## **Ubiquitous Computing**

### **Labor 1: Introduction to Arduino**

Laborteilnehmer: Marcel Kanne & Christoph Kaiser

1 Basic working with Arduino IO, PWM and Serial

1.1 First Task to be done

Aufgabenstellung:

Bringe eine LED zum Blinken und zum Verblassen. Was ist der Unterschied zwischen beiden Realisierungen? Beschreibe das Vorgehen in ein paar Wörtern und gebe ein Beispiel. Du kannst dazu das Tutorial verwenden.

<https://www.arduino.cc/en/Tutorial/Blinken>

<https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay>

Im ersten Part der Versuchsreihe wird untersucht, wie wir mit Hilfe eines Arduino die physikalischen Ausgänge verwenden können. Dazu bringen wir die auf dem Arduino verbaute LED zum Blinken.

Um das folgende Vorgehen vollziehen zu können, benötigen wir ein Arduino oder ein Genuino Board, eine LED (Farbe nach Wahl) und einen 220 Ohm Widerstand. Wenn die OnBoard LED des Arduinos verwendet werden soll, wird zu Durchführung ausschließlich ein Arduino verwendet. Die zusätzliche LED sowie der 220 Ohm Widerstand können hierbei vernachlässigt werden. Die OnBoard LED wird parallel zum digitalen Ausgang 13 (D13) des Arduinos angesprochen. Dieser Ausgang ist bei jeglichen Versionen des Arduinos identisch und wird auch LED\_BUILTIN genannt. So können wir die OnBoard LED einfach ansteuern, ohne dabei zusätzliche Hardware anzubringen. Die Arduino Software (IDE) stellt beim Anlegen einer neuen Datei 2 Funktionen bereit, die automatisch generiert werden. In der Funktion setup() werden Konfigurationen festgelegt, die beim Einschalten des Arduinos übernommen werden sollen. In dieser Funktion legen wir zum Beispiel fest, welche Pins mit welchem Pinmode betrieben werden soll. Der Befehl pinMode benötigt hierzu 2 Parameter, zum einen den Aufgang, der angesprochen werden soll und ob es sich bei dem Ausgang um einen Output oder Input Pin handelt. Bei Output Pins wird ein Signal, bzw. eine Spannung angelegt, während bei Input auf ein Signal gewartet wird (zB. über einen Taster oder einen Sensor). Da die OnBoard LED eine Spannung von Pin 13 erwartet, müssen wir somit Pin 13 als Output deklarieren. Dies geschieht über die Codezeilen pinMode(LED\_BUILTIN, OUTPUT);.

Die Funktion loop() beherbergt jeglichen Code, der sequentiell und wiederholt abgelaufen werden soll. Mit dem Befehl digitalWrite können wir einem Ausgang einen Wert zuweisen. Hierfür werden wieder 2 Parameter benötigt. Zum einen den Ausgang, zum anderen ob der Ausgang ein oder ausgeschaltet werden soll. Somit schreiben wir den Code digitalWrite(LED\_BUILTIN, HIGH);, um den Ausgang 13 mit 5V Spannung zu versorgen. Die LED fängt somit an zu leuchten. Mit dem Befehl digitalWrite(LED\_BUILTIN, LOW); schalten wir den Ausgang 13 wieder aus. Da beide Befehle hintereinander zu schnell abgearbeitet werden, werden wir nicht sehen können, dass die LED eingeschaltet wurde. Deshalb benötigen wir einen Delay, welches die LED über eine variable Zeit eingeschaltet lässt. Der Befehl delay() benötigt als Parameter nur eine Zeitangabe in Form eines Integers, wobei die Zahl in Millisekunden angegeben wird. Schreiben wir delay(1000); zwischen beiden digitalWrite Befehlen, wird die Spannung für 1 Sekunde an der LED gehalten, bevor sie ausgeschaltet wird. Das Problem bei dem Befehl Delay ist, dass der Arduino auf keine weiteren Inputs reagiert, oder anderen Code abarbeiten kann, da der Arduino das Programm pausiert. Signale gehen somit verloren. Um die LED ohne Delay leuchten lassen zu können, benötigen wir also eine andere Vorgehensweise. Eine Lösung wäre, wenn wir beim Durchlaufen der loop() Funktion die aktuelle Zeit messen und intern verrechnen. Wird ein bestimmtes Zeitkontingent überschritten, wird der aktuelle Status (On, Off) getauscht. Der Zustand ist somit nicht mehr vom Delay abhängig, sondern von der Zeit. Das System bleibt also ansprechbar ohne dabei Signale zu verlieren. Um die aktuelle Zeit abfragen zu können, verwenden wir den Befehl millis(). Dieser liefert als Rückgabewert die Zeit in Millisekunden, als das Board gestartet wurde. Läuft das Board seit 1 Sekunde, liefert millis() somit 1000 als Ergebnis. Mit dem Befehl if können wir eine Bedingung abfragen, ob ein Zustand erreicht wurde. Mit if (currentMillis - previousMillis >= interval) fragen wir ab, ob die aktuelle Zeit minus der zuletzt abgefragten Zeit ein bestimmtes Intervall überschritten hat. Ist dies der Fall, wird previousMillis auf den aktuellen Zeitstempel gesetzt und der LED Status wird getauscht (Von HIGH auf LOW und anders herum). Hierfür benötigen wir wieder eine Bedingung, um den aktuellen LED Status abzufragen. Den aktuellen Status erhalten wir mit if (ledState == LOW). Sollte der Status LOW sein, wird der Status auf HIGH gesetzt. Sollte der Status bereits auf HIGH sein, wird dieser auf LOW gesetzt. Nun wird nur noch der Ausgang passend zum Status mit digitalWrite(LED\_BUILTIN, ledState); gesetzt.

