NTNU | Norwegian University of Science and Technology

TTK4155 INDUSTRIAL AND EMBEDDED COMPUTER SYSTEMS DESIGN

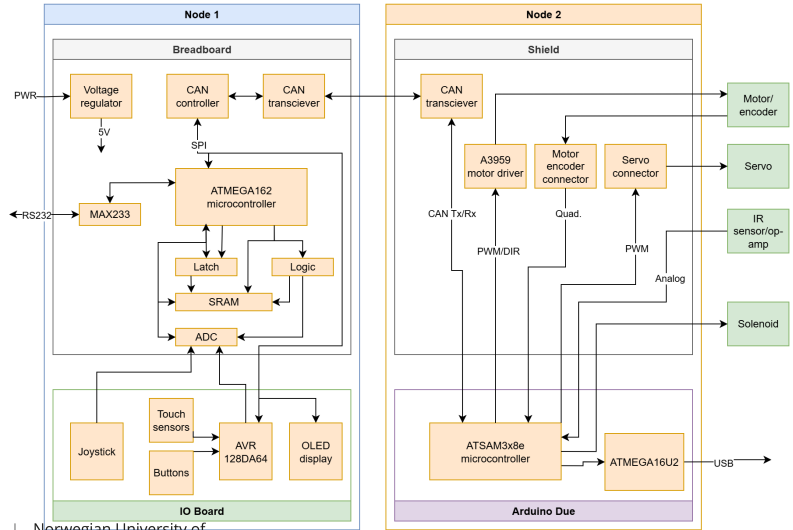
Lab Lecture 3 - Autumn 2025

Kristian Blom

Lecture Overview

- Exercise 5: SPI pt2 and CAN controller
  - Introduce a third SPI slave, the CAN controller
  - Write a driver for the CAN controller
  - Test the MCP2515 (CAN controller) in loopback mode
- Exercise 6: Getting started with ARM cortex-M3 and communication between nodes
  - Connect CAN transceiver (MCP2551) to the CAN controller
  - Create a CAN communication driver (send/receive functions)
- Bonus: Software modularisation and circuit logic

Node communication

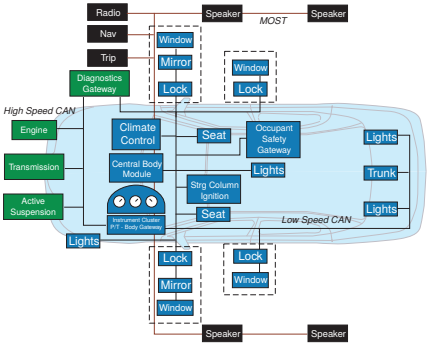


Serial Peripheral Interface (SPI)

- Any questions?

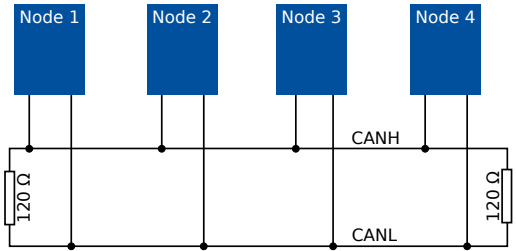
Controller Area Network (CAN)

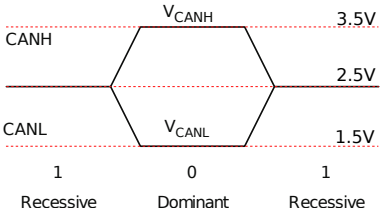
- Controller Area Network
- Vehicle bus = Noise resilient, delivery assurance etc.
- Multi-master broadcast bus protocol
- ID based ‘addressing’
- Arbitration without delay
- Limited datagram size
- Up to 1 Mbit/s
- For medium distances, <40m
- Easy to use, but hard to set up



CAN Physical Layer

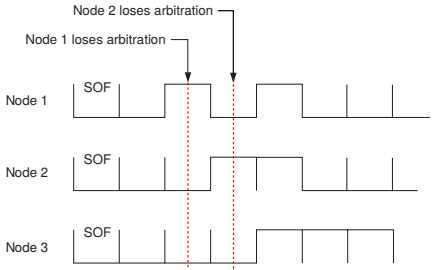
- Two wires, CANH and CANL



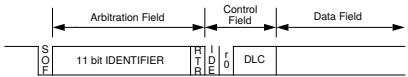


CAN Arbitration

- Frame: Identifier, IDE, DLC and Data fields are the most important
- Arbitration on ID
  - IDs starting with the most 0 wins
  - I.e. lowest/"most dominant" ID
- From CAN 2.0B spec (See **learning materials**)

