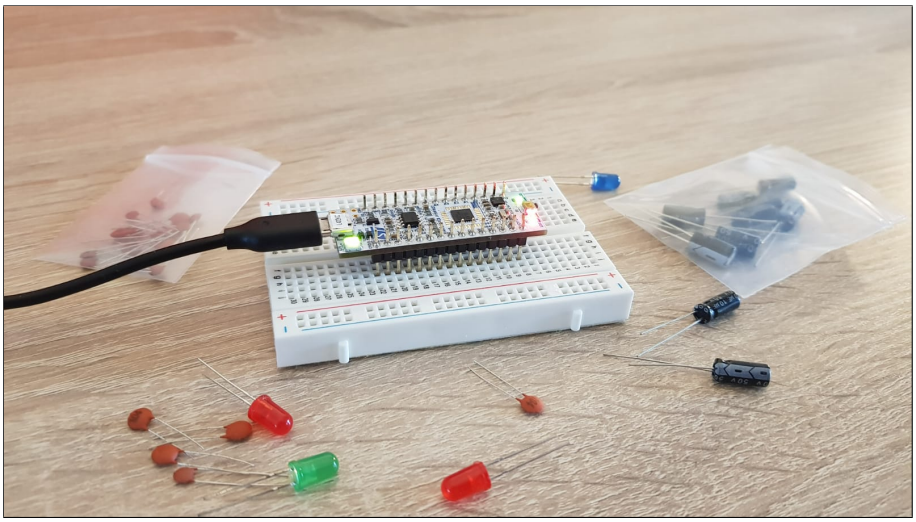
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Computer Engineering

Rechnerorganisation Beleg 1



STM32L432KC

Hai Nam La [s0578105]

Dozent: Prof. Dr. Sebastian Bauer

06.Januar 2023

Inhaltsverzeichnis

1 Vorwort **3**

2 Abstract **3**

3 BlinkyTNG **4**

3.1 Vorbetrachtung 4

3.2 Vorwiderstand 4

3.3 Implementierung 4

3.3.1 Testschaltung 4

3.3.2 LED über GPIO 6

4 Trafficlight **7**

4.1 Vorbetrachtung 7

4.2 Vorwiderstand 7

4.3 Implementierung 8

5 Trafficlight_blinky **10**

5.1 Vorbetrachtung 10

5.2 Vorwiderstand 10

5.3 Implementierung 10

6 Fazit **12**

Abbildungsverzeichnis **13**

Literatur **14**

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. Die jeweilige Software wird auf dem Gitlab Repository zu finden sein, unter den selben Namen wie der Überschrift.

2 Abstract

Im 1. Labor des Moduls Rechnerorganisation (abgekürtzt RO oder CO) befassen wir uns insgesamt mit 3 Aufgaben, welche darauf aufbaut, dass wir letztendlich ein Ampelsystem mithilfe eines Nucleo Boards STM32L432KC und externen LEDs auf einem Breadboard zusammenstellen. Die Aufgabe ein Ampelsystem nachzubauen wird in 3 Unteraufgaben geteilt:

1. BlinkyTNG: Eine externe rote LED dauerhaft leuchten zu lassen, dass sogenannte "Helloworld" der embedded Welt

2. Trafficlight: Eine Ampel mit der Nutzung von 3 externe LEDs rot, gelb und grün in einer realistischen Simulation laufen zu lassen

3. Trafficlight_blinky: Das Ampelsystem von Punkt 2 mit einer "out-of order state", durch Knopfdruck, zu erweitern.

3 BlinkyTNG

3.1 Vorbetrachtung

Um eine externe LED anzusteuern wird eine Spannungs- sowie ein GND-Anschluss genutzt. Ebenfalls wird ein Vorwiderstand verwendetet, um die LED zu schonen, da die LED selbst keinen Innenwiderstand besitzt und somit durch den "hohen" Stromfluss zerstört werden kann. Der STM32L432KC besitzt 18 nutzbare GPIOs, wobei der GPIO Pin "PB4", mit dem Namen "D12", von uns zur Steuerung der LED genutzt wird.

