

Praktikum zur Lehrveranstaltung

Mikrorechentechnik II

Versuch

Kommunikation mit Raspberry Pi

Ziel dieses Versuchs ist es, Kenntnisse und Fähigkeiten zur Netzwerkprogrammierung und zur hardwarenahen Programmierung eines Computers am Beispiel des Raspberry Pi zu vermitteln.

Raspberry Pi ist ein single-board Computer im Kreditkartenformat, der trotz geringer Größe alle wesentlichen Elemente eines modernen Computers (CPU, RAM, Speicher, Ethernet, I/O-interface, Video-Controller, USB) enthält. Er eignet sich daher besonders für den o.g. Einsatzzweck und wurde aus diesem Grund als Hardwarebasis für das Praktikum Mikrorechentechnik ausgewählt.

Die Aufgabe im Rahmen dieses Versuchs besteht darin, ein Client- und ein Serverprogramm in der Programmiersprache C zu erstellen, die über das Ethernet-Netzwerk kommunizieren sollen. Dabei liest das Client-Programm periodisch Temperaturwerte von einem Sensor ein, wandelt diese in ein geeignetes Format und sendet sie an das Server-Programm. Dieses nimmt die Daten entgegen und gibt sie in geeigneter Weise auf einem OLED-Matrixdisplay aus.

Die Aufgabenstellung beinhaltet folgende Teilaufgaben:

- Senden und Empfangen von Daten über das Ethernet-Netzwerk-Interface
- Abfragen des Temperatursensors über das SPI-Interface des Raspberry Pi
- Ansteuerung eines OLED-Displays über das I2C-Interface des Raspberry Pi

Zur Lösung dieser Teilaufgaben ist es erforderlich, die entsprechenden Hardwaredokumentationen zu SPI-, I2C- und Netzwerkinterface des Raspberry Pi sowie die zugehörigen Dokumentationen zu den benötigten Softwarebibliotheken zu verstehen.

Zur Einstimmung auf die Aufgabenstellung können folgende Dokumente studiert werden:

1. SPI and I2C auf Raspberry Pi
<https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial>
2. Externe Hardware: SSD1306, MCP3008 and TMP64
<https://www.adafruit.com/datasheets/SSD1306.pdf>
<https://www.adafruit.com/datasheets/MCP3008.pdf>
http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf
3. Dokumentation WiringPi-GPIO-Software Bibliothek
<http://wiringpi.com/reference/>

Versuchsanleitung zum Praktikumsversuch

Kommunikation mit Raspberry Pi

Inhaltsverzeichnis:

1. Versuchsplatzbeschreibung
 - 1.1 Nutzung von SPI- und I2C-Schnittstellen
 - 1.2 Aktivierung der Schnittstellen
 - 1.2.1 SPI-Interface
 - 1.2.2 I2C-Interfaces
2. Programmierung des Netzwerkinterfaces
 - 2.1 TCP und UDP
 - 2.2 Network Sockets
 - 2.3 UDP-Socket Programmierung in C unter Linux
 - 2.3.1 Einbinden der Header-Dateien
 - 2.3.2 Initialisierung eines UDP-Socket
 - 2.3.3 Adressinformationen
 - 2.3.4 Senden und Empfangen von Daten
3. Hinweise für die Arbeit am Praktikumsplatz
 - 3.1 Compilieren des Programms
 - 3.2 Debuggen des Programms
4. Hinweise zum Kolloquium
5. Quellen und weiterführende Dokumentationen

1. Versuchsplatzbeschreibung

Im Praktikumsversuch Kommunikation mit Raspberry Pi wird ein Raspberry Pi 2 Mod. B benutzt (Bild 1). Für den nutzerfreundlichen Zugang zu verschiedenen Schnittstellen und Ein-/Ausgabeports wird dieser um ein Zusatzmodul „Linker Kit Baseboard“ [1] (Bild 2) erweitert. Dieses Zusatzhardware-Paket enthält außer der Grundplatine diverse Bauelemente, wie Sensoren, Aktoren, LEDs sowie ein OLED-Display und wurde bereits im Praktikum MRT-I benutzt.



