isep Instituto Superior de Engenharia do Porto

P.PORTO

Communication Technologies for Critical Systems

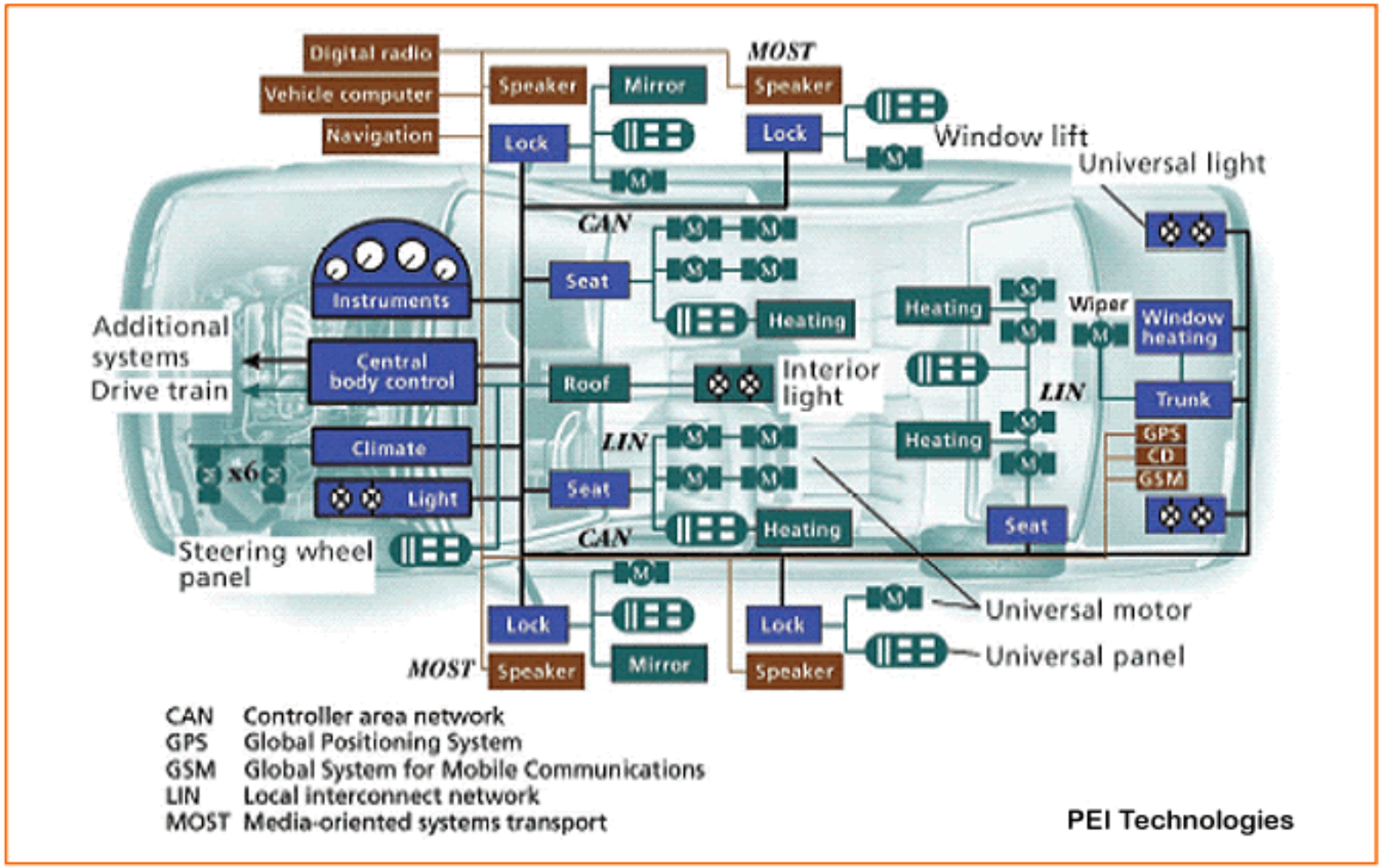
Communication Protocols: Controler Area Network (CAN)

