**Microprocessor Systems Design**

**EEE42101**

**Experiment 3: Communication Modules: UART Peripheral and Logging**

# Objectives:

* Put hand on the communication module
* Acquire knowledge about microcontrollers debugging techniques.
* Display the information handled in the microcontroller unit.

# Tools:

1. PC
2. Arduino Nano board
3. Testing board
4. MiniB-USB cable

Note: all material and sources of this course will be available on:

<https://github.com/ashrafmalraheem/Microprocessor_Course>

Feel free to download, study and modify for your own projects.

# Communication Modules:

For a microcontroller to communicate with other devices outside its die, there are several **serial** communication protocols used:

1. USART(UART): **U**niversal (**S**ynchronous) **A**synchronous **R**eceiver & **T**ransmitter.
2. I2C: **I**nter-**I**ntegrated **C**ircuit.
3. SPI: **S**erial **P**eripheral **I**nterface.
4. CAN, I2S, … etc.

In this experiment we are interested only in UART.

* 1. USART:

The USART module is used to communicate between two devices by sending data as a sequence of blocks. Each block has:

1. start bit to indicate the beginning to transmission
2. Data bits vary from 5-9 bits. Most common is 7 bits ASCII.
3. Parity bit, even or odd parity.
4. Stop bit. On or two bits.

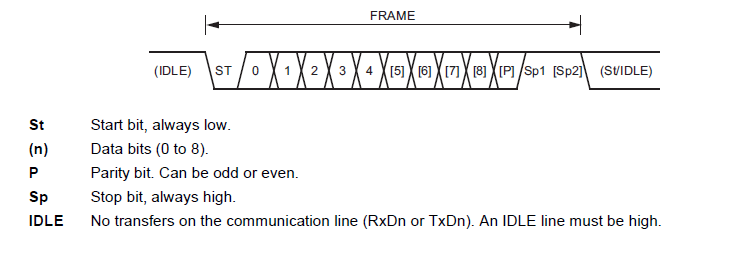


Figure 1 UART Data Frame

User can send any form of data using ASCII format using the USART as sequence of characters. The Tx/Rx pin are usually high (5V or Vcc). When the transmission start, it is driven to low (0V). The duration for the electrical signal to remain high/low to indicate either 1/0 is known as baud rate. Baud rate is the speed of sending one bit. Transmitter and receiver should have the same baud rate, same parity, same no. of data and the same no. of stop bits in order to work correctly. If there is a mismatch in one of these settings the received data will be corrupted.

The USART module has the following characteristics:

* Simple serial communication protocol
* Bidirectional data transfer support full duplex.
* 2 Data lines (Tx and Rx).
* Can work synchronous (needs more wires for clock) and asynchronous.
* Data transmitted in frames blocks (Start bit, Data, parity, Stop bit).
* Data transmission speed called baud rate (110, 300, 600, 1200, 2400 **bps** ‘bit per second’ …etc.) the two devices should have the same baud rate.

