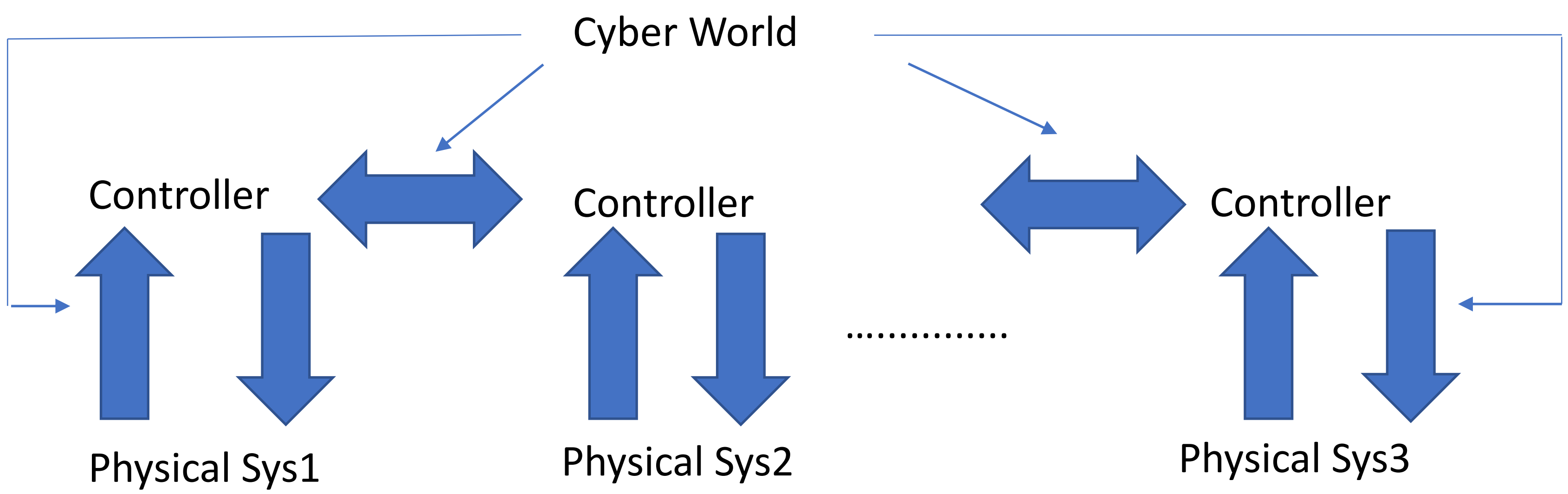
COMPUTATIONAL FOUNDATIONS OF CYBER PHYSICAL SYSTEMS (CS61063)

