Microprocessors

Fall 2020

9. Serial Communications

Tentative Weekly Schedule

- Week x1 - Introduction to Course
- Week x2 - Architecture
- Week x3 - Assembly Language Introduction
- Week x4 - Assembly Language Usage, Memory and Faults
- Week x5 - Embedded C and Toolchain
- Week x6 - Exceptions and Interrupts
- Week x7 - GPIO, External Interrupts and Timers
- Week x8 - Timers
- **Week x9 - Serial Communications**
- Week xA - Serial Communications
- Week xB - Analog Interfacing
- Week xC - DMA
- Week xD - RTOS
- Week xE - Wireless Communications

Review: Timers in STM32G0

There are different timer related modules in G0

Inside the ARM core

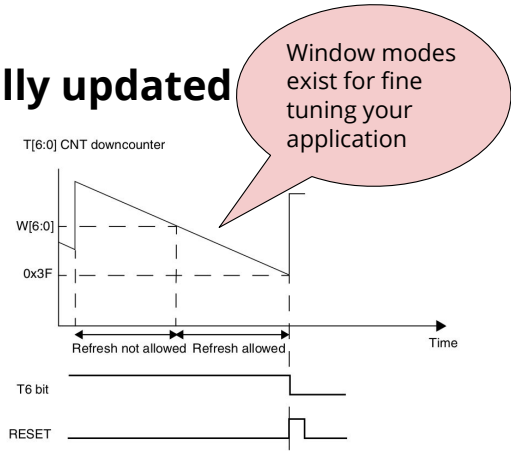
- SysTick Timer

As peripherals

- General-Purpose Timers
- Advanced-control Timers - various additional functionalities such as input capture, output compare and PWM
- Watchdog Timers
  - Independent WDG
  - Window WDG

Review: Watchdog Timer

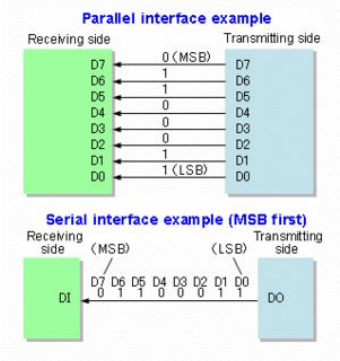
- To increase reliability, and prevent any lockup situations, vendors include a **Watchdog Timer** module that will **generate a reset** in case of **a software failure**.
- It needs to be **periodically updated**
  - or ***fed***, as in ***feeding the dog***
- In case it doesn't get updated in a given period of time, it will **generate a reset**



Serial vs. Parallel Communication

There are two approaches for transmitting data between devices.

- **Parallel communication** - data is chunked into multiple bits, and sent at the same time using multiple **same length** channels
  - Faster, more wires and I/O, properly length match in hw, crosstalk (EMI)
- **Serial communication** - data is chunked into bits, and sent **one bit at a time** using one channel
  - Slower, one wire

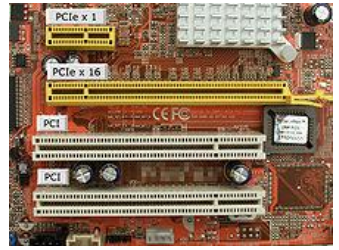


Parallel Communication Systems

Parallel Communication is used especially in computer systems. Some of the examples are:

- **ISA** (*Industry Standard Architecture*) - 16-bit internal bus of IBM PC/AT and similar computers. Superseded by **PCI**

- **PCI** (*Peripheral Component Interconnect*) - a local computer bus for attaching hardware devices in a computer and is part of the PCI Local Bus standard. Superseded by **PCI-Express**

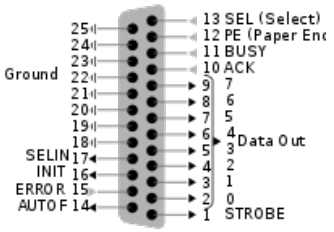


Parallel Communication Systems

- **PATA** (*Parallel Advanced Technology Attachment*) - developed in 1980s used to connect hard drives, CD or DVD drives. Superseded by **Serial ATA**

