Lecture 14
USB & CAN

CE346 – Microprocessor System Design

Branden Ghena – Fall 2022

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

Administrivia

- Hardware handout again at the end of class today
  - I’ve got a *bunch* of hardware!

- We’ll hand out Microbits next week Tuesday at the end of class
  - Also the date of our last quiz!

- No lab this Friday! Everybody enjoy your extra time!
  - And use it to work on projects!

Administrivia

- Lecture schedule for the rest of the quarter
  - Thursday (11/10) – Wireless Communication

  - Tuesday (11/15) – Nonvolatile Memory & Energy Management
    - Also the final quiz

  - Thursday (11/17) – Microprocessors + Wrapup

  - Tuesday (11/22) – Embedded Systems Research
    - Tuesday before Thanksgiving

  - Tuesday (11/28) & Thursday (12/01) – Project Office Hours

Today's Goals

- Discuss more advanced wired communication protocols
  - With a little less detail
  - Just give a taste of what they are like

- Think about higher-layer concerns like data routing, interpretation, and reliability

Outline

- **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.mmto.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

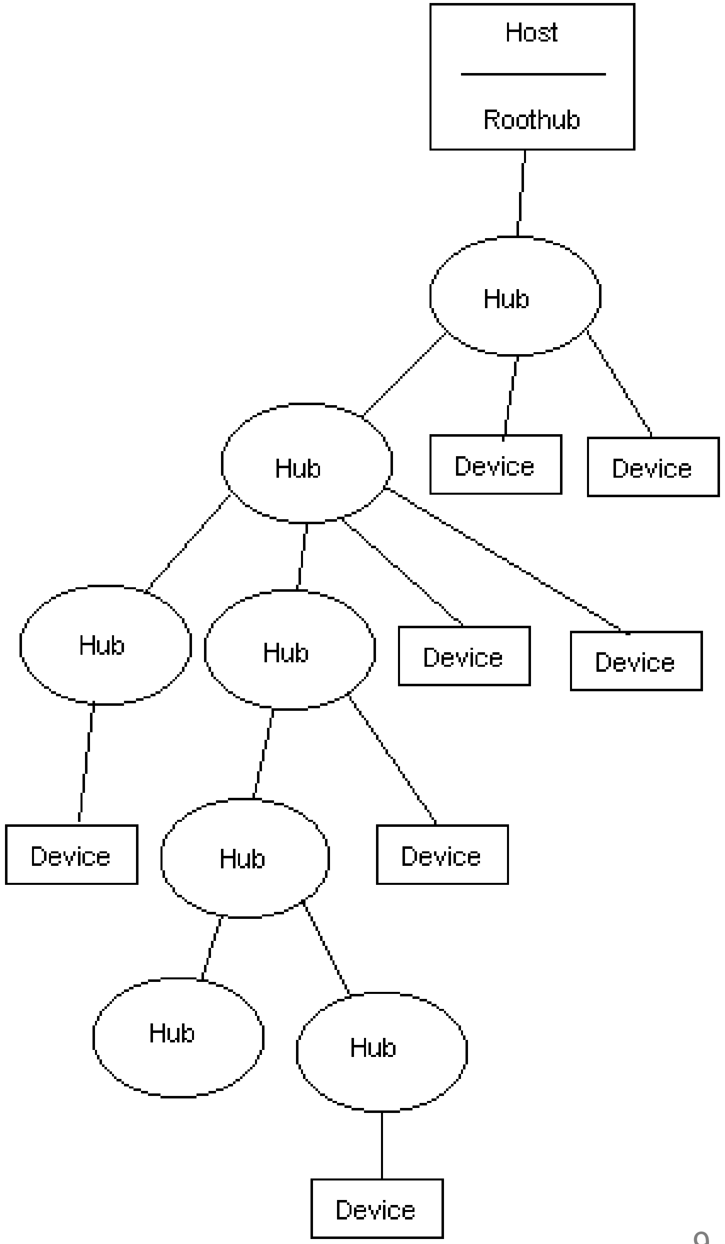
USB is a layered protocol

- 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 more complicated, compared to others
  - SPI: only how to electrically send bits
  - UART and I2C: how to send frames of bytes