Mestrado em Engenharia de Sistemas Computacionais Críticos

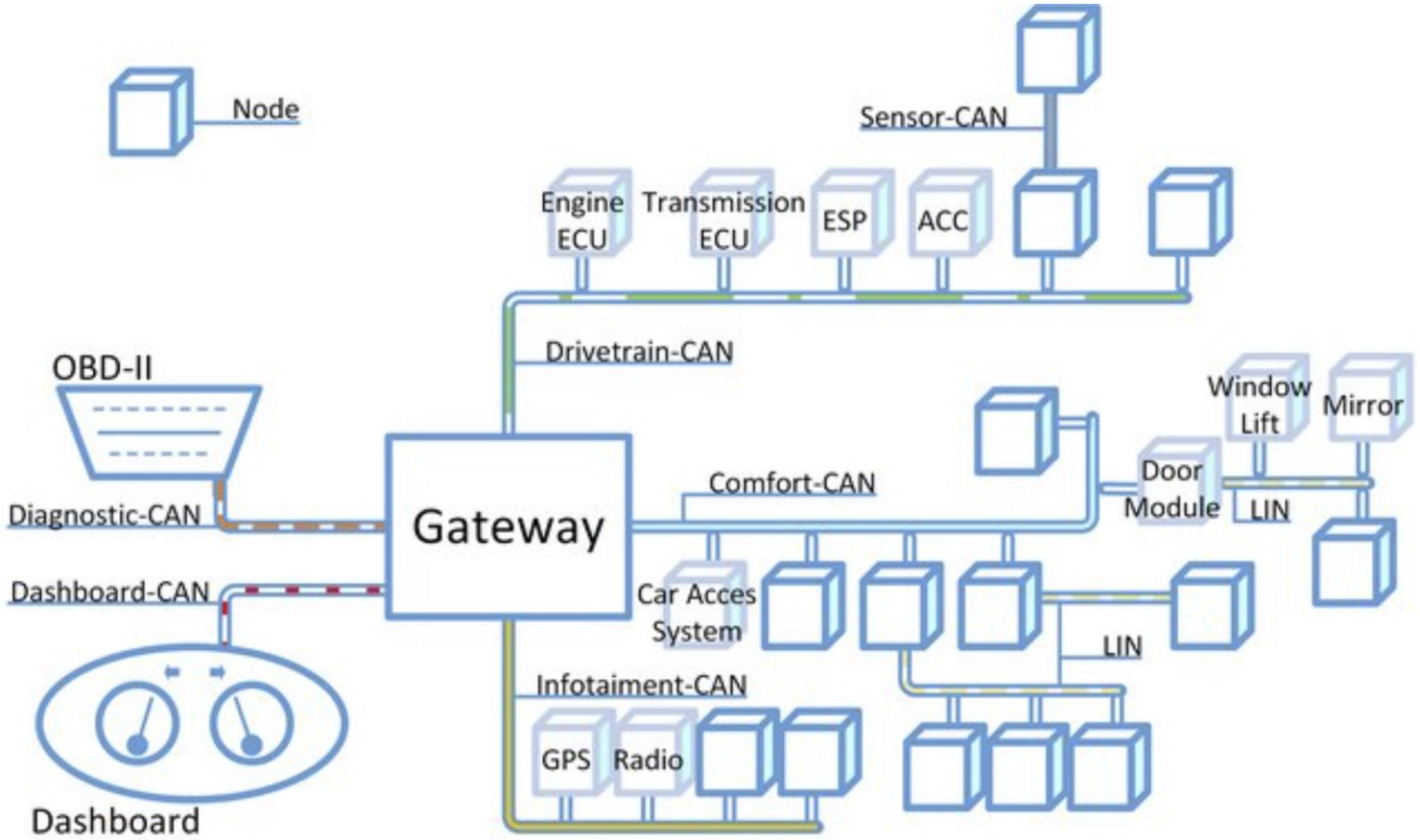
History of CAN

- Created in mid-1980s for automotive applications by Robert Bosch.
- Design goal was to make automobiles more reliable, safer, and more fuel efficient
  - by reducing wiring weight and costs
  - Improving the in-car communication capabilities
- The latest CAN specification is the version 2.0 made in 1991.
- First CAN controller chips:
  - Intel (82526) and Philips (82C200) in 1987
- First production car using CAN
  - 1991 Mercedes S-class (W140

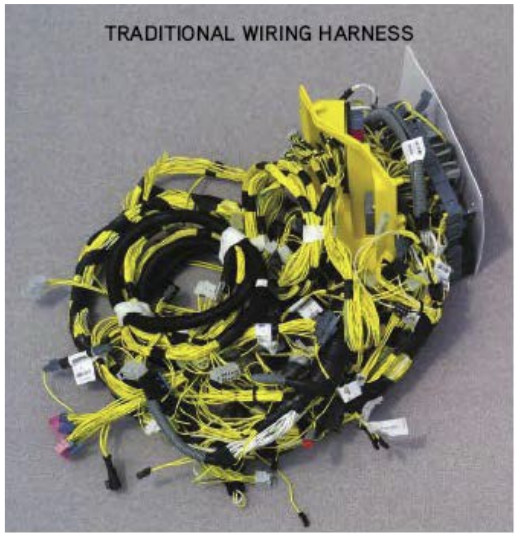
CAN systems


Digital radio
Vehicle computer
Navigation
Speaker
Mirror
Lock
MOST
Speaker
Lock
Window lift
Universal light
CAN
Seat
Heating
Instruments
Additional systems
Drive train
Central body control
Roof
Interior light
Heating
Wiper
Window heating
LIN
Trunk
Climate
x6
Light
LIN
Seat
Heating
GPS
CD
GSM
Steering wheel panel
CAN
Heating
Seat
Universal motor
Universal panel
Lock
Mirror
Lock
MOST
Speaker
Speaker
CAN Controller area network
GPS Global Positioning System
GSM Global System for Mobile Communications
LIN Local interconnect network
MOST Media-oriented systems transport
PEI Technologies

CAN systems


Node
Sensor-CAN
Engine ECU
Transmission ECU
ESP
ACC
Drivetrain-CAN
OBD-II
Window Lift
Mirror
Door Module
LIN
Comfort-CAN
Diagnostic-CAN
Gateway
Dashboard-CAN
Car Acces System
LIN
Infotaiment-CAN
GPS
Radio
Dashboard

Main (initial) motivation



- Traditional point-to-point wiring
  - 30Kg wiring harness
  - 1km of copper wire
  - 300 connectors, 2000 terminals, 1500 wires
- Expensive to manufacture, install and maintain
  - Door system with 50+ wires
- Multiplex approach (e.g. CAN)
  - Massive reduction in wiring costs
  - Door system reduced to just 4 wires
  - Small added cost of CAN controllers, transceivers etc.
    - Reduced as CAN devices became on-chip peripherals

CAN today

- CAN is used in nearly all cars sold today
  - +-1 billion CAN enabled microcontrollers sold each year
  - Cars today have 20 – 30 ECUs inter-connected via 2 or more CAN buses
- Multiple networks
  - High speed" (500 Kbit/sec) network connecting chassis and power train ECUs
    - transmission control, engine management, ABS etc.
  - Low speed (100-125 Kbit/sec) network(s) connecting body and comfort electronics
    - Door modules, seat modules, climate control etc.
  - Data required by ECUs on different networks
    - typically "gatewayed" between them via a powerful microprocessor connected to both

Information Examples

- CAN used to communicate signals between ECUs
  - Signals typically range from 1 to 16-bits of information
  - Wheel speeds, oil and water temperature, battery voltage, engine rpm, gear selection, accelerator position, dashboard switch positions, climate control settings, window switch positions, fault codes, diagnostic information etc.
  - > 2,500 signals in a high-end vehicle
  - Multiple signals piggybacked into CAN messages to reduce overhead, but still 100's of CAN messages
- **Real-time constraints on signal transmission**
  - End-to-end deadlines in the range **10ms – 1sec**
  - Example **LED brake lights**