1.2 Second Task to be done

Aufgabenstellung:

Programmiere einen kleinen Lautsprecher (Buzzer). Beschreibe das Vorgehen in eigene Wörter und gebe ein Beispiel dafür, wie ein Buzzer funktioniert und was man benötig, um einen Buzzer in Betrieb zu nehmen.

<https://www.arduino.cc/en/Tutorial/toneMelody>

In Part zwei der Versuchsreihe wird untersucht, wie wir mit Hilfe eines Buzzers individuelle Töne ausgeben lassen können.

Um das folgende Vorgehen vollziehen zu können, benötigen wir ein Arduino oder ein Genuino Board, einen Piezo Buzzer oder Lautsprecher, sowie einige hook-up Kabel, um die verwendete Hardware miteinander zu verbinden. Der Pluspol des Lautsprechers wird mit dem digitalen Pin 8 des Arduinos verbunden, der Minuspol entsprechend an dem GND Port (Masse). Für unseren Code verwenden wir die Arduino IDE inklusive Bibliothek "pitchers.h". Diese Bibliothek beinhaltet alle Pitchwerte der typischen Noten. Mit NOTE\_C4 können wir zum Beispiel den mittleren Ton C generieren. Die angegebene Noten Tabelle wurde von Brett Hagman geschrieben und wird für die Basistöne der Arduinofunktion tone() verwendet.

Im ersten Teil des Quellcodes wird die Bibliothek eingebunden. Hierfür schreiben wir #include "pitchers.h". Nun benötigen wir ein Array, welches alle vorkommenden Noten in gewünschter Reihenfolge beinhaltet. Als nächstes benötigen wir ein Array, welches alle gewünschten Zeiten beinhaltet. In unserem Beispiel verwenden wir viertel und achtel Noten. In der Funktion setup() deklarieren wir eine for-Schleife, die durch alle in dem Array deklarierten Noten iteriert. Hierzu schreiben wir for (int thisNote = 0; thisNote <8; thisNote++). In der Schleife selbst greifen wir auf das Array mit den viertel- und achtel Noten zu und berechnen uns mit int noteDuration = 1000 / noteDurations[thisNote] die Zeit des abzuspielenden Tons. Nun kommt erstmals die Funktion tone() zum Einsatz. tone() benötigt zwei bis drei Parameter. Der erste Parameter gibt an, an welchem Arduino Ausgang der Buzzer angeschlossen ist (hier Ausgang 8). Der zweite Parameter gibt an, welcher Ton, bzw. welche Frequenz ausgegeben werden soll und der dritte Parameter, wie lange der Ton in Millisekunden ausgegeben werden soll (optional). Wird der dritte Parameter nicht angegeben, wird der Ton so lange abgespielt, bis die Funktion noTone(8); geschrieben wird. Hierdurch wird erreicht, dass auf Pin 8 des Arduinos das Signal nicht weiterhin gesendet wird. Die loop() Funktion wird an dieser Stelle nicht benötigt, da alle notwendigen Funktionen bereits in der setup() bzw. In eigenen Funktionen deklariert sind.

1.3 Third Task to be done

Aufgabenstellung:

Erweitere die Übung aus 1.1 mit einem Taster oder einem Buzzer und Timer. Verwende zur Durchführung die bisher gelernten Inhalte und beschreibe das Vorgehen in eigene Wörter.

Im dritten Part der Versuchsreihe erweitern wir die Funktionalität der ersten Übungsaufgabe. Hierzu benötigen wir die bisher verwendete Hardware, sowie einen Taster und einen hohen Widerstand mit zum Beispiel 10k Ohm. Der hohe Widerstand dient als pull-down Widerstand und lässt die nicht benötigte Spannung, die auf den Taster gespeist wird, verpuffen. Dies dient zum Schutz der Schaltung, da es sonst zu einem Kurzschluss käme, wenn der Taster nicht gedrückt wird. Um den Taster in das Projekt einzubinden, speisen wir 5V der Versorgungsspannung an den Taster und klemmen auf der gegenüberliegenden Seite des Tasters den pull-down Widerstand auf GND (Masse). Zwischen Taster und pull-down Widerstand wird nun ein hook-up Kabel angeschlossen und mit einem freien digitalen Pin des Arduinos verbunden. Der Buzzer wird wie in 1.2 in das Projekt eingebunden. Bisher haben wir für die Interaktion der digitalen Pins nur die Funktion digitalWrite() verwendet. Nun benötigen wir zum Lesen eines Signals die Funktion digitalRead(). Als Parameter wird hier der Ausgang (hier Eingang) des Arduinos angegeben, an dem der Taster angeschlossen ist. Da ein digitaler Pin nur Werte zwischen 1 und 0 (HIGH, LOW) annehmen kann, erhalten wir mit der Funktion digitalRead(Taster\_Pin) den Wert, der an dem Pin anliegt. Wird der Taster gedrückt, werden die 5V Versorgungsspannung des Tasters an den digitalen Pin weitergeleitet. Somit erhalten wir 1 oder HIGH. Damit ein digitaler Pin Signale empfangen soll, müssen wir in der setup() Funktion den Pin auf INPUT setzen. Hierzu schreiben wir in die Methode setup() pinMode(Taster\_Pin, INPUT);. Nun müssen wir nur abfragen, welcher Wert an dem digitalen Pin anliegt und dem entsprechend handeln. Zum Abfragen des Pins schreiben wir if (digitalRead(Taster\_Pin) == 1). Wird der Taster gedrückt, ist die If-Bedingung wahr und wir treten in den weiteren Verlauf ein. In der If-Bedingung selbst können wir nun einen Ton mit tone(Ton\_Pin, 500, 1000); ausgeben lassen. Hierbei handelt es sich um einen Ton mit 500Hz, der 1 Sekunde auf Pin Ton\_Pin ausgegeben wird. Wird der Taster nicht gedrückt, werden die 5V Versorgungsspannung des Tasters direkt über den pull-down Widerstand an GND (Masse) abgeleitet. Somit erhalten wir mit einem digitalRead(Taster\_Pin); eine 0 oder ein LOW.