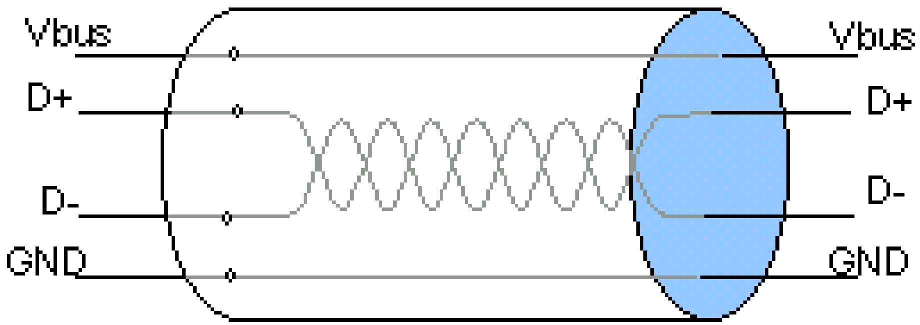
Roles and topology

- 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

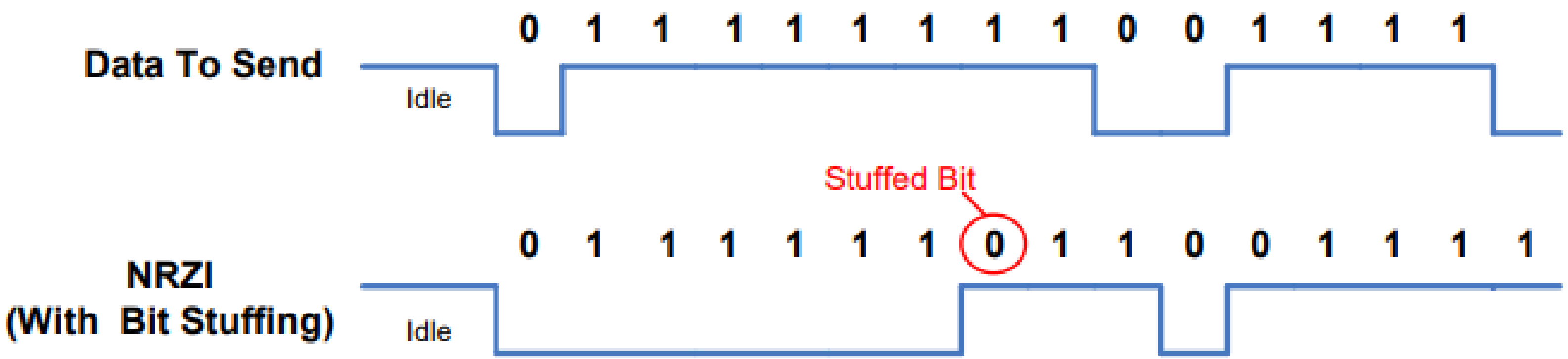


- D+/D- are a *differential pair*
  - Signals are inverses of each other
    - Usually, occasionally act separately to signal special conditions
    - Increases voltage difference between states (5 - -5 = 10 volts)
  - Wires are twisted to avoid interference

