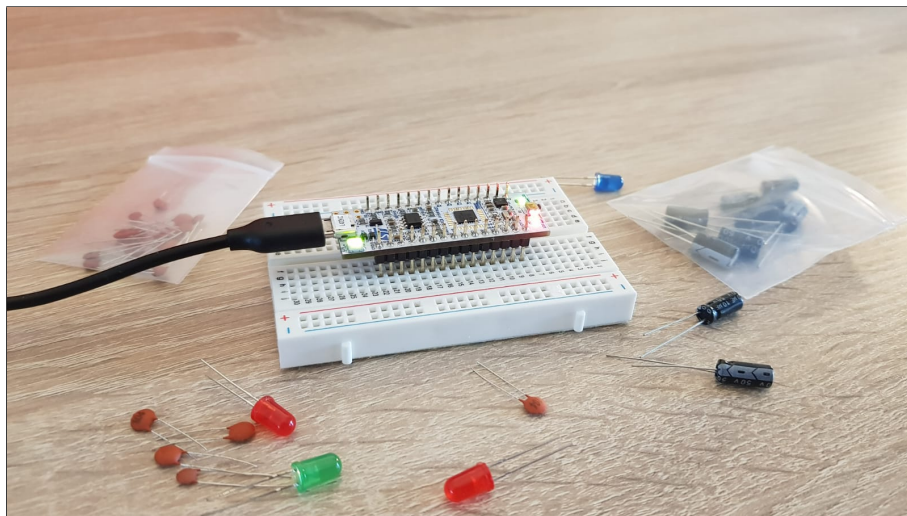


Rechnerorganisation Beleg 4



STM43L432KC

Hai Nam La [s0578105]

Dozent: Prof. Dr. Sebastian Bauer

08.Januar 2023

Inhaltsverzeichnis

| | | |
|-----------|------------------------------|-----------|
| 1 | Vorwort | 3 |
| 2 | Abstract | 3 |
| 3 | STM32I432KC Register | 4 |
| 4 | Makefile | 6 |
| 5 | Codestruktur | 6 |
| 6 | Therealhellworld | 7 |
| 6.1 | Vorbetrachtung | 7 |
| 6.2 | Implementation | 7 |
| 7 | Blinkyprime | 10 |
| 7.1 | Vorbetrachtung | 10 |
| 7.2 | Implementation | 10 |
| 8 | Stopwatch | 11 |
| 8.1 | Vorbetrachtung | 11 |
| 8.2 | Implementation | 11 |
| 9 | Stopwatch Display | 14 |
| 9.1 | Vorbetrachtung | 14 |
| 9.2 | Implementation | 15 |
| 10 | Fazit | 16 |
| | Abbildungsverzeichnis | 18 |
| | Literatur | 19 |

1 Vorwort

In diesem Beleg wird im bare metal style programmiert. Dabei wird in Assembly mit Nutzung von Registern gearbeitet, wobei die Registerfunktionen des Boards sowie alle Register aus dem Reference Manual(RM) oder Programming Manual(PM) des STM32L432KCs entnommen werden. Alle Aufgaben wurden mithilfe von VSCode als IDE geschrieben und die STM32CubeIDE einige Male zur Überprüfung der Register zu Rate genommen. Die Software wird dann auf dem STM32L432KC Nucleo Board getestet indem eine ".bin" erstellte Datei, durch make, auf das STM32 Board geflasht wird. Außerdem ist dieser Beleg eine Weiterführung vom 3. Beleg, weshalb nicht spezifisch auf alle Elemente eingegangen wird sondern davon ausgegangen wird, dass der 3. Beleg ebenfalls durchgegangen wurde. 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

Es wird im 4. Beleg von Rechnerorganisation wie schon erwähnt in bare metal programmiert. Dabei werden 4 Aufgaben bewältigt die wie in den anderen Belegen auch aufeinander aufbauen.

