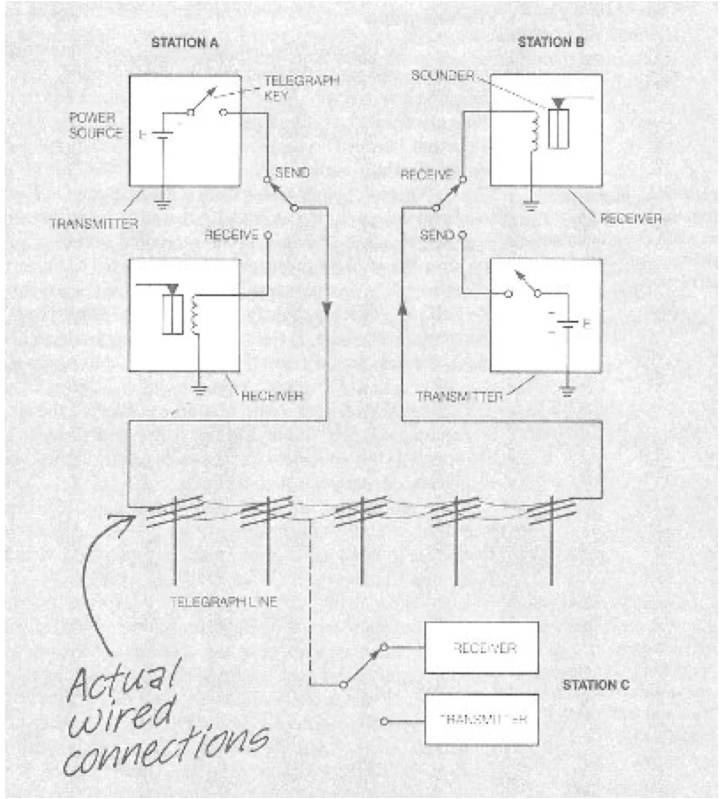
Serial I/O

Lecture Outline

- Serial and Parallel Communications Fundamentals
- Serial Protocols and the Atmel 2560

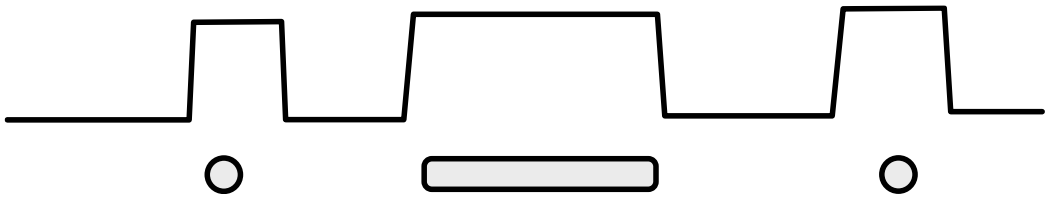
Communicating Information

- Goal: Transmit information from one point to another (possibly distant) point
- Early example: telegraph
  - Morse code sent one letter at a time over wires
  - Operators have to learn to recognize the code of dashes (long beeps) and dots (short beeps)
  - One dash or dot at a time, building up to a letter, is *serial* communication



Voltages on a Telegraph Wire

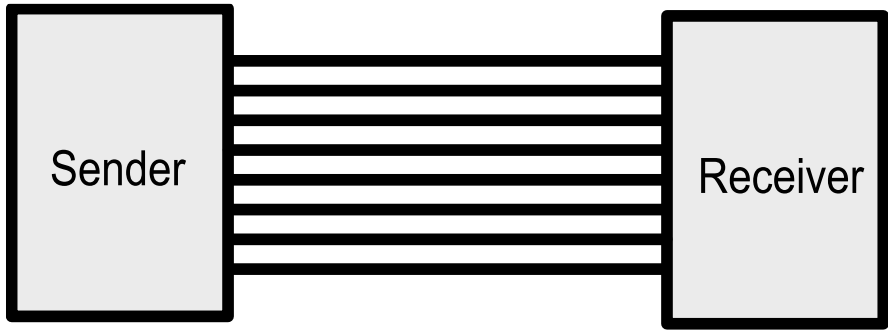
- When the switch is pressed, the voltage goes high
- Dots and dashes were made by the relative length of the signal
- Telling the difference between adjacent letters is a learned skill
- The time between letters is dependent on the operator!



The letter R

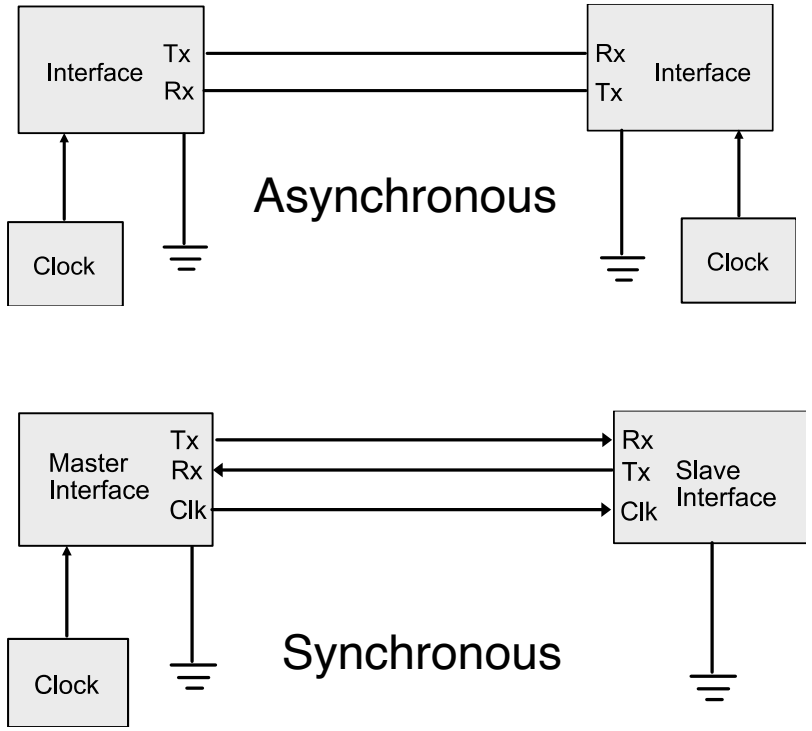
Parallel Communication

- All bits transferred in parallel
- The busses we have studied are all parallel
- Faster transfer than sending one bit at a time, but
  - Much more expensive
  - Susceptible to electrical effects, so distance limited



Serial Communication

- One bit transferred at a time
- The bits transferred per second is known as the *bit rate (bps)*
- The time to send one bit is t_{bit} , so bps is equal to $1/t_{bit}$
- The *baud rate* is the number of symbols transferred per second
  - If one bit represents one symbol, the the baud rate is the same as the bit rate
- Two general categories:
  - Asynchronous: no common clock signal
  - Synchronous: common clock signal between sender and receiver

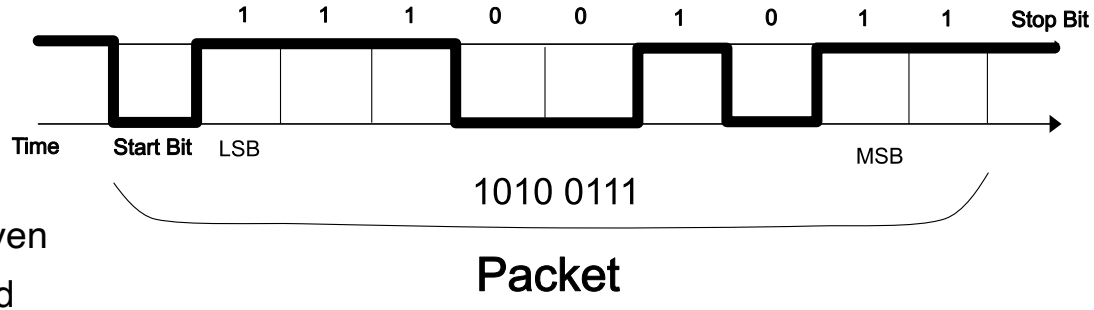


Types of Serial Channels

- Simplex
  - Data is transferred from sender to receiver in a unidirectional link
- Half-duplex
  - Communication is bidirectional over a single link
  - Only one end can transmit at a time
    - Think walkie talkies
- Full duplex
  - Two links provide the ability for both ends to transmit / receive at the same time

