Systemtechnik

Hardwarenahe Programmierung
Protokoll 4: Kommunikation

Verfasser:

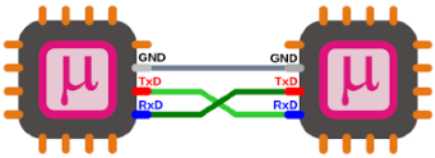
- **Jonas Ruhland**
- **Clemens Rumpfhuber**

Übungsleiter: DI Mag. Ferdinand Hell

Theoretische Ausarbeitung 2
Universal Asynchronous Receiver Transmitter (UART) 2
Inter-Integrated Circuit (I^2C) 2
Serial Peripheral Interface (SPI) 3
Controller Area Network (CAN) 3
Praktische Ausarbeitung 4
Universal Asynchronous Receiver Transmitter (UART) 4
Inter-Integrated Circuit (I^2C) 10
Serial Peripheral Interface (SPI) 16
Controller Area Network (CAN) 22

Theoretische Ausarbeitung

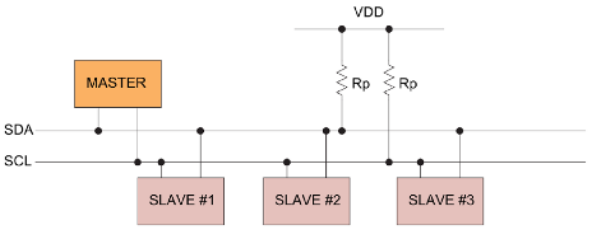
Universal Asynchronous Receiver Transmitter (UART)[1]



Universal Asynchronous Receiver Transmitter ist eine elektronische Schaltung, die zur Realisierung digitaler serieller Schnittstellen dient. Dabei kann es sich sowohl um ein eigenständiges elektronisches Bauelement, wie zum Beispiel ein UART-Baustein, oder um einen Funktionsblock eines höherintegrierten Bauteils, wie zum Beispiel eines Mikrocontrollers, handeln. In unserem Beispiel kommunizieren zwei Mikrocontroller miteinander.

Eine UART-Schnittstelle dient zum Senden und Empfangen von Daten über eine Datenleitung und bildet den Standard der seriellen Schnittstellen an PCs und Mikrocontrollern. Auch im industriellen Bereich ist die Schnittstelle mit verschiedenen Interfaces (z. B. RS-232 oder EIA-485) sehr verbreitet.

Inter-Integrated Circuit (I²C)



I²C, fürInter-Integrated Circuit ist ein 1982 entwickelter serieller Datenbus.[2]

Aus Lizenzgründen heißt der I²C-Bus bei manchen Herstellern auch TWI, two wire interface. Im PC wird ein dem I²C-Bus sehr ähnliches System benutzt, um z. B. die Daten eines SDRAM-Modules auszulesen.[3]

Er wird meistens geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen benutzt, zum Beispiel zwischen einem Controller und Peripherie-ICs. Das ursprüngliche System wurde von Philips in den frühen 1980er Jahren entwickelt, um verschiedene Chips in Fernsehgeräten einfach steuern zu können.

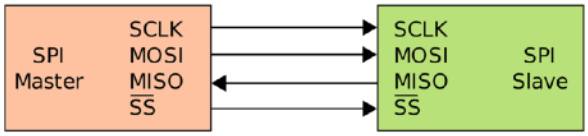
BUSSYSTEM

I²C ist als Master-Slave-Bus konzipiert. Ein Datentransfer wird immer durch einen Master initiiert; der über eine Adresse angesprochene Slave reagiert darauf. Mehrere Master sind möglich (Multimaster-Betrieb). Wenn im Multimaster-Betrieb ein Master-Baustein auch als Slave arbeitet, kann ein anderer Master direkt mit ihm kommunizieren, indem er ihn als Slave anspricht.

[1] https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter (17.02.2022 10:55 Uhr)
[2] https://de.wikipedia.org/wiki/I²C (17.02.2022 11:00 Uhr)
[3] https://www.mikrocontroller.net/articles/I²C (17.02.2022 11:12 Uhr)

Serial Peripheral Interface (SPI)[4]

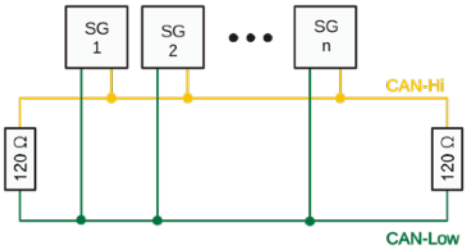
Das Serial Peripheral Interface (SPI) ist ein im Jahr 1987 entwickeltes Bus-System und stellt einen „lockeren“ Standard für einen synchronen seriellen Datenbus (Synchronous Serial Port) dar, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können.



EIGENSCHAFTEN

- SLK (englisch Serial Clock) auch SCK, wird vom Controller zur Synchronisation ausgegeben
- POCI (peripheral out/controller in) oder MISO (englisch Master Input, Slave Output)
- PICO (peripheral in/controller out) oder MOSI (englisch Master Output, Slave Input)
- Vollduplexfähig - Gleichzeitiges Senden und Empfangen
- Unterschiedliche Taktfrequenzen bis in den MHz-Bereich sind zulässig.
- Vielfältige Einsatzmöglichkeiten zur Datenübertragung zwischen Mikrocontrollern

Controller Area Network (CAN)[5]



Der CAN-Bus (Controller Area Network) ist ein serielles Bussystem und gehört zu den Feldbussen. Dieser ermöglicht eine Geschwindigkeit von bis zu 1 Mbit/s, der den seriellen Datenaustausch zwischen Steuergeräten ermöglicht.

Sein Zweck ist es, Kabelbäume zu reduzieren und hiermit Kosten und Gewicht zu sparen. Die beiden gängigsten Realisierungen der physischen Schichten sind nach ISO 11898-2 (Highspeed physical layer) und ISO 11898-3 (Fault tolerant physical layer) definiert. Sie unterscheiden sich in zahlreichen Eigenschaften und sind nicht zueinander kompatibel.[6]

ÜBERTRAGUNGSVERFAHREN

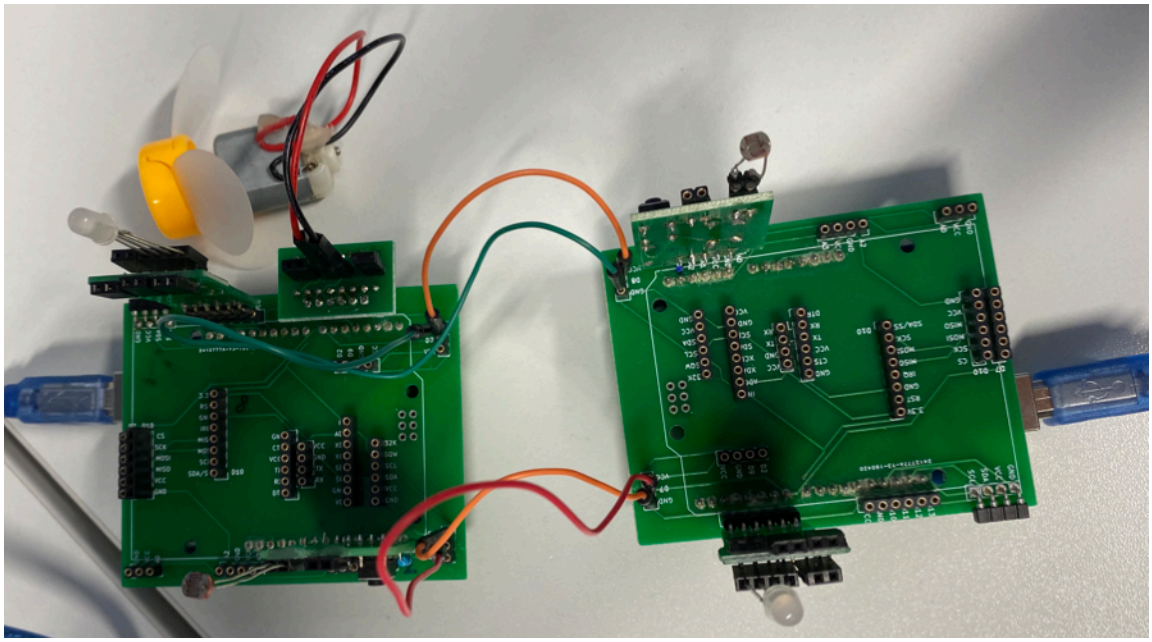
Jedes Steuergerät ist über eine CAN-Schnittstelle (Bus-Controller + Bus-Transceiver) an den Bus angeschlossen und prüft zunächst, ob die über den Bus gesendeten Datenpakete von Bedeutung für es sind. Das geschieht über sogenannte Identifiers, die in jedem Datenpaket enthalten sind und Auskunft über Dateninhalt und die Priorität der Information geben. Sollten mehrere Steuergeräte gleichzeitig versuchen, Informationen zu senden, wird überprüft, welche Nachricht die höchste Priorität hat. Diese Nachricht wird zuerst versendet, die anderen Nachrichten folgen nach Priorität, sobald der Bus wieder frei ist. Der CAN-Bus kann darüber hinaus fehlerhafte Übertragungen erkennen und entsprechend wiederholen.

