Lecture 11
Wired Communication: UART

CE346 – Microcontroller System Design

Branden Ghena – Fall 2024

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley), Alanson Sample (Michigan)

Administrivia

- Project purchasing is ready to go!
  - Send me requests and I'll start buying stuff tomorrow
  - 35 days, counting holidays and weekends, until Project Demo Day
    - Remember that shipping takes a couple days…

- Quiz today!

Today’s Goals

- Explore tradeoffs in wired communication
  - Signals, Speed, Timing, Topology
- Describe wired serial communication protocol: UART
- Discuss nRF52 implementation of UART

Outline

- **Wired Communication**
- UART
- nRF52 UARTE

Purpose of communication

- Goal: convey digital information between two devices

- Simple solution
  - Digital I/O pin – 1-bit of information

- Complex solution
  - Send multiple bits (arbitrarily many)

  - While also minimizing
    - Time, Energy, Pins, Errors, etc.

Wired versus wireless communication

- Wired
  - Send digital signals across one or more wires
  - Advantages: Reliable, Low energy, Often simpler topology
  - Disadvantages: Physically limiting

- Wireless
  - Send digital signals across another medium (usually RF)
  - Advantages: Physically flexible,
  - Disadvantages: Unreliable, High energy, Usually broadcast

Wired versus wireless communication

- Wired
  - Send digital signals across one or more wires
  - Advantages: Reliable, Low energy, Often simpler topology
  - Disadvantages: Physically limiting
  - **Today + next two lectures: UART, I2C, SPI, USB**

- Wireless
  - Send digital signals across another medium (usually RF)
  - Advantages: Physically flexible,
  - Disadvantages: Unreliable, High energy, Usually broadcast
  - **After that: Wireless Communication**

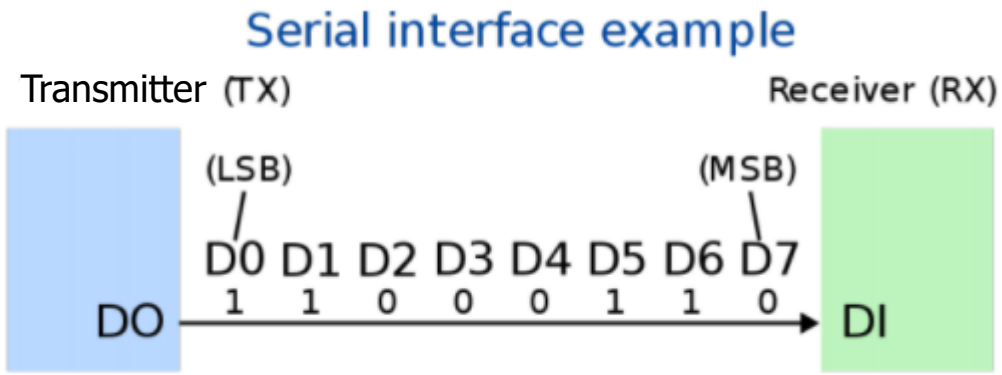
Tradeoffs in Wired Communication

- Number of signals
- Communication speed
- Controlling timing
- Network topology

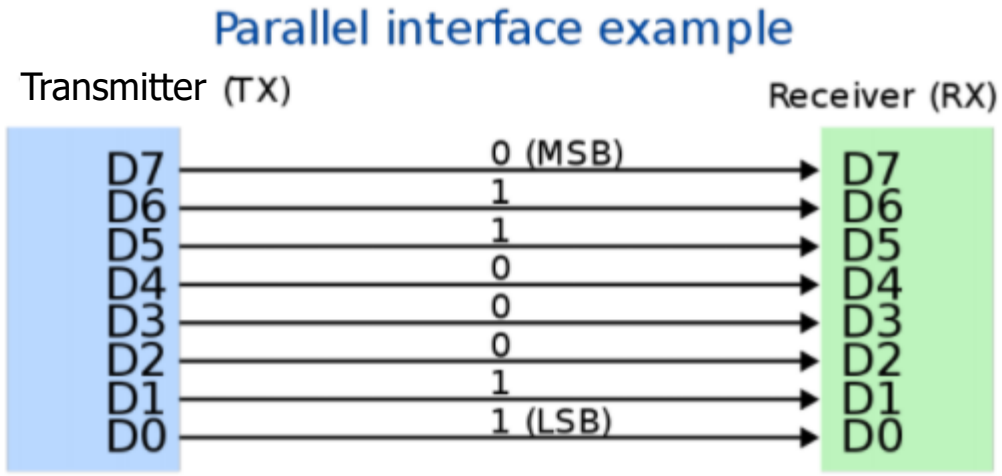
Let’s talk about each of these in the coming slides

Tradeoff: number of signals

- Serial interface
  - Single wire
  - Transmit data as a "series" of bits separated by time



- Parallel interface
  - Multiple wires
    - How many depends on the system
  - Transmit data across multiple "parallel" wires simultaneously
  - Still separate by time for more data than wires



Serial versus parallel

- Serial
  - Cheaper to use less wires
  - Slower to transmit data
  - Examples
    - RS-232, UART, I2C, USB (2.0)

- Parallel
  - More expensive for more pins and wires
  - Faster to transmit data
  - Examples
    - Internal buses, PCI, USB (3.0)

Tradeoffs in Wired Communication

- Number of signals
  - Parallel versus Serial
- **Communication speed**
- Controlling timing
- Network topology

Tradeoff: communication speed

- Round-trip-time inherently limited by the speed of light
  - Speed of electricity 50-99% of that
  - 29 cm (11.4 in) = 1 nanosecond
    - Totally relevant for Gbps speeds within computers

- Also limited by interference
  - Faster signals are harder to distinguish
  - More susceptible to interference (matters less for wired comms)

- Limited by whether the other device can keep up
  - Might need some flow-control signaling to slow down when not ready

Example communication speed

- Internal, low-energy communication
  - UART, 1-1000 kbps
  - I2C, 100-400 kbps
  - SPI, 1-100 Mbps
- External (mostly serial) communication
  - USB, 1-10000 Mbps
  - Ethernet, 1-1000 Mbps
  - HDMI, 4-48 Gbps
- Internal parallel communication
  - PCI, 8-32 Gbps
  - RAM, 12-25 Gbps

