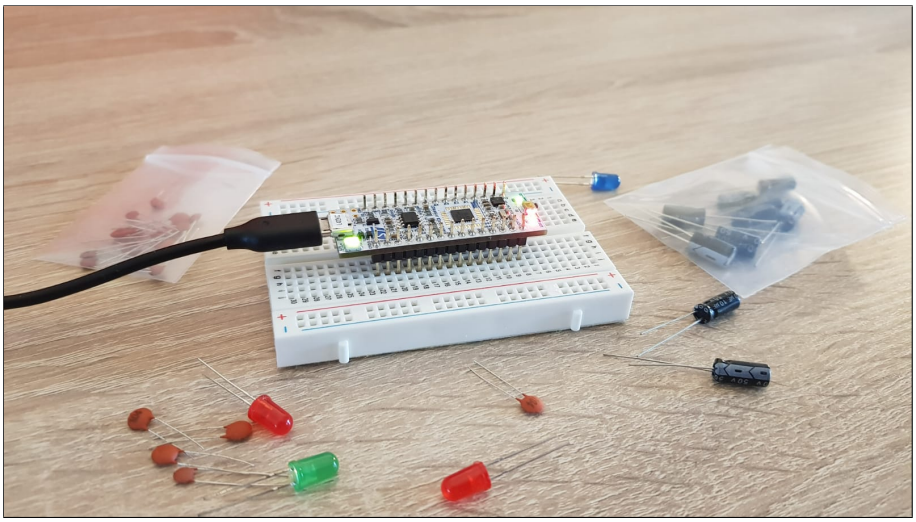
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Computer Engineering

Rechnerorganisation Beleg 2



STM43L432KC

Hai Nam La [s0578105]

Dozent: Prof. Dr. Sebastian Bauer

07.Januar 2023

Inhaltsverzeichnis

1 Vorwort **4**

2 Abstract **4**

3 Melody **5**

3.1 Vorbetrachtung 5

3.2 Implementierung 5

4 Colorcycle **7**

4.1 Vorbetrachtung 7

4.2 Implementierung 8

5 Jukebox **9**

5.1 Vorbetrachtung 9

5.2 Implementierung 10

6 ADC **13**

6.1 Vorbetrachtung 13

6.2 Implementierung 13

7 LDR **14**

7.1 Vorbetrachtung 14

7.2 Implementierung 14

8 Charger **16**

8.1 Vorbetrachtung 16

8.2 Implementierung 16

9 Fazit **18**

Abbildungsverzeichnis **19**

Literatur **20**

1 Vorwort

Alle Aufgaben wurden mithilfe der STMCubeMX gebaut, wobei VSCode als IDE genutzt wurde. Die Software wird dann auf dem STM32L432KC Nucleo Board getestet indem eine ".bin" erstellte Datei, durch make, auf das STM32 Board geflashed wird. Achtung die genutzten Bilder sind alle selbst gemacht aber nicht im nachhinein überprüft worden. Die Projekte funktionieren aber die Anordnung einiger Bauteile sind nicht gemäß der "Normen". Für die richtige Anordnung bitte auf die Schaltbilder achten. Außerdem ist dieser Beleg eine weiterführung vom ersten Beleg, in der nicht spezifisch auf alle Elemente eingegangen werden sondern davon ausgegangen wird, dass der 1. Beleg ebenfalls durchgegangen wurde. Die jeweilige Software wird auf dem Gitlab Repository zu finden sein, unter den selben Namen wie der Überschrift.

2 Abstract

Es wird im 2. Labor des Moduls Rechnerorganisation (abgekürtzt RO oder CO) vorwiegend mit dem Timer gearbeitet. Dabei soll schlussendlich eine Jukebox gebaut werden mit Buzzer, Taster und einer RGB LED. Insgesamt wird es 3 Aufgaben geben:

1. Melody: Einen beliebig gewählten Song abspielen
2. Colorcycle: Mithilfe einer RGB LED die Farben des Farbkreises darstellen
3. Jukebox: Mit einem Taster zwischen mindestens 3 Songs wechseln als auch die abgespielten Töne durch Farben darzustellen

Als 2. große Teilaufgabe wird mit dem ADC gearbeitet und Spannungswerte mit Nutzung eines LDRs und die Lade- und Entladekurve eines Kondensators überprüft.