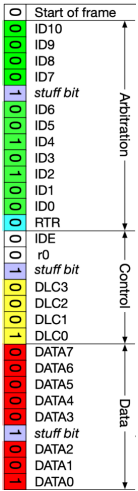


Standard Format



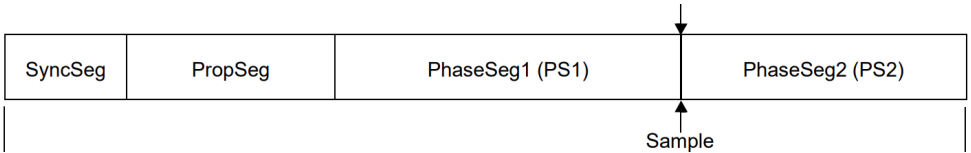
CAN frame breakdown

- SOF: Start of Frame
- ID: Identifier
- RTR: Remote Transmission Request
- IDE: ID exended bit
- DLC: Data Length Counter
- DATA: Data contents
- Hidden confusion: stuff bits
- Some unimportant fields after data



CAN bit timing: quanta

- Asynchronous $\implies$ need other bit synchronisation method.
- Solution: Split bits into smaller *segments* defined by a number of 'quanta'.
- Rising edge acts as synchronisation segment.
- Signal is sampled between phase 1 and 2.
- MCP2515: Quantum length decided by t_{OSC} and BRP
- Segment time should be the same wall clock time for each unit on the bus.
- Read chapter 5 of the MCP2515 datasheet for more info

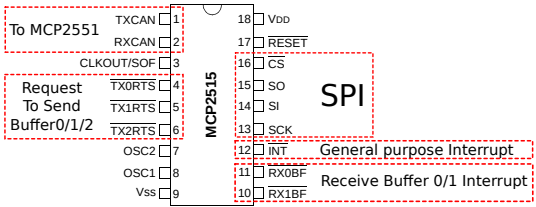


CAN message ID

- Lowest ID: priority
- Broadcast $\Longrightarrow$ recipients filter irrelevant messages.
- Our responsibility: create a clever ID system.
- MASK: Determine which ID bits are (not) DC.
- FILTER: Determine the accetable value of non-DC bits.
- See table 4-2 in MCP2515 for an example.

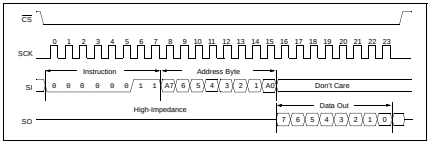
CAN Controller (MCP2515)

- Uses SPI bus.
- Handles the framing stuff (transport layer).
- Controlled by writing and reading registers, much like the IO board units.
- External interrupts.
- Needs its own crystal.

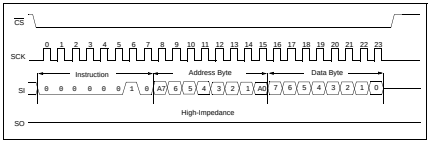


Using the MCP2515

- Read



- Write



- SPI Interface
  - Select chip (CS= 0)
  - Send one byte instruction
  - Send/read additional bytes (address, bit mask, data)
  - Deselect chip(CS = 1)
- Section 12 of the datasheet contains instruction set.

Modes of MCP2515

- Configuration mode: Setup filters, masks and transceiver bit timings.
- Normal mode: Normal functionality.
- Sleep mode: Saves power when device is not used.
- Listen-only mode: Only receiving.
- Loopback mode: Internal transmission.
- Sections 10-11 detail modes and registers.
- CAN controller mode $\neq$ SPI mode :)

CAN configuration summary

What information must be configured in the node on init, and what is encoded in each data frame?

Configuration

- Bit timing
- Mask and filter
- Mode

= Sent via SPI on init

Frame

- ID, data length, etc
- The data itself

= Sent via SPI for each CAN message

$\Longrightarrow$ Make a suitable data structure

Transmission and Reception MCP2515

Reception

- Wait for a received message
  - Interrupt pin (enable using CANINTE.RXnIE)
  - Read status registers (check CANINTF.RXnIF)
- Read message
  - ID (RXBnSIDH & RXBnSIDL)
  - Data length (RXBnDLC)
  - Data (RXBnDM)
- Filter and Masks
  - RXBxCTRL.FILHIT<2:0> with RXFnSIDH, RXMnSIDH. . . .
- Datasheet chapter 4

Transmission

- Setup
  - Message ID (TXBnSIDH & TXBnSIDL)
  - Data length (TXBnDLC)
  - Data (TXBnDm)
- Request-to-send command. (TXBnCTRL.TXREQ)
- Datasheet chapter 3

MCP2515: summary of important registers

- CANCTRL
- CNFn
- CANSTAT
- TXBnSIDH/TXBnSIDL
- RXBnSIDH/RXBnSIDL
- CANINTE/CANINTF
- TXBnDLC/RXBnDLC

Example: Useful low-level functions

MCP/CAN controller

- `mcp2515_read()`
- `mcp2515_write()`
- `mcp2515_request_to_send()`
- `mcp2515_bit_modify()`
- `mcp2515_reset()`
- `mcp2515_read_status()`

