STM32 - GPIO und Timer

u23 2013

andy, florob, gordin, ike, meise, tobix, zakx

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

Cologne
2013-10-28

GPIO

- GPIO = **G**eneral **P**urpose **I**nput/**O**utput
- = durch Software wackelnde Pins am Mikrocontroller
- Kennen zwei Modi: Input und Output
- Standardkonfiguration eines Pins = GPIO Input
- STM32F4 hat GPIOA bis GPIOI mit je 16 Pins (ne Menge!)
- Als Vergleich Atmega32: GPIOA bis GPIOD mit je 8 Pins

Konfiguration

Der Kram muss konfiguriert werden:

```
GPIO_InitTypeDef GPIO_InitStructure;

/* Enable the GPIO_LED Clock */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

/* Configure the GPIO_LED pins */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Der Reihe nach (1)

Clock einschalten, damit der GPIO-Kern irgendwas tut:

```
/* Enable the GPIO Clock */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
```

Der Reihe nach (2)

Wir konfigurieren hier Pins 12 bis 15 ...

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
    GPIO_Pin_14 | GPIO_Pin_15;
```

... als Ausgänge ...

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
```

... im Modus Push/Pull ...

```
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

Der Reihe nach (2)

Wir konfigurieren hier Pins 12 bis 15 ...

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
    GPIO_Pin_14 | GPIO_Pin_15;
```

... als Ausgänge ...

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
```

... im Modus Push/Pull ...

```
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

Der Reihe nach (2)

Wir konfigurieren hier Pins 12 bis 15 ...

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
    GPIO_Pin_14 | GPIO_Pin_15;
```

... als Ausgänge ...

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
```

... im Modus Push/Pull ...

```
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

Der Reihe nach (3)

... mit aktiviertem Pullup (eigentlich sinnfrei bei Push-Pull-Ausgängen) ...

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
```

... und max. 50 MHz Geschwindigkeit.

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

BÄM!

```
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Der Reihe nach (3)

... mit aktiviertem Pullup (eigentlich sinnfrei bei Push-Pull-Ausgängen) ...

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
```

... und max. 50 MHz Geschwindigkeit.

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

BÄM!

```
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Der Reihe nach (3)

... mit aktiviertem Pullup (eigentlich sinnfrei bei Push-Pull-Ausgängen) ...

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
```

... und max. 50 MHz Geschwindigkeit.

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

BÄM!

```
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Pins nutzen

So setzt man Pins:

```
GPIO_SetBits(GPIOD, GPIO_Pin_12 | GPIO_Pin_15);
```

So löscht man Pins:

```
GPIO_ResetBits(GPIOD, GPIO_Pin_13 | GPIO_Pin_14);
```

So liest man Pins:

```
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == Bit_SET)
{
...
}
```

Noch mehr, lest das mal selbst nach: *stm32f4xx_gpio.h*

Pins nutzen

So setzt man Pins:

```
GPIO_SetBits(GPIOD, GPIO_Pin_12 | GPIO_Pin_15);
```

So löscht man Pins:

```
GPIO_ResetBits(GPIOD, GPIO_Pin_13 | GPIO_Pin_14);
```

So liest man Pins:

```
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == Bit_SET)
{
...
}
```

Noch mehr, lest das mal selbst nach: *stm32f4xx_gpio.h*

Pins nutzen

So setzt man Pins:

```
GPIO_SetBits(GPIOD, GPIO_Pin_12 | GPIO_Pin_15);
```

So löscht man Pins:

```
GPIO_ResetBits(GPIOD, GPIO_Pin_13 | GPIO_Pin_14);
```

So liest man Pins:

```
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == Bit_SET)
{
...
}
```

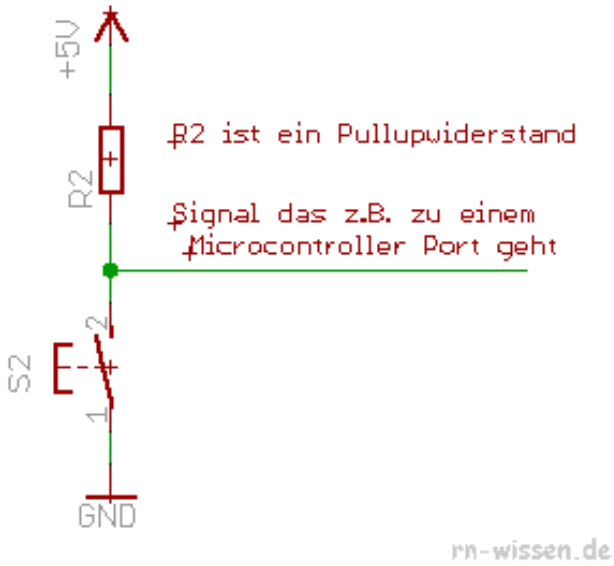
Noch mehr, lest das mal selbst nach: *stm32f4xx_gpio.h*

Pullup/down Widerstände

Annahme: Wir klemmen einen Schalter an einen Input an.

