USART and Asynchronous Communication

The USART is used for synchronous and asynchronous serial communication.

USART = Universal Synchronous/Asynchronous Receiver Transmitter

Our focus will be on asynchronous serial communication.

Asynchronous communication does not use a clock to validate data.

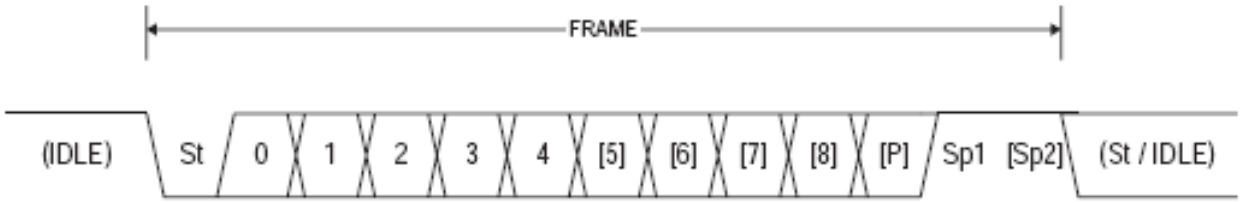
Serial data is transferred one bit at a time.

Asynchronous serial interfaces are cheap, easy to use, and until recently, very common. USB is well on its way to replace the serial comm ports on PCs.

The USART communicates in a full-duplex mode (simultaneous xmit, rcv)

USART and Asynchronous Communication

Serial frame format:



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.

Every frame will have at least

- -one start bit
- -some data bits (5,6,7,8 or 9)
- -one stop bit

Parity is optional

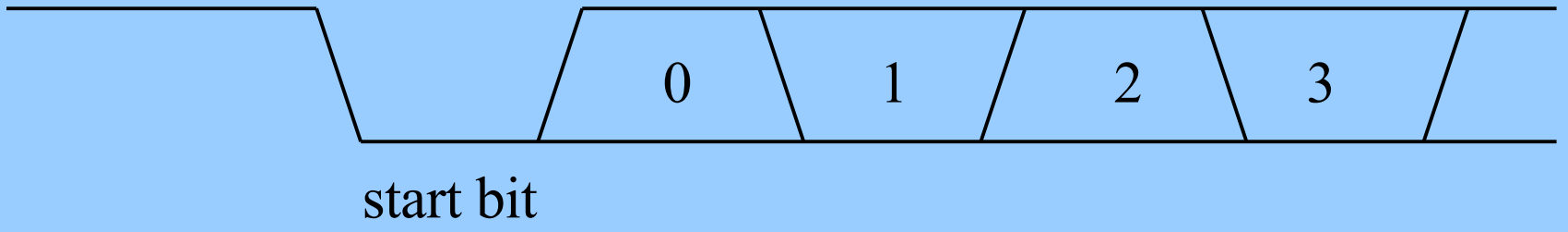
USART and Asynchronous Communication

How can two asynchronous devices communicate with each other?

There is no known phase relationship between the two AVR boards

How can a receiving board “know” where the center of a bit frame is?

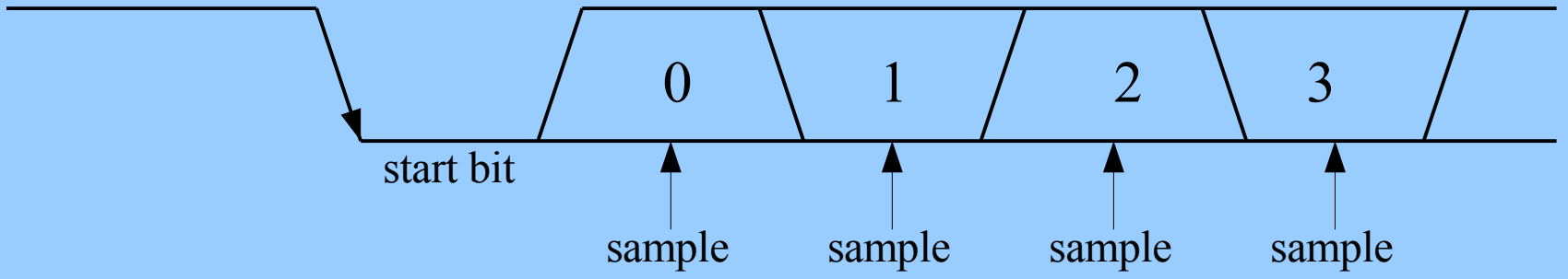
How can it know where the start bit is?



