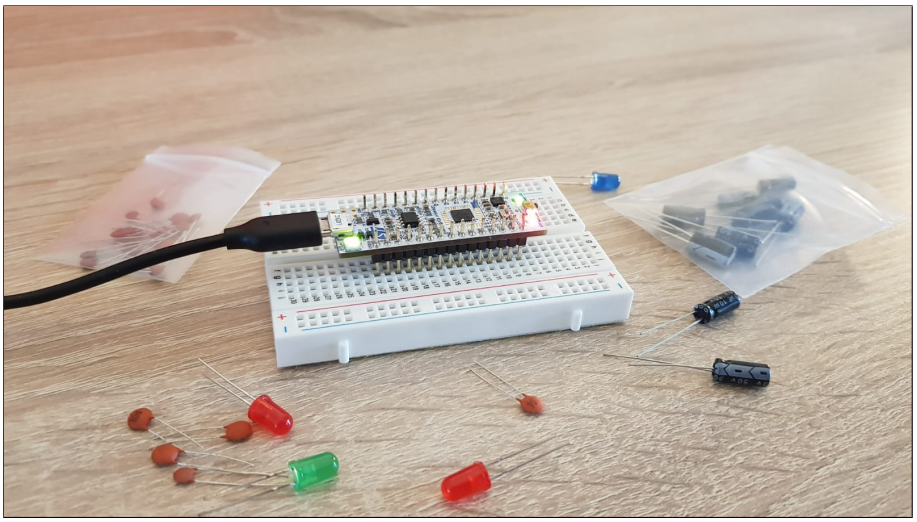
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Computer Engineering

Rechnerorganisation Beleg 3



STM43L432KC

Hai Nam La [s0578105]

Dozent: Prof. Dr. Sebastian Bauer

08.Januar 2023

Inhaltsverzeichnis

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 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

Es wird im 3. 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. Blinky Assembler, das blinky aus Beleg 1 in assembler darzustellen
2. SOS Blinky die Zeichenkette SOS durch Morsecode auf der LED darzustellen
3. SOS Blinky 2 die Aufgabe 2 mit einem Taster zu erweitern
4. Secret Bin es wird eine .bin Datei disassembliert

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
GPIOB_BASE	BasisAdresse	-	0x4800 0400	-	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

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. Darauffolgend 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 Blinky Assembler

5.1 Implementierung

Es wird wie im ersten Beleg das Helloworld der embed Welt programmiert, diesmal nur in Assembly. Dafür wird das RM sowie PM des STM32L432KC zu Hilfe genommen und losgelegt. Um embed Assembly zu benutzen wird immer zuerst alle benötigten Clock Register angeschaltet um Beispielsweise eine GPIO zu benutzten. Für das Blinky wollen wir den GPIO Pin PB3 benutzen, welches eine aus Werk eingelötete LED anspricht. Dafür wird zuallererst vom reset and clock control(RCC) Register der peripheral clock enable register(AHB2ENR) am 1. Bit auf 1 gesetzt (1«1). Dadurch werden die GPIOB Pins alle aktiviert.

```
/* Get contents of AHB2 bus clock enable register */
ldr     r0, =RCC_AHB2ENR // 0x4002104c
ldr     r1, [r0]

/* Set bit to enable clock for GPIO port B */
ldr     r2, =RCC_AHB2ENR_PB // (1<<1)
orrs    r1, r2
str     r1, [r0]
```

Darauffolgend wird im GPIO port mode register(GPIO_MODER) der 6. Bit auf 1 gesetzt um den 3. Pin(PB3) auf output zu setzen.

```
/* Configure GPIO pin as output */
```

```
    ldr     r0, =GPIOB_MODER // 0x48000400
    ldr     r1, =GPIO_MODER_PIN3O // (1<<6)
    str     r1, [r0]
```

Jetzt ist wichtig, dass der GPIO port output data register (GPIO_ODR) genutzt wird um den Pin anzusprechen. Es wird der 3. Bit auf 1(1«3) gesetzt um den Pin PB3 anzuschalten und auf 0 um ihn auszuschalten (0«3). Natürlich muss noch ein Delay erstellt werden, da sonst der blinky Effekt nicht zu erkennen ist. und somit kann dann eine einfache loop ".blinky" erschaffen werden, die dauerhaft den Pin PB3 ein und ausschalten kann.

```
    /* Load address of GPIOB output data register in r0 */
    ldr     r0, =GPIOB_ODR
    /* Load bitmask of gpio pin3 in r1 */
    ldr     r1, =GPIO_PIN3_HIGH // (1<<3)
  .blink:
    /* Store current r1 into GPIOB output data register */
    str     r1, [r0]

    /* Busy wait */
    ldr     r2, =DELAY // 0x80000
  .loop:
    subs    r2, r2, #1
    bne     .loop

    /* Toggle bit corresponding to bit 3 */
    eors    r1, #GPIO_PIN3_HIGH //exclusivor (1<<3) or (0<<3)

    /* Repeat endlessly */
    b       .blink
```

