EECS 388 Lab #3

Universal Asynchronous Receiver/Transmitter (UART)

In this lab, we take a deeper look into our hardware platform, the Arduino Uno ATMEGA328P board, including its CPU architecture, memory map, and external GPIO pin map. We will implement the **uart_init**() and **`ser_read()`** function, to initialize and read data from the UART connection.

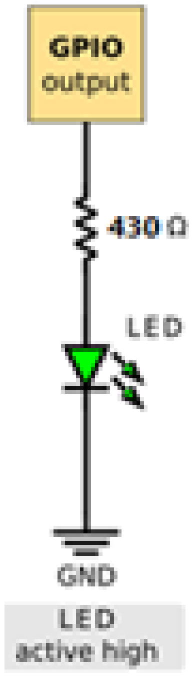
Task 0: Download starter code, setup circuit, read UART serial documentation

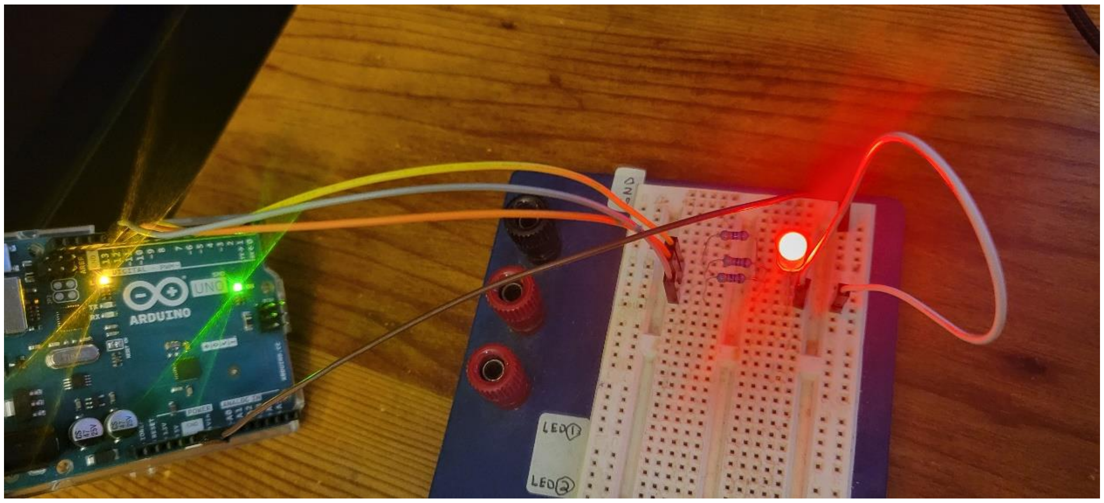
Task 0.0 Download starter code

Log into Canvas and go to the EECS 388 course page. Go to "Files>Labs" and under "Lab 3" click on **lab03.zip** to download the lab source code. Extract the files into a folder on your PC and open the folder in **VSCode**, same as with Lab 2.

Task 0.1 Setup circuit from lab02

Setup the same active-high GPIO LED circuit that you used with lab 2. Refer to the picture and schematic for setup. Once you have the circuit setup, review it with your GTA before turning on the Arduino Uno.





Task 0.2 Read UART serial documentation

To complete this project, you need to understand a little more about the UART hardware. Let's first open the "Serial Port Guide (`lab03/docs/Serial_Port_ATMega328P.pdf`). This document includes lots of helpful board-specific information. Review section 2.1 about commonly used baud rates. Read section 2.2 about data bits, parity bits, stop bits, and start bit. **Read section 4** about serial communications with the ATmega 328P, especially the equation for calculating UBRR which will help set the baud rate. Note that our Arduino has a 16MHz clock (16x10^6) for f_{osc}.

$$\mathrm{UBRR} = \frac{f_{osc}}{16 \times BAUD} - 1$$

The full documentation for the ATMega328P will also be helpful for understanding this lab. It is located in **lab03/docs/ Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf**. Here is the top level block diagram of the microcontroller:

Figure 2-1. Block Diagram

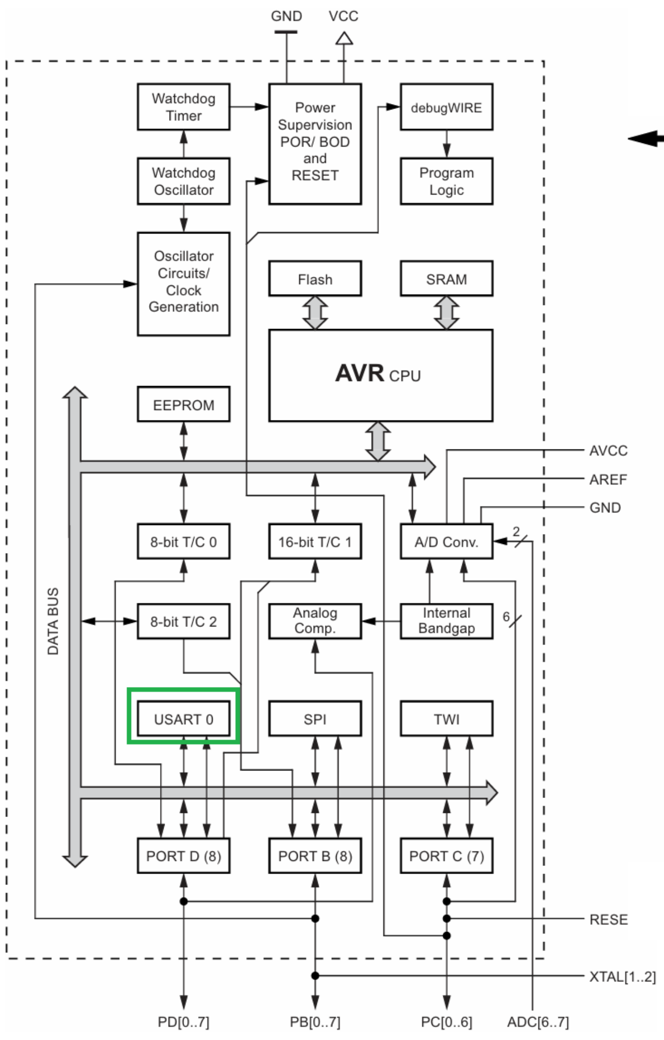