Bild 1: Raspberry Pi 2 Mod.B

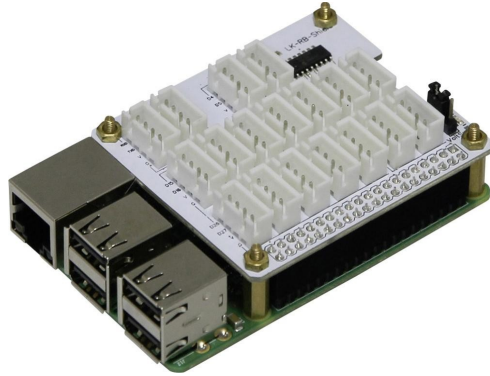


Bild 2: Raspberry Pi 2 incl. Linker Kit

Für den hier beschriebenen Versuchsaufbau finden neben der Platine ein Temperatursensor (Bild 3) sowie das OLED-Display (0,96 Zoll, 128*64) (Bild 4) Verwendung.



Bild 3: Temperatursensor

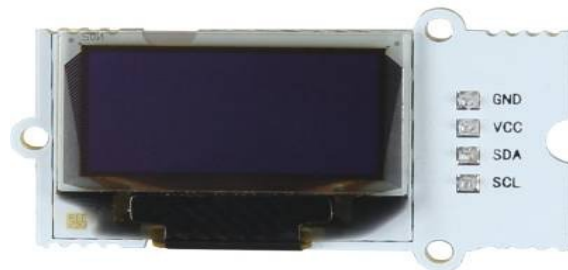


Bild 4: OLED-Display

Die einzelnen Bauelemente sind für das Praktikum vorkonfiguriert und über geeignete Adapterkabel mit den Schnittstellen verbunden. Das Pin-Layout der Linker-Kit-Platine incl. der vorhandenen Schnittstellen zeigt Bild 5.

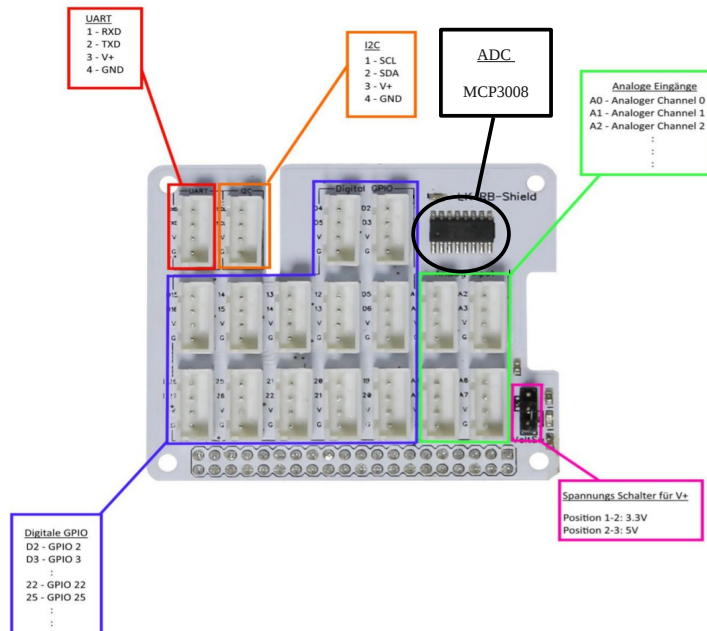


Bild 5: Pin-Belegung und Bedeutung der Schnittstellen bei Linker Kit Baseboard

1.1 Nutzung von SPI- und I2C- Schnittstellen

Auf dem LK-Baseboard gibt es vier Arten verschiedener Schnittstellen, nämlich analoge Eingänge, digitale Ein- und Ausgänge, I2C sowie UART Schnittstellen.

Außerdem gibt es einen 8-Kanal Analog/Digital-Konverter (MCP3008 ADC). Als Eingänge dafür werden die analogen Eingänge A0 bis A7 verwendet, der digitalisierte Ausgang kann über die SPI-Schnittstelle abgelesen werden. Bei diesem Praktikumsversuch werden A0 als analoger Eingang und SPI sowie I2C als Ausgänge verwendet. Der Temperatursensor ist also mit A0 verbunden, das OLED-Display mit I2C und der A/D-Wandler mit dem SPI-Interface.