CAN basic concepts

- prioritization of messages
- guarantee latency times
- configuration flexibility
- multicast reception with time synchronization
- system wide data consistency
- multimaster
- error detection and signalling
- automatic retransmission of corrupted messages as soon as the bus is idle again
- distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

CAN layers

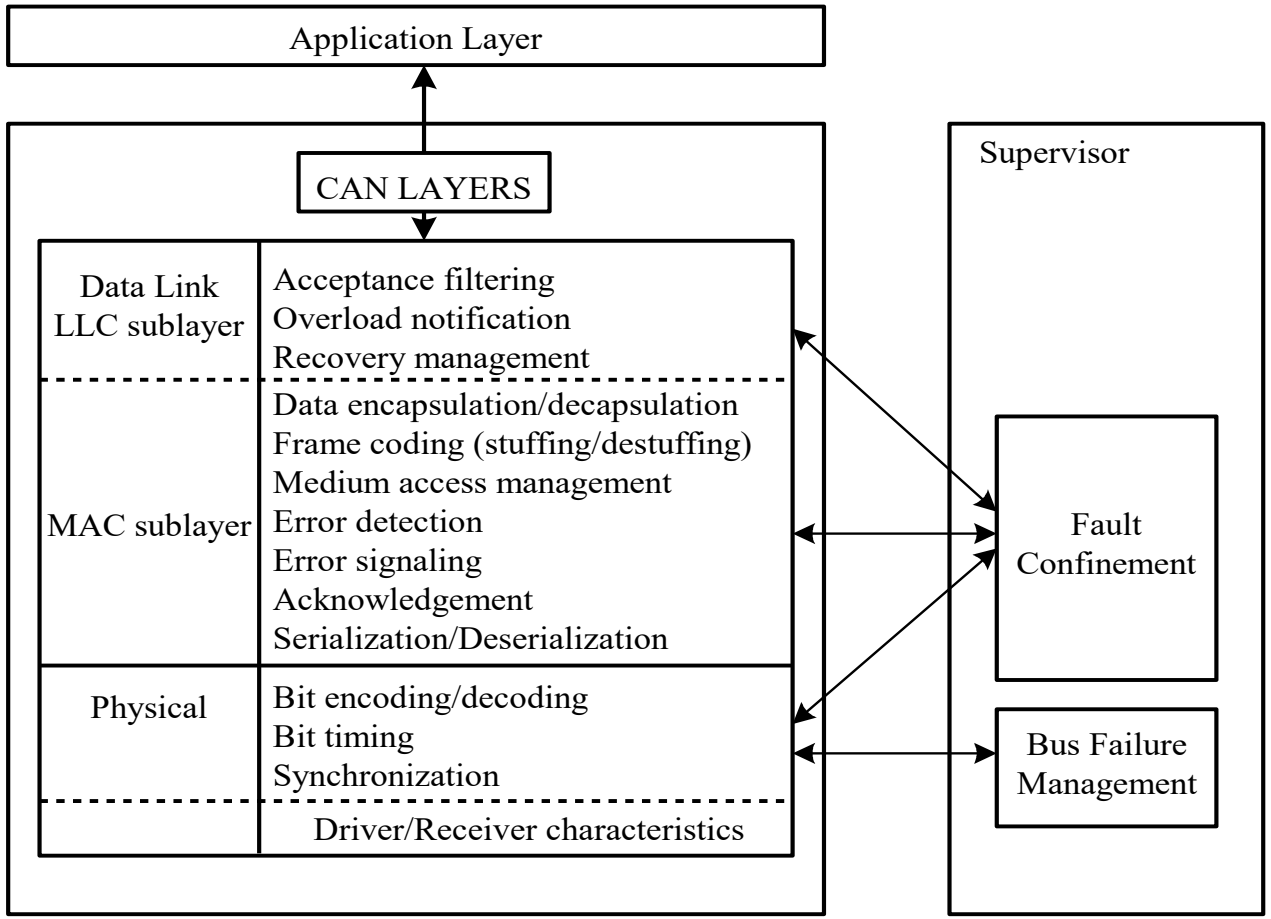
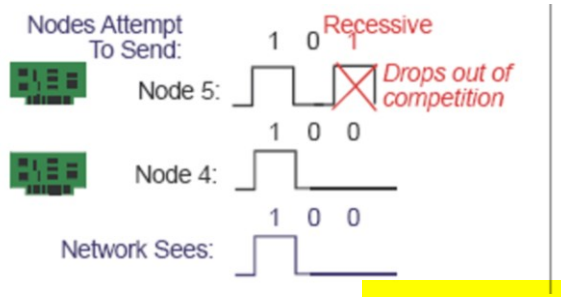