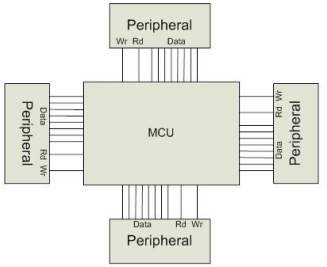
- **Parallel Port** - a type of interface found on computers (personal and otherwise) for connecting peripherals. Originally used for printers.

DB-25 connector

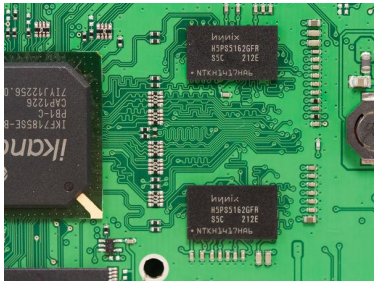


Parallel Communication Problems

- **More wires, I/O, channels** - requires multiple lines / channels connecting to a device.

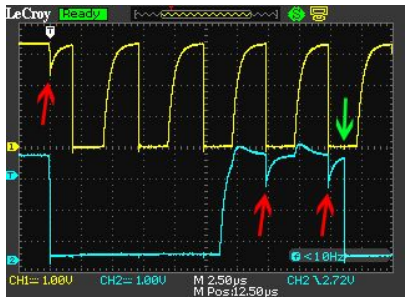


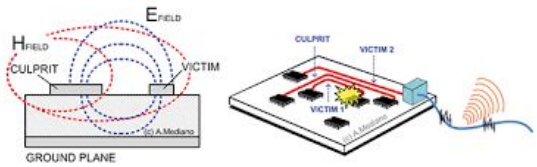
- **Properly length match of each signal in hw** - causes synchronization problems if not done properly



Parallel Communication Problems

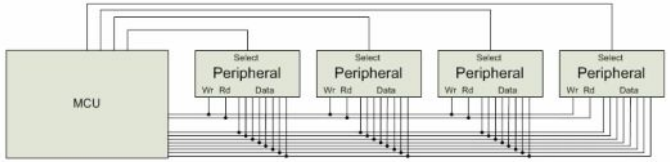
- **Crosstalk** (Electromagnetic interference) - a change in one line might and can affect the other line
  - **Erratic** behavior (false logic 1 / 0 detection)





Parallel Buses

- All devices use the same bus to read and write data
- MCU can use individual select lines to address each peripheral.
  - Can also arrange it to assign address blocks
- MCU can communicate with only one peripheral at a time
- Only one device can communicate at a time

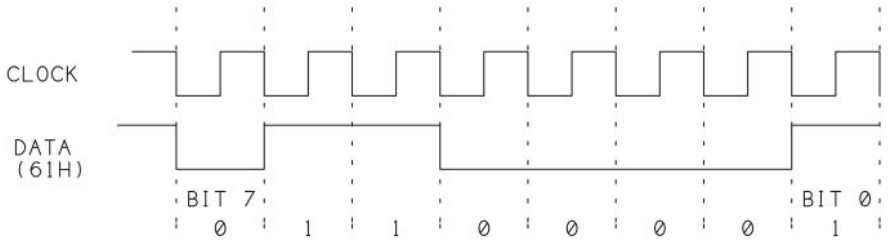


Serial Communication Protocol Types

- **Synchronous** - includes a clock line
  - typically controlled by one device, and all transmitted bits are synchronized to that clock
  - each transmitted bit is valid at a defined time after a clock's rising, or falling edge depending on the protocols
- **Asynchronous** - does not include a clock line. Instead each computer needs to provide its own clock source for timing reference
  - Usually devices must agree on a clock frequency beforehand, and the actual frequencies must be very close to the agreed frequency
  - Requires a start condition (start bit) to synchronize the clocks