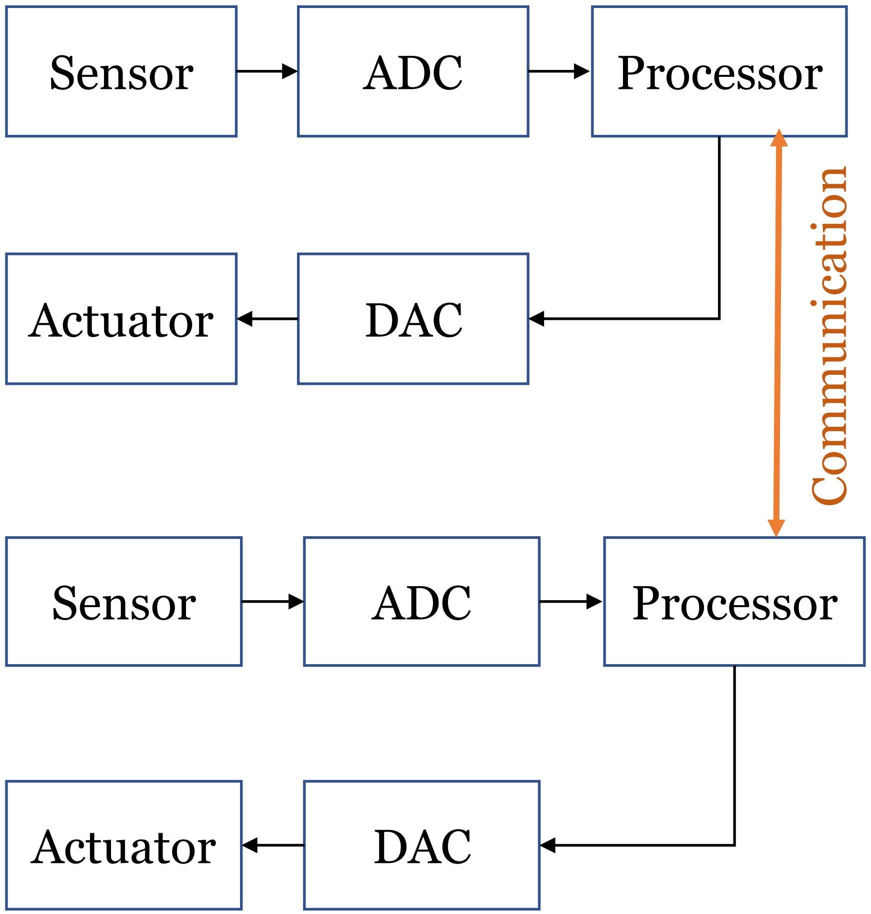


- Soumyajit Dey
- CSE, IIT Kharagpur

CPS Compute Platforms

- CPS involves significant on-board computation
  - Signal processing – filtering the plant state data
  - State estimation
  - On-board intelligence (can run several optimizations for real time problem solving)
- Low power computation
  - Typically performed with help from on board battery
- Need to use low power processors instead of workstation class processors
  - RISC CPUs- ARM, PowerPC
  - Microcontrollers – Atmel (8 bit RISC ATmega328)

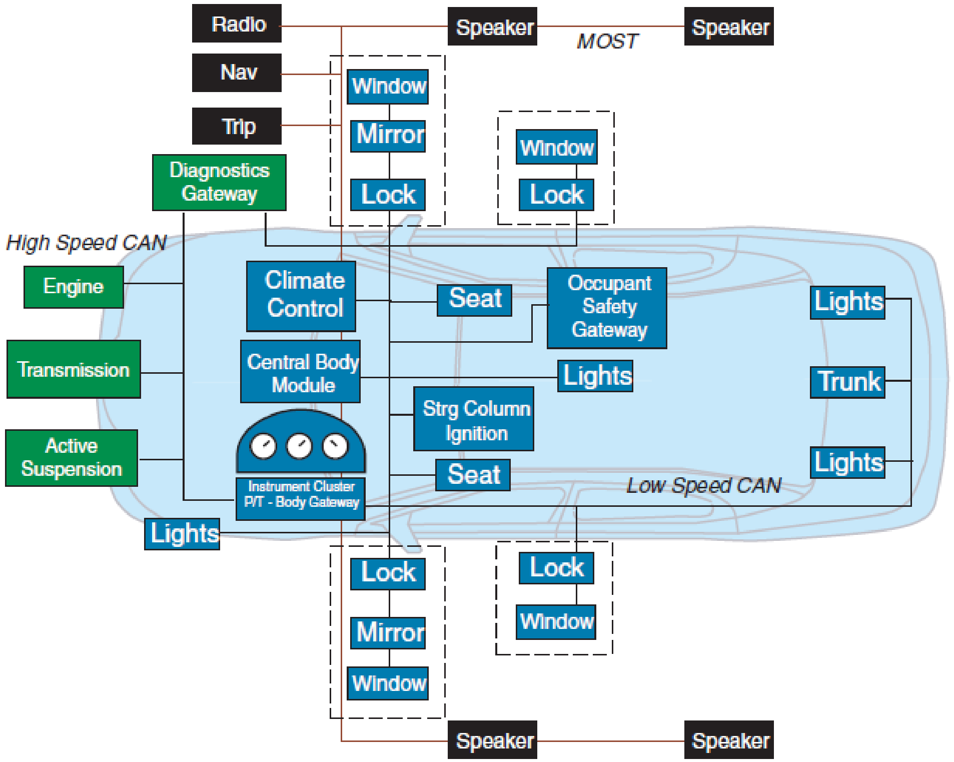
CPS Implementation: What about Communication



