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 Aufbau LED-Matrix

vorgelegt von

Anton Kraus - Matrikelnummer: 00804697

Simon Obermeier - Matrikelnummer: 00800498

Eingereicht: 14.06.2022

Betreuer: Andreas Laubhahn, Sergej Lamert

Inhalt

[Deckblatt 0](#_Toc106958860)

[Abbildungsverzeichnis 2](#_Toc106958861)

[Code Verzeichnis 2](#_Toc106958862)

[Erklärung 2](#_Toc106958863)

[Vorwort 3](#_Toc106958864)

[Zeitplan 3](#_Toc106958865)

[Aufbau 4](#_Toc106958866)

[Software (Linux) 4](#_Toc106958867)

[Aufbau und Funktion einer LED-Matrix 4](#_Toc106958868)

[Umsetzung eines Raspberry Aufsteckboards 5](#_Toc106958869)

[Kernel Driver 8](#_Toc106958870)

[Kernel Module für die LED-Matrix 9](#_Toc106958871)

[Matrix Kontroller 9](#_Toc106958872)

[GPIO Initialisierung 9](#_Toc106958873)

[I/O Control 9](#_Toc106958874)

[LED-Matrix Treiber 11](#_Toc106958875)

[Makefile 12](#_Toc106958876)

[User App 13](#_Toc106958877)

[User App Funktionen 13](#_Toc106958878)

[User App Controller 14](#_Toc106958879)

[User App 15](#_Toc106958880)

[User App Steuerung 17](#_Toc106958881)

[Install Skripte 17](#_Toc106958882)

[User App ausführen 17](#_Toc106958883)

[Code-Ausschnitte 19](#_Toc106958884)

[Literaturverzeichnis 23](#_Toc106958885)

# Abbildungsverzeichnis

[Abbildung 1 Aufbau LED-Matrix 0](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958886)

[Abbildung 2 Zeitplan 3](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958887)

[Abbildung 3 Aufbau einer LED-Matrix 4](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958888)

[Abbildung 4 Schematischer Anschluss LED-Matrix and Raspberry 5](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958889)

[Abbildung 5 Layout Pi Hat 6](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958890)

[Abbildung 6 PI Hat PCB Rückseite 7](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958891)

[Abbildung 7 PI Hat PCB Vorderseite 7](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958892)

[Abbildung 8 Aufbau Kernel 8](file:///B:\GIT\Sysprog-AI-B6-Projekt\doc\LED-Matrix-Driver-Documentation.docx#_Toc106958893)

# Code Verzeichnis

[Sourcecode 1: Einzelnen Pixel setzen - ./src/kernel\_driver/controller/matrix\_controller.h 19](#_Toc106958894)

[Sourcecode 2: Status eines einzelnen Pixel erhalten - ./src/kernel\_driver/controller/matrix\_controller.h 19](#_Toc106958895)

[Sourcecode 3: Initialisierung der High Treiber - ./src/kernel\_driver/gpio\_inits/gpio\_init.h 19](#_Toc106958896)

[Sourcecode 4: Eigene Typedefs für vereinfachung des Codes - ./src/kernel\_driver/ioct\_cmd.h 20](#_Toc106958897)

[Sourcecode 5: IOCT Kommandos - ./src/kernel\_driver/ioct\_cmd.h 20](#_Toc106958898)

[Sourcecode 6: Device-Read Zugriff Funktion - ./src/kernel\_driver/led\_matrix\_driver.c 21](#_Toc106958899)

[Sourcecode 7: Schreiben in den Treiber - ./src/kernel\_driver/led\_matrix\_driver.c 21](#_Toc106958900)

[Sourcecode 8: IOCT Kommandos - ./src/kernel\_driver/led\_matrix\_driver.c 22](#_Toc106958901)

[Sourcecode 9: Makefile Targets - ./src/kernel\_driver/Makefile 22](#_Toc106958902)

# 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

# Vorwort

Im Rahmen des Faches System-Programmierung wurde mithilfe eines Raspberry PI 4, innerhalb einer Linux Distribution, mit dem Namen PI OS, ein Kernel-Treiber für eine LED-Matrix programmiert. Unter der Aufsicht von Andreas Laubhahn wurden die aufgelisteten Materialien 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.