3 Melody

3.1 Vorbetrachtung

Es soll mithilfe eines passiven Buzzers eine Melody abgespielt werden. Dazu wird der Timer genutzt beziehungsweise die PWM generation des Timers. Dafür wird GPIO PA1/TIM2 genutzt. Ebenfalls wird ein Vorwiderstand von ($220\,\Omega$) genutzt welches von der Aufgabenstellung gegeben war.

3.2 Implementierung

Der Buzzer wird mit den Vorwiderstand in reihe zum GPIO PA1 und GND verbunden.(Abbildung 1)

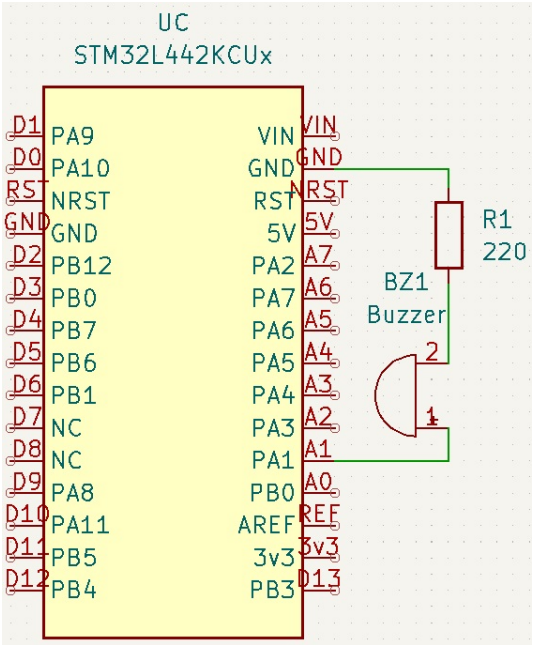


Abbildung 1: Einfache Buzzer Schaltung[1].

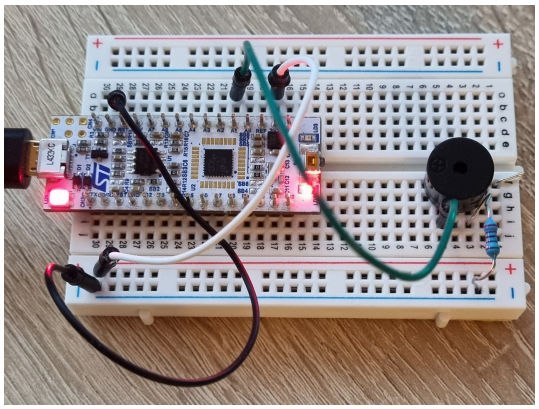


Abbildung 2: Angeschlossener Buzzer[2].

Es werden 3 Funktionen deklariert, welche die Schritte zur Musikgenerierung zusammenfassen:

1. frequency_change(int note)
2. play_note(int ryth)
3. play(int note, int rythm)

Dabei wird in der Funktion frequency_change() der Timer konfiguriert für die Note die abgespielt werden soll. Es wird der AutoReloadRegister für die Note berechnet und gesetzt, sowie ein Duty Cycle von 50% durch HAL Funktionen.

```
void frequency_change(int note) {
  ARR = clock/note;
  CCR = ACC/2;
  __HAL_TIM_SET_COUNTER(&htim2, 0);
  __HAL_TIM_SET_AUTORELOAD(&htim2, ARR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, CCR);
};
```

Dabei setzt die COUNTER Funktion den Timer immer beim Aufruf zurück, darauffolgend wird der AutoReloadRegister und schlussendlich der CCR mit einem Duty Cycle von 50% gesetzt. In der play_note() Funktion werden dann die ganzen Takte der Melody bestimmt und die Note dann abgespielt.

```
void play_note(int rythm){
  wholenote = 240000 / tempo;
  if(rythm <= 16 && rythm >= 0){     // normal note timing
    duration = wholenote / rythm;
  }else if(rythm < 0){               // dotted note timing
    duration = wholenote / abs(rythm);
    duration *= 1.5;
  }else{                             // tied note timing
    duration = wholenote * 10 / rythm;
    duration *= 0.5;
```

```
  }
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
  HAL_Delay(duration*0.9);
  HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_2);
  HAL_Delay(duration);
};
```

Die letzte Funktion play() ruft dann die beiden oben genannten Funktionen wobei ein Parameter $note$ und $rythm$ erwartet wird, damit die Note mit dem Takt abgespielt werden kann.

```
void play(int note, int rythm){
  frequency_change(note);
  play_note(rythm);
};
```