What do we need to know?

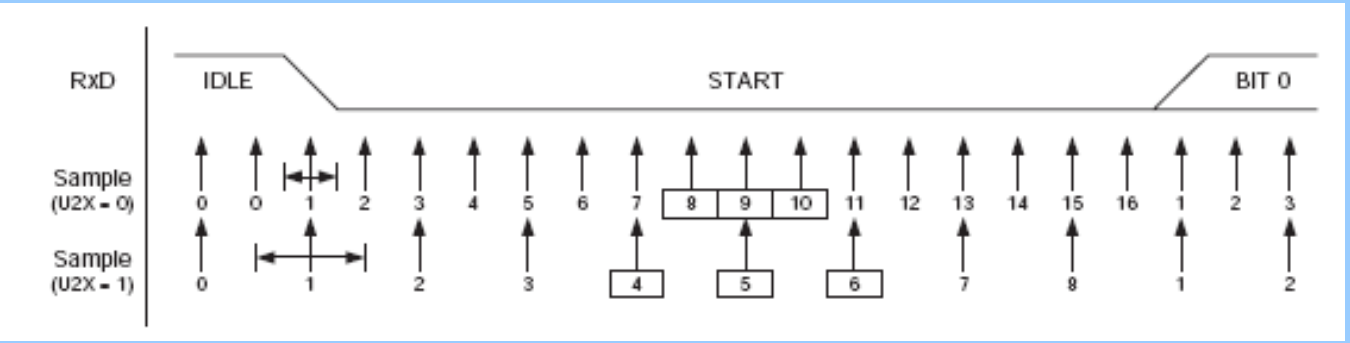
USART and Asynchronous Communication

- Need to know how fast the bits are coming (baud rate)
- Need to know where the start bit begins (its falling edge)
- Then we know when to sample



USART and Asynchronous Communication

- The USART uses a 16x internal clock to sample the start bit.



The incoming data is continuously sampled until a falling edge is detected. Once detected, the receiver waits 6 clocks to begin sampling.

One clock before the expected center of the start bit, 3 samples are taken.

If 2 or 3 are detected as high, the search for another falling edge is started.

If at least one sample is low, the start bit is validated. If so, begin sampling 16 clocks from center of start bit.

Revalidate again with each data byte. Synchronization happens on each byte.

USART and Asynchronous Communication

- The USART internal clock is set by the UBRR register.

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	

The internal clock is derived from the internal CPU clock.

Both transmitter and receiver use the same clock to operate from. The transmitter is synchronous to the clock. In coming data to the receiver is not.

The baud rates are not exact power of 2 multiples of the CPU clock, thus baud rate inaccuracies occur at some settings. This is why you see some "funny" crystal oscillator frequencies.

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1 Mbps		2 Mbps	

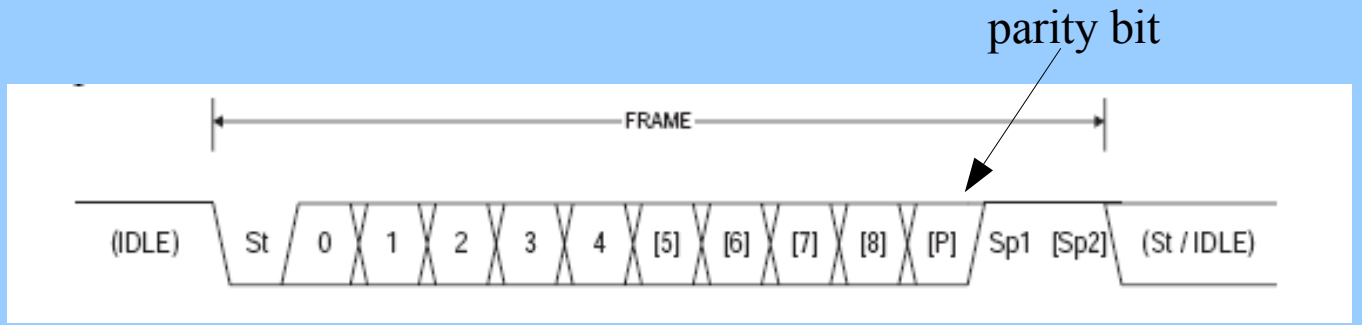
1. UBRR = 0, Error = 0.0%

USART and Asynchronous Communication

- Error detection
  - -Parity
    - -created by an XOR of the data bits
    - -parity can be even or odd
    - -P_{even} = dn XOR dn-1 XOR dn-2 ..... XOR d0
    - -P_{odd} = dn XOR dn-1 XOR dn-2 ..... XOR d0 XOR 1

