

Lecture 14

USB

CE346 – Microprocessor System Design
Branden Ghena – Spring 2021

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

Today's Goals

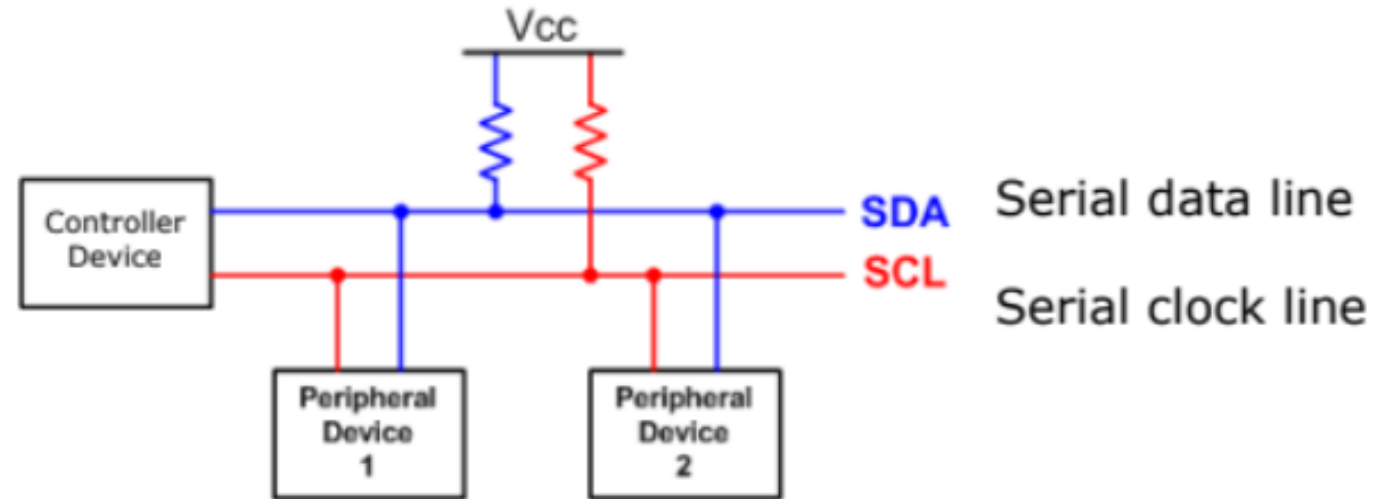
- Discuss more advanced wired communication protocols
 - With a little less detail
- Think about higher-layer concerns like data routing, interpretation, and reliability

Outline

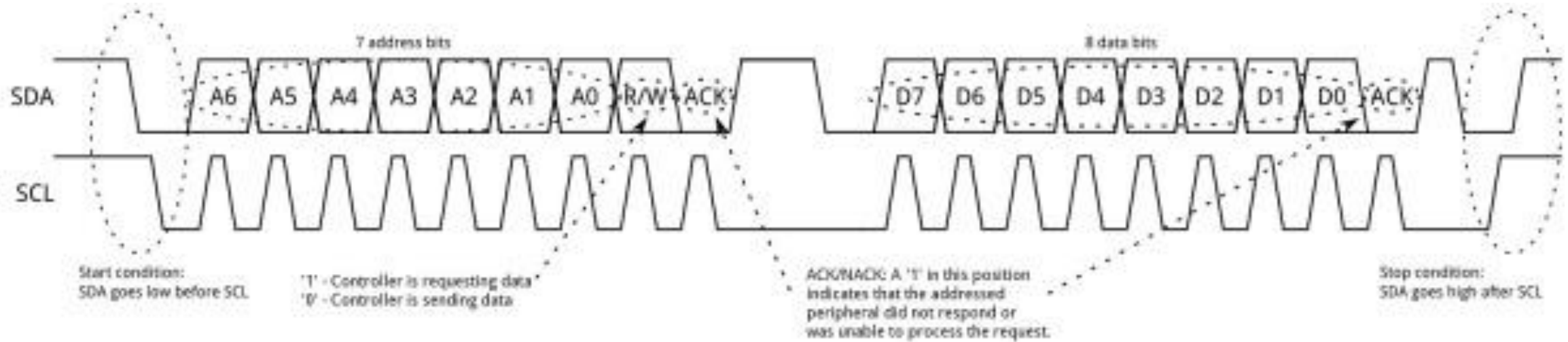
- **I2C** (wrapup)
- USB
- CAN

I2C overview

- SDA – Serial Data
- SCL – Serial Clock
 - Usually 100 kHz or 400 kHz
- Communication is a shared bus between all controller(s) and peripheral(s)
- Pull-up resistors for open-drain communication