- Tip: Header file for MCP2515 with register names and addresses is provided on the Blackboard under ‘Lab Support Data->Miscl. Resources’

SPI

- `SPI_send()`
- `SPI_read()` - Remember, to read something from a slave the master must transmit a dummy byte
- `SPI_init()`

Example Code - Low Level

```
uint8_t mcp2515_read(uint8_t address)
{
    uint8_t result;

    PORTB &= ~(1<<CAN_CS); // Select CAN-controller

    SPI_write(MCP_READ); // Send read instruction
    SPI_write(address); // Send address
    result = SPI_read(); // Read result

    PORTB |= (1<<CAN_CS); // Deselect CAN-controller

    return result;
}
```

Example Code - MCP driver

```
uint8_t mcp2515_init()
{
    uint8_t value;

    SPI_init(); // Initialize SPI
    mcp2515_reset(); // Send reset-command

    // Self-test
    mcp2515_read(MCP_CANSTAT, &value);
    if ((value & MODE_MASK)  != MODE_CONFIG) {
        printf("MCP2515 is NOT in configuration mode after reset!\n");
        return 1;
    }

    // More initialization

    return 0;
}
```

Tips for Exercise 5

- CAN
  - Loopback mode: no messages visible on data lines
  - Chapter 3 of MCP2515 contains info on minimum setup
  - Hardest part of the project, probably
- To do steps
  1. Power-up CAN controller MCP2515(power+crystal...)
  2. Connect MCP2515 with Atmega162 via SPI
  3. Write MCP2515 driver
  4. Test CAN in loopback mode

Executive summary of CAN and E5

- MCP2515 datasheet chapters 2-5, 7, 10-12
- CAN 2.0B spec: Learning materials, strongly encouraged.
- CAN bus: asynchronous, arbitration on message ID.
- CAN nodes: must have identical bit timing settings.
- CAN frame, aka message: some 120 bits of metadata + data.
- CAN controller: MCP2515 communicates via SPI.
- E5: Get MCP2515 working in loopback mode, use the provided h-file.

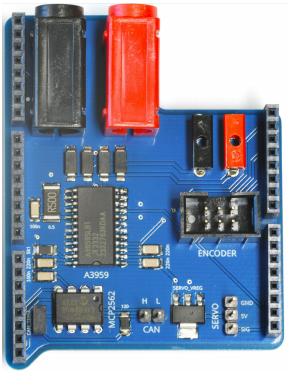
Questions?

Exercise 6: Introducing node 2 - MCU

- Arduino DUE with expansion card ("shield")
- We will **NOT** use the Arduino development framework
- ARM Cortex-M3 architecture: ATSAM 3X8E
  - 32-bit architecture (includes registers)
  - PMC: Power Management Controller. Most peripherals are deactivated by default. See table 9-1 in datasheet. You will need to activate the clock for the peripherals you will be using.
  - Integrated CAN controller
  - Beware of the watchdog (WDT)
- Has additional Atmega 16U2 that acts as an UART to USB bridge.
- ATSAM 3X8E programmed through JTAG.
- **3.3V logic** (not 5v tolerant)

Exercise 6: Introducing node 2 - Shield

- Motor driver: Allegro A3959, Phase/Enable via Timer Counter PWM
- 3x2 encoder connector: quadrature signal via timer counter encoder
- MCP2562 CAN transceiver and terminator jumper
- 3x1 servo connector: angular position via timer counter PWM

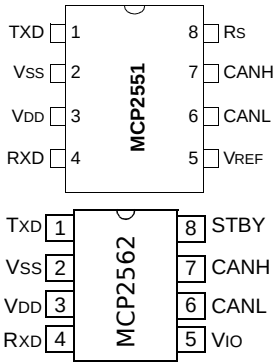


Exercise 6: Blackboard Libraries

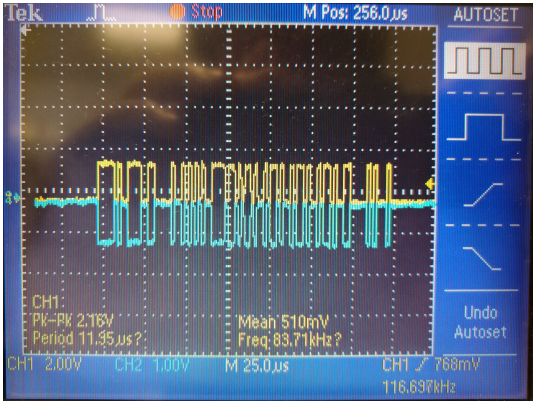
- Basic UART and CAN drivers available on Blackboard
  - Study and try to understand these before your lab day. Search the ATSAM3X8E datasheet for acronyms that you don't understand.
- CAN
  - INIT: You must find a fitting value for CAN_BR. Search in datasheet (40.9.6 and 40.7.4 w/Example at end). This must correspond to the value you set in the CNFx registers on MCP2515.
  - Interrupt handler example
- UART
  - Find/set the UART baud in uart.c

