TTK4155

Industrial and Embedded Computer Systems Design

NTNU – Trondheim
Norwegian University of
Science and Technology

Exercise 5 & 6

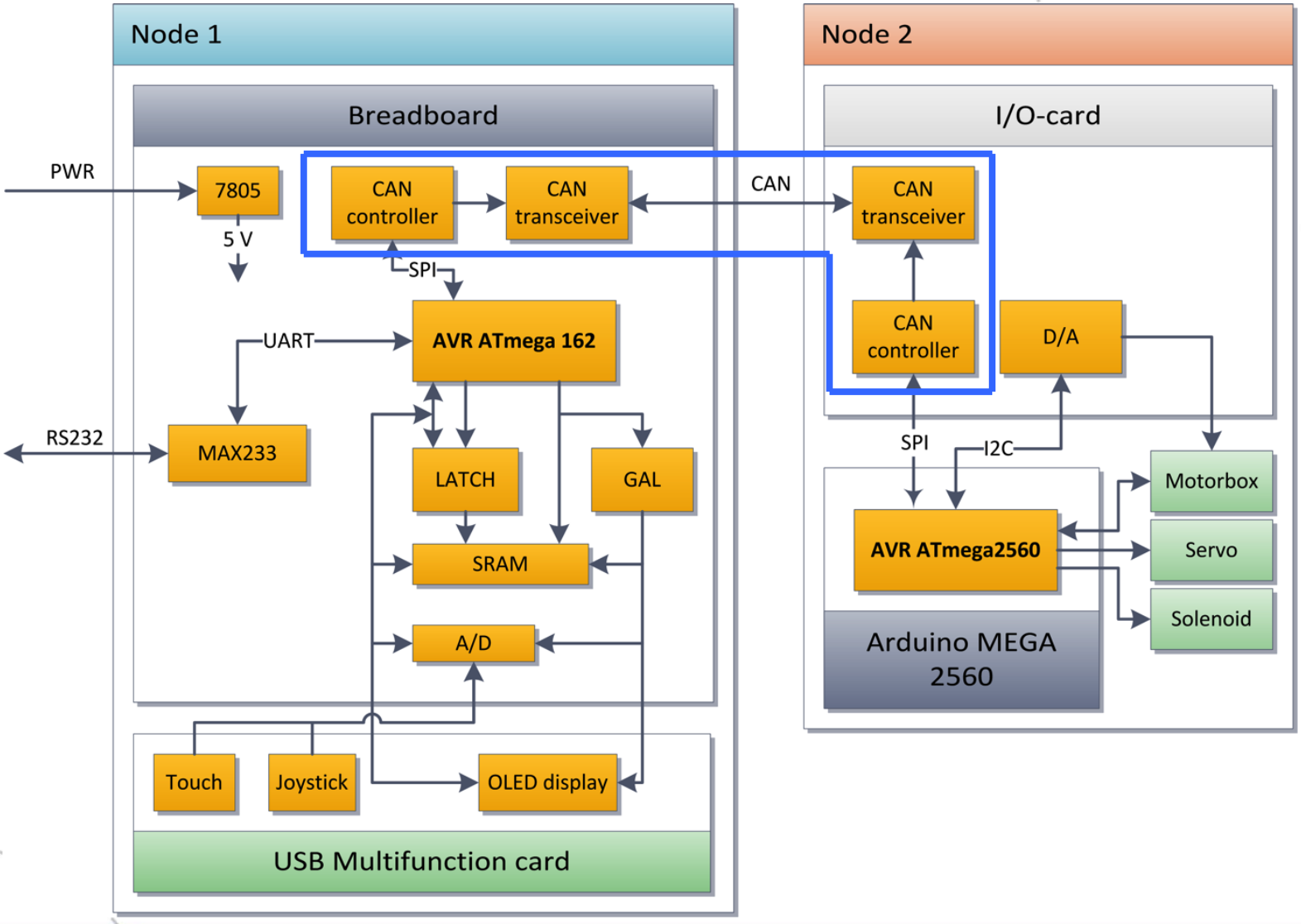
- SPI & CAN bus
- CAN controller & transceiver
- Node 1 & 2 comm.

