TTK4155

Industrial and Embedded Computer Systems Design

NTNU – Trondheim
Norwegian University of
Science and Technology

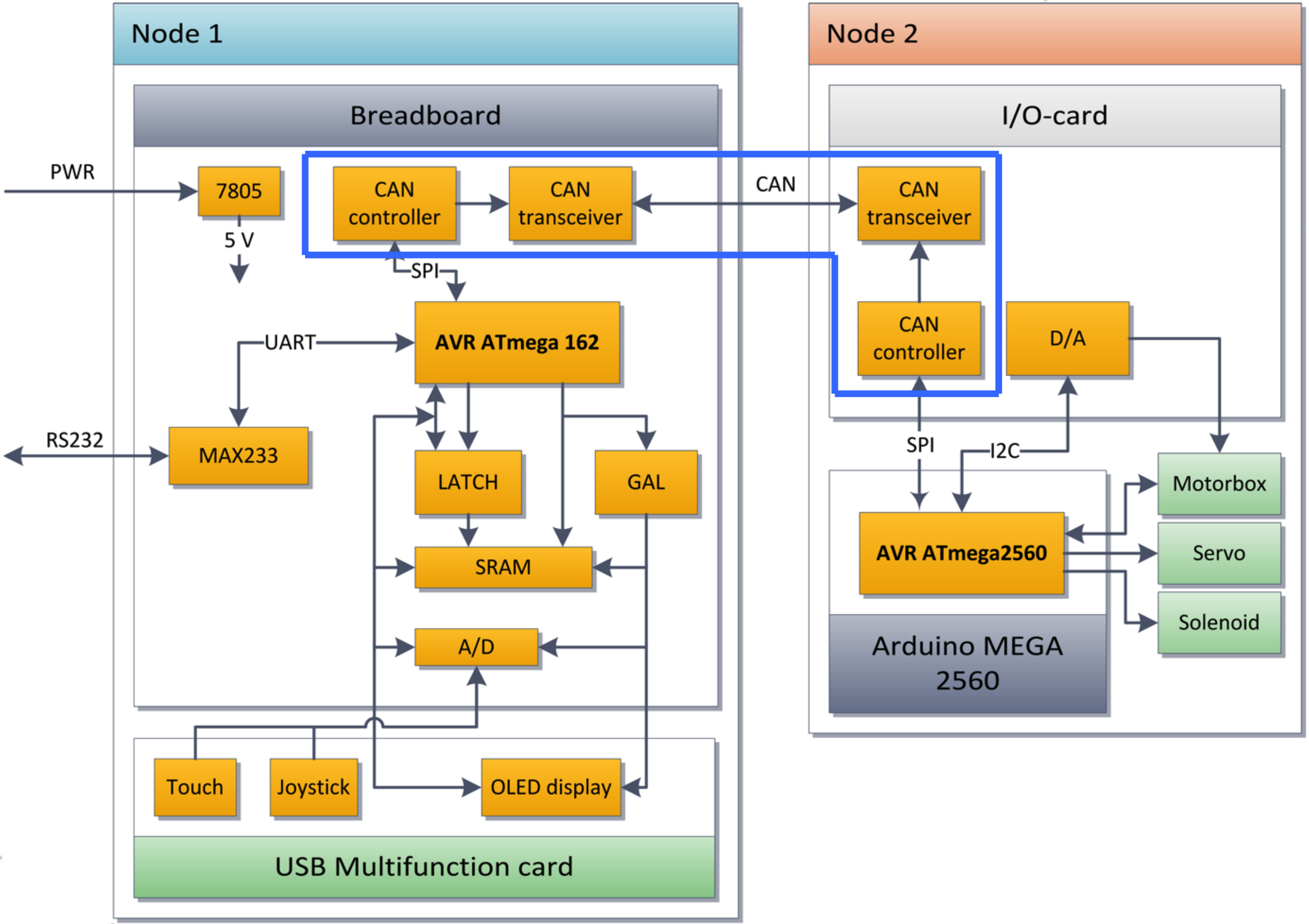
Lab lecture 5

- SPI bus
- CAN protocol
- CAN controller

Exercise 5: SPI and CAN controller

- In this exercise, you will
  - Connect basic circuitry for a CAN controller (MCP2515).
  - Connect the CAN controller to the MCU via SPI bus.
  - Create an SPI driver.
  - Create a CAN controller (MCP2515) driver (write/read registers etc.).
  - Create a high level CAN driver.

