

CONTROLLER AREA NETWORK JOURNAL TIL CAN ØVELSE

E4ISD2

Janus Bo Andersen (JA67494)

Maj 2020

CAN OVERBLIK

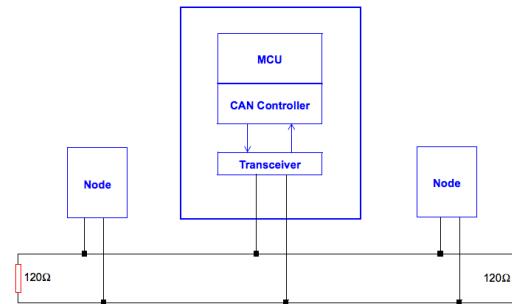
Punkt

Hvad er det?

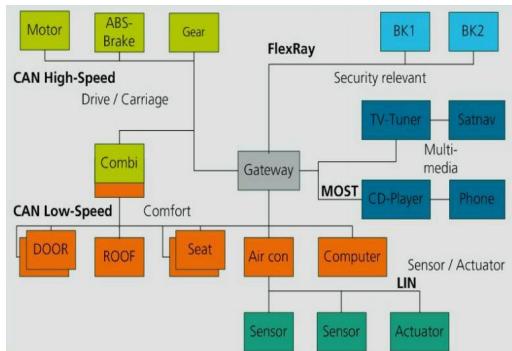
Detaljer

- CAN er en ISO-standard for arkitektur og protokol til robust, semi-hurtig, prioritert og multicast/besked-baseret kommunikation mellem flere noder på en multi-master bus.
 - Robust = spec.'et til at fungere i miljø med støj:
 - To-ledning-bus med differentieret signal giver højere støjimmunitet end single-ended.
 - Indbygget error detection vha. CRC.
 - Semi-hurtig:
 - Hastighed varier med transmissionsafstand, op til 1 Mbit/s under 25 m [1, s. 14].
 - Prioriteret:
 - Anvendelse af arbitrering baseret på node ID implementerer, at laveste ID altid får forrang i kommunikation. Men en besked allerede under afsendelse kan ikke pre-emptes [1, s. 20-21].
 - Garanterer maksimal latency (hård realtid - men nok kun for laveste ID ☺) [1, s. 7].
 - Multicast/besked-baseret kommunikation:
 - Alle forbundne enheder lytter med.
 - Data sendes ud på bussen i frames, hver frame indeholder op til 8 bytes data.
 - Flere noder (se fig. 1):
 - Microcontrollers eller andre enheder (sensorer, aktuatorer) tilkobles bussen.
 - Multi-master bus (se fig. 1):
 - Bus uden en host, så flere noder kan tage ansvar for at være "masters", dvs. initiere beskeder ud på bussen.
- Oprindeligt til automotive, hvor mange sub-systemer er distribuerede dele af et samlet system i en begrænset afstand til hinanden (se fig. 2). En enkelt bus minimerer lange, komplicerede ledningstræk i bilen, giver single-point adgang til alle tilkoblede enheder, og med en bus af twisted-pair gives endnu bedre støjimmunitet.
- Anvendeligt i mange lignende situationer, hvor mechatroniske / embeddede, distribuerede enheder inden for en kort afstand skal kommunikere, evt. i et støjfyldt miljø (se fig. 3)
- Andre områder er fx robotsystemer med mange microcontrollers, sensorer og aktuatorer til hver robot-del. Military og aviation: Oplagt til robust kommunikation i fx fly, våbensystemer eller lign.