- Real time guarantee is required
- Modern automobiles may have seventy or more microprocessors communicating over several networks to accomplish shared tasks
- Consider a car: The on-board control loops shall exchange messages at a high rate (~10 – 100 ms)
- Some new cars contain more than 3 miles of wire
- Inappropriate to connect all pairs of communicating entities with their own wires - $O(n^2)$ wires
- A communication protocol is needed that ensures lossless fast data transmission which is crucial for a real time safety critical system

Introduction to CAN

- ➢The most commonly used network for control in automotive and manufacturing applications is the Controller Area Network, or CAN
- ➢Developed by Robert Bosch GmbH for automotive applications in the late 1980s. Current version is 2.0 from 1991



Introduction to CAN

- All nodes are connected to a bus
- A central node may or may not be present
- Nodes may be added anytime, even if the network is operating (**hot-plugging**)
- **Multimaster**- When the bus is free any node can send a message
- **Multicast**- All nodes may receive and act on the message
- CAN interconnects a network of modules(or nodes)using two wire, **twisted pair cable** which makes it **resistant to interference**
- Highly **fault tolerant**
- **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

- CAN does not specify a physical layer
- Common PHY choice: Twisted pair with **differential voltage**
  - Current flowing in each signal is equal but opposite in direction (**balanced**)
  - Results in a field-cancelling effect - key to **low noise emissions**
  - Can operate with degraded noise resistance when one wire is cut
  - Fiber optic is also used, but not commonly
- Each node needs to be able to transmit and listen at the same time
  - Including listening to itself

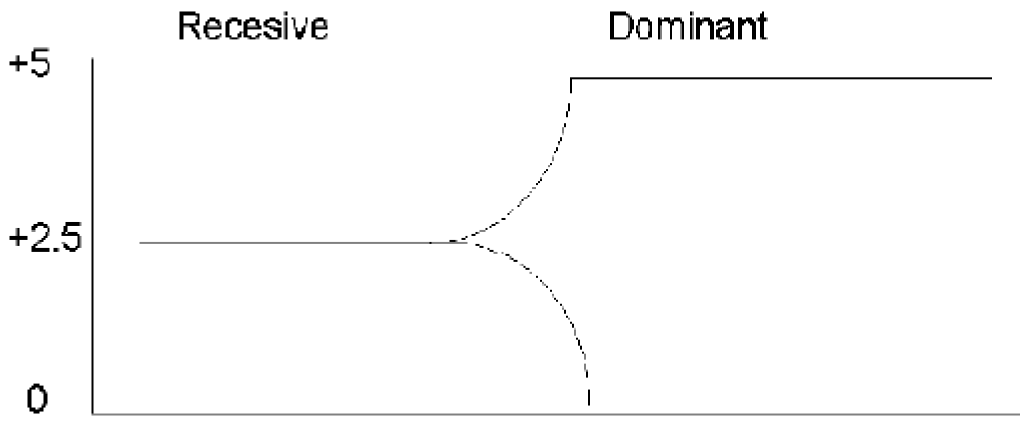
CAN Bus

- 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 nodes sold
  - 200 million in 2001
  - 300 million in 2004
  - 400 million in 2009

Dominant and Recessive

- Two signal lines of the bus, CANH and CANL, in recessive state, are passively biased to 2.5 V creating no voltage difference.
- Dominant state of the bus takes CANH 1 V higher to 3.5 V and takes CANL 1 V lower to 1.5 V, creating 2V difference.
- Bit encoding
  - Dominant state is encoded as logic "0"
  - Recessive state is encoded as logic "1"



Bus Synchronization: Bit stuffing

- 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 de-stuffing to get the original message back

CAN Messages

SOF	ID	RTR	Reserved	DLC	Data Field	CRC	CRC Delimiter	ACK	ACK Delimiter	EOF
1 bit	11 bits	1 bit	2 bits	4 bits	0-8 Bytes	15 bits	1 bit	1 bit	1 bit	7 bits
Arbitration Field			Control Field			CRC Field		Acknowledge Field		

SOF: Start of Frame. Always dominant.