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

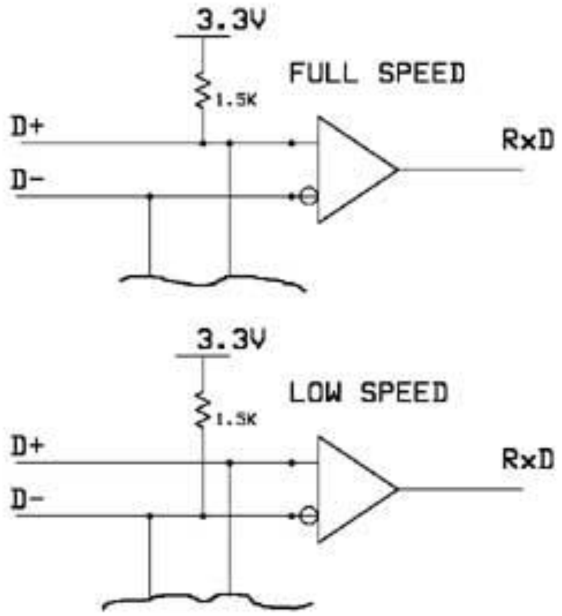


USB speeds

- 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



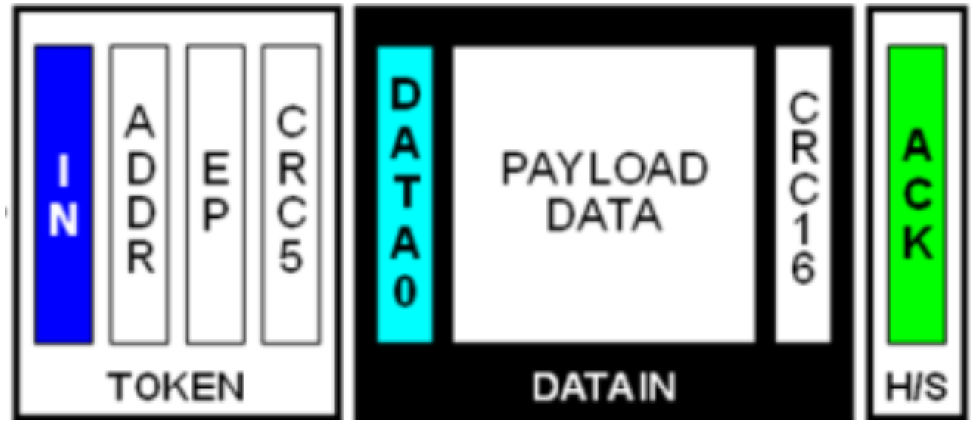
- Pull-up resistors allow for detection of a plugged device

- Also identify speed

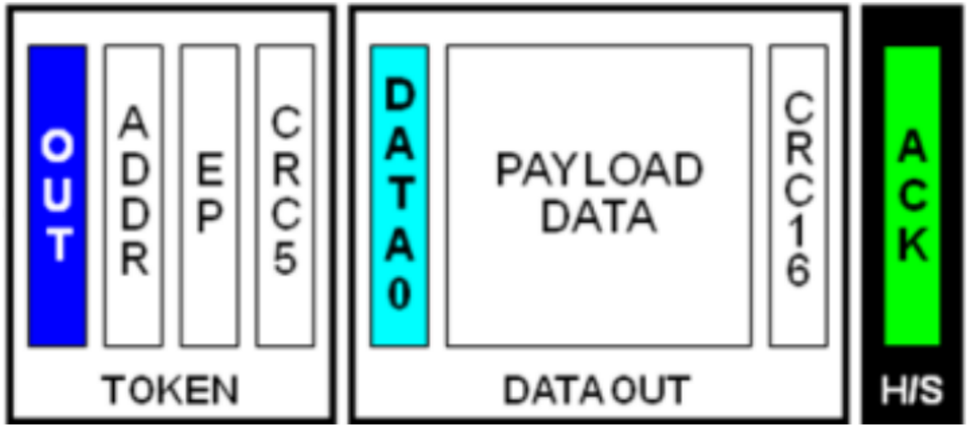
USB interactions

- General transaction format
  1. Host sends a Token packet: identifies transfer direction and device
  2. Host or Device send data depending on direction
  3. Other side acknowledges receipt of data
- Like a maxed-out version of the I2C transaction pattern
  - Host *always* initiates communication

Reading data from Device



Writing data to Device



USB token packets

- Packet fields
  - Sync field, allows transmitter and receiver clocks to synchronize
  - Packet ID, determines what type of packet is being sent
    - Token type: Setup device, Read from device, or Write to device
  - Address+Endpoint to identify Device
  - CRC, (Cyclical Redundancy Check) to detect bit errors
    - 5-bit CRC

USB data packets

- Packet fields
  - Sync field, allows transmitter and receiver clocks to synchronize

  - Packet ID, determines what type of packet is being sent
    - Data: application data

  - Data, up to 1023 bytes (full speed, often capped at 64 for microcontrollers)