Figure 13.1 CAN layers

Main MAC principles

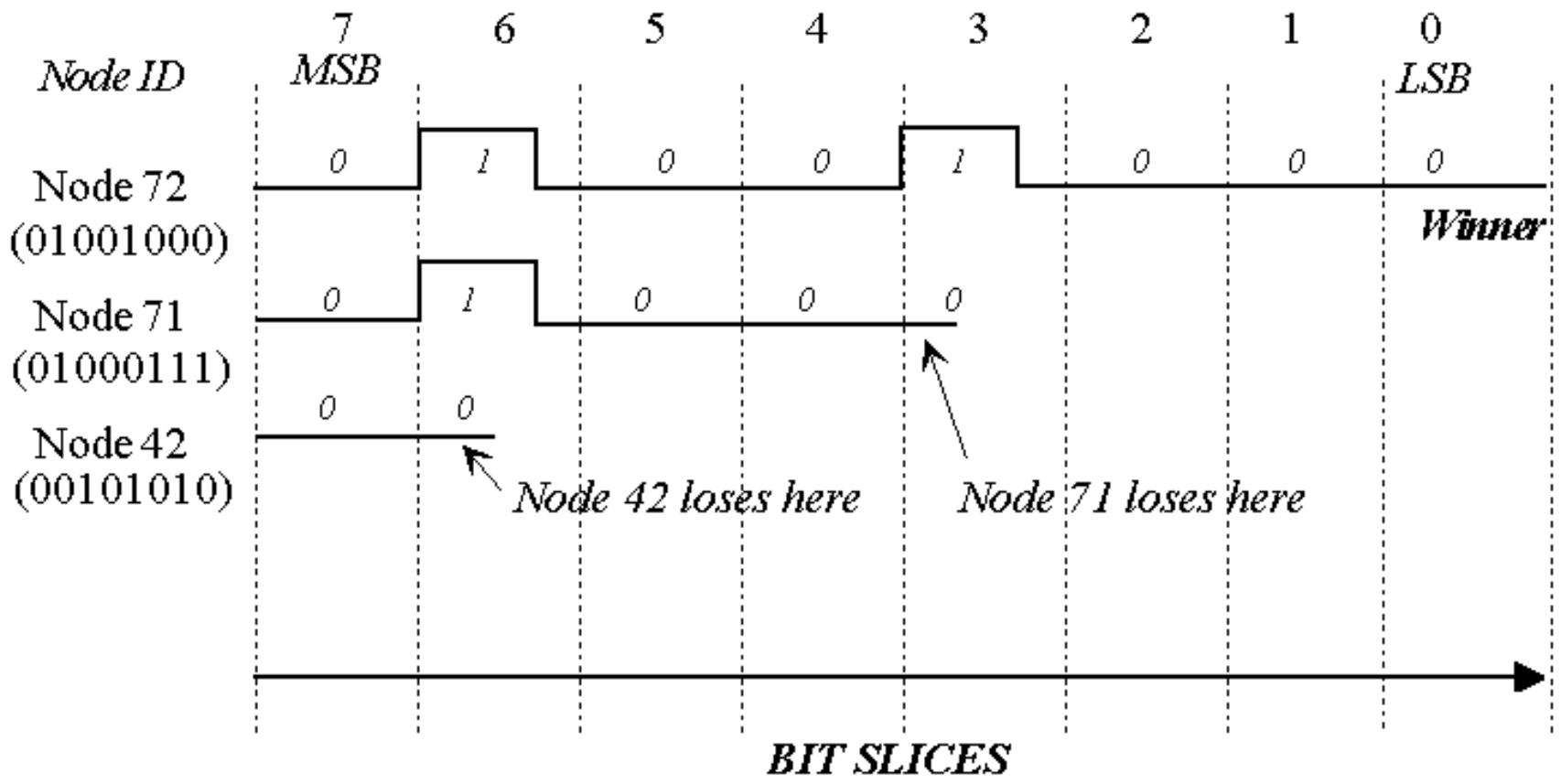
- MAC based on Carrier Sense Multiple Access with Collision Detection (CSMA/CD)
  - Every node on the network must monitor the bus (carrier sense) for a period of no activity before trying to send a message on the bus.
  - Once the bus is idle, every node has equal opportunity to transmit a message.
  - If two nodes happen to transmit simultaneously, **a nondestructive arbitration method** is used to decide which node wins.

Arbitration method principles

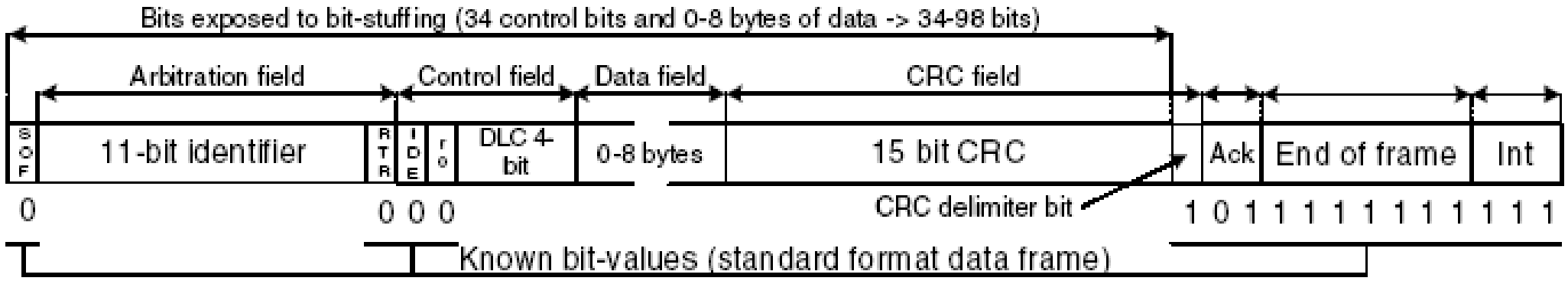


- Each node transmits a message with an **unique identifier** which us used to compite for medium access
- A node drops competition if it detects a dominant bit while transmitting a recessive bit
- Consequently, the nodes with the lowest identifier value drops out of competition, without bandwidth comsumption

Example


7 6 5 4 3 2 1 0
Node ID
MSB
LSB
Node 72
(01001000)
0 1 0 0 1 0 0 0
Winner
Node 71
(01000111)
0 1 0 0 0
Node 42
(00101010)
0 0
Node 42 loses here
Node 71 loses here
BIT SLICES

CAN Frame



- Start of frame (synchronisation)
- Identifier determines priority for access to bus (11-bit or 29-bit)
- Control field (Data length code)
- 0-8 bytes useful data
- 15-bit CRC
- Acknowledgement field
- End of frame marker
- Inter-frame space (3 bits)

CAN Protocol

- CAN is a **multi-master CSMA/CR serial bus**
  - Collision resolution is based on priority
  - CAN physical layer supports two states: “0” dominant, “1” recessive
- Message transmission
  - CAN nodes wait for “bus idle” before starting transmission
  - Synchronise on the SOF bit (“0”)
  - Each node starts to transmit the identifier for its highest priority (lowest identifier value) ready message
  - If a node transmits “1” and sees “0” on the bus, then it stops transmitting (lost arbitration)
  - Node that completes transmission of its identifier continues with remainder of its message (wins arbitration)
  - Unique identifiers ensure all other nodes have backed off

Error Detection

- The CAN protocol requires each node to monitor the CAN bus to find out if the bus value and the transmitted bit value are identical.
- The CRC checksum is used to perform error checking for each message.
- The CAN protocol requires the physical layer to use bit stuffing to avoid long sequence of identical bit value.
- Defective nodes are switched off from the CAN bus

Types of CAN Messages

- Data frame
- Remote frame
- Error frame
- Overload frame

Data Frame

- A data frame consists of seven fields: start-of-frame, arbitration, control, data, CRC, ACK, and end-of-frame.