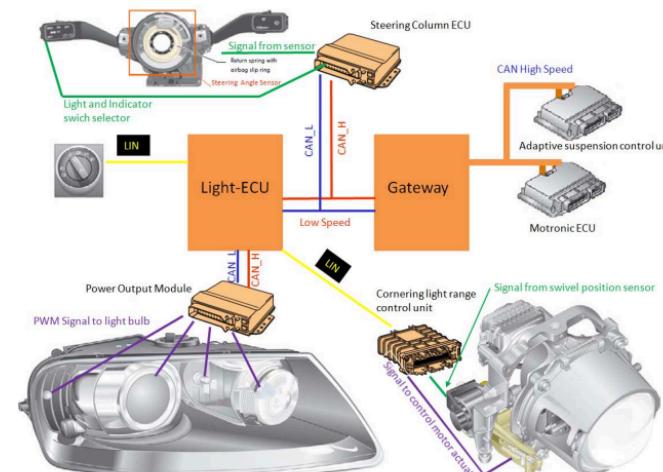
Anvendelser af
CAN i komplekse
distribuerede
systemer



Figur 1. Topologi til et simpelt system med CAN-bus [2]



Figur 2. Topologi til et CAN-bus-system i en bil [3]



Figur 3. Eksempel på anvendelse. Styrevinkel på rattet og lysindstillinger fra instrumentbræt er via CAN-bus forbundet til en ECU, som igen via CAN forbinder til sub-systemet til styring af forlygternes lysstyrke og vinkel [3]

[1]: Marco Di Natale. Understanding and using the Controller Area Network. 2008.

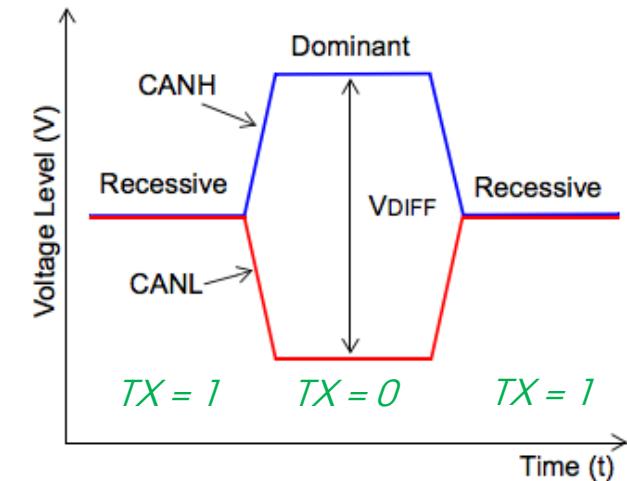
[2]: Microchip. A CAN Physical Layer Discussion. 2002. URL: <http://ww1.microchip.com/downloads/en/appnotes/00228a.pdf>

[3]: R Steinhilper, S. Freiberger & A. R. Nagel. Understanding the communication between automotive mechatronics and electronics for remanufacturing purposes. 2012.

BUS-SIGNALER OG INTERFACES

Differentielt CAN-bus signal

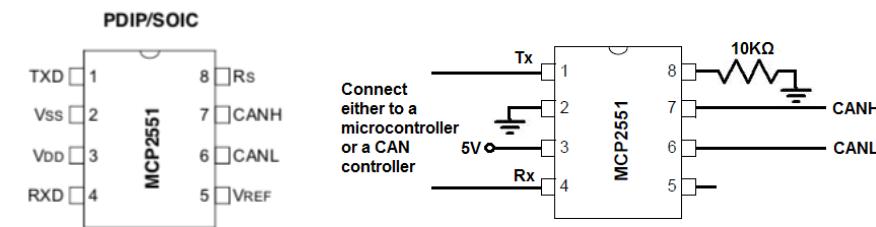
- TX = 1 (logisk høj) implementeres med "recessive" bit-tilstand på bussen (se fig. 1).
- TX = 0 (logisk lav) implementeres med "dominant" bit-tilstand.
 - -> Dominante bits på bussen vil altid "vinde" over recessive.
 - -> Arbitrering: laveste ID får altid forrang.



Figur 1. Bit-tilstande på bussen [2]

Interfaces fra controller til bus

- Transceiver er bindeleddet mellem controller og bus (se fig. 2), og driver spændingsniveauerne på bussen.
- Single-ended TX/RX fra/til controller interfacere med transceiver.
- Transceiver interfacere med bus vha. CANH og CANL.