6 SOS Blinky

6.1 Implementierung

Es wird das in Blinky Assembler gewonnene Wissen genutzt um jetzt den Morsecode von SOS auszugeben. Wie in Blinky werden alle Register angeschaltet die genutzt werden:

- RCC_AHB2ENR auf Pin Port B (1«1)
- GPIO_MODER auf Pin 3 (1«6)
- GPIO_ODR auf Pin 3 (1«3)

Danach werden mehrere "Funktionen" implementiert, die die Buchstaben S O S in der Reihenfolge in Morsecode blinken lassen, wobei jeder Buchstabe erneut aus Funktionen besteht, die das Morsecode simulieren.

```
loop:
   bl playS
   bl playO
   bl playS
   bl wordEnd

   b loop
```

7 SOS Blinky 2

7.1 Implementierung

Für SOS Blinky 2 wird ein Taster implementiert der die Zeichenkette SOS in Morsecode einmalig abspielt, immer wenn es gedrückt wird. Außerdem wird eine externe

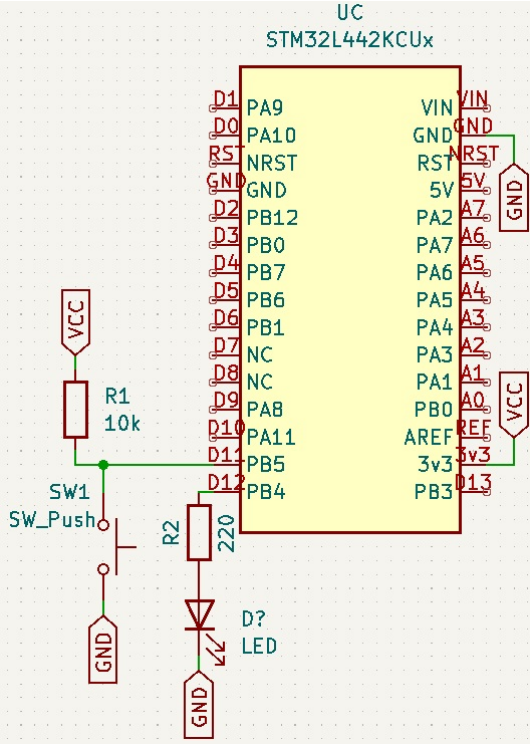


Abbildung 1: Einfache LED Schaltung mit Taster[1].

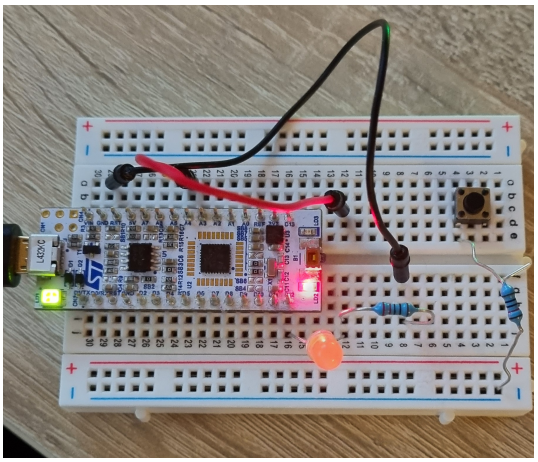


Abbildung 2: LED mit Taster aber nicht angeschlossen[2].