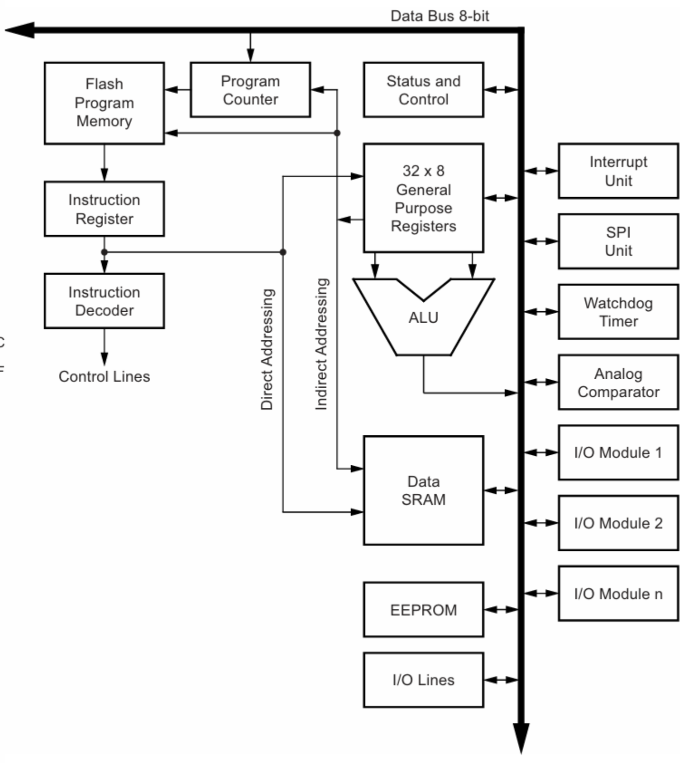
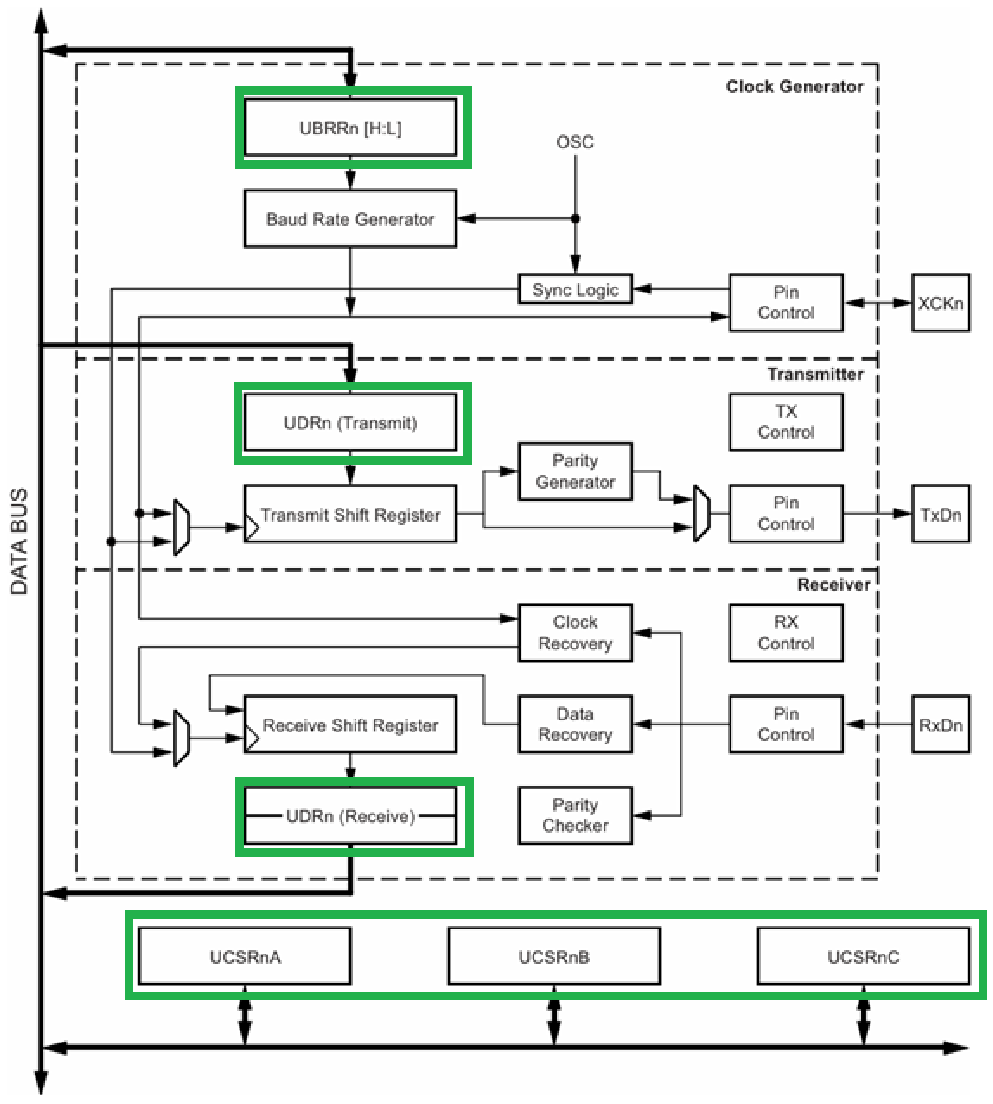