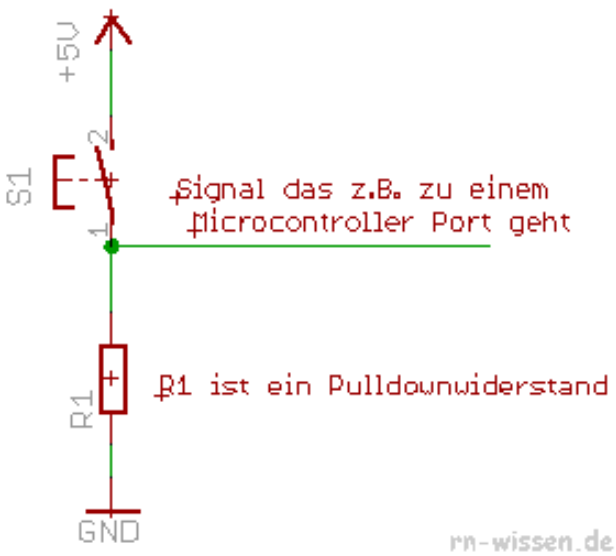
1. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter geschlossen ist?
2. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter offen ist?

Pullup-Widerstand



1. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter geschlossen ist?
2. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter offen ist?

Pulldown-Widerstand



1. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter geschlossen ist?
2. Welchen Pegel hat der Mikrocontrollerpin, wenn der Schalter offen ist?

Pinmodi

GPIO_Mode ist ziemlich selbsterklärend:

- *GPIO_Mode_IN* – Input
- *GPIO_Mode_OUT* – Output
- *GPIO_Mode_AF* – Auxiliary Function
- *GPIO_Mode_AN* – Analog Input

Pinmodi

GPIO_OType (O = Output) kennt nur zwei Modi:

- *GPIO_OType_PP* – Push/Pull = hartes ziehen oder drücken auf 0V oder 3,3V
- *GPIO_OType_OD* – OpenDrain = hartes ziehen auf 0V oder hochohmig (mit Pullup dann plötzlich high)

Pinmodi

GPIO_PuPd kümmert sich um die Pullups/Downs:

- *GPIO_PuPd_NOPULL* – Kein Pullup/down
- *GPIO_PuPd_UP* – Pullup an
- *GPIO_PuPd_DOWN* – Pulldown an

GPIO Interrupts

Anstatt die ganze Zeit die Pins abzufragen, ob da was passiert ist, können sie auch Interrupts werfen. Sample dazu ist *04_gpiointerrupt*.

GPIO Interrupts

Pin konfigurieren wie sonst auch. PA0 ist der User-Button, auf der Platine ist ein Pullup-Widerstand. Bonuspunkte für Leute die ihn mir im Schaltplan raussuchen und mir sagen wie er heisst und wie groß er ist.

```
//Clock einschalten
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

// Pinmodus konfigurieren
GPIO_Init(GPIOA, &(GPIO_InitTypeDef){
.GPIO_Speed = GPIO_Speed_50MHz,
.GPIO_Mode = GPIO_Mode_IN,
.GPIO_OType = GPIO_OType_PP,
.GPIO_PuPd = GPIO_PuPd_NOPULL, //no internal pullup or pulldown, is present on PCB
.GPIO_Pin = GPIO_Pin_0
});
```

GPIO Interrupts

GPIO Interrupts werden durch den EXTI-Kern behandelt.
Konfigurieren...

```
EXTI_Init(&(EXTI_InitTypeDef){
.EXTI_Line = EXTI_Line0,
.EXTI_Mode = EXTI_Mode_Interrupt,
.EXTI_Trigger = EXTI_Trigger_Rising,
.EXTI_LineCmd = ENABLE
});
```

Was geht da? Ich bin faul findet das zur Abwechslung mal selbst raus ;) *STM32 Reference Manual Page 199ff*

GPIO Interrupts

Nur soviel:

- Es gibt 16 Leitungen im Chip: EXTI0 bis EXTI15
- An EXTI0 hängen: PA0, PB0, PC0, PD0, ..., PI0
- An EXTI1 hängen: PA1, PB1, PC1, PD1, ..., PI1
- ... klar, oder?
- Wir haben hier aber EXTI0 konfiguriert. Wie kriegen wir jetzt also PA0 and EXTI0?

So:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

GPIO Interrupts

Nur soviel:

- Es gibt 16 Leitungen im Chip: EXTI0 bis EXTI15
- An EXTI0 hängen: PA0, PB0, PC0, PD0, ..., PI0
- An EXTI1 hängen: PA1, PB1, PC1, PD1, ..., PI1
- ... klar, oder?
- Wir haben hier aber EXTI0 konfiguriert. Wie kriegen wir jetzt also PA0 and EXTI0?

So:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

GPIO Interrupts

Als letztes müssen wir den Interrupt im Interrupt-Controller noch einschalten:

```
NVIC_Init(&(NVIC_InitTypeDef){
    .NVIC_IRQChannel = EXTI0_IRQn,
    .NVIC_IRQChannelPreemptionPriority = 0x00,
    .NVIC_IRQChannelSubPriority = 0x00,
    .NVIC_IRQChannelCmd = ENABLE
});
```

GPIO Interrupts

Und so sieht die Interruptroutine aus:

```
void EXTI0_IRQHandler()
{
    //huh? You talking to me?
    if(EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        GPIO_ToggleBits(GPIOD, GPIO_Pin_12 | GPIO_Pin_13 |
            GPIO_Pin_14 | GPIO_Pin_15);

        //Clear the interrupt bit and tell the controller we handled the
            interrupt
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}
```

GPIO Interrupts

War viel? Keine Sorge, guckt ins Example 01 bis 04. Da steht alles kommentiert und zusammen in jeweils einem File.

Timer

- Timer sind vom Konzept her relativ einfach, aber durch die vielfältigen Anwendungsgebiete doch relativ komplexe Biester
- Deswegen werden wir hier das Thema nur kurz anschneiden (wie leider fast alles -_-)
- Ich beherrsche auch nicht alle Modi auswendig, mit Grundverständnis und etwas Googlen + Datasheets kommt man aber ganz gut hin
- Timer sind simpel: Sie haben ein x Bit breites Register und bei jedem Auftreten eines Clockimpulses wird das register inkrementiert
- Kühles Howto zum nachlesen: `http://visualgdb.com/tutorials/arm/stm32/timers/`

Timer

Annahme: 16 Bit Timer, 40 MHz Clock, wie lange tickt das Ding, bis es überläuft?

- $2^{16} = 65536$, Wertebereich des Timers als 0 bis 65535
- $40 MHz = 40 * 10^6$ Ticks pro Sekunde = 40 000 000 Hz
- Der Timer inkrementiert also 40000000 pro Sekunde
- Damit läuft der Timer als $40000000 MHz / 65536 = 610,3515625$ pro Sekunde über
- Somit ist $1s / 610,3515625 = 0,0016384s = 1,6384ms$
- **Würde man also einen 16 Bit Timer mit 40MHz von 0 an laufen lassen, hat man einen Überlauf alle 1,6384ms**
- Rechnet mal, wie lange ein 32 Bit Timer bei 100 MHz braucht, bis der überläuft!

Timer

Annahme: 16 Bit Timer, 40 MHz Clock, wie lange tickt das Ding, bis es überläuft?