Exercise 5 & 6: SPI and CAN communication

- In exercise 5:
  - Connect the CAN controller (MCP2515) to the MCU via SPI bus
  - Create SPI and CAN controller drivers (write/read registers etc.)
  - Test MCP2515 in loopback mode

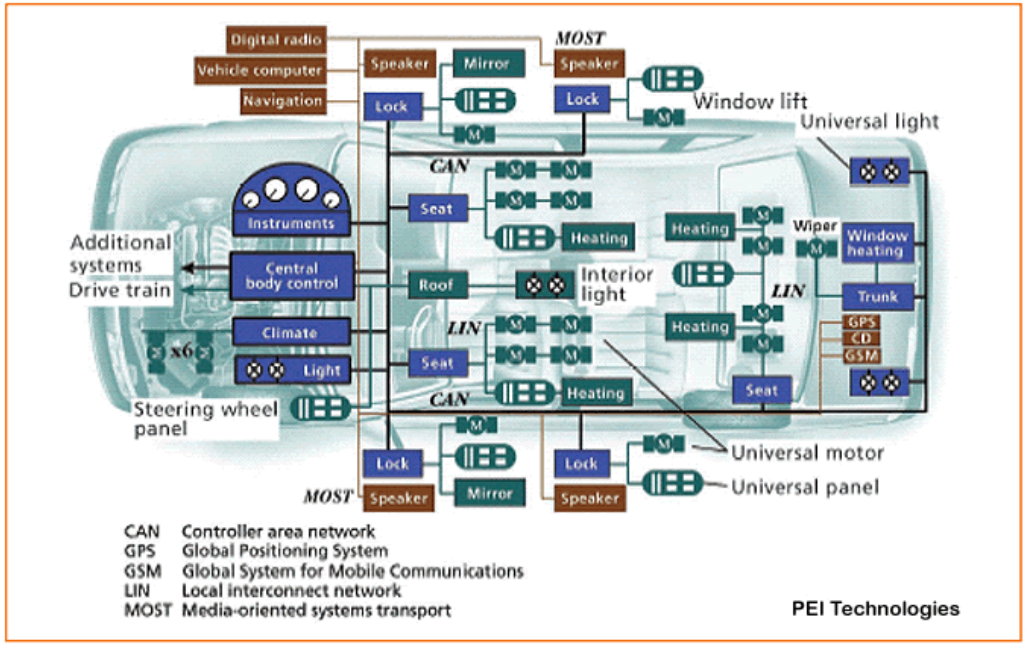
- In exercise 6:
  - Connect CAN transceiver (MCP2551) to the CAN controller
  - Create a CAN communication driver (send/receive functions)
  - Program and test communication with node 2

Communication bus

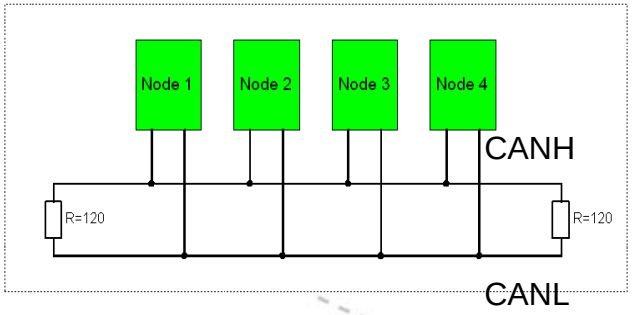


Overview of CAN

- Controller Area Network.
- Vehicle bus = Noise resilient, delivery assurance etc.
- Multi-master broadcast bus protocol.
- Arbitration without delay.
- Limited datagram size
- Up to 1 Mbit/s
  (for <40m)