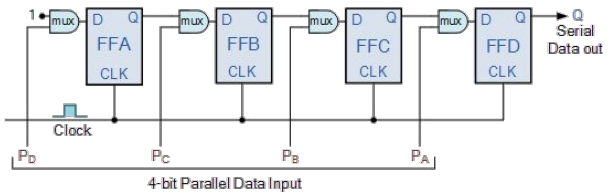
Synchronous Communication

- Transmitter generates a clock and starts sending the bits. An example transmission can happen:
  - Transmitter sends bits on clock's falling edge
  - Receiver reads bits on clock's rising edge
- Synchronous protocols usually **send MSB first**

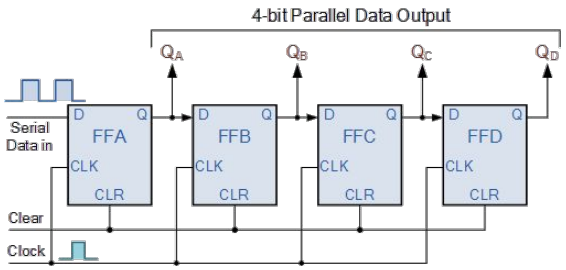


Synchronous Communication - HW

- Transmitter Hardware Design



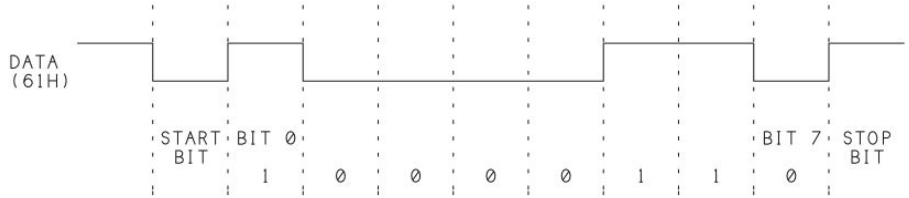
- Receiver Hardware Design



Shift Registers

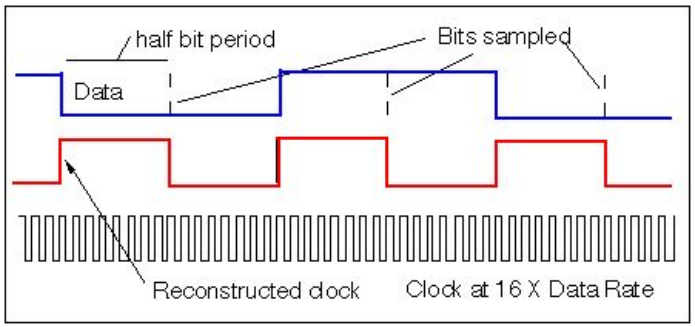
Asynchronous Communication

- Transmitter uses an internal clock to determine when to send the next bit. (i.e. using a state machine) (**start condition**)
- Receiver detects the falling edge of the start and starts its internal clock to read the following bits.
- Asynchronous protocols usually **send LSB first**
- After the bits are transferred a stop bit sent (**stop condition**)



Asynchronous Communication - HW

- Have an internal counter that will start when the start condition happens (i.e. falling edge)
- Count 8x or 16x faster than the agreed transmission speed
- Sample the line in the middle of the bit length