1.2 Aktivierung der Schnittstellen

Die Aktivierung von SPI- sowie I2C-Schnittstellen sowie das Laden der zugehörigen Kernel Module können durch den Befehl **rasp-config** ausgeführt werden (siehe: „Advanced Options“). Durch den Befehl **lsmod** kann das erfolgreiche Laden der Module überprüft werden. Falls kein aktives Interface angezeigt wird, dann war die Aktivierung nicht erfolgreich.

```
$ lsmod | grep spi
$ lsmod | grep i2c
```

Um Peripherie-Anschlüsse im C Programm auf einfache Weise benutzen zu können, wird die Bibliothek wiringPi (<http://wiringpi.com/>) verwendet. Nach der Installation der Bibliothek kann man den folgenden Befehl benutzen, um eine Tabelle von Informationen der GPIO-Pins zu bekommen.

```
$ gpio readall
```

Um die Initialisierung von wiringPi durchzuführen, muss eine „Setup Funktion“ am Anfang des Programms aufgerufen werden.

```
// Beispiel Code:

#include <wiringPi.h>

if(wiringPiSetup () == -1) {
    printf("wiringP init does not work. exiting.\n");
    exit(1);
}
```

1.2.1 SPI-Interface

Die auf dem Gerät aktivierten SPI Kanäle können durch folgenden Befehl geprüft werden.

```
$ ls /dev | grep spi
# Output:
/dev/spidev0.0 /dev/spidev0.1
```

Die Ausgabe zeigt, dass zwei SPI-Kanäle 0 und 1 auf dem Raspberry Pi aktiviert sind. Falls der Temperatur Sensor mit der analogen Schnittstelle A0 verbunden ist, dann sind digitalisierte Daten von Kanal 0 auszulesen. Es stehen Funktionen zur direkten Verwendung von MCP3004 (ähnlich MCP3008, aber mit 4 Kanälen) in der wiringPi-Bibliothek zur Verfügung

(<https://github.com/hamishcunningham/wiringpi/blob/master/wiringPi/mcp3004.c>).

Um die digitalisierten Daten zu erhalten, muss eine Initialisierung des MCP3004 erfolgen, danach kann die **analogREAD** Funktion aufgerufen werden.

```
// Beispiel Code:

#include <mcp3004.h>

#define BASE 200 // wiringPi „device node“ BASE Nummer
/*
„Device node“ ist eine im wiringPi realisierte Struktur für die Nutzung einiger
allgemeiner Pin Funktionen, wie z.B. analogRead(), digitalWrite() usw. Falls
ein neuer „node“ erzeugt wird, können oben genannte Funktionen mit dieser BASE
Nummer als Parameter direkt aufgerufen werden. Siehe folgendes Beispiel:
*/
# define SPI_CHAN 0 // SPI Kanal

mcp3004Setup(BASE, SPI_CHAN);

digitalValue = analogRead(BASE); // digitalisierter Wert aus ADC
```

1.2.2 I2C-Interface

Für die Verwendung des OLED-Displays (verbunden mit der I2C Schnittstelle) steht ein Betriebsprogramm **ssd1306_i2c** zur Verfügung. Das Programm basiert auf den I2C-Funktionen von wiringPi. Um das Betriebsprogramm zu initialisieren, ist die Funktion **ssd1306I2CSetup** aufzurufen.

```
// Beispiel Code:

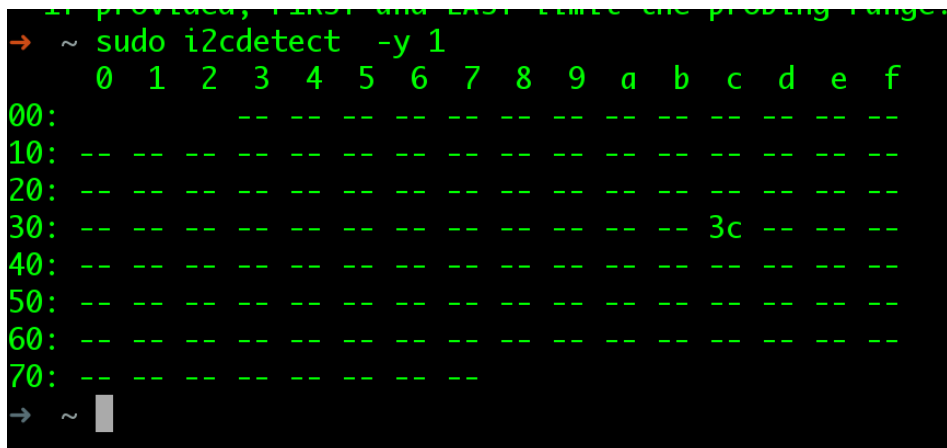
#include "ssd1306_i2c.h"
#define I2CADDRESS 0x3C

fd = ssd1306I2CSetup(I2CADDRESS); // I2C Adresse als Parameter
```

