TECHNISCHE HOCHSCHULE DEGGENDORF

FAKULTÄT ANGEWANDTE INFORMATIK

# Deckblatt

Wissenschaftliche Arbeit zum Thema

**Steuerung einer LED-Matrix mithilfe eines Treibers auf einem Raspberry Pi 4**

Ein Bild, das Text, Elektronik enthält.

Automatisch generierte Beschreibung

Abbildung 1 Aufbau LED-Matrix

vorgelegt von

Anton Kraus - Matrikelnummer: 00804697

Simon Obermeier - Matrikelnummer: 00800498

Eingereicht: 14.06.2022

Betreuer: Andreas Laubhahn

Inhalt

[Deckblatt 0](#_Toc106142367)

[Abbildungsverzeichnis 2](#_Toc106142368)

[Abkürzungsverzeichnis 2](#_Toc106142369)

[III Erklärung 2](#_Toc106142370)

[1 Vorwort 3](#_Toc106142371)

[1.1 Aufgabenstellung 3](#_Toc106142372)

[1.2 Zeitplan 3](#_Toc106142373)

[1.3 Ausgangssituation 3](#_Toc106142374)

[2. Aufbau 4](#_Toc106142375)

[2.1 Software (Linux) 4](#_Toc106142376)

[2.2.1 Aufbau und Funktion einer LED-Matrix 4](#_Toc106142377)

[2.2.2 Umsetzung eines Raspberry Hats 5](#_Toc106142378)

[3. Kernel Driver 8](#_Toc106142379)

[3.1 Kernel Module für die LED-Matrix 9](#_Toc106142380)

[3.1.1 Matrix Kontroller 9](#_Toc106142381)

[3.1.2 GPIO Initialisierung 9](#_Toc106142382)

[3.1.3 I/O Controll 10](#_Toc106142383)

[3.1.4 LED-Matrix Treiber 11](#_Toc106142384)

[3.2 Makefile 14](#_Toc106142385)

[4. User App 15](#_Toc106142386)

[4.1 User App Funktionen 15](#_Toc106142387)

[4.2 User App Controller 16](#_Toc106142388)

[4.3 User App 17](#_Toc106142389)

[5. User App Steuerung 19](#_Toc106142390)

[5.1 Install Skripte 19](#_Toc106142391)

[5.2 User App ausführen 19](#_Toc106142392)

[Literaturverzeichnis 21](#_Toc106142393)

# Abbildungsverzeichnis

[Abbildung 1 Aufbau LED-Matrix 0](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142034)

[Abbildung 2 Zeitplan 3](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142035)

[Abbildung 3 Aufbau einer LED-Matrix 4](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142036)

[Abbildung 4 Schematischer Anschluss LED-Matrix and Raspberry 5](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142037)

[Abbildung 5 Layout Pi Hat 6](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142038)

[Abbildung 6 Pi Hat PCB back 7](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142039)

[Abbildung 7 Pi Hat PCB front 7](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142040)

[Abbildung 8 Aufbau Kernel 8](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106142041)

# Abkürzungsverzeichnis

GPIO General Purpose Input Outpu

LED Light Emitting Diode

# III Erklärung

Hiermit versichern wir, Anton Kraus und Simon Obermeier, diese Dokumentation, ohne die Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum, Simon Obermeier

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum, Anton Kraus

# 1 Vorwort

Im Rahmen des Faches System-Programmierung wurde mithilfe eines Raspberry PI 4 innerhalb einer Linux Distribution, Pi OS, ein Kernel-Treiber für eine LED-Matrix programmiert. Unter der Aufsicht von Andreas Laubhahn wurden die aufgelisteten Materielaien bereitgestellt. Im Folgenden Dokument wird nochmals die Aufgabenstellung, unser Zeitplan, sowie der Aufbau software- und hardwareseitig geschildert. Für die Hardware haben wir ebenfalls ein umsetzbares Aufsteckboard für den Pi entworfen.

Im Anschluss wird der Kernel Treiber erklärt, sowie der Aufbau unseres Treibers. Zuletzt wird unser Application zur Nutzung unseres Treibers erklärt, sowie Probleme, auf welche wir gestoßen sind und wie diese von uns gelöst wurden.

Zudem kann es scheinen, als ob das wichtige Code Teile in dieser Dokumentation fehlen würden, dies hat den Hintergrund, dass hier nur die am wichtigsten empfunden Code Teile erklärt werden.