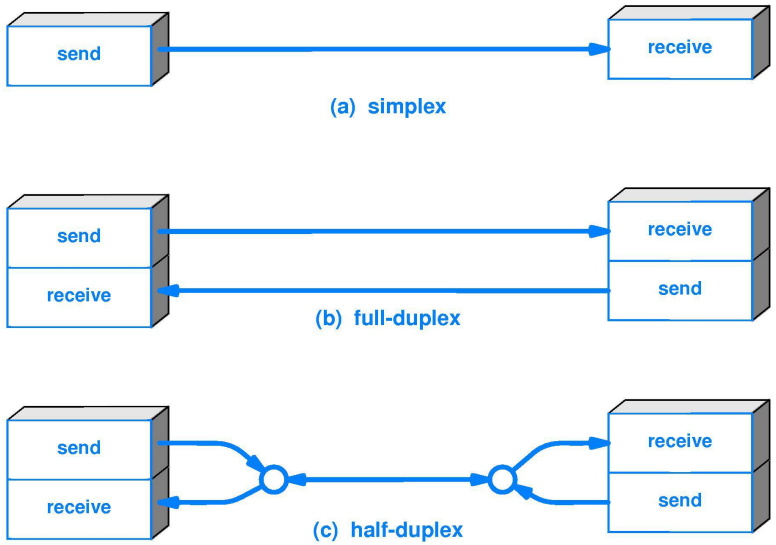


Serial Comm. Transmission Modes

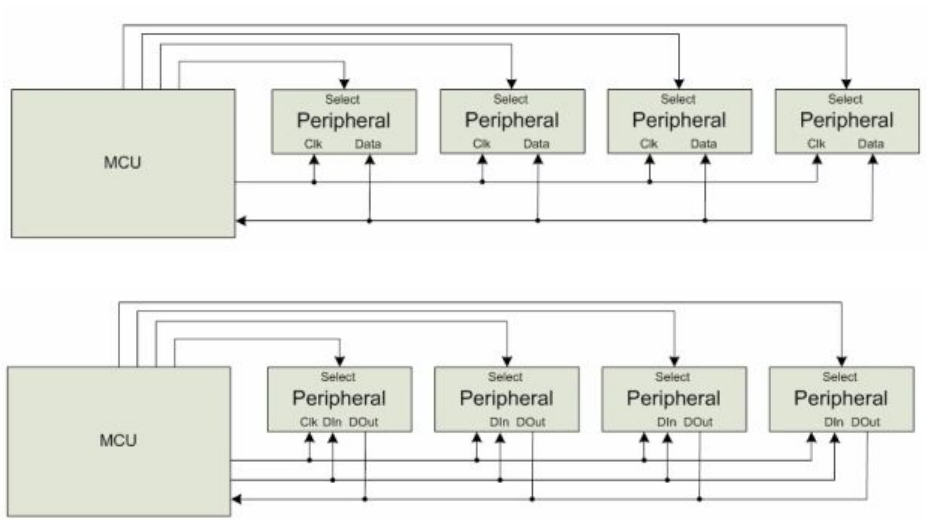
There are a couple transmission modes depending on the type of transmission, connection and data transfer

- **Simplex** - the communication can take place only in one direction, meaning a device can either send or receive data in a unidirectional method
- **Half-Duplex** - the communication can happen in both directions, but only one direction is active at any time. A device can both send **or** receive, but not both at the same time
- **Full-Duplex** - the communication can happen in both directions simultaneously, meaning the device can both send **and** receive at the same time

Serial Comm. Transmission Modes

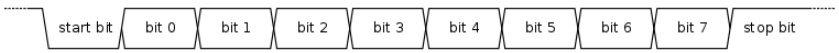


Synchronous Half/Full Duplex Data Bus



Serial Comm. Protocols - Concepts

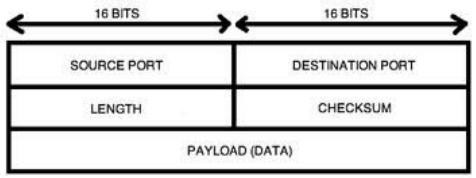
- **Bit rate** is expressed as the number of transmitted bits per second (bps)
- **Baud rate** is the number of possible events or data transitions per second. Usually they are the same
- In simple protocols, transmission happens in **8-bits** (octets, words) of data with start and stop conditions called a **frame**.