1.4 Third Task to be done

Aufgabenstellung:

Lerne die serielle Kommunikation zwischen verschiedene Module kennen. Lese und verstehe, was eine serielle Kommunikation ist, was es für andere Möglichkeiten gibt und welche Vor- und Nachteile diese nach sich ziehen. Beschreibe hierfür zwei zusätzliche Kommunikationsmethoden.

<https://www.arduino.cc/en/Tutorial/DigitalReadSerial>

Im vierten Part der Versuchsreihe wird untersucht, wie wir mittels der seriellen Schnittstelle zwischen dem Arduino und dem Computer (über USB) kommunizieren können.

Hierzu benötigen wir erneut einen Arduino. Um den Wert eines Tasters über die serielle Schnittstelle ausgeben lassen zu können, benötigen wir außerdem einen Taster mit entsprechendem 10k Ohm Widerstand, sowie einige hook-up Kabel und zur übersichtlicheren Verkabelung ein Steckbrett. Der Taster und der Widerstand werden wie bereits in 1.3 erläutert in das Projekt eingebunden.

Im Programmcode müssen wir nun die serielle Schnittstelle öffnen und die Anzahl Bits pro Sekunden der seriellen Schnittstelle festlegen. Hierzu schreiben wir in der Methode setup() Serial.begin(9600);. Im nächsten Schritt muss der Pin deklariert werden, welcher auf den Taster lauschen soll. Angenommen Pin 2 wird zum Lesen des Tasters verwendet, so schreiben wir pinMode(2, INPUT);. Wir verwenden INPUT, da Pin 2 ein Signal erwartet und kein Signal schicken soll. Einen digitalen Wert kann man mit Hilfe der Print-Methode ausgeben lassen. Hierzu ist es sinnvoll, den digitalen Wert in einer Variablen zwischen zu speichern. Da digitale Pins wie bereits erläutert nur einen Wert zwischen 1 und 0 annehmen können, benötigen wir hierfür den Datentyp Integer. Mit int buttonState = digitalRead(2); speichern wir den Wert temporär in die Variable buttonState ab und können somit auch im Nachhinein auf diesen Wert zugreifen. Zum Ausgeben des Wertes greifen wir nun auf eine Print-Methode zurück, wobei wir zwischen print und println unterscheiden. Während println nach einer Ausgabe einen Zeilenumbruch hinzufügt, wird dieser bei der print Methode vernachlässigt. Somit können wir zum Beispiel den Wert mit Zeilenumbruch ausgeben, indem wir Serial.println(buttenState); schreiben. Der Zusatz Serial ist notwendig, um dem Arduino mitzuteilen, dass die Ausgabe über den Serial-Port geschickt werden soll. Es gibt neben Serial weitere Möglichkeiten, Ausgaben zu tätigen. Beispielsweise lassen sich mittels eines Displays Printangaben erstellen, wobei hierfür eine zusätzliche Hardware in Form eines Displays oder einer LED Matrix benötigt werden. Außerdem benötigen wir dann zum Ansprechen des Displays die dazugehörige Bibliothek, die in das Projekt eingebunden werden muss.

1.5 Third Task to be done

Aufgabenstellung:

Kombinieren Sie Aufgabe 1 bis Aufgabe 4 und seien Sie dabei kreativ. Erstelle dazu ein eigenes Programm, welches LED, Buzzer, Timer und die serielle Kommunikation verwendet. Beschreibe deine Idee und wie du sie umsetzen würdest.