Figur 2. Interfacing via transceiver (MCP2551) [4] [5]

[2]: Microchip. A CAN Physical Layer Discussion. 2002. URL: <http://ww1.microchip.com/downloads/en/appnotes/00228a.pdf>

[4]: Microchip. High-Speed CAN Transceiver. 2007. URL: <http://ww1.microchip.com/downloads/en/devicedoc/21667e.pdf>

[5]: Learning about electronics. How to Build a CAN Transceiver Circuit with an MCP2551 Chip. URL: <http://www.learningaboutelectronics.com/Articles/MCP2551-CAN-transceiver-circuit.php>

AFSENDELSE AF BESKEDER OG INDHOLD I EN DATAFRAME

CAN er asynkront, så hver node skal holde styr på timing / frekvens (op til 1 Mbit/s).

Hver node afsender beskeder uafhængigt af andre noders tilstand (besked-baseret protokol):

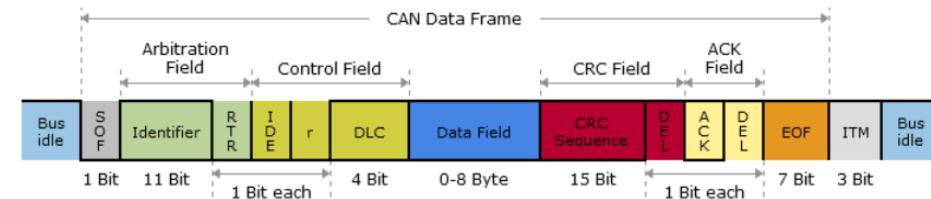
- Flere noder sender samtidig -> arbitrering vha. ID (laveste ID er dominant), men beskeder under afsendelse kan ikke pre-emptes.
- Beskedtyper kan være [1, s. 17]:
 - Data Frames -> Afsende data.
 - Remote Frames (RTR) -> Forespørge data.
 - Error Frames og Overload Frames (flag) -> Rapporterer detekterede fejl/overloading.

En CAN dataframe indeholder (se fig. 1):

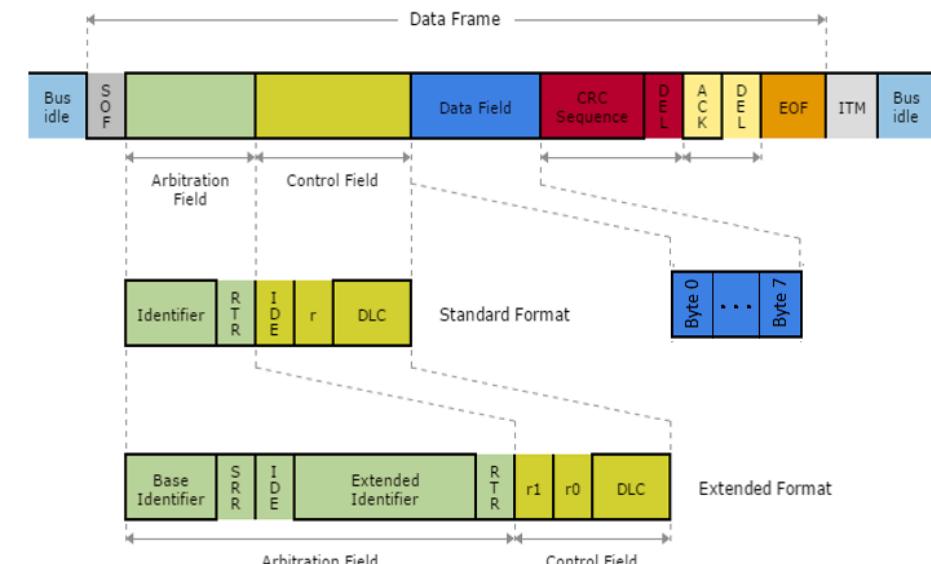
- **SOF:** Start of frame, 1 dominant bit.
- **Identifier:** Unikt node ID på CAN-bussen.
- **RTR:** Kun til Remote Transmission Request (Remote Frames).
- **IDE, r:** Recessiv markerer ext. ID (29-bits)ift. std. 11-bits ID. "r" er reserveret til kombination med IDE.
 - Se fig. 1b for sammenligning af formater med std. ID og extended ID.
- **DLC:** Længde på dataindhold (0-8).
- **Data:** Op til 8 bytes data.
- **CRC, DEL:** Cyclic redundancy check til fejldetektion + delimiter.
- **ACK, DEL:** Til acknowledgement af modtaget besked (modt. på bussen sender dominant) + delim.
- **EOF:** End of frame: 7 recessive bits.