LED benutzt. Für die LED wird GPIO Pin PB4 benutzt und für den Taster GPIO Pin PB5. In der Abbildung 1 und 2 ist die Schaltung, sowie das Implementierte zu sehen. Leider fehlt ein Bild, bei der der Taster verbunden ist in der gebauten Schaltung. Die GPIOs werden dann folgendermaßen angeschalten:

- RCC_AHB2ENR auf Pin Port B (1«1)
- GPIO_MODER auf Pin 3 (1«8) +(1«11) / PB4 + PB5
- GPIO_ODR auf Pin 3 (1«3)

Dabei wird bei PB5 nicht wie bei den LED GPIOs auf "01" gesetzt sondern auf "10". Dadurch wird der Pin im alternate function mode gestartet, womit man einen Input auslesen kann, im GPIO port input data register (GPIO_IDR). Die in SOS Blinky pro-

grammierte Software wird dann übernommen bis auf der Initialisierung der Pins sowie einer kleinen Erweiterung in der "loop" Funktionen.

```
loop:
    ldr r0, = GPIOB_IDR // == 0x20 when push button on PB5 is pressed
   ldr r0, [r0]
   ldr r1, = 0x20 // could be (1<<5) but never tested
   cmp r0 , r1
   bne loop
    loop:
   bl playS
   bl playO
   bl playS
   bl wordEnd

   b loop
```

Bei der Erweiterung wird im polling style immer überprüft, ob der Taster gedrückt wird. Wird er gedrückt wird normal das SOS abgespielt wenn nicht, dann springt er wieder zum Anfang der loop Funktion.

8 Secret Bin

8.1 Vorgehen

Es ist eine .bin Datei gegeben ohne .elf und es ist die Aufgabe diese Datei zu disassemblieren und zu verstehen. Dafür wird mit beidem net.werwolv.ImHex und arm-none-eabi-objdump gearbeitet. Da wir schon wissen, dass der FLASH Registers des STM32L432KC bei 0x0800 0000 beginnt wird auch diese Adresse als start ausgewählt. Wenn der disassemblierte Code beachtet wird, wird festgestellt, dass ab dem

Offset 0 der Flash Adresse bis zum Offset 0x100 nicht viel passiert. Eine Sache ist aber, dass der SRAM initialisiert wird oder zumindest aufgerufen wird. Es kann angenommen werden, dass ab dem Offset 0x3C das Programm startet, da dort ein weiterer Aufruf des Flashes beginnt aber erst ab Offset 0x100 wirklich Code ausgeführt wird oder Register aufgerufen werden. Es soll nun versucht werden zu verstehen was im Code passiert. Da ist die Adresse 0xE000 E010 aufgefallen, was die Adresse des Systicks ist. Es wird vermutet, dass der Systick Timer genutzt wird als Delay um entweder ein Morsecode darzustellen oder einfach für ein bestimmtes Blinkmuster. Ab Offset 0x21E kommt es zu einer Langen Reihe von ähnlichen Werten, woraus ein loop vermutet wird. Wird jetzt an dem Offset 0x174 der Wert geändert zu etwas höheren und die .bin auf den Stm32 Board geflashed so ist ein langsameres blinken zu erkennen. Leider ist immernoch etwas komisch an dem Blinken, vorallem die Pausen dazwischen sind manchmal sehr schnell oder sehr langsam. Ich hab mich entschieden die oben genannte Loop zu interpretieren, ab Offset 0x21E. Dabei habe ich heraus gefunden, dass wenn auf den disassemblierten Code geachten wird, immer Funktionen oder zumindest zu einer Stelle gesprungen wird, die eine Sequenz ausführt. Dabei führt der Code ab dem Offset 0x1D0 ein kurzes Blinken aus, ab dem Offset 0x170 eine Pause, ab dem Offset 0x7C ein neues Wort und ab 0x1B4 ein langes Blinken. Dadurch wurde herausgefunden, dass das Secret Bin, das Wort Monkey Island in Morsecode verschlüsselt hat, welches als Antwort auf das Plfichtmusikbeispiel in Rechnerorganisation 2 ist.

9 Fazit

Im 3. Labor des Moduls Rechnerorganisation wird die Grundlage des bare metal programmieren für das STM32L432KC gewonnen. Dabei werden Assembly und die verschiedenen Datenblätter des Boards benutzt, um die Register setzen zu können. Wie

auch in den vorherigen Laboren bauen die Aufgaben aufeinander auf. Es wird wieder beim Anfang begonnen, wobei ein einfaches Blinky Programm aufgesetzt wird. Schlussendlich wird aus dem Blinky ein SOS Programm, welches das SOS in Morsecode per LED darstellt. Dazu gab es noch eine Aufgabe in der mithilf e von Deassemblieren eine .bin Datei erschlossen wird. Es wurde in diesem Beleg gelernt:

- Assembly programmieren
- Datenblätter analysieren und auswerten
- Ansteuerung von Registern für das STM32L432KC Board
  - RCC
  - GPIO
- deassemblieren und verstehen von .bin Dateien

Abbildungsverzeichnis

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-stmicroele
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