Goals for this slide set

- Describe the CAN protocol
- Explain how to use the CAN driver API under µC/OS-II
- Apply the response-time analysis to a real-time system comprised of nodes communicating via a CAN bus

Outline

- Introduction to communication in automotive real-time systems
- CAN protocol
- Timing properties
- Schedulability analysis
- CAN under µC/OS-II

Summary of real-time primitives so far

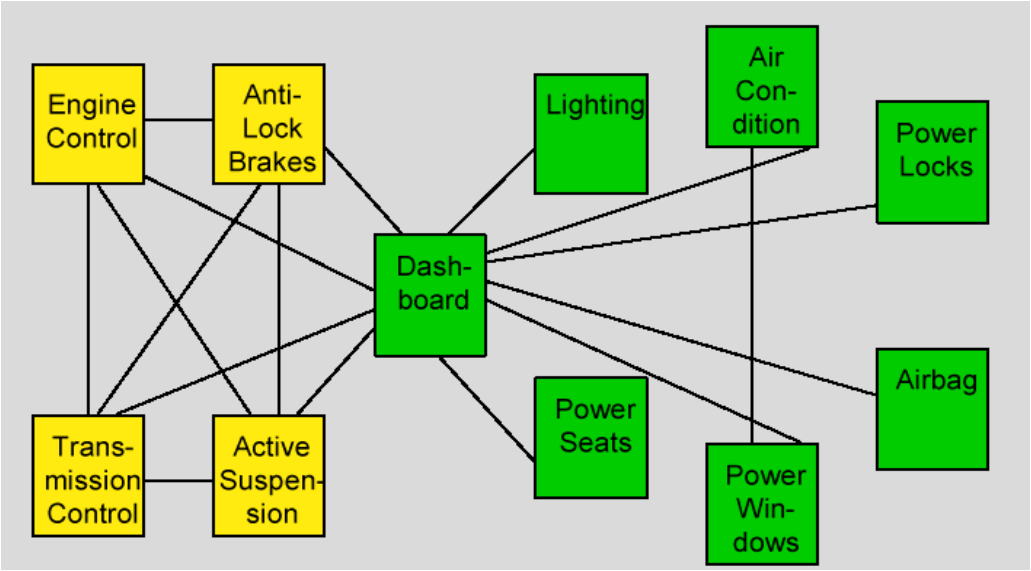
- ✓ Cyclic executive (AFAP, periodic)
  - Goal: several tasks executing on the same ECU
- ✓ Preemptive scheduler
  - Goal: higher priority Task1 preempts a long Task2
- ✓ Disabling/enabling interrupts/scheduler
  - Goal: execute a portion of a task non-preemptively
- ✓ Mutex
  - Goal: Task1 and Task2 share a resource (shorter blocking time than disabling interrupts or scheduler)
- ✓ Semaphore
  - Goal: Task1 triggers Task2 **running on the same ECU**
- Communication
  - Goal: Task1 triggers Task2 **running on a different ECU**

Example: distributed control

- Requires communicating the sensed value with another ECU
- Communication requires a **protocol**
  - Protocol describes the frame (message) format, frame sequence, timing, physical constraints, …
  - Infeasible to have a different protocol for every sensor/ECU
  - Standard protocols used in automotive: **CAN**, FlexRay