4 Colorcycle

4.1 Vorbetrachtung

Es soll mit einer RGB LED alle Farben eines Farbkreises dargestellt werden. Die RGB LED besitzt dafür 3 interne LEDs die alle durch Nutzung einer PWM angesprochen werden können. Dabei hat jede LED eine Intensität von 0 bis 255, wobei die Intensität und Mischung der LEDs eine Farbmischung darstellen, wie zum Beispiel: orange = (rot:255, grün:127, blau:0) . Es werden für alle 3 LED Anschlüsse TIM1 verwendet, da TIM2 wegen der Buzzer Anschluss über PA1 nur noch 2 weitere Anschlüsse zur verfügung stellt. Für die LEDs werden folgende Anschlüsse benutzt:

1. rote LED: PA8/TIM1

2. blaue LED: PA10/TIM1

3. grüne LED: PA9/TIM1

Wie auch in Rechnerorganisation Beleg 1 beschrieben benötigen LEDs Vorwiderstände und es werden dabei für alle LED Anschlüsse $220\,\Omega$ Widerstände benutzt.

4.2 Implementierung

Bei der Verschaltung einer RGB LED ist zu beachten, dass es 4 Pins gibt wobei 3 Pins die 3 LEDs: rot, grün und blau darstellen und der letzte Pin den GND.

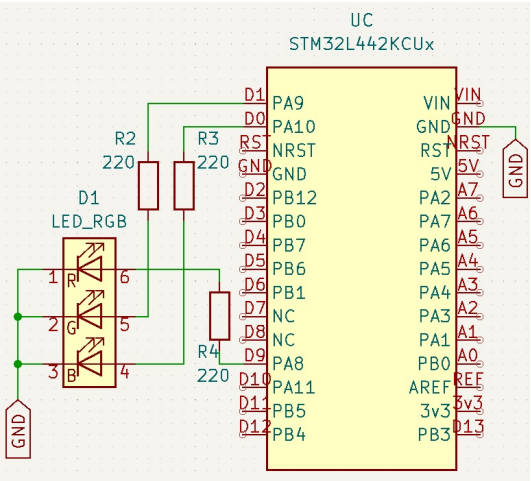


Abbildung 3: RGB Schaltung[3].

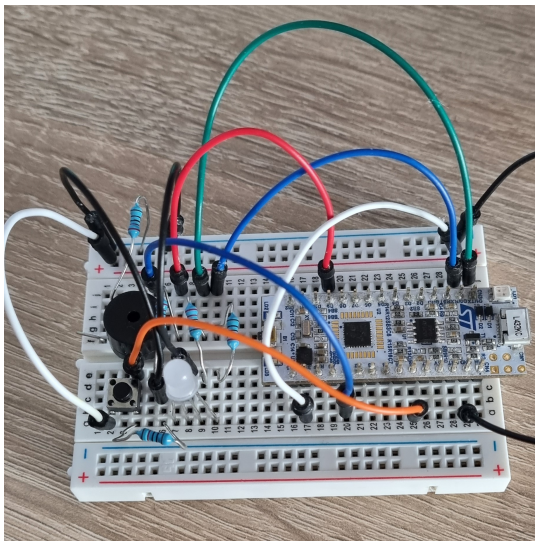


Abbildung 4: RGB mit Taster[4].

Die Nutzung von RGB LEDs ist hierbei sehr simpel. Es wird eine Funktion set_color(int red, int green, int blue) definiert:

```
void set_color(int red, int green, int blue){
  htim1.Instance->CCR1 = red * 255;
  htim1.Instance->CCR2 = green * 255;
  htim1.Instance->CCR3 = blue * 255;
}
```

Diese Funktion benutzt 3 Parameter, welche die Intensität der Farbe darstellt. Wird die Funktion aufgerufen, so wird der CCR des TIM1 für den jeweiligen Channel gesetzt, wobei alle drei Parameter erneut mit 255 multipliziert werden um einen maximal Tastgrad von 100% zu erreichen, wenn als Intensität 255 eingesetzt wird und 50% bei einer Intensität von 127 etc.