[4] https://de.wikipedia.org/wiki/Serial_Peripheral_Interface (17.02.2022 11:20 Uhr)
[5] https://www.mein-autolexikon.de/elektronik/can-bus.html (17.02.2022 11:42 Uhr)
[6] https://de.wikipedia.org/wiki/Controller_Area_Network (17.02.2022 11:29 Uhr)

Praktische Ausarbeitung

Universal Asynchronous Receiver Transmitter (UART)

VERKABELUNG & AUFBAU



Slave D7 <-> Master D8 Slave D8 <-> Master D7

CODE - SLAVE.INO

```
#include <SoftwareSerial.h>

#define PIN_RX     7
#define PIN_TX     8
#define PIN_LED    4
#define PIN_SWITCH 2
#define PIN_LDR    A1

#define LED_OFF      0x01
#define LED_ON       0x02
#define READ_SWITCH 0x03
#define READ_LDR    0x04
#define NO_ACTION   0xFF
#define WRITE_LDR   0x05

#define MOTOR_A     10
#define MOTOR_B     11
#define MOTOR_FWD   12
#define MOTOR_BWD   13

volatile byte received;
int motorValue = 0;

SoftwareSerial softSerial(PIN_RX, PIN_TX);

void setup()
{
  Serial.begin(115200);

  // setting LED pin as OUTPUT
  pinMode(PIN_LED, OUTPUT);

  setupSerialComm();
```

```
  // set output mode on motor
  pinMode(MOTOR_A, OUTPUT);
  pinMode(MOTOR_B, OUTPUT);
  pinMode(MOTOR_FWD, OUTPUT);
  pinMode(MOTOR_BWD, OUTPUT);

  // set default motor rotation to forward
  digitalWrite(MOTOR_FWD, HIGH);

  // print usage message onto device connected to HW Serial
  printInfo();
}

void loop() {
  // fetch received command
  if (softSerial.available() > 0) {
    // receive byte as a character
    received = softSerial.read();
    // process command received from master
    doCommand();
  }
}

void doCommand() {
  static boolean prevLedState = false;
  int ldrValue;

  // distinquish command received
  switch (received) {
    case LED_ON:
      if (!prevLedState) {
        // sets LED pin HIGH to switch LED on
        digitalWrite(PIN_LED, HIGH);
        Serial.println(F("Slave: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        // sets LED pin LOW to switch LED off
        digitalWrite(PIN_LED, LOW);
        Serial.println(F("Slave: LED off"));
        prevLedState = false;
      }
      break;
    case READ_SWITCH:
      //Serial.println(F("Slave: read switch"));
      // send the button state to master
      writeSerialByte(digitalRead(PIN_SWITCH) ? LED_ON : LED_OFF);
      break;
    case READ_LDR:
      //Serial.println(F("Slave: read LDR"));
      ldrValue = analogRead(PIN_LDR);
      Serial.print(F("Slave: LDR: "));
      Serial.print('('); Serial.print((ldrValue >> 8) & 0xFF);
      Serial.print(' '); Serial.print(ldrValue & 0xFF);
      Serial.print(") ");
      Serial.println(ldrValue);
      writeSerialInt(ldrValue);
      break;
    case WRITE_LDR:
      int sensorValue = 0;
      // read sensor
      requestSerialInt(&sensorValue);
      // debug logging
      Serial.print("WRITE LDR ");
      Serial.println(sensorValue);
      // map values for motor
      motorValue = map(sensorValue, 45, 1024, 0, 255);
      break;
    case NO_ACTION:
```

```
      // with SPI: will be sent from master following READ_SWITCH to get button state
      break;
    default:
      Serial.print(F("Slave: undefined command 0x"));
      Serial.println(received, HEX);
  }

  // move motor a
  analogWrite(MOTOR_A, motorValue);
}

void printInfo() {
  Serial.println(F("UART communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: 8 -> 7, 9600 BAUD"));
  Serial.println(F("Slave switches LED on/off according to command received (LED_ON,
LED_OFF)"));
  Serial.println(F("       reads own switch if requested and prepares result for next
transfer (READ_SWITCH)"));
  Serial.println(F("       reads LDR if requested, prepares 2 result bytes (MSB first)
for next transfers (READ_LDR)"));
  Serial.println(F("       prints LDR value sent to master (2 bytes, MSB folowed by LSB)
onto Serial"));
}

void setupSerialComm() {
  softSerial.begin(9600);
}

void writeSerialByte(byte value) {
  softSerial.write(value);
}

void writeSerialInt(int value) {
  softSerial.write((value >> 8) & 0xFF); // send the high byte of the LDR value
  softSerial.write(value & 0xFF);  // send low byte of LDR value to master second
}

boolean requestSerialInt(int *value) {
  boolean success = false;
  byte received[2];
  delay(2);                     // give slave time to accomplish work (1 is not enough)
  success = softSerial.available() >= 2;
  if (success) {                // slave may send less than requested
    received[0] = softSerial.read();       // receive MSB byte of LDR value from slave
    received[1] = softSerial.read();       // receive LSB byte of LDR value from slave
    *value = received[0] * 256 + received[1]; // build int from 2 bytes
    Serial.print(F("Master: LDR: "));
    Serial.print('('); Serial.print(received[0]);
    Serial.print(' '); Serial.print(received[1]);
    Serial.print(") ");
  }
  return success;
}
```