RT communication used to look like this…

It used to look like this…

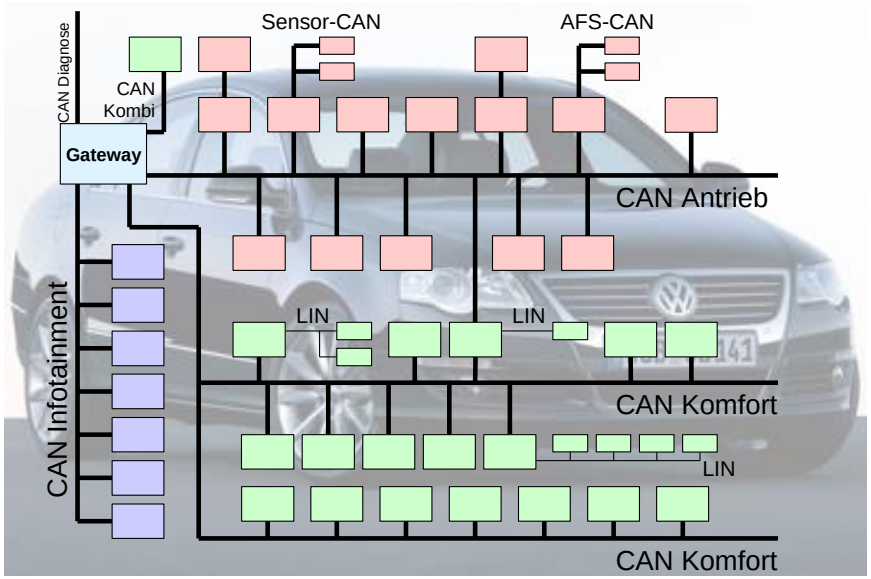


As the number of electronic devices grew

- the wiring gets more “messy”
- the weight of the car increases

Network architecture of a car

- In modern cars, point-to-point wiring is replaced by a common communication bus
- Bus
  - Connects individual ECUs
  - Examples: CAN, FlexRay
- Interconnect between buses
- Benefits:
  - Cost reduction
  - Flexibility



Polling based sensing

- Check for a condition at regular intervals
  - **Busy-waiting**: check continuously
    - E.g. implementation of the ATD driver: check continuously if the status flag in the ATDSTAT register was written (indicating ATD conversion was completed)
  - **Suspension**: check again after a delay
    - E.g. alternative implementation of the ATD driver: delay (and suspend) between consecutive checks of the ATDSTAT status flag

Polling based sensing

- Shortcomings:
  - Wasted processor cycles (which could be otherwise used to do more useful work, or save energy)
    - Note: some devices may provide only means for polling
  - Polling period sets a lower bound on event handling latency
    - Potentially long latency between the occurrence of an event and its handling

