Topic 10: Communicate Serial Communication

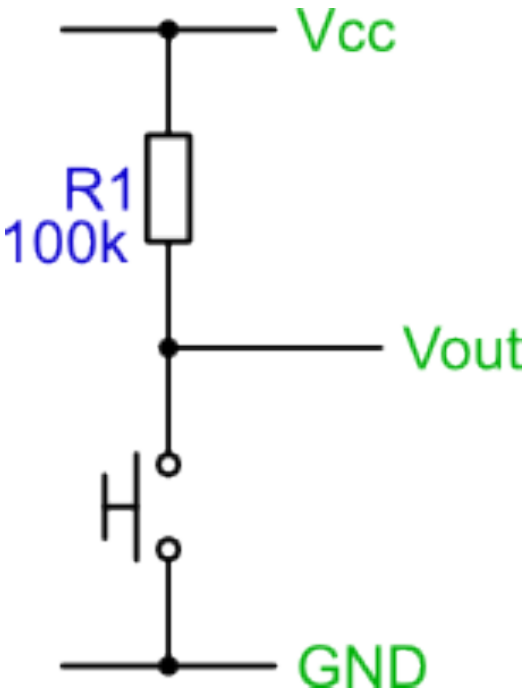
CAB202. Topic 10

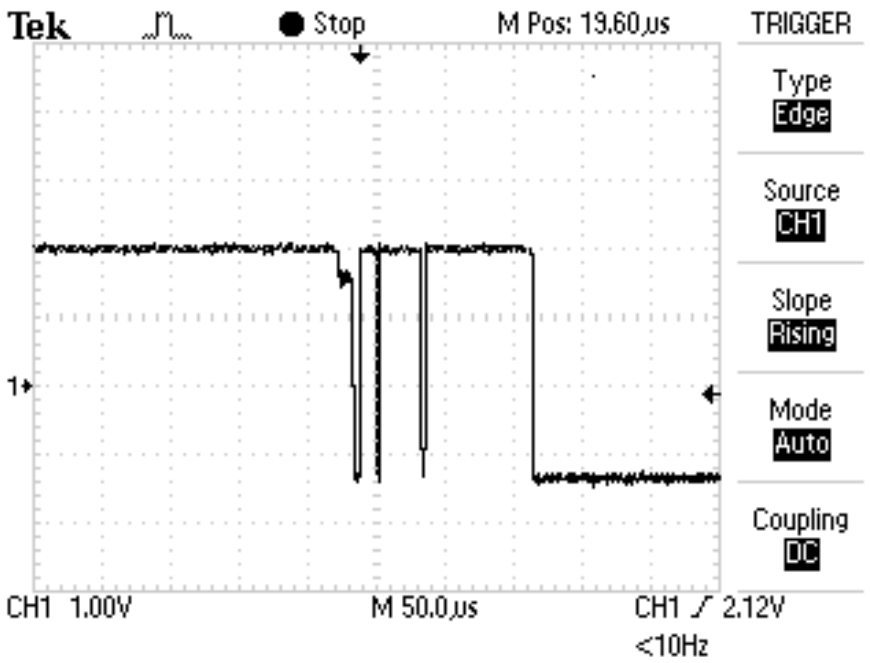
Luis Mejias

Outline

- Revisit Topic 9
- Serial communication on the Atmega32U4
- Virtual USB with the Atmega32U4
- Examples

Bouncing/Debouncing





Debouncing

```
#include <stdint.h>
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include <cpu_speed.h>

#include <graphics.h>
#include <macros.h>

void setup(void) {

}

void process(void) {
}

int main(void) {
}
```

```
uint16_t counter = 0;

void process(void) {
        // Detect a Click on left button
        if ( BIT_IS_SET(PINF, 6) ) {
                while ( BIT_IS_SET(PINF, 6) ) {
                        // Block until button released.
                }
                // Button has now been pressed and released...
                counter ++;
        }

        // Display and wait if joystick up.
        if ( BIT_IS_SET(PIND, 1) ) {
                draw_all();

                while ( BIT_IS_SET(PIND, 1) ) {
                        // Block until joystick released.
                }
        }
}
```

Code for topic 9 provides versions of a program with different ways of debouncing buttons.

- BounceDemo
- DelayDebounceDemo
- NonblockingDebounceDemo

Timers

- Our microcontroller has four timers, Timer 0, Timer 1, Timer 3, and Timer 4
- Each timer is associated to a counter and a clock signal.
- The counter is incremented by 1 in every period of the timer's clock signal
- The clock signal can come from
  - The internal system clock
  - An external clock signal

Timers
How often does the timer overflow?