5 Jukebox

5.1 Vorbetrachtung

Es werden die beiden Aufgaben Melody und Colorcycle fusioniert und eine Jukebox gebaut. Dazu kommt ein Taster der die abgespielte Musik der Melody ändert. Die RGB Farben sollen sich an den abgespielten Tönen anppassen. Dazu wurden die gleichen Töne an eine Farbe gebunden:

1. C3, C4, C5 : rose
2. C3 sharp, C4 sharp, C5 sharp :magenta
3. D3, D4, D5 : violet
4. D3 sharp, D4 sharp, D5 sharp: blue
5. E3, E4, E5 : azure
6. F3, F4, F5 : cyan
7. F3 sharp, F4 sharp, F5 sharp :spring green
8. G3, G4, G5 : green
9. G3 sharp, G4 sharp, G5 sharp :chartreusse

10. A3, A4, A5 : yellow

11. A3 sharp, A4 sharp, A5 sharp :orange

12. H3, H4, H5 : red

Es wird, wie in Rechnerorganisation Beleg 1, ein EXTI Interrupt PA7 und Timer TIM1 für den Taster gewählt und die gleichen Pin Belegungen wie bei Melody und Colorcyle.

5.2 Implementierung

Es werden die Schaltungen von Melody und Colocycle zusammengelegt, wozu noch ein Taster mit pull up angeschlossen wird.

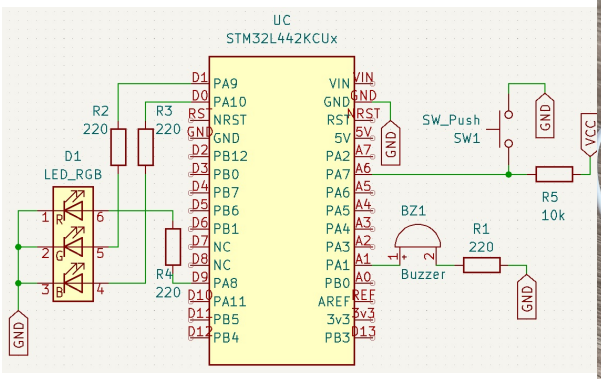


Abbildung 5: Jukebox Schaltung[5].

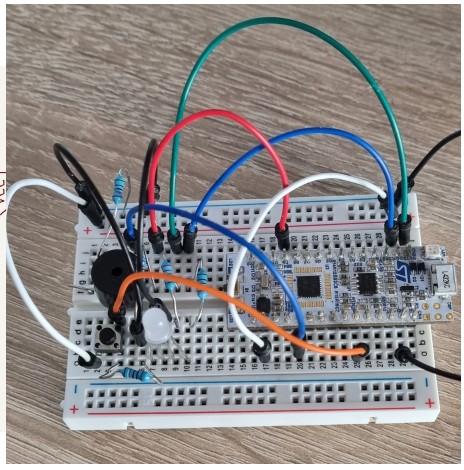


Abbildung 6: Jukebox mit Taster, RGB und Buzzer[6].

Dazu werden insgesamt 4 Musikbeispiele dazu gegeben.

1. godfather

2. monkey island theme song, welches von der Aufgabe vorgegeben war

3. imperial march

4. tetris

Es wird wenn der Taster gedrückt wird ein Interrupt mit debounce Timer aufgereufen, der dann immer die Variable "count" erhöht und ab 4 wieder zurücksetzt. Somit werden in der Haupt while(1) Schleife in einer switch case die Musikbeispiele durchgewechselt

```
while (1){
  if (count == 0) {
    imperial_march();
  };
  if (count == 1) {
    tetris();
  };
  if (count == 2) {
    godfather();
  };
  if (count == 3) {
    monkey_island();
  };
```

Auch während ein Musikbeispiel abgespielt wird kann zwischendurch gewechselt werden, durch Implementiereung einer Art polling. Dazu wird in der Funktion play_note() immer die Variable count des Tasters mit der Variable "status_buffer" des abgespielten Liedes verglichen. Wird also der der Taster gedürckt, so ist der status_buffer != count und das nächste Musikbeispiel wird abgespielt.
Um dann die RGB LED den Tönen anzupassen wird die Funktion get_note(int note) implementiert, obwohl der Name set_notecolor(int note) eher gepasst hätte.

```
int* get_note(int note){

  if(note == 131 || note == 261 || note == 523){
    return rose;
  }else if(note == 138 || note == 277 || note == 554){
    return magenta;
  }else if(note == 147 || note == 293 || note == 587){
    return violet;
  }else if(note == 155 || note == 311 || note == 622){
    return blue;
  }else if(note == 165 || note == 329 || note == 659){
    return azure;
  }else if(note == 174 || note == 349 || note == 698){
    return cyan;
  }else if(note == 185 || note == 370 || note == 740){
    return spring_green;
  }else if(note == 196 || note == 392 || note == 784){
    return green;
  }else if(note == 207 || note == 415 || note == 830){
    return chartreuse;
  }else if(note == 220 || note == 440 || note == 880){
    return yellow;
  }else if(note == 233 || note == 466 || note == 932){
    return orange;
  }else{
    return red;
  }
}
```

