Abgabe 1 + 2 + 3 + 4 + 5: Einführung, GPIOs, LCD, ADC, DMA, TIMER, NVIC, PROJEKT 1 (Lüfterregelung), PROJEKT 2 ((Wetterstation)

Dimitry Ntofeu Nyatcha
Matrikelnummer: [redacted]

23.05.2025

Inhaltsverzeichnis

1 Entwicklungssystem (CMSIS, Cube)

1.1 1. Welche Grundarchitektur besitzt der Cortex-M4?

- ⋆ 32-Bit RISC Architektur
- ⋆ Max Takt: 180MHz
- ⋆ Flash-ROM : 2MByte (Fur Programmcode und statische Daten)
- ⋆ 256KByte(SDRAM) + 4KByte(SRAM) =¿ (Fur dynamische Daten)

1.2 2. Wozu dient der DMA-Controller?

Der Direct Memory Access (DMA) Controller ermöglicht es, Daten schnell zwischen Peripheriegeräten und Speicher zu übertragen, ohne dass die CPU direkt eingreifen muss. Dies entlastet den Prozessor und verbessert die Effizienz, besonders bei groën Datenmengen oder wiederholten Operationen.

1.3 3. Welche Vorteile bietet der Thumb-2 Befehlssatz?

- ⋆ kombiniert 16-Bit und 32-Bit Instruktionen, wodurch der Speicherverbrauch reduziert wird
- ⋆ reduziert die Grösse der Befehle auf etwas Kürzeres (von 32 auf 16 Bit) ohne die Rechenleistung stark zu beeinträchtigen.
- ⋆ enthält auch alle Thumb Befehle
- ⋆ unterstützt komplexere Befehle als der ursprüngliche Thumb-Befehlssatz, darunter auch Hardware-Operationen wie Multiplikation und Division.

1.4 4. Besitzt unser Controller eine FPU? Beschreiben Sie auch, wozu diese dient.

Ja, unser STM32F429 Mikrocontroller besitzt eine FPU (Floating Point Unit). Diese Einheit ist darauf spezialisiert, Gleitkommaberechnungen (z.B. Divisionen, Wurzeln) besonders effizient durchzuführen. Ohne FPU müsste die CPU diese Operationen softwarebasiert umsetzen, was deutlich langsamer ist.

Zudem bietet die FPU die Konvertierung zwischen Festkomma- und Fließkommadatenformaten sowie Anweisungen für Fließkomma-Konstanten."

1.5 5. Welche Vorteile bietet CMSIS für den Entwickler?

- ⋆ Hardware Unabhängigkeit: Entwickler können Code schreiben, ohne sich tief mit der spezifischen Hardware jedes Mikrocontrollers beschäftigen zu müssen
- ⋆ Wiederverwendbarkeit von Code: Module, Treiber und Bibliotheken lassen sich leicht zwischen Projekten teilen

⋆ Einheitliche API: Vereinfachte Entwicklung dank einheitlicher Funktionsaufrufe und Namenskonventionen.

1.6 6. Warum vereinfacht CMSIS die Portierung von Software auf unterschiedliche Mikrocontroller?

⋆ Der Quellcode muss beim Wechsel auf einen anderen Cortex-M-Mikrocontroller kaum oder gar nicht angepasst werden

⋆ Nur die hardwareabhängigen Teile (z.B. Treiber oder Konfigurationsdateien) werden ggf. angepasst, der restliche Code bleibt gleich

⋆ Die Portierung ist dadurch schneller, einfacher und weniger fehleranfällig

1.7 7. Beschreiben Sie das Cube Ökosystem, von dem die CubeHAL auch nur ein Bestandteil ist.

Das STM32Cube Ökosystem umfasst mehrere Komponenten:

- STM32CubeMX: Tool bzw User-interface zur grafischen Konfiguration der Peripherie und Projektgenerierung bzw Codegenerierung.
- STM32CubeIDE: Integrierte Entwicklungsumgebung basierend auf Eclipse.
- STM32Cube HAL: Hardware-Abstraktionsschicht mit Funktionen für einfache Nutzung der Peripherie.
- STM32Cube LL: Low-Level Bibliothek für Entwickler, die mehr Kontrolle brauchen.
- **STM32Cube Firmware Packages:** Gerätespezifische Pakete, die HAL, LL, CMSIS, Beispielprojekte und Board Support Packages enthalten.
- **Middleware:** Integrierbare Komponenten wie USB-Stacks, FATFS-Dateisystem, TCP/IP, oder RTOS (z. B. FreeRTOS) zur Unterstützung komplexerer Anwendungen.

Dieses System reduziert die Einstiegshürden und beschleunigt die Entwicklung durch Wiederverwendbarkeit und Klarheit.

Quelle: STM32Cube-Ecosystem – STMicroelectronics

2 Arbeitsweise mit der CubeIDE-Debugger-Funktionen

2.1 1. Setzen von Breakpoints (Haltepunkte) und Watchpoints (Beobachtungspunkten). Welche Funktion erfüllen Sie

1-) Breakpoint

- Ein Breakpoint ist eine Markierung im Code, bei der das Programm im Debug-Modus automatisch angehalten wird

- Sie ermöglichen es, den Programmzustand abzurufen und zu analysieren (z.B. Variablenwerte, Register) an einer bestimmten Stelle.

□

2-) Setzen eines Breakpoints

- Linksklick am linken Rand der Codezeile (grauer Bereich) → roter Punkt erscheint.

□

3-) Watchpoints (Beobachtungspunkte):

- Ein Watchpoint beobachtet den Speicherbereich einer Variable.
- Das Programm wird angehalten, sobald diese Variable verändert wird.

□

4-) Setzen eines Watchpoints:

- markieren Sie die Variable im Editor oder wählen Sie sie in der Gliederungsansicht aus.
- Click Run - Toggle Watchpoint.

□

2.2 Schrittweise Ausführung von Programmcode mit und ohne Überspringen von Funktionsaufrufen

Im Debug-Modus kann man das Programm Schritt für Schritt durchlaufen:

- Step Into (F5): Führt die aktuelle Zeile aus und springt in Funktionsaufrufe hinein.
- Step Over (F6): Führt die aktuelle Zeile aus, überspringt aber Funktionsaufrufe (sie werden im Hintergrund ausgeführt).
- Step Return (F7): Führt die aktuelle Funktion bis zum Ende aus und springt zurück zur aufrufenden Stelle.
- Resume (F8): Führt das Programm normal weiter bis zum nächsten Breakpoint.