  - CRC, (Cyclical Redundancy Check) to detect bit errors
    - 32-bit CRC

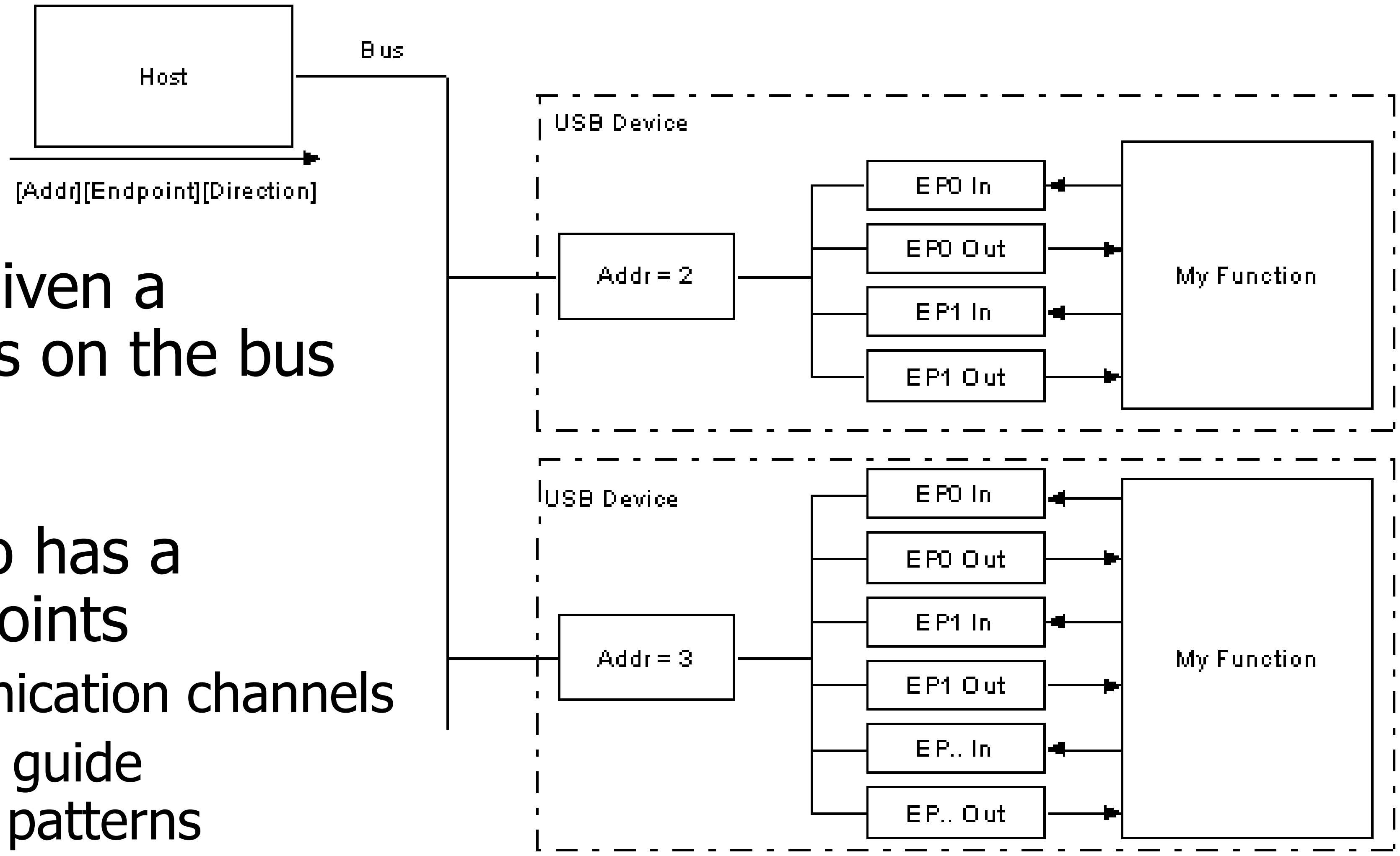
Cyclic Redundancy Check (CRC)

- Determines if the data received matches the data sent
  - CRC value is calculated on original data and appended to message
  - CRC value is recalculated on the received data
  - Value appended to message and value recalculated MUST match

- Essentially some kind of hash operation
  - Turns many bits into some smaller number of bits that are unique-ish

- CRC algorithms are:
  - Particularly good at single bit errors AND contiguous bit errors
  - Relatively simple to calculate
  - Very widely used in communication

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

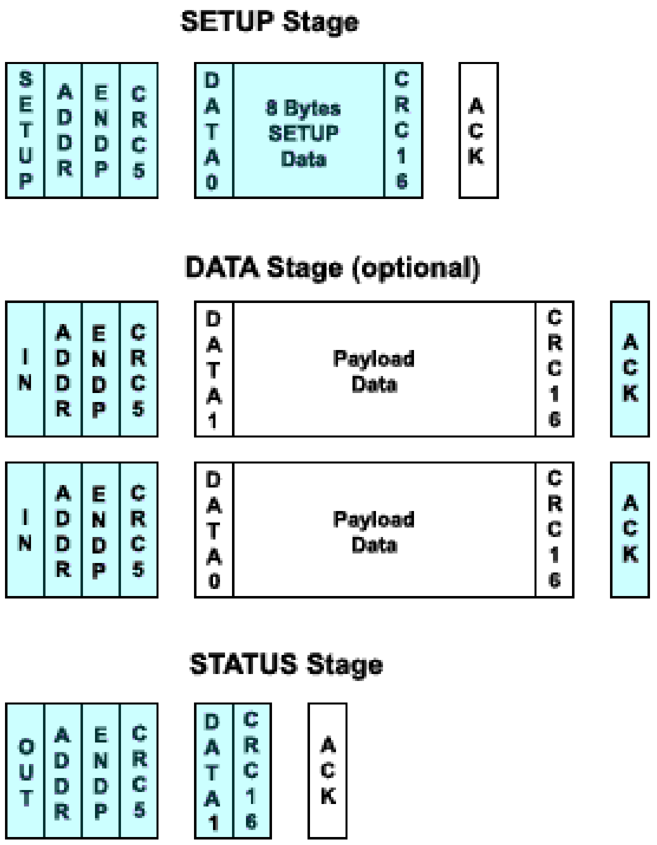
- Isochronous transfers
  - Guaranteed data rate, unreliable communication
  - Continuous data streaming (audio and webcams)
  - Polled frequently by host