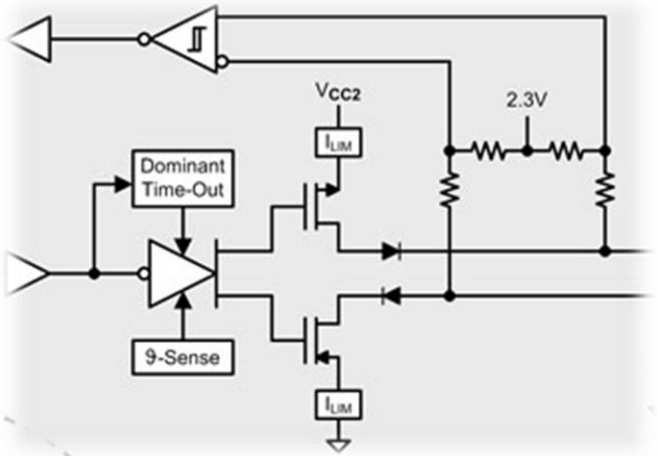


CAN Physical Layer



- Two wires, CANH & CANL

- Two states
  - Logical 1: Recessive state – CANH = CANL = Vcc/2
  - Logical 0: Dominant state – CANH ≈ Vcc and CANL ≈ Gnd



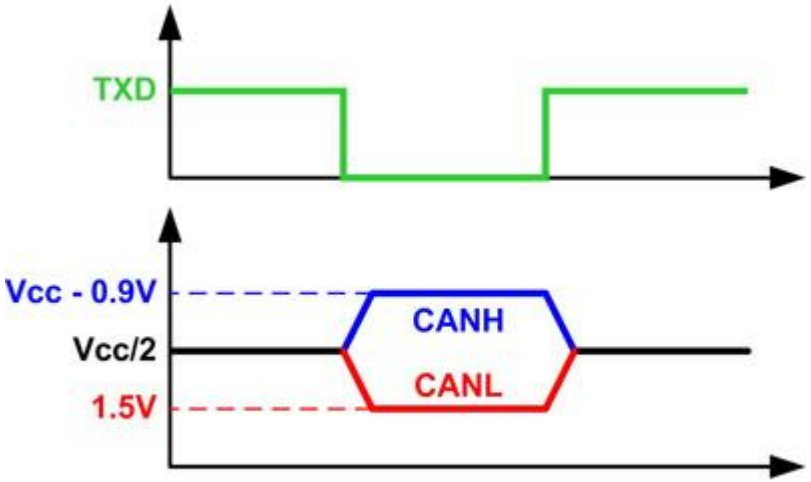
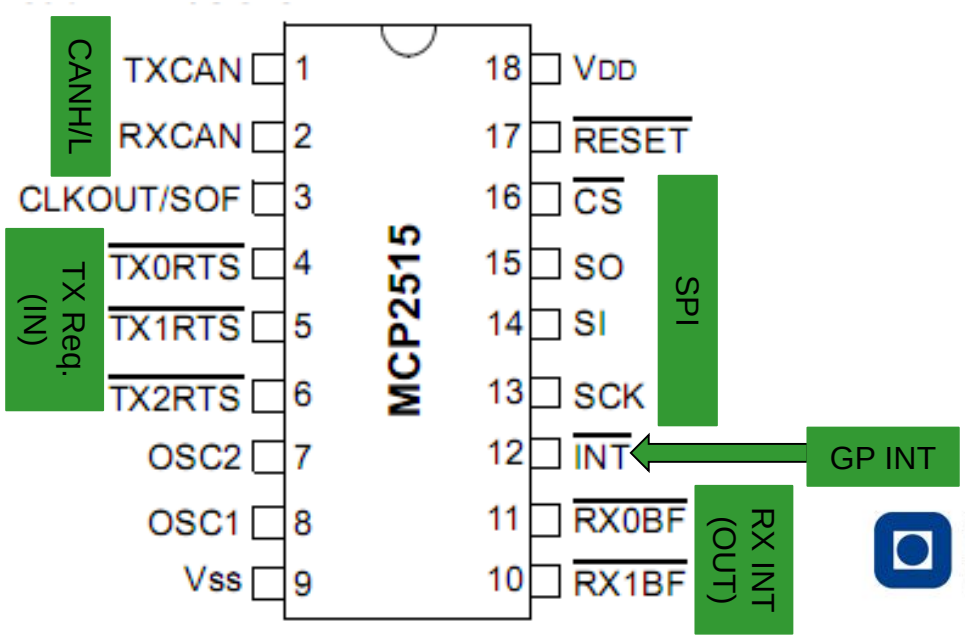