- Clock speed (8MHz)
- Prescaler (1, 8, 64, 256, 1024)
- Counter size (8, 10 or 16 bit) = 2^8, 2^10, 2^16

- Timer 0: Normal timer mode, Prescaler of 1024, 8 Bit timer
- Timer Speed
  = 1 / (Clock Speed/Prescaler)
  = 1 / (8000000/1024)
  = 128 micro seconds (sometime also called timer resolution)

- Timer Overflow Speed
  = (Timer Speed * 256)
  = 32768 micro seconds

Timers/Interrupts

Steps:

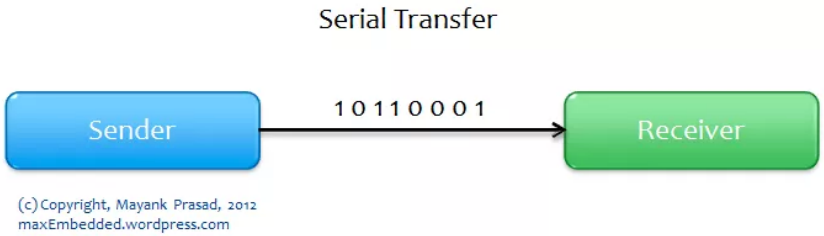
1. Set up the timer with correct prescaler.
2. Turn on interrupts
3. Write the Interrupt Service Routine that is called when the timer overflows. Data shared between the ISR and your main program **must be both volatile and global in scope** in the C language. i. e `volatile int overflow_counter;`

```
#define FREQ     (8000000.0)
#define PRESCALE (1024.0)

void setup(void) {
        CLEAR_BIT(TCCR0B,WGM02);
        SET_BIT(TCCR0B,CS02);
        CLEAR_BIT(TCCR0B,CS01);
        SET_BIT(TCCR0B,CS00);

        // Enables the Timer Overflow interrupt for Timer 0
        SET_BIT(TIMSK0, TOIE0);

        // Enable global interrupt
        sei();
}

volatile int overflow_counter = 0;

ISR(TIMER0_OVF_vect) {
        overflow_counter ++;
}

void process(void) {
        double time = ( overflow_counter * 256.0 + TCNT0 ) * PRESCALE  / FREQ;
}

int main(void) {
        setup();

        for ( ;; ) {
                process();
        }
}
```

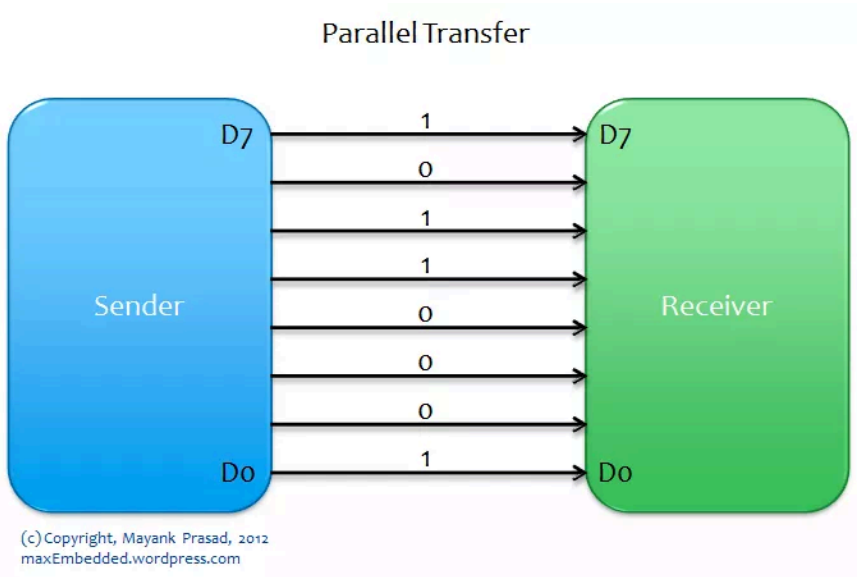
Communications - Overview

- Two different ways to communicate
  - Serial: a single bit is transferred one at a time
  - Parallel: multiple data bits are transferred simultaneously
- Will often come across the following for serial communication
  - Shift registers: convert a byte to/from serial bits and vice versa
  - Modems: convert serial bits to/from audio