Figure 6-1. Block Diagram of the AVR Architecture



The block diagram shows the USART0, which is the UART we will be using. **Chapter 19 in the datasheet** will help us understand more about the data and control registers available for USART0.

Figure 19-1. USART Block Diagram[(1)]



We need to configure the control registers and be able to use the data register to send and receive. These topics are covered in chapter 19.10 of the datasheet. These are the registers we will need for lab03:

- **URD0** – 8-bit Data register (used for RX/TX) – Holds byte to transmit/receive
- **UCSR0A** – 8-bit Control register A
    - RXC0 – bit 7 flag used to check if something is received in URD0

        - TXC0 – bit 6 flag used to check if transmit complete, code uses UDRE0 (bit 5) to see if the data register is empty and ready to accept a new character
    - **UCSR0B** – 8-bit Control register B
        - RXEN0 – bit 4 – Set to enable serial to receive
        - TXEN0 – bit 3 – Set to enable serial to transmit
    - **UCSR0C** – 8-bit Control register C
        - UCSZ00/UCSZ01 – bit 1/2 – Used to set data size to 8-bits
    - **UBRR0H/UBRR0L** – 8-bit each - Used for configuring baud rate

19.10.1 UDRn – USART I/O Data Register n

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

19.10.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

19.10.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

19.10.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

19.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Task 1: UART init/read/write functions

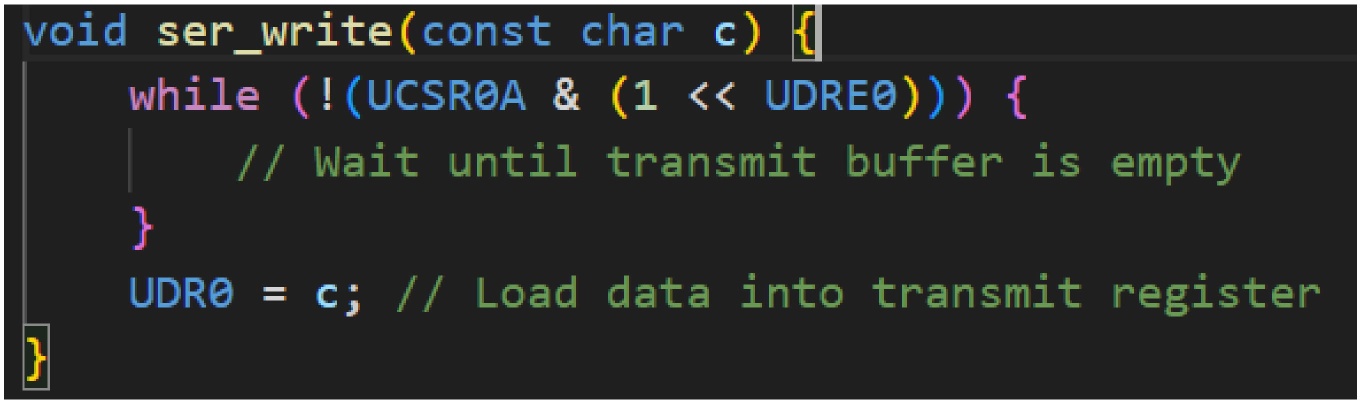
Task 1.0 Setup the baud rate in the uart_init()
Calculate the 'ubrr' value needed for 9600 baud based on the formulate in Task 0. Set the value in the code, and uncomment the line of code, so it will be set in the UBRR0H and UBRR0L registers.

Task 1.1 Enable the UART receive in uart_init()
Currently, the UCSR0B control register is setup to enable transmit. We need to modify the code to enable both transmit and receive at the same time. We need both bit 3 and bit 4 to be "set" to 1 in the 8-bit value. The resulting value will be equivalent to this binary value: 0b0001_1000. You can accomplish this with a combination of bitwise or '|', shift left '<<' operators, or by setting a decimal value directly to UCSR0B directly. See how USCR0C is setup for an example of how to set multiple bits in a register.

Task 1.2 Review ser_write() and ser_printline()

- ser_write() has already been provided as part of the EECS388 library (**src/eecs_388_lib.c, src/eecs_388_lib.h**). The implementation uses a blocking call, which will wait until the transmit buffer is empty before loading a new value to transmit into the UDR0 data register.
- ser_printline() is used to print a string of characters using the ser_write() function.