3.2 Vorwiderstand

Um den Vorwiderstand einer LED zu bestimmen wird folgende Gleichung verwendet: $Vorwiderstand = \frac{Spannung}{Strom}$ Dabei ergibt sich durch einsetzen der Werte und der Betrachtung, dass eine rote LED genutz wird mit 10mA:
$130\,\Omega = \frac{1.3V}{10mA}$
Genutzt wird der nächst verfügbare höhere Widerstand $220\,\Omega$.

3.3 Implementierung

3.3.1 Testschaltung

Als Test wird zuallererst eine einfache LED Testschaltung gebaut. Es wird die LED an den 3V3 Pin mit dem ausgerechneten Vorwiderstand in Reihe und den GND Pin des STM32 angeschlossen. Wird jetzt der der STM32 mit Strom versorgt durch Anstecken des Micro-USB-Kabels, so leuchtet die LED auf.(Abbildung 1 und Abbildung 2)
Die Schaltung wird dann erweitert mit einem Taster.(Abbildung 3 und Abbildung 4)

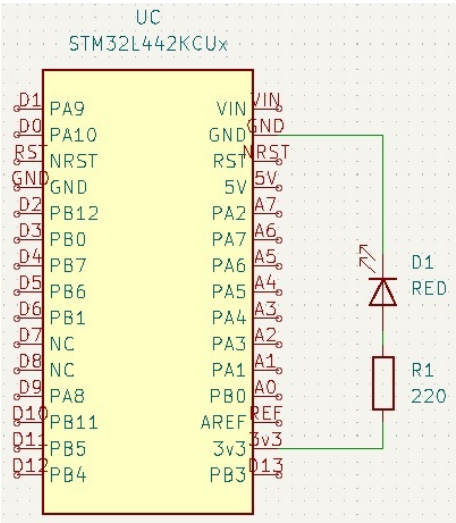


Abbildung 1: einfache LED Schaltung[1].

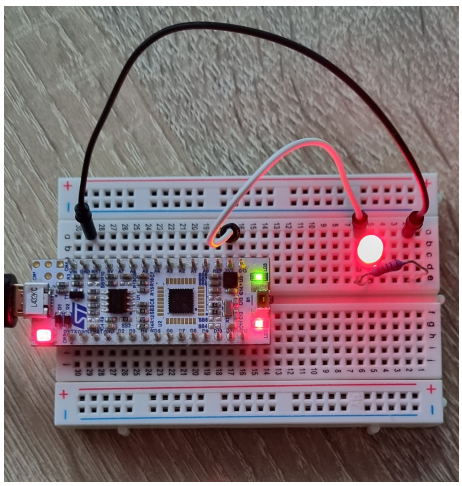


Abbildung 2: Leuchtende LED[2].

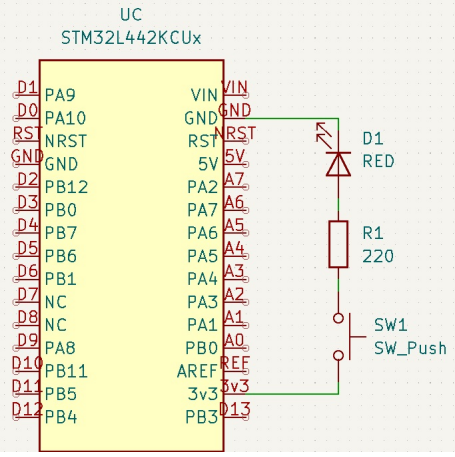


Abbildung 3: LED Schaltung mit Taster[3].

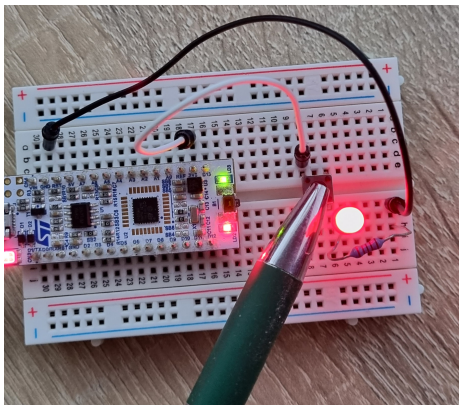


Abbildung 4: LED mit Taster[4].