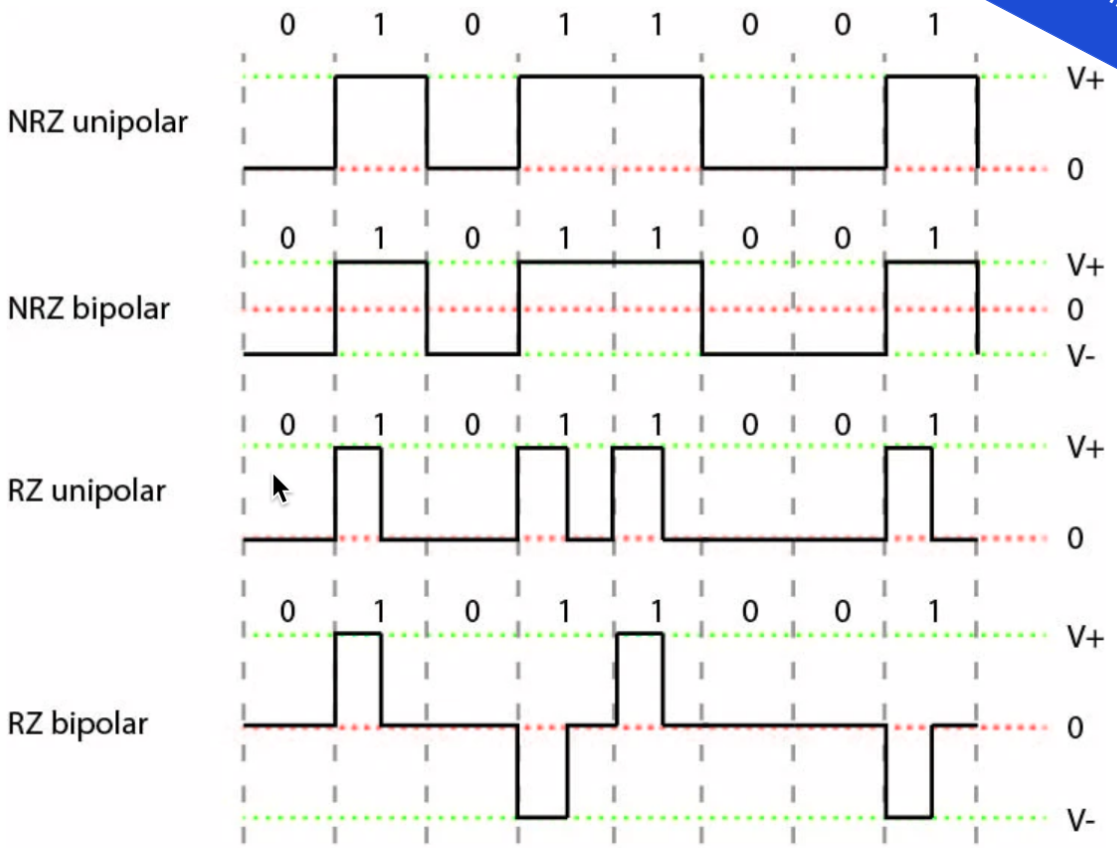
The Asynchronous Packet Format

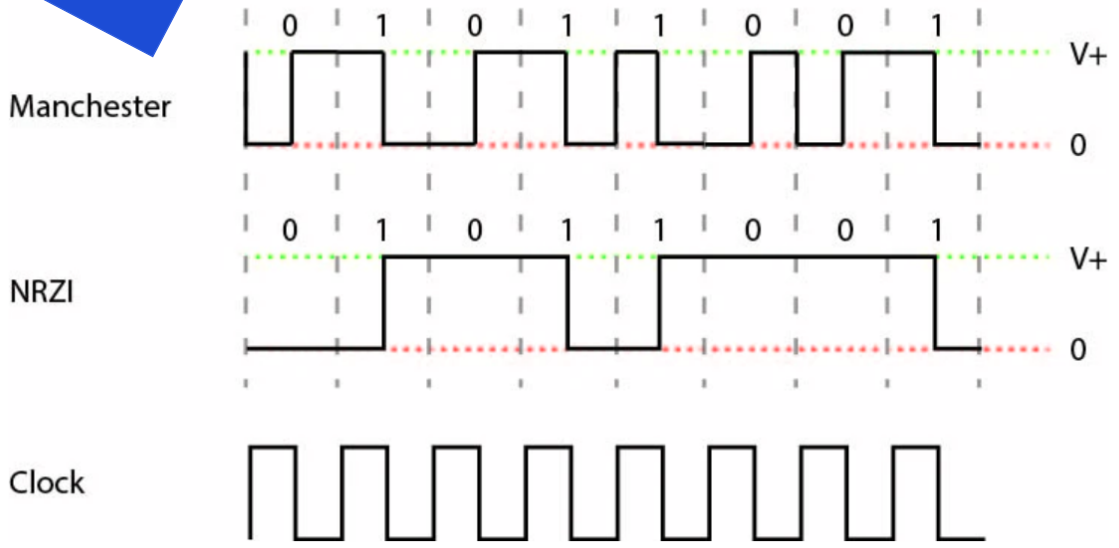
- A *packet* is one unit of information, including both data and control information
- Start bit (low) indicates the start of the packet, stop bit is high
- The time required for 1 bit to be transmitted t_{bit} is determined by the data transmission rate
- LSB is sent first, MSB last (in general)
- Error checking using *parity* :
  - Even parity: the number of 1s *including the parity bit* should be even
  - Odd parity: the number of 1s *including the parity bit* should be odd
  - Tx and Rx sides must agree on which type of parity to use
  - The parity bit is set to force the sequence to have even or odd parity
  - If the parity does not match, and error has occurred in transmission



Bit Encoding

NRZ - latched at the beginning of the cycle, and sampled in the middle of the cycle



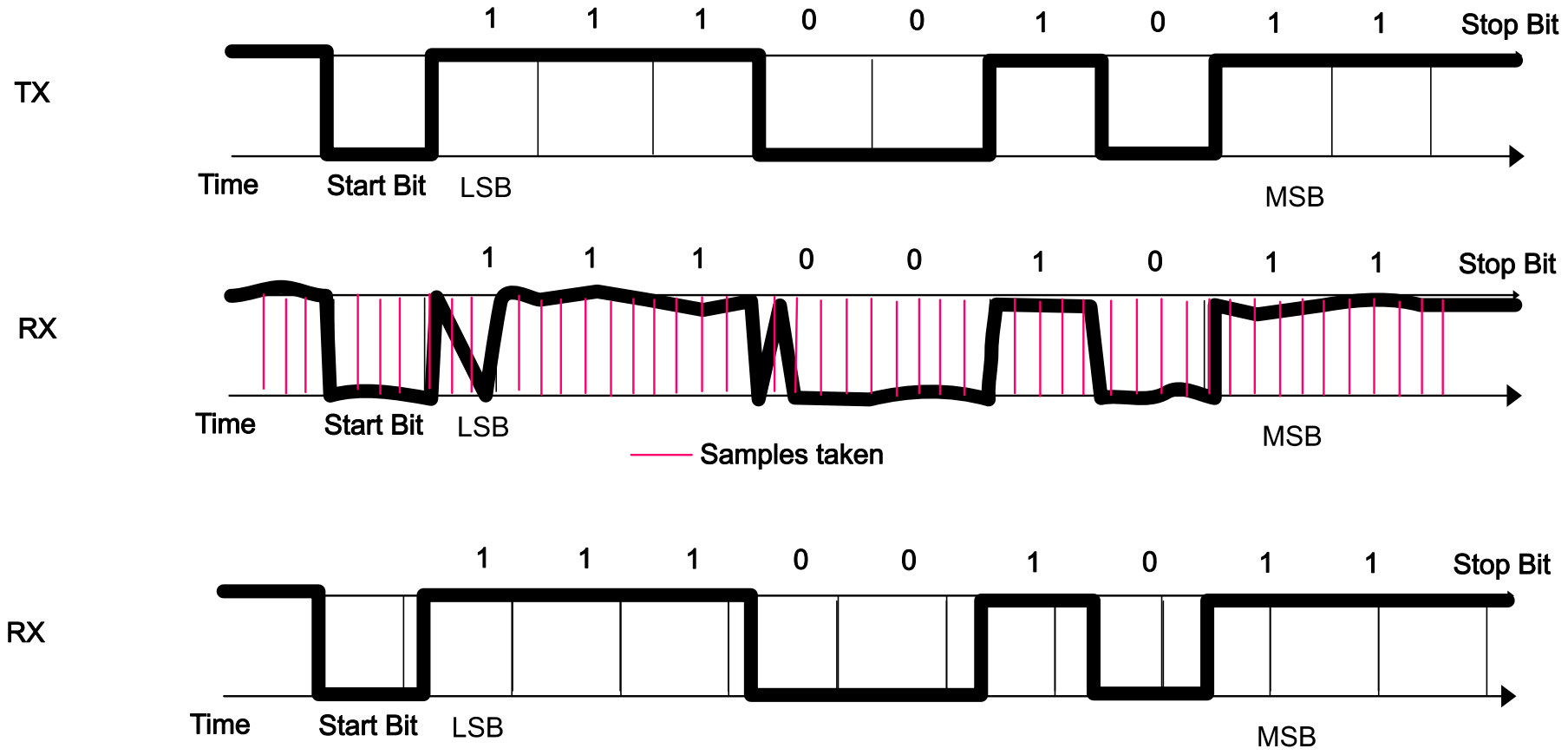


https://www.electronicdesign.com/technologies/communications/article/21802271/whats-the-difference-between-nrz-nrzi-and-manchester-encoding