□

2.3 Arbeiten mit Variablen und Ausgabe ihres Inhalts

Variables mit Datentypen deklarieren und auf LCD (als Parameter zb für LCD-Funktionen eingeben) anzeigen lassen oder ihren Inhalt Beim debugging mit hilfe von Breakpoints abrufen

2.4 Arbeiten mit Registern und Ausgabe ihres Inhalts

Analyse des Inhalts von Registern beim Debugging ..(Inhalt sichtbar auf die rechte seite also das rechte Fernster.)

3 LED$_o n$

1. Wie oben erläutert, ist die grüne LED an PG13 angeschlossen. An welchen Pin ist die rote LED des Discovery Boards angeschlossen?

- Die rote LED des Discovery Boards ist am Pin 14 des Ports G angeschlossen..

  siehe discovery-kit user-manual.pdf 6.5 : LED

4 GPIO: Digitale Ausgabe

1. Dokumentieren Sie in einer Tabelle die feste Zuweisung von GPIO-Pins zu 8-Segment-Leitungen, um im Folgenden nicht mehr die verschiedenen Schaltpläne abgleichen zu müssen.

- Siehe $DEFINE$ im Code

2. Bietet die STM32Cube HAL eine Funktion um nur mithilfe von Software eine Sekunde zu warten.

- Ja, die STM32Cube HAL bietet so eine Funktion und zwar void HAL Delay(uint32 t delay) und die sieht so aus:

```
* __weak void HAL_Delay(uint32_t Delay)
  {
    uint32_t tickstart = HAL_GetTick();
    uint32_t wait = Delay;

    /* Add a freq to guarantee minimum wait */
    if (wait < HAL_MAX_DELAY)
    {
      wait += (uint32_t)(uwTickFreq);
    }

    while((HAL_GetTick() - tickstart) < wait)
    {
    }
  }
```

5 GPIO: Digitale Eingabe

1. An welche GPIO-Pins ist der Joystick angeschlossen? Betrachten Sie dafür die Beschriftung des Joy sticks und der daneben liegenden Jumper!

- Siehe $DEFINE$ im code

2. Welcher GPIO-Mode muss für die Pins eingestellt werden, damit digitale Signale eingelesen werden können?

- Der GPIO muss auf Input eingestellt sein

3. Betrachten Sie den folgenden Ersatzschaltplan des Joysticks. Muss für die Pins ein Pull-Up- oder ein Pull-Down-Widerstand aktiviert werden?

- für die Pins muss ein Pull-Up-Widerstand aktiviert werden wegen des LOW-ACTIVE-EINGANG

4. Müssen Sie die Speed-Komponente der Initialisierungsstruktur füllen? Betrachten Sie hierzu im Reference Manual [4] Tabelle 36.

- Die Speedkomponente ist nicht wichtig beim Einlesen von Pins nur bei Auslesen ist das wichtig

5. Welche Funktion bietet die HAL, um Daten von einem GPIO-Port einzulesen?

- `GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin){}`

6. Fällt Ihnen bei dem Betätigen des Joysticks im Bezug auf Ihr Programmverhalten etwas ungewöhn liches auf? Hierbei handelt es sich um einen bekannten mechanischen Störeffekt von Tastern. Wie nennt sich dieser Störeffekt und welche Möglichkeiten existieren, um diesem entgegenzuwirken?

- Es kommt vor, dass die Taste bei einer einzigen Betätigung mehrmals ausgelöst wird
  - ⋆ Das Phänomen heisst : $DEBOUNCING$
  - ⋆ Als Lösung könnte man ein delay benutzen also: die funktion:

    `utils_delay_ms(t); mit t = 1000ms oder 500ms zb..`

6 LCD

1.Welche Funktionen gibt es, um Text auf dem Display auszugeben?

- ```
  void lcd_draw_text_at_line(const char* text, uint8_t line, uint16_t color, uint1
  {
   uint16_t y = line * (8*size)+10;

   ILI9341_Draw_Text(text, 10, y, color, size, background_color);
  }
  ```

  : Zeichnet einen Text in einer bestimmten Zeile und nimmt als Parameter:
  - * @param text Der zu zeichnende Text
  - * @ line Die Zeile auf dem Bildschirm
  ```

  - * @param color Die Textfarbe
  - * @param size Die Textgröße
  - * @param background color Die Hintergrundfarbe

- ```
  void lcd_draw_text_at_coord(const char* text, uint16_t x, uint16_t y, uint16_t c
  {
   ILI9341_Draw_Text(text, x, y, color, size, background_color);
  }
  ```

  : Zeichnet einen Text ab einer bestimmten Position mit Koordinate(x,y). und nimmt als Parameter:

  - * @param text Der zu zeichnende Text
  - * @param x Die x-Koordinate auf dem Bildschirm
  - * @param y Die y-Koordinate auf dem Bildschirm
  - * @param color Die Textfarbe
  - * @param size Die Textgröße
  - * @param background color Die Hintergrundfarbe

7 ADC

1-a. An welche GPIO-Pins des Mikrocontrollers sind die zwei Potentiometer des Analog Test Boards angeschlossen?

- die zwei Potentiometer des Analog Test Boards sind am PIN 6 bzw PIN 7 des GPIOA-PORTS..
  - MISO = ADC2 = PA6
  - MOSI = ADC1 = PA7

1-b. Welchen analogen Input-Channeln sind diese zwei Pins zugewiesen?

```
PA6 = ADC12_IN6
PA7 = ADC12_IN7

siehe STM32F429-Datasheet Table 10 Seite 56
```

```
2.a Wie lange dauert ein einzelnes Sample, wenn Sie einen ADC mit
 ADC_handle_structure.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV6;
 ADC_handle_structure.Init.Resolution = ADC_RESOLUTION_12B;
 ADC_channel_structure.SamplingTime = ADC_SAMPLETIME_3CYCLES;
 einstellen?
```

- Sampling Time = 3 cycles
- Fréquence ADC = 84 / 6 = 14MHz
```

- TconvCycles = (3 + 12) = 15 zyklus
- TconvZeit = (15 / 14MHZ) = 71,4ns
- ein einzelnes Sample entspricht 15 Zyklus , wobei ein Zyklus (1 / 14MHZ) dauert das heisst dann,dass ein Sample (15 / 14MHZ) = 71,4ns dauert

2-b Welche Sample-Rate ergibt sich? Beschreiben Sie, wie Sie auf diese Werte kommen und was diese bedeuten.

```
Sample_Rate = ( 1 / (71,4 * 10^(-9)) ) = 14005602,24 sample/s
```

4. Erläutern Sie, welche beiden weiteren Möglichkeiten neben dem Polling die HAL bietet, um das Ergebnis der Wandlung in Erfahrung zu bringen..