Note: Speeds are always measured in **bits per second**

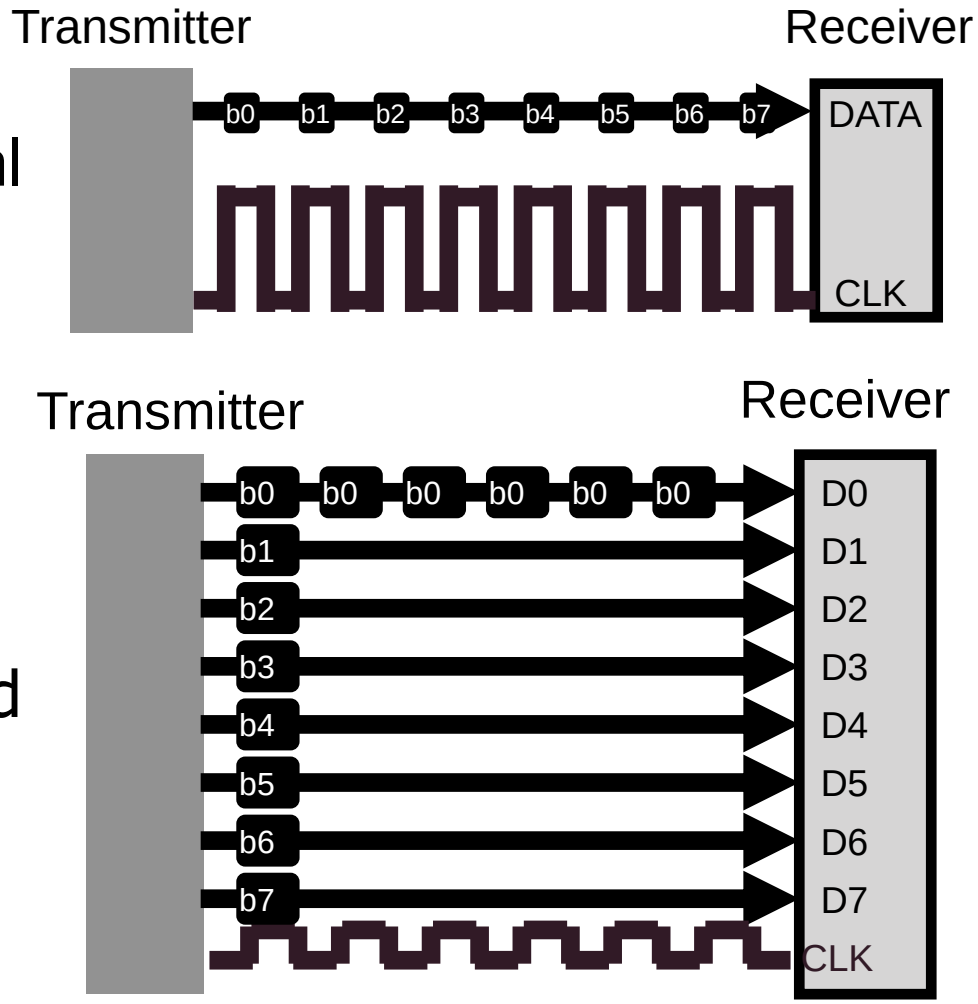
Tradeoffs in Wired Communication

- Number of signals
  - Parallel versus Serial
- Communication speed
  - 1000 bps to 10000000000 bps
- **Controlling timing**
- Network topology

Tradeoff: controlling timing

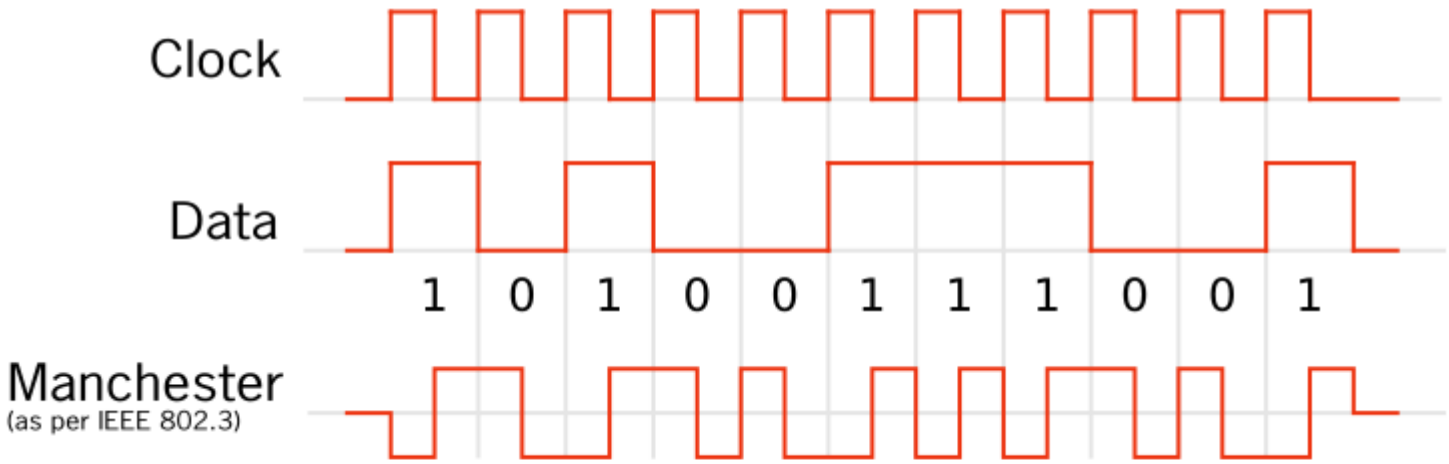
- Synchronous communication
  - Clock signal sent along with data
  - Data is captured at edge of clock signal (rising or falling)
  - Advantage: send signals very fast
  - Disadvantage: extra pin and wire
- Asynchronous communication
  - Agree upon timing in advance and read data at that rate
  - Advantage: no need for clock signal
  - Disadvantage: clock synchronization

Synchronous systems can be Serial (top) or Parallel (bottom)



Compromise: combining signals and clocks

- There is a method of recovering the clock from the signal
  - Either the clock is directly encoded in the signal
  - Or the signal will have mandatory high/low changes to synchronize



https://en.wikipedia.org/wiki/Clock_recovery
https://en.wikipedia.org/wiki/Self-clocking_signal

Tradeoffs in Wired Communication

- Number of signals
  - Parallel versus Serial
- Communication speed
  - 1000 bps to 10000000000 bps
- Controlling timing
  - Synchronous versus Asynchronous
- **Network topology**

How to connect: point-to-point networks

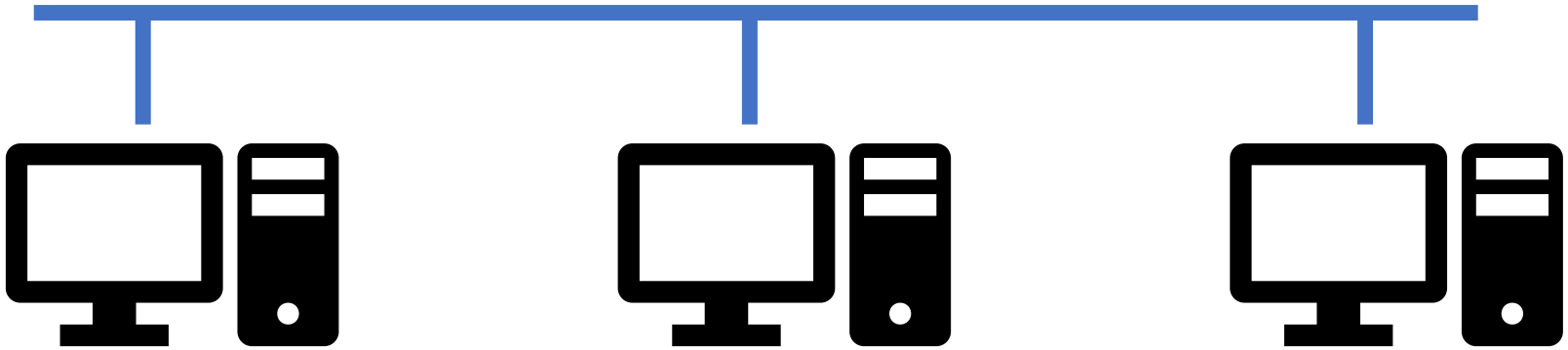
- How do we connect computers in a network?
  - This is a question of "network topology"
  - Simple option: just connect them directly

- Problem: what if I find a third computer?