Communications - Overview



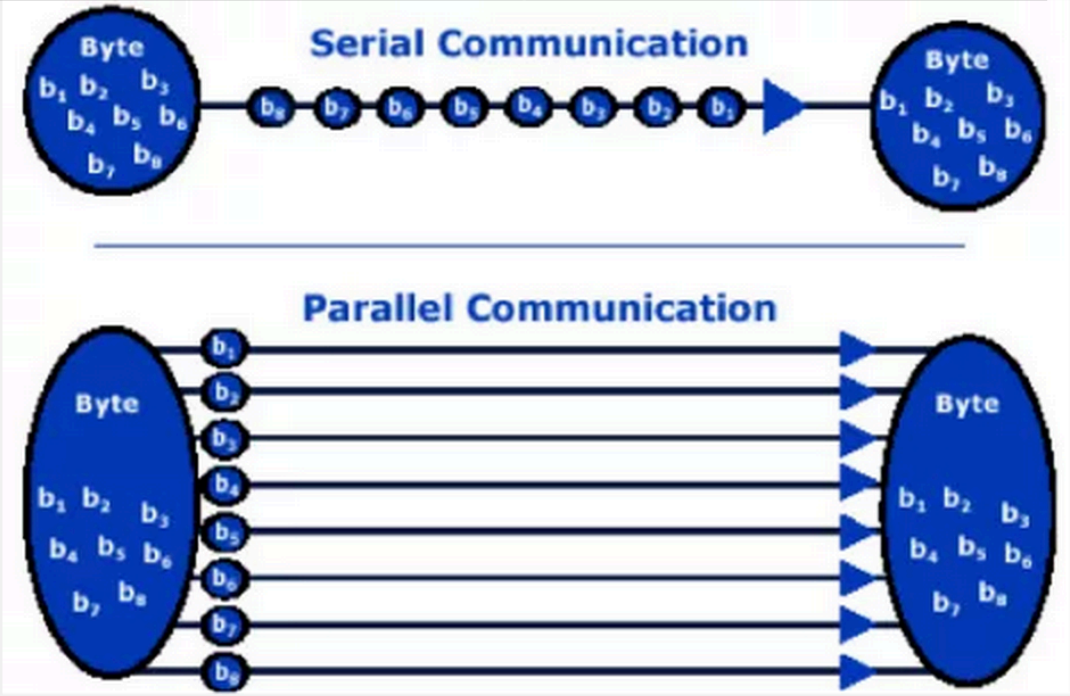
Data is sent/received in one bit at the time.



Data is sent/received in multiple bits at the time through multiple channels

Communications - Overview

Serial Communication	Parallel Communication
1. One data bit is transceived at a time	1. Multiple data bits are transceived at a time
2. Slower	2. Faster
3. Less number of cables required to transmit data	3. Higher number of cables required



Serial vs Parallel

Communications - Overview

- Some serial communication protocols have been developed over the past few decades. These are:

1. SPI – Serial Peripheral Interface
   - Is the three-wire based communication system. One wire each for master to slave and vice-versa, and one for clock pulses. There is an additional SS (slave select) line, which is mostly used when we want to send/receive data between multiple ICs.
2. I2C – Inter-Integrated Circuit
   - It has 2 wires, one for clock and the other is the data line which is bi-directional. Transmission speed 400kHz. Often called Two Wire Interface (TWI).
3. Firewire
   - Developed by apple for high-speed communication and isochronous real-time data transfer. The bus may contains a number of wires, 4-pins, 6-pins, or 8-pins.

Communications - Overview

4. Ethernet
   - Used mostly in LAN connections, the bus consists of 8 lines, or 4 Tx/Rx pairs.
5. Universal Serial Bus (USB)
   - It’s a four-wire physical wire system with two power lines and two data lines. There are no physical control lines. Initial speed was 12 Mbps, increased to 480 Mbps with USB2 and up to 5 Gbps "Superspeed" mode with USB3.
6. RS232
   - The RS-232 is typically connected using a DB9 connector, which has 9 pins, out of which 5 are input, 3 are output, and one is Ground. You can still find this so-called “Serial” port in some old PCs.