Für den Vollständigen Code verweise ich hier an das hierfür angelegt [Git Repository](https://github.com/SenpaiSimon/Sysprog-AI-B6-Projekt).

## 1.1 Aufgabenstellung

Für das Fach Systemprogrammierung soll ein Treiber (Character Device Driver) und eine Applikation programmiert werden. Sourcen mit Quellenangaben als Anhang.

## 1.2 Zeitplan



Abbildung 2 Zeitplan

## 1.3 Ausgangssituation

Auf einem Raspberry Pi 4 mit Rasperry Pi OS als Betriebssystem soll ein Treiber (Character Device Driver) und eine Applikation programmiert werden.

Genauer, es soll ein LED-Treiber für eine 8x8 LED-Matrix programmiert werden. Dieser Treiber soll aus einem Kernel Driver bestehen, sowie einer User Application, um die Matrix zu steuern.

# 2. Aufbau

Im Folgenden Abschnitt wird der Aufbau softwareseitig erklärt, sowie der Aufbau einer LED-Matrix. Auch wie ein Schiel bzw. Hat, also ein direktes Aufsteckboard für den Raspberry aussehen konnte wurde in dem CAD Programm KiCad umgesetzt

## 2.1 Software (Linux)

Raspberry Pi OS, früher Raspbian ist eine offizielle Linux- Distribution.

Eine Linux-Distribution ist eine Version des Open-Source-Betriebssystems Linux in Verbindung mit weiteren Komponenten wie etwa Installationsprogrammen, Verwaltungswerkzeugen und zusätzlicher Software wie zum Beispiel dem KVM-Hypervisor.

Linux-Distributionen, die auf dem Linux-Kernel basieren, sind für Anwender oft einfacher zu installieren als die originale Open-Source-Version von Linux. Das liegt unter anderem daran, dass den Anwendern der Schritt des Kompilierens des kompletten Linux-Systems aus dem Quellcode erspart wird. Außerdem ist die Unterstützung durch den Distributor oftmals ein wichtiges Plus. (ComputerWeekly.de 2022)

Mittlerweile gibt es hunderte verschiedene Linux-Distributionen für jeden nur erdenklichen Einsatzzweck, seien Server, Mobilgeräte oder einfache Desktops. Grundsätzlich bestehen solche Distributionspakete aus Softwarepaketen, welche wiederrum aus Anwendungen oder Diensten bestehen. Ein solches Paket kann z.B. aus Schriftarten oder Webbrowsern bestehen.

Ermöglicht wird dies durch den Open Source (also frei Verfügbaren) Ansatz von Linux welcher unter dem Copyleft-Manifest der Free Software Fundation entwickelt, die der GNU General Public License (GPL) entspringt. (ComputerWeekly.de 2022)

### 2.2.1 Aufbau und Funktion einer LED-Matrix



Abbildung 3 Aufbau einer LED-Matrix

Eine LED-Matrix ist im Grunde ein Array, welches aus Dioden besteht. Hierbei gibt es immer einen Eingangs- bzw. Ausgangspin, welche die LEDs in einer Reihe bzw. einer Spalte verbinden. Diese Pins können je nach Art des Displays durch Anlegen einer Spannung oder eines Signals zum Leuchten gebracht werden.

Wichtig hierbei wie auch bei unserem Aufbau, sollte direkt eine Spannung angelegt werden müssen Vorwiederstände verwendet werden, damit die LEDs nicht durch die Spannung zerstört werden. Auch werden die Reihen und Spalten nicht mit 1-8 deklariert, sondern wie in der Informatik üblich 0-7.

Ebenso gibt es wie in Arrays keine x und y Koordinaten, sondern die einzelnen LEDs, auch Dots genannt, werden mit Reihen und Spalten angesprochen (Ewald 2020).

### 2.2.2 Umsetzung eines Raspberry Hats

Unter einem Raspberry Hat/Shield wird ein Aufsteckt-Bord verstanden, welches direkt auf den Raspberry gesteckt wird, sodass dieser Aufbau nicht über Breadboard realisiert werden muss.

#### 2.2.2.1 Schaltplan



Abbildung 4 Schematischer Anschluss LED-Matrix and Raspberry

In Abb.4 zu sehen ist der Schematische Aufbau der LED Matrix an den Raspberry Pi.