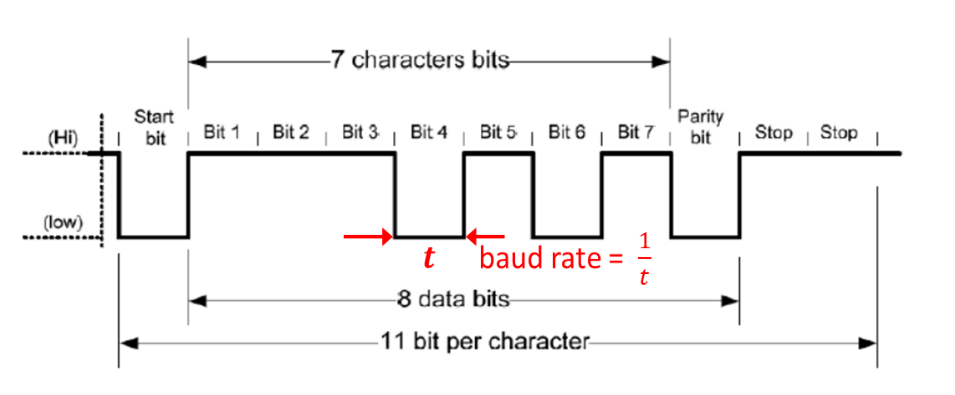


Figure 2 UART Data physical layer

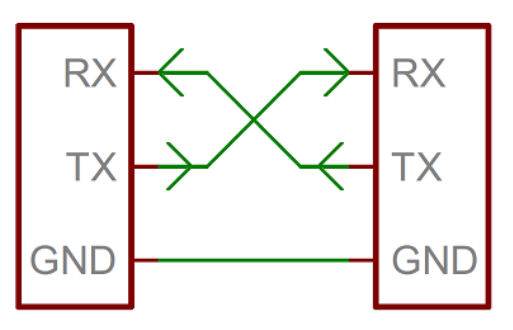


Figure 3 UART Connection[[1]](#footnote-1)

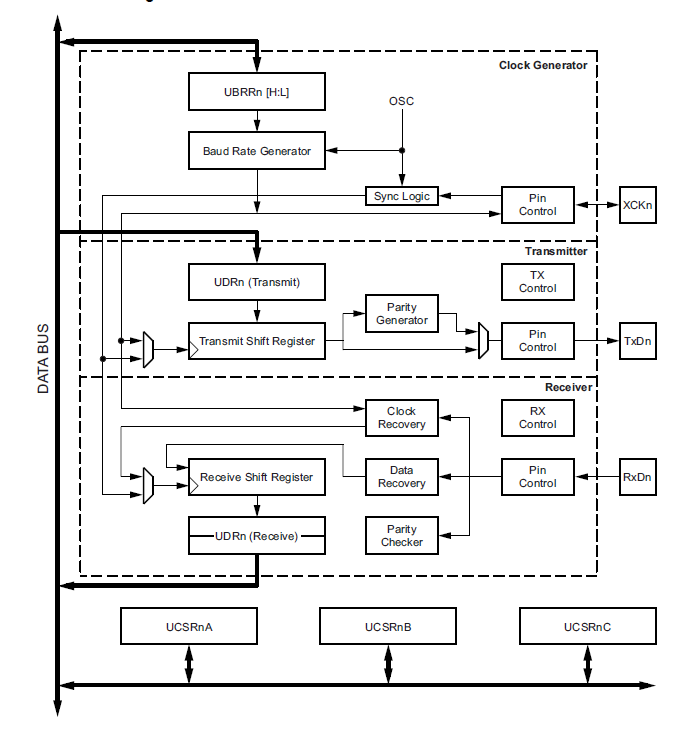


Figure 4 USART Block Diagram ATmega328P[[2]](#footnote-2)

# Part 1: Sending One character using UART

In this experiment, you should try to send one character through UART module in your microcontroller. You need to send: **7-bit characters**, **even parity**, **one stop** bit and baud rate **is 19200 bps**. To activate the UART modules you need to make the following configurations:

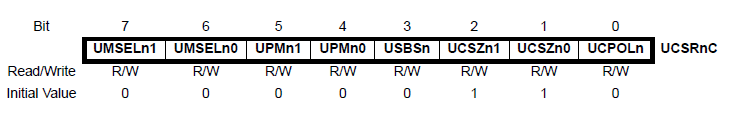


Figure 5 UCSR0C UART control register

1. Set character size through (**UCSZ0**) bits in **UCSR0C** control register:

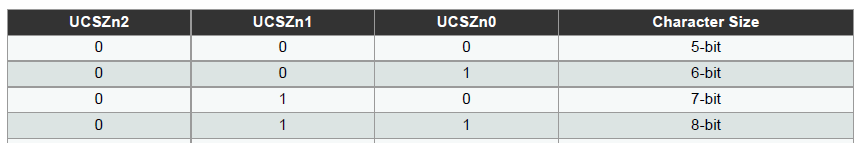


Figure 6 Character size control register

1. Set the parity mode using **UPM0n** bits in **UCSR0C** control register:

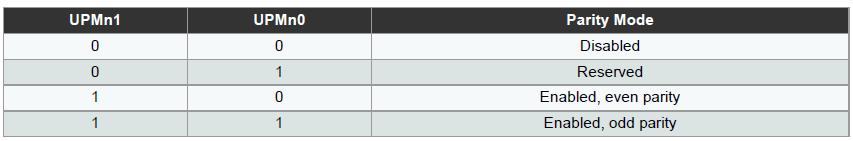


Figure 7 Parity mode control bits.