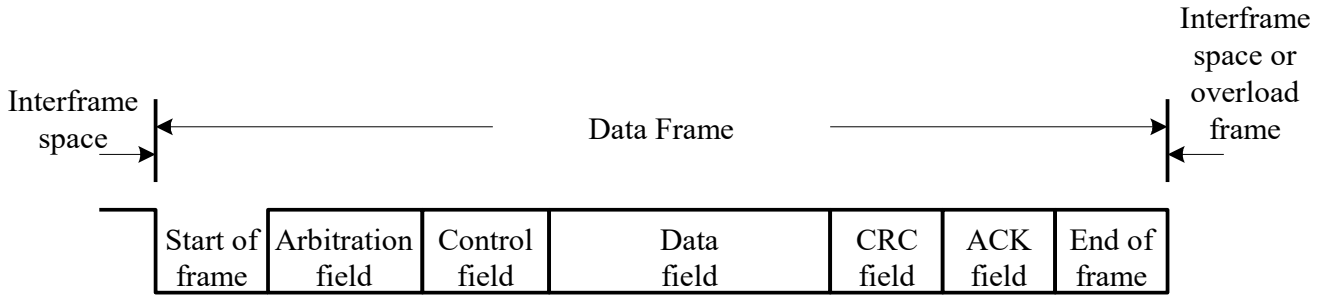


Figure 13.2 CAN Data frame

Start of Frame

- A single dominant bit to mark the beginning of a data frame.
- All nodes have to synchronize to the leading edge caused by this field.

Arbitration Field

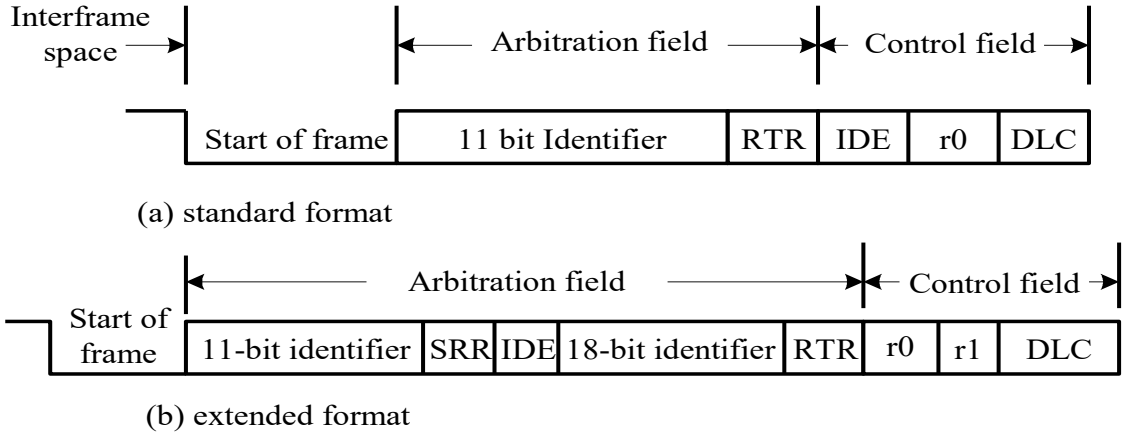


Figure 13.3 Arbitration field

- The identifier of the standard format corresponds to the base ID in the extended format.
- The RTR bit is the Remote Transmission Request and must be 0 in a data frame.
- The SRR bit is the Substitute Remote Request and is recessive.
- The IDE field indicates whether the identifier is extended and should be recessive in the extended format.
- The extended format also contains the 18-bit extended identifier.

Control Field

- The first bit is the IDE bit for the standard format but is used as reserved bit r1 in extended format.
- r0 is a reserved bit.
- DLC3…DLC0 stands for data length and can be from 0000 (0) to 1000 (8).

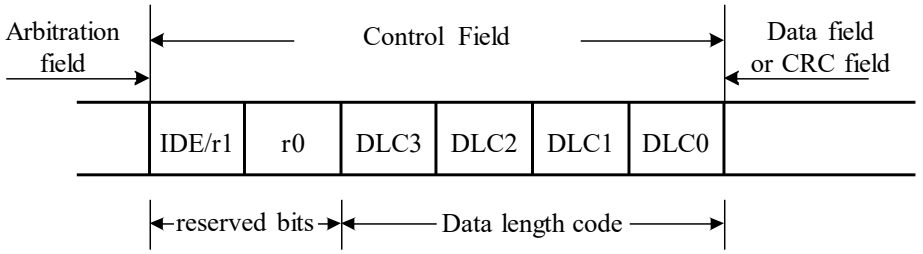


Figure 13.4 Control field

CRC Field

- It contains the 16-bit CRC sequence and a CRC delimiter.
- The CRC delimiter is a single recessive bit.

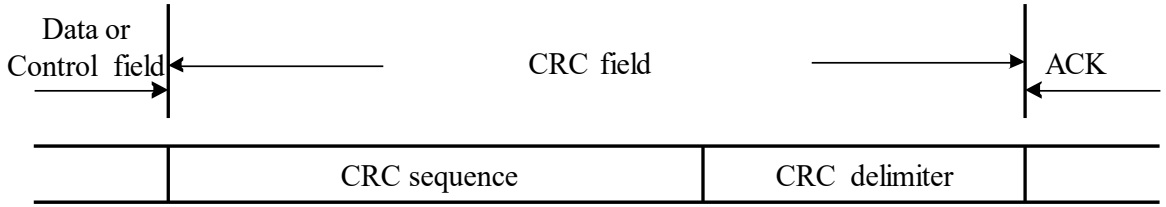


Figure 13.5 CRC field

ACK Field

- Consists of two bits
- The first bit is the **acknowledgement bit**.
  - This bit is set to recessive by the transmitter, but will be reset to dominant if a receiver acknowledges the data frame.
- The second bit is the **ACK delimiter** and is recessive.