Øvrigt:

- Bit-stuffing: Sekvens af 5 bits med samme polaritet -> stuffing 1 bit med modsat polaritet. Dataframes får variabel længde (antal bits, der "stuffles" ind).
- En dataframe efterfølges af Interframe spacing (ITM) på bussen: 3 recessive bits.
- En dataframe kan fx implementeres som i RTOS journal (se fig. 2).



Figur 1a. CAN datatransmission og CAN dataframe [3]



Figur 1b. Standard og extended (CAN 2.0A)

```
25 /* The CAN datagram type */
26 typedef struct {
27     unsigned int ID;           // holds 11/29 bit ID
28     unsigned char ID_ext;      // set to 1 if ID is extended -29 bit, else 0
29     unsigned char DLC;         // how many bytes of payload
30     unsigned char data[8];     // the data payload
31 } can_tlg_t;
```

Figur 2. CAN datastruktur (fra egen RTOS-journal)

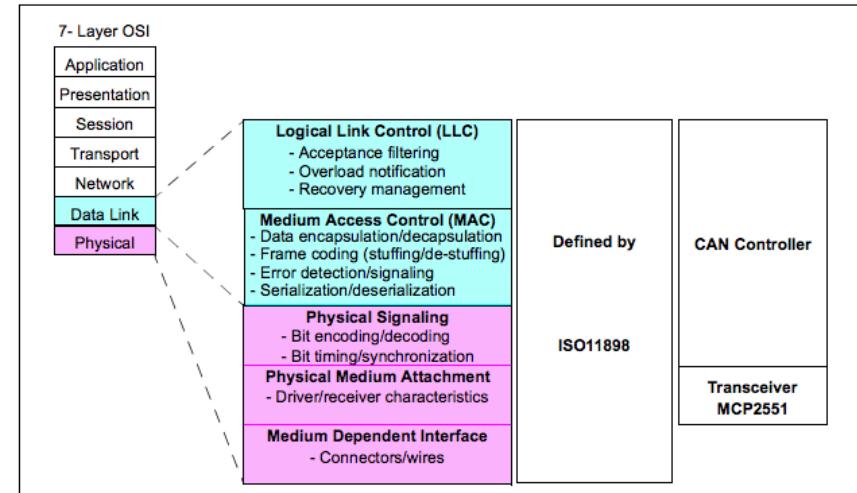
CAN OG OSI-MODELLEN RÅ CAN OG SOCKETCAN

CAN implementerer de to nederste lag i OSI-modellen (se fig. 1 til højre)