Decoding an Async Packet

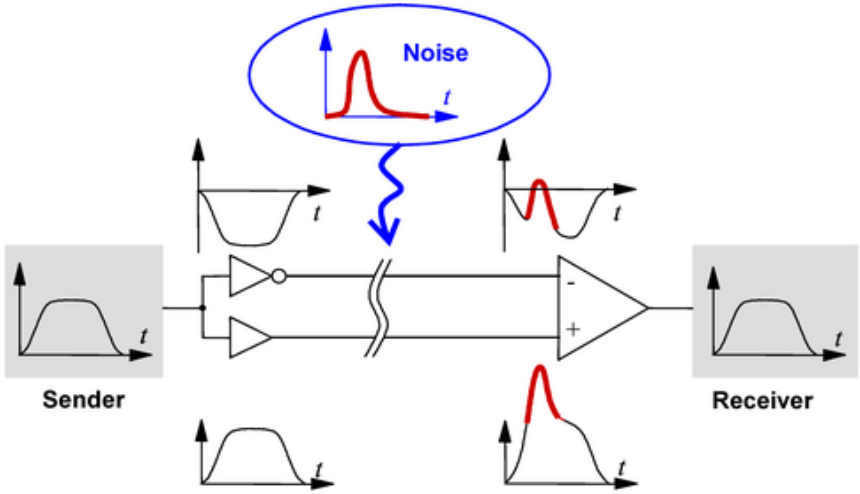
- *Start bit* is detected when a 1 to 0 transition is detected
- The line is sampled again ½ t_{bit} later to confirm a start
  - If the line is high, it is assumed that there was a *framing error*
  - If the line is still high, the start is assumed to be valid
- Data is transmitted at the transmission rate determined by the transmitter clock
- The receiver starts its clock after the start bit at the same rate
  - Remember – the clocks are NOT synced in any way
- The receiver samples the signal at some multiple of the clock rate in order to allow for some variance
- The start bit for each packet resync both clocks. This is important as both clocks can *drift* over time

Effects of Noise and Clock Error/Drift



Wired Connections

- Wired connections can be *single ended* or *differential*
- Single ended: one link wire and a ground reference
  - Susceptible to noise
- Double ended:
  - 3 wires: +,- and ground
  - Signal is sent along +/- wires 180 degrees out of phase
  - Differential amplifier takes the difference of the signals
  - Induced noise has the same phase in both lines and is reduced
  - The measure of the capability to reduce this noise is called Common Mode Rejection Ratio (CMRR)

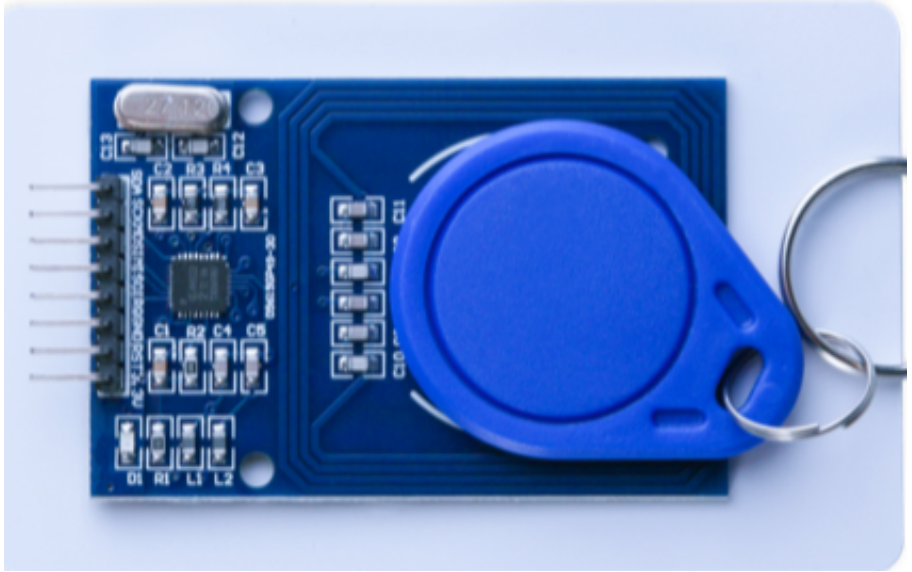


Transmission Protocols

- A protocol is “a set of conventions governing the treatment and especially the formatting of data in an electronic communications system” (Merriam Webster)
- Transmission protocols can define, among other things,
  - Clock rates
  - Packet/frame formats
  - Specifications for voltages, wiring, etc.

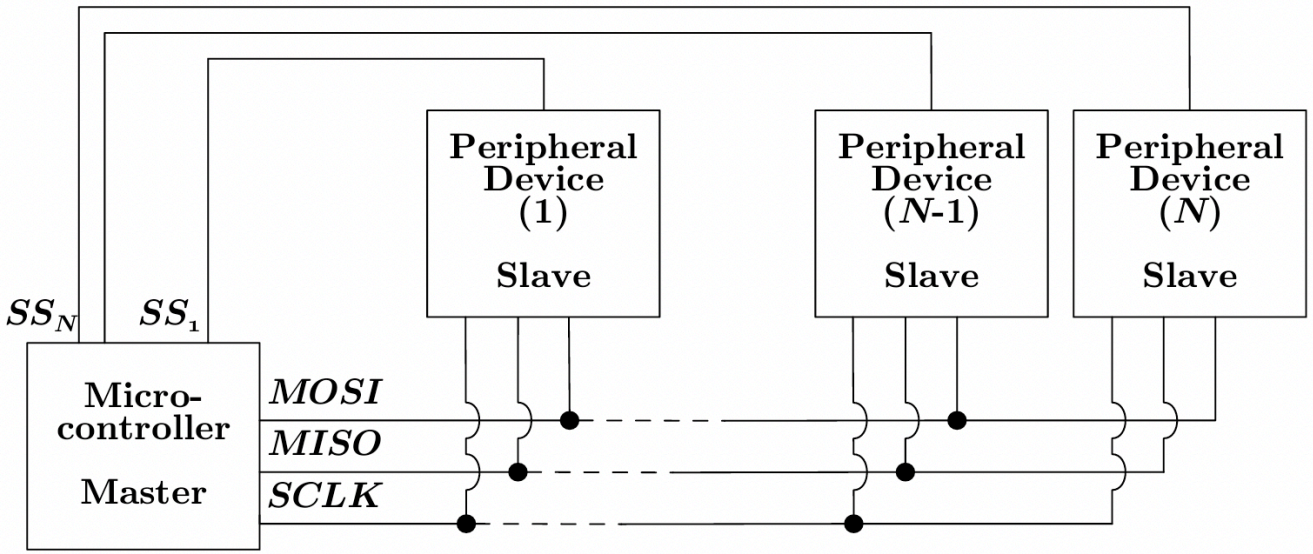
Serial Protocols: SPI

- **S**erial **p**eripheral **i**nterface
- Developed by Motorola in the 1980's
- *Synchronous* transmission
- Full Duplex
  - Master and slave can transmit at the same time
  - Simultaneous
  - Bidirectional
- Master/Slave Architecture



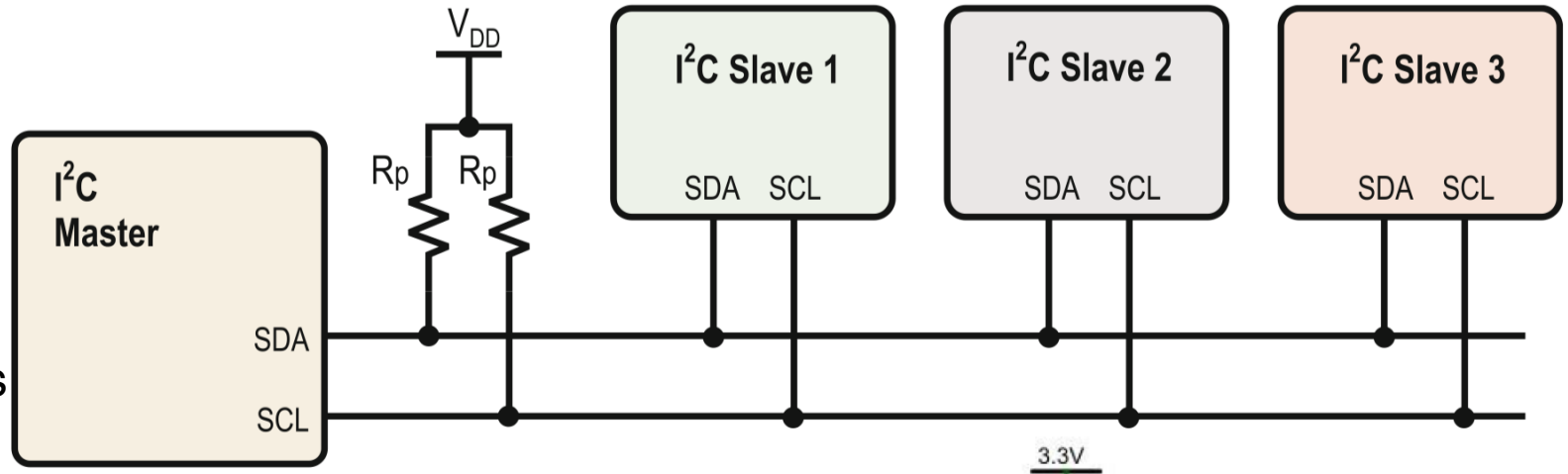
Serial Protocols: SPI (2)