Remote Frame

- Used by a node to request other nodes to send certain type of messages
- Has six fields
  - These fields are identical to those of a data frame
  - with the exception that the RTR bit in the arbitration field is **recessive** in the remote frame.

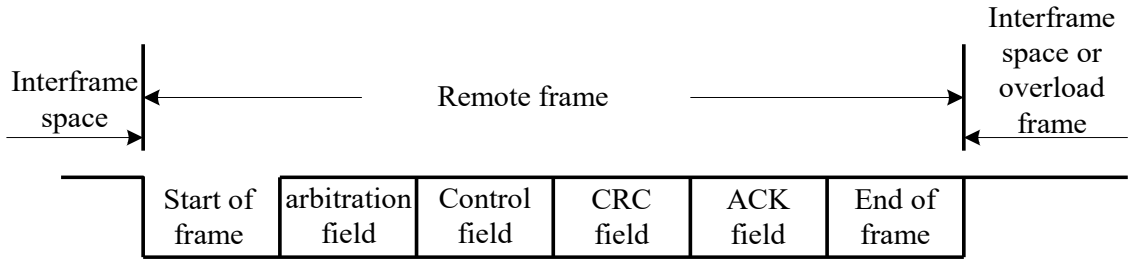


Figure 13.7 Remote frame

Error Frame

- The first field is given by the superposition of error flags contributed from different nodes.
- The second field is the error delimiter.
- Error flag can be either active-error flag or passive-error flag.
  - Active error flag consists of six consecutive dominant bits.
  - Passive error flag consists of six consecutive recessive bits.
- The error delimiter consists of eight recessive bits.

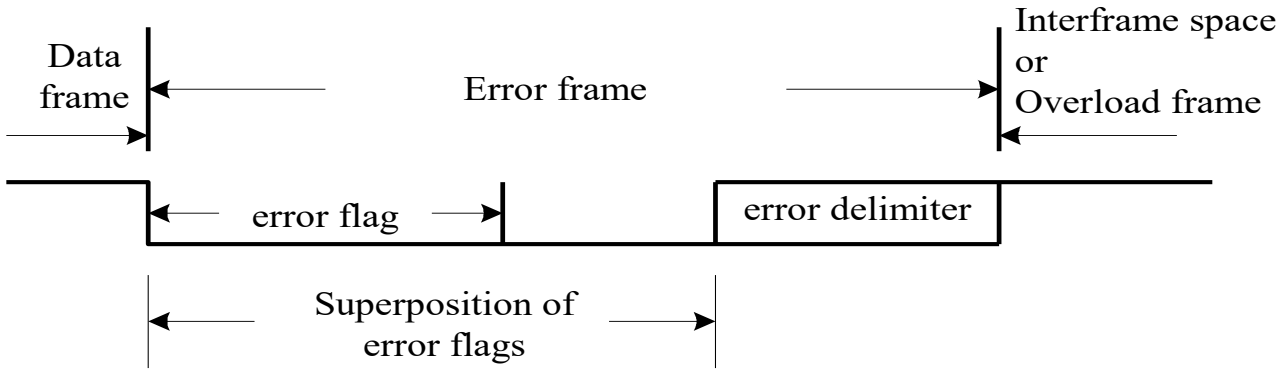


Figure 13.8 Error frame

Overload Frame

- Consists of two bit fields: overload flag and overload delimiter
- Three different overload conditions lead to the transmission of the overload frame:
  - Internal conditions of a receiver require a delay of the next data frame or remote frame.
  - At least one node detects a dominant bit during intermission.
  - A CAN node samples a dominant bit at the eighth bit (i.e., the last bit) of an error delimiter or overload delimiter.
- The overload flag consists of six dominant bits.
- The overload delimiter consists of eight recessive bits.

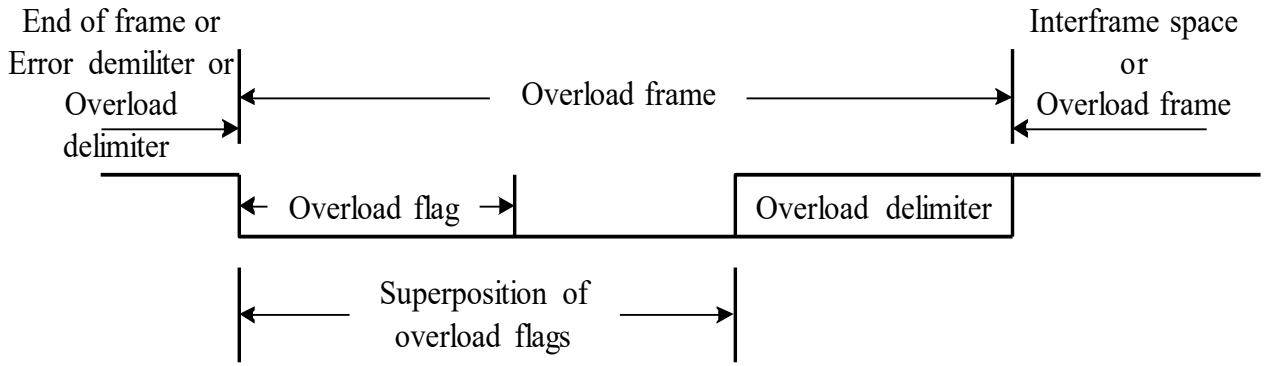


Figure 13.9 Overload frame

Interframe Space

- Data frames and remote frames are separated from preceding frames by the interframe space.
- Overload frames and error frames are not preceded by an interframe space.