Diese Funktion wird in der, in Colorcyle definierten, Funktion set_color() aufgerufen, die dann in der, in Melody aufgerufenen, Funktion play_note() aufgerufen wird um

immer während der Tonausgabe die Farbe anzuschalten.

Als Demo wurde ein etwa 1 minütiges Video zur Funktion gemacht welches auf der HTW cloud zu finden ist.

6 ADC

6.1 Vorbetrachtung

Es wird mit dem ADCs des STM32 Boards Spannungswerte gemessen. Dazu wird der Pin PA3/ADC1_IN8 als ADC konfiguriert.

6.2 Implementierung

Es werden Spannungen gemessen durch Implementierung folgender Sequenz:

```
  while(1){
  /*start adc and save values in variables value/voltage and time*/
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
  value = HAL_ADC_GetValue(&hadc1);
  time = HAL_GetTick();

  /* variables Convert to string for transmission*/
  sprintf(str_time, "%u\t", time);
  sprintf(msg, "%li\r\n", value);

  /*transmit to terminal/stdout*/
  HAL_UART_Transmit(&huart2, (uint8_t*)str_time, strlen(str_time), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
}
```

Dazu wird der ADC zuallererst gestartet und die eingelesenen Werte werden dann in Variablen gespeichert. Um dann die gepeicherten Werte zu überprügen werden die Werte in strings umgewandelt und anschließend durch die HAL Funktion HAL_UART_Transmit() ausgegeben. Es werden kleine Tests durchgeführt und folgende Werte gemessen:

1. AREF = 4095
2. GND = 0

Dabei ist bei AREF mit der Referenzspannung von 3V3 ein Wert von 4095 gemessen worden, welches ein sinnvolles Ergbenis ist, wenn bedacht wird, dass der ADC ein 12-bit ADC ist und somit der höchst mögliche Output $x^{12} = 4096 - 1$ ist. Auch bei GND ist ein sinnvoller Wert zu betrachten, wenn bedacht wird, dass GND = 0.

7 LDR

7.1 Vorbetrachtung

Die Software der LDR Messung wird im selben Verzeichnis des ADC zu finden sein. Ein LDR ist ein Fotowiderstand, welches seinen Widerstandswert je nach auffallendem Licht verändert. Dabei ist der Widerstand am geringsten wenn "viel" Licht auf den Widerstand fällt und am größten wenn "kein" Licht auf den Widerstand fällt.

7.2 Implementierung

Es wird folgende Schaltung implementiert, mit einem Spannungsteiler, mit ($20\,000\,\Omega$), in der Abbildung 7.
Dazu wird mithilfe von R die Werte in einem Graphen ausgewertet.
In Abbildung 9 ist zu sehen, dass anfangs eine Spannung bei etwa 2V2 gemessen

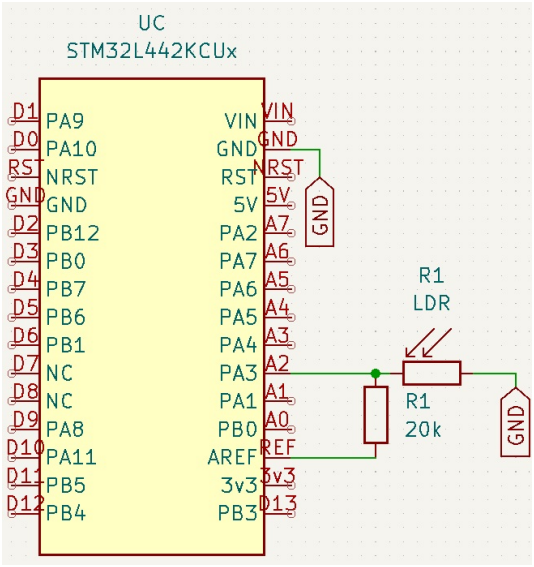


Abbildung 7: LDR Schaltung[7].