Hierbei wurden die GPIO (General Purpose Input / Output) Pins des Raspberrys an die LED-Matrix angeschlossen. Die Anoden, also die Postiven Pins (Hx) wurden über einen Vorweiderstand zur Strombegrenzung angeschlossen. Die Kathoden, also negativen Pins (Lx) ebenso an einen GPIO Port.

Wie diese GPIO Pins angesprochen werden wird in „3.1.1 Matrix Kontroller“ erklärt.

#### 2.2.2.2 Layout



Abbildung 5 Layout Pi Hat

Damit nun aus dem Schaltplan eine PCB werden kann wurde zunächst allen Bauteilen ein Footprint zugewiesen und diese in ein sogenanntes Layout Fenster übertragen. Die Anordnung erfolgt nach dem Schaltplan, die Mounting-Holes wurden and die Abstände des Raspberry PIs angepasst.

Weitere Erklärungen würden den Rahmen dieser Arbeit sprengen, können auf Nachfrage jedoch von den Autoren erklärt werden.

#### 2.2.2.3 Pi-Hat PCB



Abbildung 6 Pi Hat PCB back

Abbildung 7 Pi Hat PCB front

Nach dem fertigen Layout kann nun in eine 3D-Ansicht gewechselt werden, um die PCB zu betrachten, wie diese fertig aussehen würde.

Links in Abbildung 7 und Abbildung 6 zu sehen sind die Pin Sockets welche auf die Pin-Leiste des Raspberry gesteckt werden würde.

# 3. Kernel Driver

Ein Kernel ist wie der Name bereits aussagt der Kern eines Betriebssystems. Aufgrund seines Monolithischen Aufbaus ist er in der Lage Module zur Laufzeit zu laden und entladen. (Kernelmodule › Wiki › ubuntuusers.de 2022)

Laufen diese Module im Privilegierten Modus, haben diese unbeschränkten zugriff auf die Hardware. Ausgenommen Treiber, welche für den Start des Systems verantwortlich sind, kann nahezu jeder Treiber auch als Modul zur Verfügung stehen und vom System dynamisch nachgeladen werden.

Die im System laufende Programme bekommen wiederum vom Kernel Prozessorzeit zugewiesen. Jeder dieser Prozesse erhält einen eigenen, geschützten Speicherbereich und kann nur über Systemaufrufe auf die Gerätetreiber und das Betriebssystem zugreifen. Die Prozesse laufen dabei im Benutzermodus (user mode), während der Kernel im Kernel-Modus (kernel mode) arbeitet. Die Privilegien im Benutzermodus sind sehr eingeschränkt. Abstraktion und Speicherschutz sind nahezu vollkommen, ein direkter Zugriff wird nur sehr selten und unter genau kontrollierten Bedingungen gestattet. Dies hat den Vorteil, dass kein Programm z. B. durch einen Fehler das System zum Absturz bringen kann. (Wikipedia 2022)

Eine gute Veranschaulichung liefert hier die auf Wikipedia zu findende Grafik:



Abbildung 8 Aufbau Kernel

Wie in der Grafik zu sehen, besteht ein Kernel aus mehreren Schichten, die Basis bildet Hierbei die Hardware, in unserem Fall die LED-Matrix, danach der Kernel mit den Treibern, Prozessen, Netzwerk und dem Dateisystem. In dieser Schicht befindet sich später ebenfalls unser Treiber. Mit diesem Treiber kommunizieren wir später aus dem User Mode, also unserer User Application. Wie genau hierbei unser Programm funktioniert wird in Abschnitt 4 näher erklärt.

Grundsätzlich braucht man ein Kernel für Schnittstellen zu Anwenderprogrammen. Gleichzeitig hat der Kernel vollen (und möglicherweise nur) Zugriff auf Prozessoren, Geräte und Speicher und weist Benutzerprogrammen Ressourcen (z. B. Prozessorzeit) zu. Darüber hinaus ist der Kernel für die Strukturierung von Ressourcen, die Lösung von Zugriffskonflikten, die Virtualisierung von Ressourcen und die Steuerung des Zugriffs auf Dateien und Geräte für Mehrbenutzersysteme verantwortlich. (Was ist ein Kernel? – Definition im IT-Lexikon 2022)

Nachdem nun klar ist, was ein Kernel ist und welche Funktion dieser erfüllt wird nun im nächsten Anschnitt der Aufbau unserer Module im Kernel für die LED-Matrix erklärt

## 3.1 Kernel Module für die LED-Matrix

Nachdem es uns nicht gelungen ist verschiedene C-Files zusammen zu kopieren wurde für die meisten aber nicht alle Globale Variablen dieses mit verschiedenen Header Files gelöst, weswegen bereits hier einige Funktionen für den Treiber bereits erklärt werden.