Im fünften und letzten Teil der Versuchsreihe der ersten Aufgabenstellung wird das bisher Erlernte zusammengefasst, um die neugesammelten Erfahrungen zu festigen. Hierzu benötigen wir jegliche Hardware, die bereits eingesetzt wurde, sprich fünf LEDs (Rot\_Gross, Grün\_Gross, Orange\_Gross, Rot\_Klein, Grün\_Klein), einen kleinen Lautsprecher, einen Taster, Widerstände, ein Steckbrett, hook-up Kabel, sowie einen Arduino. Zur Lösung der Aufgabenstellung soll eine Ampelsteuerung implementiert werden, die bei Bedarf (Knopfdruck durch Passanten) geschaltet werden kann. Wird der Taster betätigt, sollen Fahrzeuge durch das Schalten der LEDs benachrichtigt werden, dass diese nicht weiter die Ampel passieren dürfen. Fußgänger hingegen sollen zudem durch ein akustisches Signal signalisiert bekommen, dass diese die Straße queren dürfen. Nach Ablauf des Timers soll der ursprüngliche Zustand wieder zurückkehren, sodass die Fahrzeuge die Straße queren können. Damit Passanten und Fahrzeugführer genau wissen, wann wer an der Reihe ist, soll zudem über eine serielle Schnittstelle signalisiert werden, in welchem Zustand sich die Ampelsteuerung befindet. Die komplette Steuerung wird in Form einer Zustandsmaschine realisiert.

Die LEDs werden wie in Versuch 1 in das System eingebunden, wobei jede LED einen eigenen digitalen Ausgang zur Verfügung gestellt bekommt. Dabei ist zu beachten, dass vor jeder LED ein Vorwiderstand mit ca. 180 Ohm vorgeschaltet werden muss, da diese sonst bei er zu großer Spannung überhitzen und sonst kaputt gehen würden. Der Buzzer und der Taster werden wie in 1.2 und 1.4 eingebunden, dabei ist darauf zu achten, dass keines der Ausgänge zwei Mal belegt ist. Jedes Bauteil erhält somit seinen eigenen digitalen Ausgang. Die Ausgänge der LEDs sowie der Buzzer werden mit dem pinMode Befehl so konfiguriert, dass diese als Ausgang fungieren. Angenommen die fünf LEDs und der Buzzer werden in der Reihenfolge D2 bis D7 angeschlossen, so können wir eine for-Schleife im setup()-Teil implementieren, um den Quellcode klein zu halten. Hierzu schreiben wir:

for (int i = 2; i <= 7; i++) { pinMode(i, OUTPUT);}

Somit wird durch jeden Ausgang von D2 bis D7 iteriert und der Ausgang auf OUTPUT gesetzt. Da der Taster wie bereits in 1.4 beschrieben ein Signal erwartet, wird der dazugehörige Ausgang mit dem Pinmode INPUT initialisiert. Nun müssen die LEDs so konfiguriert werden, dass beim Einschalten des Systems Passanten warten und Fahrzeuge passieren dürfen. Somit schreiben wir initial:

digitalWrite(Rot\_Gross, LOW);

digitalWrite(Orange\_Gross, LOW);

digitalWrite(Grün\_Gross, HIGH);

digitalWrite(Rot\_Klein, HIGH);

digitalWrite(Grün\_Klein, LOW);

Um die Funktionalität der seriellen Kommunikation einzubinden und um die Funktion setup() zu vervollständigen, benötigen wir außerdem die Codezeile Serial.begin(9600);. Nun widmen wir uns der Funktion loop(), in der die eigentliche Anlagensteuerung implementiert wird. Hierzu wäre es eine gute Möglichkeit, die Ampelsteuerung als Zustandsmaschine zu implementieren. Hierzu könnte ein Datenkonstrukt Enum eingebaut werden, welches alle möglichen Zustände beinhaltet. In den jeweiligen Zuständen selbst könnte man nun mit einem Timer festlegen, wie lange jeder Zustand beigehalten werden soll, bis der nächste Zustand in Kraft gesetzt wird. Im Ausgangszustand, in dem Fahrzeuge passieren dürfen und Fußgänger halten müssen, wird auf das Signal des Buttons gewartet. Dieser stößt den nächsten Zustand an, der die Ampel für die Fahrzeuge auf orange stellt und grün ausschaltet. Kommt man in den Zustand, dass Fußgänger passieren dürfen, kommt der Timer zum Einsatz. Dieser soll nach Ablauf einer festgelegten Zeit ein Signal auf den Buzzer geben. Ein Beispiel könnte so aussehen, dass der Ton mit der Frequenz 850 Hz für eine Sekunde im 2 Hz Takt ausgegeben wird. Außerdem kann nun ein Display über eine serielle Schnittstelle so konfigurieren werden, dass Fußgängern mitgeteilt wird, dass diese nun die Straße queren dürfen. Nach Durchlauf des letzten Zustandes wird in den Ausgangszustand gewechselt, indem wiederum auf das Signal des Buttons gewartet wird.

2 LED Matrix

2.1 First Task to be done

Aufgabenstellung:

Erstelle mit gegebener Hardware (Arduino + LED Matrix 16X8 LED) eine Applikation mit einem Zähler. Lese dazu die Dokumentation des Moduls und wende sie an. Erkläre was zur Durchführung benötigt wird und wie es funktioniert. Beschreibe außerdem die Parameter.