How to connect: bus networks

- Connect everything to one wire in parallel
  - Actually a “multidrop bus”
  - Scales pretty well to many computers

- Problem: which computer gets to transmit when?
  - Simultaneous transmissions conflict
  - Need a scheme for “arbitration”, deciding who transmits when



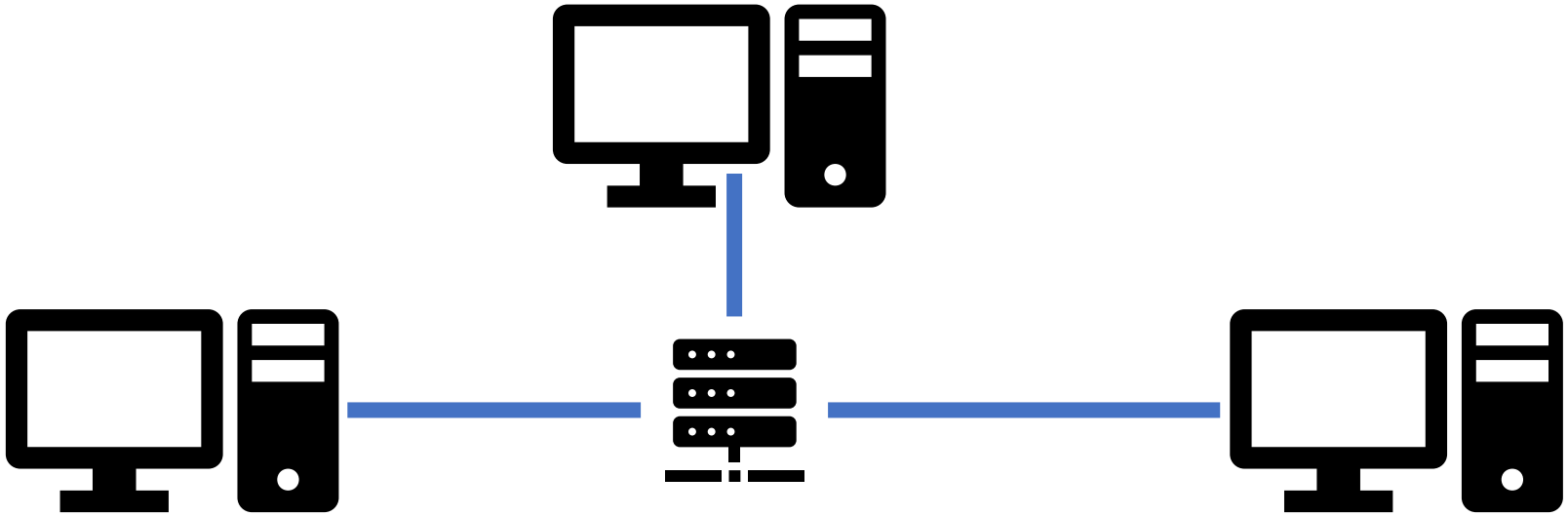
How to connect: ring networks

- Connect everything with point-to-point connections
  - Connect the last computer back to the first computer
  - Also known as Daisy-Chain (without the last connection back to the start)

- Problem: what if a computer stops sending?

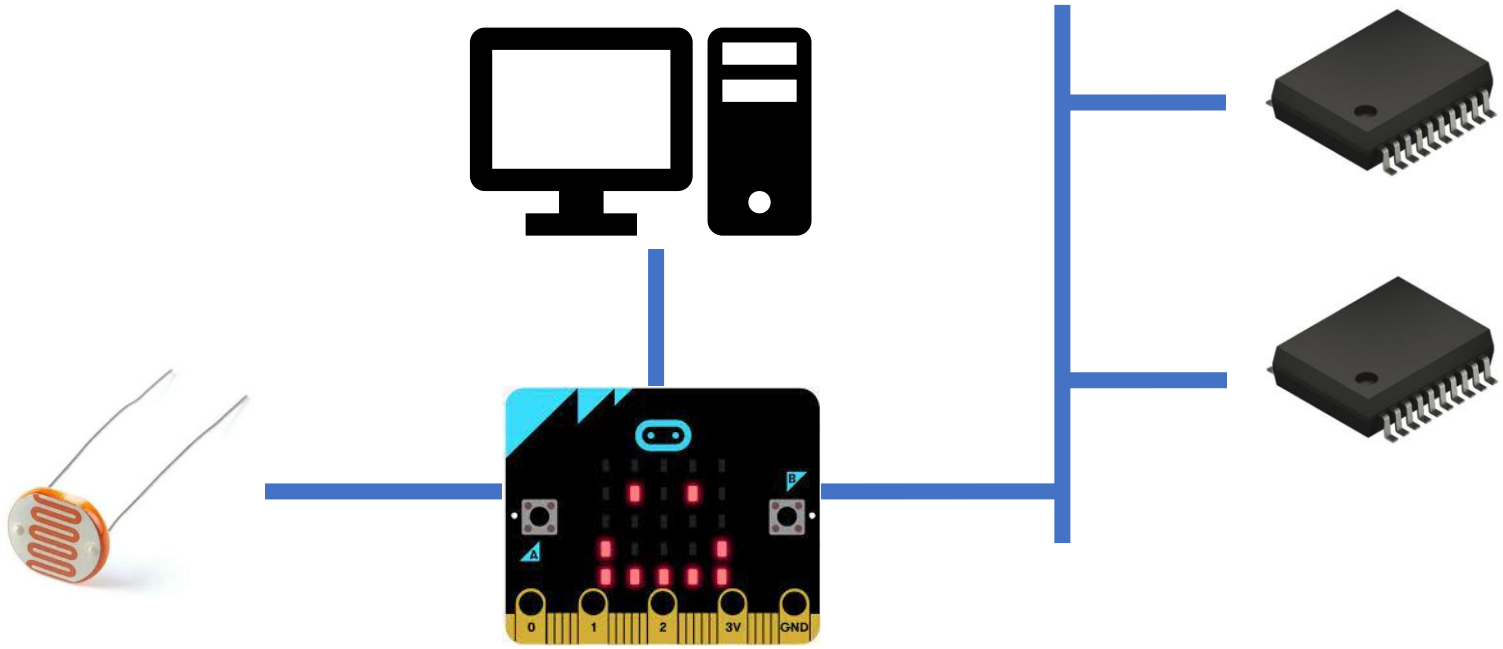
How to connect: star networks

- Connect to a hub with point-to-point connections
  - Hub connects all computers
  - Hub is a simple computer with one job: transfer communications between computers
    - Hopefully more reliable than any of the computers



Microcontrollers are often hubs of star networks

- Connect to multiple different sensors
- Sometimes a few sensors are connected on a bus



Tradeoffs in Wired Communication

- Number of signals
  - Parallel versus Serial
- Communication speed
  - 1000 bps to 10000000000 bps
- Controlling timing
  - Synchronous versus Asynchronous
- Network topology
  - Point-to-Point, Bus, Ring, Star

Break + Open Question

- What network topology is WiFi?
  a) Point-to-Point
  b) Shared Bus
  c) Ring
  d) Star

Break + Open Question

- What network topology is WiFi?
  - ~~a) Point-to-Point~~
  - **b) Shared Bus**
  - ~~c) Ring~~
  - ~~d) Star~~ (but close)

- Any wireless device can hear any other wireless device (in range)
  - BUT, it tries to emulate a Star topology, where each device only talks to the router