- DMA
- Interrupt.

5. In der Praktikumsanleitung wird erwähnt, dass bei der Einstellung eines GPIOs zum Einlesen eines analogen Signals kein Pull-Up- bzw. Pull-Down-Widerstand benötigt wird. Können Sie sich erklä ren, warum diese Widerstände nicht gebraucht werden?

- Im Analogmodus verhält sich der GPIO nicht wie ein digitaler Eingang. Hier Ziel ist es, kontinuierliche Werte zu messen (z. B. eine Spannung von 0 V bis 3,3 V, nicht nur 0 oder 1). Er muss keinen hohen oder niedrigen Pegel festlegen (wie bei der Digitaltechnik).

  Das analoge Signal muss frei schwanken können, ohne dass ein Widerstand nach oben oder unten „zieht“.

  Ein Pull-Up oder Pull-Down würde das Signal dazu zwingen, etwas höher oder tiefer zu gehen, als es eigentlich ist, was die echte analoge Messung verfälschen würde!

8 DMA

1. Sie haben in der Praktikumsanleitung einige Informationen präsentiert bekommen. Insbesondere die letzten Punkte, wie der ADC in Verbindung mit dem DMA genutzt werden kann, ist für die Pra xis relevant. Beschreiben Sie, wo diese Informationen außerhalb der Praktikumsanleitung in denen zur Verfügung gestellten Unterlagen zu finden sind. Dazu zählt auch die Information, in welcher Reihenfolge die Konfiguration stattfinden muss.

- Im Reference Manual des STM32F429 (Kapitel ADC und DMA) Im Kapitel ADC 13.8 Data management 13.8.1 Using the DMA kann man schon lehrreichende Info haben
- In der CubeHAL-Dokumentation (stm32f4xx_hal_adc.c, stm32f4xx_hal_dma.c).

Die korrekte Initialisierungssequenz lautet:

- zuerst DMA Konfigurieren

- Dann ADC konfigurieren und (ADC mit DMA verbinden)
- Schließlich das Ganze mit

  `HAL_ADC_Start_DMA starten.`

2. Betrachten Sie Tabellen 43 und 44 im Reference Manual [4] und benennen Sie für die folgenden Peri pheriekomponenten die dazugehörigen DMA, Stream und Channel

- a. Channel 3 von Timer 1: (DMA2 : Channel 6 : Stream 6)
- b. ADC3: (DMA2 : Channel 2 : Stream 0 und Stream 1)
- c. DAC2: (DMA1 : Channel 7 : Stream 6)

3. Welchen Vorteil erzielen Sie durch das Vorgehen bei Aufgabe 3?

- Die Vorteile dieser Technik sind eine schnelle Datenübertragung bei gleichzeitiger Entlastung des Prozessors

4. Wenn Sie den arithmetischen Mittelwert bilden, teilen Sie durch die Anzahl der Elemente. Versuchen Sie anstelle dessen nur durch 1/4 der Elemente zu teilen. Sie merken, dass die Auflösung größer wird. Was für Vor- und Nachteile gibt es hierbei?

- Vorteile:
  - Signal-Verstärkung ohne zusätzliche Hardware
    - nützlich bei schwachen Signalen.
  - Stärkere Unterscheidbarkeit in Visualisierung
    - für den Bargraph zb sieht man leichter, dass sich etwas bewegt.
- Nachteile:
  - Kein realer Spannungswert mehr
    - man kann die Werte nicht mehr direkt als mV oder Prozent interpretieren, sie sind künstlich hochskaliert.
  - Gefahr von Überlauf / Wertebereichüberschreitung
    - Wenn wir mit

      `.uint16_t`

      zb arbeiten, kann sum leicht über 65535 hinausgehen.

9 TIMER

`1. Initialisieren Sie das Zählwerk (TimeBase) von Timer 1. Stellen sie dabei den Prescaler so ein, dass der Timer mit einem Takt von 10 kHz zählt und nach einer Periodendauer von 1 Sekunde von vorne beginnt. Printen Sie den aktuellen Counter-Wert des Timers (verwenden Sie für den zugriff das HAL-Makro __HAL_TIM_GET_COUNTER(__HANDL`

```
 innerhalb der while(1)-Schleife auf den LCD und schätzen Sie
 , ob der Überlauf tatsächlich nach einer Sekunde stattfindet.
 Wechseln Sie nun von Timer 1 auf Timer 2, behalten den Rest
 der Konfiguration allerdings gleich. Fällt Ihnen bei
 dem Zeitverhalten bis zum Überlauf etwas auf? Woran liegt das?

-------------------------------------------------------------------
-------------------------------------------------------------------
 Einfach anpassen für Timer2:

 #include "stm32f4xx.h"
#include <stdio.h>
#include <lcd/lcd.h>

extern TIM_HandleTypeDef tim_handle_struct;

void timer1_init() {
 __HAL_RCC_TIM1_CLK_ENABLE();

 tim_handle_struct.Instance = TIM1;
 tim_handle_struct.Init.Prescaler = (SystemCoreClock / 10000) - 1; // 10 kHz
 tim_handle_struct.Init.Period = 10000 - 1; // overflow nach 1 s
 tim_handle_struct.Init.CounterMode = TIM_COUNTERMODE_UP;
 tim_handle_struct.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 tim_handle_struct.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

 HAL_TIM_Base_Init(&tim_handle_struct);
}

int main(void) {
 HAL_Init();
 lcd_init();

 timer1_init();
 HAL_TIM_Base_Start(&tim_handle_struct);

 char buffer[32];
 uint32_t time;

 while (1) {
  time = __HAL_TIM_GET_COUNTER(&tim_handle_struct);
  sprintf(buffer, "time: %lu", time);

  lcd_draw_text_at_line(buffer, 0, YELLOW, 2, BLACK);

 }
}
```

- Bei Timer 2 habe ich den Eindruck, dass der Überlauf nach 1s stattfindet im Vergleich zu Timer 1, der mir schneller erscheint und bei dem der Überlauf vor oder kurz nach 1s stattfindet (sorry kann nicht genau sagen denn ich habe nicht sehr gute Augen aber ist auf jeden Fall schneller)..
  - Timer 1 erscheint schneller, vermutlich weil er an den APB2-Bus angeschlossen ist, der in der Regel eine höhere Taktfrequenz als der APB1-Bus, wo der Timer 2 angeschlossen ist besitzt."nämlich (84MHz * 2 für Timer1 UND 42Mhz für Timer 2 da ihr Prescaller grösser 1 ist also 2 bzw 4).