1. Set no. of stop bits using **USBS0** bits in **UCSR0C** control register:

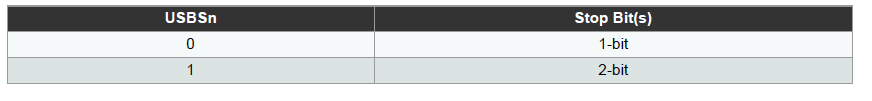


Figure 8 No. of Stop bits

1. Set the baud rate using registers **UBRR0H** and **UBRR0L**:

The formula to calculate baud rate register value:

Then:

// shift to the right by 8 to obtain MSB

1. Finally enable the transmit and receive bits (**RXEN0**, **TXEN0**) in **UCSR0B** register.[[3]](#footnote-3)

|  |
| --- |
| /\*\*\*\* Set Character size \*\*\*\*/  UCSR0C |= ??; // 7 bits  /\*\*\*\* Set Parity mode \*\*\*\*/  UCSR0C |= ??; // even parity  /\*\*\*\* Set Character size \*\*\*\*/  UCSR0C |= ??; // 1 stop bit.  /\*\*\*\* Set Baud rate \*\*\*\*/  UBRR0H= ??; //  UBRR0L = ??; //  /\*\*\*\* Enable Transmit and receive \*\*\*\*/  UCSR0B |= ??; // enable the transmit and receive |

If you set your UART module correctly, you be able to send characters by writing in **UDR0** register. The receive use the same register **UDR0** to store the received data. If a data is received RXC0 bit in **UCSR0A** is set to 1

|  |
| --- |
| char i;  char received;  ‘  ‘  while(1){  /\*\*\*\* send 1 character \*\*\*\*/  UDR0 = i;  /\*\*\*\* receive 1 character \*\*\*\*/  Received = UDR0; // if RXC0 is one  } |

To view the sent data and send one-character use **Real-Term** software in the main folder of your course. Ensure to configure **Real-Term** correctly.