Outline

- Wired Communication
- **UART**
- nRF52 UARTE

UART Overview

- Universal Asynchronous Receiver Transmitter
  - Serial communication between two devices
  - Two wires: transmit and receive
  - Simple to implement and very common on microcontrollers

- Tradeoff choices: Serial, Low speed, Asynchronous, Point-to-Point

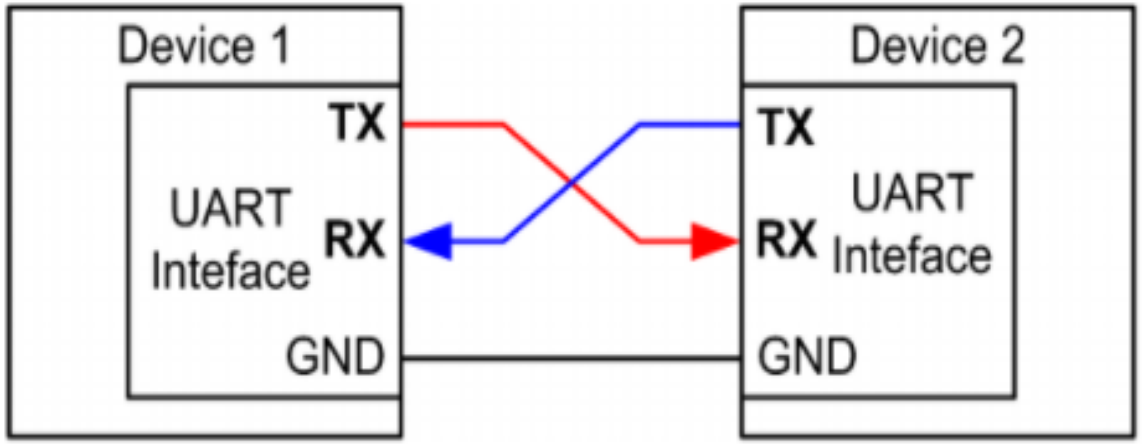
- Most frequently used to send text data between devices
  - Microcontroller **`printf()`** output
  - GPS to microcontroller
  - Radio AT commands to/from microcontroller

UART: chip-to-chip communication

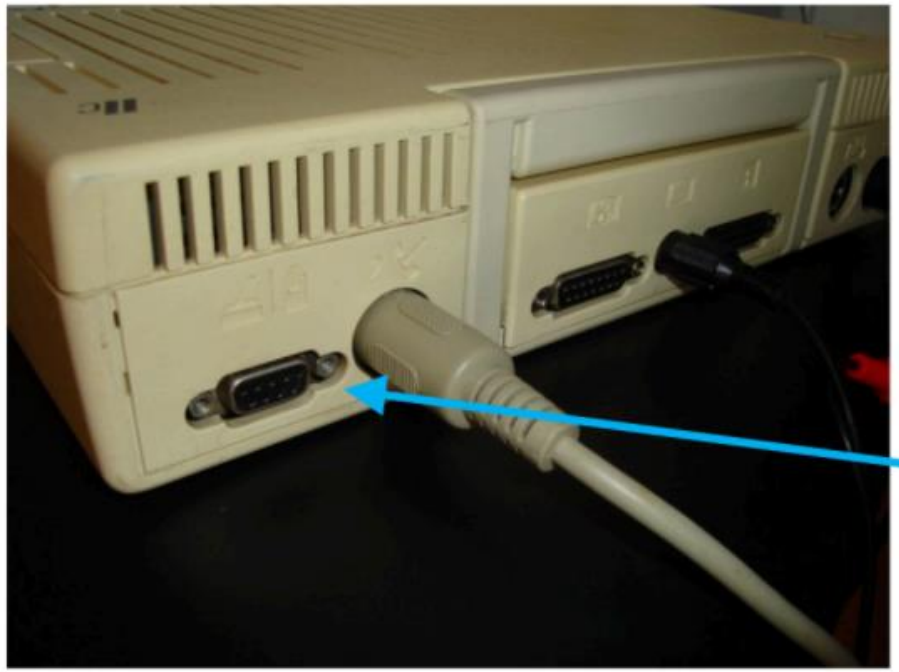
- Usually implemented as a two-wire interface
  - TXD: Transmits data
  - RXD: Receives data
  - Optionally two additional pins for flow control

  - No clock signal! Asynchronous

- Note: TX connects to RX (you'll always get this wrong on the first try)



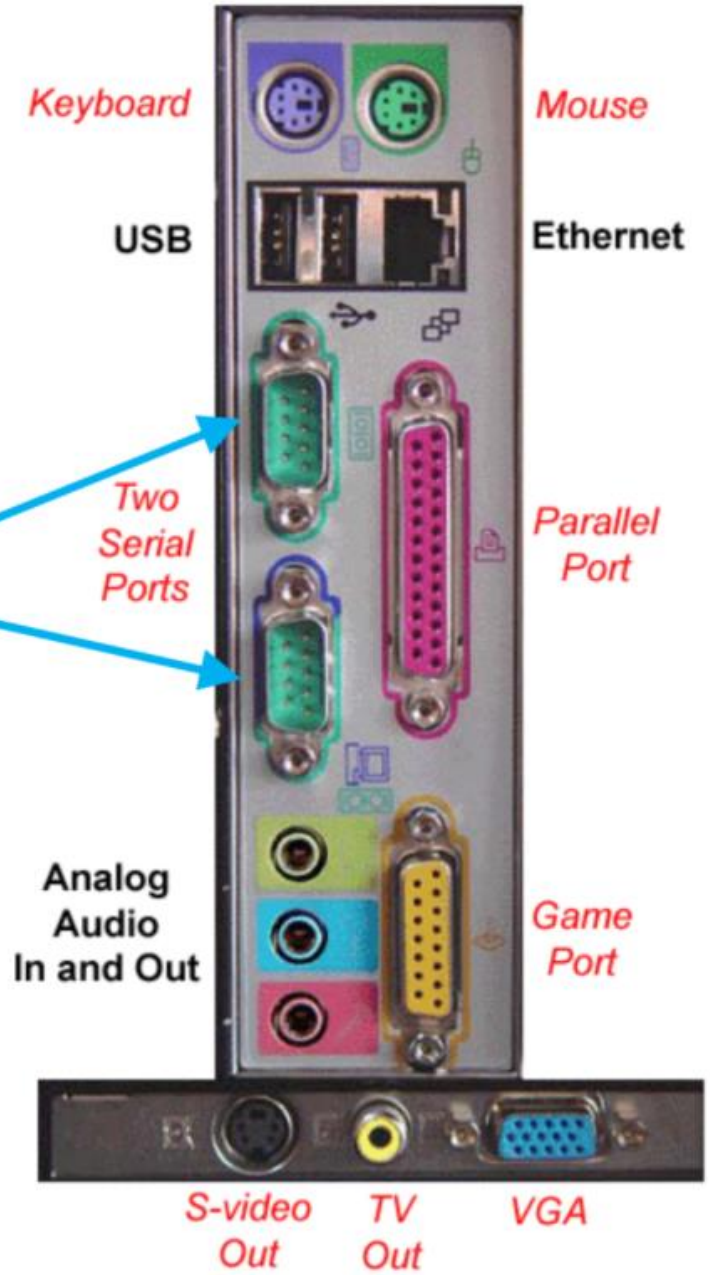
A little history…



Western Digital developed the first widely available single-chip UART (the WD1402A) around 1971.