1. Therealhelloworld die Zeichenkette "Hello World" über USART auszugeben
2. Blinkyprime ein blinky mit systick interrupt
3. Stopwatch eine stopuhr zu entwickeln

4. Stopwatch Display die Stopuhr in Aufgabe 3 mithilfe von einer 7 Segment Anzeige anzeigen zu lassen

3 STM32L432KC Register

Es werden alle wichtigen Register aufgezeichnet, die dann genutzt werden. Zur Verständlichkeit der Implementierungen ist es ratsam das Reference Manual(RM) und Programming Manual(PM) des STM32L432KC ebenfalls offen zu haben, um die Register Nutzung nachzuvollziehen.

| Name | Art | Offset | Adresse | Wert | Quelle und Notiz |
|------------------|------------------|--------|-------------|-------|--------------------|
| SRAM_BASE | BasisAdresse | - | 0x2000 0000 | - | PM p.32 / RM p. 67 |
| SRAM_END | Endadresse | - | 0x200F FFFF | - | PM p.32 |
| FLASH_BASE | BasisAdresse | - | 0x0800 0000 | - | RM p.67 |
| RCC_BASE | BasisAdresse | - | 0x4002 1000 | - | RM p. 68 |
| RCC_AHB2ENR | RegisterAdressen | 0x4C | 0x4002 104C | - | RM p. 218 |
| RCC_AHB2ENR_PB | Wert | - | - | (1«1) | RM p. 218 |
| RCC_APB1ENR1 | RegisterAdresse | 0x58 | 0x4002 1058 | - | RM p. 220 |
| GPIOB_BASE | BasisAdresse | - | 0x4800 0000 | - | RM p. 68 |
| GPIOB_MODER | RegisterAdressen | 0x00 | 0x4800 0400 | - | RM p. 267 |
| GPIO_MODER_PINB3 | Wert | - | - | (1«6) | RM p. 267 |
| GPIOB_ODR | RegisterAdressen | 0x14 | 0x4800 0414 | - | RM p. 269 |
| GPIO_PINB3_HIGH | Wert | - | - | (1«3) | RM p. 269 |
| GPIOB_IDR | RegisterAdressen | 0x10 | 0x4800 0410 | - | RM p. 269 |
| GPIOA_BASE | BasisAdresse | - | 0x4800 0000 | - | RM p. 68 |
| GPIOA_MODER | RegisterAdressen | 0x00 | 0x4800 0000 | - | RM p. 267 |
| USART2_BASE | BasisAdresse | - | 0x4000 4400 | - | RM p. 70 |
| USART2_CR1 | RegisterAdresse | 0x00 | 0x4000 4400 | - | RM p. 1238 |
| USART2_BRR | RegisterAdresse | 0x0C | 0x4000 440C | - | RM p.1249 |
| SYSTICK_BASE | BasisAdresse | - | 0xE000 E010 | - | PM p. 246 |
| SYSTICK_CTRL | RegisterAdresse | 0x00 | 0xE000 E010 | - | PM p. 247 |
| SYSTICK_LOAD | RegisterAdresse | 0x04 | 0xE000 E014 | - | PM p. 248 |
| SYSTICK_VAL | RegisterAdresse | 0x08 | 0xE000 E018 | - | PM p. 249 |

4 Makefile

Es werden alle nötigen Dateien mit arm-none-eabi-gcc kompiliert mit den Optionen -mcpu-cortex=m4, wegen unserer STM32L432KC Chiparchitektur ARM Cortex-M4, -mthumb, um thumb Instruktionen zu generieren, -c, um die kompilierten Dateien noch nicht zu linken, und -o um es in der nachfolgend definierten .o Datei zu schreiben. Daraufgehend wird eine .elf Datei kompiliert, die die Sprungadresse bei 0x0800 0000 setzt(-Wl,-Ttext=0x800 0000), da ab da der Flash des STM32L432KC beginnt. Schlussendlich wird eine .bin erstellt, die dann auf dem STM32 geflashed werden kann.