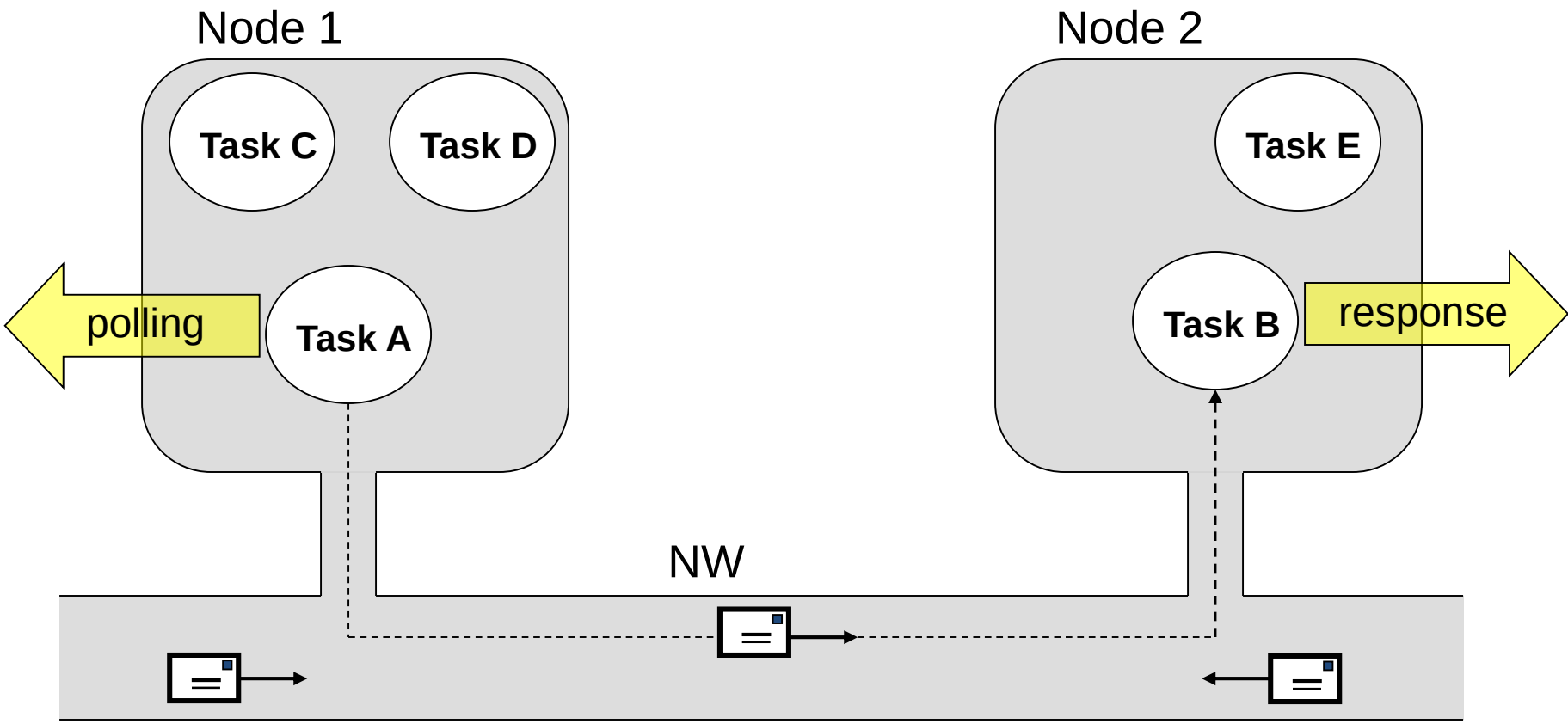
Interrupt based sensing

- Interrupts are hardware signals generated by external hardware indicating that handling is needed
  - Arriving interrupt releases a corresponding ISR
  - May result in the triggering of a task
- Examples:
  - Timer has expired
  - CAN message has arrived