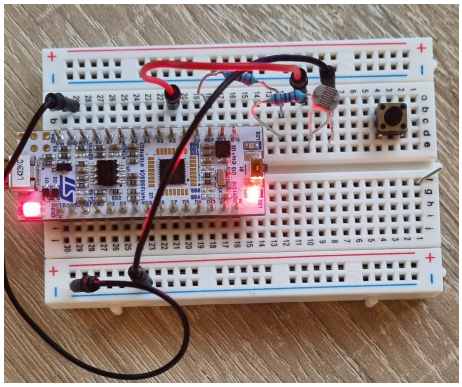


Abbildung 8: Angeschlossener LDR[8].

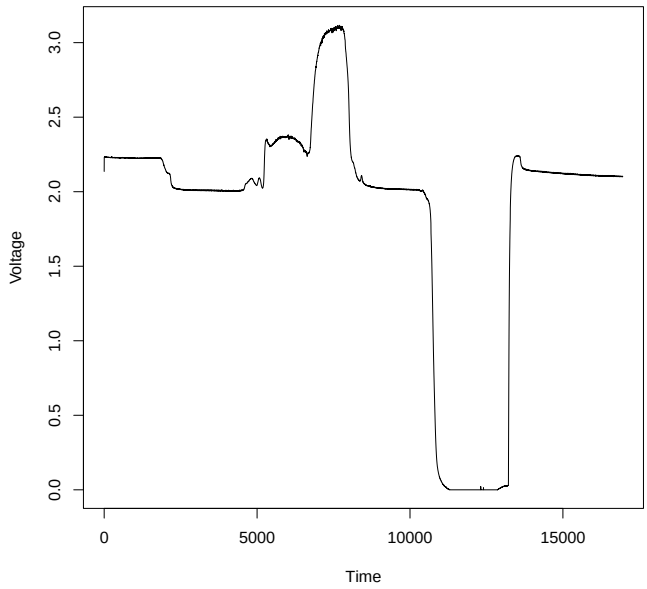


Abbildung 9: Graph mit Spannung in Abhängigkeit zur Zeit für LDR[9].

wird, die explosionsartig bei etwa 6000ms auf 3V3 wächst und bei etwa 11000ms auf 0V. Grund ist, dass beim Anstieg meine Smartphonelampe auf den LDR geschienen hat und beim Abfall meine Hand über dem LDR war. Zu erkennen ist, dass der erwartete Verlauf eines LDRs zu erkennen ist.

8 Charger

8.1 Vorbetrachtung

Ein Kondensator besitzt die Eigenschaft sich "aufladen" und "entladen" zu können. Dabei wird durch den ADC die Auflade-, sowie Entladekurve aufgezeichnet und in einem Graphen dargestellt. Es wird dabei ein Taster benutzt, um so den Kondensator durch Tastendruck Aufladen oder Entladen zu können. Der Taster wird wie in den anderen Fällen per EXTI Interrupt PB7 und Timer TIM1 integriert. Natürlicherweise wird auch ein GPIO benutzt um den Ladevorgang zu starten oder abzubrechen: PB4.

8.2 Implementierung

Es wird die Schaltung wie in Abbildung 10 zu sehen implementiert.
Bis auf die Nutzung des Interrupts mit Timer ändert sich nicht viel an der Software vergleichsweise zum ADC und LDR. Dabei wird durch das Drücken des Tasters nur der GPIO PA7 angesteuert und entweder an oder aus geschalten. Nach Wert Aufnahme wird folgender Graph, in Abbildung 12, mit R dargestellt.
Dabei ist zu erkennen, dass nach etwa 6000ms wenn der taster gedrückt wird, eine Ladekurve zu sehen ist. Genauso ist nach etw 15000ms nach erneutem Tasterdruck eine Entladekurve zu sehen.