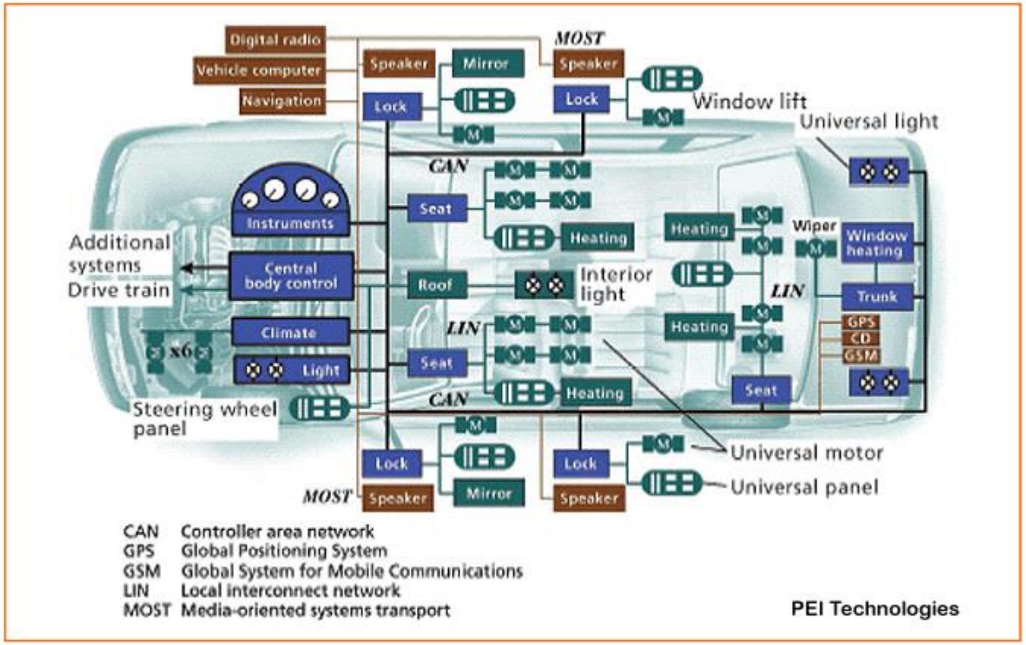
Communication bus



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	Controller area network
GPS	Global Positioning System
GSM	Global System for Mobile Communications
LIN	Local interconnect network
MOST	Media-oriented systems transport

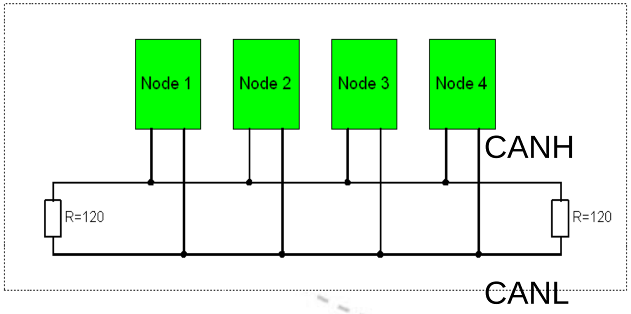
PEI Technologies

CAN Layers

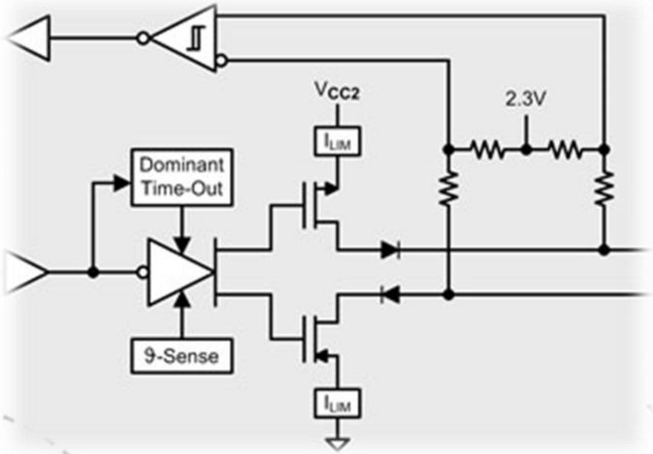
- Defines 2 layer in OSI model.
- Transfer Layer
  - Fault confinement
  - Error detection
  - Message validation
  - Acknowledgement
  - Arbitration
  - Message framing
  - Transfer rate and Timing
  - Information routing
- Physical Layer
  - Signal level and bit representation
  - Transmission medium
- Layers like application and network is not defined. You are free to define this for your application. Or use some predefined protocol (CANopen, DeviceNet, EnergyBus, etc)