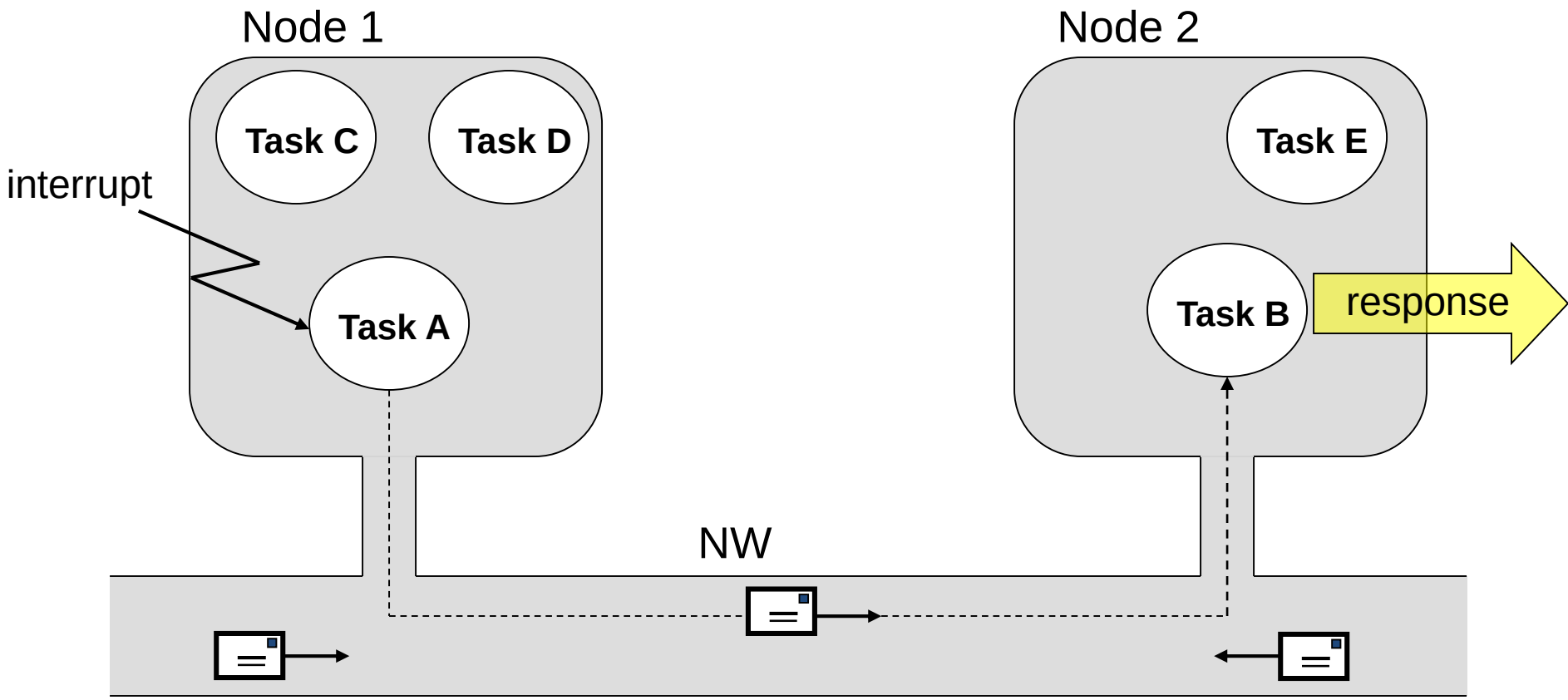
Polling vs. interrupts

- Polling:
  - Task (periodically) checks for a condition
  - **Predictable** time when the condition is checked
  - Time-triggered: due to the timer interrupt
    - i.e. timer interrupts can be used to implement polling
- Interrupts:
  - Task is notified when a condition is satisfied
  - **Unpredictable** time when the condition is satisfied
  - Event-triggered: due to any other interrupt