3.3.2 LED über GPIO

Da für eine Ampel die LEDs angesprochen werden müssen, um ihren Zustand zu ändern,wird anstatt des 3V3 Pins ein GPIO Pin genutzt. Wir benutzen den, in der Vorbetrachtung erwähnten, Pin PB4. Dazu ist wichtig, dass der GPIO per Software angesprochen wird um einen Signal 1(3V3) zu senden. Wir nutzen die von der STM32CubeMX generierte Struktur und schreiben in der "main.c" Datei, mit Nutzung der HAL Bibliothek, eine von STM gegebene Bibliothek zur Nutzung von deren STM32 Boards, die Funktion:

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
```

Diese Funktioniert setzt dann das GPIOB Pin 4 auf 1, wodurch die LED ebenfalls zum leuchten gebracht wird. Wird diese Funktion dann in eine while(1) loop integriert und mit einem delay folgend benutzt, wird ein Blinken erzeugt mit einer Periode von 0,5 Sekunden:(Abbildung 5 und Abbildung 6)

```
while(1){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_Delay(500);
}
```

oder einfacher:

```
while(1){
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4);
    HAL_Delay(500);
}
```

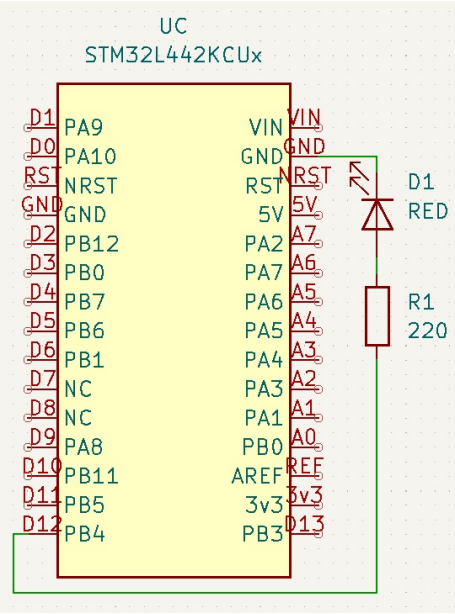


Abbildung 5: LED Schaltung mit Nutzung von PB4[5].

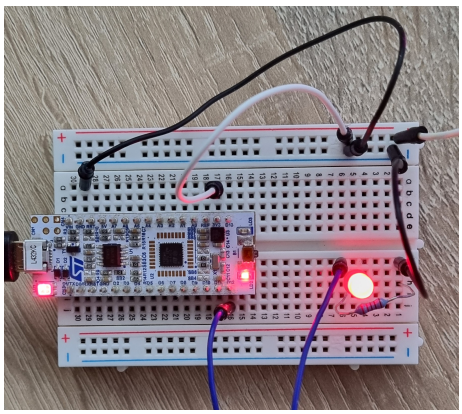


Abbildung 6: leuchtende LED mit PB4[6].

4 Trafficlight

4.1 Vorbetrachtung

Es wird das gleiche wie bei 3.1 Vorbetrachtung in Betrachtung gezogen. Dazu kommen 2 neue LEDs grün und gelb dazu, für die auch 2 weitere GPIOs benutzt werden:

1. rot = PB4
2. gelb = PB5
3. grün = PA11

4.2 Vorwiderstand

Es werden für alle LEDs Vorwiderstände von $220\,\Omega$ genutzt, welche sich daraus ergeben, dass die Widerstände durch einsetzen in die, in BlinkyTNG genannte, Gleichung

folgende ergeben:

1. rot = $130\,\Omega$
2. gelb = $130\,\Omega$
3. grün = $0\,\Omega$

und es dann einheitlicher ist den gleichen $220\,\Omega$ Widerstand zu nutzen.

4.3 Implementierung

Für die Implementierung der Ampel werden die jeweiligen LEDs mit einen, $220\,\Omega$ Vorwiderstand, den jeweiligen Pin, welcher in der Vorbetrachtung festgesetzt wurde und GND verbunden.(Abbildung 7 und Abbildung 8)

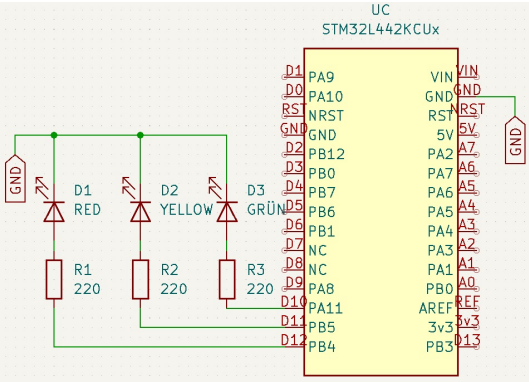


Abbildung 7: Ampelschaltung[7].

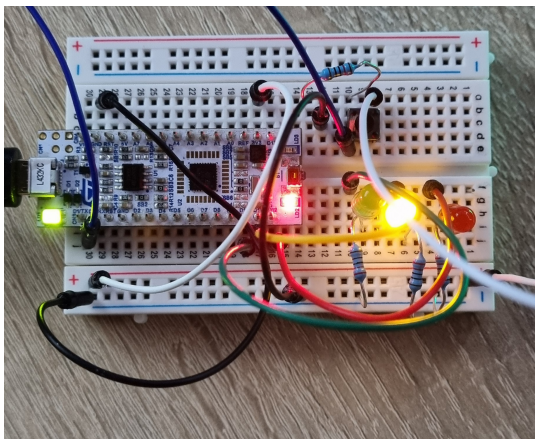


Abbildung 8: Ampel mit Taster[8].

Um eine Ampel zu realisieren wird dann in der "main.c" while(1) loop eine Sequenz von Funktionen benutzt die eine Ampel simulieren.