In Teil zwei der Versuchsdurchführung widmen wir uns der Ansteuerung einer 16X8 LED Matrix und untersuchen, wie wir mit dieser Technik umgehen können und wie wir sie in Projekte verwendet. Zur Durchführung wird ein Arduino mit einer 16X8 LED Matrix benötigt. 16X8 steht hierbei für die Anzahl der LEDs (16 horizontal, 8 vertikal). Zur Hardwareverbindung benötigen wir die beiden Pins SDA und SCL der Matrix, die wir mit den entsprechenden Pins am Arduino (A4 und A5) verbinden. Zur Stromversorgung verbinden wir außerdem GND und 3,3 V. Um die Matrix über Quellcode ansprechen zu können, wird eine Bibliothek benötigt, mit welcher wir die LED Matrix ansteuern können. Eine gängige Bibliothek ist die Adafruit Bibliothek. Diese binden wir mit der Codezeile "import adafruit\_ht16k33.matrix" in das Projekt ein und erstellen eine Instanz mit "matrix = adafruit\_ht16k33.matrix.Matrix16x8(i2c)". Um mit der LED Matrix über die Schnittstelle I2C zu kommunizieren, benötigen wir außerdem die Bibliotheken board sowie bitbangio, welche wir mit import board und "import bitbangio as io" initialisieren. Auch hier benötigen wir eine Instanz, die wir mit "i2c = io.I2C(board.SCL, board.SDA)" erstellen. Beim Testdurchlauf der LED Matrix kann es vorkommen, dass am Ende noch einige LEDs aufleuchten. Dies hängt damit zusammen, dass der Speicher nicht aufgeräumt wird. Mit der Codezeile matrix.fill(0) setzen wir alle LEDs auf den Zustand AUS, bzw. überschreiben den Speicher der Matrix mit Nullen. Möchten wir die erste LED zum Leuchten bringen, schreiben wir matrix[0, 0] = 1. Die Ansteuerung folgt somit über die Dimensionen der Matrix. Die neue Matrix wird standardmäßig automatisch übernommen, diese Funktion kann man jedoch mit matrix.auto\_write = False ausschalten. Somit muss man jegliche Änderung mit matrix.show() übernehmen. Um nun einen Zähler zu implementieren, benötigen wir für jegliche Zahlen eine individuelle Matrix, in der die benötigten LEDs eingeschaltet werden. Wir setzen also zunächst alle LEDs auf 0 damit nicht mehr benötigte LEDs ausgeschaltet werden und rufen dann die neue Matrix auf, die die benötigte Zahl ausgibt. Nun könnte man mit Hilfe einer Zustandsmaschine definieren, welche Zahl als nächstes ausgegeben werden soll. Die Zahlen müssten hierfür vorher in Arrays deklariert werden, wobei die einzelnen LEDs, die für die jeweilige Zahl benötigt werden, eingeschaltet werden. Der Zustand soll sich nach 1 Sekunde ändern. Dies erreichen wir, in dem wir am Ende des Zustandes delay(1000); schreiben und somit eine Sekunde in den jeweiligen Zustand verharren, bis dann anschließend in den nächsten Zustand gewechselt wird. Möchten wir nun auch noch die Helligkeit der einzelnen LEDs anpassen, können wir matrix.brightness = 0 schreiben. Die Helligkeitsstufen sind von 0 bis 15 definiert. Außerdem kann mit matrix.blink\_rate = 1 die Blinkrate der Matrix angepasst werden. Diese ist in 4 (0 bis 3) Stufen definiert und lässt die Matrix von 0 bis 0,5 Hz blinken.

2.2 Second Task to be done

Aufgabenstellung:

Zähle mit den LEDs von 0 bis 128. Du kannst dazu die Funktion “matrix.drawPixel(X, Y, COLOR)” verwenden. Wie funktioniert diese Funktion und welche Parameter gibt es? Wie bindet man dies in das Projekt ein?

In Abschnitt zwei der zweiten Versuchsreihe wird mit dem in 2.1 erläuterten Versuchsaufbau ein Zähler implementiert, der von 0 bis 128 zählen und die aktuelle Zahl auf dem Display ausgeben soll. Auch hier können wir mit Hilfe einer Zustandsmaschine den Zähler wie in 2.1 definieren, wobei die Erstellung der einzelnen Zahlen durch individuelles Einschalten der benötigten LEDs sehr viel Zeit beanspruchen würde. Um uns die Arbeit abzunehmen, ändern wir die Bibliotheken mit #include <Adafruit\_GFX.h> und #include <Adafruit\_SSD1306.h>. Diese Bibliotheken benötigen keine eigenständige Erstellung der Matrix. Das Anschließen des Displays folgt wie in 2.1. Der Reset Pin des Displays definieren wir mit "#define OLED\_RESET 4" und "Adafruit\_SSD1306 display(OLED\_RESET);". Nun können wir in der Setup-Funktion unserer Anwendung definieren, welche Adresse das Display hat, welches wir mit display.begin(SSD1306\_SWITCHCAPVCC, 0x3C); einstellen. Außerdem können wir die Textgröße, die Cursor Farbe und die Cursorposition festlegen. Dies geschieht mit

display.setTextSize(1);

display.setTextColor(WHITE);

display.setCursor(0, 0);.