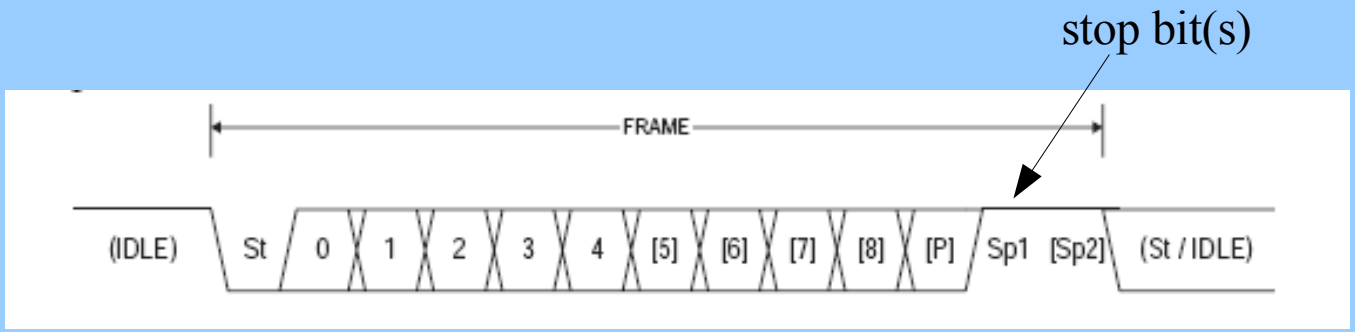
If we have a data byte, 0b0010_1101
and we want odd parity, the parity bit is set to a “one” to make a total of 9 bits which is an odd number of 1's.... i.e., odd parity

Thus, the new data byte with parity is:
0b0010_1101 plus the parity bit 1



USART and Asynchronous Communication

- Error detection
  - -Frame error
    - -a frame error occurs when the receiver is expecting the stop bit and does not find it.
    - -also known as a synchronization failure
    - -the frame (byte) must be resent



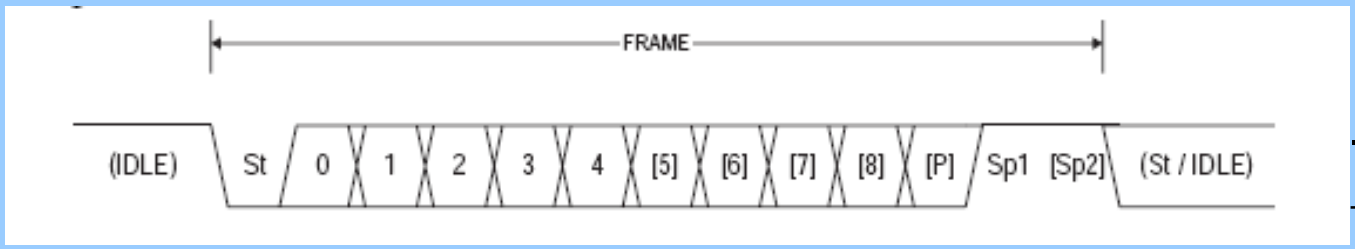
  - -Data overrun error
    - -data was not removed in the receive buffer before another byte came and overwrote it.