2. Beantworten Sie folgende Fragen inklusive Rechenweg
   - Welchen Prescaler müssen sie bei einem Timer einstellen, um bei einem Systemtakt von 50MHz auf einen Zähltakt von 1 kHz zu kommen?

```
Timer_freq = Systemtakt / (Prescaller + 1)

Gesucht:  Timer_freq von 1khz = 1000hz = 1000 tics/s
          mit  Systemtakt = 50Mhz = 50.000.000hz

          Wir hätten dann 1000 = 50.000.000 / (Prescaller + 1)
          also prescaller + 1 = 50.000.000 / 1000 = 50.000

          Der Prescaller müsste dann = 50.000 - 1 = 49.999
```

   - Welche Periodendauer müssen sie einstellen, wenn der Zähler des Timers alle 2 Minuten (wieder) den Wert 0 erreichen soll?

```
        2min = 120s
 Da der Timer mit einem Zähltakt von 1khz (also 1000 tics/s)
 zählt ergibt sich dann für 120s: 120 * 1000 = 120.000 tics

        Der Periodenwert wäre dann: Period = 120.000 - 1 = 119.999
        Denn wir fangen immer von 0 an zu zählen.

        PERIOD = 119999
```

   - Sind diese errechneten Einstellungen in der Praxis möglich und erlaubt? Falls nein, warum nicht?

```
 Das hängt von der Breite des Timers ab:

Mit einem 16-bit Timer wo wir nur bis 65535 [also 2^(16) - 1 ]
 zählen können wäre das nicht möglich da 119.999 > 65535


Mit einem 32-bit Timer 4.294.967.295 [also 2^(32) - 1 ]
wäre es aber möglich denn 4.294.967.295 deutlich grösser als 119.999
```

- Bietet die Timer-Konfigurationsstruktur eine Einstellung, um das gewünschte Ziel den noch zu erreichen?

  ```
  Ja
  entweder ein 32-bit Timer benutzen

  oder wenn man unbedingt  ein 16-bit Timer benutzen will
  könnte man zb den Prescaller und Period so einstellen , dass
  ein Überlauf jeder 1s passiert und dann die Anzahl der Überlauf zählen

  Zb für 120s würde man so ungefähr 120 überlauf brauchen
  ```

- Zusatzaufgabe: Markieren Sie sich in Abbildung 16 des Reference Manuals den eingestellten Taktbaum von Ihrem Projekt und notieren Sie die dazugehörigen Frequenzen.

  ```
  Wir arbeiten mit einem SysCLK (SystemCoreClock) von 168Mhz
  Dieser Frequenz bekommt er vom dem PPL () , der selbst
  erstmal eine frequenzQuelle auswählen soll HSI oder HSE
  (im unserem Fall HSE = 8Mhz ) siehe: system_stm32f4xx.c

  aber Wie geht das alles?

  Dafür  müssen wir:

   *PLLM (Eingangsteiler: teilt die HSE-Quelle ) so auswählen, dass:
    (HSE / PPLM) = 1 oder 2MHZ  ergibt
    Mit HSE = 8 und PPLM = 8  haben wir dann VCO_INPUT = 8/8 = 1Mhz

   *PLLN (Multiplikator multipliziert die Eingangsfrequenz VCO) so auswählen, dass:

    VCO_OUTPUT zwischen 100 und 432Mhz liegt
    und was wir wollen ist 336Mhz da wir später teilen
    werden, um 168Mhz zu bekommen
    ALso PLLN = 336
    VCO_OUTPUT = (336 * 1) = 336 MHZ

   *PLLP (Ausgangsteiler teilt den VCO-Ausgang,um SYSCLK zu erzeugen)

   so auswählen, dass:

   VCO_OUTPUT / PLLP = 168Mhz
   Also PLLP wäre dann 2

   Und So bekommt dann SysCLK von PPL 168MHZ.
   Dannach bekommt AHB von SysCLK auch 168Mhz (AHBx_Prescaller = 1)
   dann bekommt APB1 42Mhz (da Prescaller = 4 also 168 / 4)
   und APB2 84Mhz (da Prescaller = 2 also 168 / 2) und die Timer,
  ```

```
die dann an den jeweiligen Bus angeschlossen sind bekommen
das doppelte also für Timer1 , den wir benutzen hätten wir
folgender Weg:

HSE(8MHz)-> PLL_CLOCK(168MHz) -> SYS_CLK(168MHz) -> AHB(168MHz)
-> APB2(Prescaller = 2 -> 84MHz) -> Timer1(168MHz wegen verdopplung)

Source: https://waijung1.aimagin.com/block_diagrams.htm
        https://www.youtube.com/watch?v=CUgPXt4JE_c&t=53s
```

- Zusatzaufgabe: Implementieren sie einen Treppenhausautomaten der ihre LED auf Tastendruck (oder Druck auf den Reset-Knopf) für 59 Sekunden aktiviert und anschließend wieder ausschaltet. Könnte hier der One-Pulse-Mode hilfreich sein?

```
    Dafür benutze ich die Funktion: treppenhaus_init(); schon im dot.c
    enthalten hier nur noch die main() zum testen denn ich weiss nicht
    wo ich die sonst noch ablegen kann:

Achtung: für USER_TASTE eher GPIOA PIN 0 benutzen

#include <lcd/lcd.h>
#include "stm32f4xx.h"
#include "utils/utils.h"
#include "dot/dot.h"

int main(void)
{
 HAL_Init();
 //dot_gpio_init();
 treppenhaus_init();

 while(1) {

  // button gedrückt? Led anschalten und timer starten
  if ((utils_gpio_port_read(JOY_GPIO_PORT) & JOY_PIN_PRESS) == 0) {

   //Led anschalten
   HAL_GPIO_WritePin(GPIOE, DOT, GPIO_PIN_RESET);

   // Reinitialise le flag de fin de compteur
   __HAL_TIM_CLEAR_FLAG(&tim_handle_struct, TIM_FLAG_UPDATE);

   // Timer starten
   HAL_TIM_Base_Start(&tim_handle_struct);
```

```
    }

    // prüft, ob die 59s vergangen sind
    if (__HAL_TIM_GET_FLAG(&tim_handle_struct, TIM_FLAG_UPDATE)) {

     // dann Led ausschalten
     HAL_GPIO_WritePin(GPIOE, DOT, GPIO_PIN_SET);

     // Flag löschen
     __HAL_TIM_CLEAR_FLAG(&tim_handle_struct, TIM_FLAG_UPDATE);
    }
   }
  }
```

```
Und Ja, der One-Pulse-Mode ist hier hilfreich, weil er es
ermöglich, den Timer einmalig zu starten.
Der Timer zählt dann bis zum Ende des definierten
Zeitraums (hier 59s) und stoppt automatisch, ohne dass wir noch was tun müssen.
Also Der Led schaltet sich dann nach 59s allein aus
```