CODE - MASTER.INO

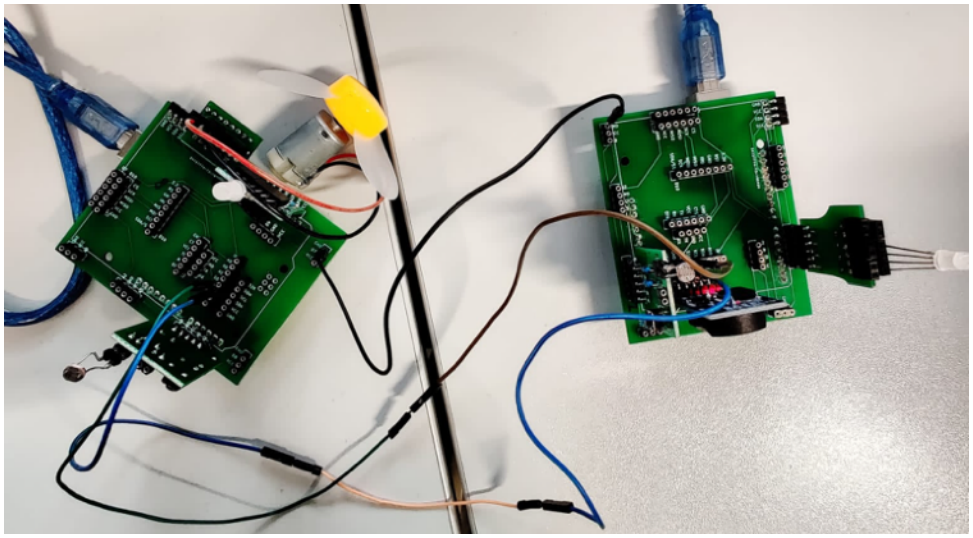
```
#include <SoftwareSerial.h>

#define PIN_RX        7
#define PIN_TX        8
#define PIN_LED       4           // LED on pin 4
#define PIN_SWITCH    2           // tactile switch on pin 2
#define PIN_LDR       A1

#define LED_OFF     0x01          // switch LED off command
#define LED_ON      0x02          // switch LED on command
#define READ_SWITCH 0x03          // read switch state command
#define READ_LDR    0x04
#define WRITE_LDR   0x05          //write ldr value to slave
#define NO_ACTION   0xFF          // dummy command for 'do nothing'
#define WRITE_LDR   0x05

#define READ_SWITCH_DELAY 1000    // delay between 2 read switch commands
#define READ_LDR_DELAY    2500    // delay between 2 read LDR commands
#define WRITE_LDR_DELAY   2000

#define COMM_WRITE_READ_DELAY 2   // delay between write command and read back result

SoftwareSerial softSerial(PIN_RX, PIN_TX);
unsigned long now = 0;
unsigned long ldrTime = 2000L;
int currentLdrValue = 0;

void setup(void) {
  Serial.begin(115200);             // start Serial Communication at Baud Rate 115200
  pinMode(PIN_LED, OUTPUT);         // set LED pin as output

  setupSerialComm();

  printInfo();              // print usage message onto device connected to HW Serial
  //delay(1000);     // wait a moment to avoid unwanted flooding log at the beginning
}

void loop(void) {
  /*
  doSwitch();
  readSlaveSwitch();
  readSlaveLDR();
  */

  // ldr send every 2 seconds
  now = millis();
  if (now - WRITE_LDR_DELAY > ldrTime){
    currentLdrValue = analogRead(PIN_LDR);
    writeSerialInt(WRITE_LDR, currentLdrValue);
    ldrTime = now;
  }
}

void readSlaveSwitch() {
  static unsigned long prevReadSwitchCmd = 2000L; // time of most recent read switch cmd
  byte received = 0;                // byte received upon read switch command
  unsigned long now = millis();      // milliseconds from power up

  if (now - READ_SWITCH_DELAY > prevReadSwitchCmd) {
    prevReadSwitchCmd = now;
    //Serial.println(F("Master: read switch"));
    if (requestSerialByte(READ_SWITCH, &received)) { // slave may send less needed
      doCommand(received);
    }
  }
  //delay(10);
}
```

```
void readSlaveLDR() {
  static unsigned long prevReadLDRCmd = 2000L;  // time of most recent read switch cmd
  byte received[2];                             // byte received upon read switch command
  unsigned int ldrValue;
  unsigned long now = millis();                 // milliseconds from power up

  if (now - READ_LDR_DELAY > prevReadLDRCmd) {
    prevReadLDRCmd = now;
    //Serial.println(F("Master: read LDR"));
    if (requestSerialInt(READ_LDR, &ldrValue)) {  // slave may send less than requested
      Serial.println(ldrValue);
    }
  }
  //delay(10);
}

void doSwitch() {
  static boolean prevSwitchState = false;   // previous state of tactile switch
  boolean switchState = digitalRead(PIN_SWITCH);
  if (switchState != prevSwitchState) {
    prevSwitchState = switchState;
    writeSerialByte(switchState ? LED_ON : LED_OFF);
  }
}

void doCommand(byte received) {
  static boolean prevLedState = false;   // previous state of LED to reduce dig.Write
  switch (received) {
    case LED_ON:      // set the LED state depending upon command received from slave
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);         //Sets LED pin HIGH
        Serial.println(F("Master: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);          //Sets LED pin LOW
        Serial.println(F("Master: LED off"));
        prevLedState = false;
      }
      break;
    case NO_ACTION:
      break;
    default:
      Serial.print(F("Master: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("UART communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: 5 -> 3, 9600 BAUD"));
  Serial.println(F("Master eagerly reads own tactile switch and sends changes to slave
(LED_ON, LED_OFF)"));
  Serial.println(F("requests tactile switch of slave every second (READ_SWITCH)"));
  Serial.println(F("requests LDR value of slave every 2 and a half second (READ_LDR)"));
  Serial.println(F("fetches values from slave (>= 1 byte) by sending according number of
NO_ACTION commands"));
  Serial.println(F("builds result from multiple byte responses (MSB first) (LDR)"));
  Serial.println(F("switches own LED according to slaves answer (LED_ON, LED_OFF)"));
  Serial.println(F("prints LDR value received from slave (2 bytes, MSB followed by LSB)
onto Serial"));
}

void setupSerialComm() {
  softSerial.begin(9600);
}
```

```
void writeSerialByte(byte value) {
  softSerial.write(value);       // sends command byte LED_ON or LED_OFF to slave
}

boolean requestSerialByte(byte command, byte *value) {
  boolean success = false;
  softSerial.write(command);        // send read switch request READ_SWITCH to slave
  delay(COMM_WRITE_READ_DELAY);             // give slave time to accomplish work
  success = softSerial.available() >= 1;
  if (success) {                // slave may send less than requested
    *value = softSerial.read();                 // receive 1 byte from slave
  }
  return success;
}

boolean requestSerialInt(byte command, int *value) {
  boolean success = false;
  byte received[2];
  softSerial.write(command);       // send read switch request READ_LDR to slave
  delay(COMM_WRITE_READ_DELAY);   // give slave time to accomplish work (1 not enough)
  success = softSerial.available() >= 2;
  if (success) {              // slave may send less than requested
    received[0] = softSerial.read();  // receive MSB byte of LDR value from slave
    received[1] = softSerial.read();  // receive LSB byte of LDR value from slave
    *value = received[0] * 256 + received[1]; // build int from 2 bytes
    Serial.print(F("Master: LDR: "));
    Serial.print('('); Serial.print(received[0]);
    Serial.print(' '); Serial.print(received[1]);
    Serial.print(") ");
  }
  return success;
}

void writeSerialInt(byte command, int value) {
    softSerial.write(command);
    softSerial.write((value >> 8) & 0xFF);
    softSerial.write(value & 0xFF);
}
```

Inter-Integrated Circuit (I²C)