In der Funktion loop() schreiben wir eine for-Schleife, die von 0 bis 128 iteriert.

for (int i = 0; i <= 128; i++)

Mit display.println(i); geben wir nun in der Schleife den aktuellen Wert des Zählers auf dem Display aus. Im Anschluss wird mit delay(1000); eine Sekunde gewartet, bis dann anschließend die Zählvariable i um eins hochgezählt wird. Im nächsten Durchlauf wird also nicht mehr der Wert 0,

sondern der Wert 1 ausgebeben. Ist die Schleife durchgelaufen, wird die Zählvariable wieder mit 0 initialisiert und die Prozedur fängt von neu an. Somit geben wir nun kontinuierlich die Zahl 0 bis 128 auf dem Display aus. Alternativ könnte man einen Modulozähler in die Schleife bauen, wobei abgefragt wird, ob es sich um den Wert um eine gerade oder ungeraden Zahl handelt. Handelt es sich bei dem Wert der Variablen um einen geraden Wert, wird die Zahl in weiß geschrieben, an sonsten in blau. Wir schreiben somit in die Schleife

if (i % 2 == 0) {

display.setTextColor(WHITE);

} else {

display.setTextColor(BLUE);

}, um zwischen zwei Farben hin und her zu wechseln.

2.3 Second Task to be done

Aufgabenstellung:

Zeichne eine einfache Bitmap auf das Display. Du kannst hierzu die Funktion “matrix.drawBitmap(0, 8, om\_bmp, 8, 8, HT16K33\_BLINK\_CMD);” verwenden. Wie funktioniert diese Funktion und was wird zudem benötigt?

Im dritten Part des zweiten Abschnitts soll mit Hilfe des zuvor verwendeten Displays eine Bitmap ausgegeben werden. Hierzu benötigen wir eine Rastergrafik, die aus einer rasterförmigen Anordnung von Pixeln besteht, die jeweils einer Farbe zugeordnet sind. Ein Beispiel hierfür könnte wie folgt aussehen:

static const unsigned char PROGMEM logo16\_glcd\_bmp[] =

{ B00000000, B11000000,

B00000001, B11000000,

B01111100, B11110000,

B01110000, B01110000,

B00000000, B00110000 };

Wir haben somit eine Bitmap definiert, die wir mit display.drawBitmap(30, 16, logo16\_glcd\_bmp, 16, 16, 1); auf das Display ausgeben lassen können. DrawBitmap wird mit 6 Parametern aufgerufen, wobei die ersten beiden Parameter die X und Y Koordinaten entsprechen, auf dem das eigentliche Bild platziert werden soll. Der dritte Parameter enthält das eigentliche Bild, welches auf dem Display ausgegeben werden soll. Im vierten und fünften Parameter wird angegeben, wie breit, bzw. wie hoch die Darstellung sein soll. Der letzte Parameter gibt an, in welcher Farbe das Bild dargestellt werden soll. Um eigene Bitmaps erstellen zu können, gibt es online einen jpg zu bitmap Konverter, der das zu konvertierende Bild in eine Rastergrafik umrechnet.

2.4 Third Task to be done

Aufgabenstellung:

Gebe einige Texte auf dem Display aus. Hierzu kannst du die Funktion “matrix.print("Hello");” und “matrix.setCursor(x,0);” verwenden. Wie funktioniert es, welche weiteren Optionen gibt es, wie reagiert das Display auf zu lange Texte und was passiert mit dem Speicher?

Im vierten Part der zweiten Versuchsreihe wird untersucht, wie die Matrix/Display-Funktionen verwendet werden und wie sich eine zu lange Ausgabe auf das Display auswirkt. Außerdem soll überprüft werden, welche weitere Optionen zur Verfügung stehen und was mit dem Speicher passiert, wenn das Display bereits voll beschrieben wurde.

Für diesen Versuch benötigen wir wie im Versuch zuvor einen Arduino und eine Matrix bzw. ein Display. Wir verwenden während unserer Versuchsdurchführung ein 0.96" OLED Display mit 128x64er Auflösung und I2C Anbindung. Wie auch in Versuch 2.2 erläutert, verbinden wir zunächst die analogen Pins A4 und A5 des Arduinos, sowie GND und +3,3V mit dem Display und binden zudem die Adafruit\_GFX und Adafruit\_SSD1306 Bibliothek ein. Um das Display nun verwenden zu können, benötigen wir den Code display.begin(SSD1306\_SWITCHCAPVCC, 0x3C);, um dem Arduino die Adresse des Displays mitzuteilen. Die Funktion display.print("Hello"); schreibt wie auch bei der seriellen Ausgabe ein Hello, mit dem Unterschied, dass die Ausgabe nicht auf dem seriellen Monitor, sondern auf dem Display projiziert wird. Die Funktion display.setCursor(x,0); setzt den Text, der ausgegeben werden soll an die Position, die als Parameter übergeben wird. Initial wird der Text an die Position 0,0 platziert. Weitere Funktionen sind display.setTextSize(1); und display.setTextColor(WHITE);, die die Textgröße und die Textfarbe anpassen. Mit display.clearDisplay() wird der gesamte Displayinhalt entfernt. Werden zu lange Texte auf dem Display ausgegeben, wird der Text von links nach rechts und von oben nach unten auf das Display projiziert, bis das Display vollgeschrieben ist. Weitere Texte werden ignoriert und nicht mehr ausgegeben. Da das Display einen eigenen Speicher enthält, wird der zuletzt übertragene Text beim Ausschalten beibehalten und bei Wiedereinschalten des Systems erneut ausgegeben, ohne dass der Text erneut vom Arduino übertragen werden muss.

