#### CT2 Praktikum

## **General Purpose Input/Output (GPIO)**

### 1 Einleitung

GPIOs erlauben eine universelle Verwendung von Ein- und Ausgabepins. Dazu müssen Sie vor dem Zugriff konfiguriert werden. Lesen und schreiben von Werten geschieht in der Regel mittels Register (Abbildung 1). Die Datei reg\_stm32f4xx.h definiert Strukturen und Makros für diese Registerzugriffe.

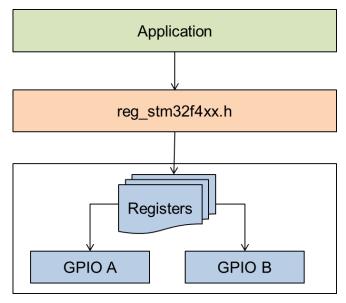


Abbildung 1: Zugriff auf GPIO mittels Register

In diesem Praktikum realisieren Sie die Konfiguration und die Ein- und Ausgabe mittels Registerzugriff für die GPIO der verwendeten Prozessorfamilie STM32F4xx.

#### 2 Lernziele

- Sie können GPIO mit Hilfe von technischen Dokumentationen initialisieren.
- Sie sind in der Lage GPIO zur Ein- und Ausgabe zu verwenden.
- Ihr Wissen zu den verschiedenen Konfigurationsoptionen ist gefestigt.
- Sie k\u00f6nnen die Strukturen des untersten Hardware Abstraction Layers (HAL), d.h. di (Basisadressen, Macros und structs in reg\_stmf4xx.h, zur Konfiguration von Registern in eigenen Programmen einsetzen.

#### 3 Material

- 1x CT Board
- 3x Verbindungsdrähte (female-female)

### 4 Vorbereitungsfragen

Bereiten Sie sich mit der Beantwortung der folgenden Fragen auf das Praktikum vor. Öffnen Sie das vorbereitete Projekt in Keil uVision und 'builden' Sie es. Analysieren Sie den gegebenen Source Code.

1. Ein Registerzugriff erfolgt beispielsweise über GPIOB->MODER. Dabei gibt GPIOB die Basisadresse an und MODER den Offset. Wo sind diese zwei Symbole definiert (Dateiname und Zeile).

GPIOB in reg\_stm32f4xx.h, Zeile 236 Moder in reg\_stm32f4xx.h, Zeile 195

Sie können gegebene Definitionen direkt von Keil suchen lassen. Klicken Sie dazu mit der rechten Maustaste auf den Text und wählen Sie "Go to Definition of…" aus. **Voraussetzung ist**, dass Sie mindestens einmal einen Build durchgeführt haben.

2. Was macht das folgende Macro?

#define GPIOB ((reg\_gpio\_t \*) 0x40020400)

Es ermöglicht einen einfachen zugriff auf die Register des GPIO-Port B, da diese Register ab 0x40020400 abfangen

3. Bitte beschreiben Sie in ein bis zwei Sätzen, was struct reg\_gpio\_t enthält?

Es ist eine kurze Representation von einem GPIO register. Damit kann man die Modi verändern, Geschwindigkeit, bit set, ect festlegen.

4. Erklären Sie, wie das C-Statement GPIOB->MODER = 0x00000280; funktioniert. Wie erfolgt der Zugriff auf das Register?

Dieser Command schreibt den Wert 0x00000280 direkt ins MODER-Register des GIOP-Port B, auf welchen GPIOB zeigt.

### 5 Applikation und Testaufbau mit den Verbindungsdrähten

GPIO A und B sind direkt auf die Ports P5 und P6 des CT Boards herausgeführt. Die Pinbelegung für GPIO B / Port P6 ist in Abbildung 2 dargestellt, das Pin-Mapping von GPIO A auf P5 ist identisch. Details finden sich im CT-Board Wiki / GPIO (https://ennis.zhaw.ch).