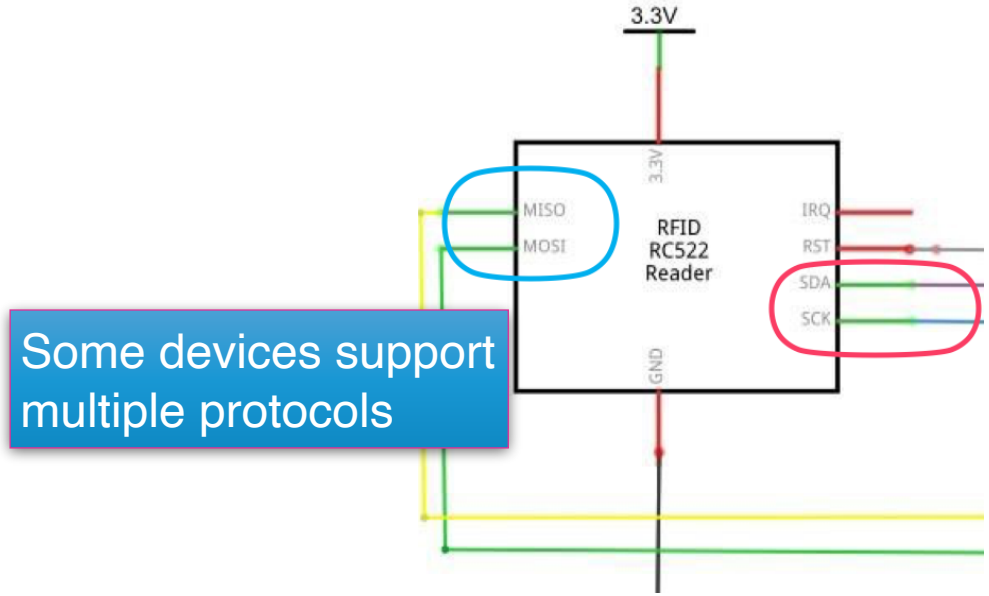
- 4 Wires
  - SCLK – Serial Clock (generated by master)
  - MOSI – Master OUT Slave IN
  - MISO – Master IN Slave OUT
  - SS – Slave Select
    - Note that there is no address line.
    - An SS line for each peripheral is required
- Very Common
  - High Speed ADCs
  - Displays



Serial Protocols: I²C

- Synchronous transmission
- Inter-integrated Circuit serial protocol
- Allows more than two devices to communicate
- Shared bus protocol made of masters and slaves
- Synchronous (master's clock is shared)
  - Two Wires
  - SCL – Serial Clock
  - SDA – Serial Data
- Very Common
  - ADCs
  - OLED Displays
  - Sensors (Brightness, Cameras, GPS)

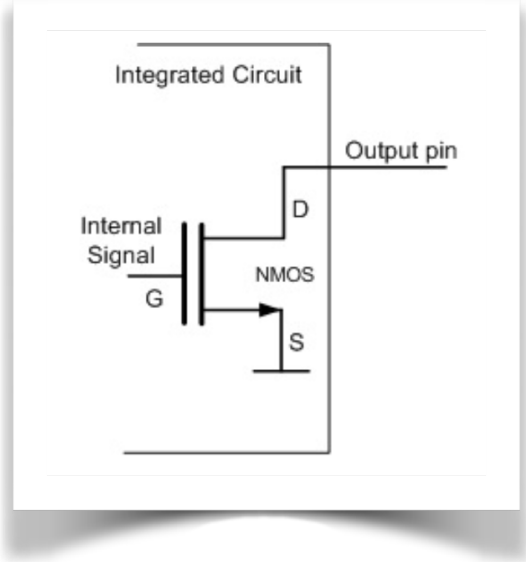


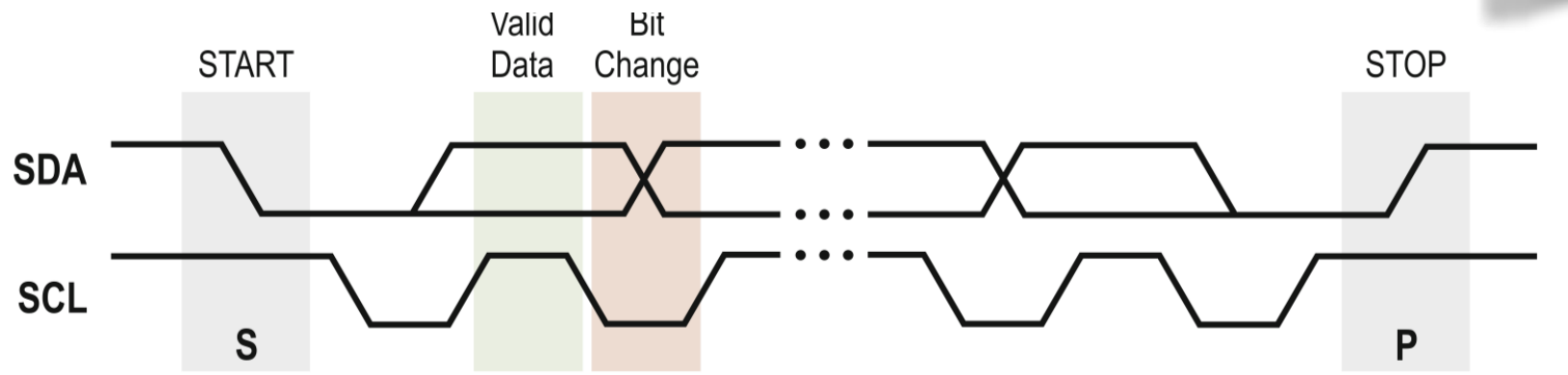


Some devices support multiple protocols

SDA and SCL Lines

- SDA and SCL lines held HIGH when not active
- Outputs of all devices open-drain (collector) so either at ground or open (Hi Z)
- START occurs when SDA goes LOW when SCL is high
- STOP occurs when SDA goes high when SCL is high



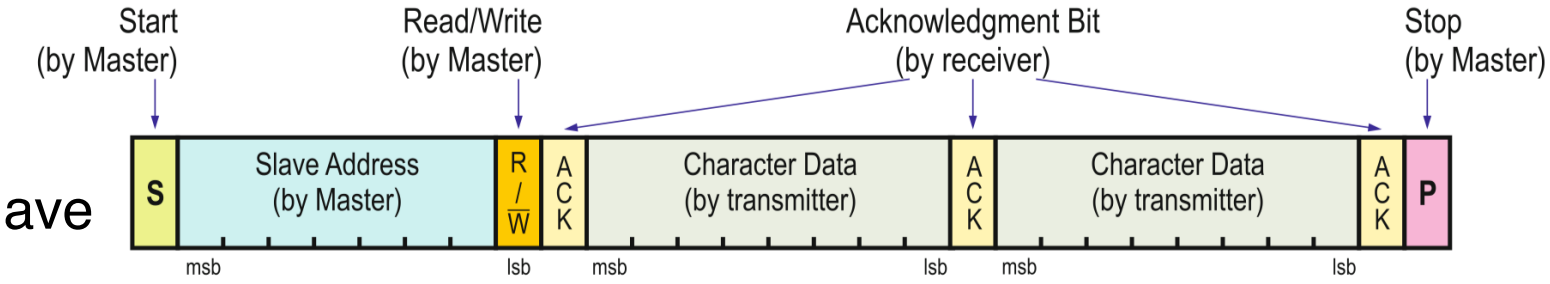


I2C Data Packets

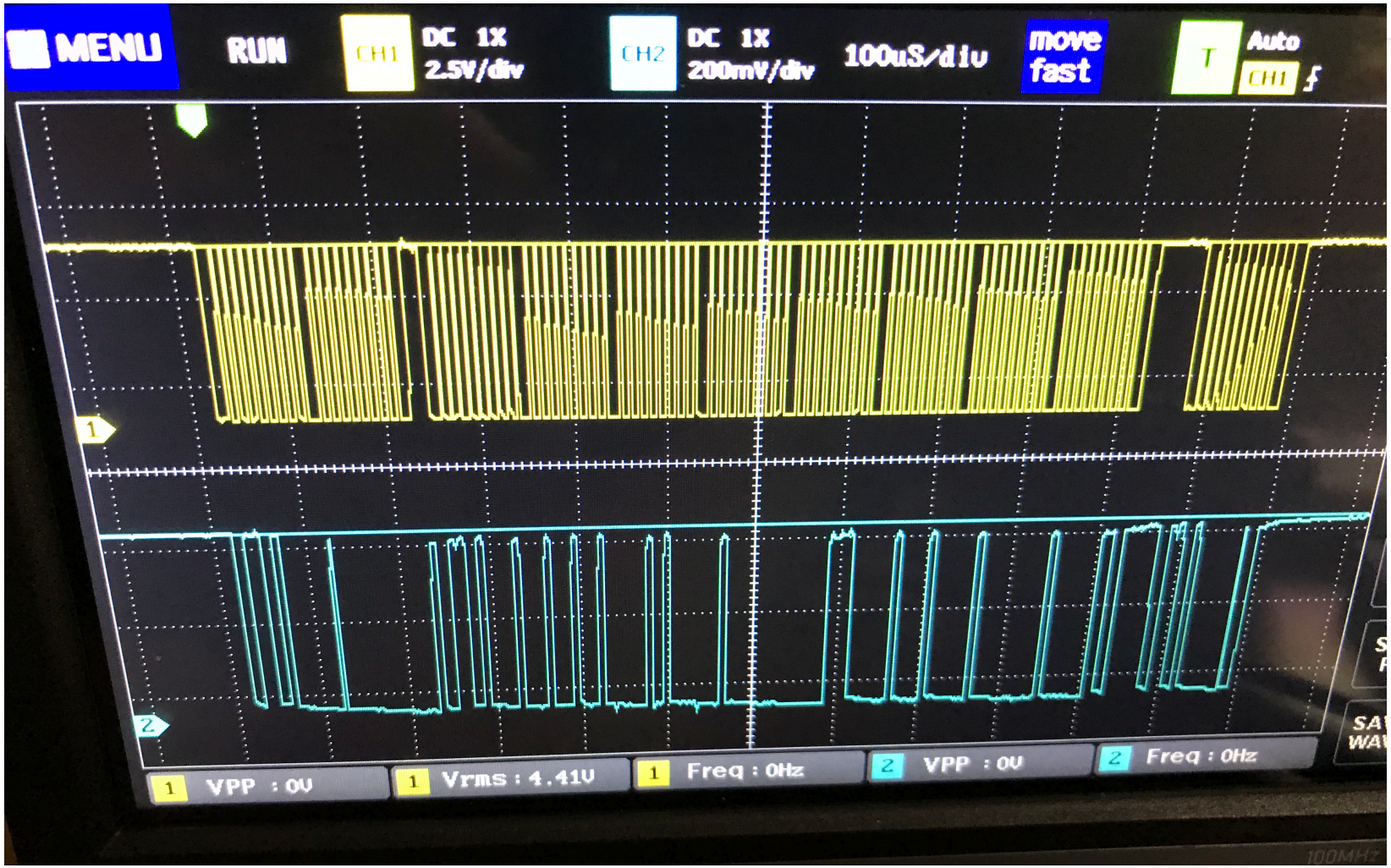
- Packet consists of 9 bits: 8 bit character from master plus the ACK bit from the slave
- Data sent in 8 bit characters
- Data transfer is most significant bit (MSB) first
- Bits may only change when clock is LOW (a bit is only considered valid when the clock is HIGH)

