Last Time

- Embedded networks
  - Characteristics
  - Requirements
  - Simple embedded LANs
    - Bit banged
    - SPI
    - I2C
    - LIN
    - Ethernet

Today

- CAN Bus
  - Intro
  - Low-level stuff
  - Frame types
  - Arbitration
  - Filtering
  - Higher-level protocols

Motivation

- Some new cars contain > 3 miles of wire
- Clearly inappropriate to connect all pairs of communicating entities with their own wires
  - $O(n^2)$ wires
- CAN permits everyone on the bus to talk
  - Cost ~$3 / node
    - $1 for CAN interface
    - $1 for the transceiver
    - $1 for connectors and additional board area

CAN Bus

- Cars commonly have multiple CAN busses
  - Physical redundancy for fault tolerance
- CAN nodes sold
  - 200 million in 2001
  - 300 million in 2004
  - 400 million in 2009

What is CAN?

- Controller Area Network
  - Developed by Bosch in the late 1980s
  - Current version is 2.0, from 1991
- Multi-master serial network
- Bus network: All messages seen by all nodes
- Highly fault tolerant
- Resistant to interference
- Lossless in expected case
- Real-time guarantees can be made about CAN performance

More about CAN

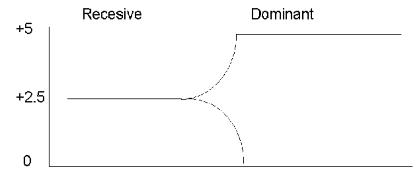
- Message based, with payload size 0-8 bytes
  - Not for bulk data transfer!
  - But perfect for many embedded control applications
- Bandwidth
  - 1 Mbps up to 40 m
  - 40 Kbps up to 1000 m
  - 5 Kbps up to 10,000 m
- CAN interfaces are usually pretty smart
  - Interrupt only after an entire message is received
  - Filter out unwanted messages in HW – zero CPU load
- Many MCUs – including ColdFire – have optional onboard CAN support

CAN Bus Low Level

- CAN does not specify a physical layer
- Common PHY choice: Twisted pair with differential voltages
  - Resistant to interference
  - Can operate with degraded noise resistance when one wire is cut
  - Fiber optic also used, but not commonly
- Each node needs to be able to transmit and listen at the same time
  - Including listening to itself

Dominant and Recessive

- Bit encoding:
  - Voltage difference → "dominant" bit == logical 0
  - No voltage difference → "recessive" bit == logical 1



Bus Conflict Detection

- Bus state with two nodes transmitting:

		Node 2: dominant	Node 2: recessive
Node 1	dominant	dominant	dominant
Node 1	recessive	dominant	recessive

- So:
  - When a node transmits dominant, it always hears dominant
  - When a node transmits recessive and hears dominant, then there is a bus conflict
- Soon we'll see why this is important

More Low Level

- CAN Encoding: Non-return to zero (NRZ)
  - Lots of consecutive zeros or ones leave the bus in a single state for a long time
  - In contrast, for a Manchester encoding each bit contains a transition
- NRZ problem: Not self-clocking
  - Nodes can easily lose bus synchronization
- Solution: Bit stuffing
  - After transmitting 5 consecutive bits at either dominant or recessive, transmit 1 bit of the opposite polarity
  - Receivers perform destuffing to get the original message back

CAN Clock Synchronization

- Problem: Nodes rapidly lose sync when bus is idle
  - Idle bus is all recessive – no transitions
  - Bit stuffing only applies to messages
- Solution: All nodes sync to the leading edge of the "start of frame" bit of the first transmitter
- Additionally: Nodes resynchronize on every recessive to dominant edge
- Question: What degree of clock skew can by tolerated by CAN?
  - Hint: Phrase skew as ratio of fastest to slowest node clock in the network