USART and Asynchronous Communication

- USART Electrical path

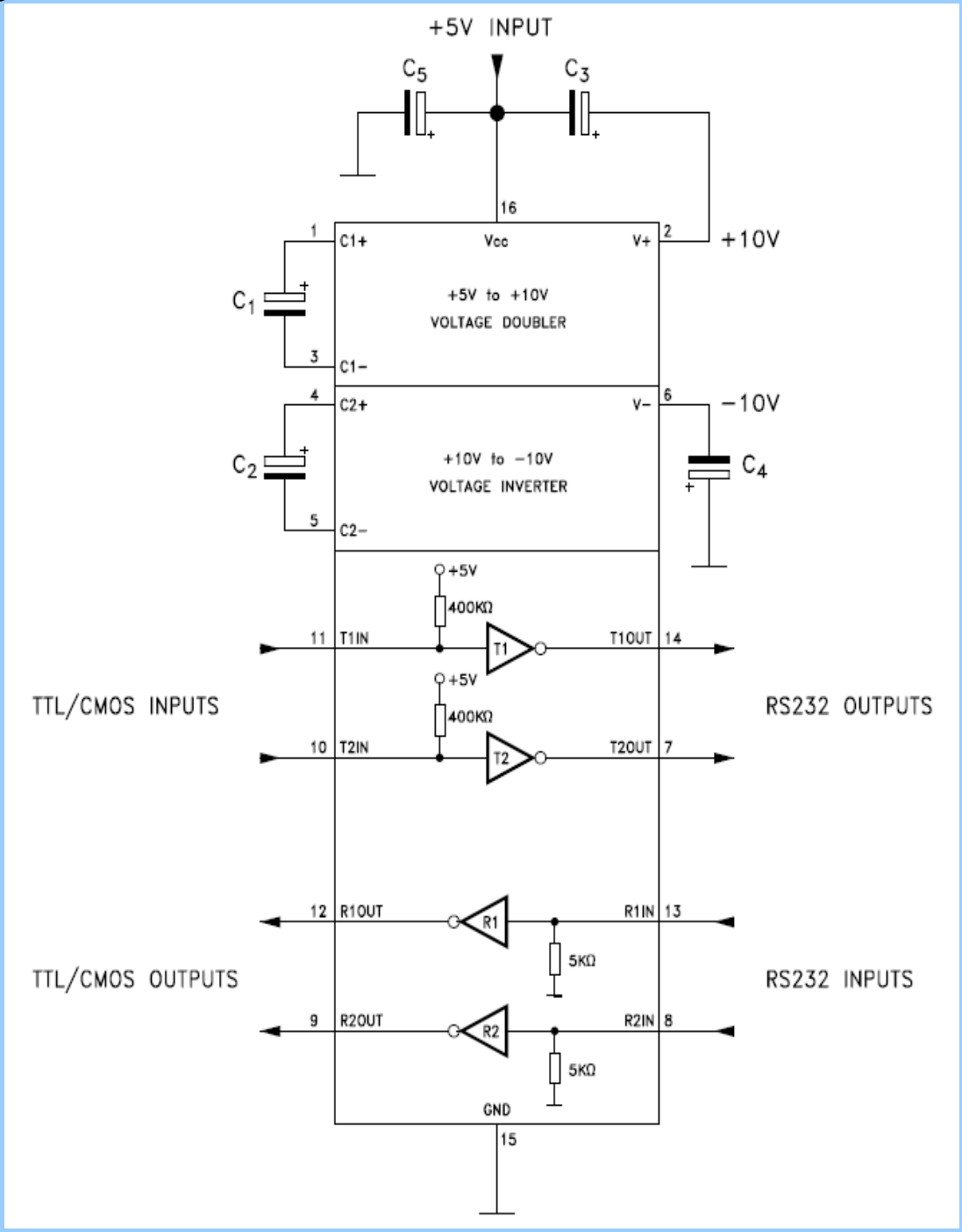
  We use the RS-232 standard for communications to a PC.
  RS-232 specifies an inverted +/- 3 Volt interface.

  <= -3 volts is considered a “1” or “mark”
  >= +3 volts is considered a “0” or “space”



USART and Asynchronous Communication

- USART RS232 transceiver
-Charge Pump inverter used to generate negative voltages.