3 LED Matrix as Terminal Output

Kombiniere das Wissen aus den beiden Versuchsreihen zuvor in einen Versuch und schreibe einen LED-Banner, welches die über das Terminal gesendeten Daten des Arduinos anzeigt. Beschreibe wie dein Programm funktioniert und wie dein Konzept lautet. Überlege vor der eigentlichen Programmierung, wie das Programm aufgebaut werden sein soll.

In der dritten Versuchsreihe soll ein LED-Banner programmiert werden, welches Terminaleingaben des Arduinos an die LED-Matrix übermittelt und dort ausgibt. Hierzu benötigen wir wiederum einen Arduino, sowie einen LED-Banner, den wir wie in den Aufgaben zuvor anschließen und einbinden. Um nun Daten über die serielle Schnittstelle entgegen zu nehmen, benötigen wir die Funktion Serial.read();. Diese liest die Eingabe im seriellen Monitor und kann dann in einem char-Array zwischengespeichert werden. Somit benötigen wir einen globalen Array, welches zB. 512 Werte zwischen speichern kann, was wir mit char str[512] erreichen. Zudem benötigen wir einen Zähler, mit dem wir auf die Indices des char Arrays zugreifen können. Diesen Zähler erstellen wir als globale Variable mit int count = 0;. Im setup()-Teil des Programms wird die Adresse sowie die Datenrate der seriellen Verbindung und des Displays festgelegt. Dies geschieht mit den Funktionen Serial.begin(9600); und display.begin(SSD1306\_SWITCHCAPVCC, 0x3C);. Im Nächsten Schritt konfigurieren wir das Display mit

display.setTextSize(1);

display.setTextColor(WHITE);

display.clearDisplay();

und setzen so die Textgröße auf 1, die Textfarbe auf Weiß und alle LEDs zurück. Nun ist das Display vorbereitet und kann die terminal Eingaben entgegen nehmen. In der loop()-Funktion benötigen wir eine if-Bedingung, die prüft, ob eine serielle Schnittstelle verfügbar ist, bzw. ob ein Input gelesen werden kann. Mit str[count++] greifen wir auf die erste ([0]) Stelle des Arrays zu und speichern den entgegenkommenden Wert mit (char) Serial.read();. Die Typumwandlung (char) wird an dieser Stelle benötigt, da die Werte sonst in ASCII-Code gespeichert würden. count++ sorgt dafür, dass der Zähler nach Ende des Zugriffs um eins erhöht wird. Die Eingabe soll beendet und an das Display übermittelt werden, sobald ein "\n" eingegeben wurde. Diese Art Zeichenketten zu beenden ist in der Programmiersprache C üblich. Im nächsten Schritt prüfen wir somit, ob an der vorletzten Stelle ein \ und an der letzten Stelle ein n eingelesen wurde. Ist dies der Fall, wird der bisherige Displayinhalt gelöscht und der Cursor auf den Anfangszustand gesetzt. Da sich in unserem char-Array die Zeichen \n befinden, wir diese aber nicht mir ausgegeben haben möchten, benötigen wir eine for-Schleife, die das Array von Anfang bis count-2 ausgibt. So werden die Zeichen \n nicht berücksichtigt. In der Schleife selbst können wir nun mit dem schleifeneigenen Index auf die Indices des Arrays zugreifen, was wir mit display.print(str[i]); erreichen. Da i nach jedem Durchlauf der Schleife um eins hochgezählt wird, erreichen wir, dass jeder eingespeicherte Buchstabe von links nach rechts ausgegeben wird. Am Ende der Prozedur aktualisieren wir das Display mit display.display();.

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define OLED\_RESET 4

Adafruit\_SSD1306 display(OLED\_RESET);

char str[512];

int count = 0;

void setup() {

Serial.begin(9600);

display.begin(SSD1306\_SWITCHCAPVCC, 0x3C);

display.setTextSize(1);

display.setTextColor(WHITE);

display.clearDisplay();

}

void loop() {

if (Serial.available() > 0) {

str[count++] = (char) Serial.read();

if (str[count - 2] == '\\' && str[count - 1] == 'n') {

display.clearDisplay();

display.setCursor(0, 0);

for (int i = 0; i < count - 2; i++) {

display.print(str[i]);

}

count = 0;

}

}

display.display();

}

4 Wire layout for temperature measurement(Arduino)

Aufgabenstellung:

Der Thermometer DS18B20 kann in den zwei Betriebsmodi "Parasite-Powerd" und "External Supply" betrieben werden. Wähle die Betriebsart, die du bevorzugst. Lese dazu das Datenblatt, um die Vor- / und Nachteile der Betriebsmodi kennen zu lernen.

4.1 Implementation

Der digitale Thermometer verwendet für die Kommunikation das One-Wire Protokoll. Glücklicherweise stellt uns die Arduino IDE dieses Protokoll zur Verfügung. Dieses ist in der one-wire Bibliothek hinterlegt und kann somit in unser Projekt eingebunden werden. Für diese Aufgabenstellung werden wir die zur Verfügung gestellte Bibliothek verwenden. Lade dir die Bibliothek dafür auf der Seite <http://playground.arduino.cc/Learning/OneWire> herunter und integriere sie in deinem Projekt.