Durch den **i2cdetect** Befehl kann man die I2C-Adresse bestimmen.

```
# für raspberry pi 2
$ sudo i2cdetect -y 1
```

Dadurch wird eine Ergebnis in der folgenden Form erzeugt:



```
→ ~ sudo i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:      -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- 3c -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
→ ~ █
```

Daraus kann man ablesen, dass die I2C-Adresse den Hex-Wert 0x3c annimmt. Nach der Initialisierung der I2C Schnittstelle können die in der Bibliothek **ssd1306_i2c** realisierten Funktionen aufgerufen werden. Die Dokumentation zu diesen Funktionen sind in der Header-Datei **ssd1306_i2c.h** zu finden.

```
// Beispiel Code:

#include "ssd1306_i2c.h"
#define I2CADDRESS 0x3C

fd = ssd1306I2CSetup(I2CADDRESS); // I2C Adresse als Parameter

displayOn(fd); // schalten das Display ein
// zeichnen „hello world“ in zwei Zeilen
draw_line(1, 1, "hello");
draw_line(2, 1, "world");
updateDisplay(fd); // aktualisieren das Display
```

2. Programmierung des Netzwerkinterfaces

2.1 TCP und UDP

Für die Datenübertragung in Computernetzwerken gibt es zwei wichtige Protokolle in der sogenannten Transportschicht, die Bestandteile des Internet-Protokoll (IP) sind, nämlich das Transmission Control Protocol (TCP) und das User Datagram Protocol (UDP). TCP ist ein verbindungsorientiertes Protokoll, bei dem zuerst eine logische Verbindung zwischen zwei Hosts aufgebaut und nach der Datenübertragung wieder abgebaut werden muss. Damit garantiert TCP eine zuverlässige Übermittlung der Daten. Hierbei werden z.B. verlorene Datenpakete automatisch erneut gesendet. Im Vergleich dazu stellt UDP keine Sicherheitsmechanismen zur Verfügung. UDP ist verbindungslos und unzuverlässig, wodurch es allerdings auch einen geringen Verwaltungsaufwand und damit weniger Protokoll-Overhead benötigt. Da bei diesem Praktikumsversuch Paketverluste keine wesentliche Rolle spielen, wird einfach UDP verwendet.

2.2 Network Sockets

Network Sockets bilden eine standardisierte Schnittstelle (Application Programming Interface - API) zwischen einem Programm und der Netzwerkprotokoll-Implementierung des Betriebssystems. Ein Programm fordert einen Socket vom Betriebssystem an und kann über diesen Daten mit einem bestimmten Protokoll senden und auch empfangen. In unserem Beispiel ist demzufolge ein UDP-Socket für UDP-Datenübertragungen zu verwenden.

2.3 UDP-Socket Programmierung in der Sprache C unter Linux

2.3.1 Einbinden der Header-Dateien

Für den Aufruf der Socket-Funktionen von Linux müssen zuerst die benötigten Header-Dateien eingebunden werden.

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
```

2.3.2 Initialisierung eines UDP-Socket

Für den Aufbau eines UDP-Socket ist dieser zuerst vom Betriebssystem anzufordern.

```
int socket(int domain, int type, int protocol);
```

Der Parameter *domain* steht für die zu verwendende Adressfamilie. Wir verwenden im Praktikum `PF_INET` für IPv4.

Mit dem Parameter *type* wird die Übertragungsart gewählt. Für die Datenübertragung mittels UDP-Datagrammen ist `SOCK_DGRAM` zu verwenden. Der Parameter *protocol* wird gewöhnlich auf Null gesetzt, dann wird das standardmässige Protokoll benutzt, in unserem Fall UDP.

Falls ein Fehler bei der Anforderung auftritt, gibt die Funktion -1 als Rückgabewert zurück.

```
// Beispiel Code:

// einen UDP socket anfordern
if((sockFd = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
{
    // print error and exit when fails
    perror("socket");
    exit(1);
}
```

2.3.3 Adressinformationen

Bevor man mit dem angeforderten Socket Daten senden und empfangen kann, müssen die Adressinformationen für den Remote-Host festgelegt werden. In einem IP-Netzwerk ist ein Tupel von (IP_Addr, Port_Num) für die Adressierung zu benutzen. In C gibt es eine spezielle Struktur *sockaddr_in* für IP-Anwendungen.