5 Codestruktur

Wie schon erwähnt wird in Assembly geschrieben. Dabei wird eine head.S Datei genutzt um die Einsprungadresse des STM32 Boards sowie den __start der Software zu initialisieren. Die __start Funktion selbst wird aber in einer .c Datei definiert. Es wird also im allgemeinen im C Syntax programmiert, mit Nutzung der Registerfunktionen des STM32L432KC. Dabei wird eine weitere .c Datei genutzt mit dazugehörigem header.h. Eine graphische Darstellung der Codestruktur ist in Abbildung 1 zu finden.

Bei den ersten beiden Aufgaben Therealhelloworld und Blinkyprime wird die genutzte .c und .h Datei usart genannt, da nur die USART Nutzung definiert wird. Ab Stopwatch werden beide Dateien utility.c und .h genannt.

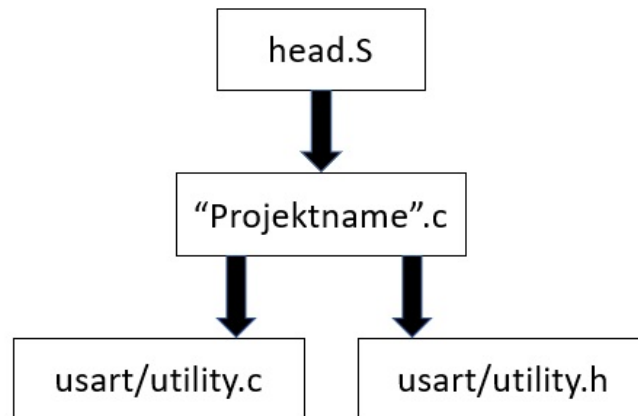


Abbildung 1: Codestruktur¹.

6 Therealhellworld

6.1 Vorbetrachtung

Kennengelernt haben wir das Helloworld ausgeben im Modul CEIntro des 1. Semesters. Dabei haben wir das USART2 benutzt und die `HAL_UART_Transmit()` um einen String auszugeben. Aus der Erkenntnis wollen wir erneut das USART2 nutzen um unser Helloworld in bare metal auszugeben. Dazu werden beide USART Pins benötigt, den `USART_TX(PA2)`, der Transmitter und den `USART_RX(PA3)`, der Receiver. Beides sind Pins des Ports A, weshalb der GPIO Port A zuallererst aktiviert werden muss.

6.2 Implementation

Es wird eine `usart.c` und die dazu gehörige `usart.h` Datei definiert. Ähnlich wie bei den Aufgaben aus Rechnerorganisation Beleg 3 müssen alle benötigten Register/Clocks aktiviert werden um das in der Vorbetrachtung genannte GPIO und somit USART nutzen zu können. Dazu wird zuallererst im RCC die GPIOA Ports aktiviert und an-

schließlich die USART2 Peripheral. Außerdem werden die Pins PA2 und PA3 im GPIO_MODER im "alternate function mode" aktiviert für die Nutzung als USART. Schlussendlich werden dann im USART_BASE Register die beiden genannten Pins als USART Pin aktiviert.

```

*(uint32_t*)(RCC_BASE + 0x4C) |= (1 << 0); // set RCC GPIOAEN in AHB2ENR
*(uint32_t*)(RCC_BASE + 0x58) |= (1 << 17); // set RCC USART2EN in APB1ENR1

*(uint32_t*)(GPIOA_BASE) = 0xabffffaf; //set PA3 and PA2 moder somehow its not
possible to set it per shifting but only like this (2 << 4) + (2 << 6)

```

Auch wenn jetzt die USART Pins "aktiviert" wurden, müssen noch kleine Änderungen vorgenommen werden, sodass die USART Ausgabe auch wirklich gesendet und empfangen werden kann. Dazu muss die oben im GPIO_MODER aktivierte "alternate function" im GPIO alternate function low register ebenfalls gesetzt werden auf 7 (7<<8) (7<<12). Dann werden USART, Transmitter und Receiver, wirklich aktiviert in dem die ersten 3. bits, des USART control register (USART_CR1), jeweils auf 1 gesetzt werden (1<<0)+(1<<1)+(1<<2). Anschließend wird die Baudrate gesetzt. Die soll 115200 Bd(baud per second) sein und ist zusammengesetzt aus Mantissa und Fraction, welche sich folgend berechnen lassen:

$$USARTDIVISION = \frac{Clockspeed}{baudrate * 16}$$

$$Mantissa = (int)(USARTDIVISION)$$

$$Fraction = (USARTDIVISION)fraction * 16$$

und für unser Projekt gilt folgendes:

$$2.17 = \frac{4Mhz}{115200 * 16}$$

$$2 = (int)2.17$$

$$3 = 0.17 * 16$$

```

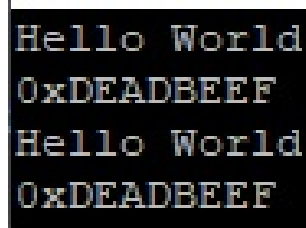
*(uint32_t*)(GPIOA_BASE + 0x20) |= (7 << 8); //Set PA2 alternnet function
*(uint32_t*)(GPIOA_BASE + 0x20) |= (7 << 12); //set PA3 alterate function

*(uint32_t*)(USART2_BASE + 0x00) |= 0x00; //Set all bits = 0
*(uint32_t*)(USART2_BASE + 0x00) |= (1 << 0); //set UE=1
*(uint32_t*)(USART2_BASE + 0x0C) |= (3 << 0); //set baudrate fraction
*(uint32_t*)(USART2_BASE + 0x0C) |= (2 << 4); //set baudrate mantissa

*(uint32_t*)(USART2_BASE + 0x00) |= (1 << 2); //enable receiver
*(uint32_t*)(USART2_BASE + 0x00) |= (1 << 3); //enable transmitter

```

Damit haben wir die Vorlage geschaffen um mit dem USART Zeichenketten zu senden. Es werden 3 Funktionen `usart_putc()` , um einen Character zu senden, `usart_puts()` , um einen String zu senden und `usart_putx()`, um eine hexadezimale Zahl als String zu senden, definiert. Anschließend wird die `therealhelloworld.c` definiert, die die Funktionen in einer `while(1)` Schleife laufen lassen. Als Ausgabe wird Hello World und in der nächsten Zeile `0xDEADBEEF` erwartet und in Abbildung 2 wurde die Ausgabe auch erfolgreich ausgegeben.



```

Hello World
0xDEADBEEF
Hello World
0xDEADBEEF

```

Abbildung 2: USART Ausgabe im Terminal².

7 Blinkyprime

7.1 Vorbetrachtung

Es soll ein Blinkyprogramm geschrieben werden, welches den SysTick Interrupt nutzt. Dafür werden erneut erstmal der zu Nutzende GPIO Pin aktiviert sowie der SysTick, es soll auch die die Ausgabe aus Therealhellworld, immer wenn ein Interrupt geschieht, gesendet werden. Dafür wird in der head.S der systick vector initialisiert. Als Gpio wird der GPIO Pin PB3 genutzt mit der aus Werk aus eingelöteten LED.

7.2 Implementation

Es wird der Code aus Therealhellworld übernommen und der Vector in der head.S Datei um den SysTick erweitert. Um dann eine LED leuchten zu lassen wird wie in den vielen Schritten vorher der GPIO Port aktiviert werden müssen und dann der MODER.

```
*(uint32_t*)(RCC_BASE + 0x4C) |= (1 << 1); // set RCC GPIOBEN
*(uint32_t*)(GPIOB_BASE + 0x00) = 0xfffffe7f; // set GPIOB_PIN_3in MODER as output
// somehow (1<<2) doesnt work
```

Jetzt muss der SysTick aktiviert werden(1 « 0), sowie der SysTick Interrupt(1 « 1) und die Clocksource auf den ProcessorClock gesetzt werden(1 « 2). Alles geschieht durch anpassen des SysTick control and status register(STK_CTRL).

```
*(uint32_t*)(SYSTICK + 0x00) |= (1 << 0); // enable counter
*(uint32_t*)(SYSTICK + 0x00) |= (1 << 1); // enable systick interrupt request
*(uint32_t*)(SYSTICK + 0x00) |= (1 << 2); // set clocksource to processorclock (4 Mhz)
```