ID: Identifier that indicates priority. Lower the binary value, higher the priority

RTR: Remote Transmission Request

DLC: Data Length Code indicates number of bytes of data being transmitted

CRC: Cyclic Redundancy Check. Based on a standard polynomial

CRC Delimiter: Always recessive

ACK: Acknowledge. Transmitter sends recessive and receiver asserts dominant. Hence, transmitter knows that frame was received by at least one receiver.

ACK Delimiter: Always recessive

EOF: End of Frame. Always recessive

CAN Clock Synchronization Using SOF

- **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"

CAN Messages

4 types of CAN messages:

1. **Data frame** is the standard CAN message, broadcasting data from the transmitter to the other nodes on the bus.
2. **Remote frame** is broadcast by a transmitter to request data from a specific node.
3. **Error frame** may be transmitted by any node that detects a bus error.
4. **Overload frame**s are used to introduce additional delay between data or remote frames.

CAN Data Frame

- Composed of 7 fields: SOF, arbitration, control data, CRC, ACK and EOF
- SOF field consists of one dominant bit. All network nodes waiting to transmit synchronize with the SOF and begin transmitting at the same time

CAN Data Frame: Arbitration

- Determines which of the nodes attempting to transmit will actually control the bus
- Ethernet solution: CSMA/CD
  - Carrier sense with multiple access – anyone can transmit when the medium is idle
  - Collision detection – Stop the current packet if two nodes transmit at once
    - Why is it possible for two nodes to transmit at once?
  - Random exponential back-off to make recurring collisions unlikely
- Problems with this solution:
  - Bad worst-case behavior – repeated back-offs
  - Access is not prioritized

CAN Data Frame: 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)
  - Message priority is determined by the numerical value of the identifier in the arbitration field, with the lowest numerical value having the highest priority
  - Non-destructive, bit-wise arbitration
  - **Nondestructive** means that the node winning arbitration just continues on with the message, without the message being destroyed or corrupted by another node.
  - **Bit-wise** means the bus can be thought of as acting like an AND gate