```
struct sockaddr_in {
    sa_family_t sin_family; // Adress Familie
    unsigned short int sin_port; // Port Nummer
    struct in_addr sin_addr; // IP Adress
    unsigned char pad[8]; // Größe der Struktur
};
```

Port-Nummern niedriger als 1023 liegen im Bereich der sogenannten well-known Ports und sind für Standardanwendungen vorgesehen. Wir verwenden deshalb eine größere Nummer, z.B. 9000.

Weil in einem heterogenen Netz Rechner unterschiedliche Architekturen und damit unterschiedliche Anordnung der Daten bezüglich ihrer Reihenfolge haben können, ist gegebenenfalls eine Konvertierung der Byte-Reihenfolge zwischen dem eigenen Endsystem und dem Netzwerk nötig. So ist die einheitliche Form für die Übertragung im Netzwerk „Big-Endian“ (höchstwertiges Byte zuerst gespeichert, d.h. bei der kleineren Speicheradresse), wogegen die Raspberry Pi – ARM-Architektur „Little-Endian“ verwendet (also: kleinstwertiges Byte an kleinerer Adresse). Für die Konvertierung werden vier Funktionen bereitgestellt. Wenn die Datenübertragung zwischen zwei Endsystemen mit gleicher Architektur erfolgt (wie in unserem Fall), dann ist die sog. „Endianness“ nicht von Bedeutung.

```
/* short integer from host to network byte order */
unsigned short int htons(unsigned short int hostshort);
/* long integer from host to network byte order */
unsigned long int htonl(unsigned long int hostlong);
/* short integer from network to host byte order */
unsigned short int ntohs(unsigned short int netshort);
/* long integer from network to host byte order */
unsigned long int ntohl(unsigned long int netlong);
```

Mit Hilfe der Funktion *inet_addr()* kann man die Adresse einfach als Zeichenkette übergeben. Für einen lokalen Test, bei dem Sender und Empfänger auf ein und demselben Raspberry Pi laufen, kann man die Loopback Adresse benutzen, nämlich 127.0.0.1.

```
// Beispiel Code
```



```
// Adressinformation festlegen
#define UDP_PORT 9000

struct sockaddr_in remote_addr;
memset(&remote_addr, 0, sizeof(remote_addr)); // alle Null setzen

remote_addr.sin_family=AF_INET; // IPv4
remote_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
remote_addr.sin_port=htons(UDP_PORT);
```

2.3.4 Senden und Empfangen von Daten

Die UDP-Übertragung ist verbindungslos, d.h. man kann Daten einfach direkt senden und empfangen, ohne vorher eine Verbindung aufbauen zu müssen. Für das Senden der Daten mittels UDP verwendet man die *sendto()* Funktion.

```
ssize_t sendto(int sock, const void *buf, size_t len,
               int flags, const struct sockaddr *to,
               socklen_t tolen);
```

sock - ist der angeforderte Socket.

Buf - ist ein Pointer für die zu übertragenden Daten.

Len - ist die Länge der zu übertragenden Daten (in Bytes)

flag - liefert mehr Information dazu, wie Daten gesandt werden. *Flag=0* bedeutet normale Daten.

Andere flags kann man in der Dokumentation online finden.

*struct sockaddr *to* - ist ein Pointer für die Struktur der Adressinformationen des Remote-Host, d.h. in unserem Fall des Empfängers. Deshalb ist hier die Konvertierung in einen Pointer notwendig.

```
(struct sockaddr *)&remote_addr // konvertieren zum Pointer
```

tolen - ist die Größe der Adress-Struktur.

Falls ein Fehler bei der Sendung auftritt, gibt die Funktion -1 als Rückgabewert aus.

```
// Beispiel Code für Sendung von „hello world“

char *msg = "hello world\n";

if((len = sendto(sockFd, msg, strlen(msg), 0, (struct sockaddr *)&remote_addr,
sizeof(struct sockaddr))) < 0)
{
    perror("sendto");
    exit(1);
}
```

Für den Empfang der Daten mittels UDP kann man die *recvfrom()* Funktion aufrufen.

```
ssize_t recvfrom(int sock, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
```

Die Parameter sind dabei die gleichen wie bei der *sendto()* Funktion.

```
// Beispiel Code für den Empfang von Daten
#define BUFFSIZE 1024

char buffer[BUFFSIZE];

if ((len=(recvfrom(sockFd, buffer, BUFFSIZE, 0, (struct sockaddr *)&remote_addr,
&sin_size))) < 0) {
    perror("recvfrom");
    exit(1);
}

buffer[len] = '\0'; // add string flag at the end of buffer
printf("receive: %s\n", buffer);
```

Am Ende der Übertragungen müssen die Sockets geschlossen werden.