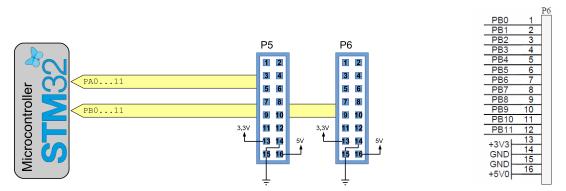


Abbildung 2: Pin-Mapping GPIO A und B auf Ports P5 und P6

In diesem Versuch werden Sie die Position der drei DIP-Schalter S10..S8 einlesen, die Werte über GPIO PB2..PB0 ausgeben und über GPIO PA2..PA0 wieder einlesen. Die eingelesenen Informationen werden Sie auf LED18..LED16 anzeigen. Zusätzlich verwenden wir LED10..LED8 und LED2..LED0 zu Debugging Zwecken. Abbildung 3 zeigt den Signalfluss.

Hierzu müssen die GPIO Ports entsprechend als Inputs und Outputs konfiguriert werden.

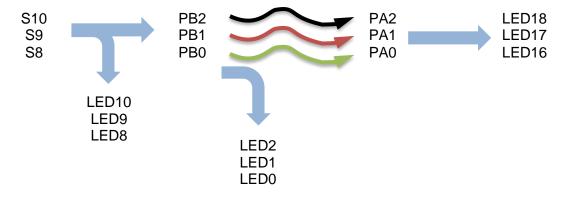


Abbildung 3: Signalfluss

Stellen Sie noch keine Verbindung zwischen den Pins auf dem CT Board her!

#### Eingelesene Werte GPIOA □C2 □C3 □C4 **GPIOA** LED19..16 (Input) JT1 Verbindungen über isolierte R1002 R1001 Drähte **GPIOB** (Output) LED3..0 LED11..B Zurückgelesene **DIP-Switches zur** Ausgabe GPIOB Bedienung Eingelesene DIP-Switch Positionen

### Abbildung 4 zeigt die verwendeten Elemente des CT-Boards.

Abbildung 4: Im Versuch verwendete Elemente des CT-Boards

### 6 Aufgaben

Im Folgenden implementieren Sie schrittweise einen einfachen Registerzugriff für die GPIO der Microcontrollerfamilie STM32F4xx.

Ergänzen Sie den vorgegebenen Code an den dafür gekennzeichneten Stellen. Verwenden Sie die Konstanten, Macros und structs aus reg stm32f4xx.h, um auf die Register zuzugreifen.

Um die Eigenschaften verschiedener GPIO-Konfigurationen zu untersuchen, konfigurieren wir die Ein- und Ausgänge unterschiedlich:

GPIO	Mode/Type	Pull-up/Pull-down	Speed	Observe on
PB0	Push-Pull Output	NO Pull-up/-down	Low Speed	LED0
PB1	Open Drain Output	NO Pull-up/-down	Medium Speed	LED1
PB2	Open Drain Output	Pull-up	High Speed	LED2
PA0	Input	Pull-down	-	LED16
PA1	Input	Pull-up	-	LED17
PA2	Input	NO Pull-up/-down	-	LED18

### 6.1 Konfiguration der Eingänge: PA2..PA0

Überlegen Sie, welche Werte für die Konfiguration der Eingänge in die Konfigurationsregister von Port GPIO A geschrieben werden müssen.

Füllen Sie dazu die untenstehende Tabelle aus (Sie haben in der Vorlesungs-Übungsaufgabe zur GPIO Konfiguration zwei unterschiedliche Arten kennen gelernt, wie die Masken für das Löschen und Setzen der Bits definiert werden können).