CAN Physical Layer

- Two wires, denoted CANH and CANL



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



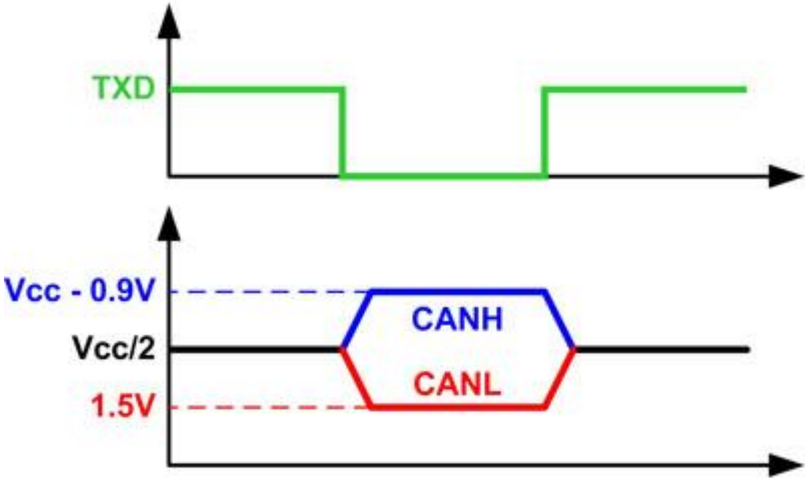


Figure 3. CAN Bus Signals

CAN Physical Layer

- What if multiple nodes transmits at the same time?
- Arbitration:
  - Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA)
- From the CAN standards document:
  - “If 2 or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the IDENTIFIER. The mechanism of arbitration guarantees that neither information nor time is lost.”
  - The sender with the message with the lowest ID will win arbitration. Other senders will back off.

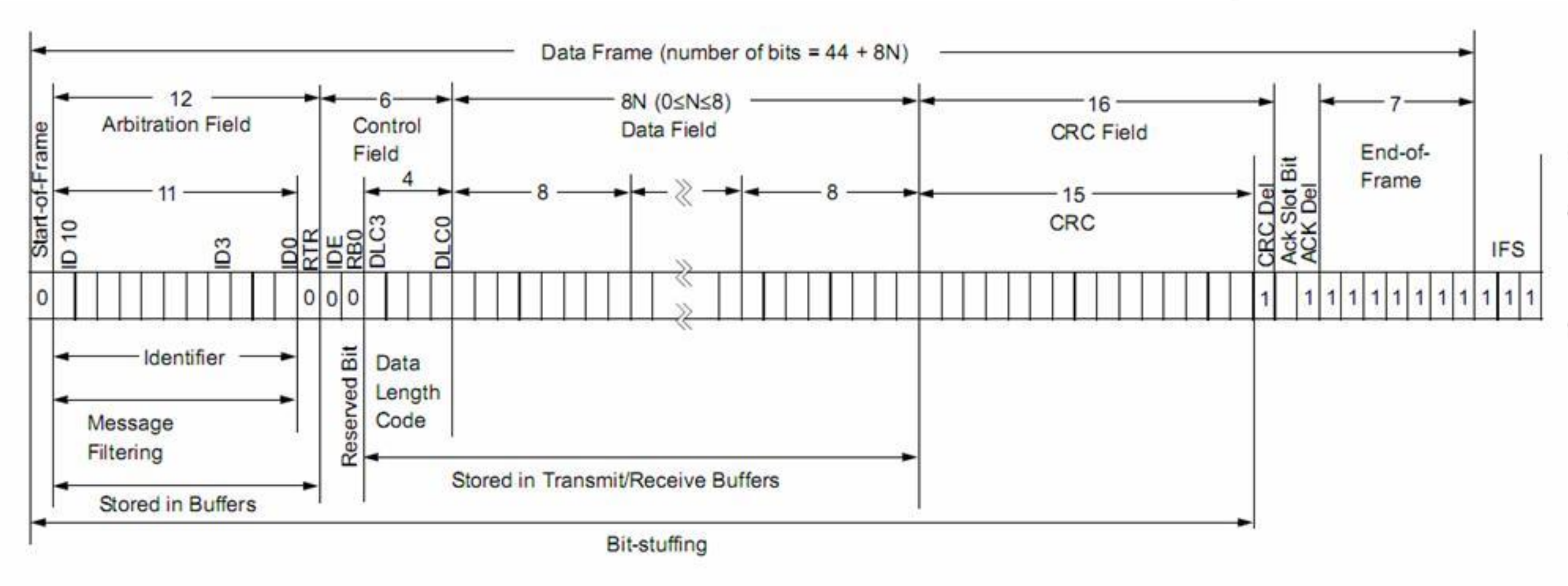
CAN Frame Types

- **Standard Frame**
  - Start-of-Frame-Bit
    - Synchronization
  - Arbitration
    - 11-bit identifier
    - Remote transmission request
  - Control Field
    - Data length
  - Data Frames
    - X*8 bit data
  - CRC (15+1 bits)
  - ACK (2 bits)