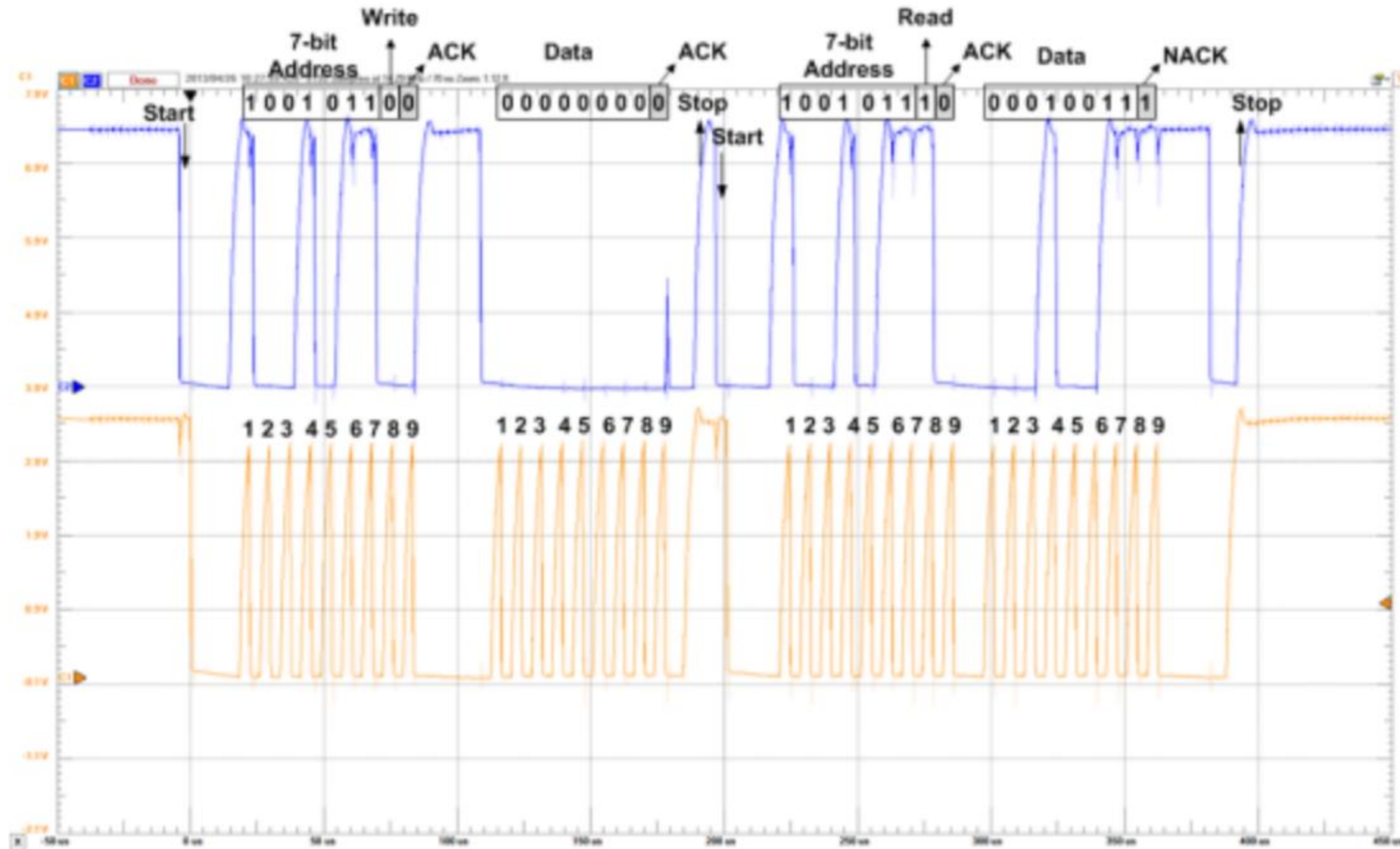


I2C transactions



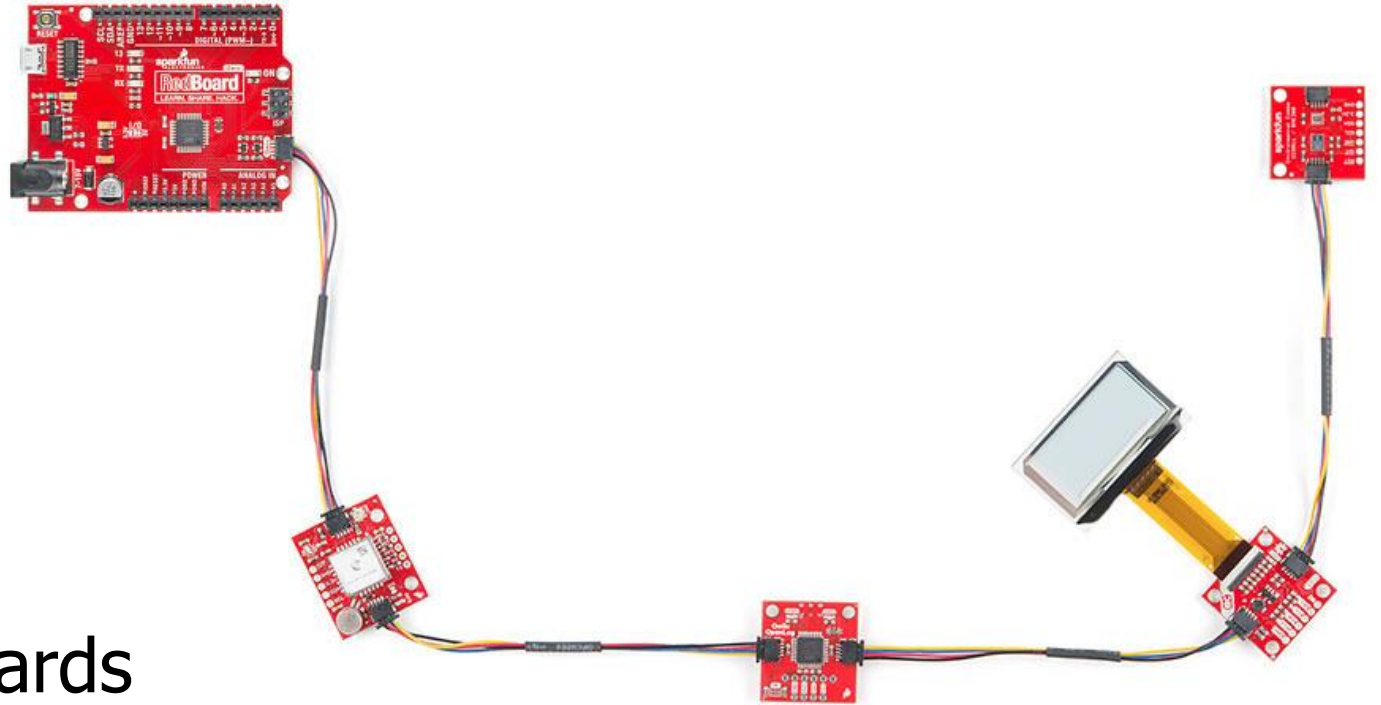
- Start and Stop Conditions: change SDA while SCL is low
- Address + R/W bit, then data byte(s)
- Common pattern:
 - Write to device to choose an internal data (known as registers)
 - Repeated-start, followed by read to get data value

Real-world I2C transactions



Sparkfun Qwiic connect system

- System for wiring multiple prototyping boards together
- Four-pin connector
 - VCC (3.3 volts)
 - Ground
 - SDA
 - SCL
- Daisy-chains through boards
 - Actually connects to chips in parallel as a bus



<https://www.sparkfun.com/qwiic>

I2C use cases

- Various sensors
 - Usually low to medium speed
 - Even relatively high speed stuff often has I2C for convenience
 - Accelerometers and microphones
 - Often with intelligent filtering built in
- Communication between microcontrollers
 - Either can act as the Controller when necessary
- Commonly exists internally within smartphones and laptops too
 - Light sensors, Temperature sensors, etc.