	Port Register Name	Register Bits (Nr)	Bit Werte (Binär)	Clear/Set Maske (shift, invert)	Clear/Set Maske (absolut, hex)
	PA0	Input			
	MODER	10	00	clr: ~(0x03 << 0)	clr: 0xFFFFFFC
				set: 0x00 << 0	set: 0x00
Direction /	PA1	Input			
Mode	MODER	32	00	clr: ~(0x03 <<2)	clr: 0xFFFFFF3
Wode				set: 0x00 <<0	set: 0x00
	PA2	Input			_
	MODER	54	00	clr: ~(0x03 <<4)	clr: 0xFFFFFCF
	WODER	54		set: 0x00 <<0	set: 0x00
	PA0	Pull-down			
	DUDDD	1.0	10	clr: $\sim (0x03 << 0)$	clr: 0xFFFFFFC
	PUPDR 10	10		set: 0x01 <<1	set: 0x02
Dull up /	PA1	Pull-up			
Pull-up / Pull-down	PUPDR 32	2 2	04	clr: ~(0x03 <<2)	clr: 0xFFFFFF3
r dii-dowii		01	set: 0x01 <<2	set: 0x04	
	PA2	No Pull-up/down			
	PUPDR 54	00	clr: ~(0x03 <<4)	clr: 0xFFFFFCF	
		54	00	set: 0x00 <<0	set: 0x00

Konfigurieren Sie die GPIOs im vorgegebenen Programmrahmen und lesen Sie in der Endlosschleife im Hauptprogramm die Werte von PA2..PA0 ein. Zeigen Sie diese auf LED18..LED16 an. Verwenden Sie hierzu struct CT LED->BYTE

Da der Elementzugriff im HAL über Pointer (\*) erfolgt, muss anstatt des Punktes (.) ein Pfeil (->) verwendet werden.

Beispiel: pointer->element entspricht (\*pointer).element

a) Bevor Sie das Programm ausführen: Welche LEDs erwarten sie bei der Ausführung hell, welche dunkel?

Hell: LED17 Dunkel: LED16 Undefiniert: LED18

b) Testen Sie Ihr Programm auf dem CT-Board. Entspricht das Ergebnis Ihren Erwartungen?

LED17 ist pull-up => level of VDD, LED16 ist pull-down => Level of GND LED18 ist no pull-up/pull-down => level ist floating, kann über Vout (Pin) gesteuert werden

c) Streichen Sie mit dem Finger über die Pins von P5 und beobachten Sie das Verhalten der LEDs. Wie erklären Sie sich dieses Verhalten?

Reibungselektrizität

Achtung: Das Menu erscheint erst im Debugger.

Sie können in der Keil uVision Entwicklungsumgebung die Werte der verschiedenen GPIO Register anschauen, indem Sie im Debugger Menü View -> System Viewer -> GPIO -> GPIOA auswählen und dann im Debug-Modus durch ihr Programm steppen. Im Beispiel ist für PA0 Pull-down gesetzt. 1-ZHAW\12-CT2\04-Praktika\P\_04\_GPI Core Peripherals k - μVision GPIOA View Project Flash Debug Peri dow Help RNG ✓ Status Bar **譚 //≟ //½ | 2** Value DCMI Toolbars MODER1 0x00 FMC 📆 - 💹 - 🔢 MODER0 Project Window DBG ⊕ OTYPER Books Window ▶ PDR |= (0x0 DMA ⊕ GPIOB\_OSPEE... PUPDR {} Functions Window RCC PUPDR15 0x01 (). Templates Window GPIO ✓ GPIOA PUPDR14 0x02 Source Browser Window SYSCFG ✓ GPIOB PUPDR13 0x01 Build Output Window GPIOC PUPDR12 0x00 X Error List Window SDIO GPIOD PUPDR11 0x00 PUPDR10 0×00 Find In Files Window ADC PUPDR9 USART **GPIOF** Command Window DAC GPIOG PUPDR7 0x00 Disassembly Window **PWR** PUPDR6 **GPIOH** 0x00 Symbols Window PUPDR5 0×00 **GPIOI** Registers Window PUPDR4 0x00 IWDG Call Stack Window PUPDR3 0x00 WWDG GPIOK PUPDR2 0x00 Watch Windows RTC PUPDR1 Memory Windows BLE(); PUPDR0 TIM Serial Windows ⊕ IDR GPIOA specif Ethernet Analysis Windows MODER CRC [Bits 31..0] RW (@ 0x40020000) GPIO port mode register Trace PEEDR = 0x00USB\_OTG\_FS System Viewer  $PDR = 0 \times 6400$ ▶ YPER = 0x000 CAN Toolbox Window GPIOB GPIOA RL = 0x00000NVIC  $RH = 0 \times 000000$ FLASH

### 6.2 Konfiguration der Ausgänge: PB2..PB0

Implementieren Sie die notwendigen Registerzugriffe für eine Output-Konfiguration. Gleich wie bei der Konfiguration der Inputs, müssen auch hier die entsprechenden Registerbits zuerst gelöscht und anschliessend richtig beschrieben werden.