I2C Data Packets

- Message format:
  - START
  - Slave Address (7 bits, but max is 119 due to some values restricted)
  - Read/Write bit (1 = read)
  - ACK or NACK (ACK is when the slave pulls the SDA line low
  - Data transfer (again, 8 bits plus ACK/ NACK)
  - STOP



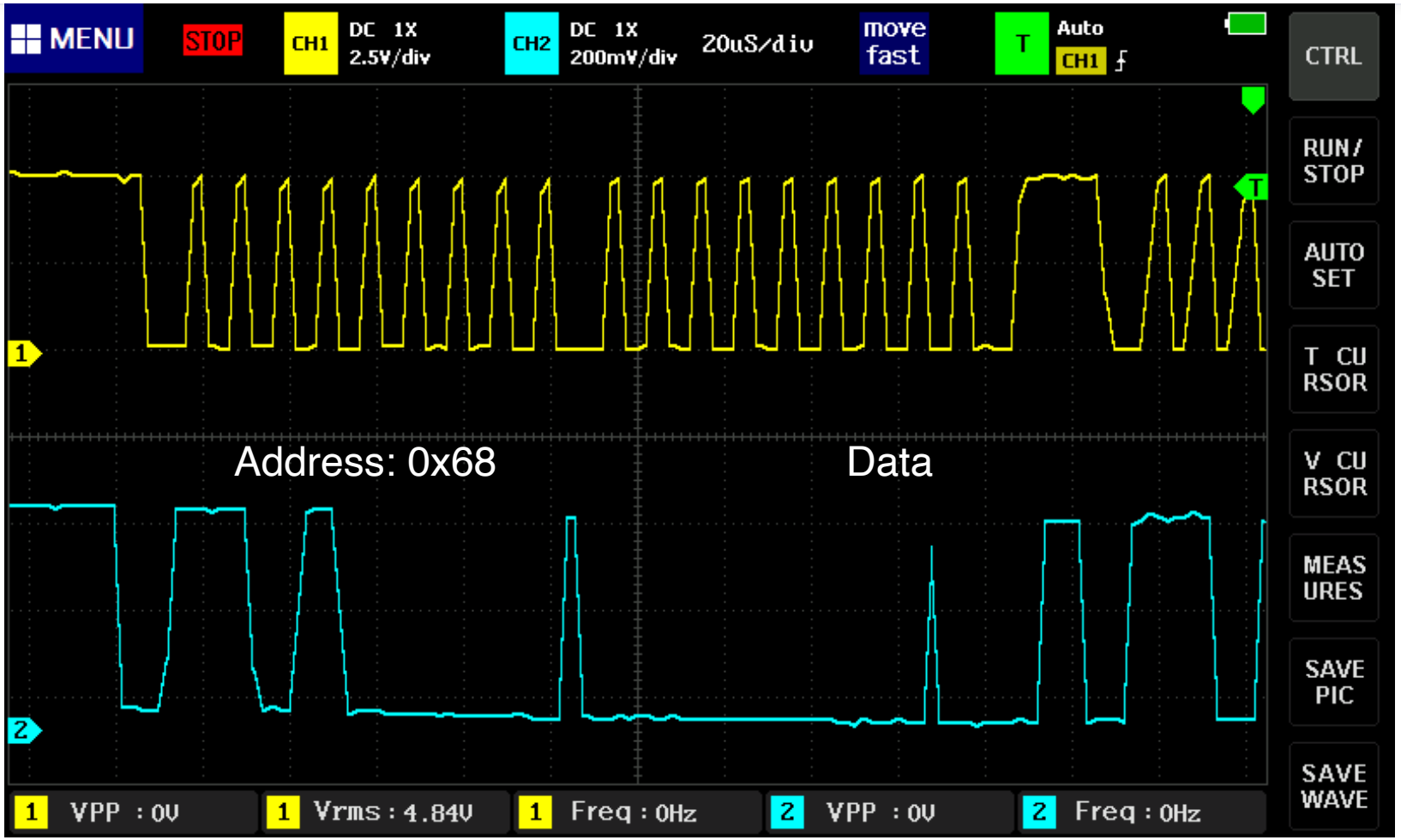
Full Transmission from Real-Time Clock



Real-time Clock Example

Raw data: 2020-10-11 19:27:46

0x68 = 1101000



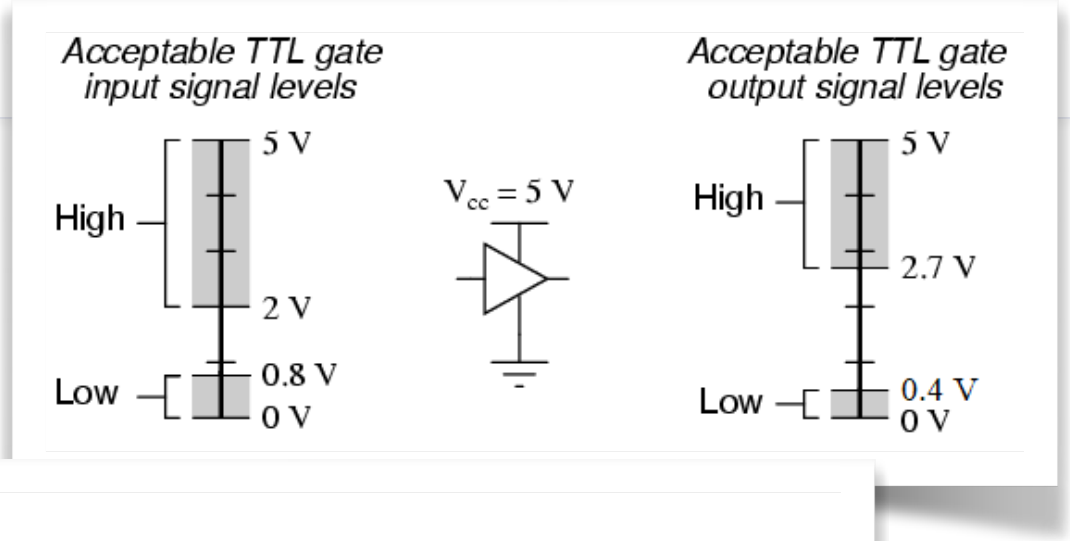
Serial Protocols: RS232

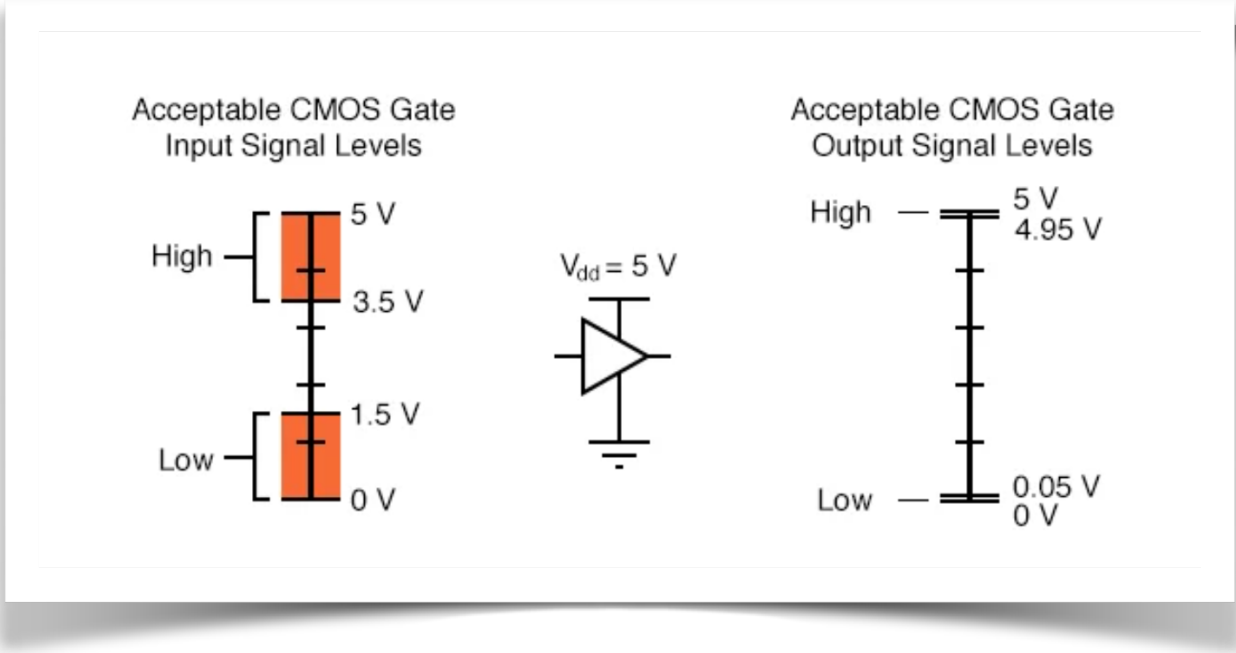
- Developed in the 1960s for connecting terminals to computers
- Asynchronous transmission: no clock signal
- Largely supplanted by USB
- Currently used for the most part in industrial equipment
- Ancestor of USB, Firewire, TCP/IP
- Hardware and transmission standards defined