In dieser Arbeit wird primär der Kernel Treiber erläutert und dessen innerer Aufbau aufgezeigt. Zuletzt wird unser Nutzerapplikation erklärt. Probleme, auf welche wir gestoßen sind, finden dann am Ende der Arbeit noch Anklang.

Zudem kann es vorkommen, das wichtige Code Teile in dieser Dokumentation fehlen würden, dies hat den Hintergrund, dass hier nur die Schlüsselrolle spielenden Codeteile ihren Platz finden.

Für den Vollständigen Source kann das angelegte [Git-Repository](https://github.com/SenpaiSimon/Sysprog-AI-B6-Projekt) referenziert werden oder das mitgelieferte Archiv.

## Zeitplan

Dies ist nur grober Zeitplan für uns selber um das Projekt besser zu organisieren.



Abbildung Zeitplan

# Aufbau

Im Folgenden Abschnitt werden der Aufbau der Software, sowie der physische Aufbau einer LED-Matrix erklärt. Auch wie ein direktes Aufsteckboard für den Raspberry aussehen könnte, wurde über das CAD Programm KiCad umgesetzt.

## 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 und auch dessen oft sehr große Community ein wichtiges Plus. (ComputerWeekly.de 2022)

Mittlerweile gibt es hunderte verschiedene Linux-Distributionen für jeden nur erdenklichen Einsatzzweck, seien es Server, Mobilgeräte oder einfache Desktops. Grundsätzlich bestehen solche Distributionspakete aus Softwarepaketen, welche wiederrum aus Anwendungen oder Diensten bestehen. Ein solches Paket kann zum Beispiel 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 wurde. Diese entspringt der GNU General Public License (kurz „GPL“) (ComputerWeekly.de 2022).

### Aufbau und Funktion einer LED-Matrix



Abbildung Aufbau einer LED-Matrix

Eine LED-Matrix ist im Grunde ein Feld (Array), welches aus Dioden besteht. Hierbei gibt es immer einen Eingangs- bzw. Ausgangspin, welche die LEDs in einer Reihe bzw. einer Spalte verbinden. Dabei steht die Abkürzung LED für „Light Emitting Diode“, sprich „Licht Emittierende Diode“. 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 den Strom zerstört werden. Auch werden die Reihen und Spalten nicht mit 1-8 deklariert, sondern wie in der Informatik üblich mit 0-7.

Ebenso gibt es wie in Arrays keine „x“ und „y", auch „Dots“ genannt, werden mit Reihen und Spalten angesprochen (Ewald 2020).

### Umsetzung eines Raspberry Aufsteckboards

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

#### Schaltplan



Abbildung Schematischer Anschluss LED-Matrix and Raspberry

In Abbildung 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 strombegrenzenden Vorwiederstand angeschlossen. Analog dazu die Kathoden, also negativen Pins (Lx), nur ohne Vorwiederstand.

Wie diese GPIO Pins angesprochen werden wird im Kapitel „Matrix Kontroller“ erklärt.

#### Layout

Dies ist da physische Layout des vorher gezeigten Schaltplans. Hier werden die Leiterbahnen verlegt und auch die Komponenten platziert.



Abbildung 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 (Anschraublöcher) wurden an die Abstände des Raspberry PIs angepasst.

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

#### Pi-Hat PCB

KiCad bringt auch die Funktion die fertigen Platinen in 3D zu betrachten. Hier gibt es dann nur ein Bild in digitaler Form, da wir die Platine für den Rahmen dieser Arbeit nicht bestellt haben und somit nicht physisch vor uns haben.



Abbildung PI Hat PCB Rückseite

Abbildung PI Hat PCB Vorderseite

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

# 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 auch zu 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 laufenden Programme bekommen wiederum vom Kernel Prozessorzeit zugewiesen. Jeder dieser Prozesse erhält einen eigenen, geschützten Speicherbereich auf welchen nur über Systemaufrufe zugegriffen werden kann. 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, zum Beispiel durch einen Fehler, das System zum Absturz bringen kann (Wikipedia 2022).

Eine gute Veranschaulichung liefert die auf Wikipedia zu findende Grafik:



Abbildung 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 Applikation. Wie genau hierbei unser Programm funktioniert wird in im Abschnitt „User App“ näher erklärt.

Grundsätzlich braucht man einen Kernel für Schnittstellen zu Anwenderprogrammen. Gleichzeitig hat der Kernel vollen (und möglicherweise nur) Zugriff auf Prozessoren, Geräte und Speicher und weist dabei 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).