- Optional error checking bit is added for catching bit errors called **parity**

Serial Comm. Protocols - Concepts

- Network/Bus protocols have more information in each data frame
  - **Medium access control** - when multiple nodes are on the bus, they must arbitrate for permission to transmit
  - **Addressing information** - to determine the message recipient
  - **Larger data payload**
  - Stronger **error detection** or **error correction** information
  - Request for immediate response



Error Detection

- Can send additional information to verify data was received correctly
- Need to specify which parity to expect: even, odd or none
- Parity bit is set so that the total number of 1's in data and parity is even for even parity, and odd for odd parity
  - **01010101** has 4 1's, **0** for **even, 1** for **odd** parity bit
  - **10010001** has 3 1's, **1** for **even, 0** for **odd** parity bit
- Single parity bit detects odd number of corrupted bits
- Stronger error detection codes (e.g. Cyclic Redundancy Check) exist and use multiple bits, and can detect many more corruptions
  - Used for CAN, USB, Ethernet, Bluetooth

Questions

- **Q.** What is the number of **words** (8-bits) that can be sent in one second using the following transmission scheme with a **9600** baud rate?

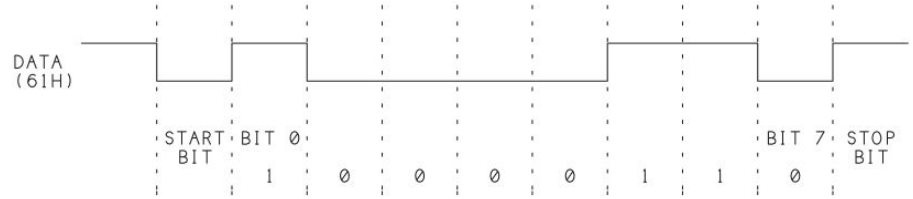
  `8d + 1st + 1 sp = 10` bits each word
  `9600 bps / 10 = 960` words per sec

- **Q.** What is the **length of one bit**?

  `time 1 bit = 1/9600 = 104 us`

- **Q.** What is the **length of transmission?**

`time 1 tx = t_1b*10 = 10.4 us`



Serial Communication Protocols

There are many protocols, some of the popular ones are

- **RS-232**, **UART**
- **I2C**, **SPI**,
- Ethernet
- USB
- CAN bus
- Microwire
- 1-Wire
- PCI Express
- Serial ATA

RS-232

- Formed around 1960s for **connecting modems with terminals**
- Between Data Communication Equipment (**DCE**) and Data Terminal Equipment (**DTE**)

- Later, it is used for connecting many peripherals, TVs computers, DVRs, Oscilloscopes, etc.
- Mostly superseded by USB