By Hubert Berberich (HubiB) - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=30189075

Connecting the Arduino to RS 232

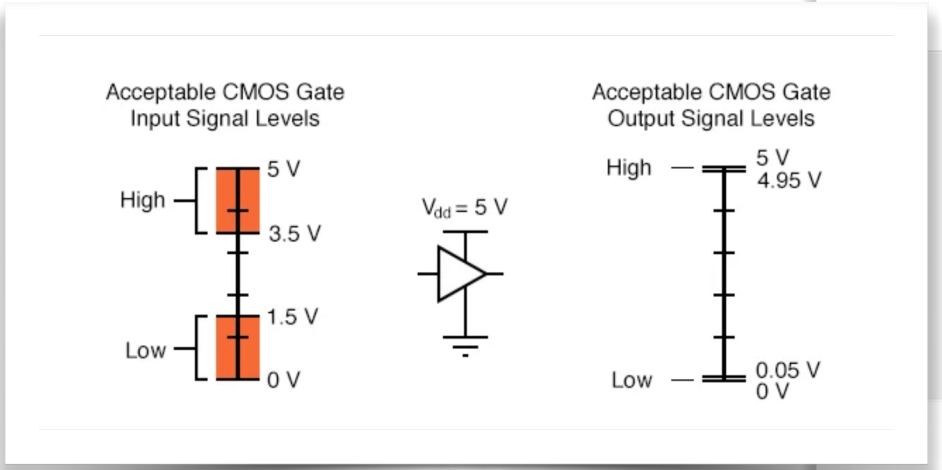
- The Arduino logic levels are transistor-transistor levels (TTL)
- RS 232 Voltage Levels
  - 1: -3 to -25 V
  - 0: 3 to 25 V
- Solution: Use converter such as MAX232

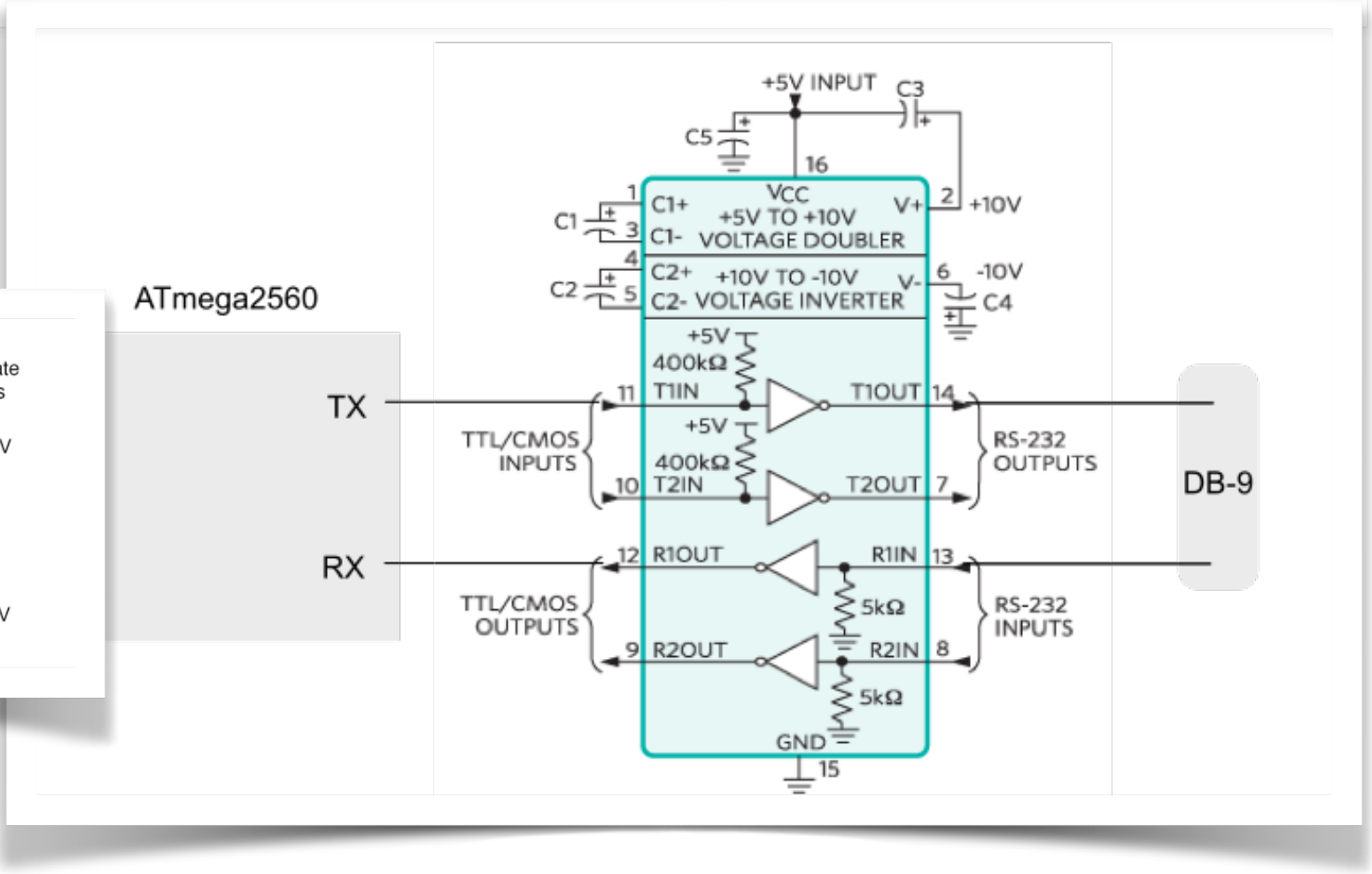




Connecting the AVR to RS 232 (2)

- Solution: Use converter such as MAX232





Universal Serial Bus (USB)

- Largely supplanted the RS 232 interface
- More than just a standard for cables and voltages
  - Protocol specifies physical, link, session, and application layers
- USB 2.0: Half-duplex, 480Mbps
- USB 3.0: Full-duplex, up to 5 Gbps

USB Cable Specification

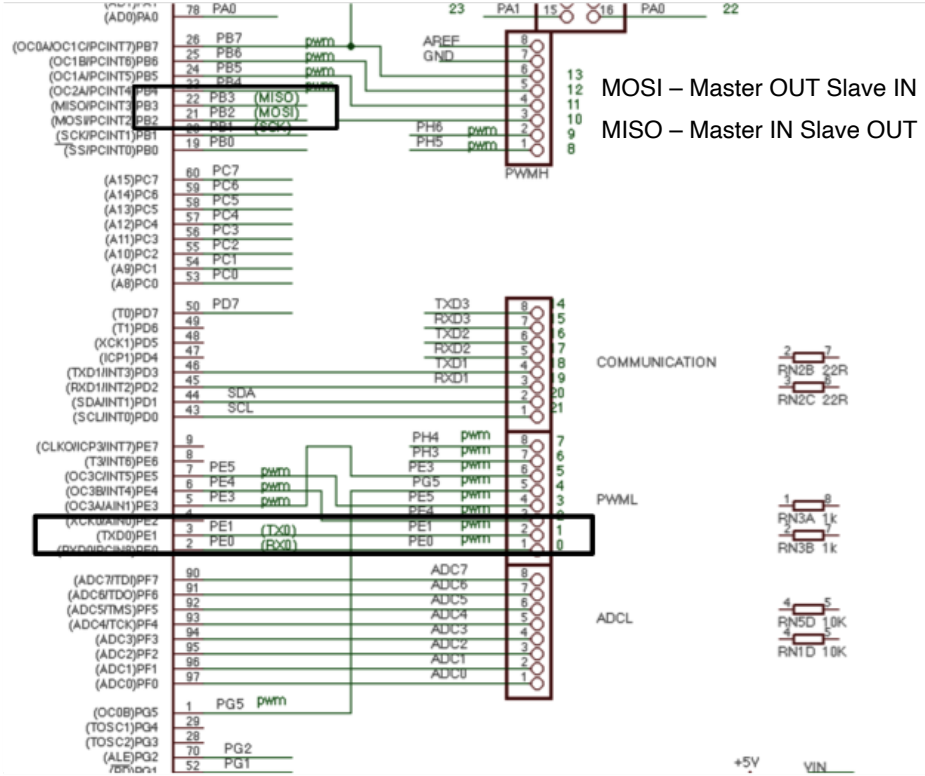
- USB 2.0

Pin No.	Standard	Mini	Micro
1	$V_{Bus} = 5\,\text{V}$	$V_{Bus} = 5\,\text{V}$	$V_{Bus} = 5\,\text{V}$
2	D−	D−	D−
3	D+	D+	D+
4	GND	NC	NC
5	N/A	GND	GND