VERKABELUNG & AUFBAU



SDA <-> SDA SCL <-> SCL

CODE - SLAVE.INO

```
#include<Wire.h>

#define LEDS_I2C_ADDR 4           // I2C address of LED, LDR, and tactile switch slave

#define PIN_LED    4
#define PIN_SWITCH 2
#define PIN_LDR    A1

#define LED_OFF     0x01
#define LED_ON      0x02
#define READ_SWITCH 0x03
#define READ_LDR    0x04
#define MOTOR       0x05
#define NO_ACTION   0xFF

volatile byte received;
volatile boolean jobAvailable;
volatile byte unsentData[2];
volatile int unsentCount;           // initialized to 0 to indicate no data belonging to
previous command is unsent

void setup()
{
  Serial.begin(115200);
  pinMode(PIN_LED, OUTPUT);               // setting LED pin as OUTPUT

  setupI2cComm();

  printInfo();                // print usage message onto device connected to HW Serial
}

void loop() {
  if (jobAvailable) {                     // react to command received from master
    noInterrupts();
    jobAvailable = false;                 // indicate no message to process
    interrupts();

    doCommand();                          // process command received from master
  }
}
```

```
void doCommand() {
  static boolean prevLedState = false;
  int ldrValue;
  switch (received) {                           // distinquish command received
    case LED_ON:
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);            // sets LED pin HIGH to switch LED on
        Serial.println(F("Slave: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);             // sets LED pin LOW to switch LED off
        Serial.println(F("Slave: LED off"));
        prevLedState = false;
      }
      break;
    case READ_SWITCH:
      //Serial.println(F("Slave: read switch"));
      writeI2cByte(digitalRead(PIN_SWITCH) ? LED_ON : LED_OFF);
      break;
    case READ_LDR:
      //Serial.println(F("Slave: read LDR"));
      ldrValue = analogRead(PIN_LDR);
      Serial.print(F("Slave: LDR: "));
      Serial.print('('); Serial.print((ldrValue >> 8) & 0xFF);
      Serial.print(' '); Serial.print(ldrValue & 0xFF);
      Serial.print(") ");
      Serial.println(ldrValue);
      writeI2cInt(ldrValue);
      break;
    case NO_ACTION:                             // with SPI: will be sent from master
following READ_SWITCH to get button state
      break;
    default:
      Serial.print(F("Slave: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("I2C communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: SDA<->SDA, SCL<->SCL (SDA/A4, SCL/A5 on UNO)"));
  Serial.println(F("Slave switches LED on/off according to command received (LED_ON,
LED_OFF)"));
  Serial.println(F("       reads own switch if requested and prepares result for next
transfer (READ_SWITCH)"));
  Serial.println(F("       reads LDR if requested, prepares 2 result bytes (MSB first)
for next transfers (READ_LDR)"));
  Serial.println(F("       prints LDR value sent to master (2 bytes, MSB folowed by LSB)
onto Serial"));
}

void setupI2cComm() {
  jobAvailable = false;                     // nothing received yet -> nothing to do
  Wire.begin(LEDS_I2C_ADDR);                // join i2c bus with address #4
  Wire.onReceive(receiveEvent);             // register event handler for receiving data
  Wire.onRequest(requestEvent);       // register event handler for answering requests
}

void writeI2cByte(byte value) {
  noInterrupts();
  unsentData[0] = value; // send the button state on next transfer to master
  unsentCount = 1;                              // remember count of unsent bytes
  interrupts();
}

void writeI2cInt(int value) {
  noInterrupts();
```

```
  // preserve low byte of LDR value in unsent data at index 0 (MSB first!)
  unsentData[0] = value & 0xFF;
  // send the high byte of the LDR value on next transfer to master
  unsentData[1] = (value >> 8) & 0xFF;
  unsentCount = 2;        // remember count of unsent bytes
  interrupts();
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  static byte receiveMultipleBytes = 0x00;
  static int receiveMultipleBytesAmount = 0;
  static byte receiveInt[2];

  if (Wire.available() > 0) // fetch received command
  {
    received = Wire.read(); // receive byte as a character

    // check if command is 'motor' ==> receiving bytes
    if (received == MOTOR) {
      receiveMultipleBytes = MOTOR;

    } else {

      // check if multiple bytes are incoming
      if (receiveMultipleBytes != 0x00) {

        // save byte to array and inrement counter
        receiveInt[receiveMultipleBytesAmount] = received;
        receiveMultipleBytesAmount++;

        // check if all bytes are received
        if (receiveMultipleBytesAmount == 2) {

          // reset
          receiveMultipleBytes = 0x00;
          receiveMultipleBytesAmount = 0;

          // build bytes to int
          int ldr = receiveInt[0] + receiveInt[1] * 256;

          Serial.print("MY LDR: ");
          Serial.println(ldr);
        }

      } else {
          // only one byte ==> no command attributes
          // trigger loop as new command to be processed is available
          jobAvailable = true;
      }
    }
  }
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  while (unsentCount > 0) {
    Wire.write(unsentData[--unsentCount]);
  }
}
```