### 3.1.1 Matrix Kontroller

Zuerst beinhaltet unser Kernel Treiber die .h File für den Matrix Kontroller, in welcher der State also der Status, in welchem sich die einzelnen Dots bzw. hier Pixel befinden. Hier kann der State auf 1, also „HIGH/EIN“ oder auf 0, also „LOW/AUS“ gesetzt und/oder zurückgegeben werden.

|  |  |
| --- | --- |
| Sets the state of a pixel | 1 = on 0 = off | Returns the state of a pixel | 1 = on 0 = off |
| static void setPixel(int row, int line, int state) {  if(row > ROWS && row <= 0) {  printk(KERN\_ERR "Row: %d out of range", row);  return;  } else if (line > LINES && line <= 0) {  printk(KERN\_ERR "Line: %d out of range", line);  return;  }  gpio\_set\_value(hPin\_arr[row - 1], state);  gpio\_set\_value(lPin\_arr[line - 1], !state);  } | static int getPixelState(int row, int line) {  if(row > ROWS && row <= 0) {  printk(KERN\_ERR "Row: %d out of range\n", row);  return 2;  } else if (line > LINES && line <= 0) {  printk(KERN\_ERR "Line: %d out of range\n", line);  return 2;  }  return gpio\_get\_value(hPin\_arr[row - 1]) && !gpio\_get\_value(lPin\_arr[line - 1]);  } |

### 3.1.2 GPIO Initialisierung

Des Weiteren befindet sich hier die Header File, in welcher die GPIO Pins definiert werden, sowie die Initialisierung der GPIO Pins jeweils für „Active High“ sowie „Active Low“ Treiber. Die Initialisierung läuft wie folgt:

1. Check if the GPIO is valid

2. Request the GPIO

3. Set the direction of the GPIO to output and state when effective

4. Export the GPIO with no direction change

|  |
| --- |
| static void init\_all\_gpios(void) {  // Active HIGH Drivers  for(int i = 0; i < ROWS; i++) {  if(!gpio\_is\_valid(hPin\_arr[i])) {  printk(KERN\_ERR "Err, Pin No %d not valid!\n", hPin\_arr[i]);  return;  } else {  if(gpio\_request(hPin\_arr[i], hPin\_arr\_names[i])) {  printk(KERN\_ERR "Err, Pin No %d request failed!\n", hPin\_arr[i]);  return;  } else {  if(gpio\_direction\_output(hPin\_arr[i], 1)) {  printk(KERN\_ERR "Err, Pin No %d set direction failed!\n", hPin\_arr[i]);  return;  } else {  if(gpio\_export(hPin\_arr[i], false)) {  printk(KERN\_ERR "Err, Pin No %d export failed!\n", hPin\_arr[i]);  return;  }  }  }  }  } |

### 3.1.3 I/O Controll

Input/Output Control (kurz ioctl) ist eine generische Operation oder ein Systemaufruf, der in den meisten Treiberklassen verfügbar ist. Dies ist ein Systemaufruf, der in allen Fällen funktioniert. ioctl() wird verwendet, wenn es keine anderen Systemaufrufe gibt, die bestimmte Anforderungen erfüllen. Praktische Beispiele sind die Lautstärkeregelung für Audiogeräte oder in unserem Fall die Anzeigekonfiguration der LED-Matrix (Open Source For You 2011).

