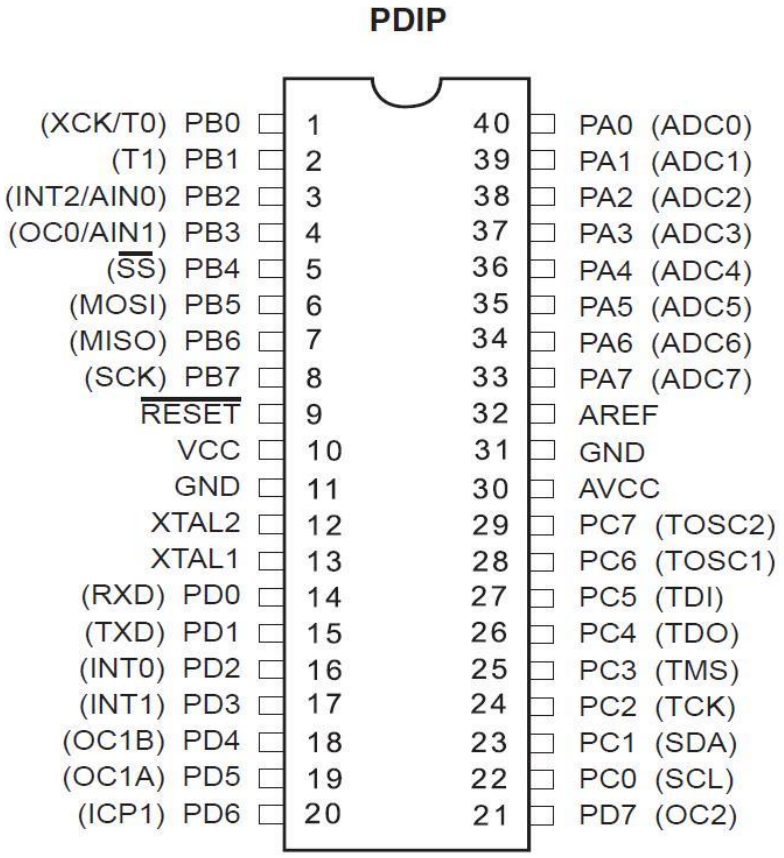

Figure 2: Representation of ATMega32 MCU

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.

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:

The first thing to do is to create the new project in Atmel Studio 7. We will select the type of the project we want to create, give the name and select a directory for the new project.

As shown in the image below, I created the project named *labNr1*.

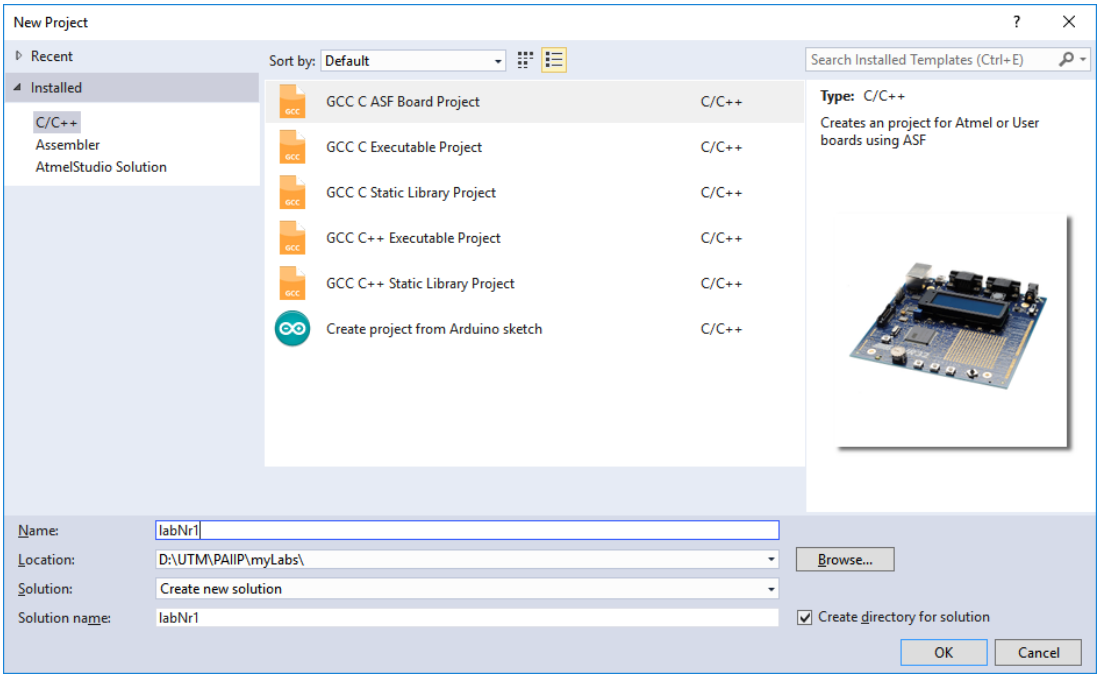

Figure 2.1: Creating and defining the new project in Atmel Studio

As we proceed we will have to select the device for simulation, which in our case would be ATmega32.