CODE - MASTER.INO

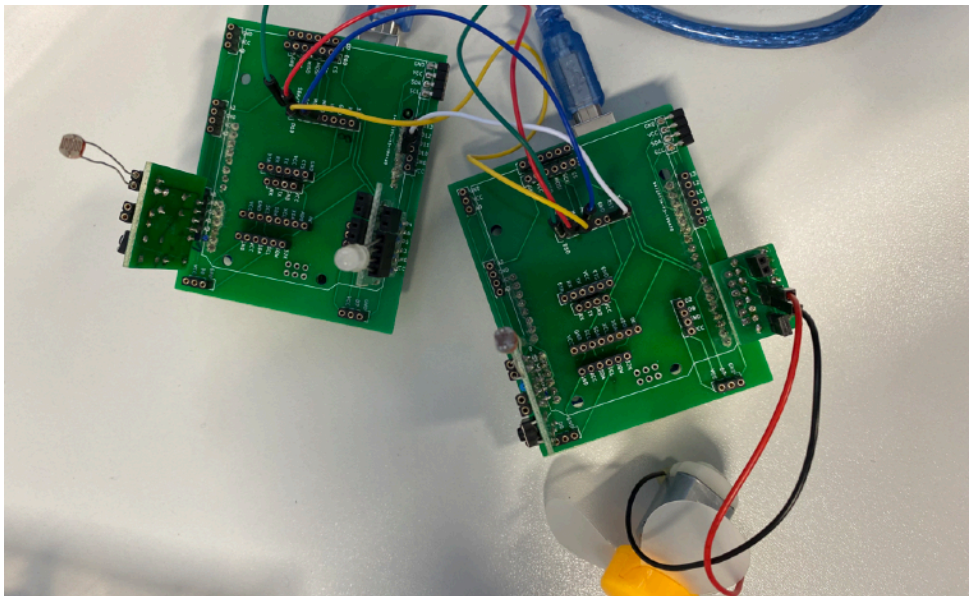
```
#include<Wire.h>                    // Library for SPI einbinden

#define LEDS_I2C_ADDR 4            // I2C address of LED, LDR, and tactile switch slave

#define PIN_LED       3            // LED on pin 3
#define PIN_SWITCH    2            // tactile switch on pin 2
#define PIN_LDR       A1

#define LED_OFF     0x01           // switch LED off command
#define LED_ON      0x02           // switch LED on command
#define READ_SWITCH 0x03           // read switch state command
#define READ_LDR    0x04
#define MOTOR       0x05
#define NO_ACTION   0xFF           // dummy command for 'do nothing'

#define READ_SWITCH_DELAY 1000     // delay between 2 read switch commands
#define READ_LDR_DELAY    2500     // delay between 2 read LDR commands

#define COMM_WRITE_READ_DELAY 1    // delay between write command and read back result

unsigned long previousMillis = 0;
unsigned long currentMillis = 0;
const long interval = 1000;

volatile byte unsentData[2];
volatile int unsentCount;           // initialized to 0 to indicate no data belonging to
previous command is unsent

void setup(void) {
  Serial.begin(115200);            // start Serial Communication at Baud Rate 115200
  pinMode(PIN_LED, OUTPUT);               // set LED pin as output

  setupI2cComm();

  printInfo();             // print usage message onto device connected to HW Serial
  //delay(1000);      // wait a moment to avoid unwanted flooding log at the beginning
}

void loop(void) {
  doSwitch();
  readSlaveSwitch();
  readSlaveLDR();

  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    writeLDR();
  }

}

void readSlaveSwitch() {
  static unsigned long prevReadSwitchCmd = 2000L; // time of most recent read switch
command
  byte received = 0;                                // byte received upon read switch
command
  unsigned long now = millis();               // milliseconds from power up

  if (now - READ_SWITCH_DELAY > prevReadSwitchCmd) {
    prevReadSwitchCmd = now;
    //Serial.println(F("Master: read switch"));
    if (requestI2cByte(READ_SWITCH, &received)) {               // slave may send less
than requested
      doCommand(received);
    }
  }
  //delay(10);
}

void readSlaveLDR() {
```

```
  static unsigned long prevReadLDRCmd = 2000L;   // time of most recent read switch
command
  byte received[2];                             // byte received upon read switch command
  unsigned int ldrValue;
  unsigned long now = millis();                 // milliseconds from power up

  if (now - READ_LDR_DELAY > prevReadLDRCmd) {
    prevReadLDRCmd = now;
    //Serial.println(F("Master: read LDR"));
    if (requestI2cInt(READ_LDR, &ldrValue)) {               // slave may send less than
requested
      Serial.println(ldrValue);
    }
  }
  //delay(10);
}

void doSwitch() {
  static boolean prevSwitchState = false;      // previous state of tactile switch (to
reduce repeating messages)
  boolean switchState = digitalRead(PIN_SWITCH);
  if (switchState != prevSwitchState) {
    prevSwitchState = switchState;
    writeI2cByte(switchState ? LED_ON : LED_OFF);
  }
}

void doCommand(byte received) {
  static boolean prevLedState = false;       // previous state of LED to reduce
dig.Write and Serial.print
  switch (received) {
    case LED_ON:                             // set the LED state depending upon command
received from slave
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);         //Sets LED pin HIGH
        Serial.println(F("Master: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);          //Sets LED pin LOW
        Serial.println(F("Master: LED off"));
        prevLedState = false;
      }
      break;
    case NO_ACTION:
      break;
    default:
      Serial.print(F("Master: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("I2C communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: SDA<->SDA, SCL<->SCL (SDA/A4, SCL/A5 on UNO)"));
  Serial.println(F("Master eagerly reads own tactile switch and sends changes to slave
(LED_ON, LED_OFF)"));
  Serial.println(F("       requests tactile switch of slave every second
(READ_SWITCH)"));
  Serial.println(F("       requests LDR value of slave every 2 and a half second
(READ_LDR)"));
  Serial.println(F("       fetches values from slave (>= 1 byte) by sending according
number of NO_ACTION commands"));
  Serial.println(F("       builds result from multiple byte responses (MSB first)
(LDR)"));
  Serial.println(F("       switches own LED according to slaves answer (LED_ON,
LED_OFF)"));
```

```
  Serial.println(F("       prints LDR value received from slave (2 bytes, MSB followed
by LSB) onto Serial"));
}

void setupI2cComm() {
  Wire.begin();                           // begin the I2C commnuication
}

void writeI2cByte(byte value) {
  Wire.beginTransmission(LEDS_I2C_ADDR); //Starts communication with to device number
LEDS_I2C_ADDR
  Wire.write(value);       // sends command byte LED_ON or LED_OFF to slave
  Wire.endTransmission();          // stop transmitting -> transfer data
}

boolean requestI2cByte(byte command, byte *value) {
  boolean success = false;
  Wire.beginTransmission(LEDS_I2C_ADDR);
  Wire.write(command);           // send read switch request READ_SWITCH to slave
  Wire.endTransmission();
  delay(COMM_WRITE_READ_DELAY);            // give slave time to accomplish work
  Wire.requestFrom(LEDS_I2C_ADDR, 1);     // request 1 byte from slave device (tactile
switch state LED_ON or LED_OFF)
  success = Wire.available() >= 1;
  if (success) {                // slave may send less than requested
    *value = Wire.read();                // receive 1 byte from slave
  }
  return success;
}

boolean requestI2cInt(byte command, int *value) {
  boolean success = false;
  byte received[2];
  Wire.beginTransmission(LEDS_I2C_ADDR);
  Wire.write(command);                       // send read switch request READ_LDR to slave
  Wire.endTransmission();
  delay(COMM_WRITE_READ_DELAY);             // give slave time to accomplish work
  Wire.requestFrom(LEDS_I2C_ADDR, 2);  // request 2 byte from slave device (10 bit LDR)
  success = Wire.available() >= 2;
  if (success) {              // slave may send less than requested
    received[0] = Wire.read();       // receive MSB byte of LDR value from slave
    received[1] = Wire.read();       // receive LSB byte of LDR value from slave
    *value = received[0] * 256 + received[1]; // build int from 2 bytes
    Serial.print(F("Master: LDR: "));
    Serial.print('('); Serial.print(received[0]);
    Serial.print(' '); Serial.print(received[1]);
    Serial.print(") ");
  }
  return success;
}

void writeI2cInt1(int value) {
  unsentData[0] = value & 0xFF;        // preserve low byte of LDR value in unsent data
at index 0 (MSB first!)
  unsentData[1] = (value >> 8) & 0xFF; // send the high byte of the LDR value on next
transfer to master
  unsentCount = 2;                   // remember count of unsent bytes

  writeI2cByte(unsentData[0]);
  writeI2cByte(unsentData[1]);
}

void writeLDR() {
    int ldrValue = analogRead(PIN_LDR);
    Serial.println(ldrValue);
    writeI2cByte(MOTOR);
    writeI2cInt1(ldrValue);
}
```

Serial Peripheral Interface (SPI)

VERKABELUNG & AUFBAU



Master auf Slave:

- MOSI – MOSI
- MISO – MISO
- SCK – SCK
- SDA/SS - SDA-SS
- GND - GND

CODE - SLAVE.INO

```
#include<SPI.h>

#define PIN_LED    4
#define PIN_SWITCH 2
#define PIN_LDR    A1

#define LED_OFF     0x01
#define LED_ON      0x02
#define READ_SWITCH 0x03
#define READ_LDR    0x04
#define NO_ACTION   0xFF
#define WRITE_LDR   0x05

#define MOTOR_A     10
#define MOTOR_B     11
#define MOTOR_FWD   12
#define MOTOR_BWD   13

volatile byte received;
volatile boolean jobAvailable;
volatile byte unsentData[2];
volatile int unsentCount = 0;           // initialized to 0 to indicate no data
belonging to previous command is unsent
volatile int unreceived;
byte receivedInt[2];

int sensorValue = 0;
int motorValue = 0;
```

```
void setup()
{
  Serial.begin(115200);
  pinMode(PIN_LED, OUTPUT);              // setting LED pin as OUTPUT

  // set output mode on motor
  pinMode(MOTOR_A, OUTPUT);
  pinMode(MOTOR_B, OUTPUT);
  pinMode(MOTOR_FWD, OUTPUT);
  pinMode(MOTOR_BWD, OUTPUT);

  // set default motor rotation to forward
  digitalWrite(MOTOR_FWD, HIGH);

  setupSpiComm();

  printInfo();                 // print usage message onto device connected to HW Serial
}

void loop() {
  if (jobAvailable) {                    // react to command received from master
    noInterrupts();
    jobAvailable = false;                // indicate no message to process
    interrupts();

    doCommand();                         // process command received from master
  }
}

void doCommand() {
  static boolean prevLedState = false;
  int ldrValue;
  switch (received) {                          // distinquish command received
    case LED_ON:
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);             // sets LED pin HIGH to switch LED on
        Serial.println(F("Slave: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);              // sets LED pin LOW to switch LED off
        Serial.println(F("Slave: LED off"));
        prevLedState = false;
      }
      break;
    case READ_SWITCH:
      //Serial.println(F("Slave: read switch"));
      writeSpiByte(digitalRead(PIN_SWITCH) ? LED_ON : LED_OFF);
      break;
    case READ_LDR:
      //Serial.println(F("Slave: read LDR"));
      ldrValue = analogRead(PIN_LDR);
      Serial.print(F("Slave: LDR: "));
      Serial.print('('); Serial.print((ldrValue >> 8) & 0xFF);
      Serial.print(' '); Serial.print(ldrValue & 0xFF);
      Serial.print(") ");
      Serial.println(ldrValue);
      writeSpiInt(ldrValue);
      break;
    case WRITE_LDR:
      sensorValue = receivedInt[0] * 256 + receivedInt[1];
      motorValue = map(sensorValue, 45, 1024, 0, 255);
      analogWrite(MOTOR_A, motorValue);
      break;
    case NO_ACTION:                            // with SPI: will be sent from master
following READ_SWITCH to get button state
      break;
```

```
default:
      Serial.print(F("Slave: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("SPI communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: MOSI-MOSI, MISO-MISO, SCK-SCK, SS-SS (pin 10 on UNO)"));
  Serial.println(F("Slave switches LED on/off according to command received (LED_ON,
LED_OFF)"));
  Serial.println(F("       reads own switch if requested and prepares result for next
transfer (READ_SWITCH)"));
  Serial.println(F("reads LDR if requested, prepares 2 result bytes (MSB first) for next
transfers (READ_LDR)"));
  Serial.println(F("prints LDR value sent to master (2 bytes, MSB folowed by LSB) onto
Serial"));
}

void setupSpiComm() {
  pinMode(MISO, OUTPUT);       // sets MISO as OUTPUT (Have to Send data to Master IN)

  jobAvailable = false;                    // nothing received yet -> nothing to do
  SPCR |= _BV(SPE);                        // turn on SPI in Slave Mode

  SPI.attachInterrupt();                   // interrupt ON is set for SPI commnucation
}

void writeSpiByte(byte value) {
  noInterrupts();
  SPDR = value; // send the button state on next transfer to master via SPDR
  interrupts();
}

void writeSpiInt(int value) {
  noInterrupts();
  unsentData[0] = value & 0xFF;    // preserve low byte of LDR value in unsent data at
index 0
  unsentCount = 1;                           // remember count of unsent bytes
  SPDR = (value >> 8) & 0xFF; // send the low byte of the LDR value on next transfer to
master via SPDR
  interrupts();
}

ISR (SPI_STC_vect) {    // interrupt service routine function
  if (unreceived == 0) {
    received = SPDR;     // value received from master is store in variable received
    if (received == WRITE_LDR){
      unreceived = 2;
      SPDR = NO_ACTION;
    } else if (unsentCount > 0) {
      SPDR = unsentData[--unsentCount]; // send first unsent byte and reduce counter of
unsent bytes
    } else {
      SPDR = NO_ACTION;  // indicate nothing to be done on master (will be sent at next
transfer)
      jobAvailable = true;  // trigger loop as new command to be processed is available
    }

  } else {
    if(unreceived == 2){
      receivedInt[0] = SPDR;
    } else{
      receivedInt[1] = SPDR;
      jobAvailable = true;   // trigger loop as new command to be processed is available
    }
    SPDR = NO_ACTION;  // indicate nothing to be done on master
    unreceived--;
   }
}
```