In unserem Fall besteht die IOCTL aus den benötigten Datenstrukturen, sowie defines für die Commands der LED-Matrix.

|  |
| --- |
| typedef struct state {  int row;  int line;  int state;  } state\_t;  typedef struct dimensions {  int rows;  int lines;  } dimensions\_t; |
| Set a specific Pixel  #define SET\_PIXEL \_IOW('a', 'a', state\_t\*)  Read a specific pixel  #define READ\_PIXEL \_IOWR('a', 'b', state\_t\*)  Get the dimensions of the Pixel Array  #define GET\_DIMS \_IOR('a', 'c', dimensions\_t\*) |

### 3.1.4 LED-Matrix Treiber

Am wichtigsten natürlich ist der Matrix Treiber. Dieser besteht wieder aus einer Header File in welcher sich die Funktions-Prototypen befinden.

|  |
| --- |
| static int dev\_uevent(struct device \*dev, struct kobj\_uevent\_env \*env);  static int \_\_init dev\_init(void);  static void \_\_exit dev\_exit(void);  static int dev\_open(struct inode \*inode, struct file \*file);  static int dev\_release(struct inode \*inode, struct file \*file);  static ssize\_t dev\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*offset);  static ssize\_t dev\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*offset);  static long dev\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg); |

Die C File hingegen besteh aus den aus den Praktika bereits bekannten und benötgten Funktionen, wobei auch hier wieder auf die wichtigsten Funktionen beschränkt wurde. Wie aus den Praktika bekannt wurde eine File Operation implementiert, sowie ein Event für das Device. Das Device wurde initialisiert, sowie ein Device init/exit Funktion, um das Modul zu laden bzw. zu entladen:

|  |  |
| --- | --- |
| static int \_\_init dev\_init(void) {  // Initialize the device itself  int err;  dev\_t dev;  err = alloc\_chrdev\_region(&dev, 0, 1, DEVICE\_NAME);  dev\_major = MAJOR(dev);  dev\_class = class\_create(THIS\_MODULE, DEVICE\_NAME);  dev\_class->dev\_uevent = dev\_uevent;  cdev\_init(&cdev, &fops);  cdev.owner = THIS\_MODULE;  cdev\_add(&cdev, MKDEV(dev\_major, 0), 1);  device\_create(dev\_class, NULL, MKDEV(dev\_major, 0), NULL, DEVICE\_NAME);  printk(KERN\_INFO "Init GPIO Led Matrix Driver\n");  //Initialize the GPIOs  init\_all\_gpios();  return 0;  } | static void \_\_exit dev\_exit(void)  {  //free up all the GPIOs  exit\_all\_gpios();  // Remove and unregister the device  device\_destroy(dev\_class, MKDEV(dev\_major, 0));  class\_unregister(dev\_class);  class\_destroy(dev\_class);  unregister\_chrdev\_region(MKDEV(dev\_major, 0), MINORMASK);  printk(KERN\_INFO "Unregister GPIO Led Matrix Driver\n");  return;  } |

Ebenso muss das Device von der User App geöffnet, sowie geschlossen werden können

|  |  |
| --- | --- |
| static int dev\_open(struct inode \*inode, struct file \*file)  {  printk(KERN\_INFO "GPIO LED Driver started access\n");  return 0;  } | static int dev\_release(struct inode \*inode, struct file \*file)  {  printk(KERN\_INFO "GPIO LED Matrix Driver stopped access\n");  return 0;  } |

Zudem beinhaltet der LED-Matrix Treiber die Device read Funktion über welche innerhalb der Command Line auf das Device zugegriffen werden kann und Vordefinierte Inhalte ausgelesen werden können.

|  |
| --- |
| static ssize\_t dev\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*offset)  {  int len;  char buffer[650];  char temp[64];  int matrixState[ROWS][LINES];  printk(KERN\_INFO "GPIO LED Matrix Driver read\n");  // fill up the state array  for(int i = 1; i <= ROWS; i++) {  for(int j = 1; j <= LINES; j++) {  matrixState[i-1][j-1] = getPixelState(i, j);  }  }  // generate the output  sprintf(buffer, "State of the Matrix Table:\n");  for(int i = 0; i < LINES; i++) {  for(int j = 0; j < ROWS; j++) {  sprintf(temp, "| %d ", matrixState[j][i]);  strcat(buffer, temp);  }  strcat(buffer, "|\n");  // generate a one line gap excluding the last line  if(i != LINES - 1) {  for(int k = 0; k < ROWS; k++) {  strcat(buffer, "| ");  }  strcat(buffer, "|\n");  }    }  len = strlen(buffer);  // actual output to user  if(copy\_to\_user(buf, buffer, len) > 0)  pr\_err("dev\_read(): ERROR\n");  if(\*offset == 0) {  \*offset += len;  return len;  } else {  return 0;  }  } |

Dasselbige mit der Funktion auf das Device zu schreiben.