System Management Bus (SMBus)

- Related communication specification
 - A little more strict in places, but generally interoperable
- Adds ability to broadcast or unicast messages
 - Generic addresses for Controller and various peripherals (Battery)
- Adds an open-drain shared interrupt signal
 - High-impedance or pull low, just like SDA and SCL
 - Allows any device to alert a controller
 - Controller has to probe bus to determine which device wants attention

nRF TWI peripherals

- TWIM – for I2C Controller
 - This is the common case
 - Provide buffer to write from and/or read into for DMA
 - Configure registers, perform interaction, repeat
- TWIS – for I2C peripheral
 - Less common for microcontroller to be a peripheral
 - Supports dual-mode where you are switching between states
 - Provide up to two addresses that will be matched against
 - When interrupted with write data
 - Change buffer connected to read DMA (as appropriate)
 - Possibly stretching clock to make it all work

I2C Pros and Cons

- Pros
 - Wiring is simple
 - Only uses two pins
 - Very widely supported
- Cons
 - Relatively slow communication rate
 - Speed versus power use tradeoff (due to pull-down resistor)
 - Open collector makes debugging difficult

Outline

- I2C

- **USB**

- CAN

USB references

- USB in a NutShell
 - <https://www.beyondlogic.org/usbnutshell>
- Other stuff I found useful
 - <https://www.usbmadesimple.co.uk/>
 - http://kofa.mmt0.arizona.edu/stm32all/blue_pill/usb/an57294.pdf
 - <https://en.wikipedia.org/wiki/USB>

Universal Serial Bus (USB)

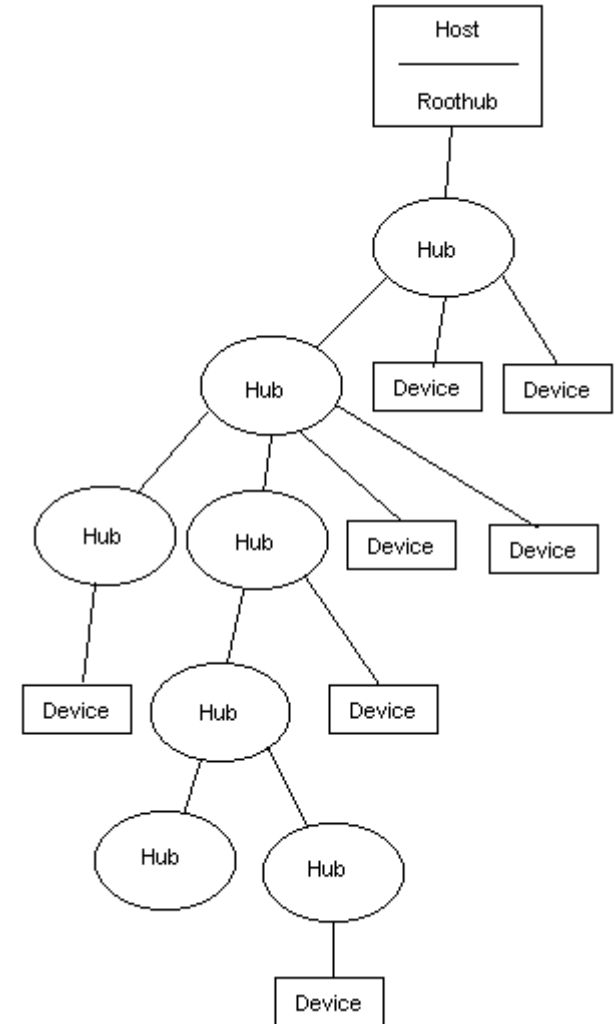
- Pervasive wired communication protocol
 - Universal accurately applies!
 - Targets predominantly external devices over a plug/cable
- Good combination of simple and capable
 - Base version for simple devices does not require too much in terms of pins or resources
 - More complex versions can transfer a significant amount of data
 - These grew organically over time though
- Great support for interoperability
 - Generic device profiles that allowed for plug-and-play
 - Supported by OS initiatives to include driver software

USB is a layered protocol

- USB protocol describes how to:
 - Electrically send bits
 - Send frames of multiple bytes
 - Communicate data between two devices
 - Communicate specific application data (through device classes)
- Much larger compared to others
 - SPI: only how to electrically send bits
 - UART and I2C: how to send frames of bytes

Roles and topology

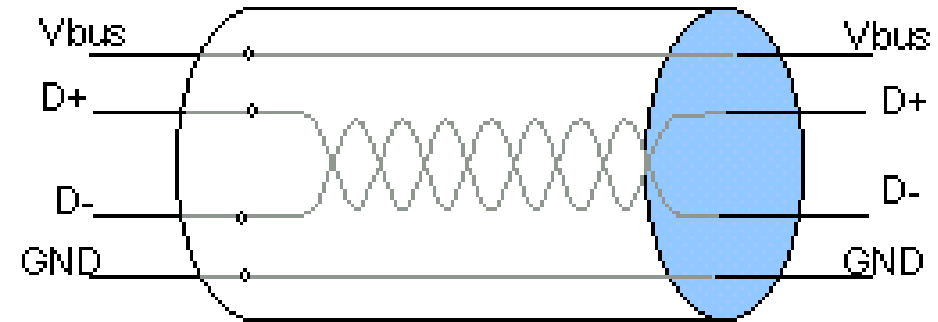
- Hosts and Devices
 - USB On-The-Go allows host negotiation
 - Added later. Support devices like smartphones
- Host is in charge of communication (“Upstream”)
- Devices provide various capabilities Host can interact with (“Downstream”)
- Tiered star topology
 - Host connects to hubs, which connect to devices
 - Up to 127 devices per hub. Up to 5 layers of hubs



USB signals

- Four signals

- Vbus (5 volts, can power devices)
- D+
- D-
- Ground

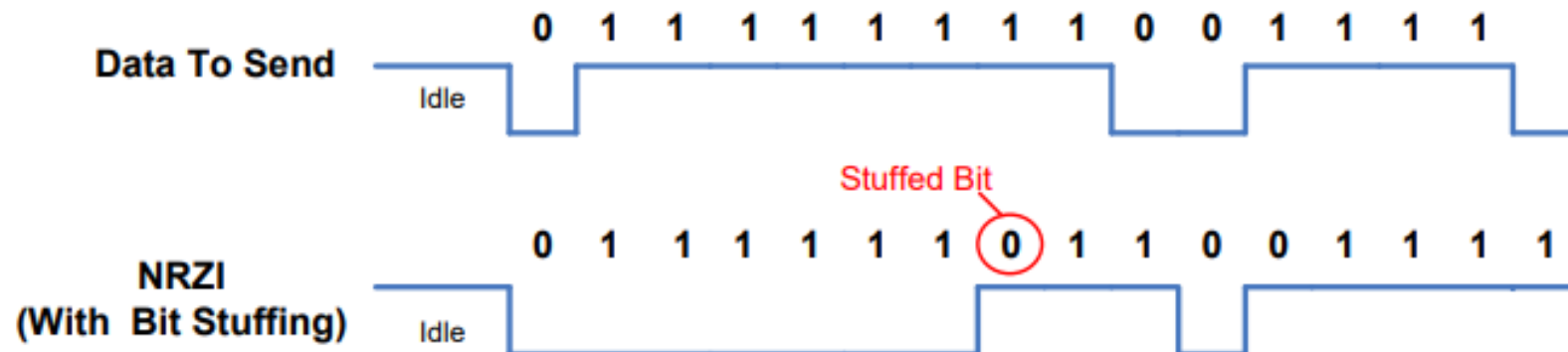


- D+/D- are a *differential pair*

- Signals are inverses of each other
 - Usually, occasionally act separately to signal special conditions
 - Increases voltage difference between states ($5 - -5 = 10$ volts)
- Wires are twisted to avoid interference