CODE - MASTER.INO

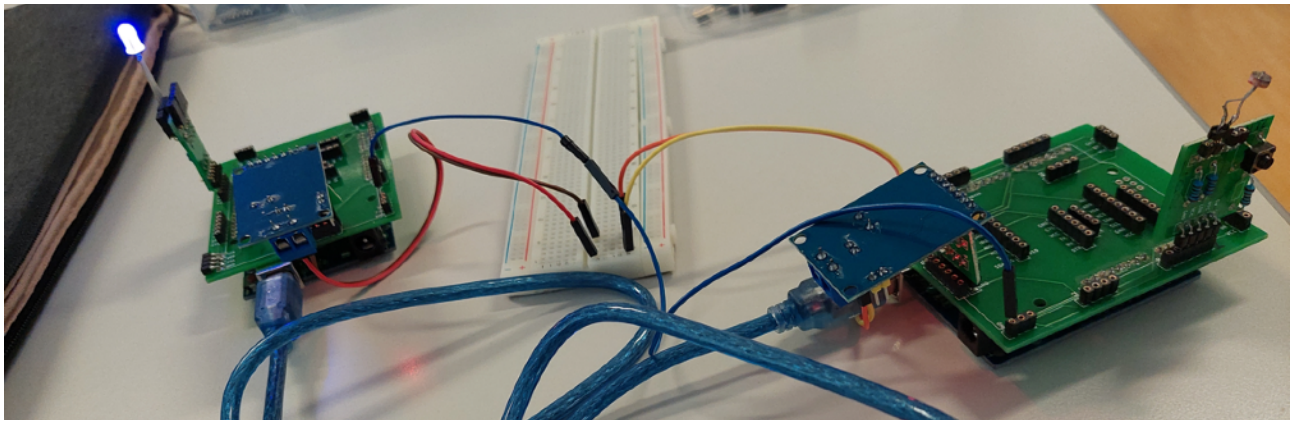
```
#include<SPI.h>                  // Library for SPI einbinden

#define PIN_LED       4          // LED on pin 4
#define PIN_SWITCH    2          // tactile switch on pin 2
#define PIN_LDR       A1

#define LED_OFF     0x01         // switch LED off command
#define LED_ON      0x02         // switch LED on command
#define READ_SWITCH 0x03         // read switch state command
#define READ_LDR    0x04
#define WRITE_LDR   0x05         //write ldr value to slave
#define NO_ACTION   0xFF         // dummy command for 'do nothing'

#define READ_SWITCH_DELAY 1000   // delay between 2 read switch commands
#define READ_LDR_DELAY    2500   // delay between 2 read LDR commands
#define WRITE_LDR_DELAY   2000

#define COMM_WRITE_READ_DELAY 2  // delay between write command and read back result

unsigned long now = 0;
unsigned long ldrTime = 2000L;
int currentLdrValue = 0;

void setup(void) {
  Serial.begin(115200);       // start Serial Communication at Baud Rate 115200
  pinMode(PIN_LED, OUTPUT);              // set LED pin as output

  setupSpiComm();

  printInfo();   // print usage message onto device connected to HW Serial
  //delay(1000);   // wait a moment to avoid unwanted flooding log at the beginning
}

void loop(void) {
  /*
  doSwitch();
  readSlaveSwitch();
  readSlaveLDR();
  */

  // ldr send every 2 seconds
  now = millis();
  if (now - WRITE_LDR_DELAY > ldrTime){
    currentLdrValue = analogRead(PIN_LDR);
    writeSPIInt(currentLdrValue);
    ldrTime = now;
  }
}

void readSlaveSwitch() {
  static unsigned long prevReadSwitchCmd = 2000L; // time of most recent read switch cmd
  byte received = 0;               // byte received upon read switch command
  unsigned long now = millis();               // milliseconds from power up

  if (now - READ_SWITCH_DELAY > prevReadSwitchCmd) {
    prevReadSwitchCmd = now;
    //Serial.println(F("Master: read switch"));
    requestSpiByte(READ_SWITCH, &received);
    doCommand(received);
  }
  //delay(10);
}

void readSlaveLDR() {
  static unsigned long prevReadLDRCmd = 2000L;   // time of most recent read switch cmd
  byte received[2];                           // byte received upon read switch command
  unsigned int ldrValue;
  unsigned long now = millis();               // milliseconds from power up
```

```
  if (now - READ_LDR_DELAY > prevReadLDRCmd) {
    prevReadLDRCmd = now;
    //Serial.println(F("Master: read LDR"));
    requestSpiInt(READ_LDR, &ldrValue);
    Serial.println(ldrValue);
  }
  //delay(10);
}

void doSwitch() {
  static boolean prevSwitchState = false;   // previous state of tactile switch
  boolean switchState = digitalRead(PIN_SWITCH);
  if (switchState != prevSwitchState) {
    prevSwitchState = switchState;
    writeSpiByte(switchState ? LED_ON : LED_OFF);
  }
}

void doCommand(byte received) {
  static boolean prevLedState = false; // previous state of LED to reduce dig.Write
  switch (received) {
    case LED_ON:    // set the LED state depending upon command received from slave
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);         //Sets LED pin HIGH
        Serial.println(F("Master: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);          //Sets LED pin LOW
        Serial.println(F("Master: LED off"));
        prevLedState = false;
      }
      break;
    case NO_ACTION:
      break;
    default:
      Serial.print(F("Master: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("SPI communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: MOSI-MOSI, MISO-MISO, SCK-SCK, SS-SS (pin 10 on UNO)"));
  Serial.println(F("Master eagerly reads own tactile switch and sends changes to slave
(LED_ON, LED_OFF)"));
  Serial.println(F("requests tactile switch of slave every second (READ_SWITCH)"));
  Serial.println(F("requests LDR value of slave every 2 and a half second (READ_LDR)"));
  Serial.println(F("fetches values from slave (>= 1 byte) by sending according number of
NO_ACTION commands"));
  Serial.println(F("builds result from multiple byte responses (MSB first) (LDR)"));
  Serial.println(F("switches own LED according to slaves answer (LED_ON, LED_OFF)"));
  Serial.println(F("prints LDR value received from slave (2 bytes, MSB followed by LSB)
onto Serial"));
}

void setupSpiComm() {
  SPI.begin();                              // begin the SPI commnuication
  SPI.setClockDivider(SPI_CLOCK_DIV8);   // set clock for SPI communication at 8
  digitalWrite(SS, HIGH);           // set SlaveSelect to HIGH (deactivates slave)
}

void writeSpiByte(byte value) {
  digitalWrite(SS, LOW);    //Starts communication with Slave connected to master
  /*value = */
   SPI.transfer(value); //Send the button state to slave, ignore synchronous response
  digitalWrite(SS, HIGH); //Ends communication with Slave connected to master
}
```

```
void requestSpiByte(byte command, byte *value) {
  digitalWrite(SS, LOW);        // start communication to Slave connected to SS pin
  /* var = */SPI.transfer(command); // send read switch request to slave (
  digitalWrite(SS, HIGH);       // end communication with Slave connected to SS pin
  delay(1);                           // give slave time to accomplish work
  digitalWrite(SS, LOW);              // start communication with same Slave
  *value = SPI.transfer(NO_ACTION);     // get button state from same Slave
  digitalWrite(SS, HIGH);             // end communication to same Slave
}

void requestSpiInt(byte command, int *value) {
  byte received[2];
  digitalWrite(SS, LOW);          // start communication to Slave connected to SS pin
  /* var = */SPI.transfer(command);    // send read LDR request to slave
  digitalWrite(SS, HIGH);         // end communication with Slave connected to SS pin
  delay(COMM_WRITE_READ_DELAY);             // give slave time to accomplish work
  digitalWrite(SS, LOW);              // start communication with same Slave
  received[0] = SPI.transfer(NO_ACTION);  // get high byte of LDR value from same Slave
  delay(COMM_WRITE_READ_DELAY);
  received[1] = SPI.transfer(NO_ACTION);    // get low byte of LDR value from same Slave
  digitalWrite(SS, HIGH);             // end communication to same Slave
  *value = received[0] * 256 + received[1]; // build int from 2 bytes
  Serial.print(F("Master: LDR: "));
  Serial.print('('); Serial.print(received[0]);
  Serial.print(' '); Serial.print(received[1]);
  Serial.print(") ");
}

void writeSPIInt(int value) {
  writeSpiByte(WRITE_LDR);
  writeSpiByte((value >> 8) & 0xFF);
  writeSpiByte((value & 0xFF));
}
```