- **Extended**
  - 29 bits ID
- **Remote Frames**
  - A node requests data from another
- **Error Frame**
  - Error Flag, Error ID (2 bits)
- **Overload Frame**
  - Error Flag, Error ID (2 bits)
- **Interframe Space**
  - “Pause” between frames

CAN Frame Structure

- Standard CAN data frame:

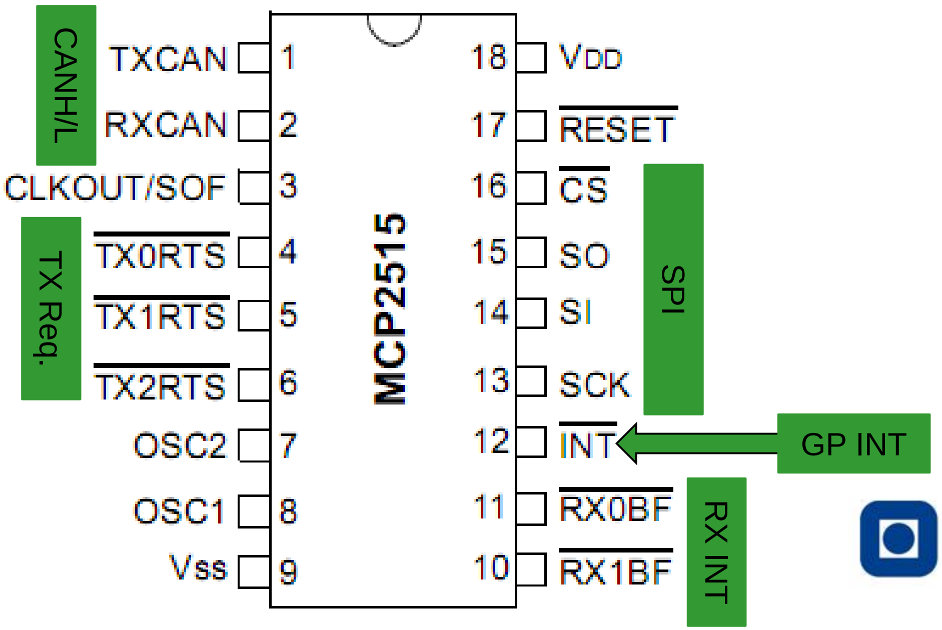


- http://en.wikipedia.org/wiki/Controller_area_network

NTNU – Trondheim
Norwegian University of
Science and Technology

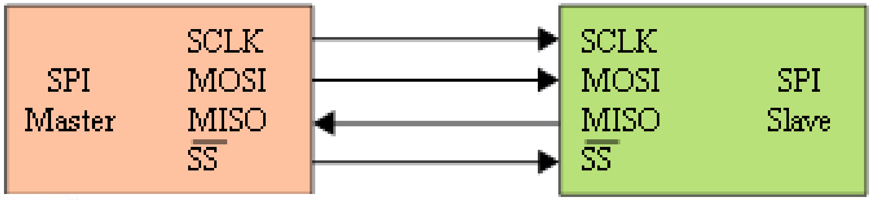
CAN Controller MCP2515

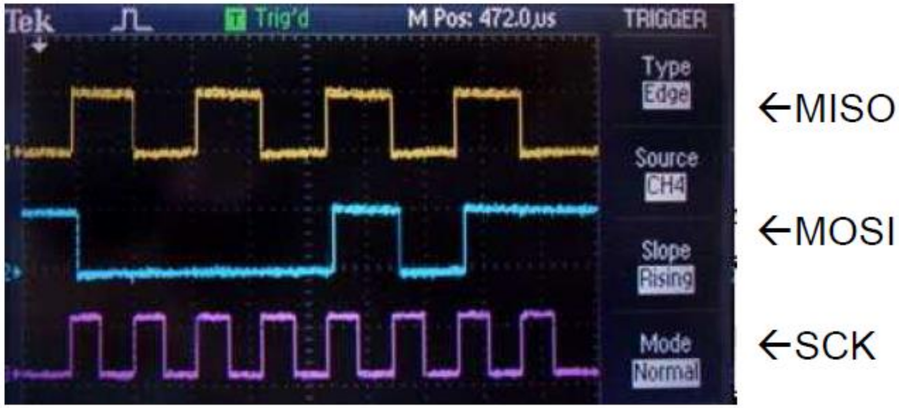
- Handles the transfer layer.
- Uses SPI (Serial Peripheral Interface).
- Controlled by writing and reading registers.
- External interrupts.