CAN is Synchronous

- Fundamental requirement: Everyone on the bus sees the current bit before the next bit is sent
  - This is going to permit a very clever arbitration scheme
  - Ethernet does NOT have this requirement
    - This is one reason Ethernet bandwidth can be much higher than CAN
- Let's look at time per bit:
  - Speed of electrical signal propagation 0.1-0.2 m/ns
  - 40 Kbps CAN bus → 25000 ns per bit
    - A bit can travel 2500 m (max bus length 1000 m)
  - 1 Mbps CAN bus → 1000 ns per bit
    - A bit can travel 100 m (max bus length 40 m)

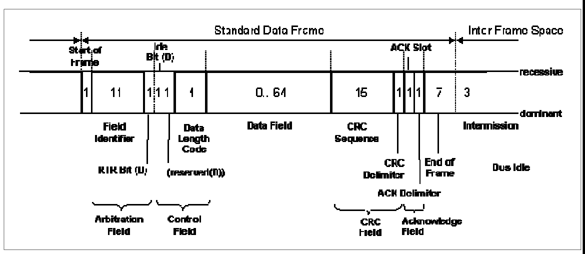
CAN Addressing

- Nodes do not have proper addresses
- Rather, each message has an 11-bit “field identifier”
  - In extended mode, identifiers are 29 bits
- Everyone who is interested in a message type listens for it
  - Works like this: “I’m sending an oxygen sensor reading”
  - Not like this: “I’m sending a message to node 5”
- Field identifiers also serve as message priorities
  - More on this soon

CAN Message Types

- Data frame
  - Frame containing data for transmission
- Remote frame
  - Frame requesting the transmission of a specific identifier
- Error frame
  - Frame transmitted by any node detecting an error
- Overload frame
  - Frame to inject a delay between data and/or remote frames if a receiver is not ready

CAN Data Frame



- Bit stuffing not shown here – it happens below this level

Data Frame Fields

- RTR – remote transmission request
  - Always dominant for a data frame
- IDE – identifier extension
  - Always dominant for 11-bit addressing
- CRC – Based on a standard polynomial
- CRC delimiter – Always recessive
- ACK slot – This is transmitted as recessive
  - Receiver fills it in by transmitting a dominant bit
  - Sender sees this and knows that the frame was received
    - By at least one receiver
- ACK delimiter – Always recessive

Remote Frame

- Same as data frame except:
  - RTR bit set to recessive
  - There is no data field
  - Value in data length field is ignored

Error Checking

- Five different kinds of error checking are performed by all nodes
- Message-level error checking
  - Verify that checksum checks
  - Verify that someone received a message and filled in the ack slot
  - Verify that each bit that is supposed to be recessive, is
- Bit-level error checking
  - Verify that transmitted and received bits are the same
    - Except identifier and ack fields
  - Verify that the bit stuffing rule is respected

Error Handling

- Every node is in error-active or error-passive state
  - Normally in error-active
- Every node has an error counter
  - Incremented by 8 every time a node is found to be erroneous
  - Decremented by 1 every time a node transmits or receives a message correctly
- If error counter reaches 128 a node enters error-passive state
  - Can still send and receive messages normally
- If error counter reaches 256 a node takes itself off the network

Error Frame

- Active error flag – six consecutive dominant bits
  - This is sent by any active-error node detecting an error at any time during a frame transmission
  - Violates the bit stuffing rule!
    - This stomps the current frame – nobody will receive it
  - Following an active error, the transmitting node will retransmit
- Passive error flag – six consecutive recessive bits
  - This is "sent" by any passive-error node detecting an error
  - Unless overwritten by dominant bits from other nodes!
- After an error frame everyone transmits 8 recessive bits