Controller Area Network (CAN)

VERKABELUNG & AUFBAU



Master auf Slave:
- H des Masters auf H des Slaves
- L des Masters auf L des Slaves

CODE - SLAVE.INO

```
#include <SPI.h>
#include <mcp2515.h>

#define PIN_LED    4
#define PIN_SWITCH 2
#define PIN_LDR    A1

#define LED_OFF     0x01
#define LED_ON      0x02
#define READ_SWITCH 0x03
#define READ_LDR    0x04
#define NO_ACTION   0xFF
#define WRITE_LDR   0x05

#define MOTOR_A     10
#define MOTOR_B     11
#define MOTOR_FWD   12
#define MOTOR_BWD   13

volatile byte received;

MCP2515 mcp2515(7);
struct can_frame canMsg;

int sensorValue = 0;
int motorValue = 0;

void setup() {
  Serial.begin(115200);
  pinMode(PIN_LED, OUTPUT);              // setting LED pin as OUTPUT

  setupCanComm();

  // set output mode on motor
  pinMode(MOTOR_A, OUTPUT);
  pinMode(MOTOR_B, OUTPUT);
  pinMode(MOTOR_FWD, OUTPUT);
  pinMode(MOTOR_BWD, OUTPUT);

  // set default motor rotation to forward
  digitalWrite(MOTOR_FWD, HIGH);
}
```

```
void loop() {
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) // fetch received command
  {
    received = canMsg.data[0];            // receive byte as a character
   //    switch (canMsg.can_id) {
    //      case 0x0F6:
    //        break;
    //      case 0x036:
    //        break;
    //    } // switch

    doCommand();                          // process command received from master
  }
}

void doCommand() {
  static boolean prevLedState = false;
  int ldrValue;
  switch (received) {                           // distinquish command received
    case LED_ON:
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);            // sets LED pin HIGH to switch LED on
        Serial.println(F("Slave: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);             // sets LED pin LOW to switch LED off
        Serial.println(F("Slave: LED off"));
        prevLedState = false;
      }
      break;
    case READ_SWITCH:
      //Serial.println(F("Slave: read switch"));
      writeCanByte(digitalRead(PIN_SWITCH) ? LED_ON : LED_OFF);
      break;
    case READ_LDR:
      //Serial.println(F("Slave: read LDR"));
      ldrValue = analogRead(PIN_LDR);
      Serial.print(F("Slave: LDR: "));
      Serial.print('('); Serial.print((ldrValue >> 8) & 0xFF);
      Serial.print(' '); Serial.print(ldrValue & 0xFF);
      Serial.print(") ");
      Serial.println(ldrValue);
      writeCanInt(ldrValue);
      break;
    case WRITE_LDR:
      int sensorValue = 0;
      receiveCanInt(&sensorValue);
      motorValue = map(sensorValue, 45, 1024, 0, 255);
      analogWrite(MOTOR_A, motorValue);
      break;
    case NO_ACTION:                           // with SPI: will be sent from master
following READ_SWITCH to get button state
      break;
    default:
      Serial.print(F("Slave: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("CAN communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: MCP2515 CAN-Connector, CS/SS 10, 125KBPS"));
  Serial.println(F("Slave switches LED on/off according to command received (LED_ON,
LED_OFF)"));
  Serial.println(F("       reads own switch if requested and prepares result for next
transfer (READ_SWITCH)"));
```

```
  Serial.println(F("       reads LDR if requested, prepares 2 result bytes (MSB first)
for next transfers (READ_LDR)"));
  Serial.println(F("       prints LDR value sent to master (2 bytes, MSB folowed by LSB)
onto Serial"));
}

void setupCanComm() {
  SPI.begin();

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();
}

void writeCanByte(byte value) {
  canMsg.can_id = 0x06F;
  canMsg.can_dlc = 1;
  canMsg.data[0] = value;
  mcp2515.sendMessage(&canMsg); // send the button state to master
}

void writeCanInt(int value) {
  canMsg.can_id = 0x06F;
  canMsg.can_dlc = 2;
  canMsg.data[0] = (value >> 8) & 0xFF; // send the high byte of the LDR value to master
first
  canMsg.data[1] = value & 0xFF;        // send low byte of LDR value to master second
(MSB first!)
  mcp2515.sendMessage(&canMsg);            // send the LDR value to master
}

boolean receiveCanInt(int *value) {
  boolean success = true;
  delay(5);             // give slave time to accomplish work (1 is not enough)
  if (success) {              // slave may send less than requested
    *value = canMsg.data[1] * 256 + canMsg.data[2]; // build int from 2 bytes (MSB @ 0,
LSB @ 1
    Serial.print(F("Master: LDR: "));
    Serial.print('('); Serial.print(*value);
    Serial.println(") ");
  }
  return success;
}
```