USB control endpoint

- Every USB Device has a special Control endpoint as well

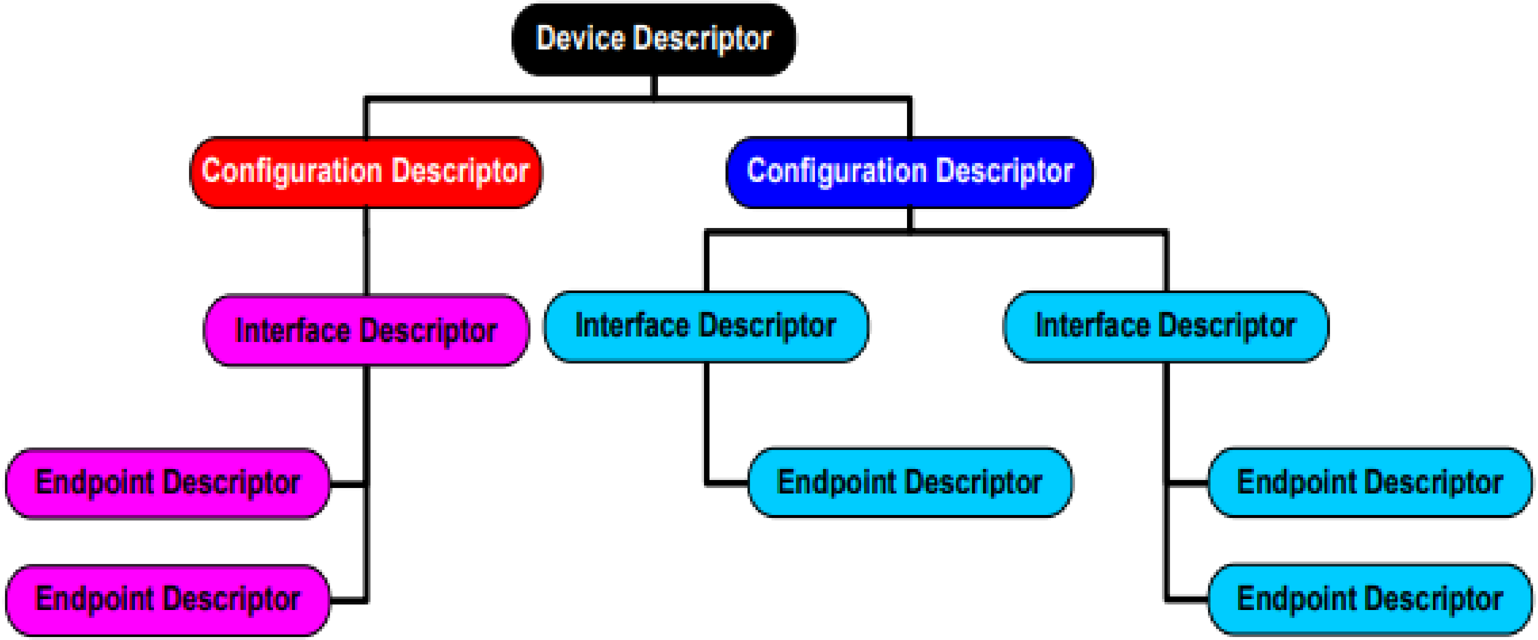
- Used for setting up the USB Device driver on the Host

- 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

} Virtual serial device

- Interface: Vendor Specific Class, Subclass, Protocol
  - Endpoint: 5, IN, Bulk
  - Endpoint: 4, OUT, Bulk

} SEGGER JTAG interface

- Interface: Mass Storage, SCSI, MSD interface
  - Endpoint: 7, IN, Bulk
  - Endpoint: 6, OUT, Bulk