Figure 3. CAN Bus Signals

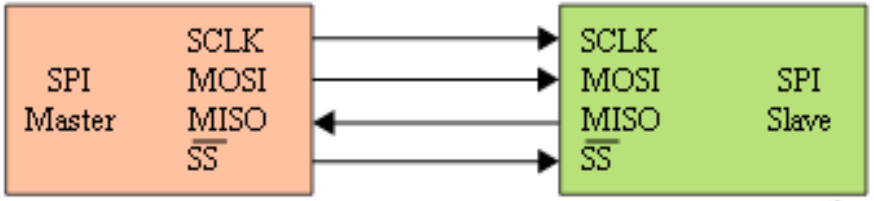
CAN Controller MCP2515

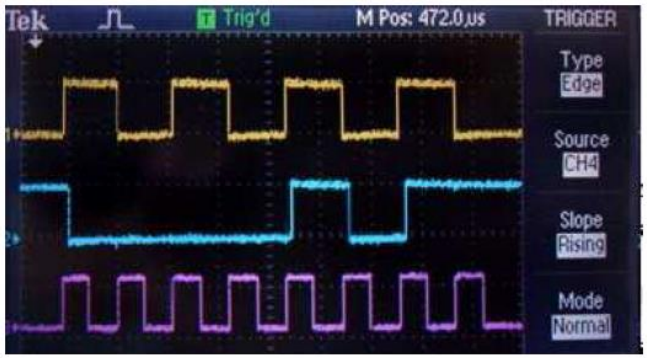
- Handles the framing stuff (transport layer).
- Controlled by writing and reading registers.
- External interrupts.
- Uses SPI bus



Serial Peripheral Interface (SPI)

- Synchronous, serial data bus.
- Master/Slave configuration.
- 4-line bus.
- Full duplex.





←MISO

←MOSI

←SCK

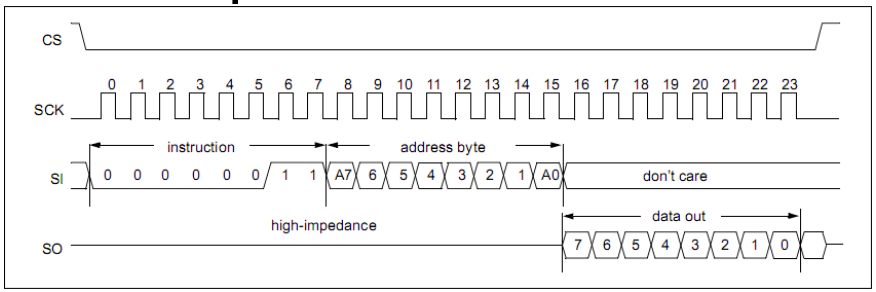
MISO:1 0 1 0 1 0 1 0 (= 170)

MOSI:0 0 0 0 1 0 1 1 (= 11)

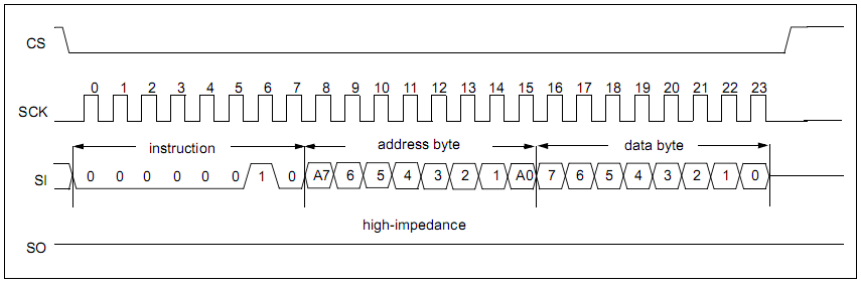
	Leading Edge	Trailing eDge	SPI Mode
CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)	1