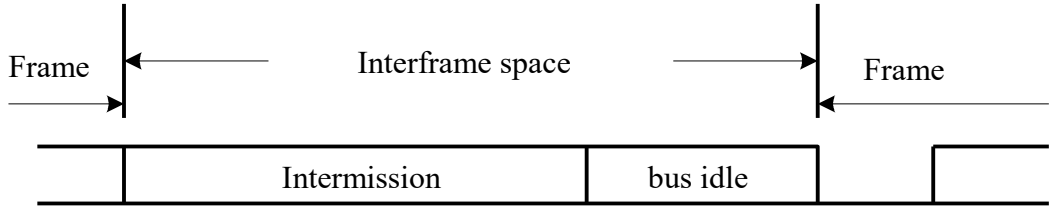


Figure 13.10 Interframe space for non error-passive nodes or receiver of previous message

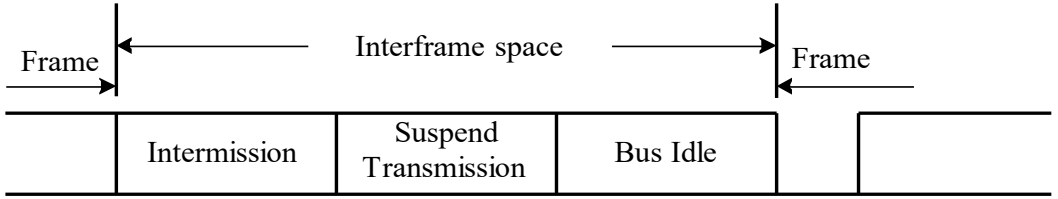


Figure 13.11 Interframe space for error-passive nodes

Interframe Space

- The intermission subfield consists of three recessive bits.
- During intermission no node is allowed to start transmission of the data frame or remote frame.
- The period of bus idle may be of arbitrary length.
- After an error-passive node has transmitted a frame, it sends eight recessive bits following intermission, before starting to transmit a new message or recognizing the bus as idle.

Message Filtering

- A node uses filter (s) to decide whether to work on a specific message.
- Message filtering is applied to the whole identifier.
- A node can optionally implement mask registers that specify which bits in the identifier are examined with the filter.
- If mask registers are implemented, every bit of the mask registers must be programmable.

Bit Stream Encoding

- The frame segments including start-of-frame, arbitration field, control field, data field, and CRC sequence are encoded by **bit stuffing**.
- Whenever a transmitter detects five consecutive bits of identical value in a bit stream to be transmitted, it inserts a complementary bit in the actual transmitted bit stream.
- The remaining bit fields of the data frame or remote frame (CRC delimiter, ACK field and end of frame) are of fixed form and not stuffed.
- The error frame and overload frame are also of fixed form and are not encoded by the method of bit stuffing.
- The bit stream in a message is encoded using the non-return-to-zero (NRZ) method.
- In the non-return-to-zero encoding method, a bit is either recessive or dominant.

Errors

- Error handling
  - CAN recognizes five types of errors.
- Bit error
  - A node that is sending a bit on the bus also monitors the bus.
  - When the bit value monitored is different from the bit value being sent, the node interprets the situation as an error.
  - There are two exceptions to this rule:
    - A node that sends a recessive bit during the stuffed bit-stream of the arbitration field or during the ACK slot detects a dominant bit.
    - A transmitter that sends a passive-error flag detects a dominant bit.

Errors

- Stuff error
  - Six consecutive dominant or six consecutive recessive levels occurs in a message field.
- CRC error
  - CRC sequence in the transmitted message consists of the result of the CRC calculation by the transmitter.
  - The receiver recalculates the CRC sequence using the same method but resulted in a different value. This is detected as a CRC error.

Errors

- Form error
  - Detected when a fixed-form bit field contains one or more illegal bits
- Acknowledgement error
  - Detected whenever the transmitter does not monitor a dominant bit in the ACK slot
- Error Signaling
  - A node that detects an error condition and signals the error by transmitting an error flag
    - An error-active node will transmit an active-error flag.
    - An error-passive node will transmit a passive-error flag.

Fault Confinement

- A node may be in one of the three states: **error-active, error-passive, and bus-off**.
- A CAN node uses an error counter to control the transition among these three states.
- CAN protocol uses 12 rules to control the increment and decrement of the error counter.
- When the error count is less than 128, a node is in error-active state.
- When the error count equals or exceeds 128 but not higher 255, the node is in error-passive state.
- When the error count equals or exceeds 256, the node is in bus off state.
- An error-active node will transmit an active-error frame when detecting an error.
- An error-passive node will transmit a passive-error frame when detecting an error.
- A bus-off node is not allowed to take part in bus communication.

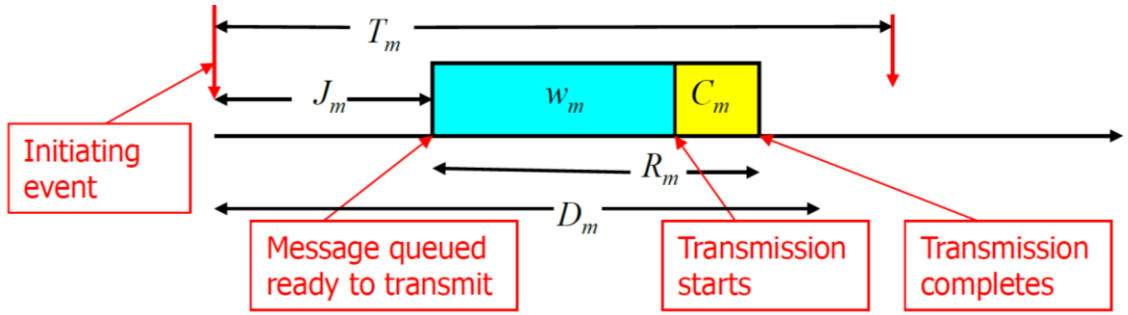
CAN Schedulability analysis

- Objective
  - Verify if all messages being transmitted on a CAN bus meet their deadlines