Serial comms. on the ATMega32

- ATmega32 provides four subsystems for serial communications.
  1. Serial Peripheral Interface (SPI)
  2. Two-wire Serial Interface (TWI)
  3. Universal Serial Bus (USB)
  4. Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

Serial Peripheral Interface (SPI)

- The receiver and transmitter share a common clock line.
- Supports higher data rates.
- Example of devices using SPI: liquid crystal display, analogue to digital converter

https://www.pjrc.com/teensy/sd_adaptor.html
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

Serial Peripheral Interface (SPI)

- We cannot use SPI to drive our LCD, because SPI runs on pins B0, B1, B2, and B3, and we have already connected these pins to other digital IO (B0 and B1 are connected to the joystick; B2 and B3 are connected to LEDs).

- Under SPI, one device is in charge: the *master*.
  - The master device manages interaction with one or more *slave* devices.
  - Where multiple slaves are controlled, each is allocated a numeric address which is used to ensure transmissions go to the correct receiver.

Two-wire Serial Interface (TWI)

- Network several devices such as microcontrollers and display boards, using a two-wire bus.
- Up to 128 devices are supported.
- Each device has a unique address and can exchange data with other devices in a small network.
- Equivalent to I2C

Synchronous vs. Asynchronous

- Synchronous serial communications
  - Sender and receiver clocks are synchronised
  - Faster transfer
  - Eg. Serial Peripheral Interface (SPI)
- Asynchronous serial communications
  - No synchronisation between sender and receiver
  - Bytes are enclosed between start and stop bits
  - Eg. RS232, USART on ATmega32
- Blocking or non-blocking

Virtual USB with Atmega32U4

- The Atmega32U4 has a USB 2.0 Full-speed/Low Speed Device Module with Interrupt on Transfer Completion.

- **Complies fully with Universal Serial Bus Specification Rev 2.0.**
  - http://www.usb.org/developers/docs/

Virtual USB with Atmega32U4

- USB: Virtual Serial Port
  - This library implements a virtual serial port compatible with most serial emulators. This library provide functions to:
  - Receive
    - Receive characters, check whether data new has been received
  - Transmit
    - Single characters, strings,
  - Set Parameters
    - Baud, stop bits, parity, etc..
  - Management
    - Initialise, check configuration status

Virtual USB with Atmega32U4

- USB: Virtual Serial Port: Brief overview
  - setup
    - void usb_init(void); // initialize USB port,
    - uint8_t usb_configured(void); // is the USB port configured
  - receiving data
    - int16_t usb_serial_getchar(void); // receive a character
      // (-1 if timeout/error)
    - uint8_t usb_serial_available(void); // number of bytes
      // in receive buffer
    - void usb_serial_flush_input(void); // discard any
      // buffered input

Virtual USB with Atmega32U4

- USB: Virtual Serial Port: Brief overview
  - transmitting data
    - int8_t usb_serial_putchar(uint8_t c); // transmit a character
    - int8_t usb_serial_putchar_nowait(uint8_t c); // transmit a character, do not wait
    - int8_t usb_serial_write(const uint8_t *buffer, uint16_t size); // transmit a buffer
    - void usb_serial_flush_output(void); // immediately transmit any buffered output
  - serial parameters
    - uint32_t usb_serial_get_baud(void); // get the baud rate
    - uint8_t usb_serial_get_stopbits(void); // get the number of stop bits
    - uint8_t usb_serial_get_paritytype(void); // get the parity type
    - uint8_t usb_serial_get_numbits(void); // get the number of data bits
    - uint8_t usb_serial_get_control(void); // get the RTS and DTR signal state
    - int8_t usb_serial_set_control(uint8_t signals); // set DSR, DCD, RI, etc

Virtual USB with Atmega32U4

- Suggested structure of a program
  - Initialise usb - usb_init()
  - Check whether is ready and configured – usb_configure()
  - Wait for the other side to indicate is ready to receive - usb_serial_get_control()
  - Start receiving characters - usb_serial_getchar()
  - Consider non-blocking reading - usb_serial_available()
  - Include usb_serial.h and compile usb_serial.c together with your program.

  Example: serial_led.c

Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

- Supports full-duplex mode between a receiver and transmitter.
- UART stands for Universal Asynchronous Receiver/Transmitter. From the name itself, it is clear that it is asynchronous i.e. the data bits are not synchronized with the clock pulses.
- USART stands for Universal Synchronous Asynchronous Receiver/Transmitter. This is the synchronous type, i.e. the data bits are synchronized with the clock pulses.
- They are basically just a piece of computer hardware that converts parallel data into serial data. The only difference between them is that UART supports only asynchronous mode, whereas USART supports both asynchronous and synchronous modes.

Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

- The USART of the AVR can be operated in three modes
  - Asynchronous Normal Mode
  - Asynchronous Double Speed Mode
  - Synchronous Mode

Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

- Asynchronous Normal Mode
  - The data is transmitted/received asynchronously, i.e. we do not need (and use) the clock pulses.
  - The data is transferred at the BAUD rate we set in the UBBR register. This is similar to the UART operation

Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

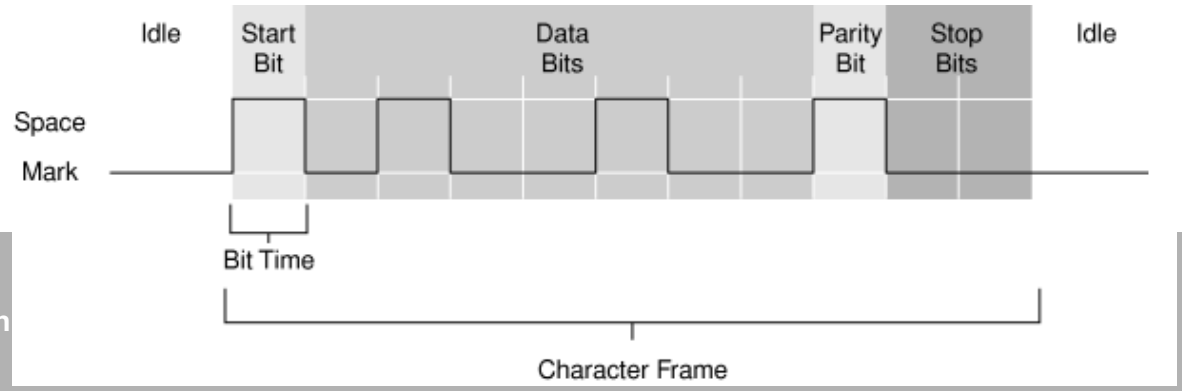
- Asynchronous Double Speed Mode
  - This is higher speed mode for asynchronous communication. In this mode also we set the baud rates and other initializations similar to Normal Mode. The difference is that data is transferred at double the baud we set in the UBBR Register.
  - Setting the U2X bit in UCSRA register can double the transfer rate

Universal Synchronous & Asynchronous Serial Receiver and Transmitter (USART)

- Synchronous Normal Mode
  - This is the USART operation of AVR. When Synchronous Mode is used (UMSEL = 1 in UCSRC register), the XCK pin will be used as either clock input (Slave) or clock output (Master).

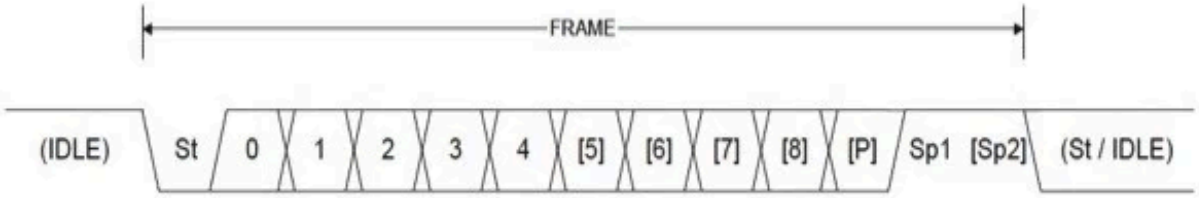
Important Concepts

- Serial communication, specially USART and RS232 requires that you specify the following four parameters
  - The baud rate of the transmission
    - Baud rate is a measure of how fast data are moving between instruments
  - The number of data bits encoding a character (frame)
  - The sense of the optional parity bit
  - The number of stop bits
  - A typical frame for USART/RS232 is usually 10 bits long: 1 start bit, 8 data bits, and a stop bit.