|  |
| --- |
| static ssize\_t dev\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*offset)  {  uint8\_t data;  int ret = copy\_from\_user(&data, buf, sizeof(data));  // convert char input to a integer  data -= '0';  if(ret == 0) pr\_info("GPIO LED Matrix Driver write success with input: %d\n", data);  for(int i = 1; i <= ROWS; i++) {  for(int j = 1; j <= LINES; j++) {  setPixel(i,j,data);  }  }  return count;  } |

Im Vorletzten Schritt mussten nun noch die IO-Controll Operationen programmiert werden.

|  |
| --- |
| static long dev\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg) {  printk(KERN\_INFO "GPIO LED Matrix Driver IOCT accessed\n");  switch(cmd) {  case SET\_PIXEL: {  state\_t input;  if(copy\_from\_user(&input, (state\_t\*)arg, sizeof(input)) > 0) {  printk(KERN\_ERR "Set Pixel Error! \n");  }  setPixel(input.row, input.line, input.state);  break;  }  case READ\_PIXEL: {  state\_t input;  if(copy\_from\_user(&input, (state\_t\*)arg, sizeof(input)) > 0) {  printk(KERN\_ERR "Get Pixel Error! \n");  }  input.state = getPixelState(input.row, input.line);  if(copy\_to\_user((state\_t\*)arg, &input, sizeof(input)) > 0) {  printk(KERN\_ERR "Get Pixel Error! \n");  }  break;  }  case GET\_DIMS: {  dimensions\_t output;  output.lines = LINES;  output.rows = ROWS;  if(copy\_to\_user((dimensions\_t\*)arg, &output, sizeof(output)) > 0) {  printk(KERN\_ERR "Get Dimensions Error! \n");  }  break;  }  default:  printk(KERN\_ERR "Unknown Command! \n");  break;  }  return 0;  } |

Als Letzter Schritt müssen die „Init“und „Exit“ Funktionen verlinkt werden,

|  |
| --- |
| module\_init(dev\_init);  module\_exit(dev\_exit); |

sowie der Kernel Treiber mit einer Signatur versehen werden.

|  |
| --- |
| MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Simon Obermeier & Anton Kraus");  MODULE\_DESCRIPTION("LED Matrix Driver");  MODULE\_VERSION("1.0"); |

## 3.2 Makefile

Ein Makefile teilt dem Programm mit dem Befehl „make“ mit, was es tun soll (das ist das »Ziel«) und wie es es tun soll (das ist die dem Ziel zugeordnete »Regel«). Außerdem kann für jedes Ziel andere Ziele oder Dateien angegeben werden, von denen es abhängt. (Becker 2007)

In unserem Makefile wird ein Kernel Objekt erzeugt, sowie in dem Directory gebildet. Zusätzlich wird eine Compiling flag verwendet, um den Treiber mit dem C99 Standard zu Kompilieren, damit for-schleifen verwendet werden können, welche unter der standardmäßigen C90 Kompilierung nicht verwendbar sind.

Mit dem „all“ Befehl wird das Default Case aufgerufen, welches bei make ohne Target lediglich das Kernel Objekt Bildet. Clean löscht dabei alle temporär erstellten Dateien. Disclean cleant die temporären Dateien, sowie das Kernel Objekt. Install kompiliert die Module, ruft clean auf und installiert das Modul mit insmod. Dieser Befehl muss mit $sudo ausgeführt werden, da es die Berechtigung benötigt Kernel Objekte zu löschen, nachdem diese in den Kernel hineingeladen wurden.

|  |
| --- |
| obj-m += led\_matrix\_driver.o  KDIR = /lib/modules/$(shell uname -r)/build  ccflags-y += -std=gnu99  all: default clean  default:  make -C $(KDIR) M=$(shell pwd) modules  clean: #cleant temp. Sachen  @rm -rf \*.o .\*.cmd .\*.flags \*.mod.c \*.order \*.mod .\*.o.d  @rm -rf .\*.\*.cmd \*.symvers \*~ \*.\*~ TODO.\*  @rm -rf .tmp\*  @rm -rf .tmp\_versions  @rm -rf gpio\_inits/\*.o gpio\_inits/.\*.o.d gpio\_inits/.\*.o.cmd  disclean: clean #disc cleant temp & kernel obj  @rm -rf \*.ko  install: default clean  @sudo insmod led\_matrix\_driver.ko  @rm -rf \*.ko |

# 4. User App

Die User App, bzw. Benutzer Application, ist das Modul mit welchem wir unseren Treiber steuern. Hierfür werden Funktionen, der Kontroller, sowie die App selbst benötigt. Der Aufbau erfolgt in verschiedenen Thematischen Files.