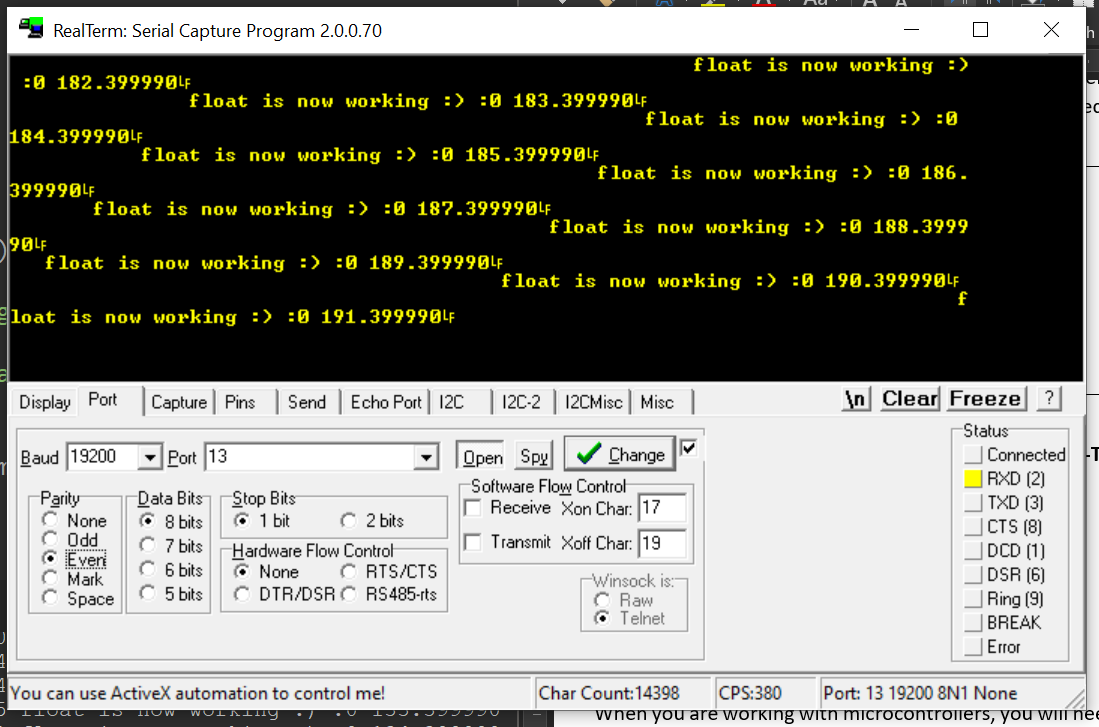


Figure 9 Real-Term interface

# Logging Introduction:

When you are working with microcontrollers, you will need to view the information that you are handling. The first and primitive method is by using simple interface (I/O) to view the information as digital info (0 or 1) e.g. by blinking and LED. Or another method is by switching the LED ON for a certain amount that is proportional to the value (pulse width). Or the number of blinks is the number you want to view. You can improvise many other methods to view your information inside the microcontroller.

However, in simple application the prementioned suggestions may work. But in real life applications there are many different information to show like integers, float numbers and characters which is inconvenient to translate them into electrical or time series signal. The most common methods to view data inside microcontroller are: displays, debuggers, logging, ...etc.

* 1. Displays:

have many types: seven segments, dot matrix and many other. Some could only display characters, graphs and other display only numbers. The drawback of using LCD is it is limited space of display. Therefore, old data should be cleared to display new ones. No record could be saved.

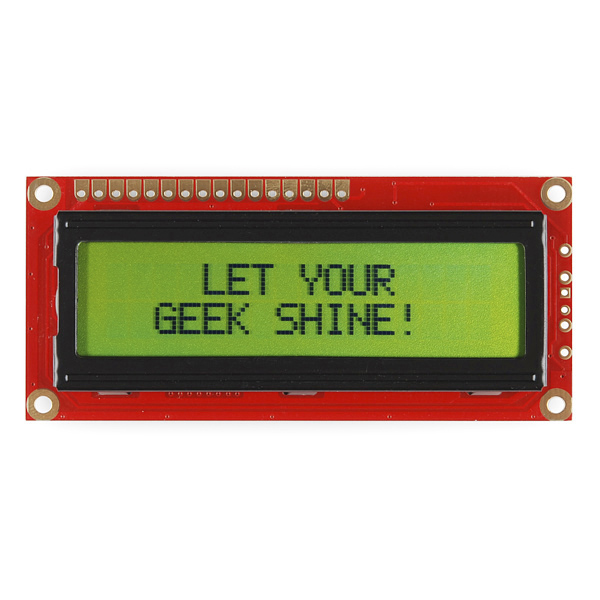


Figure 10 16x2 Character LCD

* 1. Logging

is used either by saving the data in the memory or sending them to a server (or any device with more advance capabilities). The data are saved in log files. A timestamp could be created to each log message, so a track of the data could be saved.

* 1. Debuggers

it is a method used while developing the software of the microcontroller. It needs a hardware tools as well as software. Additional instructions are added by the compiler to read/write every register inside the microcontroller. The hardware reads the data in the registers and send them to the debugging application. Most IDEs have special interface to view the variables and registers and many other peripherals inside the microcontroller. Most common methods for debugging are ICSP (in circuit serial programming), JTAG (**J**oint **T**est **A**ction **G**roup), boundary scan, etc...