```
while (1){
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11, GPIO_PIN_SET); // toggle green
    HAL_Delay(1000);                                    // wait 1 second
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11, GPIO_PIN_RESET); // turn green off
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_SET); // toggle yellow
    HAL_Delay(500);                                     // wait 0.5 second
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET); // turn yellow off
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4, GPIO_PIN_SET); // toggle red
    HAL_Delay(1000);                                    // wait 1 second
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4, GPIO_PIN_RESET); // turn red off
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_SET); // toggle yellow
    HAL_Delay(500);                                     // wait 0.5 second
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET); // toggle yellow
}
```

Dabei schaltet die Ampel für jeweils 1 Sekunde grün und rot und für die Gelbphase 0.5 Sekunden.

5 Trafficlight_blinky

5.1 Vorbetrachtung

Es wird das Gleiche wie bei 4.1 Vorbetrachtung in Betrachtung gezogen. Dazu kommt, dass ein Taster genutzt wird um die "out-of-order-state" einzuschalten. Dafür wird ein EXTI Interrupt benutzt auf dem Pin PA9 und ein Timer TIM15 um einen debounce für den Taster zu simulieren.

5.2 Vorwiderstand

Es werden die selben Vorwiderstände genutzt wie in 4.2 Vorwiderstand erwähnt.

5.3 Implementierung

Es wird die gleiche Schaltung von Abbildung 7 und 8 genutzt. Zu dem Schaltbild in Abbildung 7 kommt nur ein externer Taster hinzu, der wenn er gedrückt wird, einen Interrupt auslöst, der die 4.3 Implementation aufgeführte Sequenz abbricht und die gelbe LED dauerhaft blinken lässt. Dazu wird eine Variable count genutzt die bei "count = 1" die Ampel normal laufen lässt und bei "count = else 1" die "out-of-order-state". (Abbildung 9 und Abbildung 10)
Wie schon in der Vorbetrachtung erwähnt wird ein EXTI Interrupt mit Timer für den Taster verwendet. Dafür muss zuallerst der Prescaler des Timers auf 32000 und der AutoReloadRegister des Timers auf 50 gewählt werden ($20hz(50ms) = \frac{Clock(32Mhz}{Prescaler(32000) * AutoRelo}$). Dadurch wird ein Timer von 50 ms erstellt. Wird dann der Taster gedrückt wird der Interrupt ausgeführt, welches den Timer startet:

```
    void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) // interrupt handler
{
   HAL_TIM_Base_Start_IT(&htim1); // calls timer
```

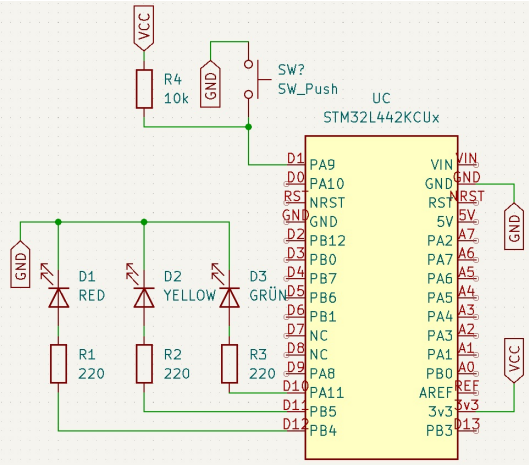


Abbildung 9: Ampelschaltung mit Taster[9].

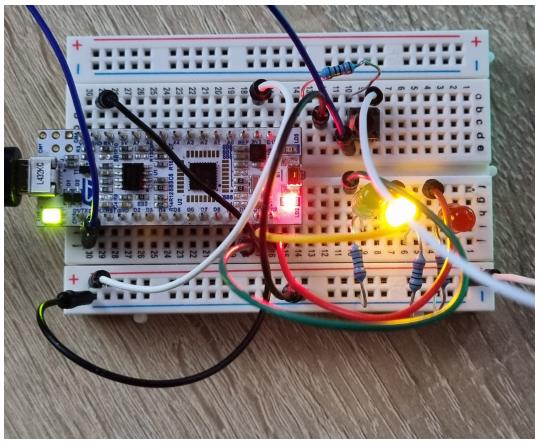


Abbildung 10: Ampel mit Taster[10].

```
    state = false;                // variable to determine debounce
}
```

Darauffolgend wird in der Timer Funktion nach 50ms erneut geprüft, ob der Taster ein Signal sendet, sodass ein Debounce vernachlässigt wird und setzt die Variable "count" dann je nach dem vorherigen Zustand auf 1 oder 2.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){ // timer function 50 ms
    if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11) == GPIO_PIN_RESET){ //check if debounce
        if(count < 2){
            count++;
        }
        else
            count = 1;
    }
state = true;
HAL_TIM_Base_Stop_IT(&htim1);
}
```

6 Fazit

Im 1. Labor des Moduls Rechnerorganisation werden die Grundlagen der embed Welt gelernt. Dazu wird mit Blinky die Grundlage der Nutzung des STM32L432KC gesetzt. Außerdem wird elektrotechnisches Wissen gefordert, welches aus dem 1. und 2. Semester des Studiengangs Computer Engineering erworben wurde. Durch den Aufgabenaufbau vom Blinky bis hin zum Trafficlight_blinky wird von Schritt zu Schritt die letzte Aufgabe Trafficlight_blinky gezielt erarbeitet, wobei die Schwierigkeitsstufe sich von Aufgabe zu Aufgabe natürlich erhöht. Es wurde in diesem Beleg gelernt:

- Programmierung vom STM32L432KC Board
- LED Ansteuerung durch GPIO
- Taster Ansteuerung mit GPIO (wobei mit einem Interrupt gearbeitet wurde, welches nicht in diesem Modul erfordert war)
- Allgemeines C programmieren

Abbildungsverzeichnis

1 Einfache LED Schaltung 5

2 Leuchtende LED 5

3 LED Schaltung mit Taster 5

4 LED mit Taster 5

5 LED Schaltung mit Nutzung von PB4 7

6 leuchtende LED mit PB4 7

7 Ampelschaltung 9

8 Ampel mit Taster 9

9 Ampelschaltung mit Taster 11

10 Ampel mit Taster 11

Literatur

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