10 NVIC

```
1. Erklären Sie, welche Auswirkung ein Define von
    "USE_HAL_TIM_REGISTER_CALLBACKS" auf 1 hat

   Wenn man USE_HAL_TIM_REGISTER_CALLBACKS auf 1 setzt:

   Dann darf man eine eigene Callback-Funktionen registrieren.

   Das heißt, man muss nicht immer nur HAL_TIM_PeriodElapsedCallback()

   verwenden ,und drin dann für verschiedenen Timer oder instance von

   Peripherie Default case oder if-bedingungen einfügen.

   Dadurch kann man flexibler programmieren, z.B. verschiedene
   Timer mit unterschiedlichen Callbacks.

   Wenn es auf 0 gesetzt ist, wird automatisch der Standard-Callback
   von HAL verwendet also: HAL_TIM_PeriodElapsedCallback() ,
   wo wir dann drin default-case oder if-bedingungen einbauen sollen
```

2. Beschreiben Sie auf welche Signalwechsel der EXTI eingestellt werden kann. Wie wird die Einstellung vorgenommen und was bedeuten diese Einstellungen?

- EXTI kann auf drei Arten von Flanken (Signalwechsel) eingestellt werden:

```
        GPIO_MODE_IT_RISING Reagiert auf steigende Flanke (LOW zu HIGH)
        GPIO_MODE_IT_FALLING Reagiert auf fallende Flanke (HIGH zu LOW)
        GPIO_MODE_IT_RISING_FALLING Reagiert auf beide Flanken

    *Diese Einstellung erfolgt über die Struktur GPIO_InitTypeDef, z.B:

     GPIO_InitTypeDef gpio_init = {
     .Pin = GPIO_PIN_7,
     .Mode = GPIO_MODE_IT_RISING, // also hier
     .Pull = GPIO_NOPULL,
     };
     HAL_GPIO_Init(GPIOB, &gpio_init);


3. Beschreiben Sie, wie man das EXTI Modul (stm32f4xx_hal_exti)
nutzen würde, um den EXTI separat von der GPIO Konfiguration zu konfigurieren

Man würde einfach diese Struktur:

/**
  * @brief  EXTI Configuration structure definition
  */
typedef struct
{
  uint32_t Line;      /*!< The Exti line to be configured. This parameter
                           can be a value of @ref EXTI_Line */
  uint32_t Mode;      /*!< The Exit Mode to be configured for a core.
                           This parameter can be a combination of @ref EXTI_Mode */
  uint32_t Trigger;   /*!< The Exti Trigger to be configured. This parameter
                           can be a value of @ref EXTI_Trigger */
} EXTI_ConfigTypeDef;

aus der stm32f4xx_hal_exti.h datei ausfüllen

Also Für GPIOx_Pin7 zb:

EXTI_ConfigTypeDef exti_config;
exti_config.Line = EXTI_LINE_7;
exti_config.Mode = EXTI_MODE_INTERRUPT;
exti_config.Trigger = EXTI_TRIGGER_RISING;

HAL_EXTI_SetConfigLine(&hexti, &exti_config);  // hexti: EXTI-Handle

wir würden noch ein EXTI_HandleTypeDef hexti; brauchen
Also das was wir mit HAL_EXTI_GetHandle() initialisierst.
also diese Funktion zur Überwachung, die in der Praktikumsanleitung steht
```

4. Was ist beim EXTI der Unterschied zwischen einem Interrupt und einem Event?

```
    Interrupt                                   Event
Ruft eine ISR-Funktion auf              Löst keine ISR aus
Beeinflusst den Programmfluss        Nur ein internes Signal
Wird vom Prozessor verarbeitet Kann von anderen Peripherien genutzt werden

Events sind eher für Hardware-Synchronisation benutzt
(z.B. Timer startet bei einem Event), Interrupts hingegen
für Reaktionen im Code also durch Aufruf von ISR-funktionen
```

5. Warum hat EXTI 23 Linien, obwohl es nur 16 GPIO-Pins pro Port gibt?

Jeder GPIO-Port hat zwar nur 16 Pins (0–15), also auch nur 16 EXTI- Leitungen für GPIOs.

Aber: Es gibt insgesamt 23 EXTI-Linien, weil die restlichen 7 für interne Funktionen gedacht sind:

Diese restlichen 7 Linien reagieren nicht auf GPIOs, sondern auf System- oder Peripherie-Ereignisse.

```
Siehe noch datei: stm32f4xx_hal_exti.h oder im reference manuel: S.380
 Figure 43. External interrupt/event GPIO mapping (STM32F42xxx and STM32F43xxx)
Alles ist dort schon Dokumentiert
```

11 Projekt 1 (Lüfterregelung)

```
1. Testen Sie das Skript pid_regulator.m , indem Sie im Command Window
(bei Octave) pid_regulator(1500) aufrufen.
 Die übergebene 1500 stellt hierbei die von dem simulierten Lüfter
  einzustellende Soll-Drehzahl an.
 Analysieren Sie das Verhalten des Reglers in den durch den Aufruf
 dargestellten Graphen.
```

Beantworten Sie die folgenden Fragen:

- Wie lange dauert die Einschwingphase? Die Einschwingphase dauert ungefährt 0.3s
- Was geschieht in dieser Phase?

```
Am Anfang ist der Ventilator zu langsam -> großer Fehler (e) -> daher
ist die PWM-Leistung am Anfang stark (nämlich fast 100%)
was dazu führt , dass Der Ventilator schneller wird, und der Fehler sinkt
und je kleiner wird der Fehler ( da sich der ist-wert dem
soll-wert annähert.) desto kleiner wird die PWM-Leistung
bis zur Stabilisierung aber hier kann man trotzdem sehen, dass
der ist-wert nie den soll-wert erreicht also eine
kleine bleibende Regelabweichung. Vermutlich wurde da der
"Proportional-Regler" angewendet
```

2. Versuchen Sie auch mal eine andere Soll-Drehzahl (z.B. 3500). Beschreiben Sie die entstandenen Unterschiede im Regelverhalten

- Als Unterschied kann man feststellen, dass der Fehler bei einem soll-wert von 3500 deutlich großer ist als bei 1500..