Time-trigged communication



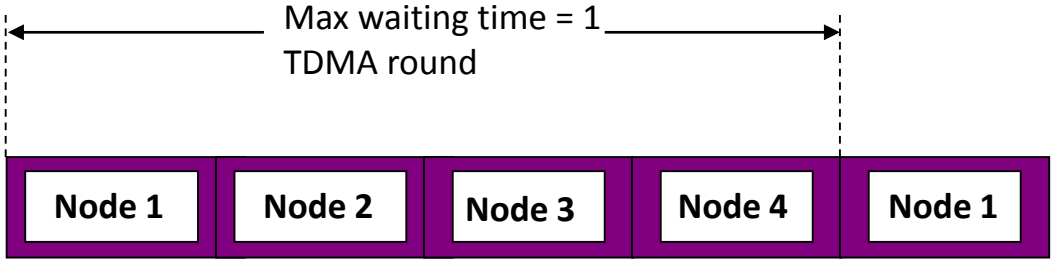
Event-trigged communication



Protocols suitable for real-time communication

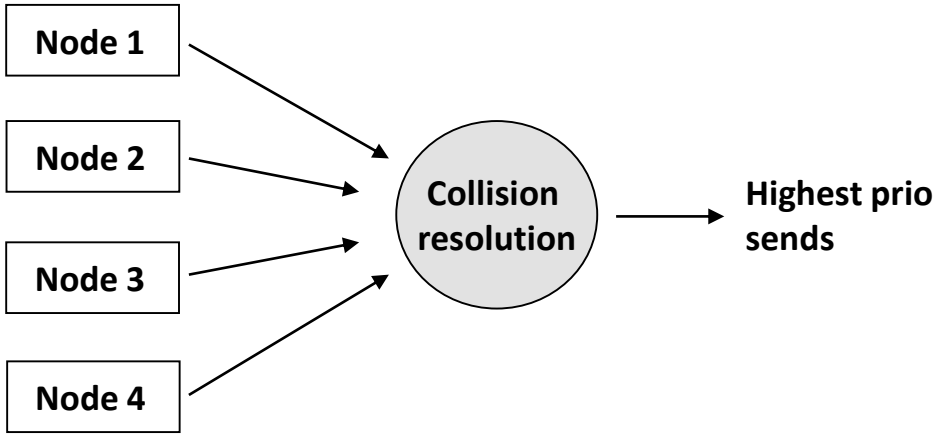
TDMA

- Time-trigged (periodic)
- High testability
- *Example: TTP-protocol*



CSMA/CR

- Priority based (online schedule)
- Flexible
- *Example: CAN-protocol*



Outline

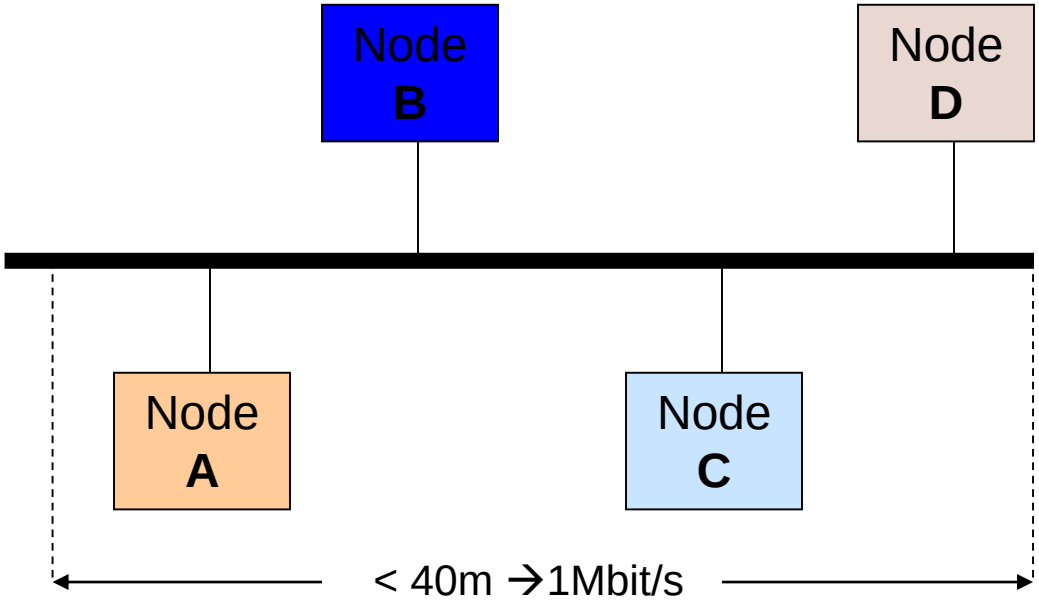
- Introduction to communication in automotive real-time systems
- CAN protocol
- Timing properties
- Schedulability analysis
- CAN under µC/OS-II

CAN – Control Area Network

- Originally developed for automotive industry needs
  - 1983: BOSCH starts CAN development (Intel joins 1985)
  - 1987: First CAN chip
  - 1990: First car with CAN (Mercedes S-class)
  - 1993: ISO standard
- Now used also in factory automation
  - Very common in machinery
  - CAN-controllers developed by Philips, Intel, NEC, Siemens …
- An implementation of CSMA/CR
  - CSMA/CR: Carrier Sense Multiple Access / Collision Resolution
  - Priority based
  - CR is the central mechanism
  - Bitwise arbitration to resolve collisions

Structure and function

- Synchronous serial communication
- A shared medium (cable) with connected nodes
- Each frame (or message) is comprised of a sequence of bits
- Broadcast: transmitted frames can be picked up by all attached nodes
- 1 Mbit/s at 40m bus length
- Behaves as an AND-grind: bus value = AND between all bits on the bus



Frame types

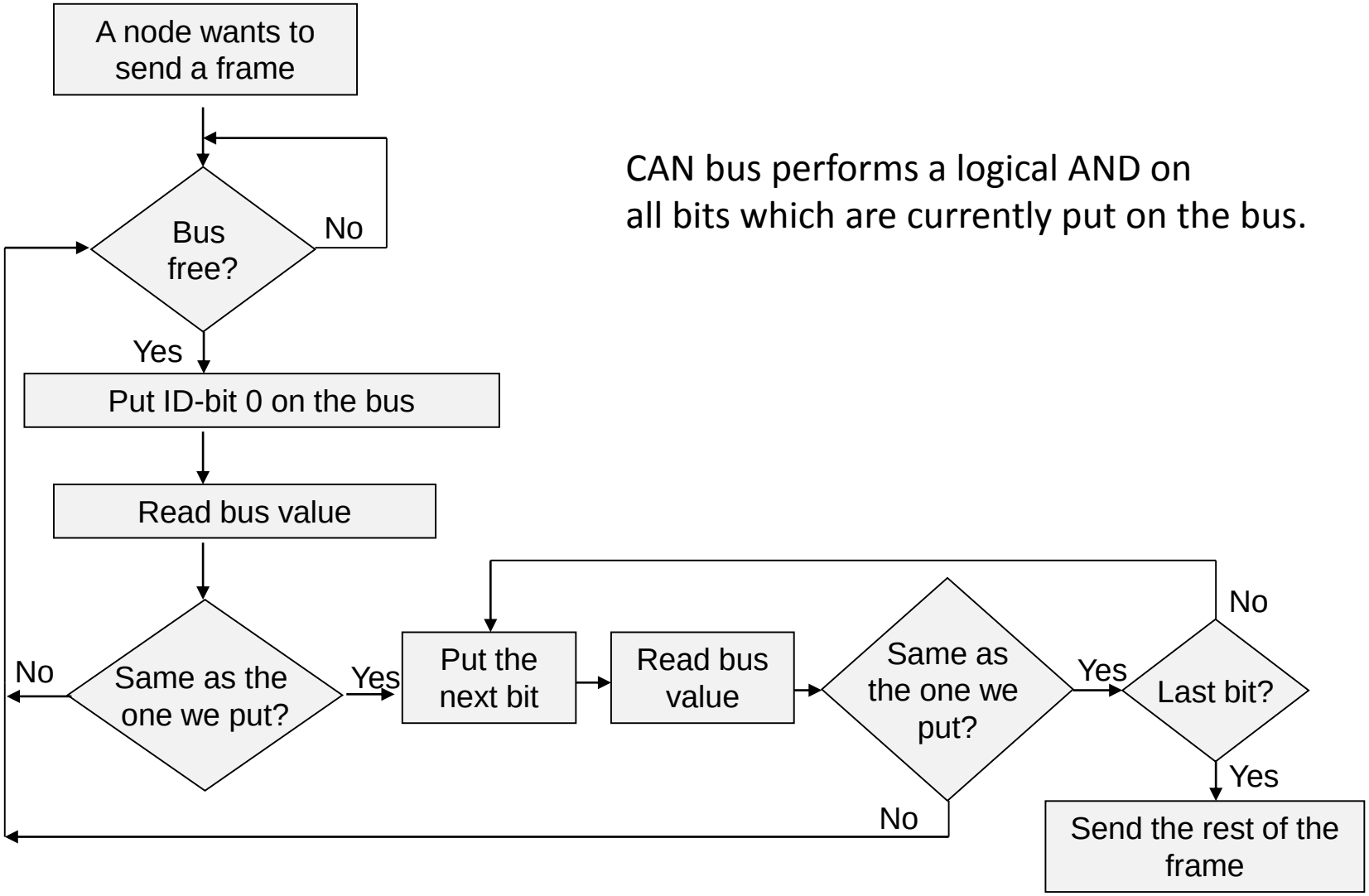
- Data frame
  - Carries data from a transmitter to the receivers
- Error frame
  - Transmitted by any node on detecting a bus error
- Remote frame
  - Transmitted by a node to request the transmission of the data frame with the same identifier
- Overload frame
  - Used to provide an extra delay between the preceding and succeeding data or remote frames, e.g. when the buffer of a receiver is full