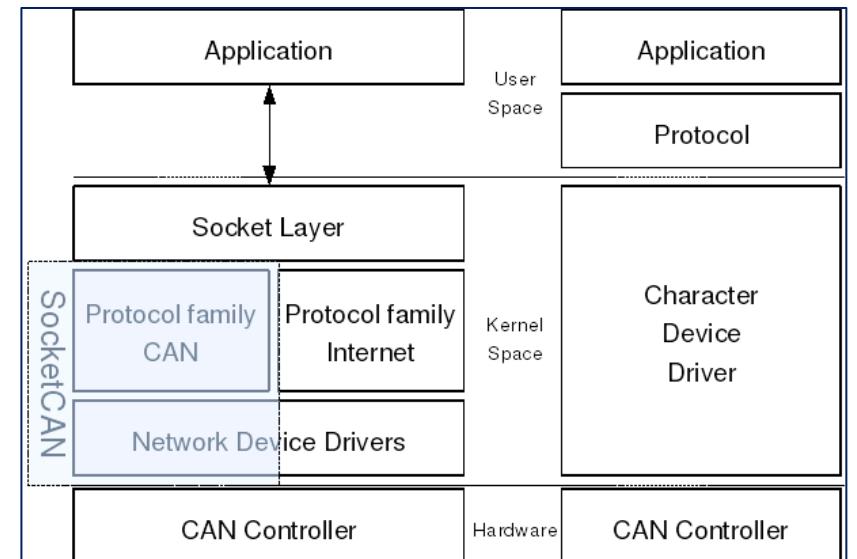
- CAN-controller (på PocketBeagle) implementerer det meste af data link layer.
- CAN-transceiver (PHY'en, MCP2551) implementerer forbindelse til det fysiske medium. Oversætter single-ended TX/RX til differentiel CAN_H, CAN_L.
- Bus-ledninger og terminering skal implementeres separat som defineret i standarden.

SocketCAN (se fig. 2) implementerer yderligere transport og netværkslag:

- CAN kan så benyttes som et hvert andet netværksinterface i Linux/Unix.
- Et CAN-device bliver en byte-strøm, som flere programmer kan læse fra / skrive til samtidig.



Figur 1. CAN og OSI-modellen [2]



Figur 2. SocketCAN-modellen [6]

[2]: Microchip. A CAN Physical Layer Discussion. 2002. Tilgængelig online: <http://www.microchip.com/downloads/en/appnotes/00228a.pdf>

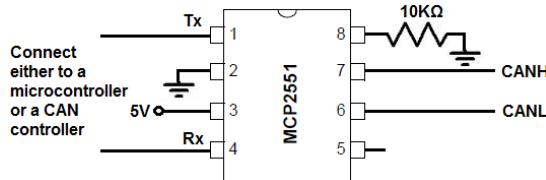
[6]: Wikipedia.org. SocketCAN. 2020. Tilgængelig online: <https://en.wikipedia.org/wiki/SocketCAN>