CAN transceivers and controllers

- CAN Controller MCP2515 and ATSAM3X8E integrated
- CAN Transceiver MCP2551 and MCP2562 (drop in replacement for MCP2551)
  - Handles the physical layer
  - Detects line errors
  - Protects against transients
  - End node termination of 120Ω
- Read *AN228; A CAN Physical Layer Discussion* (On Blackboard)
- Enable normal mode in Node 1



CAN Oscilloscope Example



- Remember the stuffing bits

CAN Driver: High-Level

- `can_init()`
- `can_message_send()`
- `can_data_receive()`
- `can_int_vect()`
- Tip: Structs could be useful for CAN messages

```
struct can_message{
    uint16_t id;
    char data_length;
    char data[8];
};
```

Using Structs for CAN messages

- Instantiatiation and Usage

```
int main(void) {
    can_init();
    struct can_message message;
    message.id = 3;
    message.length = 1;
    message.data[0] = 'U';
    can_message_send(&message);
}

void can_message_send(struct can_message* msg) {

    uint8_t i;
    for (i = 0; i < msg->length; i++)

}
```

Exercise 6: Tips

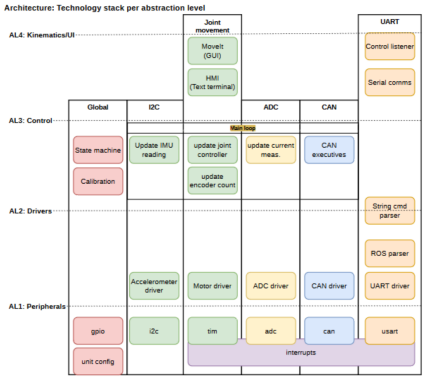
- Catch-up week
- Be careful with CAN message transmission rate
  - Should be controlled/limited e.g. using a TIMER
- Hardware tips
  - Remember 120 ohm resistors at both ends
  - Use 22K ohm resistor for MCP2551 (Rs and GND)
- To do steps
  1. Connect CAN transceiver MCP2551 to node 1
  2. Program node 2 for CAN reception
  3. Verify transmission and reception by sending a dummy byte over CAN from node 1 to 2 and displaying it e.g. on UART on node 2
  4. Test CAN transmission between node 1 and 2. e.g. joystick position

Questions

- Blackboard QnA section
- E-mail: kristian.blom@ntnu.no
- Slides available on Blackboard

Bonus: Software modularisation

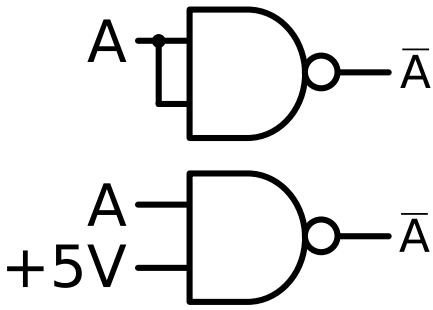
- Modules differentiated by tech stack and abstraction
- Interfaces: A module should not be able to send or receive information outside of a well defined set of functions
- Code reuse: only reuse if safe, make dedicated modules for generic code
  - Limited relevance in the embedded C context as most functionality is bespoke
- Spaghettification happens when fuzzy interfaces and liberal reuse



Bonus: Inverter from NAND gate

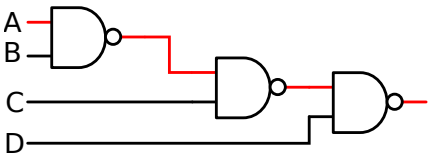
- Logically equivalent
- The first is undesirable because:
  - TTL: Higher current load on driver.
  - CMOS: Higher capacitance $\rightarrow$ slower switching time
- The SN74HC00 used in the lab is CMOS based (see datasheet)

  $\rightarrow$ slower switching time



Bonus: Logic Timing and Critical-Path

- A logic operation takes time to stabilize
- Logic operations in series adds up stabilizing time
- Decoding logic must be stable when AVR performs read/write
- The critical path can be defined as the path between an input and an output with the maximum delay.
- Not a problem in the lab
- If interesting $\rightarrow$ TFE4141 - Design of Digital Systems 1



NAND:

$t_{pd} = 20\text{ns}$

Critical path:

$t_{cp} = 3 \cdot 20\text{ns} = 60\text{ns}$

Thank you for your attention