  Bei den Soll-wert von 1500 konnten wir die PWM-Leistung varieren sehen, sodass Fehler innerhalb von 0.5s sehr klein geworden war.. Bei 3500 hingegen bleibt die PWM-Leistung Konstant und relativ hoch (99trotzdem bleibt der Fehler relativ hoch als würde auch die maximal PWM-Leistung nicht reichen , um diesen Soll-wert zu erreichen

3. Justieren Sie nun die Regelparameter, um ein besseres (schnelleres) Verhalten des Reglers zu erreichen

- Beginnen Sie mit dem Proportionalanteil Kp , wobei Sie den Integralanteil Ki vorerst auf 0 set zen. Was passiert mit dem Regelverhalten, wenn Sie Kp zu groß bzw. zu klein wählen?

  - Wenn Kp zu klein ist (bedeutet PWM auch klein): Der Lüfter (also ist-wert) reagiert sehr langsam und erreicht die Soll-Drehzahl oft nicht vollständig. Die Regelabweichung bleibt also groß.
  - Wenn Kp sehr groß ist, hängt das Verhalten davon ab, ob man die Drehzahl erhöht oder senkt:
    - Beim Hochdrehen (z.B. von 0 auf 3500 also wenn wir den ist-wert auf 3500 setzen) gibt der Regler sofort 99 percent PWM. Da dies das Maximum ist, passiert kein Überschwingen – der Lüfter kommt nur langsam auf den soll-wert abert erreicht den nie.
    - Beim Herunterdrehen (z.B. von 3500 auf 1500 also wenn wir den soll-wert eher auf 1500), kann ein großer Kp zu starkem Abbremsen führen, was dann zu Schwingungen und instabilem Verhalten führt. aber hier sehen wir trotzdem wie langsam sich den ist-wert dem soll-wert annähert.

- Fügen Sie bei konstantem und optimalem Kp nun einen Integralanteil Ki größer als 0 hinzu. Was bewirkt ein geeignetes Ki ? Was ein zu großes Ki ?

  - Mit einem geeigneten ki verschwindet die bleibende Regelabweichung, sodass wir die Soll-wert gut erreichen
  - ein großes Ki hingegen führt zu mehr und lange schwingungen, Die Regelung wird dadurch unruhig und ungenau.

Nach dem Justieren der Parameter reagiert der Regler schnell und ohne dauerhafte Schwingungen bei:

- Kp = 2.0
- Ki = 3.8

12 Projekt 2 (Wetterstation)

12.1 I2C-Protokoll

1. Wann kann ein Datentransfer auf dem Bus initiiert werden? Welchen Zustand müssen die beiden Leitungen dafür haben?

ein Datentransfer auf dem Bus wird durch eine Start-bedingung initiiert, indem:

- SDA-Leitung von HIGH auf LOW wechselt
- während SCL-Leitung auf HIGH bleibt.

2. Was stellt eine Repeated Start Condition dar und wofür ist sie nützlich?

Eine Repeated Start Condition ist eine spezielle zweite Startbedingung, die ohne vorheriges Stoppsignal gesendet wird. Wird vor allem beim Wechsel der Richtung gesendet also wenn wir vom Lese-Vorgang zu Schreib-vorgang und ungekehrt wechsel wollen..

- Sie wird genutzt, um mehrere Übertragungen hintereinander auf dem Bus durchzuführen, ohne den Bus freizugeben.

3. Wann generiert ein Empfänger ein Not Acknowledge (NACK) auf der Leitung?

Ein NACK (Not Acknowledge) wird vom Empfänger gesendet, wenn:

- Er keine weiteren Daten empfangen möchte
- Ein Fehler aufgetreten ist (zb. ungültige Adresse)

Ein NACK wird generiert, indem der Empfänger die SDA-Leitung auf HIGH lässt, während das SCL-Signal HIGH ist

4. Wie läuft der Schreibprozess von Master zu Slave ab? Welche beiden Bytes müssen geschickt wer den, bevor die eigentlichen Datenbytes folgen?

1. **Start Condition** senden.
2. **Slave-Adresse + Write-Bit** (LSB = 0) senden also `ADDR + 0`
3. Auf **ACK** warten.
4. **Register-Adresse** senden (interne Adresse im Slave, zb. Sensor-Register).
5. Auf **ACK** warten.
6. Jetzt folgen die eigentlichen **Datenbytes**, die geschrieben werden sollen.

Die zwei Bytes vor den Daten sind also:

1 **Slave-Adresse + Write-Bit**

2 **Register-Adresse**

5. Worin unterscheidet sich ein Leseprozess von dem soeben beschriebenen Schreibprozess?

Richtung der Daten

- **Schreiben** -¿ Master **sendet** Daten an den Slave
- **Lesen** -¿ Master **empfängt** Daten vom Slave

Modus-Wechsel

Lesen erfordert oft einen **Repeated Start**, um vom **Schreibmodus** (zum Setzen der Adresse) in den **Lesemodus** zu wechseln – ohne den Bus freizugeben.

Verhalten des R/W-Bits

- **Schreibvorgang:** R/W = 0
- **Lesevorgang:** R/W = 1

12.2 BME280-Umgebungssensor

1. Bevor Sie sich um die Implementierung kümmern, muss der Sensor korrekt an ihr Board ange schlossen werden. Benutzen Sie das Datenblatt des BME280 [1], um folgende Fragen zu beantwor ten:

- Mit welcher Spannung kann der Sensor betrieben werden? Wo müssen Sie also die VCC-Leitung an Ihrem Board anschließen?
  - Der Sensor kann mit 3,3V betrieben werden. Die VCC-Leitung muss daher an einen 3,3V-Pin des Mikrocontrollers angeschlossen werden
- Sie sollen die I2C1-Peripherie Ihres Mikrocontrollers verwenden. Wo schließen Sie also die Lei tungen SDA und SCL an?
  - Die SCL-Leitung des BME280 wird mit PB6, und die SDA-Leitung mit PB7 des Mikrocontrollers verbunden.
- Welche Bedeutung haben die beiden restlichen Leitungen CS und ADDR (im Daten- blatt auch CSB und SDO genannt) bei Benutzung des I2C-Interfaces? Wo schließen Sie diese an?

  *CS / CSB (Chip Select):

  - Diese Leitung entscheidet, ob der Sensor im I^2C- oder SPI-Modus arbeitet.
  - CSB = High (3,3V): I^2C-Modus aktiv
  - CSB = Low (GND): SPI-Modus aktiv

  CSB muss auf 3,3V gelegt werden, damit der Sensor im I^2C-Modus arbeitet.

  *ADDR / SDO (I^2C-Adressenwahl):

  - Mit dieser Leitung wird die I^2C-Adresse des Sensors festgelegt.