```
close(sockFd);
```

3. Hinweise für die Arbeit am Praktikumsplatz

Für den Programmtest soll direkt auf Kommandozeilenebene des Raspberry Pi gearbeitet werden. Für die Übersetzung des Programms wird ein makefile zur Verfügung gestellt.

3.1. Compilieren des Programms

Das Programm muss mit Erzeugung von Debug-Informationen kompiliert werden. Mit dem vorgegebenen makefile ist der folgende Befehl zu verwenden:

```
$ make BUILD=debug # es wird die Debug-Version erzeugt
```

3.2. Debuggen des Programms

(1) Starten des Debuggers cgdb

```
$ cgdb programm # z.B. cgdb server-udp
```

Damit wird eine Debugging Umgebung erzeugt.

(2) Starten debugging der main() Funktion

```
$ (gdb) start
```

(3) Debugging mit Kommandos des gdb.

Liste häufig verwendeter gdb-Kommandos

Anmerkung:

Durch Eingabe von [Enter] wird das letzte eingegebene Kommand noch einmal ausgeführt.

Hilfeinformationen anzeigen
(gdb) help

Programm ausführen. Das Programm stoppt bei einem Breakpoint oder wenn es Probleme gibt.
(gdb) run

Start des Programms und Setzen eines Breakpoint in der main Funktion
(gdb) start

Setzen eines Breakpoints an bestimmter Zeile
(gdb) break line_num

Anzeigen von Informationen zu gesetzten Breakpoints
(gdb) info breakpoints

Ablauf des Programms bis zum nächsten Breakpoint
(gdb) continue

Ablauf der nächsten Programmzeile als einzelner Befehl.(nicht in Funktion springen)
(gdb) next

Ablaufen der nächsten Zeile als ein Unterprogramm (Springen in die aufgerufene Funktion.
(gdb) step

Anzeige des Wertes der Variable
(gdb) print variable_name

das Programm wird unterbrochen falls sich der Wert der beobachteten Variable ändert
(gdb) watch variable_name

Ablauf des Programms bis zum Ende der aktuell laufenden Funktion
(gdb) finish

4. Hinweise zum Kolloquium

Folgende Schwerpunkten sind für eine intensive Beschäftigung mit der Aufgabenstellung unverzichtbar. Diese werden demzufolge auch Gegenstand des Eingangskolloquiums sein.

- TCP/IP [1]
 - Protokolle IP, TCP, UDP,
 - Ports, Sockets
- Client-Server Kommunikation [1]
 - wesentliche Elemente bei der Client/Server-Kommunikation
 - Aufgaben von Client und Server im Praktikumsprogramm
- Socket Programmierung für UDP [1]
 - Socket API in C
 - Socket Primitive (bind, listen, send/receive, etc.)
 - Erstellen eines Client- sowie eines Server-Programms
- Nutzung der Schnittstellen des Raspberry Pi [2, 3, 6]
 - GPIO-Interface
 - Softwarebibliothek für die Schnittstellenprogrammierung
 - Bibliothek WiringPi
- Externe Hardware [4]
 - Aufgabe und Funktionalität des A/D-Wandlers
 - Temperaturerfassung mittels Sensor
 - Geeignete Umwandlung der Daten des A/D-Wandlers
 - Ausgabe auf das OLED-Display

5. Quellen und weiterführende Dokumentationen

TCP/IP und Netzwerkprogrammierung

- [1] Prof. P. Fatourou, E. Kosmas: [Introduction to Socket Programming in C using TCP/IP](#)

Hardware

- [2] www.linkerkit.de: Informationen zu Linker Kit Baseboard
- [3] SPI and I2C auf Raspberry Pi
<https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial>
- [4] Externe Hardware: SSD1306, MCP3008 and TMP64
<https://www.adafruit.com/datasheets/SSD1306.pdf>
<https://www.adafruit.com/datasheets/MCP3008.pdf>
http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf

Software

- [5] Programmiersprache C: [Vorlesung Prof. Urbas](#)
WikiBooks: [C-Programmierung](#)
WikiBooks: [C-Sprachbeschreibung](#)
- [6] Dokumentation WiringPi-GPIO-Software Bibliothek
<http://wiringpi.com/reference/>