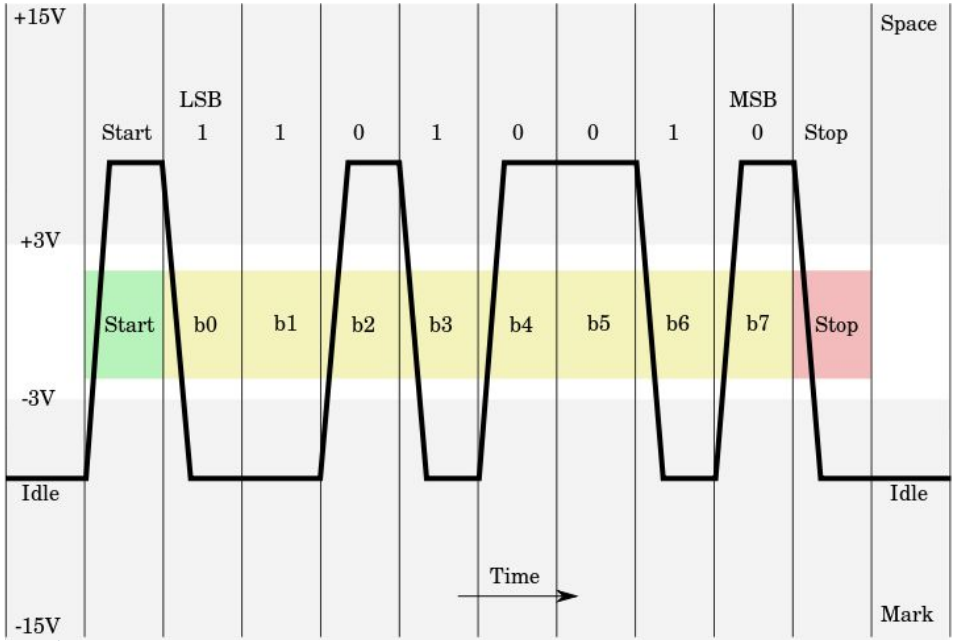
RS-232 Signaling

- Standard defines
  - Electrical signal characteristics such as voltage levels, signaling rate, timing, and slew-rate of signals, voltage withstand level, short-circuit behavior, and maximum load capacitance.
  - Interface mechanical characteristics, pluggable connectors and pin identification.
- Works with +/- 15V range
  - `+3 - +15` volts represent a logic `0`
  - `-3 - -15` volts represent a logic `1`

RS-232 Signaling

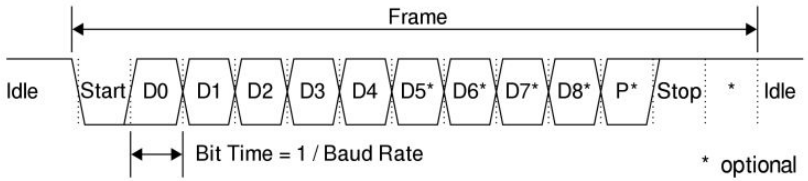
- High voltage values make it more tolerant to noise with simple cabling, and can go to longer distance
- LSB first
- **Idle** at logic 1
- **Start condition** is a logic 0
- **Stop** condition is a logic 1
- 1 line for transmit, and 1 line for receive
  - Full-duplex possible, usually half-duplex implementation
- Additional signals are used for **hardware flow control**

RS-232 Signaling



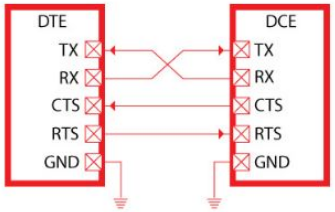
Universal Asynchronous Receiver Transmitter (UART)

- **Asynchronous** serial communication
- Uses **TTL** levels with either 0 - 3.3V or 0 - 5V
- Data is transmitted at a specific **baud rate** with **LSB first**
  - up to 1 Mbps
  - common ones are **9600** and **115200** bps
- Point-to-point communication with two pins. **TX** and **RX**
- Configurable data bit size - 5, 6, 7, 8, 9
- Start and stop bits with optional parity bit
  - Most common is **8N1** - 8 bits, No parity, 1 stop bit
- **Idle** is logic 1, **start** is logic 0, **stop** is logic 1



UART Connection

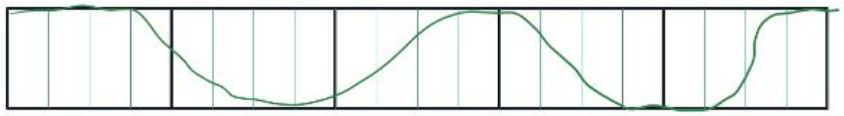
- One transmitter and one receiver
- Not a bus, TX is connected to RX of the other device and vice versa



- CTS (clear to send), RTS (ready to send) optional flow control. Not used for most cases
- Common ground
- Baud rate needs to be pre-set
  - There are auto-detection methods such as sending a 0x55 to determine the speed.