Der SysTick reload value register kann dann die Interrupt Zeit verändern in dem ein Wert dem Register zugewiesen wird, zum Beispiel 4 000 000 = 1 Sekunde(da 4Mhz

der Prozessortakt ist). Wird der SysTick current value register geändert, unabhängig von der Zahl, wird der derzeitige SysTick counter auf 0 zurückgesetzt, womit der Timer neugestartet werden kann. Es wird eine Funktion `toggle_pin()` definiert, die im xor den GPIO Pin PB3 ein und anschaltet.

```
*(uint32_t*)(GPIOB_BASE + 0x14) ^= (1<<3); // toggle GPIOB_PIN3 depending on on off  
state (xor)
```

Diese Funktion wird dann in der `systick()` Funktion aufgerufen, welche in der `head.S` initialisiert wurde. Werden dann die `usart` Funktionen ebenfalls in die `Systick` Funktion geschrieben, so ist die Aufgabe gelöst worden.

8 Stopwatch

8.1 Vorbetrachtung

Es soll eine Stoppuhr implementiert werden. Dabei soll die LED leuchten wenn die Stoppuhr gestartet wird und ausgehen wenn der Timer gestoppt wird, sowie im Terminal eine Nachricht für beide Zustände senden. Genutzt wird der Code in Aufgabe 3 Blinkyprime, wobei ab jetzt die `usart.c` und `usart.h` in `utility.c` und `utility.h` genannt wird. Zum Starten und Stoppen wird ein Taster genutzt, der über GPIO Pin PA1 angesteuert wird. Vorweg wird erwähnt dass, die Codestruktur sehr unaufgeräumt ist, da anstatt neue Funktionen zu implementieren, die initialisierung der GPIOs und weitere in den Funktionen vom USART und der LED reingeschrieben wurde.

8.2 Implementation

Es wird die Schaltung in Abbildung 3 gebaut und in Abbildung 4 veranschaulicht, wobei hier schon eine 7 Segmentanzeige verbaut ist für die nächste Aufgabe.

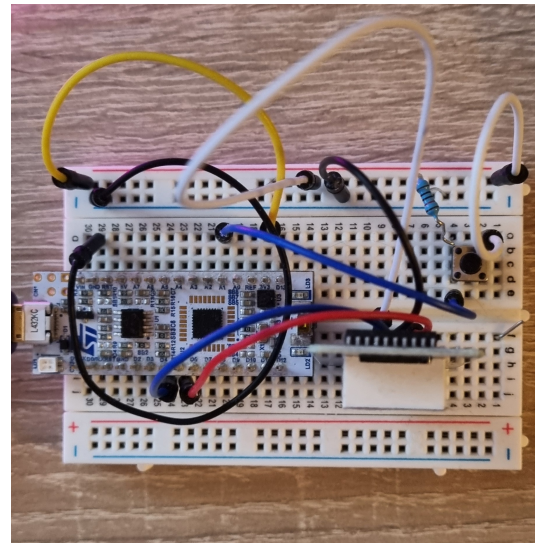
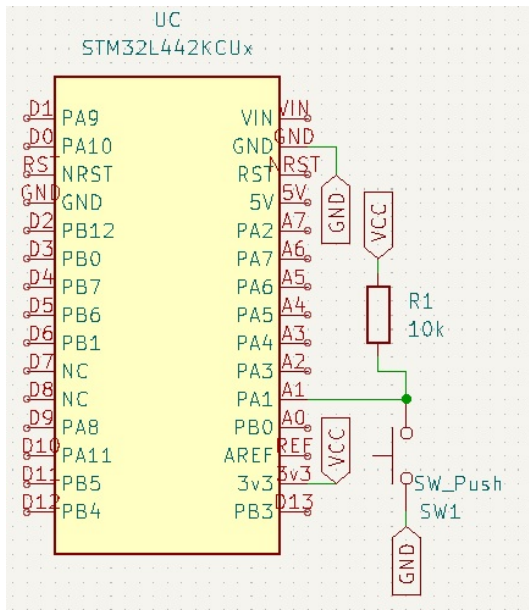


Abbildung 3: Schaltung für die Stoppuhr nur mit Taster³.