CAN data frame

SOF	ID	RTR	Control	Data	CRC	CRC DEL	ACK	ACK DEL	EOF	IFS
1 bit	11 bits	1 bit	6 bits	0-8 bytes	15 bits	1 bit	1 bit	1 bit	7 bits	min 3 bits

SOF - *Start of Frame*, start bit (always 0), used for signaling that a frame will be sent (the bus must be free)
ID - ***Identifier*, identity for the frame and its priority**
RTR - *Remote Transmission Request*
Control - indicates the length of the data field
Data - **message data**
CRC - *Cyclic Redundancy Check*,
CRC DEL - *CRC delimiter* (always 1)
ACK - *Acknowledgement*
ACK DEL - *ACK delimiter* (always 1)
EOF - *End of Frame* (always 1)
IFS - *Inter Frame Space*, resending wait time (always 1)

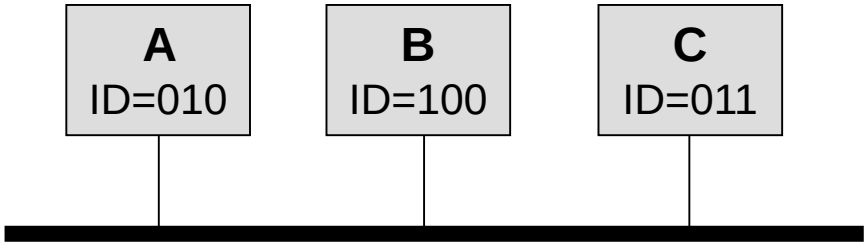
Arbitration mechanism



CAN bus performs a logical AND on all bits which are currently put on the bus.

Arbitration mechanism

Example:
Assume a simplified CAN-system with only three ID-bits and three nodes A, B, C:



000 – highest priority

111 – lowest priority

which gives:

A-high prio, C-middle, B-low

How does the arbitration look like if the nodes are sending simultaneously?