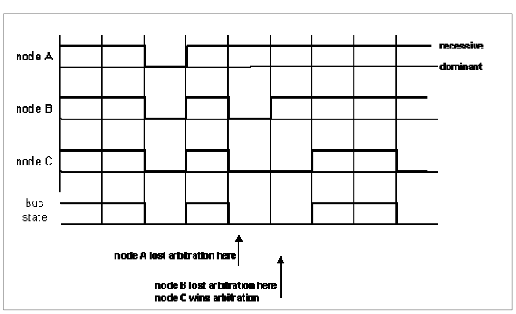
Bus Arbitration

- Problem: Control access to the bus
- Ethernet solution: CSMA/CD
  - Carrier sense with multiple access – anyone can transmit when the medium is idle
  - Collision detection – Stomp the current packet if two nodes transmit at once
    - Why is it possible for two nodes to transmit at once?
  - Random exponential backoff to make recurring collisions unlikely
- Problems with this solution:
  - Bad worst-case behavior – repeated backoffs
  - Access is not prioritized

CAN Arbitration

- Nodes can transmit when the bus is idle
- Problem is when multiple nodes transmit simultaneously
  - We want the highest-priority node to "win"
- Solution: CSMA/BA
  - Carrier sense multiple access with bitwise arbitration
- How it works:
  - Two nodes transmit start-of-frame bit
    - Nobody can detect the collision yet
  - Both nodes start transmitting message identifier
    - As soon as the identifiers differ at some bit position, the node that transmitted recessive notices and aborts the transmission

Multiple Colliding Nodes



Arbitration Continued

- Consequences:
  - Nobody but the losers see the bus conflict
  - Lowest identifier always wins the race
  - So: Message identifiers also function as priorities
- Nondestructive arbitration
  - Unlike Ethernet, collisions don't cause drops
  - This is cool!
- Maximum CAN utilization: ~100%
  - Maximum Ethernet with CSMA/CD utilization: ~37%

CAN Message Scheduling

- **Network scheduling is usually non-preemptive**
  - Unlike thread scheduling
  - Non-preemptive scheduling means high-priority sender must wait while low-priority sends
  - Short message length keeps this delay small
- **Worst-case transmission time for 8-byte frame with an 11-bit identifier:**
  - 134 bit times
  - 134 μs at 1 Mbps

"Babbling Idiot" Error

- **What happens if a CAN node goes haywire and transmits too many high priority frames?**
  - This can make the bus useless
  - Assumed not to happen
- **Schemes for protecting against this have been developed but are not commonly deployed**
  - Most likely this happens very rarely
  - CAN bus is usually managed by hardware

CAN on ColdFire

- **52233 does not have CAN**
  - But sibling chips 52231, 53324, and 52235 have "FlexCAN"
- **16 message buffers**
  - Each can be used for either transmit or receive
  - Buffering helps tolerate bursty traffic
- **Transmission**
  - Both priority order and queue order are supported
- **Receiving**
  - FlexCAN unit looks for a receive buffer with matching ID
  - Some ID bits can be specified as don't cares

More CAN on CF

- **Interrupt sources**
  - Message buffer
    - 32 possibilities – successful transmit / receive from each of the 16 buffers
  - Error
  - Bus off – too many errors

Higher Level Standards

- **CAN leaves much unspecified**
  - How to assign identifiers?
  - Endianness of data?
- **Standardized higher-level protocols built on CAN:**
  - CANKingdom
  - CANOpen
  - DeviceNet
  - J1939
  - Smart Distributed System
- **Similar to how**
  - TCP is built in IP
  - HTTP is built in TCP
  - Etc.

CANOpen

- **Important device types are described by device profiles**
  - Digital and analog I/O modules
  - Drives
  - Sensors
  - Etc.
- **Profiles describe how to access data, parameters, etc.**

CAN Summary

- Not the cheapest network
  - E.g., LIN bus is cheaper
- Not suitable for high-bandwidth applications
  - E.g. in-car entertainment – streaming audio and video
  - MOST – Media Oriented Systems Transport
- Design point:
  - Used where reliable, timely, medium-bandwidth communication is needed
  - Real-time control of engine and other major car systems