Using the MCP2515

- 'Instruction' based communication over SPI:
  - Select chip (CS= 0)
  - Send one byte instruction
  - Send/read additional bytes (address, bit mask, data)
  - Deselect chip(CS = 1)
- MCP2515 example:
  - Read:



  - Write:



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

Example of useful low-level functions

- SPI
  - SPI_send()
  - SPI_read() – Remember, to read something from a slave the master must transmit a dummy byte
  - SPI_init()
- MCP/CAN controller
  - mcp2515_read()
  - mcp2515_write()
  - mcp2515_request_to_send()
  - mcp2515_bit_modify()
  - mcp2515_reset()
  - mcp2515_read_status()
  - Page 63 in the datasheet

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

Code example – Low level

- mcp2515_read()

```
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;
}
```

Code example-MCP driver

- mcp2515_init()

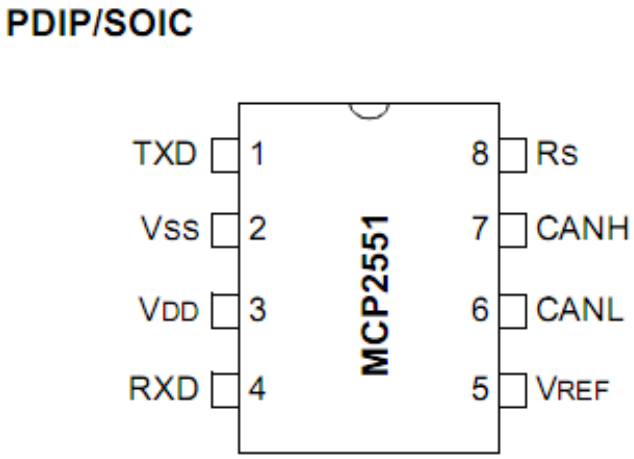
```
uint8_t mcp2515_init()
{
        uint8_t value;

        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 ex_4

- SPI
  - Full duplex, send/receive is simulatneous
  - SPDR is written first and after 8/9 clock cycles, SPDR is read
  - SPI is multi-slave bus, ensure only one slave is selected
  - Check SPI driver on oscilloscope before proceeding with MCP/CAN
- To do steps
  1. Power-up CAN controller MCP2515(power+crystal...)
  2. Connect MCP2515 with Atmega162 via SPI
  3. Develop and test SPI driver
  4. Write MCP2515 driver
  5. Test CAN in loopback mode

EX 6: CAN bus and comm. b/w nodes

- CAN Transceiver MCP2551
  - Handles the physical layer
  - Detects line errors
  - Protects against transients
  - End node termination of 120Ω



- Read AN228; A CAN Physical Layer Discussion

Node 2

- Arduino mega 2560 with expansion card.
- Programming => JTAG interface via expansion card.
- I/O card => provides CAN interface, DAC etc.
- See datasheet/schematics.

Transmission using MCP2515

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

Reception using MCP2515

- 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….

CAN Driver

- **High-level**
  - can_init()
  - can_message_send()
  - can_error()
  - can_transmit_complete()
  - can_data_receive()
  - can_int_vect()

- Tip
  - Structs could be useful for CAN messages

Structs

- **Defining**

```
struct can_message{
        unsigned int id;
        uint8_t length;
        uint8_t data[8];
};
```

- **Instantiatiation and usage**

```
int main(void) {
        can_init();
        struct can_message message;
        message.id = 3;
        message.length = 1;
        message.data[0] = (uint8_t)'U';
        can_message_send(&message);
}

void can_message_send(struct can_message* msg) {
        …
        uint8_t i;
        for (i = 0; i < msg->length; i++)
        ...
}
```

Tips for ex_5

- For node 2, **reuse drivers** from node 1 (UART, CAN etc.)
- 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)
  - Check node 2 schematics to avoid using already used pins
- 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 & 2, e.g. joystick position

Questions?

Auf wiedersehen