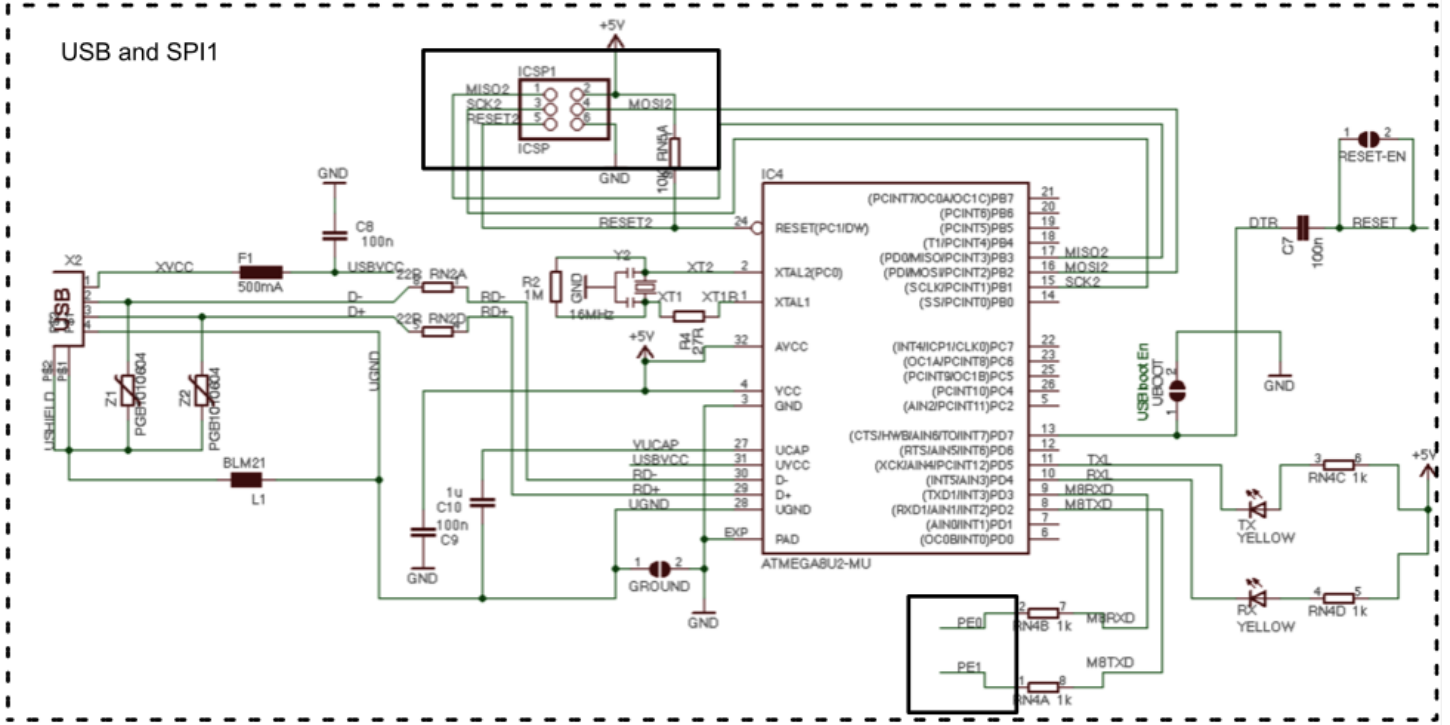
- USB 3 adds an additional twisted pair for full-duplex communication

The USB Interface on the Arduino

USB input and output is over USART 0
RX0- Pin 0, TX0 – Pin 1



ATMEGA8U2: MCU handles USB and ICSP1 I/O



Universal Asynchronous Receiver/Transmitter (UART)

- Physical layer that supports async protocols such as RS232, USB
- ATmega2560 includes USART - Universal *Synchronous/ Asynchronous* Receiver/Transmitter
  - Can be configured to generate clock signal

USARTs on the AVR

- USART – universal synchronous/asynchronous receiver transmitter
- On the AVR 2560, there are four USARTS labelled 0 through 3
- USART 0 is connected to the USB subsystem on the Arduino Mega
- Five registers are used for each USART for configuration and data transfer
- We will be focusing on asynchronous operations, at first using polling techniques
- Interrupt service routines (ISRs) will be introduced later

USART Programming Tasks

- Set desired BAUD rate (UBBR value)
- Configure the USART for TX and/or RX
- Configure character size for the session
- Set parity checking if desired
- Loop
  - TX: Wait until TX buffer is empty, then send a character
  - RX: Wait until RX buffer is full, then read character

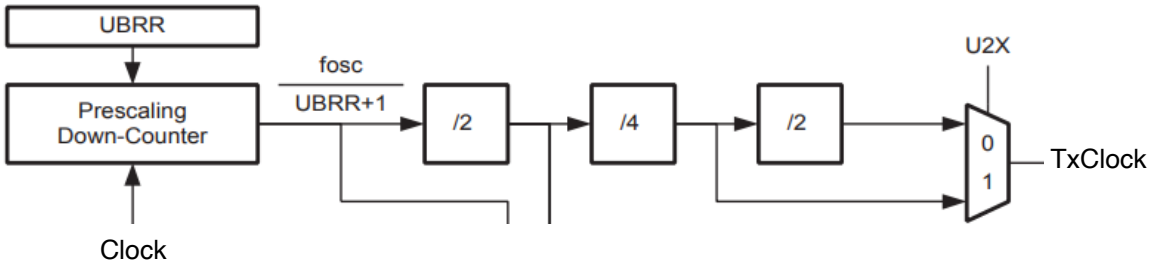
Registers Involved in UART Programming

- UBBR
  - Used to set baud rate for UART
- UCRSA, UCRSB, UCRSC
  - Control and status registers
- UDR
  - Used to send and received data

- https://www.arnabkumardas.com/arduino-tutorial/usart-register-description/

Setting the BAUD Rate - UBBR value

- UBBR register used to set the rate
  - The register consists of two 8 bit registers (H and L)
  - Only 12 bits are used – the other 4 are reserved
- The counter counts down from the UBBR value
- When it reaches 0, a clock pulse is generated
- The pulse train is further divided by 2, 8, or 16
  - 16 is the default



fosc = frequency of oscillator
BaudRate = fosc/(16 (UBBR + 1))

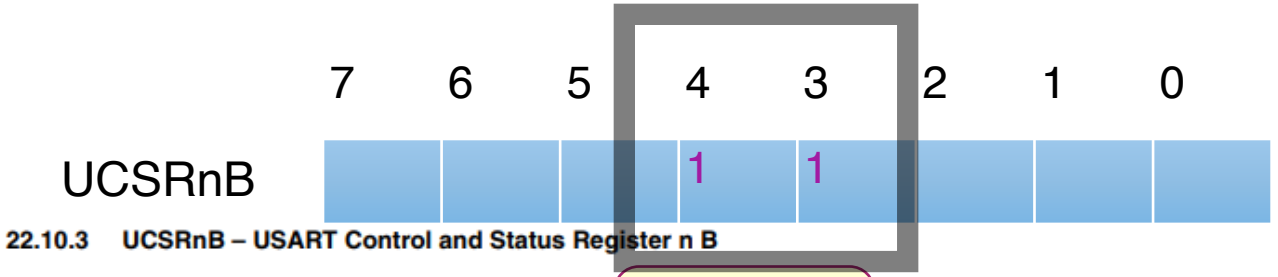
So
UBBR = fosc/(16 * BaudRate) – 1

For fosc = 16 MHz,

UBBR = (1000kHz/BaudRate) - 1
= 103 where BaudRate = 9600

Configuring the UART for Transmission/Reception

- The UCSRnB register allows enabling transmit and receive using bits 3 and 4, respectively



22.10.3 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	

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n**

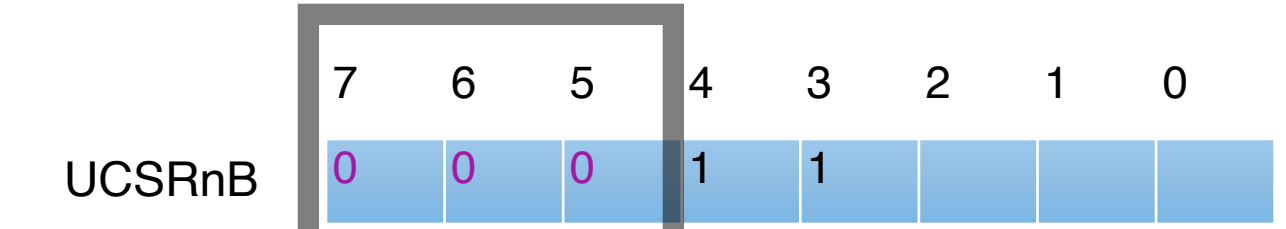
Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

Configuring the UART for Transmission/Reception

- The UCSRnB register allows enabling transmit and receive using bits 3 and 4, respectively
- Other bits (5,6,7) enable interrupts
  - We will be working with interrupts in the next module
  - Making 0 to disable interrupt
- Bits 0 and 1 are used in data frames that contain 9 bits