Im ersten Part der vierten Versuchsdurchführung werden wir einen digitalen Thermometer mit der Seriennummer (DS18B20) an einen Arduino anbinden. Hierzu benötigen wir die one-wire Bibliothek, die wir von einer externen Seite herunterladen. Um den digitalen Thermometer an den Arduino anschließen zu können, benötigen wir außerdem folgende Bauteile:

• 1 digitaler Thermometer (DS18B20)

• 1 Arduino Uno oder ähnliches Board

• 1 Widerstand mit 4,7K Ω

• Kabel (Vcc = rot, GND = schwarz)

• 1 Steckbrett

Der digitale Thermometer wird mit dem schwarzen (rechts, wenn glatte Seite hinten) Kabel an die GND-Leitung des Arduinos angeschlossen. Die Stromversorgung wird über das rote Kabel initiiert. Das Signalkabel (gelb, Mitte) wird an einem digitalen Ausgang des Arduinos verbunden. Der 4,7k Ohm Widerstand wird mit 5V Versorgungsspannung betrieben, wobei das andere Ende des Widerstandes an demselben digitalen Ausgang angebunden wird, an dem auch das Signalkabel des Thermometers angeschlossen wurde. Diese Art den Thermometer anzubinden nennt man external-supply und ist die störungsempfindlichste Verdrahtung, was in unserem Fall aber keine allzu große Rolle spielt, da die Entfernung zum Arduino größer als 5 Meter betragen müsste, um einen deutlichen Unterschied aufzuzeichnen. Eine weitere Möglichkeit den Sensor anzubinden wäre es die rote Leitung mit an GND zuschalten, wobei man diese Art der Verbindung parasitäre nennt. Der Vorteil gegenüber dem external-supply ist, dass die Speisekondensatoren genügend Zeit bekommen sich wieder aufzuladen. So könnte man auch Strecken über 5 Meter problemlos überwinden. Um komplizierte Berechnungen, bedingt durch das Umrechnen der ankommenden Bits des Sensors, zu vermeiden, binden wir die externe Bibliothek DallasTemperature.h mit dem #include Befehl in das Projekt ein. Um die Funktionalitäten der Bibliothek nutzen zu können, benötigen wir eine Instanz, die wir sensors nennen. Hierzu schreiben wir DallasTemperature sensors(&oneWire);, wobei sensors einen Zeiger auf das Objekt der oneWire Bibliothek erhält. Um uns die Temperatur im zweistelligen Nachkommabereich ausgeben zu lassen, benötigen wir außerdem eine globale Variable Celsius mit dem Datentyp float, da Integer Typen ausschließlich ganz stellige Zahlen enthalten dürfen. Mit sensors.begin(); in der Funktion setup() öffnen wir die Schnittstelle des Sensors, damit die Daten übermittelt werden können. Mit sensors.requestTemperatures(); erhalten wir nun die Sensorwerte in unsere Instanz sensors und können nun die Daten mit der Funktion Celsius = sensors.getTempCByIndex(0); in unsere Celsiusvariable speichern. getTempCByIndex(0); rechnet dabei die Sensordaten automatisch in Grad Celsius um.

4.2 Temperature reading

Aufgabenstellung:

Mit Hilfe des Arduinos und dem zuvor beschriebenen Aufbau des digitalen Thermometers (DS18B20) im external-supply Modus sollen die Sensordaten aufgezeichnet und über die serielle Schnittstelle im seriellen Monitor ausgegeben werden.

Im zweiten Part der vierten Versuchsreihe soll mit Hilfe eines Arduinos und dem digitalen Thermometer (DS18B20) eine temperaturabhängige Ausgabe auf dem Bildschirm erfolgen. Hierzu schließen wir den Sensor wie in 4.1 erläutert an den Arduino an und treffen die nötigen Vorkehrungen (siehe 1.4), mit denen wir die serielle Schnittstelle in das Projekt einbeziehen. Hierzu schreiben wir im Konfigurationsabschnitt setup() Serial.begin(9600); und die serielle Schnittstelle zu öffnen und die Datenrate auf 9600 BPS festlegen. Im Abschnitt loop() schreiben wir nun

sensors.requestTemperatures();

Celcius = sensors.getTempCByIndex(0);

Serial.print("Grad Celsius: ");

Serial.print(Celcius);

delay(1000);

Hiermit erreichen wir, dass nach jeder Sekunde die Sensordaten des Sensors abgefragt und in Grad Celsius umgerechnet und anschließend über die serielle Schnittstelle ausgegeben werden.

4.3 LED scale

Aufgabenstellung:

Erweitere das Projekt um eine LED Matrix und gebe dort die Temperatur aus.

Im letzten Part der vierten Versuchsreihe zum Thema Temperatursensor wird das momentane Projekt, bestehend aus einem Arduino und einem Temperatursensor vom Typ DS18B20, um eine LED Matrix erweitert, sodass die Temperaturangabe des Thermometers auf dem Display ausgegeben wird. Hierfür binden wir das Display, wie beispielsweise bereits in 2.4 erläutert, in das bestehende Projekt ein. Außerdem verwenden wir den in 4.2 verwendeten Quellcode und entfernen die Stellen, in denen Serial.print(xy) vorkommt und ersetzen diese durch display.print() Anweisungen. Mit display.print(Celsius); erreichen wir, dass die globale Variable Celsius nicht mehr über die serielle Schnittstelle, sondern auf dem Display ausgegeben wird.