## Kernel Module für die LED-Matrix

Nachdem es uns nicht gelungen ist verschiedene C-Files zusammen zu kompilieren wurde dafür ein passend empfundener Umweg gegangen und das Programm in verschiedene Header Files modularisiert, welche dann nicht nur Prototypen enthalten, sondern eben ganze Programmstücke. Diese werden dann in die Haupt-C-File inkludiert.

### Matrix Kontroller

Zuerst beinhaltet unser Kernel Treiber die Header File für den Matrix Kontroller, in welcher der State also der Status, in welchem sich die einzelnen Pixel befinden. Hier kann der State auf 1, also „HIGH/EIN“ oder auf 0, also „LOW/AUS“ gesetzt werden. Dabei müssen immer zwei zusammenpassende GPIOs (General Purpose Input Output) gesetzt werden. Ein HIGH Treiber auf eben HIGH und der dazugehörige LOW Treiber auf LOW (Sourcecode 1). Auch kann man natürlich den Status eines bestimmten Pixels auslesen. Dies geschieht direkt an den Pins, wobei durch eine logische Operation dann der Status des angefragten Pixels berechnet wird (Sourcecode 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. Prüfen ob der gewünschte Pin vorhanden ist

2. Den Pin Anfragen

3. Die Richtung (Input/Output) wählen

4. Den Pin Exportieren ohne möglichen Richtungswechsel

Genau dies geschieht in Sourcecode 3, wobei hier nur die erste Hälfte der Initialisierung zu sehen ist. Zu sehen sind die High-Active Treiber der Matrix. Hier werden in der Header File für Pin-Definition die Pins in Arrays geschrieben und auch die Namen der Pins in Arrays geschrieben. Dies wurde gemacht um jetzt in einer einfachen for-loop über alle Pins zu iterieren und diese vier Schritte zu befolgen. Hierbei wird für jeden einzelnen Pin in jedem einzelnen Schritt geprüft, ob die Operation auch erfolgreich war. Ist dies nicht der Fall wird eine leserliche Fehlermeldung ausgegeben und die Schleifen beendet.

### I/O Control

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 3 Kernfunktionen. Zum einen das Setzen eines Pixels, dann das Auslesen eines Pixels und zum Schluss noch das Auslesen der Dimensionen, genauer gesagt die Anzahl der Pixel in jeder Reihe und Spalte. In Sourcecode 4 sind Typedef Strukturen zu sehen, welche für die leichtere Verwendung dieser Operationen angelegt wurden. Im Typedef „State“ kann man die Reihe und Spalte eines Pixel mitgeben. Der Status wird dabei entweder zum Setzen oder auch zum auslesen des Wertes genutzt. Im Typedef „Dimensions“ wird lediglich die Anzahl der Reihen und Spalten hinterlegt. Die IOCT Kommandos werden auch an dieser Stelle definiert, damit man keine Kontinuitätsfehler im Programm hat. Diese Defines werden übrigens auch in der User-App von dieser Stelle eingebunden. In Sourcecode 5 kann man diese Kommandos sehen und auch die Art des Datenteils, welcher genutzt wird. Auch kann man sehen, dass SET\_PIXEL in den Kernel schreibend ist durch die Bezeichnung „\_IOW“ (Input Output Write). Die Richtung wird dabei immer vom User-Space in den Kernel-Space angegeben. READ\_PIXEL hat zum Beispiel einen lesenden und auch schreibenden Teil, da zuerst vom User die Reihe und Spalte eingelesen wird und dann der Status wieder an den User gegeben wird.

Auf den Treiber kann auch ohne Kommandos zugegriffen werden. Dafür kann man dann „dev\_read“ oder „dev\_write“ verwenden. Hierbei kann man einfach die Kernel-Datei auslesen und bekommt vordefinierte Daten oder kann primitive Daten setzen. Bei uns kann man sich beim Lesen den Status der LED\_Matrix anzeigen lassen. Dies heißt im Klartext, dass auf der Kommandozeile Reihe für Reihe ausgeben wird mit einer „1“ wenn der Pixel an ist und einer „0“ wenn der Pixel aus ist. Die Funktion, welche diese Funktionalität bietet kann in Sourcecode 6 gefunden werden. Beim schreibenden Zugriff in den Treiber über die Kommandozeile kann man entweder eine „1“ oder eine „0“ schreiben und somit alle Pixel der Matrix analog dazu entweder ein oder ausschalten (Sourcecode 7).