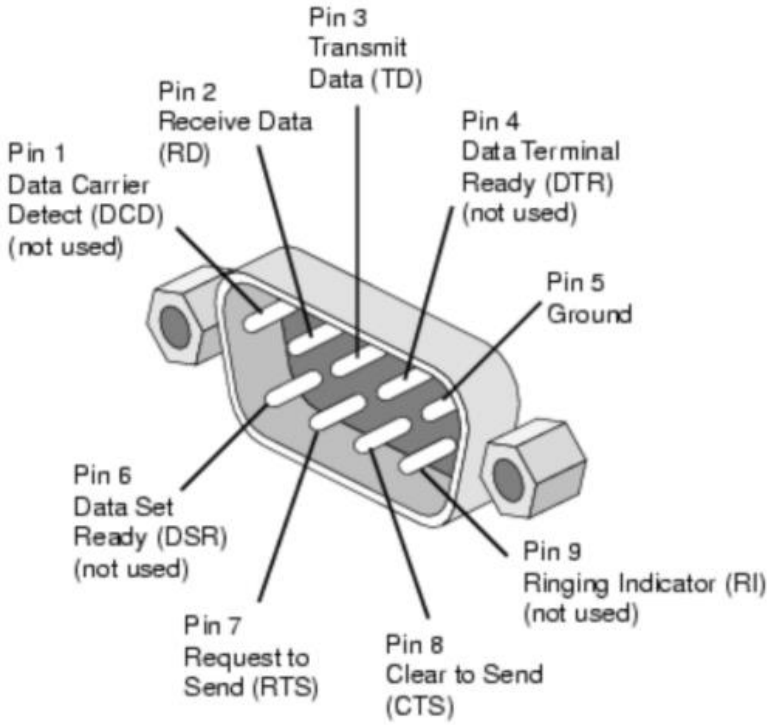
UART: Protocol
RS-232: Standard
DB9: Connector

Starting in the 2000s, most computers removed their external RS-232 COM ports and used USB ports that provided superior bandwidth performance.



UARTs is still widely used today in Billions of devices!

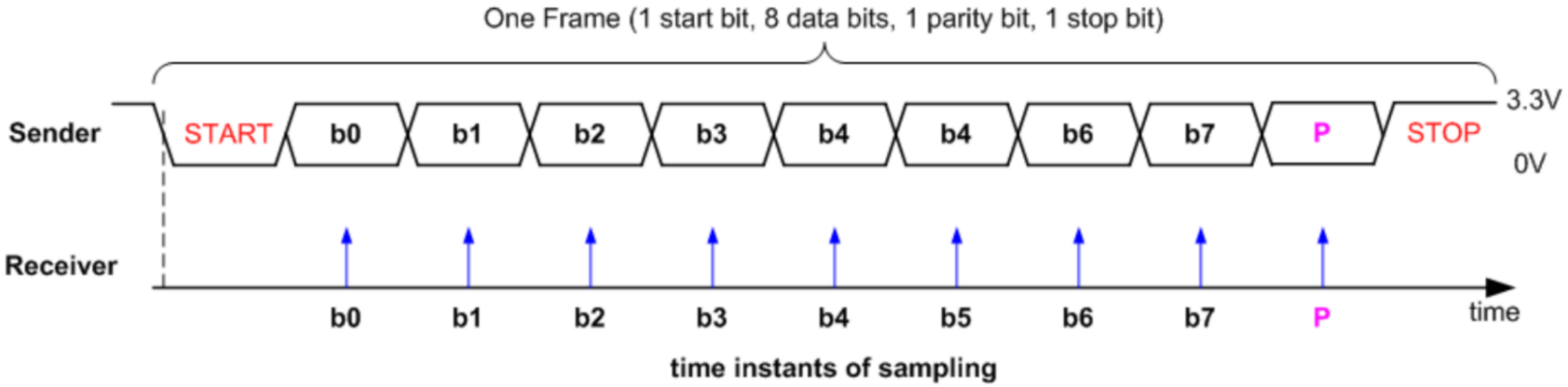
Serial communication - DB9 connector



Pin Number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring Indicator

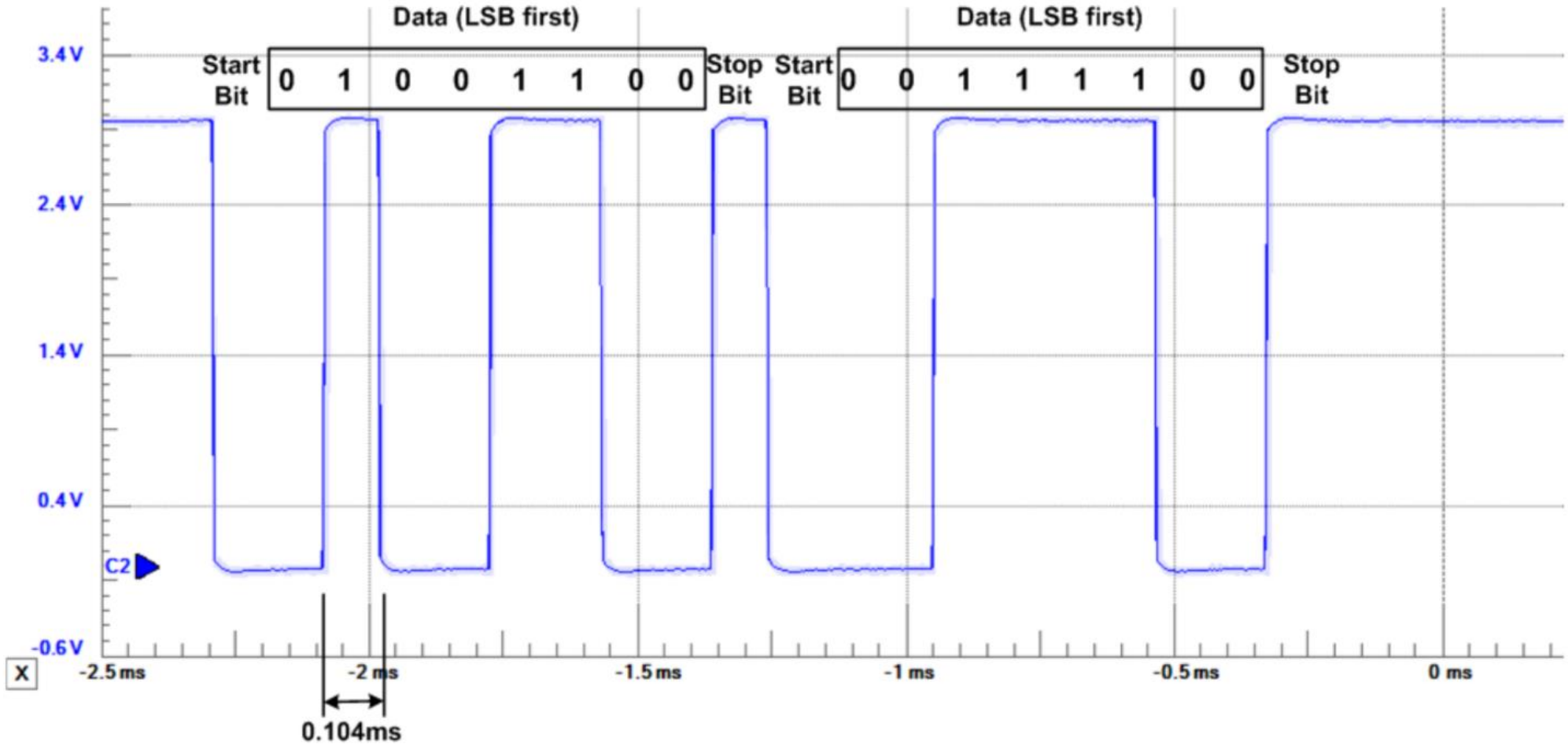
- Common pattern in cables
  - Ground (must be common), often VCC, Tx, RX
  - Plus extra wires for signaling metadata
- Signal voltage not compatible with modern microcontrollers!
  - Up to +/- 15 volts