Input Data Oversampling

- When receiving, some modules oversample the incoming data line
  - Extra samples allow voting improving noise immunity
  - Better synchronization to incoming data



Using the UART

- When transmitting
  - Transmit peripheral must be ready for data
  - Can poll the status register
  - Can use an interrupt with some kind of **data structure** (even simple variable sharing)
- When receiving
  - Receive peripheral must have data
  - Can poll the status register
  - Can use an interrupt with some kind of **data structure** (even simple variable sharing)

Using the UART - Polling example

```
void uart_polled_example(void) {
    uart_init(115200);
    while(1) {
        // read incoming data, and transfer it back. echo
        uart_tx(uart_rx());
    }
}
```

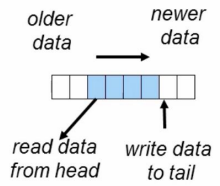
- `uart_rx()` function should have a proper timeout mechanism so that it does not stuck in polling the status register.
- A timer (i.e. SysTick) can be used to decrement a counter for timeout

Example Receiver: Display data on LCD

```
// Assume we have an LCD that is 2x16, and initialized
// using polling

int line = 0, col = 0;
while(1) {
    char c = uart_rx();
    LCD_set_cursor(col, line);
    LCD_put_char(c);
    if (++col > 15) {
        col = 0;
        if (++line > 1) {
            line = 0;
}}}
```


Example Receiver: Interrupt Handler

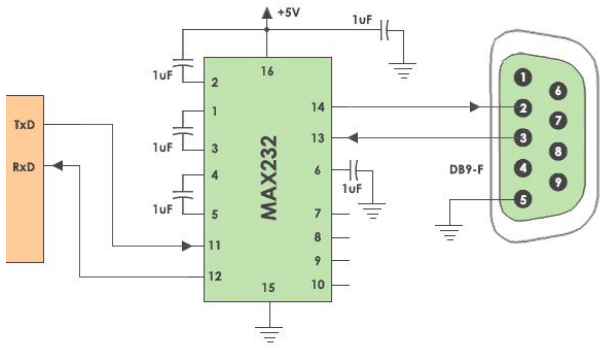
```
// Assume we implemented a queue data structure.
// We can read from UART and send it to queue.
// Process the queue item from some other routine.
Queue rx_queue;

void UART_ISR() {
    // read from receive data
    // register and send it to queue
    uint8_t c = UART->DR;
    queue_enqueue(&rx_queue, c);
}

int main(void) {
    BSP_init();
    queue_init(&rx_queue, 64);
    uart_init(115200);
    //..

}
```

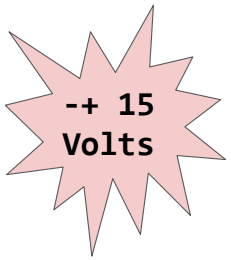



```
typedef struct {
  /* Data array, store it in heap */
  uint8_t *data;
  /* Array index of the oldests element */
  uint16_t head;
  /* Array index of the newest element */
  uint16_t tail;
  /* Size of the data array */
  uint16_t size;
} Queue;
```


RS-232 to UART

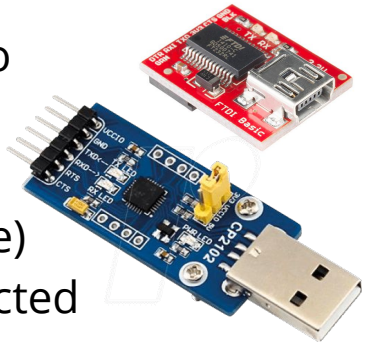
- Some devices still have RS-232 connectors.
- Need a level converter to change voltage levels. (Charge pumps)
- Never directly connect to an RS-232 port!