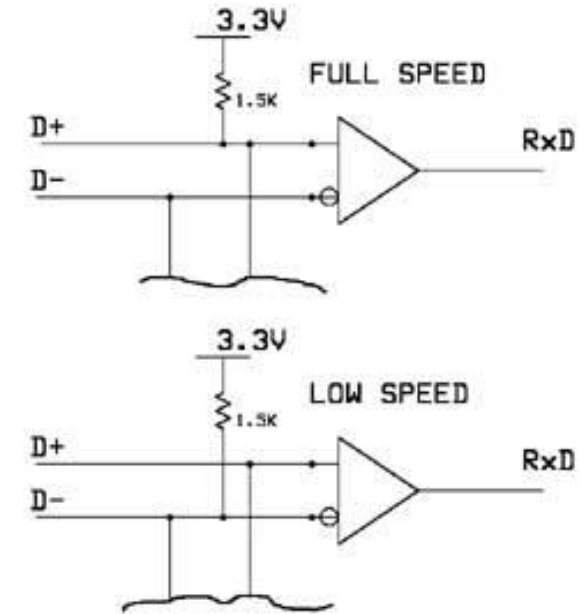
Synchronizing data

- No clock signal!! How is USB so fast?
 - Partially EE magics: better receivers, matched wire impedance
 - Partially easier to distinguish signal states
 - Also guaranteed transitions, which allow resynchronization
- Transitions are used to denote data (non-return-to-zero inverted)
 - With guaranteed transition in within every 8 bits (bit stuffing)
 - Allows clocks on the two devices to synchronize



USB speeds

- USB 1.0
 - Low Speed: 1.5 Mbps
 - Not clear if this is used anymore
 - Full Speed: 12 Mbps
 - Microcontrollers tend to support Full Speed
 - We're focusing on details from it
- USB 2.0
 - High Speed: 480 Mbps
- USB 3.0+
 - Super Speed: 5-20 Gbps
 - Adds multiple parallel data connections



- Pull-up resistors allow for detection of a plugged device
- Also identify speed

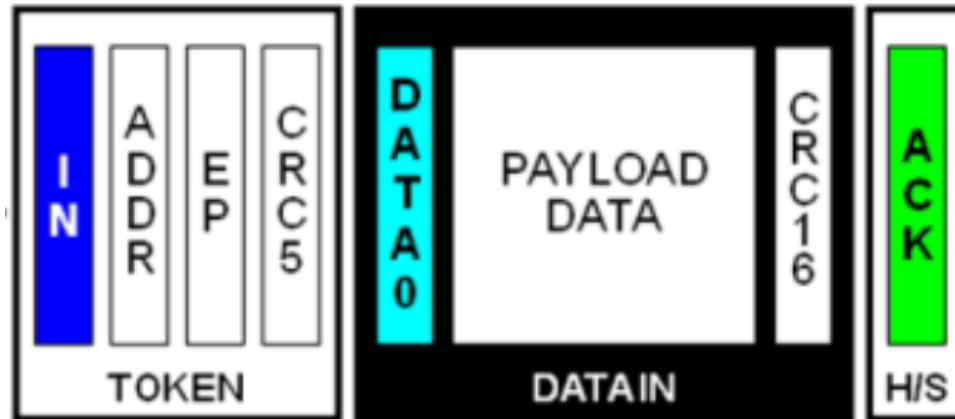
USB data frames

- General packet format
 - Sync field, allows transmitter and receiver clocks to synchronize
 - Packet ID, determines what type of packet is being sent
 - Token, setup Device or read/write data from/to Device
 - Data, application data
 - Handshake, acknowledgement (or explicit NAK)
 - Address+Endpoint, for Token packets to identify Device
 - Data, up to 1023 bytes (full speed, often capped at 64 for microcontrollers)
 - CRC, (Cyclical Redundancy Check) to detect bit errors
 - End-of-packet

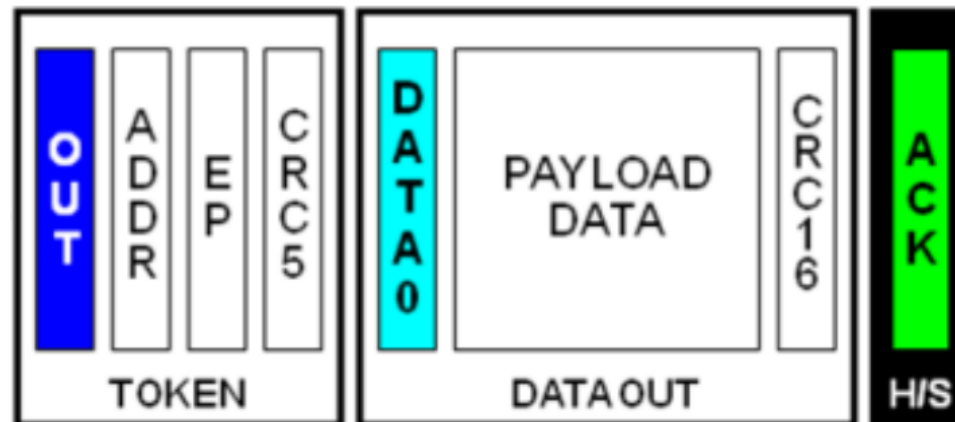
USB transactions

- All transmissions are initiated by the Host, which can periodically request reading from a Device if desired