UART data frame



- Signal is high by default
- Goes low to trigger Start
- Send each data bit (high=1, low=0), plus optionally parity bit
- Goes high to trigger Stop

UART example, transmitting 0x32 and 0x3C



1 start bit, 1 stop bit, 8 data bits, no parity, baud rate = 9600

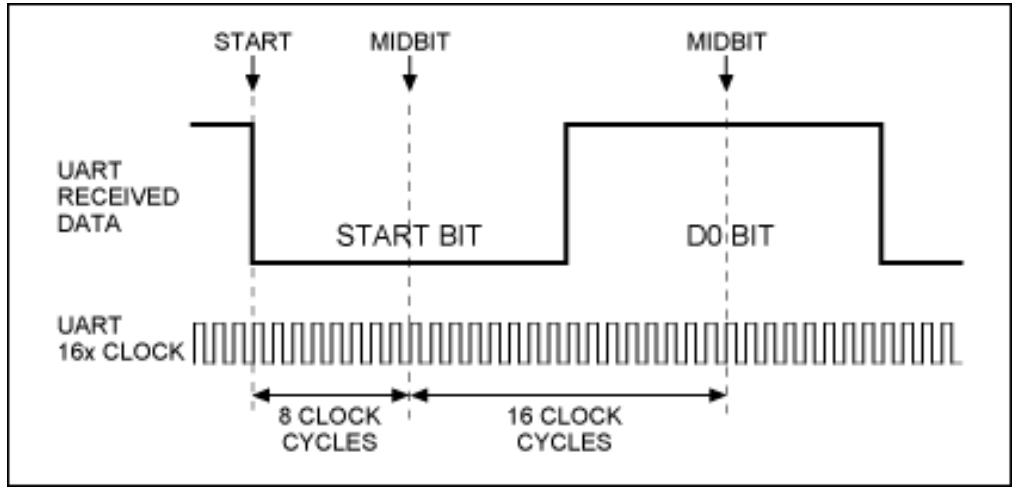
UART baud rates

- Baud rate is a measure of "symbols per second"
  - Typically 1 bit per symbol, but not always
  - UART is 1 bit per symbol, but 8 *data bits* per 10/11 symbols
- Any baud rate is possible
  - But there are a handful of normal configurations
  - 115200 and 9600 are most common
  - We use 38400 for labs!

Baud	1-8-1 (1 start, 8 data, 1 stop)	1-8-1-P (1 start, 8 data, 1 stop, 1 parity)
2400	1,920 bps	1,745 bps
4800	3,840 bps	3,490 bps
9600	7,680 bps	6,982 bps
14400	11,520 bps	10,472 bps
19200	15,360 bps	13,963 bps
38400	30,720 bps	27,927 bps
57600	46,080 bps	41,891 bps
115200	92,160 bps	83,781 bps
128000	102,400 bps	93,091 bps
256000	204,800 bps	186,182 bps

UART sampling rate

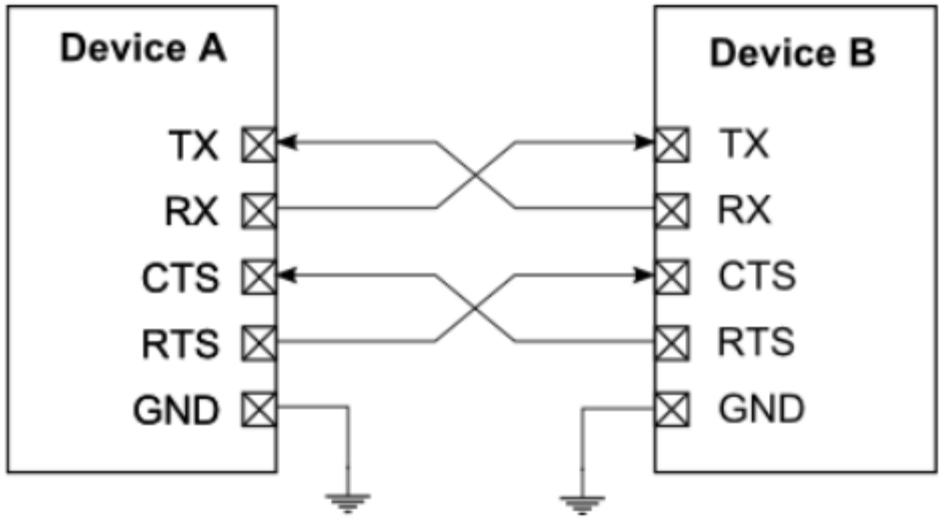
- How do we make asynchronous communication work?
  - Both sides must agree on the baud rate
  - Listen for start bit
  - Conceptually:
    - Only need to sample 9-10 more times at baud rate spacing
    - Short enough that clocks should not diverge too much
  - Realistically:
    - Sample 8 or 16 times per bit
    - Determine boundaries between bits
    - Select most common value between boundaries



UART flow control

- How do we ensure that the other device is ready for the message?
  - Add two pins for “hardware flow control”
  - Ready To Send (RTS) output, signals that you want to send data
  - Clear to Send (CTS) input, signals that other device is ready to receive
- Software flow control is possible too
  - Send special byte that means pause or resume transmissions
  - Only works with ASCII though (otherwise, byte might be valid data)

Figure 2.1. Hardware Flow Control



UART error conditions

- Parity failure
  - Bit error when receiving data
- Overrun
  - New data arrived and overwrote buffer in peripheral before it was read
- Framing
  - Did not see Stop Bit when expected (should be guaranteed “1”)
- Break condition
  - Signal is low for entire message (Zero data plus Framing Error)
  - Often used as a signal between devices

Detecting errors with parity

- If enabled, choose one configuration for all UART messages
  - Even parity: total number of "1" bits in data is even
  - Odd parity: total number of "1" bits in data is odd

- Parity bit: set to 1 or 0 based on configuration and message data
  - If message doesn't match parity configuration at receiver, there was a bit error (single error detecting)

- Example: Data = 10101011 (five "1" bits)
  - Odd parity: set parity bit to zero
  - Even parity: set parity bit to one