	Port Register Name	Register Bits (Nr)	Bit Werte (Binär)	Clear/Set Maske (shift, invert)	Clear/Set Maske (absolut, hex)
	PB0	Gen. Purpose Output			
Direction / Mode	MODER	10	01	clr: ~(0x03 <<0) set: 0x01 <<0	clr:0xFFFFFC set0x01
	PB1	Gen. Purpose Output	<u>'</u>		
	MODER	32	01	clr: ~(0x03 <<2) set: 0x01 <<2	clr:0xFFFFFF3 set0x04
	PB2	Gen. Purpose Output			
	MODER	54	01	clr: ~(0x03 <<4) set: 0x01 <<4	clr:0xFFFFFCF set0x10
	PB0	Push-Pull			
	OTYPER	0	0	clr: ~(0x01 <<0) set: 0x00 <<0	clr: 0xFFFFFFF set: 0x00
	PB1	Open Drain	•	37133	
Type	OTYPER	1	1	clr: ~(0x01 <<1) set: 0x01 <<1	clr: 0xFFFFFFD set: 0x02
	PB2	Open Drain			
	OTYPER	2	1	clr: ~(0x01 <<2) set: 0x01 <<2	clr: 0xFFFFFFB set: 0x04
	PB0	No Pull-up/-down			
	PUPDR	10	00	clr: ~(0x03 <<0) set: 0x00 <<0	clr: 0xFFFFFFC set: 0x00
Dull /	PB1	No Pull-up/-down			
Pull-up / Pull-down	PUPDR	32	00	clr: ~(0x03 <<2) set: 0x00 <<0	clr: 0xFFFFFF3 set: 0x00
	PB2	Pull-up			
	PUPDR	54	01	clr: ~(0x03 <<4) set: 0x01 <<4	clr: 0xFFFFFCF set: 0x10
	PB0	Low			
	OSPEEDR	10	00	clr: ~(0x03 <<0) set: 0x00 <<0	clr: 0xFFFFFFC set: 0x00
	PB1	Medium			
Speed	OSPEEDR	32	01	clr: ~(0x03 <<2) set: 0x01 <<2	clr: 0xFFFFFF3 set: 0x04
	PB2	High			
	OSPEEDR	54	10	clr: ~(0x03 <<4) set: 0x01 <<5	clr: 0xFFFFFCF set: 0x20

Achten Sie auf **OTYPER** (Output Type Register). Es hat eine andere Bitbreite als die übrigen Register.

Konfigurieren Sie PB2..PB0 als Output wie in der Tabelle vorbereitet.

#### 6.3 Ausgabe auf PB2..PB0

Erweitern Sie die Implementierung in der Endlosscheife im Hauptprogramm so, dass Sie die Werte der DIP Switches S10..S8 einlesen und sowohl auf LED10..LED8 ausgeben als auch in das **ODR** (Output Data Register) schreiben.

Lesen Sie den ausgegebenen Wert aus dem **ODR** wieder zurück und geben Sie ihn auf LED2..LED0 aus.

LED2..LED0 sollten nun die Position von S10..S8 anzeigen, gleich wie LED10..LED8.

#### 6.4 Vervollständigen des Aufbaus

Trennen Sie während des Anbringens der Drähte das CT Board von der Versorgungsspannung.

Bei gröberen Fehlern kann Ihr CT-Board Board Schaden nehmen und zerstört werden. (siehe auch Kapitel 8.1).

Verbinden Sie die jeweils entsprechenden Pins über die Verbindungsdrähte, also PA0 mit PB0, PA1 mit PB1 und PA2 mit PB2.