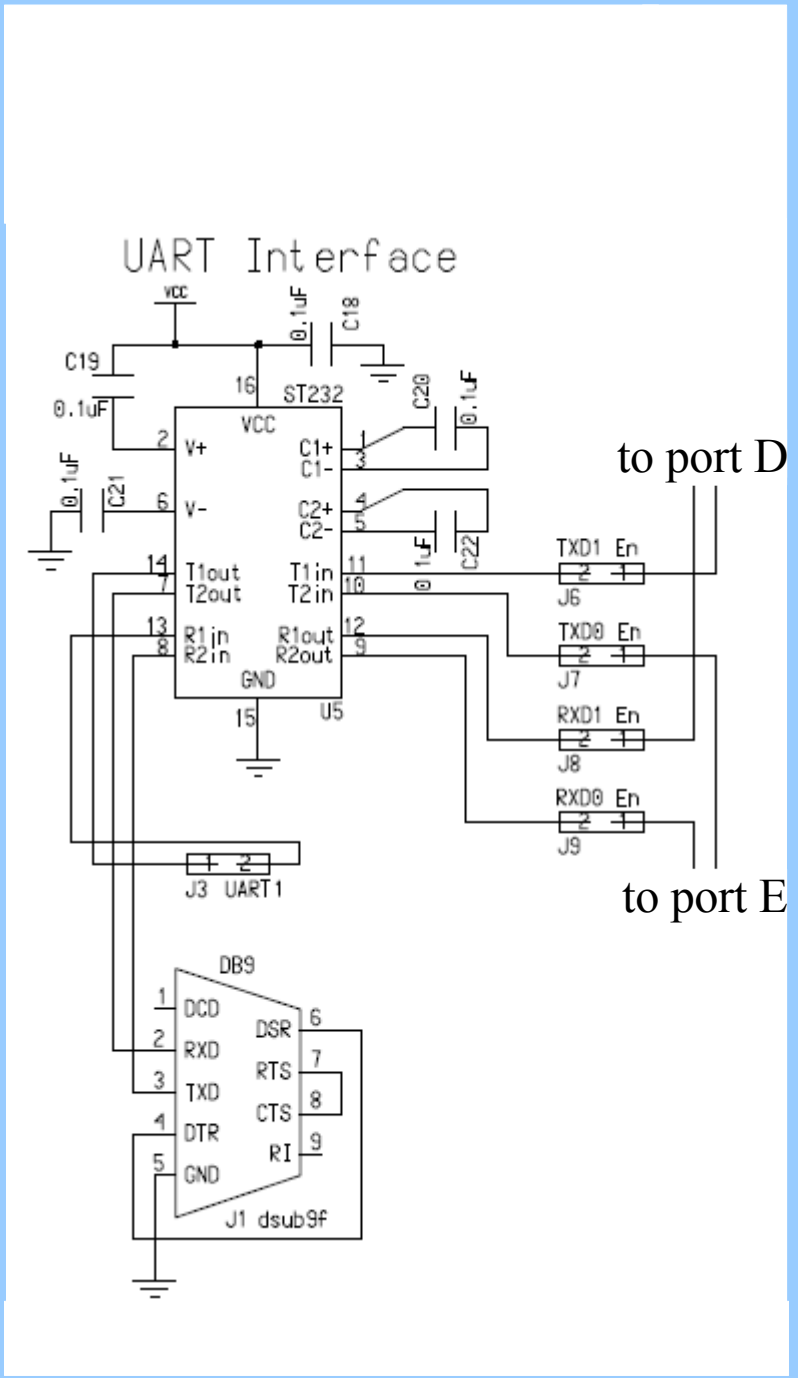
USART and Asynchronous Communication

- USART Electrical path

The mega128 board uses the MAX232 chip. It has an internal charge pump to generate the negative and positive voltages for the RS-232 interface.

The DB9 connector is standard for mating with a PC serial connector.