Important Concepts

- Order of Bits
  - Start bit (Always low)
  - Data bits (LSB to MSB) (5-9 bits)
  - Parity bit (optional) (Can be odd or even)
  - Stop bit (1 or 2) (Always high)



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

USART Configuration

- The USART registers
  - Few registers control and provide status of the USART.
    - Control and Status Register A (UCSRnA)
    - Control and Status Register B (UCSRnB)
    - Control and Status Register C (UCSRnC)
    - Baud Rate Register (UBRRn)
    - USART Data Register (UDR)
  - The data size used by the USART is set by the UCSZ2:0, bits in UCSRC Register
  - Number of stop bits to be inserted by the transmitter is set by the USBS bit in the UCSRC Register.
  - The parity setting bits are set by the UPM1:0 bits in the UCSRC Register. (see appendix)

USART Configuration - Library

- The good news is that you can make use of the uart library provided on BB to take care of configuring, reading and writing uart.
- This library allows to use the UART port in normal mode and standard configuration.
- If you want to customise your UART connection then see appendix for advance configuration. This include double speed, interrupts, etc

USART Configuration - Library

- For example, the library contains function to initilaise, read, write and check if data is available. This will be explained through the examples:
  - uart_init(BAUD)
  - uart_putchar(uint8_t)
  - uint8_t c = uart_getchar(void);
  - uint8_t avail = uart_available()

- Examples: uart_hello.c,uart1.c,uart2.c

Summary

- Serial comms are widely used in microcontrollers
  - SPI, I2C, UART and USB are the most used protocol/interfaces.
  - Choice depends on application

- Example code in Blackboard

APPENDIX

- USART Advanced configuration
  - The following show the registers associated with some advance configuration option for the UART .

USART Configuration

- Control and Status Register A (UCSRnA)
  - Useful for doubling transfer rate by setting the U2X bit in UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSRA|= (1<<U2X);

NOTE: Not used in this subject

USART Configuration

- Control and Status Register B (UCSRnB)
  - Useful for enabling Tx and Rx
    - Bit 4: RXEN – Receiver Enable: Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled.
    - Bit 3: TXEN – Transmitter Enable: Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
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	

```
UCSR1B|= (1<<TXEN1)|(1<<RXEN1);
```

USART Configuration

- Control and Status Register B (UCSRnB)
  - Also useful for enabling interrupts every time data is Tx or Rx
    - **Bit 7: RXCIE – RX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the RXC Flag. A Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set. The result is that whenever any data is received, an interrupt will be fired by the CPU.
    - **Bit 6: TXCIE – TX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the TXC Flag. Similar behaviour as RXCIE

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
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	

```
UCSR1B |= (1<<TXCIE1) ; //Tx    or
UCSR1B |= (1<<RXCIE1); //Rx
```

USART Configuration

- Control and Status Register C (UCSRnC)
  - **Bit 6: UMSEL – USART Mode Select:** This bit selects between Asynchronous and Synchronous mode of operation.
  - **Bit 5:4: UPM1:0 – Parity** Mode: This bit helps you enable/disable/choose the type of parity.
  - **Bit 3: USBS – Stop Bit Select:** This bit helps you choose the number of stop bits for your frame.
  - **Bit 2:1: UCSZ1:0 – Character Size:** These two bits in combination with the UCSZ2 bit in UCSRB register helps choosing the number of data bits in your frame

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

USART Configuration

- Control and Status Register C (UCSRnC)
  - **Bit 6: UMSEL – USART Mode Select:** This bit selects between Asynchronous and Synchronous mode of operation

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

default asynchronous.

USART Configuration

- Control and Status Register C (UCSRnC)
  - **Bit 5:4: UPM1:0 – Parity** Mode: This bit helps you enable/disable/choose the type of parity

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

default disabled

USART Configuration

- Control and Status Register C (UCSRnC)
  - **Bit 3: USBS – Stop Bit Select:** This bit helps you choose the number of stop bits for your frame.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

default 1 bit.

USART Configuration

- Control and Status Register C (UCSRnC)
  - **Bit 2:1: UCSZ1:0 – Character Size:** These two bits in combination with the UCSZ2 bit in UCSRB register helps choosing the number of data bits in your frame

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

```
UCSR1C |= (1<<UCSZ10) | (1<<UCSZ11);
```

USART Configuration

- USART Baud Rate Register (UBRRn)
  - Set the baud rate

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	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	

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

UBRR1 = BAUD;

USART Configuration - Summary

- Set the following registers:
  - Control and Status Register B (UCSRnB)
  - Control and Status Register C (UCSRnC)
  - USART Baud Rate Register (UBRRn)

Example:
1.- Asynchronous normal mode
2.- 8 data bits
3.- 1 stop bit
4.- No parity bits

```
UBRR1 = BAUD;                                  // set baud rate
UCSR1B |= (1<<TXEN1) | (1<<RXEN1);             // enable receiver and transmitter
UCSR1C |= (1<<UCSZ10) | (1<<UCSZ11);           // 8bit data format
```