## 4.1 User App Funktionen

Für die User App werden Funktionen benötigt, um alle LEDs zu setzten,

|  |
| --- |
| void led\_set\_all(uint8\_t state) {  for(int i = 0; i <= DIMENSIONS.rows; i++) {  for(int j = 0; j <= DIMENSIONS.lines; j++) {  setPixel(i, j, state);  }  }  } |

sowie einzelne Spalten und/oder Reihen.

|  |  |
| --- | --- |
| void led\_set\_row(uint8\_t row, uint8\_t set) {  for(int i = 1; i <= DIMENSIONS.rows; i++) {  setPixel(row, i, set);  }  } | void led\_set\_line(uint8\_t line, uint8\_t set) {  for(int i = 1; i <= DIMENSIONS.lines; i++) {  setPixel(line, i, set);  }  } |

Ebenfalls eine Snake Funktion, dem Spiel Snake nachempfunden, aber nicht das Spiel selbst wurde als nettes Graphisches Beispiel implementiert.

|  |
| --- |
| void led\_snake(int speed) {  led\_set\_all(0);  for(int i = 1; i <= DIMENSIONS.lines; i+=2) {  // move right  for(int j = 1; j <= DIMENSIONS.rows; j++) {  setPixel(j, i, 1);  usleep(speed\*1000);  setPixel(j, i, 0);  }  // move back left  for(int j = DIMENSIONS.rows; j >= 1; j--) {  setPixel(j, i+1, 1);  usleep(speed\*1000);  setPixel(j, i+1, 0);  }  }  } |

## 4.2 User App Controller

Der LED-Controller setzt sich wieder Mals aus einer Header Datei in welcher die benötigten Bibliotheken eingebunden, sowie die Funktions-Prototypen deklariert werden.

Der Controller beinhaltet dabei die getDims() Funktion welche die Dimensionen der Matrix in den Kernel Treiber zurückgibt,

|  |
| --- |
| dimensions\_t getDims() {  // set default values  dimensions\_t dims;  dims.lines = -1;  dims.rows = -1;  // Open the Device for use  int fd = open("/dev/led\_matrix", O\_RDWR);  if(fd < 0) {  printf("Cannot open device file\n");  return dims;  }  // Pass the data  ioctl(fd, GET\_DIMS, &dims);    return dims;  } |

sowie die Funktionen, um Pixel zu setzen und deren State in welchem sich diese befindet zurückzugeben

|  |  |
| --- | --- |
| int setPixel(uint8\_t row, uint8\_t line, uint8\_t state) {  // Open the Device for use  int fd = open("/dev/led\_matrix", O\_RDWR);  if(fd < 0) {  printf("Cannot open device file\n");  return -1;  }  // generate the data and pass it  state\_t input;  input.row = row;  input.line = line;  input.state = state;  ioctl(fd, SET\_PIXEL, &input);  // close the device after use  close(fd);  } | uint8\_t getPixelState(uint8\_t row, uint8\_t line) {  // Open the Device for use  int fd = open("/dev/led\_matrix", O\_RDWR);  if(fd < 0) {  printf("Cannot open device file\n");  return -1;  }  // generate the data and pass it  state\_t input;  input.row = row;  input.line = line;  ioctl(fd, READ\_PIXEL, &input);  // close the device after use  close(fd);  return input.state;  } |

Über die letzte implementierte Funktion lässt sich noch der Status des Gerätes asulesen

|  |
| --- |
| void readDeviceState(char \*buf) {  int fd = open("/dev/led\_matrix", O\_RDWR);  if(fd < 0) {  sprintf(buf, "Cannot open device file\n");  } else {  read(fd, buf, sizeof(buf));  }  } |

## 4.3 User App

Die User App wurde wieder Mals in einen Header sowie in eine C File aufgeteilt. In der Header File befinden sich wieder die zu Importierenden Bibliotheken, sowie ein Funktion Prototyp, um die eingegebene Nummer auszulesen.