CAN-BUS-SYSTEM MED TO NODES VIA POCKETBEAGLE CONTROLLERS

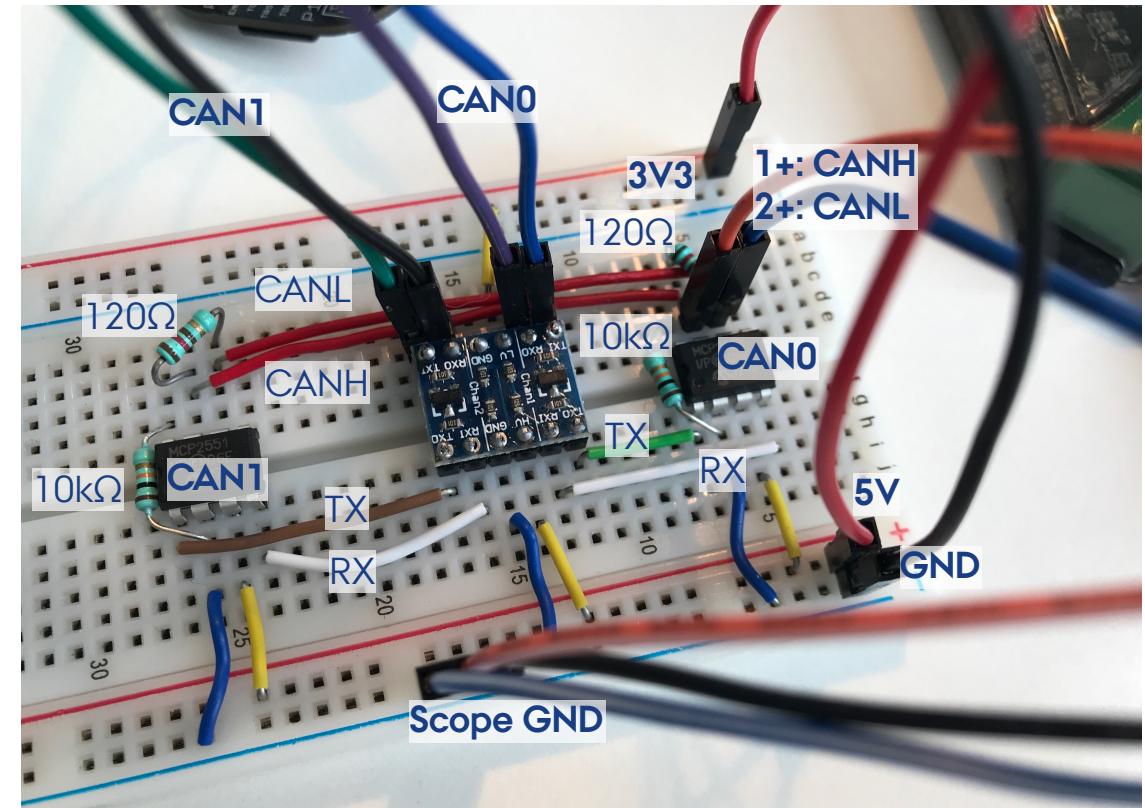
Systemet er sat op som på fig. 1

- Konvertering mellem PocketBeagles 3,3V logik TX/RX og transceiverens 5V-logik sker med en logic level converter (se midten af fig. 1).
- Bussen er termineret i hver ende med $120\ \Omega$.
- Slew Rate (SR) er sat med $10\ k\Omega$, hvilket giver ca. $24\ V/\mu s$, hvilket er relativt hurtige transitioner [4, s. 4].
- Se appendiks for billede inkl. PocketBeagle.

Forbindelser til MCP2551 er som vist i fig. 2



Figur 2. Interfacing via transceiver (MCP2551) [5]



Figur 7. Implementering af simpelt CAN-baseret system

PINMUX OG SOCKETCAN

CAN0 controller:

- TX: P1.26
- RX: P1.28



Figur 1. Pins til hhv. CAN0 og CAN1

CAN1 controller:

- TX: P2.11
- RX: P2.9

```
setup-can.sh
1  # Config CAN0
  1 #RX
  2 config-pin -a P1.28 can
  3 #TX
  4 config-pin -a P1.26 can
  5
  6 # Config CAN1
  7 #RX
  8 config-pin -a P2.9 can
  9 #TX
 10 config-pin -a P2.11 can
```

Pinmux: config-pin (fig. 2)

Netværk vha. SocketCAN (can-utils) (fig. 3)

Konklusion: Setup virker (fig. 4)



```
start-can.sh
1  # Run this script with sudo
  1 # Starts CAN0 and CAN1 with 50kbit/s
  2 ip link set can0 type can bitrate 50000
  3 ip link set can1 type can bitrate 50000
  4 ip link set can0 up
  5 ip link set can1 up
```

Figur 3. Script til start af CAN med 50kbit/s (skal køres med sudo)

Afsend:

```
janus@beaglebone:~/projects/E4ISD2/CAN$ cansend can0 014#11.22.33.
44.55.66.77.77
janus@beaglebone:~/projects/E4ISD2/CAN$ cansend can0 014#11.22.33.
44.55.66.77.77
```

Modtag:

```
janus@beaglebone:~/projects/E4ISD2/CAN$ candump can1
can1 014 [8] 11 22 33 44 55 66 77 77
can1 014 [8] 11 22 33 44 55 66 77 77
can1 014 [8] 11 22 33 44 55 66 77 77
```