CAN Schedulability analysis: Introduction

- CAN network scheduling resembles single processor fixed priority non-pre-emptive scheduling
  - Messages compete for access to the bus based on its priority, i.e. on its 11/29 bit identifier
  - Effectively a global queue with transmission in priority order
- Once wins access to the bus (arbitration) it cannot be pre-empted
- First analysis proposed by Ken Tindell during 1993-1995 from earlier work on fixed priority pre-emptive scheduling
  - Calculates worst-case response times of all CAN messages
  - Used to check if all CAN messages meet their deadlines in the worst-case
  - Possible to engineer CAN based systems for timing correctness, rather than “test and hope”

Model



- Each CAN message has a:
  - Unique priority m (identifier)
  - Maximum transmission time C_m
  - Minimum inter-arrival time or period T_m
  - Deadline $D_m <= T_m$
  - Maximum queuing jitter J_m
  - Transmission deadline $E_m = D^m - J_m$
- Compute:
  - Worst-case queuing delay w_m
  - Worst-case response time
    $R_m = w_m + C_m$
  - Compare with transmission deadline
    $R_m \leq E_m$

Maximum transmission Time C_m

- Bit stuffing
  - Bit patterns “000000” and “111111” used to signal errors
  - Transmitter inserts 0s and 1s to avoid 6 consecutive bits of the same polarity in messages
- Consequently, it increases transmission time of a message
  - 11-bit identifiers: $C_m = (55 + 10S_m)^*\tau_{bit}$
  - 29-bit identifiers: $C_m = (80 + 10S_m)^*\tau_{bit}$

Minimum inter-arrival time or period T_m

- This model assumes that a message is ready to be transmitted (queued) with a minimal period (i,e, inter-arrival time) of T_m

- This model supports events that:
  - occur strictly periodically with a period of T_m
  - occur sporadically with a minimum separation of T_m
  - occur only once before the system is reset, in which case T_m is infinite.

Jitter

- Each message is assumed to be queued by:
  - a software task, process or
  - interrupt handler executing on the host microprocessor.
- This task is either invoked by, or polls for, the event and takes a bounded amount of time between 0 and J_m to queue the message ready for transmission.
- J_m is referred to as the queuing jitter of a message and is inherited from the overall response time of the task, including any polling delay.

Deadline

- Each message has a hard deadline D_m,
  - corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving nodes that require it.
  - Tasks on the receiving nodes may place different timing requirements on the data; however in such cases we assume that D_m is the tightest of such time constraints.

Worst-case responde time

- The worst-case response time R_m of a message is defined as the longest time from the initiating event occurring, to the message being received by the nodes that require it.

- A message is said to be schedulable if and only if its worst-case response time is less than or equal to its deadline ($R_m < D_m$).
- The system is schedulable if and only if all of the messages in the system are schedulable.

CAN Schedulability Analysis

- The worst case responde time for a message m is given by:

$$R_m = J_m + W_m + C_m.$$

- J_m is the message jitter
- W_m is the queuing delay, corresponding to the longest time that message m can remain in the CAN controller slot or device driver queue before commencing successful transmission on the bus
- C_m is the longest time a message takes to transmit

CAN Schedulability Analysis: queuing delay

- The queuing delay (W_m) comprises blocking B_m, due to lower priority messages which may be in the process of being transmitted when message m is queued and interference due to higher priority messages which may win arbitration and be transmitted in preference to message m.

$$B_m = \max_{k \in \mathrm{lp}(m)} (C_k),$$

  - where lp(m) is the set of messages with lower priority than m.

CAN Schedulability Analysis: queuing delay

- The worst-case queuing delay for message *m* occurs for some instances of message queued within a priority level-busy period that starts immediately after the longest lower priority message begins transmission.
- This maximal busy period begins with a so-called **critical instant** where message m is queued simultaneously with all higher priority messages, and then each of these messages is subsequently queued again after the shortest possible time intervals.

$$W_m = B_m + \sum_{\forall k \in \mathrm{hp}(m)} \left\lceil \frac{W_m + J_k + \tau_{\mathrm{bit}}}{T_k} \right\rceil C_k,$$

CAN Schedulability Analysis

- Although W_m appears on both sides of the equation, as the right hand side is a monotonic nondecreasing function of W_m, the equation may be solved using the following recurrence relation:

$$W_m^n = B_m + \sum_{\forall k \in \mathrm{hp}(m)} \left\lceil \frac{W_m^{n+1} + J_k + \tau_{\mathrm{bit}}}{T_k} \right\rceil C_k.$$

N+1 está trocado

- A suitable starting value is $W_m^0 = B_m$. The relation iterates until either:
  - $J_m + W_m^{n+1} + C_m > D_m$, in which case the message is not schedulable or
  - $W_m^{n+1} = W_m^n$, in which case the worst-case response time of the first instance of the message in the busy period is given by $J_m + W_m^{n+1} + C_m$.

CAN Schedulability Analysis: example

Stream	Priority	Period	D	C	R
S1	1	2,5 ms	2,5 ms	1 ms	2 ms
S2	2	3,5 ms	3,5 ms	1 ms	3 ms
S3	3	5,0 ms	5,0 ms	1 ms	3 ms

All response times are lower than the deadline, consequently the system is schedulable!

Bibliography

- http://www.aa1car.com/library/can_systems.htm
- Huang Han-Way, The HCS12/9S12: An introduction
- Koopman P, Controller Area Network (CAN) slides
- Stone M, Controller Area Networks lecture slides, Oklahoma State University
- Upender B, Koopman P, Communication protocols for embedded systems, Embedded systems programming, Nov 1994.
- Robert Davis, Real-Time Scheduling and Automotive Networks
- K.W. Tindell, A. Burns, A.J. Wellings, “Calculating Controller Area Network (CAN) Message Response Times”, Control Engineering Practice, Vol 3, No 8, pp1163-1169, 1995. DOI:10.1016/0967-0661(95)00112-8

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Rua Dr. António Bernardino de Almeida, 431

4249-015 Porto, Portugal

—

T(+351) 228 340 500

F (+351) 228 321 159

—

www.isep.ipp.pt

www: moodle.isep.ipp.pt

http://www.isep.ipp.pt/mescc

e-mail: llf@isep.ipp.pt