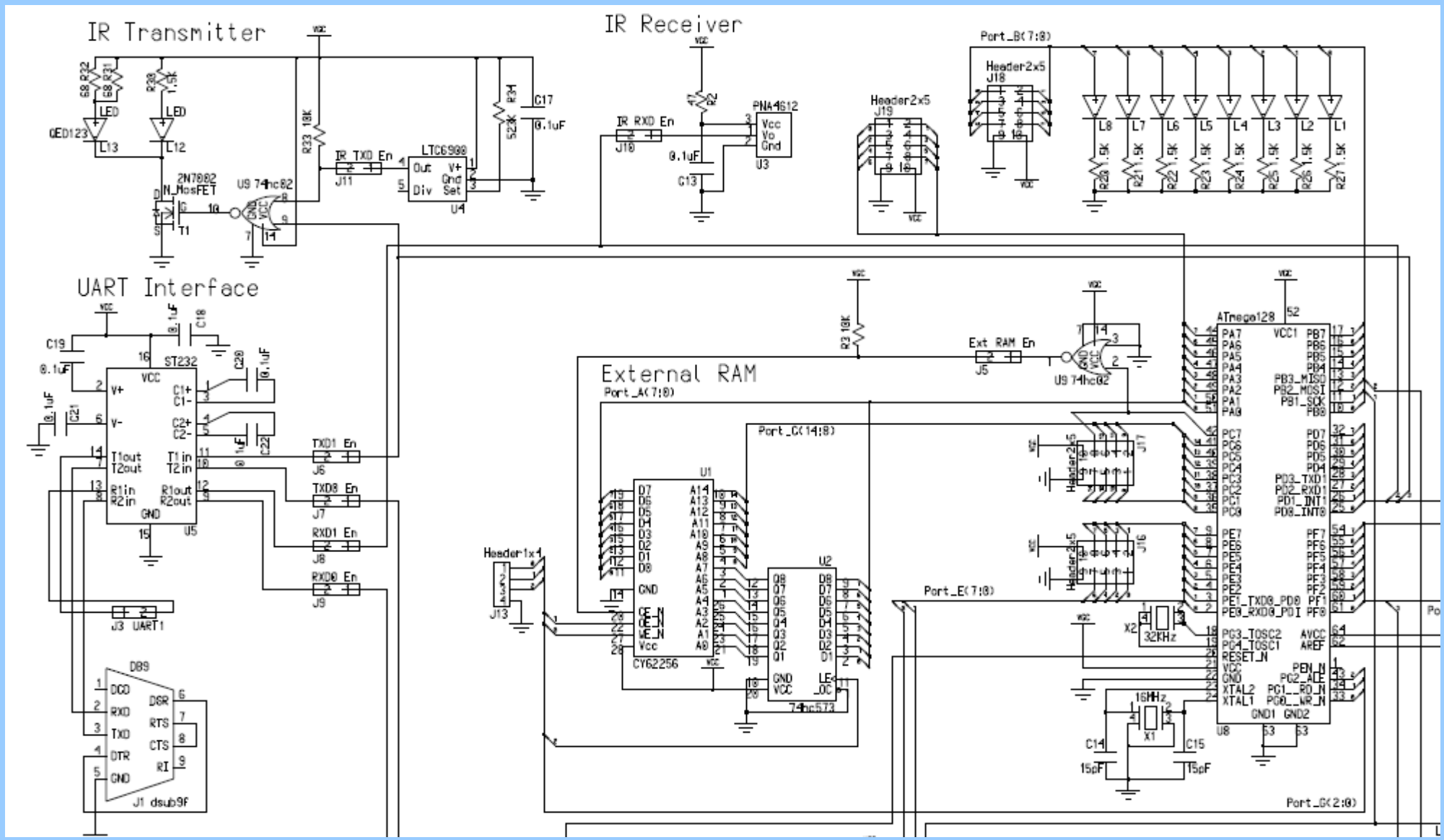
The port D interface is shared with the IR LED I/O devices. They are inverted also so that an “space” signal does not keep the IR LED on (100mA).



USART and Asynchronous Communication

USART Electrical path

-UART can be used with IR LED also.



USART and Asynchronous Communication

Some useful USART Software Routines:

```
//*************************************************************
//                            uart_init
//
//RXD0 is PORT E bit 0
//TXD0 is PORT E bit 1
//Jumpers J14 and J16 (mega128.1) OR
//Jumpers  J7 and  J9 (mega128.2)
//must be in place for the MAX232 chip to get data.
//
void uart_init(){
//rx and tx enable, 8 bit characters
  UCSR0B |= (1<<RXEN0) | (1<<TXEN0);
//async operation, no parity,  one stop bit, 8-bit characters
  UCSR0C |= (1<<UCSZ01) | (1<<UCSZ00);
//set to 9600 baud
  UBRR0H=0x00;
  UBRR0L=0x67;
}
//*************************************************************
```

USART and Asynchronous Communication

Some useful USART Software Routines:

```
//*************************************************************
//                          uart_putc
//
//Takes a character and sends it to USART0
//
void uart_putc(char data){
  while (!(UCSR0A&(1<<UDRE))); // Wait for previous transmission
  UDR0 = data;  // Send data byte
  while (!(UCSR0A&(1<<UDRE))); // Wait for previous transmission
}
//*************************************************************

//*************************************************************
//                          uart_puts
//Takes a string and sends each charater to be sent to USART0
//void uart_puts(unsigned char *str) {
void uart_puts(char *str) {
  int i = 0;
  //Loop through string, send each char
  while(str[i] != '\0'){uart_putc(str[i]); i++;}
}
//*************************************************************
```

USART and Asynchronous Communication

Some useful USART Software Routines:

```
//*************************************************************
//                          uart_getc
//Modified to not block indefinately in the case of a lost byte
//
char uart_getc(void) {
  uint16_t timer = 0;
  while (!(UCSR0A & (1<<RXC0))) {
    timer++;
    if(timer >= 16000) return 0;
  }  // Wait for byte to arrive
  return UDR0;
}
//*************************************************************
```