CODE - MASTER.INO

```
#include <stdlib.h>
#include <SPI.h>
#include <mcp2515.h>

#define PIN_LED       3           // LED on pin 3
#define PIN_SWITCH    2           // tactile switch on pin 2
#define PIN_LDR       A1

#define LED_OFF     0x01          // switch LED off command
#define LED_ON      0x02          // switch LED on command
#define READ_SWITCH 0x03          // read switch state command
#define READ_LDR    0x04
#define NO_ACTION   0xFF          // dummy command for 'do nothing'
#define WRITE_LDR   0x05

#define WRITE_LDR_DELAY   2000    // delay between 2 write LDR commands
#define READ_SWITCH_DELAY 1000    // delay between 2 read switch commands
#define READ_LDR_DELAY    2500    // delay between 2 read LDR commands

#define COMM_WRITE_READ_DELAY 5   // delay between write command and read back result

MCP2515 mcp2515(7);
struct can_frame canMsg[3];

unsigned long now = 0;
unsigned long lastWrite = 2000L;

int currentSensorValue = 0;

void setup(void) {
  Serial.begin(115200);            // start Serial Communication at Baud Rate 115200
  pinMode(PIN_LED, OUTPUT);               // set LED pin as output

  setupCanComm();
  buildCanMsgs();

  printInfo();              // print usage message onto device connected to HW Serial
  //delay(1000);       // wait a moment to avoid unwanted flooding log at the beginning
}

void loop(void) {
  /*doSwitch();
  readSlaveSwitch();
  readSlaveLDR();*/

  now = millis();
  //If time is over, send value again
  if(now - WRITE_LDR_DELAY > lastWrite){
    currentSensorValue = analogRead(PIN_LDR);
    writeCanLDR(currentSensorValue);
    lastWrite = now;
 }
}

void readSlaveSwitch() {
  static unsigned long prevReadSwitchCmd = 2000L; // time of most recent read switch cmd
  byte received = 0;                    // byte received upon read switch command
  unsigned long now = millis();               // milliseconds from power up

  if (now - READ_SWITCH_DELAY > prevReadSwitchCmd) {
    prevReadSwitchCmd = now;
    //Serial.println(F("Master: read switch"));
    if (requestCanByte(READ_SWITCH, &received)) {  // slave may send less than requested
      doCommand(received);
    }
  }
  //delay(10);
}
```

```
void readSlaveLDR() {
  static unsigned long prevReadLDRCmd = 2000L;   // time of most recent read switch cmd
  byte received[2];                              // byte received upon read switch command
  unsigned int ldrValue;
  unsigned long now = millis();                  // milliseconds from power up

  if (now - READ_LDR_DELAY > prevReadLDRCmd) {
    prevReadLDRCmd = now;
    //Serial.println(F("Master: read LDR"));
    if (requestCanInt(READ_LDR, &ldrValue)) {   // slave may send less than requested
      Serial.println(ldrValue);
    }
  }
  //delay(10);
}

void doSwitch() {
  static boolean prevSwitchState = false;      // previous state of tactile switch (to
reduce repeating messages)
  boolean switchState = digitalRead(PIN_SWITCH);
  if (switchState != prevSwitchState) {
    prevSwitchState = switchState;
    writeCanByte(switchState ? LED_ON : LED_OFF);
  }
}

void doCommand(byte received) {
  static boolean prevLedState = false;  // previous state of LED to reduce dig.Write
  switch (received) {
    case LED_ON:      // set the LED state depending upon command received from slave
      if (!prevLedState) {
        digitalWrite(PIN_LED, HIGH);         //Sets LED pin HIGH
        Serial.println(F("Master: LED on"));
        prevLedState = true;
      }
      break;
    case LED_OFF:
      if (prevLedState) {
        digitalWrite(PIN_LED, LOW);          //Sets LED pin LOW
        Serial.println(F("Master: LED off"));
        prevLedState = false;
      }
      break;
    case NO_ACTION:
      break;
    default:
      Serial.print(F("Master: undefined command 0x"));
      Serial.println(received, HEX);
  }
}

void printInfo() {
  Serial.println(F("CAN communication between two Arduino UNO (Master and slave)"));
  Serial.println(F("Master: tactile switch on pin 2, LED on pin 3"));
  Serial.println(F("Slave: tactile swich on pin 2, LED on pin 4, LDR on pin A1"));
  Serial.println(F("Connection: MCP2515 CAN-Connector, CS/SS 10, 125KBPS"));
  Serial.println(F("Master eagerly reads own tactile switch and sends changes to slave
(LED_ON, LED_OFF)"));
  Serial.println(F("requests tactile switch of slave every second (READ_SWITCH)"));
  Serial.println(F("requests LDR value of slave every 2 and a half second (READ_LDR)"));
  Serial.println(F("fetches values from slave (>= 1 byte) by sending according number of
NO_ACTION commands"));
  Serial.println(F("builds result from multiple byte responses (MSB first) (LDR)"));
  Serial.println(F("switches own LED according to slaves answer (LED_ON, LED_OFF)"));
  Serial.println(F("prints LDR value received from slave (2 bytes, MSB followed by LSB)
onto Serial"));
}

void setupCanComm() {
  SPI.begin();

  mcp2515.reset();
```

```
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();
}

void writeCanByte(byte value) {
  canMsg[1].data[0] = value;
  mcp2515.sendMessage(&canMsg[1]); // sends command byte LED_ON or LED_OFF to slave
}

boolean requestCanByte(byte command, byte *value) {
  boolean success = false;
  canMsg[0].data[0] = command;
  mcp2515.sendMessage(&canMsg[0]);   // send read switch request READ_SWITCH to slave
  delay(COMM_WRITE_READ_DELAY);               // give slave time to accomplish work
  success = mcp2515.readMessage(&canMsg[2]) == MCP2515::ERROR_OK;
  if (success) {                // slave may send less than requested
    *value = canMsg[2].data[0];           // receive 1 byte from slave
  }
  return success;
}

boolean requestCanInt(byte command, int *value) {
  boolean success = false;
  canMsg[0].data[0] = command;
  mcp2515.sendMessage(&canMsg[0]);     // send read switch request READ_LDR to slave
  delay(COMM_WRITE_READ_DELAY);  // give slave time to accomplish work (1 is not enough)
  success = mcp2515.readMessage(&canMsg[2]) == MCP2515::ERROR_OK;
  if (success) {              // slave may send less than requested
    *value = canMsg[2].data[0] * 256 + canMsg[2].data[1]; // build int from 2 bytes
    Serial.print(F("Master: LDR: "));
    Serial.print('('); Serial.print(canMsg[2].data[0]);
    Serial.print(' '); Serial.print(canMsg[2].data[1]);
    Serial.print(") ");
  }
  return success;
}

void buildCanMsgs() {
  // lowest can_id (0x00) has highest priority / is dominant
  // highest can_id (0xFF) has lowest priority / is recessive
  // Request switch state / LDR value command
  canMsg[0].can_id  = 0x0F6;
  canMsg[0].can_dlc = 8;
  canMsg[0].data[0] = 0x8E;
  canMsg[0].data[1] = 0x87;
  canMsg[0].data[2] = 0x32;
  canMsg[0].data[3] = 0xFA;
  canMsg[0].data[4] = 0x26;
  canMsg[0].data[5] = 0x8E;
  canMsg[0].data[6] = 0xBE;
  canMsg[0].data[7] = 0x86;

  // switch LED on/off command
  canMsg[1].can_id  = 0x036;
  canMsg[1].can_dlc = 8;
  canMsg[1].data[0] = 0x0E;
  canMsg[1].data[1] = 0x00;
  canMsg[1].data[2] = 0x00;
  canMsg[1].data[3] = 0x08;
  canMsg[1].data[4] = 0x01;
  canMsg[1].data[5] = 0x00;
  canMsg[1].data[6] = 0x00;
  canMsg[1].data[7] = 0xA0;

  //Send LDR Value to Slave
  canMsg[2].can_id = 0x01;
  canMsg[2].can_dlc = 3;
  canMsg[2].data[0] = 0x0E;
  canMsg[2].data[1] = 0x00;
  canMsg[2].data[2] = 0xA0;
}
```

```
void writeCanLDR(int value){
  Serial.println(value);

  canMsg[2].data[0] = WRITE_LDR;
  canMsg[2].data[1] = (value >> 8) & 0xFF; // send the high byte of the LDR value to
master first
  canMsg[2].data[2] = value & 0xFF;      // send low byte of LDR value to master second
(MSB first!)
  mcp2515.sendMessage(&canMsg[2]); // send the LDR value to master
}
```