CAN Data Frame: Arbitration

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

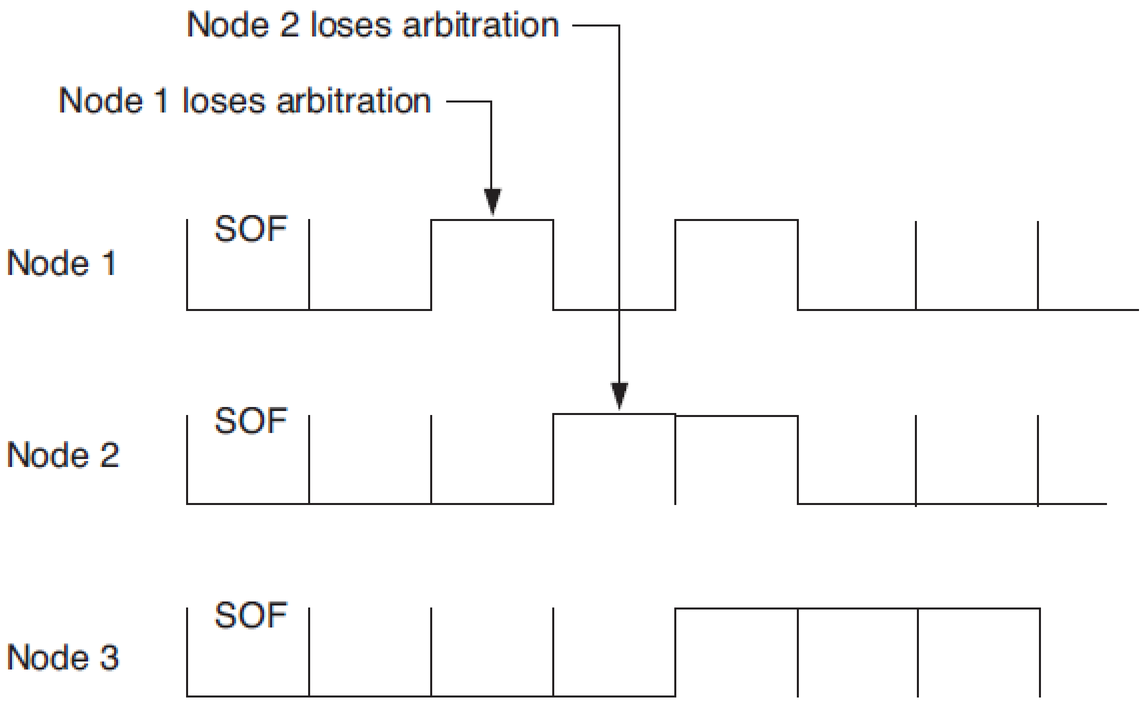
CAN Data Frame: Arbitration

How does it work?

- Two nodes transmit start-of-frame bit
  - Nobody can detect the collision yet
- Both nodes start transmitting message identifier
- If any node writes a dominant (0) bit on the bus, every node will read a dominant bit regardless of the value written by that node.
- Every transmitting node always reads back the bus value for each bit transmitted.
- As soon as the identifiers differ at some bit position, the node that transmitted recessive notices and aborts the transmission

CAN Data Frame: Arbitration

Multiple Colliding Nodes:



CAN Arbitration: Node 3 has highest, and Node 1 the lowest, priority messages.

CAN Data Frame: Arbitration

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 Data Frame: Error Handling

Message Level Checks:

1. **CRC Check:** Each receiver accepting the message recalculates the CRC and compares it against the transmitted value. A discrepancy between the two calculations causes an error flag to be set.
2. **Frame Check:** Error flag is set when receiver detects an invalid bit in the CRC delimiter, ACK delimiter, EOF or 3-bit inter-frame space.
3. **ACK Check:** Each receiving node writes a dominant bit into the ACK slot of the message frame that is read by the transmitting node. If a message is not acknowledged, an ACK error is flagged.

CAN Data Frame: Error Handling

Bit Level Checks:

1. **Monitoring:** Each transmitted bit is "read back" by the transmitter. If the monitored value is different than the value being sent, a bit error is detected.
2. **Stuffing**: If any node detects six consecutive bits of the same level, a stuff error is flagged.

CAN Error Frame

- On detecting error, transmitting or receiving node will immediate abort the transmission and broadcast an error frame
- Consists of an active error flag with 6 dominant bits and an error flag delimiter with 8 recessive bits
  - Violates bit staffing rule
  - All other nodes respond by transmitting error flags too
- **Error count** determines whether the errors are transient or permanent
  - Each node tracks number of errors detected

CAN Error Frame

- "**Error active**": 0<error count<128
  - Node remains fully functional
  - At least one error is detected
  - On detecting an error, node sends active error flag containing 6 dominant bits
- "**Error passive**": 128≤error count<255
  - Node will transmit at slower rate
  - On detecting an error, node sends passive error flag containing 6 recessive bits
- "**Bus off**": error count≥255
  - Node takes itself offline
- Receive error and Transmit error increment error count by 1 and 8 respectively
- Error free message decrements error count by 1

Valid CAN Frame

- A message is considered to be error free when the last bit of the ending EOF field of a message is received in the error-free recessive state.
- A dominant bit in the EOF field causes the transmitter to repeat a transmission.
- Corrupted messages are automatically retransmitted as soon as the bus is idle