- Reading data from Device

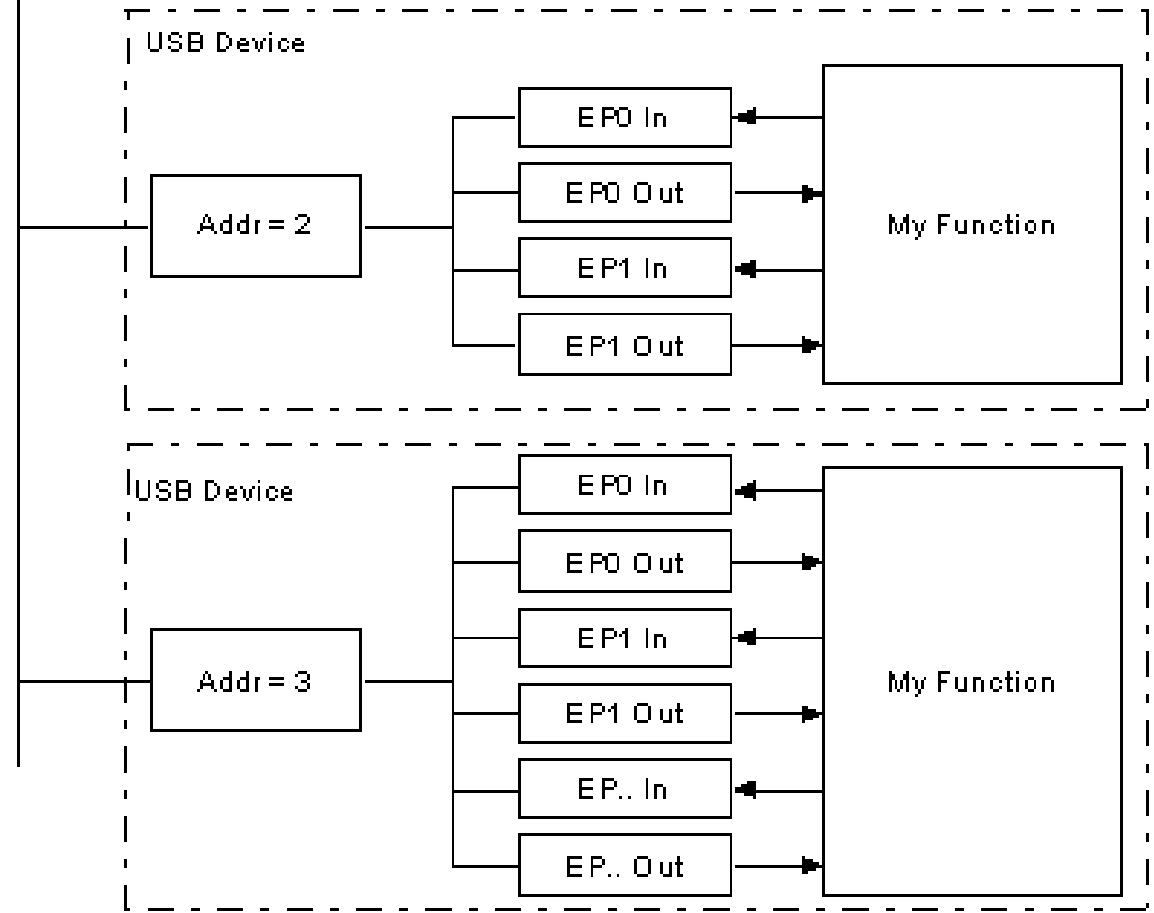
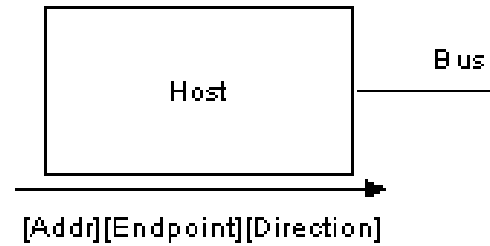


- Writing data to Device



Interacting with USB devices

- Each Device is given a separate address on the bus
- Each Device also has a number of Endpoints
 - Logical communication channels
 - Direct data and guide communication patterns

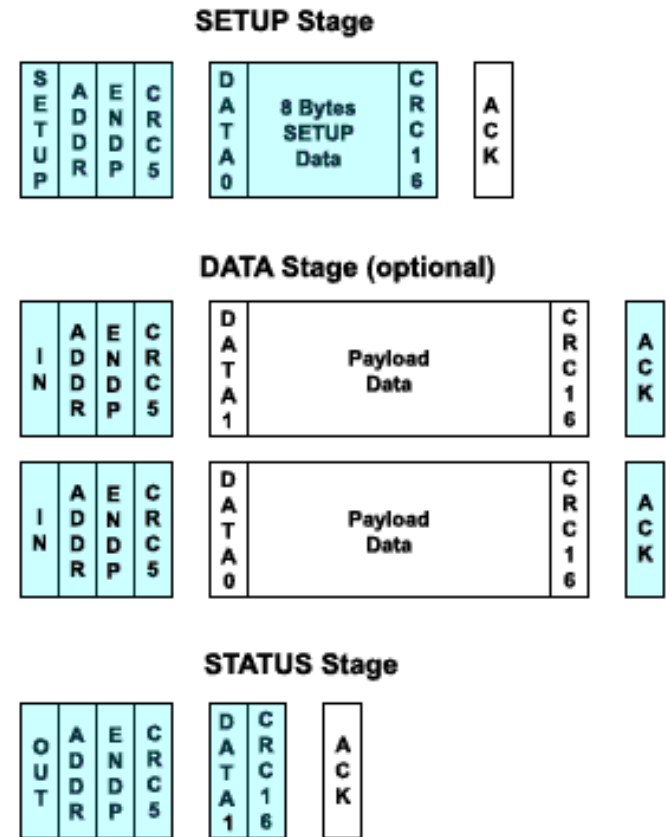


USB endpoint types

- Interrupt transfers
 - Guaranteed latency, small amounts of data
 - Important sensor data (mice and keyboards)
 - Polled frequently by Host
- Bulk transfers
 - Sporadic large transfers, reliable communication
 - General reading/writing of data (flash drives and USB serial)
 - Polled by Host whenever there is available bandwidth
- Isochronous transfers
 - Guaranteed data rate, unreliable communication
 - Continuous data streaming (audio and webcams)
 - Polled frequently by host