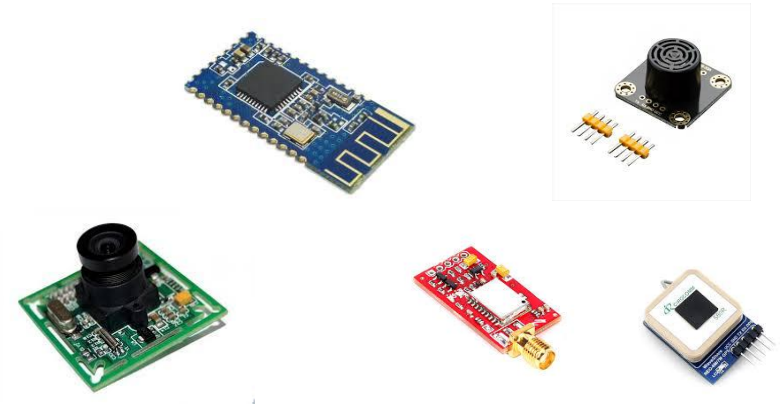
USB to UART

- Most devices nowadays use USB port.
- We can use a USB to UART bridge to convert USB signals to UART signals
- Knockoffs exist which can have labels mixed with TX / RX
  - A good rule of thumb is to probe the lines and the one that is on logic 1 is the TX (from the PC side) when the device is connected



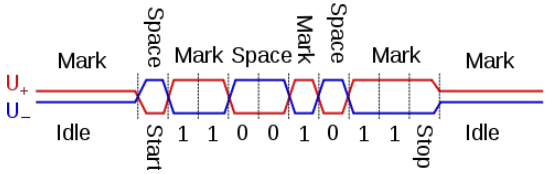
UART Usage

- Can be used with various modules such as Bluetooth, GPS, Camera, and ultrasound sensors
- Also common to connect to MCUs together



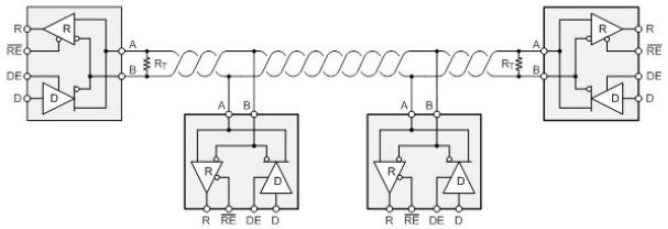
RS-485

- Developed in 1983, and defines electrical interface, does not define a communication protocol
- Allows for robust transmission over long distances in multipoint communication
- Used in industrial control systems such as factory automation
- Improves noise immunity and extends the distance supported by RS-232 by implementing a **differential** signaling technique
- Allow bidirectional communication

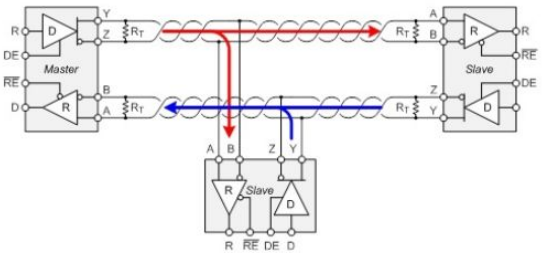


RS-485

- Half-duplex connection

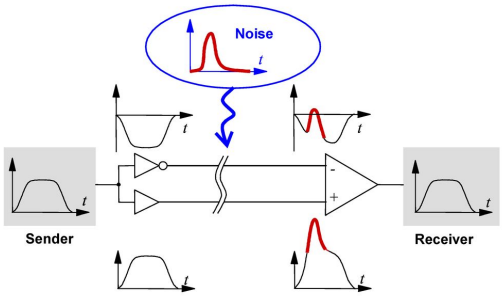


- Full-duplex connection



Differential Signaling

- Differential signaling is a method for electrically transmitting information using two complementary signals.
- Sends the same electrical signal as a differential pair
- The receiving circuit responds to the electrical difference between the two signals rather than the difference between a single wire and ground
- Can be found in PCBs, twisted-pair and ribbon cables



UART Alternatives

- Sensitive to noise and signal degradation.
- Fine for point-to-point communications, but if need to talk to more devices, becomes problematic
  - two lines per device for the MCU which can quickly fill up the available I/O
- MCU manufacturers have been developing their own serial communication systems
  - **I2C** - Inter Integrated Circuit communication
  - **SPI** - Serial Peripheral Interface
- Many MCUs support these interfaces