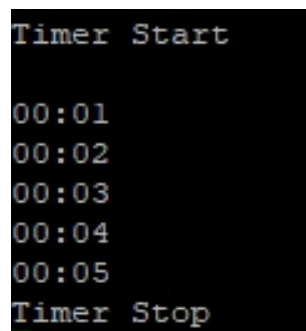
Abbildung 4: 7 Segmentanzeige mit Taster⁴.

Es werden wie auch vorher der GPIO Pin PA1 im GPIOA_Moder(1<<1) initialisiert, wobei der Port ausgelassen werden kann, weil für den USART der Port schon aktiv ist. Die Funktion `read_gpio()` wird als Funktion definiert, in der alle folgenden Anweisungen aufgerufen werden. Es wird die Taster Funktionsstruktur aus den vorherigen Rechnerorganisation Belegen übernommen. Dazu wird der Taster, beziehungsweise der GPIO_IDR des Taster Pins, abgefragt und eine Debounce Zeit abgewartet, bevor erneut abgefragt wird, ob der Taster gedrückt wird. Dabei ist der debounce auf 15 ms gewählt worden.

```
if((*(uint32_t*)(0x48000010) & (1<<1))) { // first button press
    *(uint32_t*)(SYSTICK + 0x04) = 4000000; //set reload value 4Mhz
    *(uint32_t*)(SYSTICK + 0x08) = 0; // set current systick value as 0
    for(int button_time = *(uint32_t*)(SYSTICK + 0x08); button_time >= 3940000;
        button_time = *(uint32_t*)(SYSTICK + 0x08)) {} // 15 ms delay
    if(!(*(uint32_t*)(0x48000010) & (1<<1))) { // debounce second press release
        /*stopwatch functions*/
    }
}
```

}

Es folgt eine `switch case` die 2 Zustände hat: `count = 0` und `count = 1`. Dabei startet der Timer bei `count = 0` und setzt die Variable `start_end = 1`, stoppt bei `count = 1` und setzt die Variable `start_end = 0`. Es wird die gezählte Zeit des Timers auf 10ms gewählt, da mit 1ms Probleme entstanden und die Zahl sowieso zu schnell durchgezählt hätte. Die `read_gpio()` Funktion wird dann in der `_start()`, in einer `while(1)` Schleife, ausgeführt, sodass immer geprüft wird ob der Taster gedrückt wird. In der `ystick()` Funktion, also die Funktion, die dann durch den Interrupt alle 10ms ausgeführt wird, wird eine neue `switch case` ausgeführt, die durch die Variable, die in der Taster `switch case` auf 1(`start_end`) gesetzt wird die Stoppuhr beginnt. 60 Sekunden wird außerdem als maximale Zeit eingestellt. In Abbildung 5 wird die Ausgabe im Terminal angezeigt. Dabei wird nach dem Timer Start eine Bestätigung des Startens gesendet und darauffolgend die Zeit, startend von 1ms, gezählt. Nach dem erneuten Drücken des Tasters wird der Timer gestoppt, ebenfalls mit einer Ausgabe des Stoppens.



```
Timer Start
00:01
00:02
00:03
00:04
00:05
Timer Stop
```

Abbildung 5: Stoppuhr Ausgabe im Terminal⁵.

```

uint8_t tm1637_write_byte(tm1637_t *tm, uint8_t data)
{
    /* write 8 bit data, bit by bit, least significant bit first */
    for (uint8_t i = 0; i < 8; i++)
    {
        tm->write_clk(tm, 0);
        tm->delay_us(tm, _TM1637_BIT_DELAY);
        tm->write_dio(tm, data & 0x01);
        tm->delay_us(tm, _TM1637_BIT_DELAY);
        tm->write_clk(tm, 1);
        tm->delay_us(tm, _TM1637_BIT_DELAY);
    }
}