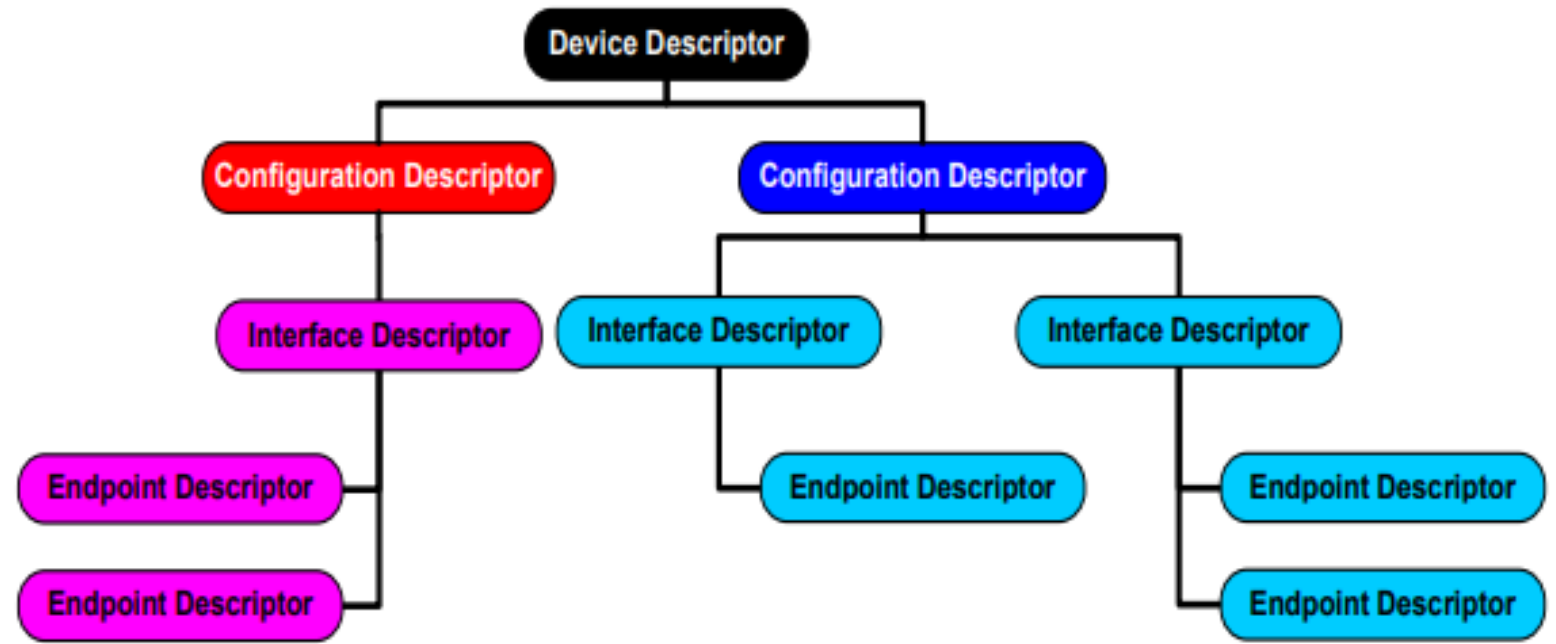
USB control endpoint

- Every USB Device has a special Control endpoint as well
- Used for setting up the USB Device driver on the Host
- Initializing a Device
 - Host sends SETUP transaction requesting device descriptor
 - Host performs IN transaction to read device descriptor
 - Host performs OUT transaction to write device status



USB device descriptors

- Packed version of tree structure describing the device
 - Interfaces it provides
 - Endpoints associated with each interface



Example Microbit


- Interface: Communications, Abstract (modem), CDC
 - Endpoint: 3, IN, Interrupt
- Interface: CDC Data, CDC DATA interface
 - Endpoint: 1, IN, Bulk
 - Endpoint: 2, OUT, Bulk
- Interface: Vendor Specific Class, Subclass, Protocol
 - Endpoint: 5, IN, Bulk
 - Endpoint: 4, OUT, Bulk
- Interface: Mass Storage, SCSI, MSD interface
 - Endpoint: 7, IN, Bulk
 - Endpoint: 6, OUT, Bulk



Virtual serial device



SEGGER JTAG interface



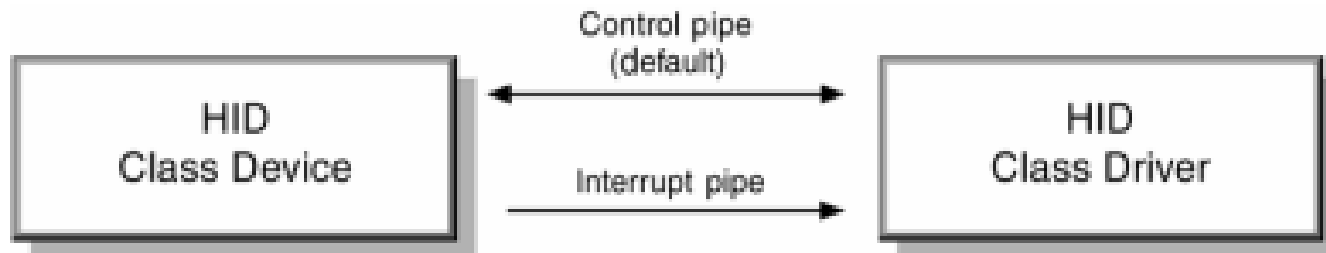
USB external filesystem

Minimal virtual serial USB Device

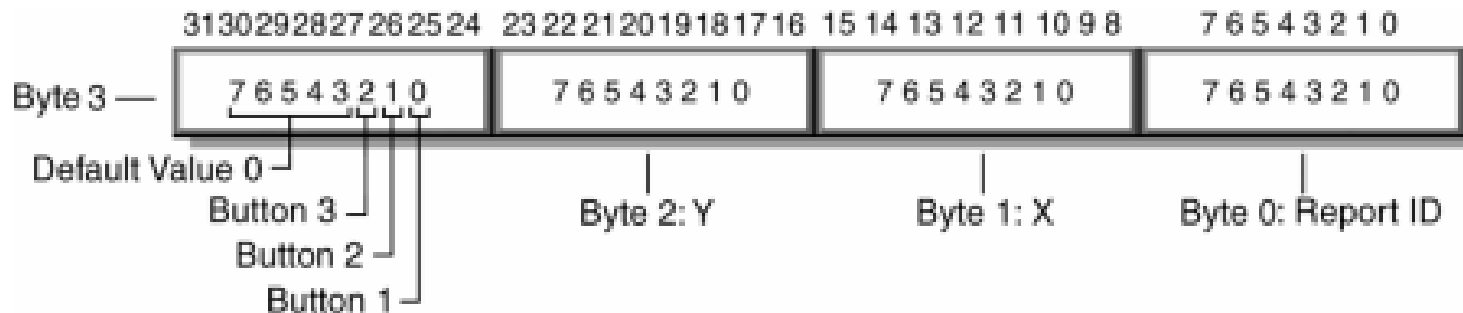
- Virtual Serial Device
 - Endpoint 0: Control, IN/OUT
 - Respond to IN requests by setting up OUT with a buffer of descriptor data of the correct size
 - Endpoint 1: Interrupt, IN
 - Needed for serial modem controls, just ignore it
 - Endpoint 2: Bulk, OUT
 - Connect to buffer from `_write()` (just takes raw characters)
 - Endpoint 3: Bulk, IN
 - Connect buffer to `_read()` (just provides raw characters)