Node	ID	Bit 0	Bit 1	Bit 2
A	010	0	1	0
B	100	1	→ abort! (bit 0 ≠ bus value)	
C	011	0	1	1
Bus value:		0	1	0

→ Send the rest of the frame

→ abort! (bit 2 ≠ bus value)

CAN error frame

- Error detection using the Cyclic Redundancy Check (CRC)
  - The CRC is used by receiving nodes to check for errors in the transmitted frame
  - If the frame is received correctly, the ACK-bit (in the transmitted data or remote frame) is set to 0
- Error signaling
  - The node that detects an error puts instantly an error flag (000000 or 111111) on the bus, followed by an error frame

EF	SEF	ED	IS
6 bits	0..6 bits	8 bits	3 bits

EF - Error Flag
SEF - Superposed Error Flag
ED - Error Delimiter
IS - Interframe Space

Bit stuffing

- We must avoid two bit-patterns that are used for error signaling (i.e. 000000 and 111111)
  - Bit stuffing: sender puts extra bits on strategic places to prevent forbidden bit-patterns:
    - After a sequence of 5 **same** bits, insert an opposite bit
  - Receiver reconstructs the original frame by removing the extra bits
- Example:

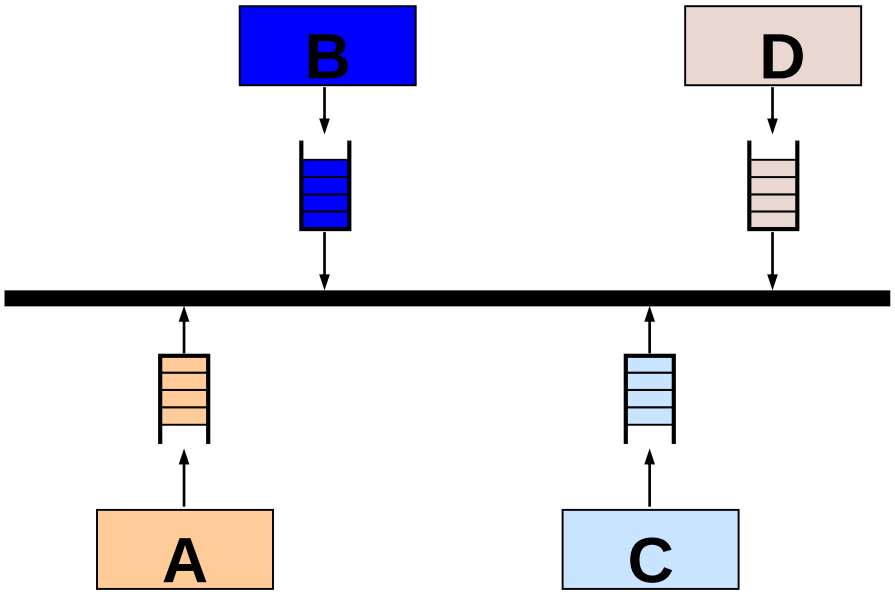
Original frame:	…001010000001 01…
Sender puts extra bits:	…00101000001**1**0101…
Bits sent on the bus:	…00101000001**1**0101…
Receiver removes extra bits:	…001010000001 01…

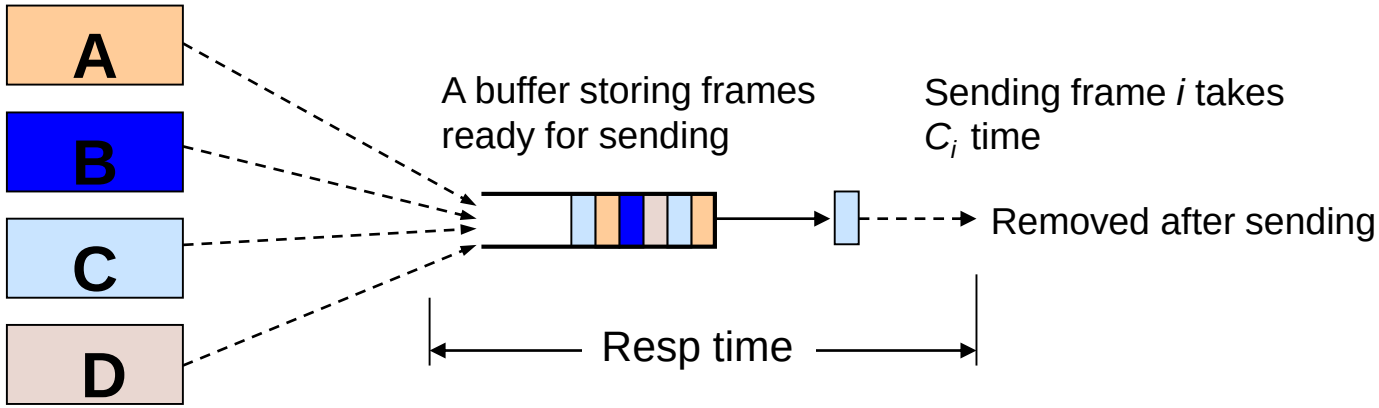
Outline

- Introduction to communication in automotive real-time systems
- CAN protocol
- Timing properties
- Schedulability analysis
- CAN under µC/OS-II

Traffic model

- Abstraction of CAN network:
  - Frames in priority queues
  - No pre-emption





Timing properties

CAN is time deterministic

- The latency can be predicted
- Possible to calculate how long it takes to deliver a frame

How many bits are sent in a CAN-frame?