```

Abbildung 6: Write_byte vorher⁶.

```

uint8_t tm1637_write_byte(tm1637_t *tm, uint8_t data)
{
    /* write 8 bit data, bit by bit, least significant bit first */
    for (uint8_t i = 0; i < 8; i++)
    {
        tm->write_clk(tm, 0);
        tm->delay_us(tm, _TM1637_BIT_DELAY);
        tm->write_dio(tm, data & (1<<i));
        tm->delay_us(tm, _TM1637_BIT_DELAY);
        tm->write_clk(tm, 1);
        tm->delay_us(tm, _TM1637_BIT_DELAY);
    }
}

```

Abbildung 7: Write_byte nachher⁷.

9 Stopwatch Display

9.1 Vorbetrachtung

Die Stoppuhr aus Aufgabe 3 Stopwatch soll mit einer 7 Segment Anzeige erweitert werden. Die ganzen Funktionen, beziehungsweise einen ganzen Treiber, für die 7 Segment Anzeige selbst zu schreiben, wäre in diesem Semester zeitbedingt nicht mehr möglich, weshalb vom Dozenten der Treiber zur Verfügung gestellt wurde(tm1637.c und tm1637.h). Dabei werden auch die genutzten Pins der 7 Segmentanzeige festgesetzt:

- CLK = PB6
- DIO = PB7
- VCC = 3V3
- GND = GND

Ein Haken gibt es, dass eine Funktion noch nicht richtig funktioniert. Es ist die `tm1637_write_byte()` Funktion, die einfach gefixt wurde, indem das geschiftete Byte mit dem Integer geändert wird, wie in Abbildung 6 und 7 zu sehen ist.

Eine weitere Voraussetzung war, dass der eigene Name auf dem Display angezeigt werden soll. Ich wählte meinen Rufnamen "NAM".

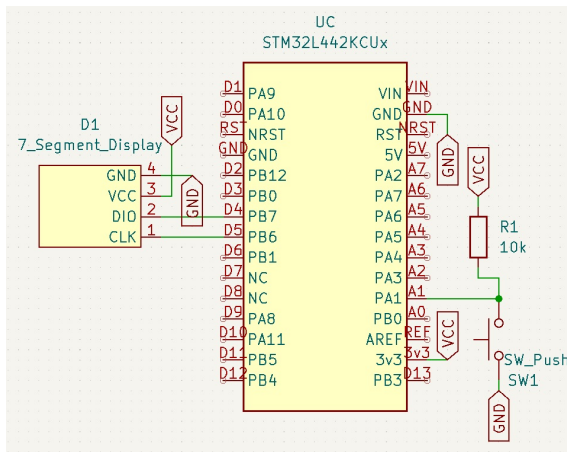


Abbildung 8: Schaltung mit 7 Segmentanzeige und Taster⁸.

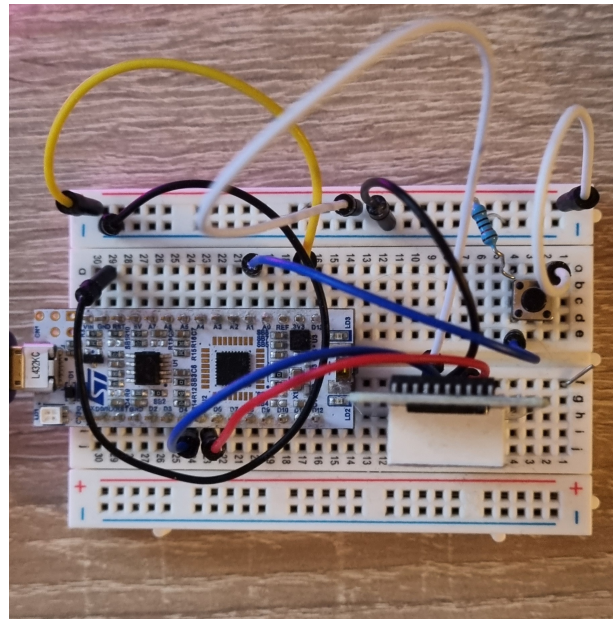


Abbildung 9: 7 Segmentanzeige mit Taster⁹.