22.10.3 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	

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n**

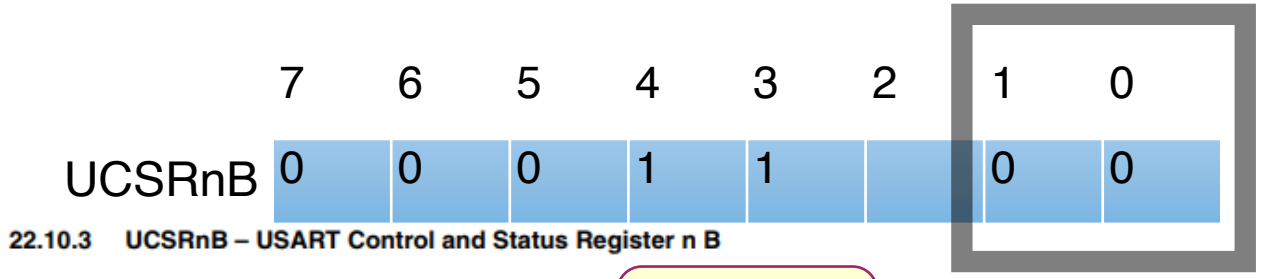
Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

Configuring the UART for Transmission/Reception

- The UCSRnB register allows enabling transmit and receive using bits 3 and 4, respectively
- Other bits (5,6,7) enable interrupts
  - We will be working with interrupts in the next module
  - Making 0 to disable interrupt
- Bits 0 and 1 are used in data frames that contain 9 bits from the UCSRnC register.



22.10.3 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	

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

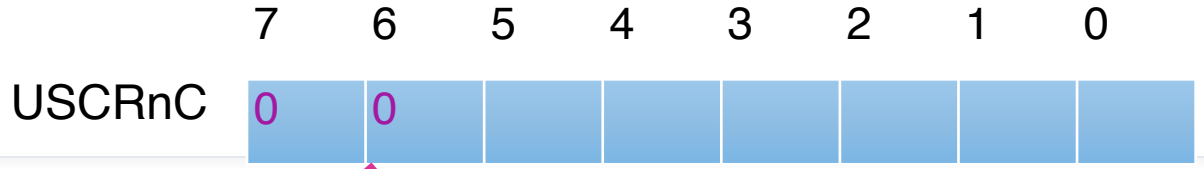
- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n**

Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

Setting the Mode, Parity, and Character Size



- UCSRnC is used to set session characteristics
- Bits 7:6 (UMSEL) set the UART mode
- Bits 2:1 (UCSZ) set the character length
  - For 9 bit characters, you must set UCSZn2 in the UCSRnB register
- Other bits set parity and number of stop bits (1 or 2)

Table 104. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	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

22.10.4 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	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 22-4.

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

- **Bits 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

Table 22-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Setting the Mode, Parity, and Character Size

	7	6	5	4	3	2	1	0
USCRnC	0	0				1	1	

- UCSRnC is used to set session characteristics
- Bits 7:6 (UMSEL) set the UART mode
- Bits 2:1 (UCSZ) set the character length
  - For 9 bit characters, you must set UCSZn2 in the UCSRnB register
- Other bits set parity and number of stop bits (1 or 2)

Table 104. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	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

Make sure to update the bit 0 and 1 in the USCRnB for the UCSZn2 value

22.10.4 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	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 22-4.

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

- **Bits 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

Table 22-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Setting the Mode, Parity, and Character Size

	7	6	5	4	3	2	1	0
USCRnC	0	0	0	0	0	1	1	0

- UCSRnC is used to set session characteristics
- Bits 7:6 (UMSEL) set the UART mode
- Bits 2:1 (UCSZ) set the character length
  - For 9 bit characters, you must set UCSZn2 in the UCSRnB register
- Other bits set parity and number of stop bits (1 or 2)

Table 104. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	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

22.10.4 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	

- **Bits 7:6 – UMSELn1:0 USART Mode Selec**

These bits select the mode of operation of the USARTn as shown in Table 22-4.

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

- **Bits 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

Table 22-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Make the remaining bits 0

Setting the Mode, Parity, and Character Size Example

- Configure transmission for
  - 8 bits with
  - one stop bit, at
  - 9600 Baud

Enabling Transmission,
UCSR0B bit 3 = 1
8 bits,
UCSR0C bit 2=1 , bit 1 = 1
UCSR0B bit 0 , 1 = 0
UBBR0L = 103

Table 104. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	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

Setting the Mode, Parity, and Character Size Example

- Configure transmission for
  - 8 bits with
  - one stop bit, at
  - 9600 Baud

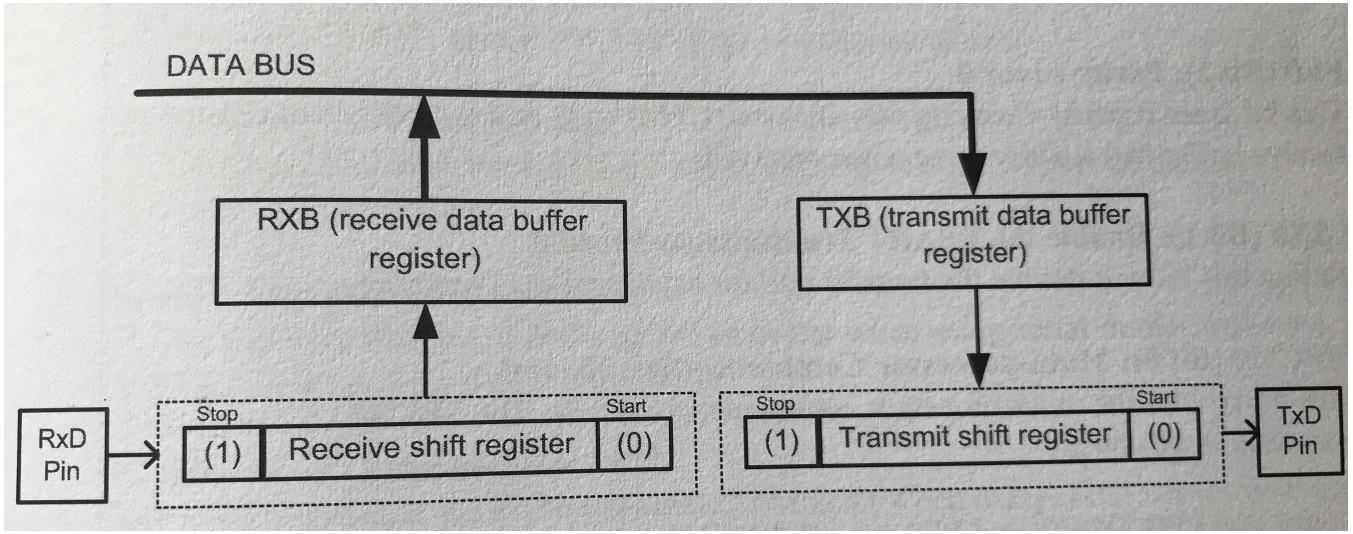
Enabling Transmission,
UCSR0B bit 3 = 1
8 bits,
UCSR0C bit 2=1 , bit 1 = 1
UCSR0B bit 0 , 1 = 0
UBBR0L = 103

```
volatile unsigned char *myUCSR0B = (unsigned char *)0x00C1;
volatile unsigned char *myUCSR0C = (unsigned char *)0x00C2;
volatile unsigned int  *myUBRR0L  = (unsigned int *) 0x00C4;
#define TXEN0 3
#define UCSZ00 1
#define UCSZ01 2

*myUCSR0B = (1 << TXEN0);
*myUCSR0C = (1 << UCSZ01) | (1 << UCSZ00)
*myUBRR0L = 103; // using decimal
```

Reading and Writing Data: The Data Register (UDR)

- Two registers at the same physical address
- Writing to this register writes to the TX buffer
- Reading this register reads from the RX buffer
- Read and write operations set flags in other registers that can be used to check register status



The AVR Microcontroller and Embedded Systems Using Assembly and C, M.A. Mazidi el al

Writing Data to the UART

- UCSRnA holds flags set or reset when transmission or reception is complete
- If UDREn bit (UCSRnA bit 5) is one, the buffer is empty, and therefore ready to be written.
- Write character to register UDRn

```
volatile unsigned char *myUCSR0A = (unsigned char *)0x00C0;
volatile unsigned char *myUDR0   = (unsigned char *)0x00C6;

#define UDRE0 5

void send(unsigned char ch) {
        while(!(*myUCSR0A & (1<<*myUDRE0));
        *myUDR0 = ch;
}
```

Reading Data from the UART

```
volatile unsigned char *myUCSR0A = (unsigned char *)0x00C0;
volatile unsigned char *myUDR0   = (unsigned char *)0x00C6;
#define UDRE0 5
#define RXC0 7

unsigned char read() {
        unsigned char ch;
        while(!(*myUCSR0A & (1<<RXC0));  // same as & 0x80
        ch = *myUDR0;
        return ch;
}
```

- Read operation uses the same UDR register
- Reads must occur after the RCX0 bit in the UCSRnA register (bit 7) has been set
  - This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty.
- Reading the RXC bit clears the bit

Reading

- Jiménez: Chapter 9.1 through 9.4
- Mazidi: Chapter 11, Chapter 18
- Atmel ATmega Datasheet: Chapter 22 (USART)
  - https://www.arnabkumardas.com/arduino-tutorial/usart-register-description/