Ministry of Education of Moldova
Technical University of Moldova
Faculty "Computers, Informatics and Microelectronics"

REPORT

Laboratory work No.1
On Embedded systems

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

Performed by st. gr. FAF-141 : Botnari Nicolae

Verified by : 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.

Theory :

Embedded system

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.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available.[5] For example, intelligent techniques can be designed to manage power consumption of embedded systems.[6]

Modern embedded systems are often based on microcontrollers (i.e. CPU's with integrated memory or peripheral interfaces), 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).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

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 for microcontroller unit) is a small computer on a single integrated circuit. In modern terminology, it is a System on a chip or SoC. A microcontroller contains one or more CPUs (processor cores) along with 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 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.

Some microcontrollers may use four-bit words and operate at frequencies as low as 4 kHz, for low power consumption (single-digit milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

Used Resources

Atmel Studio (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

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

UART driver has dependencies on.

#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 and uart_stdio.c

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

void uart_stdio_Init(void);

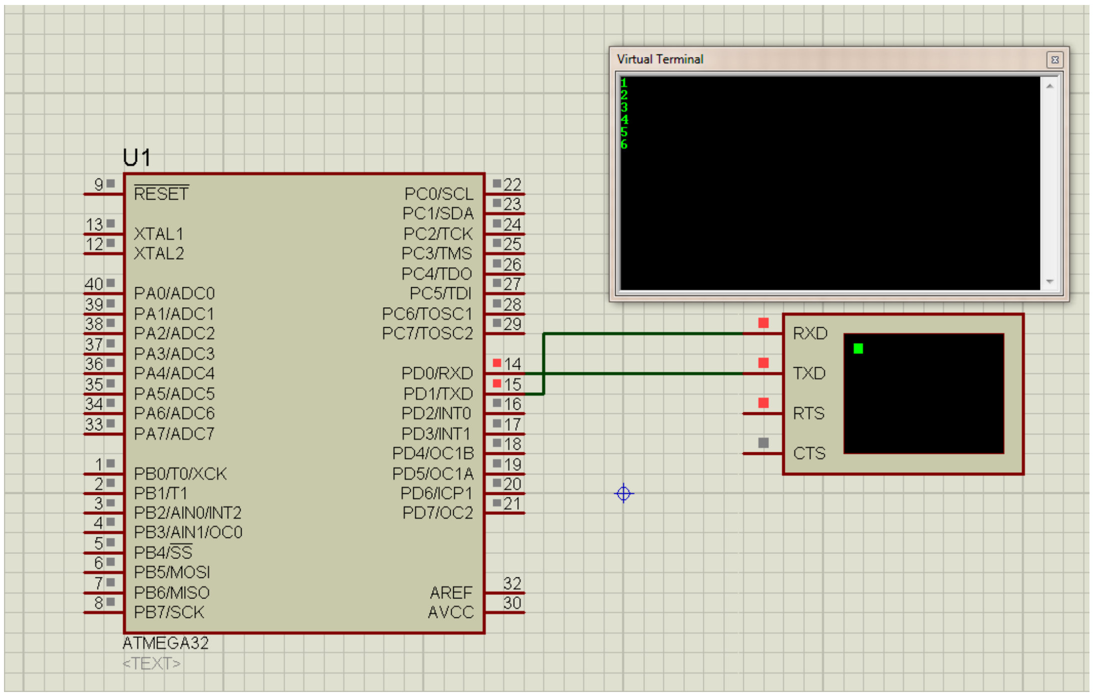
This procedure initializes UART Baud frequency in order to make peripheral device to understand our signals correct.

Int uart_PutChar(char c, FILE *stream);

Function for printing/sending char to peripheral device.

Schematics

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



Conclusion

This laboratory work gave us a basic concepts about embedded system programming and drawing schemes 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

Appendix

Uart_studio.h

```
#ifndef _UART_STDIO_H_
#define _UART_STDIO_H

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

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

#endif
```

Uart_stdio.c

```
#include "uart_stdio.h"
#define UART_BAUD 9600

FILE uart_output = FDEV_SETUP_STREAM(uart_stdio_PutChar, NULL, _FDEV_SETUP_WRITE);

int uart_stdio_PutChar(char c, FILE *stream) {

  if (c == '\a') {
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')
    uart_stdio_PutChar('\r', stream);
  while(~UCSRA & (1 << UDRE));
  UDR = c;

  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 = &uart_output;
}
```

Main.c

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

int count = 0;

void main(void) {
       uart_stdio_Init();

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


}
```