SOF	ID	RTR	Control	Data	CRC	CRC DEL	ACK	ACK DEL	EOF	IFS
1 bit	11 bits	1 bit	6 bits	0-8 bytes	15 bits	1 bit	1 bit	1 bit	7 bits	min 3 bits

Sum = 47 + 8*n*

(n = nr of data bytes)

Timing properties

- Now we can calculate the total transmission time for a CAN-frame:

$$WC_i = (47 + 8n)\, t_{bit}$$

- Transmission times for 1Mbit/s (i.e. t_{bit} = 1µs):
  - longest: n_{max}= 8: $WC_i = (47 + 8 * 8) 1\mu s = 111\mu s$
  - shortest: n_{min}= 0: $BC_i = 47\mu s$

Timing properties (with bit stuffing)

Do we need to perform bit stuffing on all 47+8*n* bits?

- No. By forbidding some ID values we can avoid bit stuffing in the frame ID. So, only 34 (of 47) control bits are affected.

SOF	ID	RTR	Control	Data	CRC	CRC DEL	ACK	ACK DEL	EOF	IFS
1 bit	11 bits	1 bit	6 bits	0-8 bytes	15 bits	1 bit	1 bit	1 bit	7 bits	min 3 bits

34+8*n* affected bits

In worst-case, 1 extra bit after 5 same bits gives:

$$\text{Nr of extra bits} = \left\lfloor \frac{34+8n}{5} \right\rfloor$$

Timing properties (with bit stuffing)

- Is this correct?
  - Not completely. We might need to stuff even more extra bits:
  - The standard allows both 000000 and 111111 for error signaling.
  - The opposite bit is inserted after **every** sequence of same 5 bits

<u>Example:</u>

Original: 1111 1000 0111 1000 0111 1

After bitstuffing: 1111 1**0**000 0**1**111 1**0**000 0**1**111 1

Extra bit after **5** original bits	Extra bit after **4** original bits	Extra bit after **4** original bits	Extra bit after **4** original bits	..etc…

To avoid forbidden bit patterns we may need to insert 1 extra bit after the first 5 bits and 1 extra bit after each 4 original bits.

Timing properties (bit stuffing)

Hence, the number of extra bits in a CAN-frame is:

$$\left\lfloor \frac{34+8n-1}{4} \right\rfloor$$

Now we can calculate the total transmission time for a CAN-frame:

$$WC_i = \left(47+8n+\left\lfloor \frac{34+8n-1}{4} \right\rfloor\right) t_{bit} = (55+10n)\, t_{bit}$$

Transmission times for 1Mbit/s (i.e. t_{bit} = 1μs):

- ***longest*: n_{max}= 8 and *including* stuff-bits**

$$WC_i = (55+10*8)1\mu s = 135\mu s$$

- ***shortest*: n_{min}= 0 and *without* extra bits:**

$$BC_i = 47\mu s$$

Outline

- Introduction to communication in automotive real-time systems
- CAN protocol
- Timing properties
- Schedulability analysis
- CAN under µC/OS-II

Response time analysis for CAN

- CAN is priority based. Can we use some kind of response time analysis (similar as for tasks)?
  - Yes, with a slight modification. The CAN-bus is non-preemptive, so when a frame has managed to send the first bit, then it will continue sending the rest of the frame (without fear of being preempted by high priority frames)

Response time analysis for CAN

- Response time for frame with priority/id i (pessimistic):

$$w_i = t_{bit} + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i}{T_j} \right\rceil WC_j$$

$$WR_i = w_i + WC_i - t_{bit}$$

where the blocking time for a frame is given by:

$$B_i = \max_{\forall k \in lp(i)} WC_k \quad \pounds 135 t_{bit}$$

hp(i) = high priority frames (that can delay the first bit)
lp(i) = low priority frames (that can block the first bit)