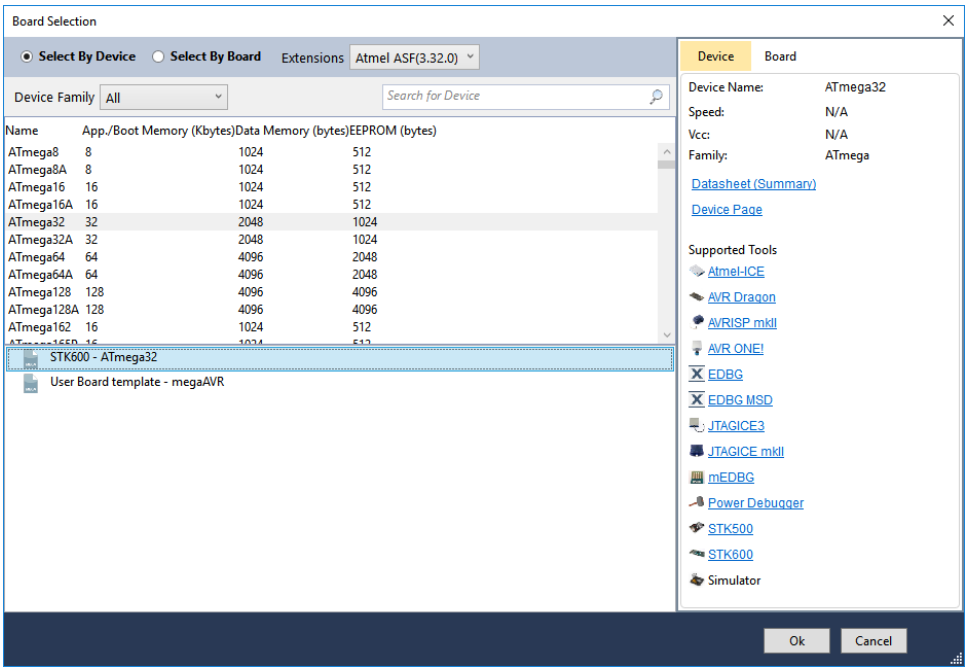

Figure 2.2: Select the device in Atmel Studio

The main and most important part of this laboratory work is to use UART therefore we have to write a driver to interact with the virtual terminal.

The main files in my project are:

uarts.h – header file for the UART written driver
uarts.c – the C file with the implementation functions for UART
main.c – the main part of the project.

Some important things to notice about the code:

In **uarts.h** file I included:

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

Also, this file contains the functions prototypes:

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

In **uart_stdio.c** file are the implimentations of the tw functions declared in uarts.h file.

In **main.c** file we first declare the variable for counting

```
int count = 0;
```

Then the UART Driver is initialized:

```
uartS_Init();
```

After that the infinite while loop is defined:

With a frequency of 1000 ms (`_delay_ms(1000)` - a function from `<avr/delay.h>` library)

Also the counter is incremented at each step and printed out.

Preparing for Proteus Simulation

In order to simulate the project in Proteus we will need to construct the scheme which is basically composed of a *Virtual Terminal* and a *Microcontroller ATmega32*. The scheme of the system is showed in the *Figure 3*, below:

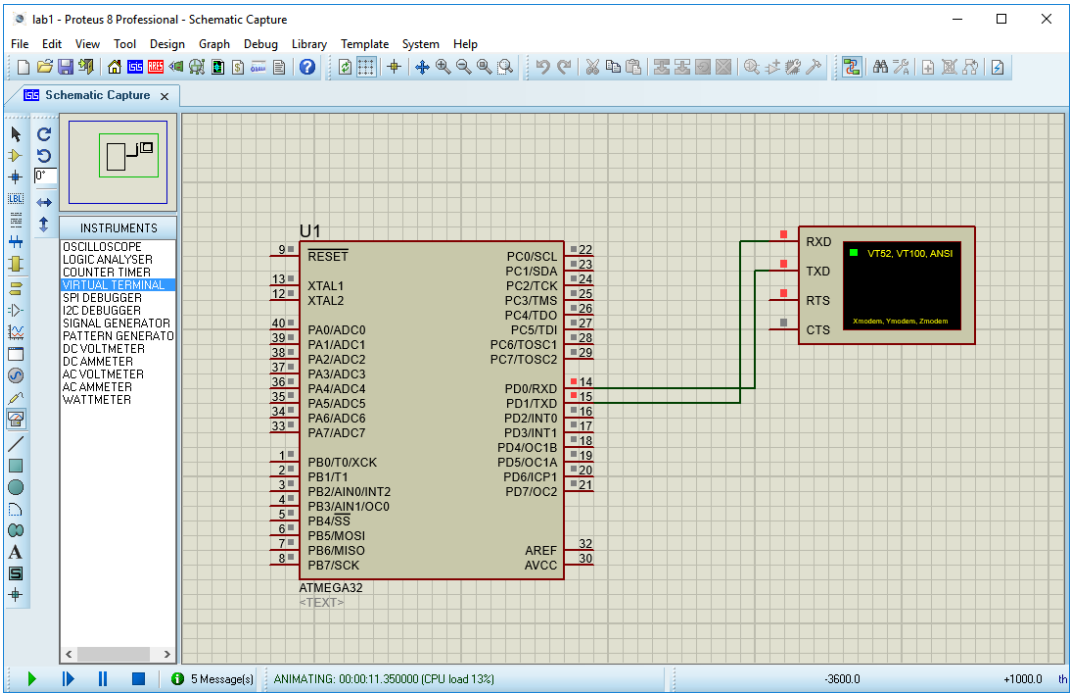

Figure 3: Construction of the scheme - Virtual Terminal Interfacing

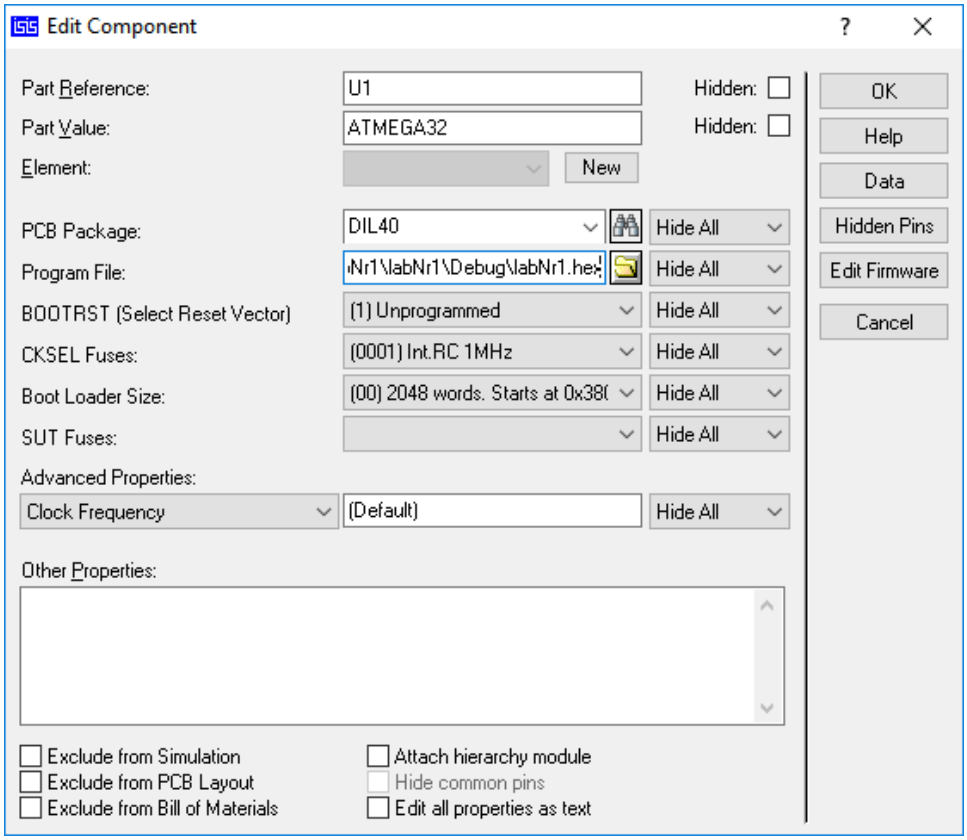

Figure 4: Setting the path to the .hex file - output of the program

Result:

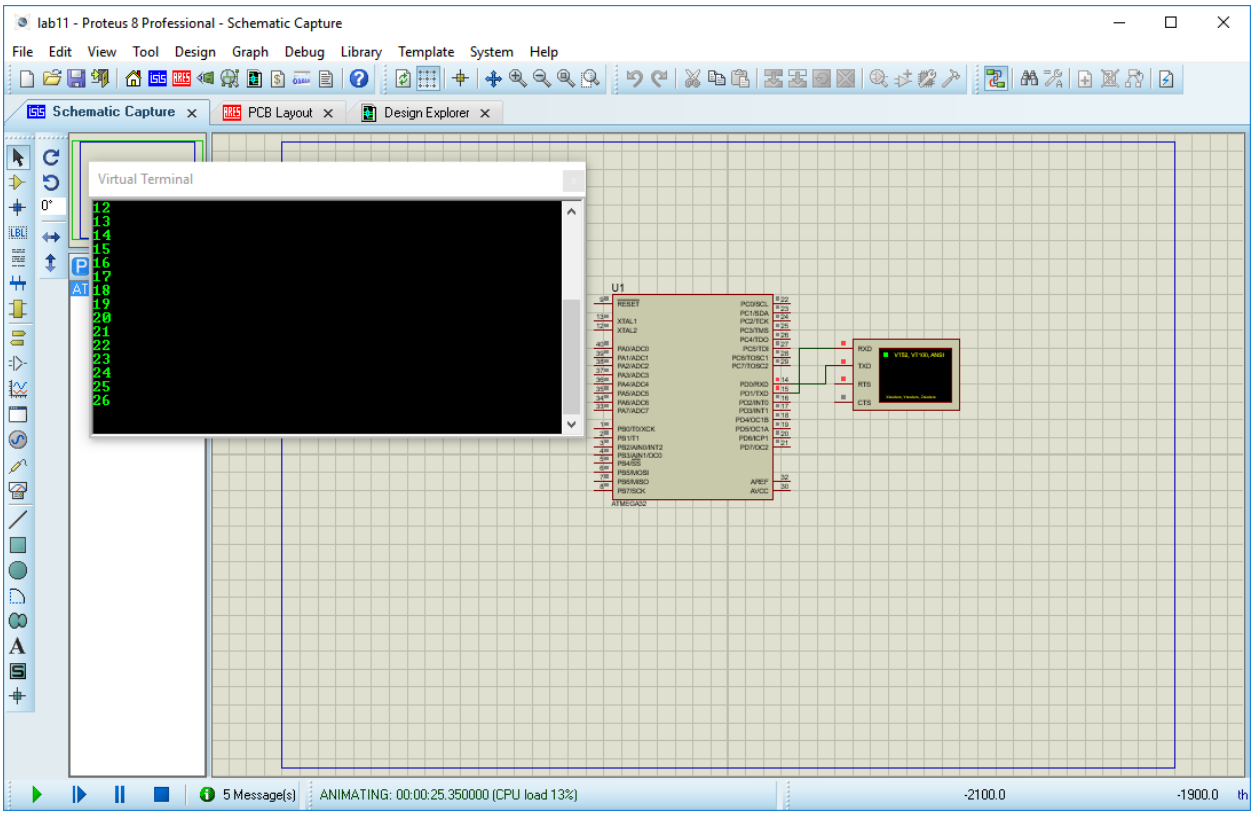

Figure 4: Output of the program

Conclusion:

In this laboratory work I was introduced to what Microcontrollers and Embedded Systems are. I learned some concepts of what a MCU is, how does it work, what is the architecture of MCU and even how it deals with communication - by working with UART. Also, I get to know some tools like Atmel Studio and Proteus, which were very helpful in order to simulate the process and to better understand how MCU works.

Bobliography:

- https://www.futureelectronics.com/en/Microcontrollers
- http://www.engineersgarage.com/microcontroller
- http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter
- http://www.atmel.com/tools/ATMELSTUDIO.aspx

Appendix:

main.c

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

int count = 0;

void main() {
	uartS_Init();

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

uarts.h

```
#ifndef UARTS_H_
#define UARTS_H_

#define UART_BAUD 9600
#define F_CPU 1000000UL

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

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



	#endif /* UARTS_H_ */
```

uarts.c

```c
#include "uarts.h"

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

void uartS_Init(void) {

	stdout = &my_stream;

	#if F_CPU < 2000000UL && defined(U2X)
	UCSRA = _BV(U2X);
	UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;
	#else
	UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;
	#endif
	UCSRB = _BV(TXEN) | _BV(RXEN); }

int uart_PutChar(char c, FILE *stream) {

	if (c == '\n')
	uart_PutChar('\r', stream);

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

	return 0;
}
```