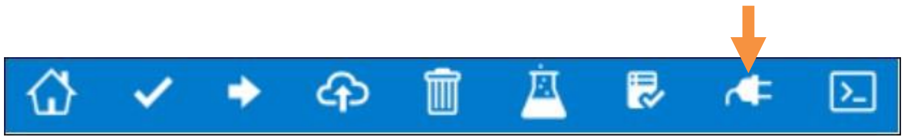
Task 1.3 Implement ser_read()
Review the implementation of ser_write() for hints. This function will read UCSR0A control register and check the UDRE0 bit to see if the transmit buffer is empty. Once it's empty, it will load a single paramater 'c' into the UDR0 data register, which will then transmit it over the UART.



```
void ser_write(const char c) {
    while (!(UCSR0A & (1 << UDRE0))) {
        // Wait until transmit buffer is empty
    }
    UDR0 = c; // Load data into transmit register
}
```

For ser_read(), we will do a similar task, reading the UCSR0A control register, but this time we want to check if there is a received character in the UDR0 data register. Look through section 19.10.2 in the datasheet to see which bit we can check that the USART Receive has been completed and finish the implementation of ser_read().

Note that to open the serial terminal in your PC, click the PlatformIO: Serial Monitor icon, which is the 2nd to last in the toolbar at the bottom left.



Task 1.4 Implement the logic for the LEDs
After you receive a character, implement the logic to turn on the correct LED in the loop() function in eecs_388_uart.cpp to complete the lab.

Once you correctly implement the `ser_read()` function, you can type ‘r’ or ‘g’ or ‘b’ characters over the serial terminal in your PC to enable red, green, and blue LEDs respectively.

Task 1.5 Implement ser_read() using pointers and pointer operators (Optional Bonus) – 1 additional point
Modify your ser_read() function to replace the UCSR0A and UDR0 macros to use pointers, dereference operators, and memory mapped IO addresses. Refer to Section 30 Register Summary in the ATmega328P datasheet and the appendix below for additional information

Submission – 40 points (Max 42/40 with bonus)

Milestone: To obtain the 10% (4 points) for this lab, you must be able to receive the text from the Arduino board on the linux machine asking the user to “type 'r' or 'g' or 'b': “.

Demo: To obtain the 40% (16 points) for this lab, you must show your GTA the working ser_read() function, as well as the proper LEDs lighting up when typing the character. You also need to be able to answer some of the following questions:

Possible questions:

- How did you calculate the UBRR value for 9600 baud?
- How many stop bits are used? 1 or 2?
- Is it even, odd, or no parity?
- What is the address in memory for the receive data register for UART0?
- How wide is the receive data field? How much data can be read each time?
- What bit is set in USCR0B to enable the serial port to be able to receive?
- What bit do we have to check in the USCR0A register to see if a byte is ready to be read in ser_read()?