9.2 Implementation

Es wird der Code aus Aufgabe 3 Stopwatch übernommen und die tm13676.c und .h hinzugefügt. Die Schaltung wurde ebenfalls mit der 7 Segmentanzeige erweitert (Abbildung 8). Die realisierte Schaltung in Abbildung 9 ist die Selbe wie in Abbildung 4. Außerdem wurde die Funktion `sysTick_delay_us()` übernommen, die einen besseren delay darstellt als der, der für den Taster mit einer for Schleife für den Taster implementiert wurde. Viel ändert sich nicht am Code bis auf, dass die Funktion `tm1367_write_uint()` aufgerufen wird in der `sysTick()` Funktion, die die Zahlen auf dem Display wiedergeben.

```
if(seconds < 1){
    time = milliseconds*10;
    tm1637_write_uint(&disp,time, 2);
}
else if(seconds < 10){
```



```

        time = seconds*100+milliseconds*10;
        tm1637_write_uint(&disp,time, 1);
    }else{
        time = seconds*100+milliseconds*10;
        tm1637_write_uint(&disp,time, 0);
    }

```

Dazu wird in einer if Anweisung festgesetzt wie die Zahl auf dem Display dargestellt wird, sodass es wie eine richtige Stoppuhr aussieht. Um den Namen darzustellen, wird der Name in der `_start()` Funktion einmalig angezeigt und dann wieder gelöscht sobald der Taster gedrückt wird.

```

    tm1637_init(&disp, tm_delay_us, tm_write_clk, tm_read_dio, tm_write_dio, NULL);
    /*print out nAM to 7 segment display*/
    static const uint8_t seg_name[] = {0x54, 0x77, 0x37};
    tm1637_write_segment(&disp, seg_name, 4, 1);

```

Die Demo der Stoppuhr mit 7 Segmentanzeige befindet sich auf der HTW cloud.

10 Fazit

Im 4. und letzten Labor des Moduls Rechnerorganisation werden die in Labor 3 gewonnenen Fähigkeiten erweitert. Dazu wird weiter in bare metal programmiert und weitere Register in Betrachtung gezogen. Es wurde dabei die Nutzung des USARTs implementiert und mit dem SysTick, sowie dem SysTick Interrupt, gearbeitet. Dazu gab es noch die Nutzung einer 7 Segmentanzeige, wobei leider die Nutzung nicht selber erschlossen wurde aufgrund der kurzen verfügbaren Zeit. Wie in den vorherigen Laboren bauen die Aufgaben aufeinander auf und es wurde letztendlich eine Stoppuhr entwickelt, die auf einer 7 Segmentanzeige die Zeit ausgibt. Es wurde in diesem Be-

leg gelernt:

- mehr Assembly programmieren
- Ansteuerung von Registern für das STM32L432KC Board
 - USART
 - SYSTICK
- Ausgabe von Strings und Charaktere über Register
- Nutzung einer 7 Segmentanzeige
- Nutzung eines SysTick Interrupts, sowie deren Zeitsetzung

Abbildungsverzeichnis

| | | |
|---|---|----|
| 1 | Codestruktur | 7 |
| 2 | USART Ausgabe im Terminal | 9 |
| 3 | Schaltung für die Stoppuhr nur mit Taster | 12 |
| 4 | 7 Segmentanzeige mit Taster | 12 |
| 5 | Stoppuhr Ausgabe im Terminal | 13 |
| 6 | Write_byte vorher | 14 |
| 7 | Write_byte nachher | 14 |
| 8 | Schaltung mit 7 Segmentanzeige und Taster | 15 |
| 9 | 7 Segmentanzeige mit Taster | 15 |

Literatur

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

[2] **STM32Microelectronics**

https://www.st.com/resource/en/reference_manual/

rm0394-stm32l41xxx42xxx43xxx44xxx45xxx46xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf, 01.2023

[3] **STM32Microelectronics**

https://www.st.com/resource/en/programming_manual/

pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf, 01.2023