- SDO = GND: Adresse = 0x76
- SDO = VDD (3,3V): Adresse = 0x77

SDO kann auf GND oder 3,3V gelegt werden, je nachdem, welche I²C-Adresse (0x76 oder 0x77) man verwenden möchte. Standardmäßig wird 0x76 verwendet, daher sollte SDO auf GND gelegt werden.

Zusammengefasst:

Pin am BME280	**Verbindung mit**	**Funktion**
VCC	3,3 V	Versorgungsspannung
GND	GND	Masse
SCL	PB6 (I2C1_SCL)	I²C-Taktleitung
SDA	PB7 (I2C1_SDA)	I²C-Datenleitung
CSB	3,3 V	Aktiviert den I²C-Modus
SDO	GND oder 3,3 V	Auswahl der I²C-Adresse (0x76 / 0x77)

Tabelle 1: Zusammenfassung der Anschlüsse des BME280 im I²C-Modus

2. Welche Funktionen bietet Ihnen die HAL an, um im Polling-Mode über I2C zu schreiben und zu lesen? Worin unterscheiden sich die beiden Arten im Hinblick auf die von Ihnen oben in den Aufga ben 4 und 5 beschriebenen Schreib- und Leseprozesse des I2C-Protokolls?

HAL-Funktion	**Beschreibung**
`HAL_I2C_Master_Transmit()`	Senden von Daten (Schreiboperation)
`HAL_I2C_Master_Receive()`	Empfangen von Daten (Leseoperation)
`HAL_I2C_Mem_Write()`	Schreiben in ein spezifisches Register des Slaves
`HAL_I2C_Mem_Read()`	Lesen aus einem spezifischen Register des Slaves

Tabelle 2: Wichtige HAL-I2C-Funktionen im Polling-Modus

- Unterschied zwischen Lese- und Schreibprozess
  - Beim Schreibprozess wird zuerst die Slave-Adresse mit dem Write-Bit (0) gesendet, dann das Register, und anschließend die Daten.
  - Beim Leseprozess muss man ebenfalls zuerst das Register angeben (also ein Schreibvorgang starten), danach aber ein Repeated Start senden, um in den Lesemodus zu wechseln. Dann folgt dieselbe Slave-Adresse, jedoch mit dem Read-Bit (1), und die Daten werden empfangen.

```
Diesen Unterschied berücksichtigt die HAL durch die getrennten
Funktionen Mem_Write() und Mem_Read(), die den Repeated Start
automatisch korrekt setzen.
```

3. Wie initialisieren Sie mithilfe der Bosch-Bibliothek den Sensor? Wie teilen Sie der Bibliothek mit, wo und wie der Sensor ausgelesen bzw. zu ihm Daten gesendet werden können?

3-a. Initialisierung des Sensors mit der Bosch-Bibliothek

Um den BME280-Sensor mit der Bosch-Bibliothek zu initialisieren, wird zuerst eine Struktur vom Typ `bme280_dev` deklariert und konfiguriert. Anschließend wird die Initialisierungsfunktion `bme280_init()` mit einem Zeiger auf diese Struktur aufgerufen.

Dabei werden während der Initialisierung bereits Register gelesen und beschrieben – deshalb muss die I²C-Peripherie des Mikrocontrollers vorher vollständig konfiguriert und aktiviert sein. genauso wie bei der

```
GPIO_InitTypeDef Struktur
```

also Struktur konfigurieren bzw ausfüllen und erst dann initialisieren

3-b. Übergabe der Kommunikationsfunktionen an die Bibliothek

Da die Bibliothek hardwareunabhängig ist, muss der Benutzer ihr mitteilen, wie Daten vom Sensor gelesen und an ihn gesendet werden können. Dazu verwendet man Funktionszeiger:

- `read`: Zeigt auf eine eigene I²C-Lesefunktion (z. B. `user_i2c_read`)
- `write`: Zeigt auf eine eigene I²C-Schreibfunktion (z. B. `user_i2c_write`)
- `delay_us`: Zeigt auf eine Delay-Funktion mit Mikrosekunden

Zusätzlich wird `intf_ptr` auf den Handle des verwendeten I²C-Interfaces gesetzt (z. B. `&hi2c1`).

Beispiel

```
struct bme280_dev dev;
dev.intf = BME280_I2C_INTF;
dev.read = user_i2c_read;
dev.write = user_i2c_write;
dev.delay_us = user_delay_us;
dev.intf_ptr = &hi2c1;

int8_t result = bme280_init(&dev);
```

4. In welchen Einheiten liefert die Bosch-Bibliothek die unterschiedlichen Sensordaten?

Sensordaten	**Datentyp**	**Einheit**
Temperatur	`double` oder `float`	Grad Celsius (°C)
Luftfeuchtigkeit	`double` oder `float`	relative Luftfeuchte in % rF
Luftdruck	`double` oder `float`	Pascal (Pa)

Tabelle 3: Übersicht der vom BME280 gelieferten Sensordaten und deren Einheiten

12.3 CAN-Protokoll

1. Welche Arten von Kommunikations-Frames gibt es beim CAN? Wofür werden diese verwendet?

Im CAN-Protokoll gibt es vier Haupttypen von Nachrichtenrahmen:

Rahmentyp	**Beschreibung**
Datenrahmen (Data Frame)	Überträgt die eigentlichen Nutzdaten (z. B. Temperatur)
Fernrahmen (Remote Frame)	Fordert einen anderen Knoten auf, seine Daten zu senden
Fehlerrahmen (Error Frame)	Wird gesendet, wenn ein Fehler auf dem Bus erkannt wurde
Überlastungsrahmen (Overload Frame)	Fügt eine Verzögerung ein, wenn ein Knoten überlastet ist

2. Wie ist ein Daten-Frame aufgebaut?

Ein CAN-Datenrahmen besteht aus folgenden Feldern:

- **SOF (Start of Frame)**: ein dominantes Bit (also 0), das den Beginn des Rahmens markiert.
- **Arbitrierungsfeld**: 11 oder 29 Bits (Nachrichtenkennung und Priorität).
- **Steuerfeld**: 6 Bits (enthält den Frame-Typ (Daten, Remote...)).
- **Datenfeld**: 0 bis 64 Bits (0 bis 8 Bytes) also die eigentlichen Nutzdaten
- **CRC-Feld**: 16 Bits zur Fehlererkennung.
- **ACK-Feld**: 2 Bits zur Bestätigung des Empfangs.
- **EOF (End of Frame)**: 7 rezessive Bits, die das Ende des Rahmens anzeigen.