} USB external filesystem

lsusb output

- `lsusb`
  - List USB devices

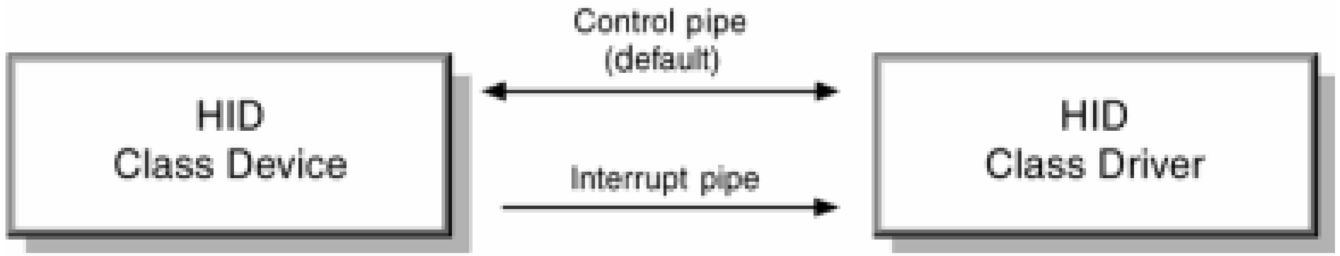
- Combine with `–s` flag to select a single device
- Combine with `–v` flag for verbose mode with more information

Minimal virtual serial USB Device

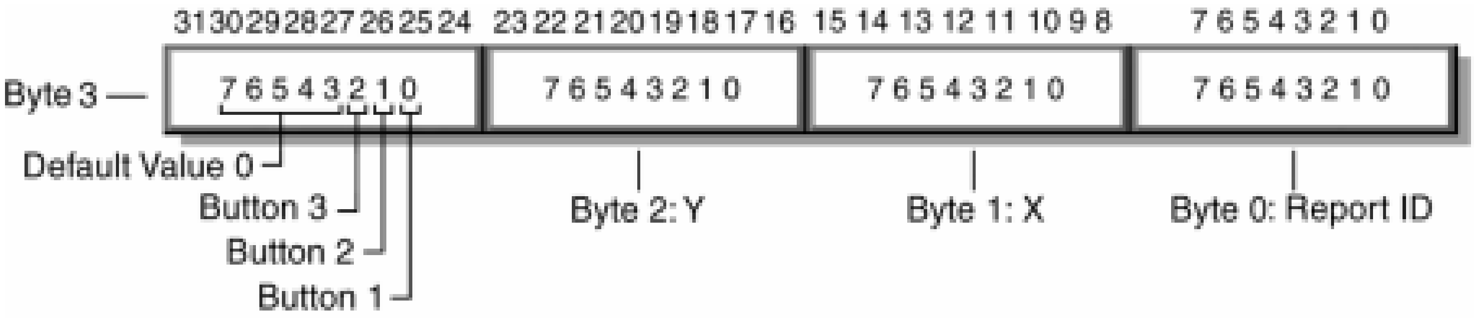
- Virtual Serial Device
  - Endpoint 0: Control, IN/OUT
    - Respond to IN requests by setting up OUT with a buffer of descriptor data of the correct size
  - Endpoint 1: Interrupt, IN
    - Needed for serial modem controls, just ignore it
  - Endpoint 2: Bulk, OUT
    - Connect to buffer from `_write()`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