### LED-Matrix Treiber

Dieser Bereich ist unsere Hauptdatei des Projekts. Sie besteht aus der einzigen C-File und einem dazugehörigen Header, welcher die Includes der anderen Programmteile und auch die Prototypen enthält.

Die C-File besteht aus den im Praktika bereits bekannten und benötigten 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.

Die „dev\_init“-Funktion wird dazu benutzt um den Treiber beim Laden initial aufzusetzen. Das Gerät wird mit einem Namen Versehen und zum Kernel hinzugefügt. Auch werden am Ende alle benötigten GPIO Pins, wie in GPIO Initialisierung beschrieben, initialisiert.

Bei der „dev\_exit“ passiert das genaue Gegenteil, hier werden zuerst alle GPIOs wieder freigegeben und vorher ausgeschaltet. Danach wird des Device aus dem Kernel gelöscht.

Hier werden auch alle Funktionen für IOCTL beziehungsweise FOPS (File Operations) definiert und dann hinterlegt. Es sind auch Funktionen möglich, welche auslösen, wenn auf das Gerät allgemein zugegriffen wird oder der Zugriff wieder beendet wird. Diese haben bei uns keine Verwendung gefunden und geben somit lediglich eine Nachricht aus um zu sehen, dass etwas passiert.

Die Kommandos für IOCTL werden hier dann durch ein Switch-Case abgehandelt und dann die jeweiligen Funktionen ausgeführt. Als Referenz kann hierbei Sourcecode 8 dazu gezogen werden.

## Makefile

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

Als Referenz für diesen Abschnitt kann Sourcecode 9 dazu gezogen werden. In unserem Makefile werden die Source-Files gebildet und auch das Kernel-Objekt erzeugt. Zusätzlich wird auch eine Compiler-Flag gestzt, um den Treiber mit dem C99 Standard zu Kompilieren. Dies hat den Hintergrund, dass for-schleifen angenehmer verwendet werden können.

Es wurden mehrere Targets definiert um die Workflows angenehmer zu gestalten. Ruft man nur „make“ auf wird das das Default-Target aufgerufen und nur der Kernel gebildet. Dies ist allerdings etwas unschön, da beim bilden sehr viele temporäre Dateien entstehen und dann auch nach dem Bilden immer noch vorhanden sind. Deshalb wurde noch ein „Clean-Target“ angelegt, welches durch Wildcards alle Temporären Dateien wieder entfernt und nur das Kernelobjekt bestehen bleibt.

Ein weiteres sehr nützliches Target ist das „install-Target“, aufrufbar mit „sudo make install“. Dieses Target muss mit Root rechten aufgerufen werden, da es das erzeugte Kernelobjekt auch direkt in den Kernel lädt und man somit diesen Schritt auch nicht selber machen muss. Am Schluss wird noch das Kernelobjekt gelöscht um die Dateien um Source Verzeichnis sauber zu halten.

# 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.

## 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);  }  }  } |

## 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));  }  } |

## 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;  } |

# 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.

## Install Skripte

Install skripte erklären

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## 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) !!!! VORWORT ANPASSEN

# Code-Ausschnitte



Sourcecode : Einzelnen Pixel setzen - ./src/kernel\_driver/controller/matrix\_controller.h



Sourcecode : Status eines einzelnen Pixel erhalten - ./src/kernel\_driver/controller/matrix\_controller.h



Sourcecode : Initialisierung der High Treiber - ./src/kernel\_driver/gpio\_inits/gpio\_init.h



Sourcecode : Eigene Typedefs für vereinfachung des Codes - ./src/kernel\_driver/ioct\_cmd.h



Sourcecode 5: IOCT Kommandos - ./src/kernel\_driver/ioct\_cmd.h



Sourcecode 6: Device-Read Zugriff Funktion - ./src/kernel\_driver/led\_matrix\_driver.c



Sourcecode : Schreiben in den Treiber - ./src/kernel\_driver/led\_matrix\_driver.c



Sourcecode 8: IOCT Kommandos - ./src/kernel\_driver/led\_matrix\_driver.c



Sourcecode 9: Makefile Targets - ./src/kernel\_driver/Makefile

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.