Figur 4. Test cansend og candump

TEST

Opsætning:

- Sender på CAN0
- Lytter på CAN1
- 50 kbit/s -> **1 bit = 20 us**

Testbesked til afkodning:

- ID: 0x014
 - 0b00000010100 (11-bits)
 - Bemærk ID resulterer i bit-stuffing
- Payload: 0x01

Oscilloskop:

- VDIFF=CANH-CANL
- VDIFF høj \Leftrightarrow dominant
- VDIFF lav \Leftrightarrow Recessiv

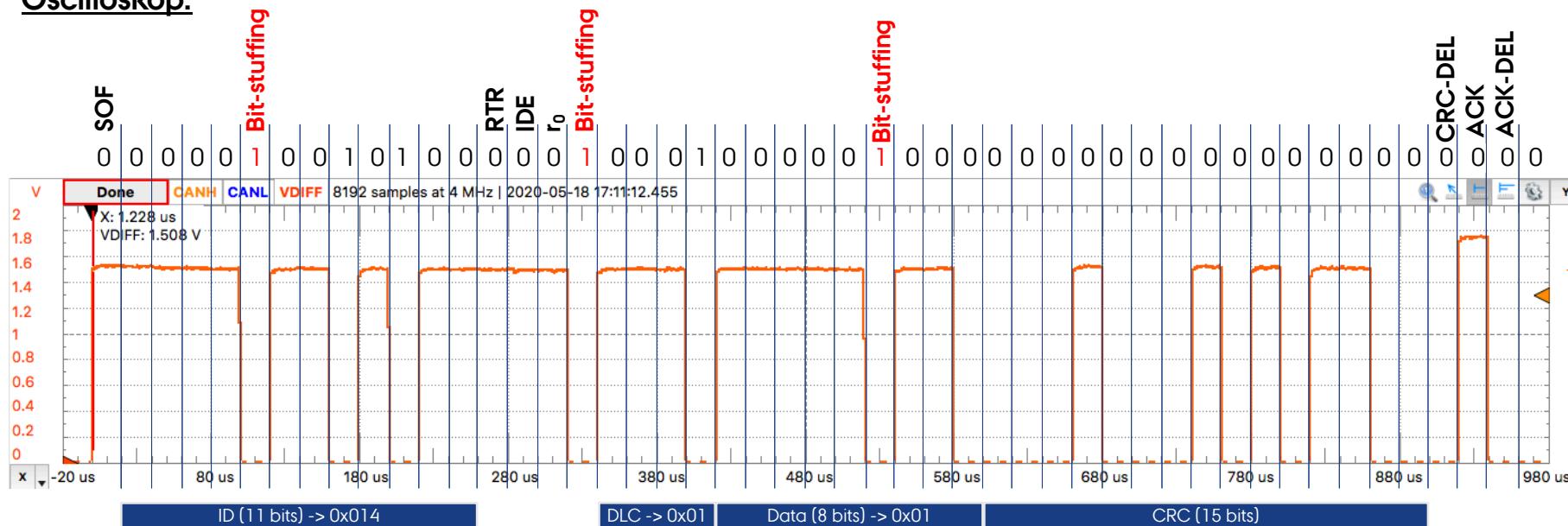
Afsend:

```
janus@beaglebone:~/projects/E4ISD2/CAN$ cansend can0 014#01
```

Modtag:

```
can1 014 [1] 01
```

Oscilloskop:



Konklusion: Beskeden kan afkodes korrekt!

KONKLUSION (1)

CAN har mange fordele til de tiltænkte anvendelser: Robusthed/støjimmunitet, fornuftig hastighed, prioriteret kommunikation, alt sammen i en relativt simpel/enkel protokol.

- Godt til distribuerede, embeddede systemer, med wired enheder inden for kort afstand.