SPI bus

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





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

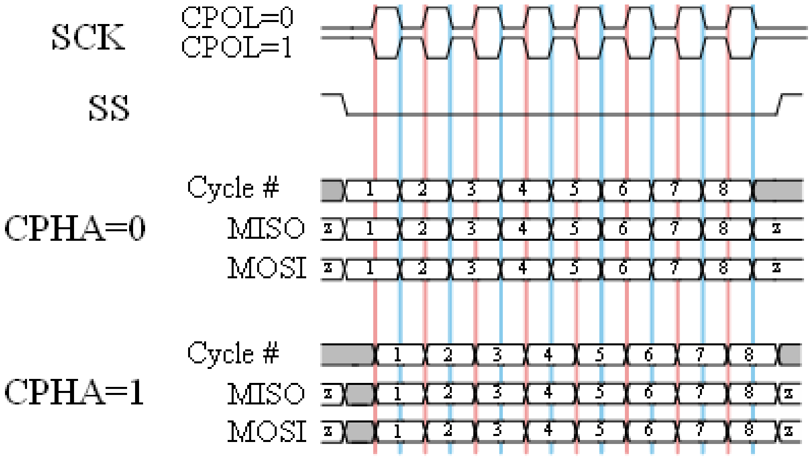
MOSI:0 0 0 0 1 0 1 1 (= 11)

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

NTNU – Trondheim
Norwegian University of
Science and Technology

SPI – Configuration & Modes

- Clock polarity (CPOL) and phase (CPHA)



- Data order (DORD), SPI enable (SPE), Clock speed (SPR1:0) , Data register (SPDR), Master mode enable (MSTR)
- Read AVR data sheet!

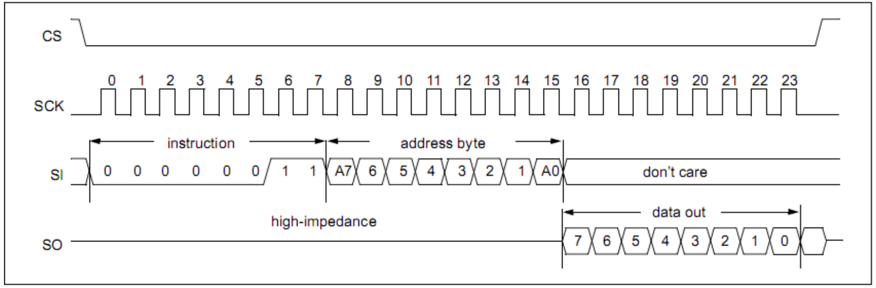
Example of useful low-level functions

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

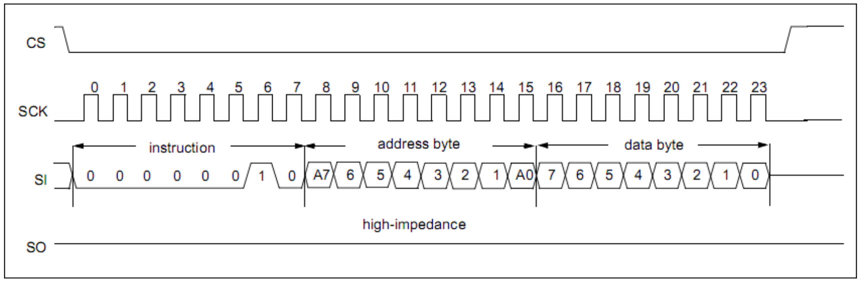
TIPS: Header file for MCP2515 with register names and addresses is provided on the Blackboard under ‘Lab Support Data/Miscl’.

Using the MCP2515

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



  - Write:



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 command
        SPI_write(address); // Send address
        result = SPI_read(); // Read result

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

        return result;
}
```

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

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()

- Tips:
  - Structs could be useful for messages

Code example

- 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;
}
```

Structs

- **Defining**

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

- **Instantiation 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++)
            ...
}
```

What to do?

- Connect CAN controller MCP2515
- Write SPI driver
- Test SPI (oscilloscope, simple read from MCP2515)
- Write MCP2515 driver
- Write CAN driver
- Test CAN in loopback mode

Questions?

Auf wiedersehen