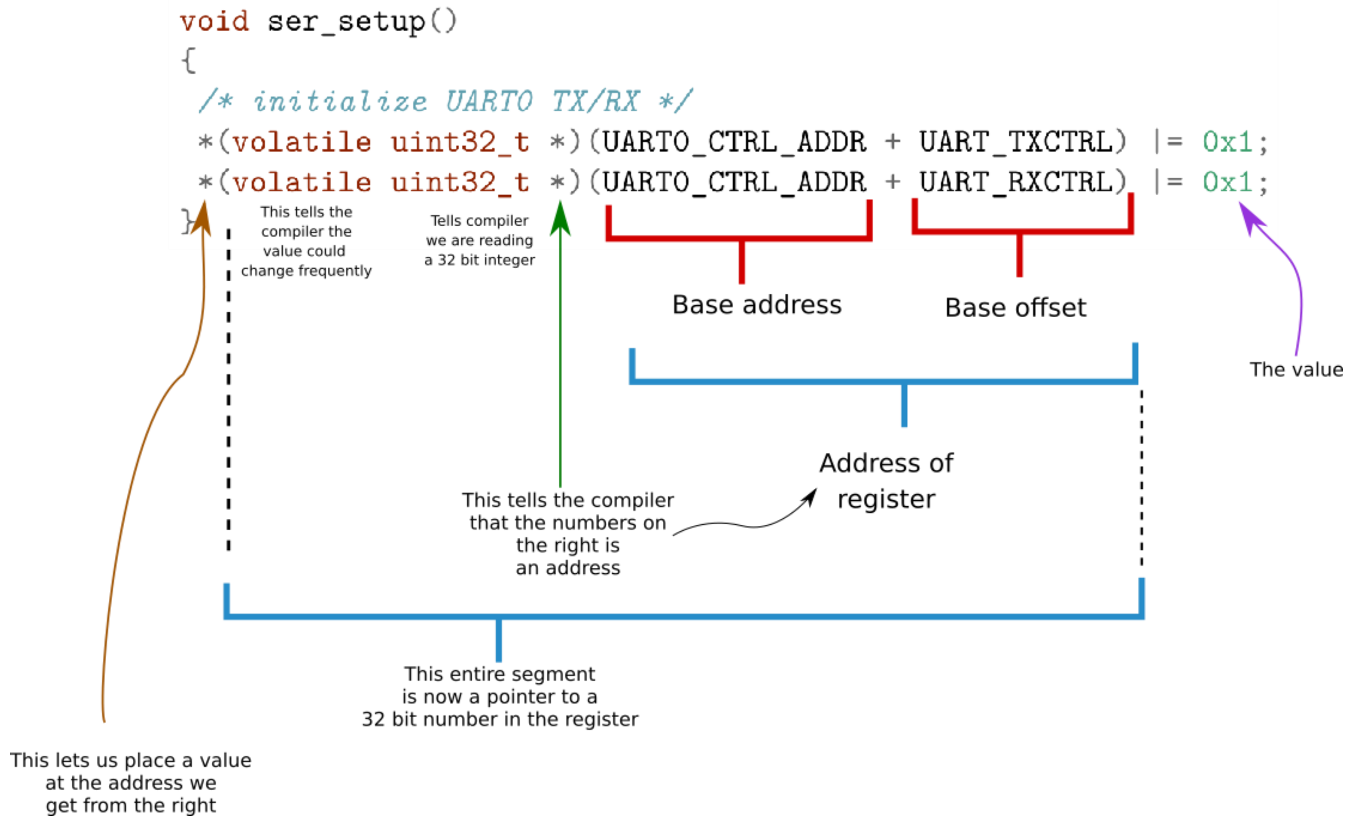
Code: To obtain the 50% (20 points), go to the "Assignments" section of the Canvas and submit your modified **eecs_388_lib.c** and **eecs_338_uart.cpp** file to the Lab 03 submission link that corresponds to your lab section. Also, make sure that you have shown your completed work to your respective GTA for a demo.

Screenshots/PDF/Text files will **NOT** be allowed/graded as submissions. You need to submit the modified **.c** file **only**.

Bonus:
To obtain the 2.5% (1 point) you need to submit the files to Canvas by the end of your lab period. You must demo your working code on the board to your GTA before submitting the code to canvas.
To obtain the 2.5 % (1 point) you need to directly use pointer addresses described in task 1.5 instead of the provided macros.

Appendix

Here is how to understand the code which accesses the register memory. The following example ***writes** a* value to the given memory addressed register using the pointer dereference operator:



```
void ser_setup()
{
 /* initialize UART0 TX/RX */
 *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXCTRL) |= 0x1;
 *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_RXCTRL) |= 0x1;
}
```

This tells the compiler the value could change frequently
Tells compiler we are reading a 32 bit integer
Base address
Base offset
The value
Address of register
This tells the compiler that the numbers on the right is an address
This entire segment is now a pointer to a 32 bit number in the register
This lets us place a value at the address we get from the right

To ***read*** a value, we place the dereferencing on the right-hand side of the **=** sign. Example:

```
char c;
c = (*(volatile uint8_t *)(mem_addr))
```