UART Pros and Cons

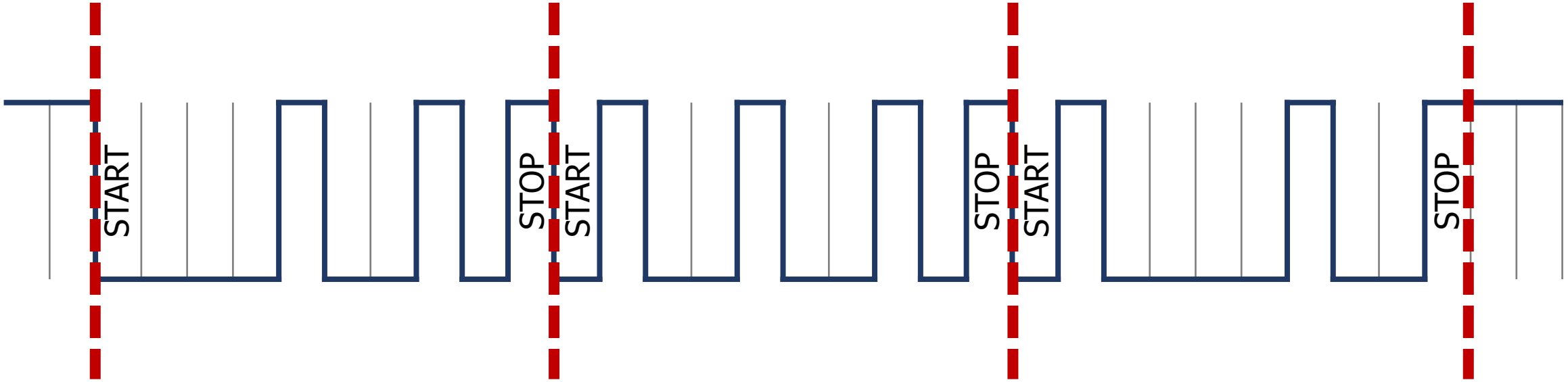
- Pros
  - Only uses two wires
  - No clock signal is necessary
  - Can do error detection with parity bit

- Cons
  - Data frame is limited to 8 bits out of 10 bits (20% signaling overhead)
  - Doesn’t support multiple device interactions (point-to-point only)
  - Relatively slow to ensure proper reception

Longer Decoding Example

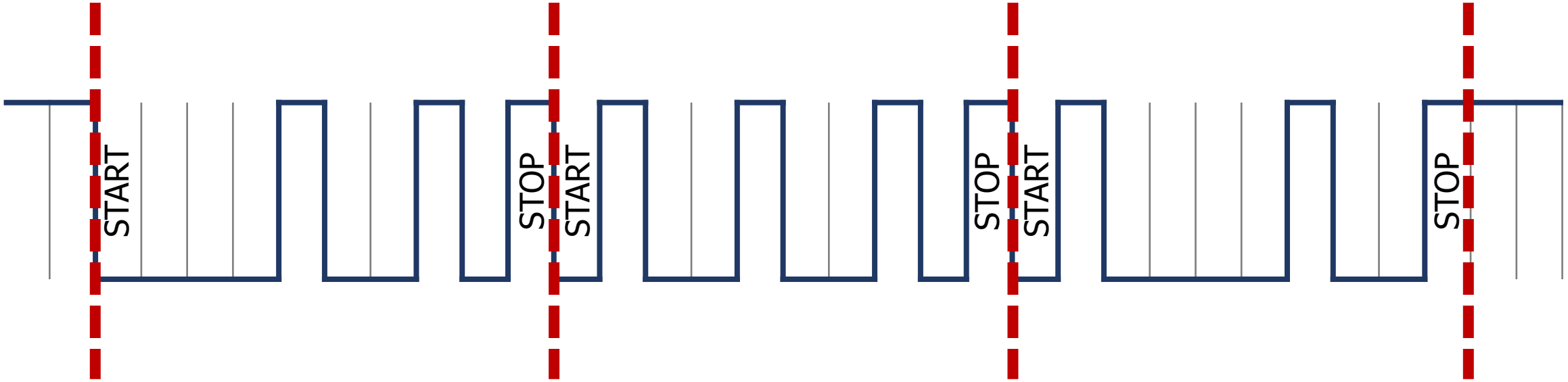
1. How many bytes are transmitted here?

Longer Decoding Example



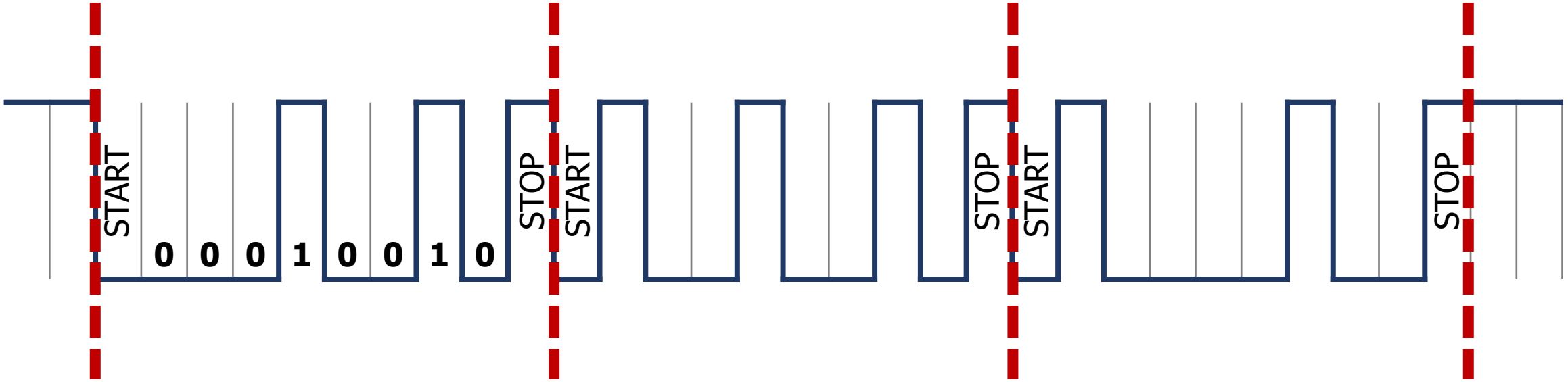
1. How many bytes are transmitted here? **3 bytes**

Break + Longer Decoding Example



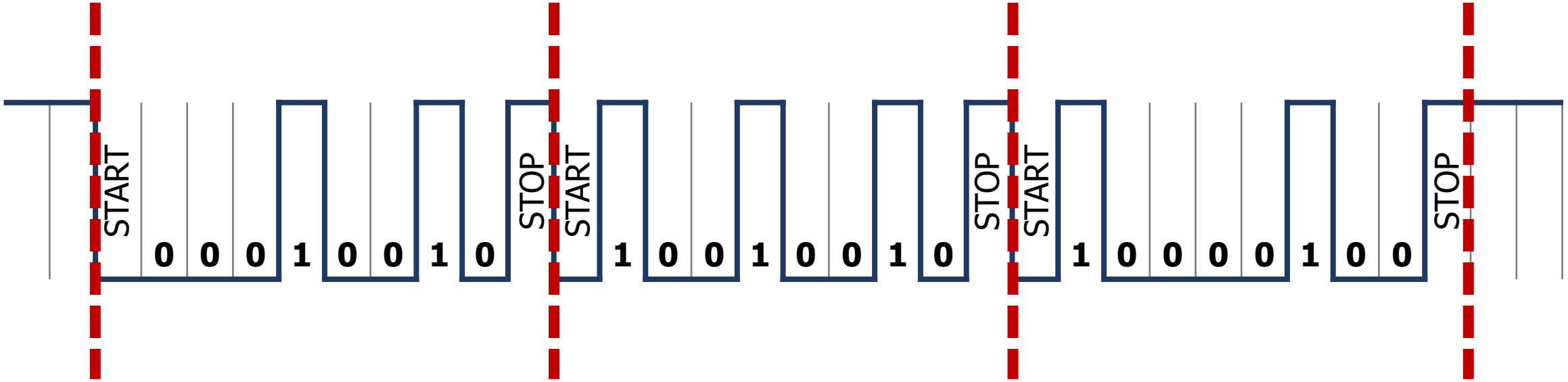
1. How many bytes are transmitted here? **3 bytes**
2. What is the data for the first byte?
   - Remember: least significant bit is first
3. What message is sent here?