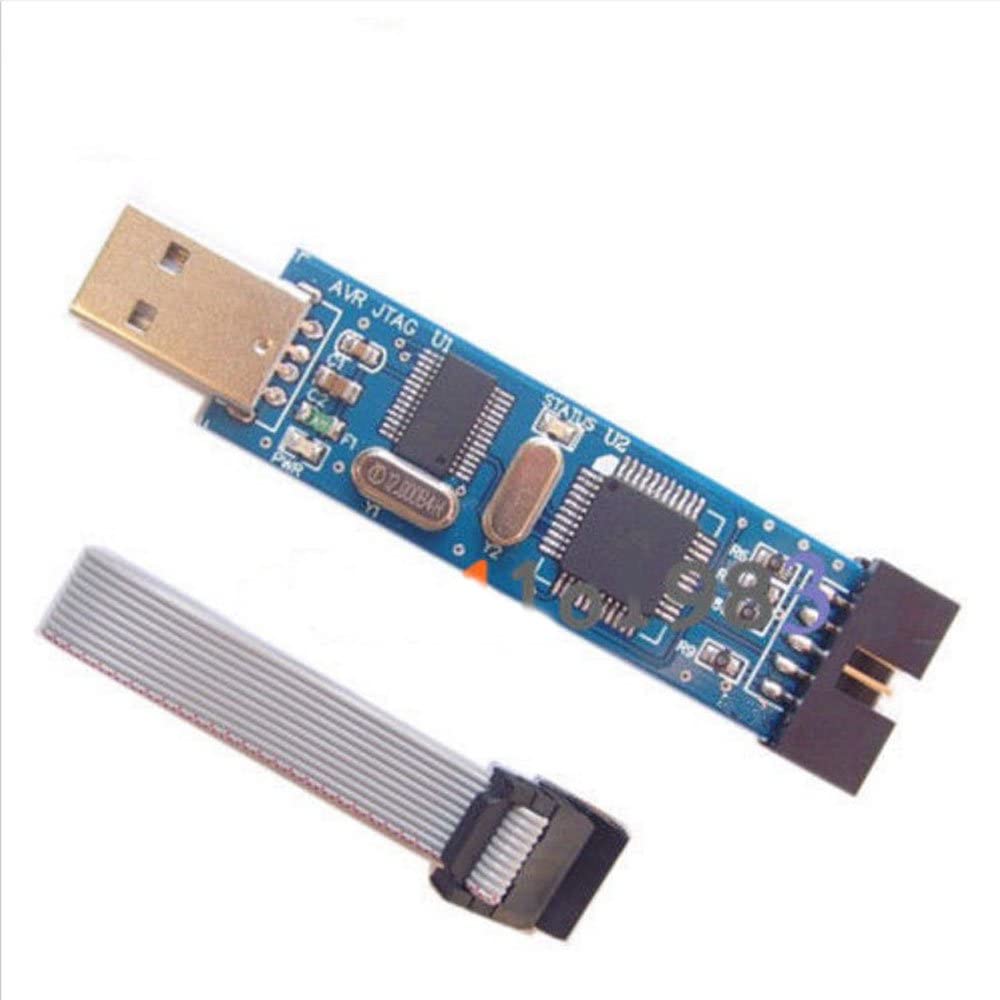


Figure 11 Hardware debugger

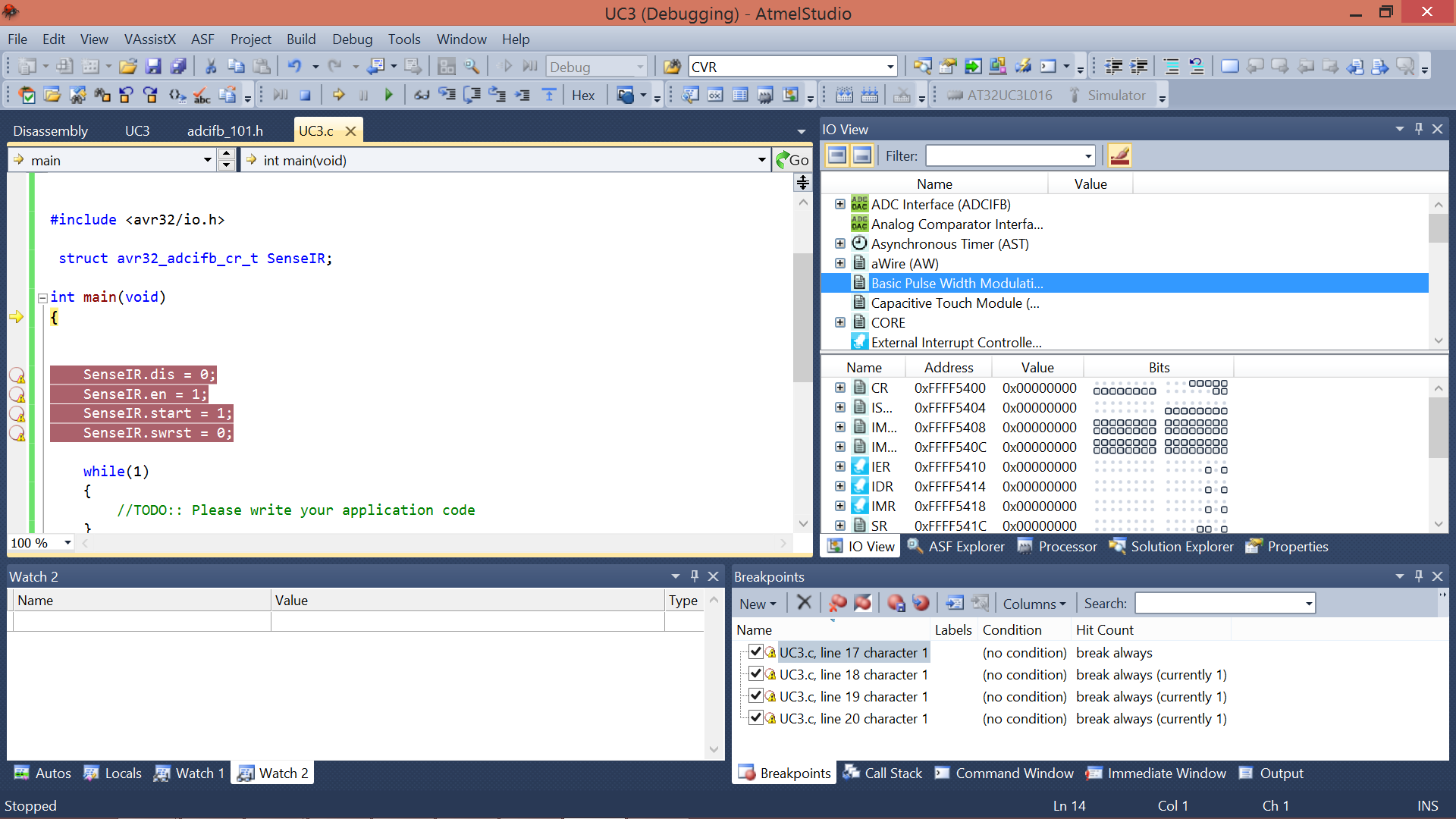


Figure 12 IDE debugger example

The debugger holds the execution of the program and wait until it reads the whole registers in the microcontroller unit. Therefore, it has a draw back when debugging Realtime applications and signals.

\*this method is beyond our course reach.

# Part 2: Log data into PC:

After configure the UART correctly in the previous part. Now you use UART to log the data into PC. All you need is to call Log functions that you will find on **LogMessage.h** use the one appropriate to your application.

Run application **Logger.exe** which will display the logs and store them in file.log.

**Note:** You should call log messages after you set your UART correctly.

**Note:** You should include **LogMessage.h** into your main

1. <http://www.engineering.union.edu/~hodgsond/MER421/Winter%202015/Lectures/MER421Serial.pdf> [↑](#footnote-ref-1)
2. ATmega328P data sheet. [↑](#footnote-ref-2)
3. You can refer to the example code in section **19.5** of the data sheet, to ease the initialization. [↑](#footnote-ref-3)