Schalten Sie die DIP Switches S10..S8 und prüfen Sie, ob die Übertragung über die Verbindungsdrähte funktioniert. Stimmen die Anzeigen auf allen LEDs miteinander überein?

#### 6.5 Untersuchung verschiedener I/O Kombinationen

Durch Umstecken der Verbindungsdrähte können Sie nun unterschiedlich konfigurierte Ausund Eingänge miteinander verschalten. Prüfen Sie alle Kombinationen und tragen die Ergebnisse in die folgende Tabelle ein. Zur Stabilitätsprüfung können Sie wieder die «Fingerprobe» vornehmen (wenn ihr Finger schmal genug ist, dass sie die Pins von P5 von unten berühren können). Für die Kombination Push-Pull / Pull-Down sind die Ergebnisse bereits eingetragen.

		Inputs		
		PA0 / LED16: Pull-Down	PA1 / LED17: Pull-Up	PA2 / LED18: No Pull
Outputs	PB0 / S8: Push-Pull, NO pull-up/down	0: 0 1: 1	0: 6 1: 1	0: 0 1: 1
	PB1 / S9: Open Drain, NO pull-up/down	0: 6 1: 0	0: 6 1: 1	0: 0 1: 1
	PB2 / S10: Open Drain, pull-up	0: 6 1: 0	0: 0 1: 1	0: 0 1: 1

Gibt es Kombinationen, die Sie kritisch sehen?

# 7 Bewertung

Bewertungskriterien	Gewichtung
Die Funktionen für die GPIO Inputs wurden gemäss Aufgabe implementiert und die Fragen können erklärt werden.	1/4
Die Output-Konfiguration ist implementiert und getestet.	1/4
Die Funktionen für die GPIO Outputs wurden gemäss Anforderungen getestet.	1/4
Die Untersuchung wurde komplett durch geführt und auffällige Ergebnisse können erklärt werden.	1/4

### 8 Anhang

### 8.1 Rücksetzen der Debugger Pins

Nur verwenden, falls die GPIO Pins des Debuggers durch einen fehlerhaften Praktikumscode falsch konfiguriert wurden und kein Code mehr aufs Board geladen werden kann.

Bei GPIOA und GPIOB sind nur die Pins 11-0 direkt abgreifbar. Die Pins 15-12 werden intern vom Discovery Board verwendet für z.B. Verbindung zum Debugger.

Falls die Pins 15-12 um konfiguriert werden, kann nicht mehr über die Debug-Schnittstelle auf den Microcontroller zugegriffen werden => Microcontroller nicht mehr umprogrammierbar. Um diesen Zustand zu verlassen, muss der Microcontroller anders booten (aus dem System Memory anstatt dem Flash Memory). Danach ist der Zugriff über die Debug-Schnittstelle wieder möglich.

Jetzt müssen die Registerzugriffe angepasst werden, die die Umkonfiguration der Pins 15-12 zufolge hatte, ansonsten ist die Debug-Schnittstelle beim nächsten Reset wieder nicht verfügbar!

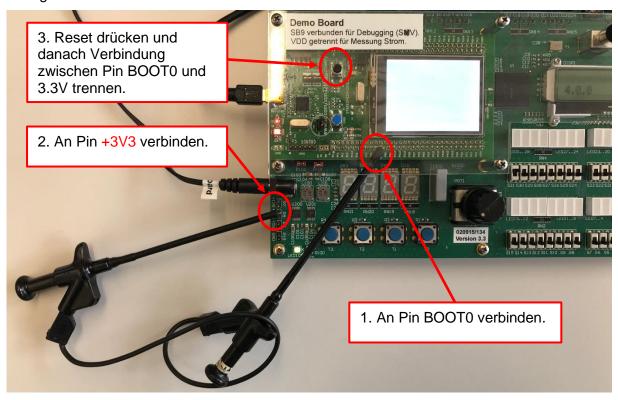


Abbildung 5: Notwendige Verbindung, um aus dem System Memory zu booten