Ulemperne er dog bl.a.:

- Begrænsede dataframes:
 - Maks. 8 bytes per frame og maks. 1 MB/s over bussen kan være *for lidt* til avancerede systemer. Fx til benyttelse af videofeed til AI/ML-baseret navigation/objektgenkendelse (tænk selvkørende biler, droner, robotter og lignende).
- Multicast og faste ID'er (fixed priority) – mangler "scheduling" på bussen:
 - Højere ID'er bliver altid "crowded out" af lavere ID'er,
 - Ingen congestion avoidance,
 - => I principippet kan der ske noget tilsvarende "priority inversion" i realtids-systemer:
 - Lav-ID node venter på data fra en højere ID-node, der bliver crowded out.
 - Ingen mulighed for "priority inheritance".
 - Ingen indbygget "connection eller session" mellem devices (skal implementeres i et højere OSI-lag)
 - => Anvendelse af bussen er ikke efficient til point-to-point komm. (bruger meget tid på arbitration, ID'er, ACK, osv.).

KONKLUSION (2)

I denne øvelse er:

- CANs karakteristika og anvendelser kort opsummeret.
- Et simpelt system med CAN-bus og to noder implementeret vha. to MCP2551 transceivers og de to CAN-controllers på PocketBeagle.
- Vist hvorledes can-utils i Linux kan benyttes til at styre CAN-controllers og foretage kommunikation ud på CAN-bussen.
- Demonstreret hvordan CAN-kommunikation kan debugges vha. oscilloskop.

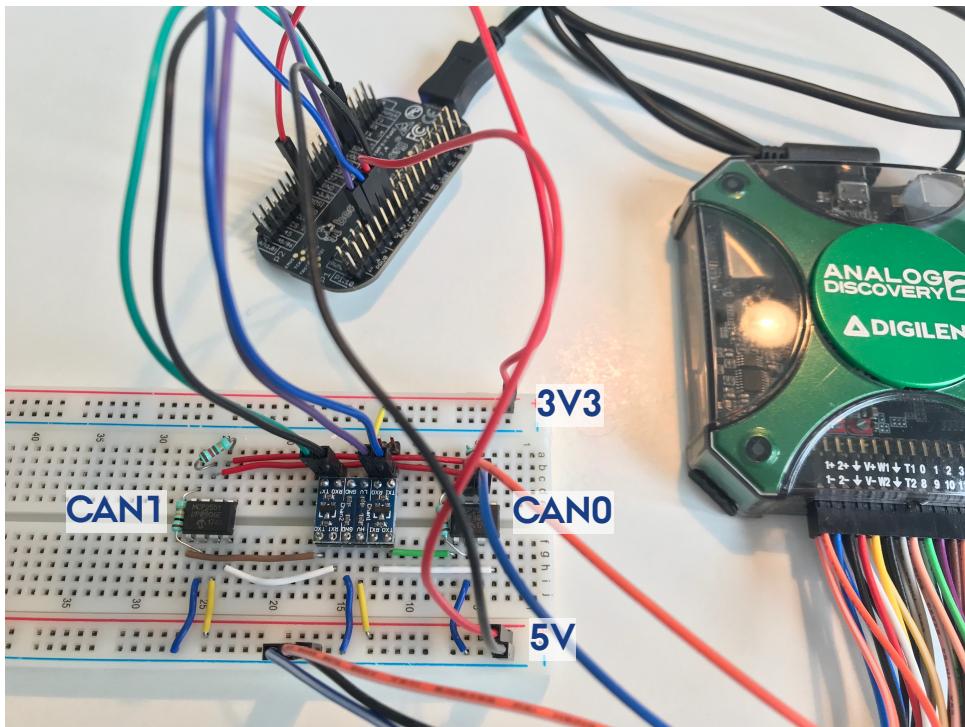
Muligt fremtidigt arbejde:

- Benytte socketCAN i et meningsfyldt C/C++ program til kommunikation mellem distribuerede enheder på en CAN-bus.



AARHUS
UNIVERSITET

APPENDIKS



PocketBeagle Expansion Headers (Rev A2a)