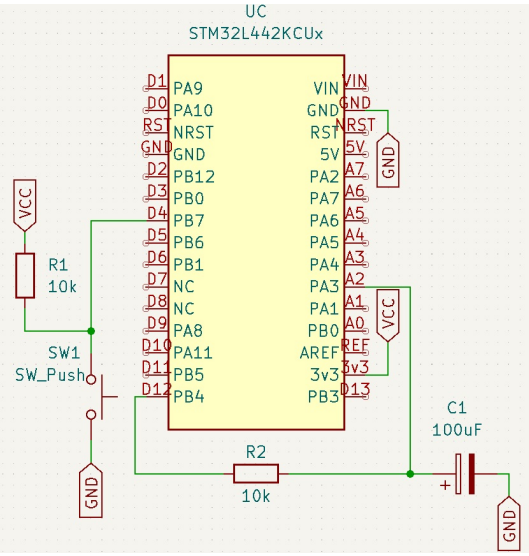


Abbildung 10: Kondensatorschaltung mit Taster und ADC[10].

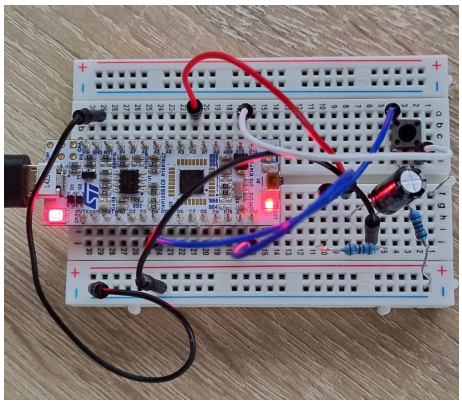


Abbildung 11: RGB mit Taster[11].

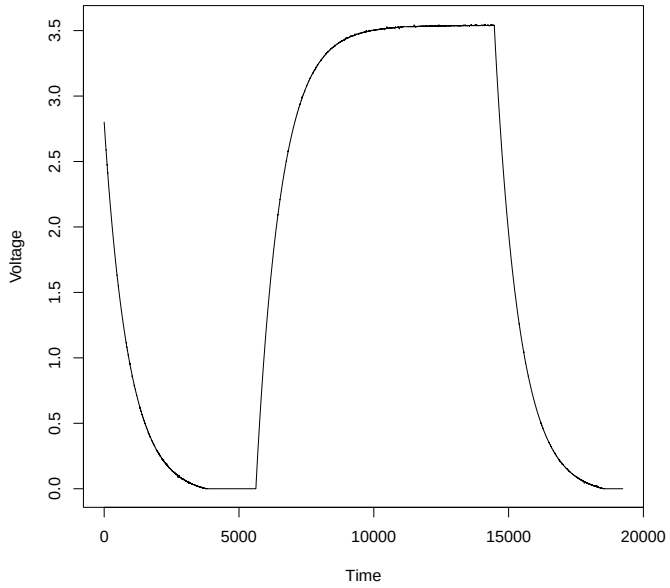


Abbildung 12: Graph mit Spannung in Abhängigkeit zur Zeit für Kondensator[12].

9 Fazit

Im 2. Labor des Moduls Rechnerorganisation wird die Programmierung eines STM32L432KC Boards vertieft und auf 2 Aspekte des Boards eingegangen: PWM und ADC. Dabei wird mithilfe des Timers des Boards ein PWM erzeugt und genutzt, sowie mithilfe des ADCs des Boards Spannungen konvertiert und somit gemessen. Wie im 1. Beleg bauen die Aufgaben aufeinander auf, wobei Jukebox auf Colorcycle und Melody und Charger auf LDR und ADC aufbauen. Es wurde in diesem Beleg gelernt:

- Nutzung des Timers für PWM Generierung
- Lesen von Notenblättern
- Nutzung eines passiven Buzzers
- Nutzung eines RGB LEDs
- Nutzung des ADCs
- Eigenschaften eines LDRs
- Eigenschaften eines Kondensators
- Erstellen von Graphen mithilfe von R
- Allgemeines C programmieren

Abbildungsverzeichnis

1 Einfache Buzzer Schaltung 5

2 Angeschlossener Buzzer 5

3 RGB Schaltung 8

4 RGB mit Taster 8

5 Jukebox Schaltung 10

6 Jukebox mit Taster, RGB und Buzzer 10

7 LDRSchaltung 15

8 Angeschlossener LDR 15

9 Graph mit Spannung in Abhängigkeit zur Zeit für LDR 15

10 Kondensatorschaltung mit Taster und ADC 17

11 RGB mit Taster 17

12 Graph mit Spannung in Abhängigkeit zur Zeit für Kondensator 17

Literatur

[1] **Prof. Dr. Bauer** https://gitlab.rz.htw-berlin.de/bauers/ce24-co-wise-22, 01.2023