CAN Remote Frame

- A node that requires data from another node on the network can request a transmission by sending a Remote Frame
  - Microprocessor controlling the central locking on your car may need to know the state of the transmission gear selector from the powertrain controller
- A remote frame is the same as a data frame, without the data field
- In data frames, the RTR bit must be dominant; in remote frames it must be recessive

CAN Overload Frames and Interframe Space

- If a CAN node receives messages faster than it can process them, then an Overload Frame will be generated to provide extra time between successive Data or Remote frames
- Overload frame consists of overload flag with 6 dominant bits and an overload delimiter with 8 recessive bits
- Interframe Space consists of a 3 recessive bits intermission and the bus idle time between Data or Remote Frames
- During the intermission, no node is permitted to initiate a transmission
  - If a dominant bit is detected during the Intermission, an Overload Frame will be generated
- The bus idle time lasts until a node has something to transmit
  - At this time a detection of dominant bit on the bus signals a SOF

Bus Loading

- Bus utilization = total bit consumption / total bits available
- Steps to calculate bus utilization:
  1. Choose a time unit ≥ the slowest fixed periodic message on network
  2. Identify all periodic messages
  3. For each of these messages approximate the total bit size of the message by adding 47 bits to the size of each data field
  4. Message bits consumed = message bit size * number of transmissions performed in one time unit
  5. Total periodic bits consumed = ∑ message bits consumed
  6. Total bits consumed = 1.1 * total periodic bits consumed (worst case traffic)
  7. Bandwidth consumption (%) = (total periodic bits consumed / total bits available) * 100

Bus Loading Example

Total bits available = 125 kbps

Message	Data (bytes)	Message Size (bits)	Rate and Period	Message Bits Consumed
MsgA	0	47	10 trx/s: 100 ms	$10 \cdot 47 = 470$ bps
MsgB	5	$5 \cdot 8 + 47 = 87$	2 trx/s: 500 ms	$87 \cdot 2 = 174$ bps
MsgC	8	$8 \cdot 8 + 47 = 111$	1 trx/s: 1 s	$111 \cdot 1 = 111$ bps
…	…	…	…	…
			Total periodic bits consumed	10000 bps
Total Bits Consumed = 1.1·(Total Periodic Bits Consumed) = 11000 bps Bandwidth Consumption = 11000/125000 = 8.8%				

Drawbacks of CAN

- Message latency is non-determinant
  - Existence of Error Frames, Overload Frames and retransmissions
- Latency increases with amount of traffic on the bus

Time Triggered CAN (TTCAN)

- Communication protocol for real time systems must guarantee that messages meet timing deadlines regardless of the load on the bus
  - – TTCAN protocol that retains CAN data link layer protocol
- TTCAN is based on a time triggered and periodic communication which is clocked by a time master's reference message
- The period between two consecutive reference messages is called the basic cycle
- The time windows of a basic cycle can be used for periodic state messages (**exclusive**) and for spontaneous state and event messages (**arbitrating**)

More on TTCAN

- Exclusive window does not compete for bus access, whereas arbitrating window does.
- For fault tolerance, there must be multiple potential master nodes
- TTCAN **does not** re-broadcast corrupted messages, nor does it invoke Error Frames

FlexRay: A competitive protocol to TTCAN

- Communication frame consists of periodically triggered "static" and "dynamic" parts
- Static segment is made up of identical length time slots assigned to connected nodes
  - Each node transmits its messages synchronously in its reserved slot
  - Unlike CAN, there is no arbitration for the bus
- Dynamic segment is a "polling" mechanism
  - Each node is given the opportunity to put an event-triggered or asynchronous message on the bus in priority order using a "mini-slotting" timing mechanism
- For redundant fault tolerance, nodes may be connected to two buses, or channels simultaneously

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

CAN on ColdFire

- 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

- CAN is ideally suited in applications requiring a large number of short messages with high reliability
- Especially well suited when data is needed by more than one location and system-wide data consistency is mandatory
  - CAN is message based and not address based
- Faulty nodes are automatically dropped from the bus
  - Prevents any single node from bringing a network down
  - Ensures that bandwidth is always available for critical message transmission

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