Im Hauptprogramm sieht diese wie folgt aus:

|  |
| --- |
| int readNumber(char \*input) {  int userInput;  char flushInput[10];  // prompt the user for input  printf("%s: ", input);  scanf("%i", &userInput);  scanf("%c", flushInput);  // clear the terminal  printf("\033[2J");    return userInput;  } |

Die Main Funktion besteht aus einem Switch Case in welchem die bereits erklärten Funktionen aufgerufen werden und somit die LED-Matrix nach Belieben angesteuert werden kann.

|  |
| --- |
| int main(void)  {  // get the dimensions for use in functions.c  DIMENSIONS = getDims();  char buf[600];  int running = 1;  while(running) {  printf("Select an Action from the Table below to perform:\r\n\r\n");  printf("1: Set all LEDs [state]\n\r");  printf("2: Set a single LED [row][line][state]\n\r");  printf("3: Set a single ROW [row][state]\n\r");  printf("4: Set a single LINE [line][state]\n\r");  printf("5: Snake [delay]\n\r");  printf("8: Print out current state\n\r");  printf("9: Exit App and Toggle off\n\r");  switch(readNumber("Select")) {  case 1:  printf("You selected: Toggle all LEDs\r\n");  led\_set\_all(readNumber("Which state to you want"));  break;  case 2:  printf("You selected: Toggle a single LED\r\n");  setPixel(readNumber("Row"), readNumber("Line"), readNumber("State"));  break;  case 3:  printf("You selected: Set a single ROW\r\n");  led\_set\_row(readNumber("Row"), readNumber("State"));  break;  case 4:  printf("You selected: Set a single LINE\r\n");  led\_set\_line(readNumber("Line"), readNumber("State"));  break;  case 5:  printf("You selected: Snake\r\n");  led\_snake(readNumber("Delay"));  break;  case 8:  readDeviceState(buf);  printf("%s", buf);  printf("\r\n");  break;  case 9:  printf("See you next time!\r\n");  led\_set\_all(0);  running = 0;  break;  default:  printf("Unknown function!\r\n");  }  }    return EXIT\_SUCCESS;  } |

# 5. User App Steuerung

Im letzten Abschnitt werden nun noch die Vorgänge erklärt um die benötigten Skripte zu installieren, sowie die benötigten Befehle erläutert um anhand der User App die LED-Matrix zu steuern.

## 5.1 Install Skripte

Install skripte erklären

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## 5.2 User App ausführen

App öffnen

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Im Kernel

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

5. Probleme und Lösungen

(Optional)

Literaturverzeichnis

Becker, Michael (2007): Eine Einführung in Makefiles. Online verfügbar unter https://wwwvs.cs.hs-rm.de/lehre/material/extern/pr04ss/dokumente/makefiles.htm, zuletzt aktualisiert am 11.05.2007, zuletzt geprüft am 11.06.2022.

ComputerWeekly.de (2022): Was ist Linux-Distribution? - Definition von WhatIs.com. Online verfügbar unter https://www.computerweekly.com/de/definition/Linux-Distribution, zuletzt aktualisiert am 28.05.2022, zuletzt geprüft am 28.05.2022.

Ewald, Wolfgang (2020): LED Matrix Display ansteuern. In: *Wolfgang Ewald*, 13.04.2020. Online verfügbar unter https://wolles-elektronikkiste.de/led-matrix-display-ansteuern#anker1, zuletzt geprüft am 28.05.2022.

Kernelmodule › Wiki › ubuntuusers.de (2022). Online verfügbar unter https://wiki.ubuntuusers.de/Kernelmodule/, zuletzt aktualisiert am 11.06.2022, zuletzt geprüft am 11.06.2022.

Open Source For You (2011): Input/Output Control in Linux | ioctl implementation. Online verfügbar unter https://www.opensourceforu.com/2011/08/io-control-in-linux/, zuletzt aktualisiert am 23.10.2020, zuletzt geprüft am 11.06.2022.

Was ist ein Kernel? – Definition im IT-Lexikon (2022). Online verfügbar unter https://it-service.network/it-lexikon/kernel, zuletzt aktualisiert am 11.06.2022, zuletzt geprüft am 11.06.2022.

Wikipedia (Hg.) (2022): Linux (Kernel). Online verfügbar unter https://de.wikipedia.org/w/index.php?title=Linux\_(Kernel)&oldid=223567921, zuletzt aktualisiert am 09.06.2022, zuletzt geprüft am 11.06.2022.