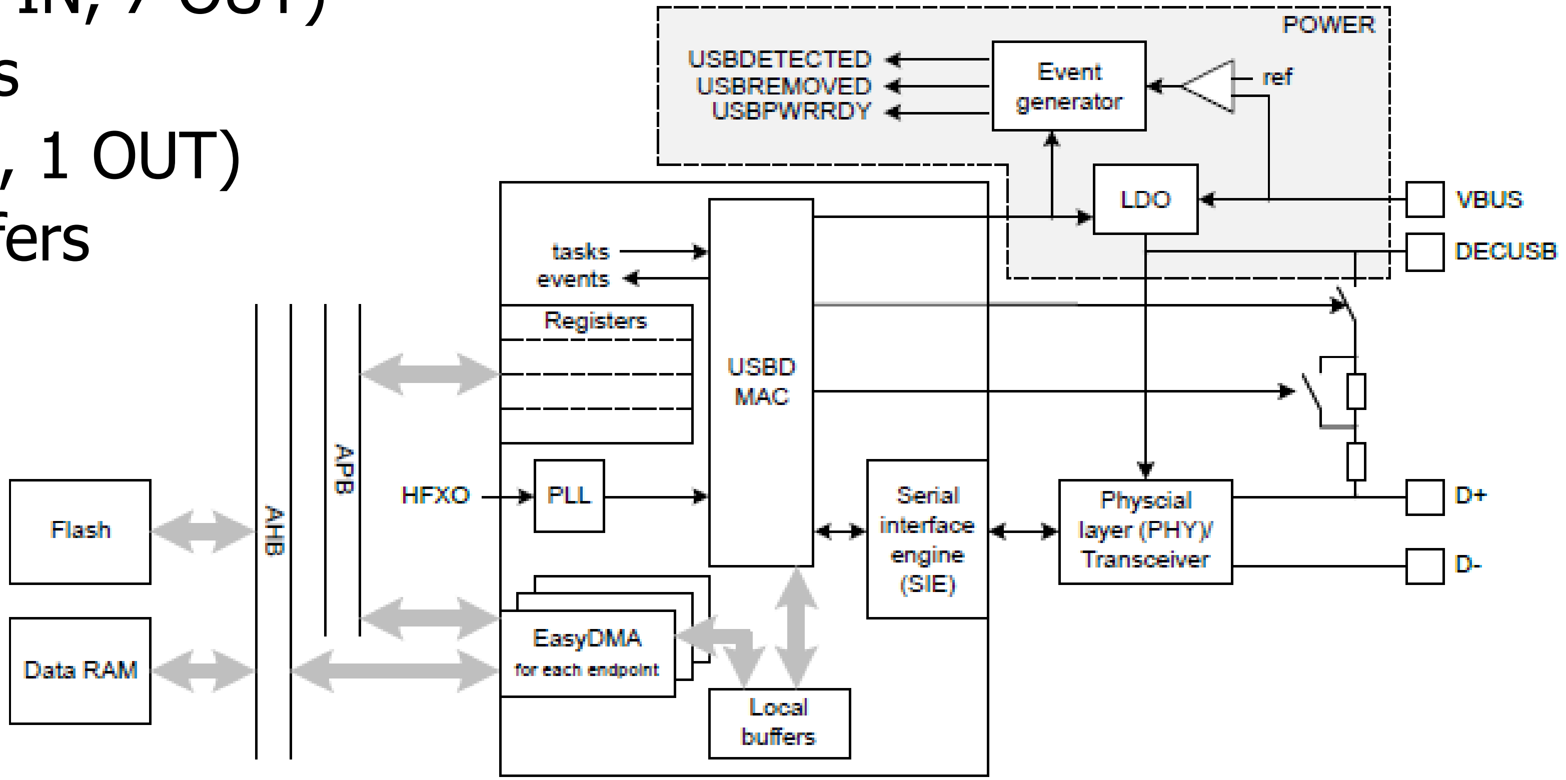
- Specification for fast data communication
- Specification for interacting with abstract device types
  - Connects correct driver to interpret and send data

- 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



Break + Question

- What are the ramifications of many USB devices sharing a bus?
  - Consider: throughput and latency

- What if I really had 127 USB mice on a single USB hub?
  - What if it was microphones instead?

Outline

- USB
- **CAN**

Controller Area Network (CAN bus)

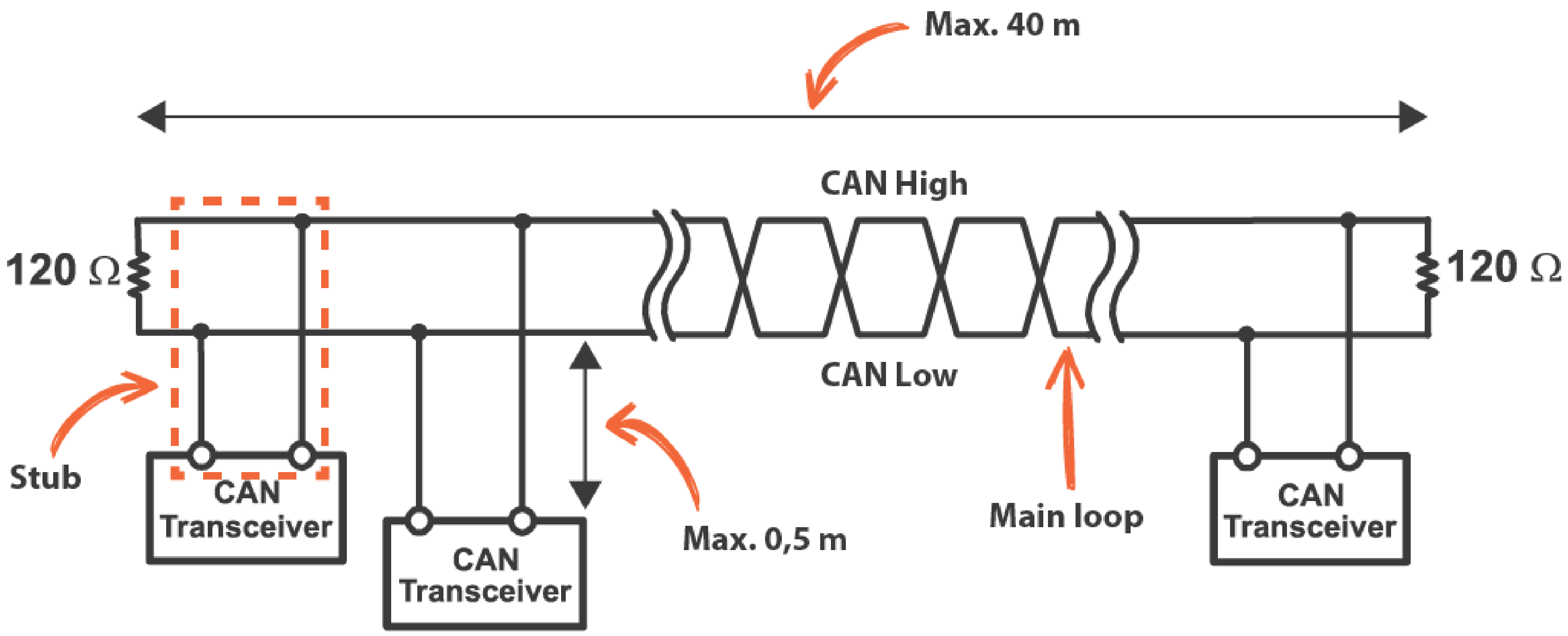
- Designed for highly reliable interactions within a vehicle