`https://fr.wikipedia.org/wiki/Bus_de_donn%C3%A9es_CAN`

3. Um Kollisionen (gleichzeitiger Buszugriff) aufzulösen, verwendet CAN ein CSMA/CR-Verfahren. Welches Bit stellt hierbei das dominante und welches das rezessive dar? Welche Auswirkung hat dies auf die Priorisierung mittels des Identifiers einer Nachricht?

- Das Bit 0 stellt das dominante Bit dar und Das Bit 1 das recessif Bit
- Auswirkung
  - Wenn mehrere Teilnehmer gleichzeitig senden wollen, vergleichen sie ihre Identifier-Bits

- Dominant (0) gewinnt gegen rezessiv (1)
- Wer verliert (also hört zb eine 0 obwohl er eine 1 geschickt hat), bricht die Übertragung sofort ab.

4. Was stellt das Bit Stuffing beim CAN dar?

- Bit Stuffing ist eine Methode, um die Synchronisation der Geräte zu sichern
- CAN verwendet NRZ-Codierung (Non-Return-to-Zero). Wenn zb. 5mal das gleiche Bit kommt, gibt es keinen Signalwechsel, was schlecht für das Timing ist.
- Das Bit Stuffing sorg dann dafür, dass Nach 5 gleichen Bits in Folge, automatisch ein umgekehrter Bit eingefügt wird

12.4 CAN-Interface des Cortex-M4

1. Das CAN-Interface bietet drei verschiedene Testmodi:

Modus	Analogie (vereinfacht) :)	Beschreibung
Silent Mode	Wie ein Kind, das nur zuhört, aber nicht spricht.	Das bxCAN-Modul empfängt Nachrichten, sendet aber selbst keine. Es ist nützlich, um den CAN-Verkehr zu beobachten, ohne diesen zu stören.
Loop Back Mode	Wie ein Kind, das mit sich selbst redet.	Das bxCAN sendet Nachrichten an sich selbst, ohne den Bus zu benutzen. Nützlich für Selbsttests.
Combined Mode	Wie ein Kind, das leise mit sich selbst redet.	Kombination aus Silent und Loop Back Mode. Nachrichten werden intern gesendet und empfangen, ohne externen Busverkehr zu stören.

2. Filterbank im Maskenmodus mit 16-Bit-Skalierung
Gegeben:

- `CAN_FiR1 = 0xFE00AA00`
- `CAN_FiR2 = 0xFE00D600`
- Skalierung: 16-Bit
- Modus: Maskenmodus (Mask Mode)

a. Welche Bits werden geprüft?

Die Maske `0xFE00` entspricht im Binärformat: `11111110 00000000`.

- Nur die oberen 7 Bits (vom Bit 15-9) werden geprüft.
- Die unteren Bits werden ignoriert.

b. Prüfung von Standard-Identifiern

- **Filter 0**: Filterwert = `0xAA00` = `10101010 00000000`
- **Filter 1**: Filterwert = `0xD600` = `11010110 00000000`

ID	Binär	Passiert?	Filter
i. 0b11000110101	11000110101	Nein	–
ii. 0b10101011001	10101011001	Ja	Filter 0
iii. 0b11010111111	11010111111	Ja	Filter 1

```
c. Geben Sie für diese Konfiguration das entsprechend gefüllte CAN_FilterTypeDef an.
    Nehmen Sie sich für das korrekte Befüllen der Mask- und Identifier-Felder
    die Implementierung der Funktion HAL_CAN_ConfigFilter()
    in stm32f4xx_hal_can.c zur Hilfe.
```

```
CAN_FilterTypeDef filter;

filter.FilterBank = 0;
filter.FilterMode = CAN_FILTERMODE_IDMASK;      // Mask Mode
filter.FilterScale = CAN_FILTERSCALE_16BIT;     // 16-bit scaling
filter.FilterFIFOAssignment = CAN_FILTER_FIFO0;
filter.FilterActivation = ENABLE;


filter.FilterIdHigh     = 0xAA00;  // Id1
filter.FilterMaskIdHigh = 0xFE00;  // Mask1

filter.FilterIdLow      = 0xD600;  // Id2
filter.FilterMaskIdLow  = 0xFE00;  // Mask2

HAL_CAN_ConfigFilter(&hcan, &filter)
```

3. Wie viele unterschiedliche Filter bildet diese Konfiguration ab?

Filterbank	Modus	Skalierung	Anzahl Filter
0	ID Mask	32-bit	1
1	ID List (deaktiviert)	16-bit	0
2	ID List	32-bit	2
3	ID Mask	16-bit	2
4	ID Mask	16-bit	2

***Gesamtanzahl der aktiven Filter:** $1 + 2 + 2 + 2 = \boxed{7}$

***Ergänzung der Ansicht um die Filter-Nummern(wie in Abbildung 343 des Reference Manuals)**

Filterbank	Filtertyp	Filteranzahl	Filternummern
0	ID Mask (32-bit)	1	0
1	(Deaktiviert)	0	–
2	ID List (32-bit)	2	1, 2
3	ID Mask (16-bit)	2	3, 4
4	ID Mask (16-bit)	2	5, 6

4. Welche Felder bietet der Struct-Typ CAN_TxHeaderTypeDef an? Was legen sie fest? Ignorieren Sie das Feld .TransmitGlobalTime , es wird nur im "Time triggered communication mode" benötigt.

Die Struktur CAN_TxHeaderTypeDef enthält folgende wichtige Felder:

- `StdId` – Standard-Identifier (11 Bit)
- `ExtId` – Extended-Identifier (29 Bit)
- `IDE` – Identifier-Typ: `CAN_ID_STD` oder `CAN_ID_EXT`
- `RTR` – Nachrichtstyp: `CAN_RTR_DATA` oder `CAN_RTR_REMOTE`
- `DLC` – Länge der Nutzdaten (0–8 Bytes)
- `TransmitGlobalTime` – (nur bei Time Triggered Mode, wird ignoriert)

*Diese Felder legen das Format einer CAN-Nachricht fest.

5. Welche beiden Funktionen (ISR und Callback) muss man implementieren, wenn man auf Interrupts bezüglich des Eintreffens einer Nachricht in FIFO 0 reagieren möchte?

Um auf den Empfang einer Nachricht in `FIFO0` per Interrupt zu reagieren, sind zwei Dinge nötig:

1. Aktivieren der Interrupt-Benachrichtigung

```
HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING);
```

2. Callback-Funktion definieren

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    // Nachricht lesen:
    CAN_RxHeaderTypeDef rxHeader;
    uint8_t data[8];
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &rxHeader, data);


}
```