HID USB Device (Human Interface Device)

- Used for human interaction devices, like keyboard/mouse



- “Report” structure is provided over Interrupt IN endpoint
 - Or on demand via Control IN endpoint



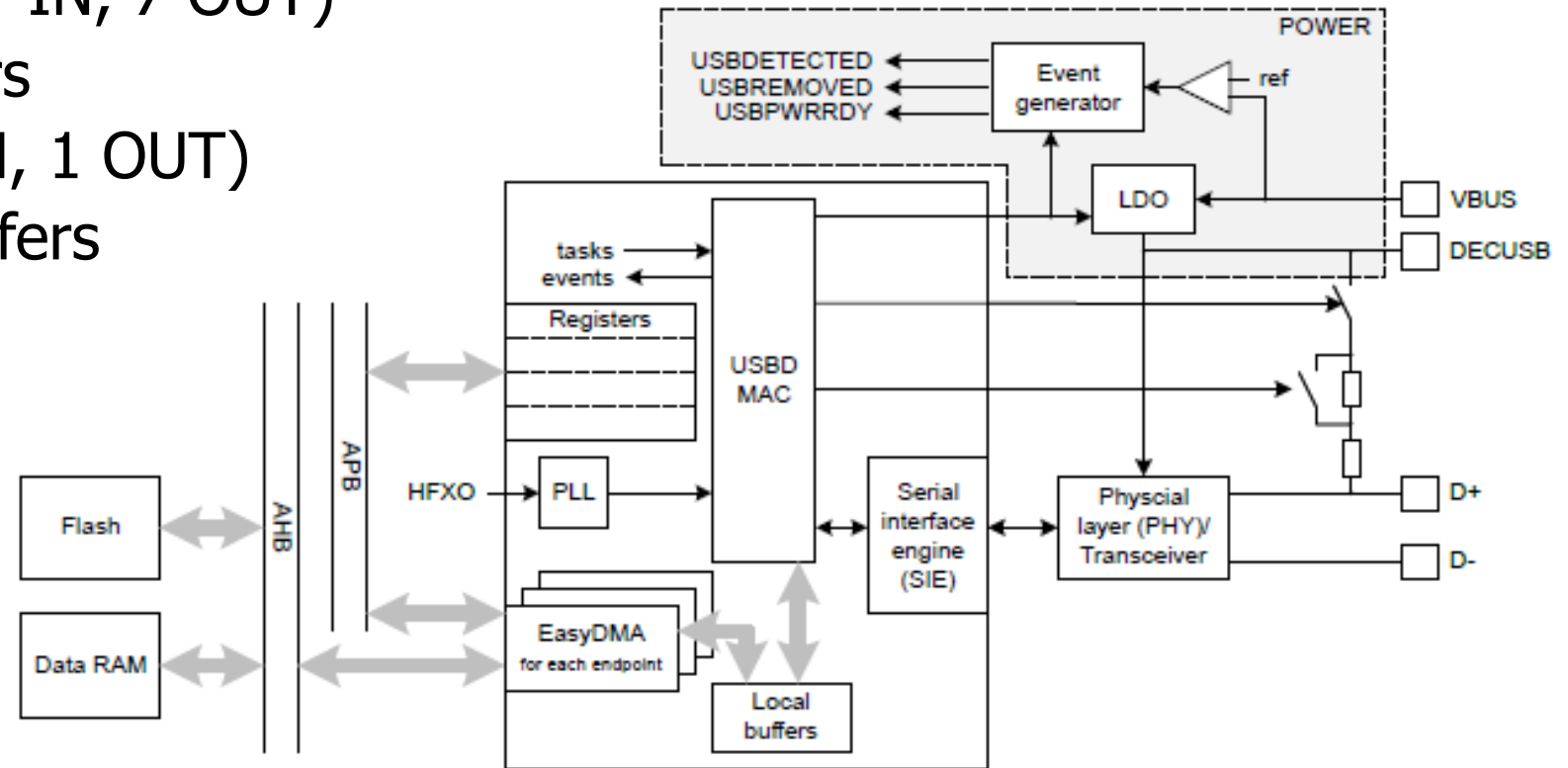
Example mouse with x,y and three buttons

USB summary

- Specification for fast data communication
- Specification for interacting with abstract device types
 - Connects correct driver to interpret and send data
- Pros
 - Very fast
 - Very interoperable
- Cons
 - Hardware and software are way more complex than simple protocols like UART, SPI, and I2C
 - Not very energy efficient

nRF52 USBD

- Implements USB Device (not Host)
 - Control endpoint
 - 14 bulk/interrupt (7 IN, 7 OUT)
 - 64-byte transfers
 - 2 isochronous (1 IN, 1 OUT)
 - 1023-byte transfers
- Full-speed USB
 - With 5 volt signals