- Multi-master with arbitration
  - Similar to I2C

- Mechanism for sending messages with “identifiers”
  - Identifies the data in the message, not the device its for
  - 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

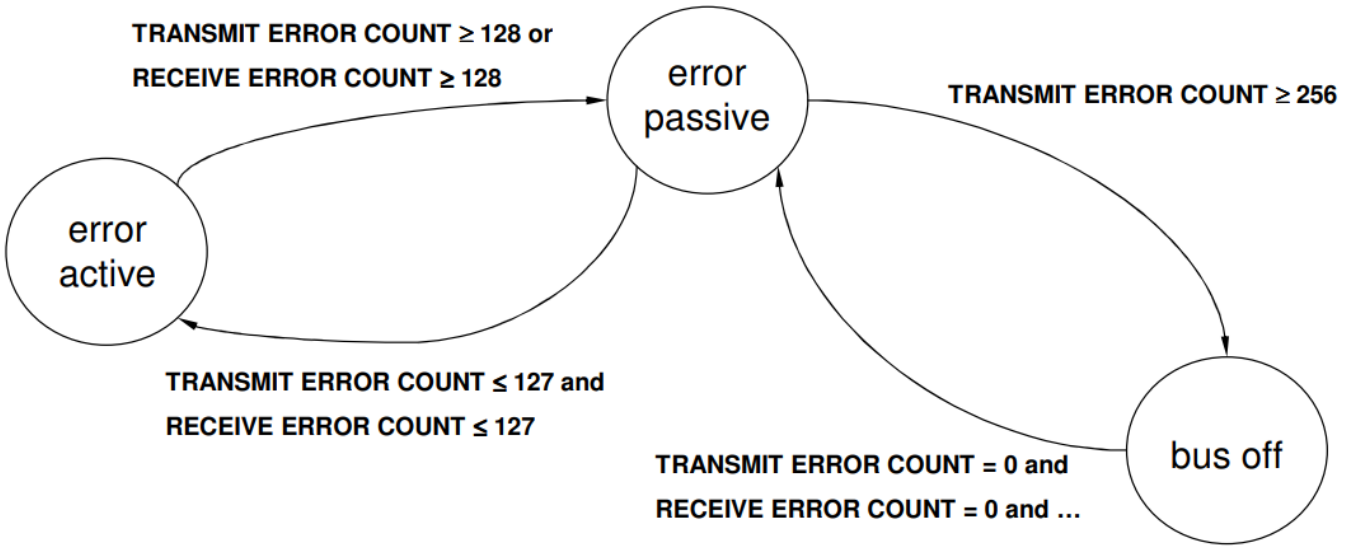
- Check for errors everywhere and appropriately handle
  - Bit error
    - If the value found on the bus differs from the one sent
  - Stuff error
    - If 6 consecutive bits of the same type are found
  - CRC error
    - If CRC does not match
  - Form error
    - Format field has unexpected values
  - 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

- USB

- CAN