Break + Longer Decoding Example



1. How many bytes are transmitted here? **3 bytes**
2. What is the data for the first byte? **0b01001000 -> 0x48**
   - Remember: least significant bit is first
3. What message is sent here?

Break + Longer Decoding Example



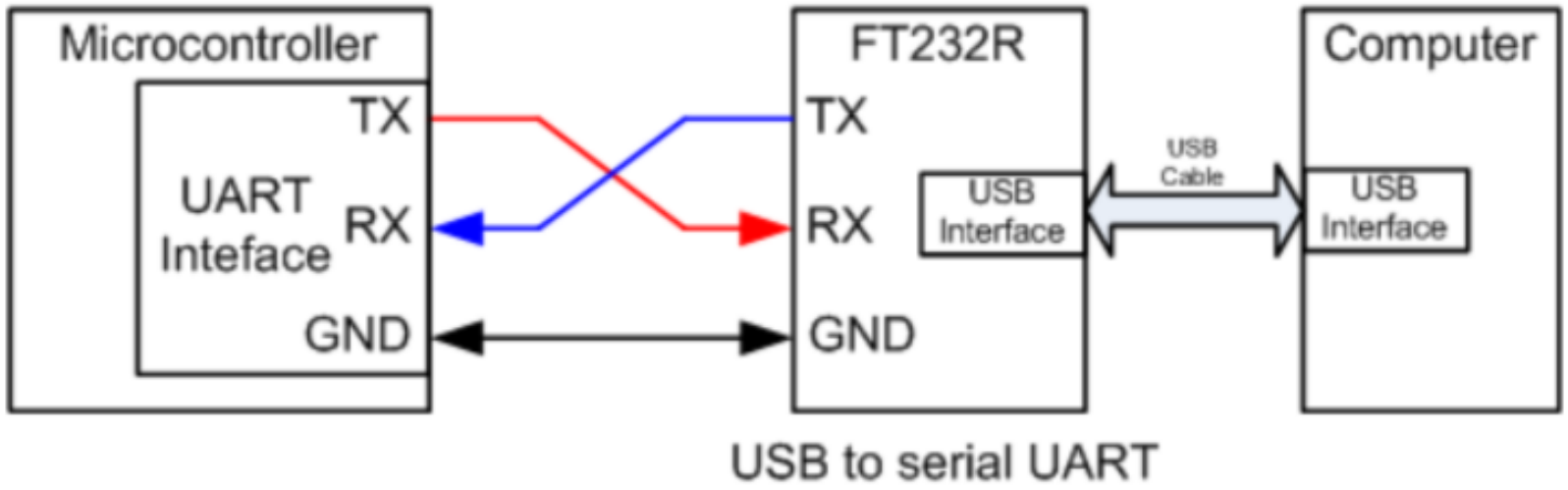
1. How many bytes are transmitted here? **3 bytes**
2. What is the data for the first byte? **0b01001000 -> 0x48**
   - Remember: least significant bit is first
3. What message is sent here? **0x48, 0x49, 0x21**
   In ASCII: HI!

Real-world signals don’t have the guide

- Assume the smallest step you ever see is the bit length
  - Check that the start/stop bits are where you expect
  - Calculate the values

- Or use a logic analyzer that can decode it for you

UART to USB bridge



- FTDI makes the most common chip to do this (FT232)
- Microbit uses a microcontroller to do this!
  - Secondary microcontroller connects to USB
  - Also connects to nRF52833 via UART and JTAG
  - ttyACM0 is a “virtual serial device” on top of USB, miniterm is a serial console

Outline

- Wired Communication
- UART
- **nRF52 UARTE**

Which UART peripheral?

- Two peripherals in the nRF52 documentation

- ~~UART peripheral~~
  - Standard UART without DMA
  - Deprecated (as in, they strongly suggest not using it)

- **UARTE peripheral**
  - Standard UART with DMA

- Their registers overlap
  - They are two different ways of using the same hardware
  - Only one at a time can be “active”

UARTE Peripheral

- Configurations
  - Pins: TXD, RXD, and optionally RTS, CTS
  - Baudrate, Parity, Flow control

- Actions
  - Transmit via DMA
    - Provide a pointer to RAM (not Flash) and a length
  - Receive via DMA
    - Provider a pointer to RAM and a length

UARTE baudrates

- Choose from standard, preconfigured buad rates
- That values are 32-bit numbers implies other baudrates are possible…

Bit number			31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID			A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
Reset 0x04000000			0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID	Acce Field	Value ID	Value	Description
A	RW BAUDRATE			Baud rate
		Baud1200	0x0004F000	1200 baud (actual rate: 1205)
		Baud2400	0x0009D000	2400 baud (actual rate: 2396)
		Baud4800	0x0013B000	4800 baud (actual rate: 4808)
		Baud9600	0x00275000	9600 baud (actual rate: 9598)
		Baud14400	0x003AF000	14400 baud (actual rate: 14401)
		Baud19200	0x004EA000	19200 baud (actual rate: 19208)
		Baud28800	0x0075C000	28800 baud (actual rate: 28777)
		Baud31250	0x00800000	31250 baud
		Baud38400	0x009D0000	38400 baud (actual rate: 38369)
		Baud56000	0x00E50000	56000 baud (actual rate: 55944)
		Baud57600	0x00EB0000	57600 baud (actual rate: 57554)
		Baud76800	0x013A9000	76800 baud (actual rate: 76923)
		Baud115200	0x01D60000	115200 baud (actual rate: 115108)
		Baud230400	0x03B00000	230400 baud (actual rate: 231884)
		Baud250000	0x04000000	250000 baud
		Baud460800	0x07400000	460800 baud (actual rate: 457143)
		Baud921600	0x0F000000	921600 baud (actual rate: 941176)
		Baud1M	0x10000000	1 megabaud

Typical UART configurations

- Baud rate 115200
- No parity
- No flow control

- Probably covers ~70% of UART communication
  - Baud rate 9600, No parity, No flow control covers another 15%

- We are the rare case of an unusual baudrate 38400
  - It was as fast as the Microbit could keep up with

UARTE driver code

- Pretty straightforward to implement driver for this
  - Definitely could have been a lab

- DMA is exactly what you want
  - Pointer to buffer of data (in RAM)
  - Length
  - Go!

- More interesting: how does `printf()` use the UART?

boards/microbit_v2/microbit_retarget.c

- `printf()` eventually calls `_write()` with formatted data

- Converts `stdio` calls into UART TX and RX
- Library just sets DMA and starts Tx

```c
int _write(int file, const char * p_char, int len)
{
    UNUSED_PARAMETER(file);

    uint8_t len8 = len & 0xFF;
    nrf_drv_uart_tx(&m_uart, (const uint8_t*)p_char, len8);
    return len8;
}

int _read(int file, char * p_char, int len)
{
    UNUSED_PARAMETER(file);

    ret_code_t result = nrf_drv_uart_rx(&m_uart, (uint8_t*)p_char, 1);
    if (result == NRF_SUCCESS) {
        return 1;
    } else {
        return -1;
    }
}
```

Outline

- Wired Communication
- UART
- nRF52 UARTE