Outline

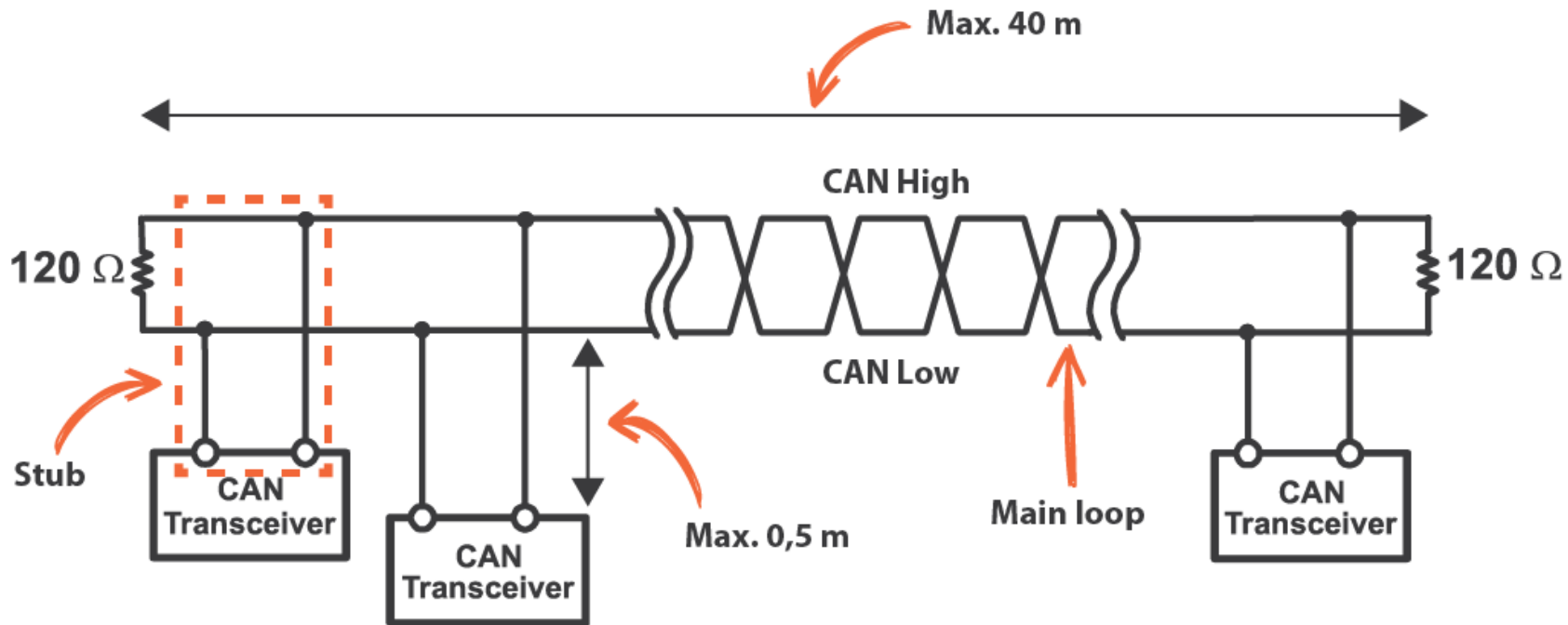
- I2C
- USB
- **CAN**

Controller Area Network (CAN bus)

- Designed for highly reliable interactions within a vehicle
- Multi-master with arbitration
 - Similar to I2C
- Mechanism for sending messages with “identifiers”
 - Identifies that data in the message, not the device
 - Lower value identifiers have high priority
 - All messages are received by all CAN nodes
 - Which can decide at higher levels which identifiers they care about

CAN physical connections

- Two differential, wired-AND signal lines
 - Transitions are used to transmit bits (non-return-to-zero) with bit-stuffing
 - Combines aspects of USB and I2C
 - 125 kHz – 5 Mbps speeds



CAN packet format



- 11-bit identifier
 - Check bits as they are sent to see if you win arbitration
- Up to 8 bytes (64 bits) of data
- CRC for checking
- Acknowledgement
 - Like I2C, let the line float and see if another device responds
 - If not, explicitly retransmit!

CAN message types

- Data frame
 - Transmission of data for a certain identifier
- Remote frame
 - Requests data transmission of a certain identifier
- Error frame
 - Transmitted when an error is detected with the previous message
- Overload frame
 - Transmitted by a node that is too busy to respond right now

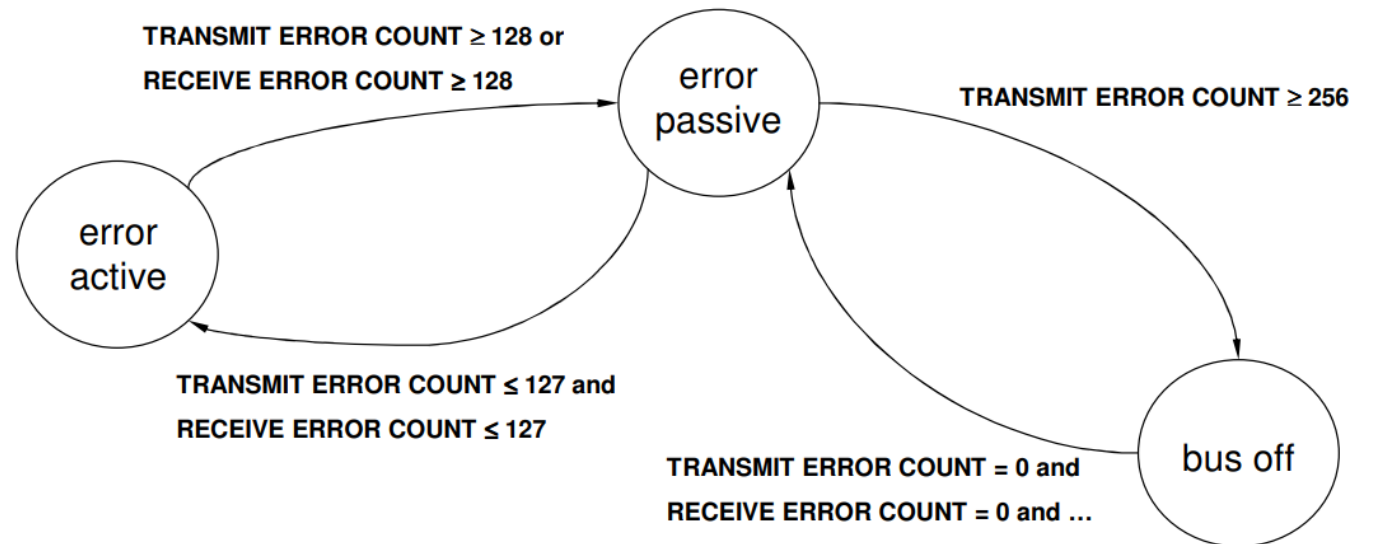
CAN reliability design – detecting errors

- Check for errors everywhere and appropriately handle
 - Bit error
 - If the value found on the bus differs from the one sent
 - Stuff error
 - If 6 consecutive bits of the same type are found
 - CRC error
 - If CRC does not match
 - Form error
 - Format field has unexpected values
 - Acknowledgement error
 - No ACK received
- Devices detecting an error broadcast a message signifying it!
 - Multiple devices sending the same message works without arbitration loss
 - Previous message is then retransmitted

CAN reliability design – handling errors

- Each node accepts the possibility that maybe it is the faulty one
- Track errors and successes and change device state
 - Passive: limited error signaling and transmissions
 - Bus off: does not transmit in any way

- Idea is that the CAN controller hardware can be faulty but still detect it in some cases



CAN summary

- Designed for reliable vehicular communication
- Multi-master bus with serial communication

- Pros
 - Highly reliable
 - Extensible to many devices
- Cons
 - Special-purpose design. Whole system has to agree on identifiers
 - Relatively slower throughput

Outline

- I2C
- USB
- CAN