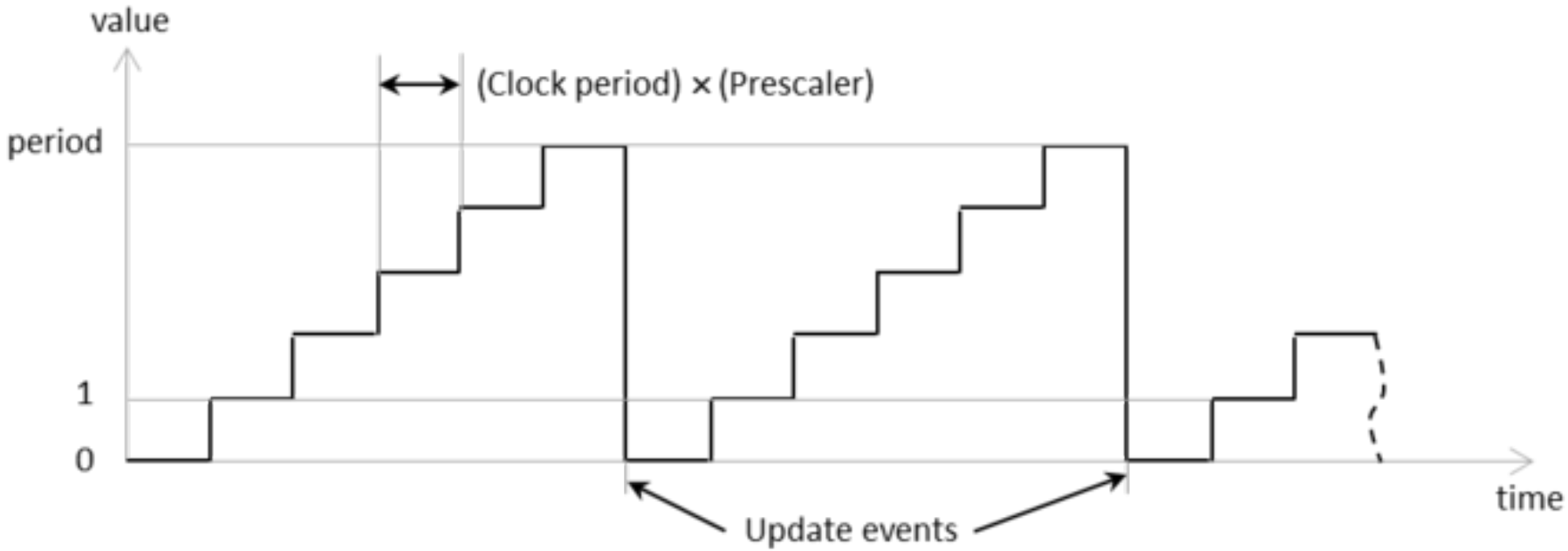
- $2^{16} = 65536$, Wertebereich des Timers als 0 bis 65535
- $40 MHz = 40 * 10^6$ Ticks pro Sekunde = 40 000 000 Hz
- Der Timer inkrementiert also 40000000 pro Sekunde
- Damit läuft der Timer als $40000000 MHz / 65536 = 610,3515625$ pro Sekunde über
- Somit ist $1s / 610,3515625 = 0,0016384s = 1,6384ms$
- **Würde man also einen 16 Bit Timer mit 40MHz von 0 an laufen lassen, hat man einen Überlauf alle 1,6384ms**
- Rechnet mal, wie lange ein 32 Bit Timer bei 100 MHz braucht, bis der überläuft!

Timer

Annahme: 16 Bit Timer, 40 MHz Clock, wie lange tickt das Ding, bis es überläuft?

- $2^{16} = 65536$, Wertebereich des Timers als 0 bis 65535
- $40 MHz = 40 * 10^6$ Ticks pro Sekunde = 40 000 000 Hz
- Der Timer inkrementiert also 40000000 pro Sekunde
- Damit läuft der Timer als $40000000 MHz / 65536 = 610,3515625$ pro Sekunde über
- Somit ist $1s / 610,3515625 = 0,0016384s = 1,6384ms$
- **Würde man also einen 16 Bit Timer mit 40MHz von 0 an laufen lassen, hat man einen Überlauf alle 1,6384ms**
- Rechnet mal, wie lange ein 32 Bit Timer bei 100 MHz braucht, bis der überläuft!

Timer



Quelle: `http://visualgdb.com/tutorials/arm/stm32/timers/`

Timer

Allgemeine Formeln

- $tickDauer = \frac{1}{timerFrequenz}$
- $zeitBisÜberlauf = zählSchritte * tickDauer$

Die Dinger sind so einfach, dass man sich die mit etwas Übung auch schnell selbst herleiten kann.

STM32 Timer

Welche Timer haben wir denn und wie schnell ticken die wirklich?

- Dieses clocking Excelsheet in *docs/* verrät euch das
- TIM 1 und 8: 16 Bit
- TIM 2 bis 5: 16 Bit und 32 Bit
- TIM 9 bis 14: 16 Bit
- TIM 6 und 7: 16 Bit
- Verschiedene Timer haben verschiedene Features
- Wie immer alles im Datasheet

Timer konfigurieren

Wir konfigurieren mal einen Timer im sogenannten TimeBase Modus (geklaut aus *05_ledpwm*).

```
//Enable Timer 4 clock
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

//We want the timer to tick with a frequency of 1MHz, calculate a prescaler
uint32_t PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 1000000) - 1;

//20 kHz PWM period (i.e. 50uS period)
uint32_t period = 1000000 / 20000;

//Configure the timer
TIM_TimeBaseInit(TIM4, &(TIM_TimeBaseInitTypeDef){
.TIM_Period = period - 1,
.TIM_Prescaler = PrescalerValue,
.TIM_ClockDivision = 0,
.TIM_CounterMode = TIM_CounterMode_Up,
});
```

Das Ding tickt einfach nur mit der angegebenen Frequenz vor sich hin. Mehr passiert da nicht.

Timer konfigurieren

Als letztes noch den Timer starten:

```
TIM_Cmd(TIM4, ENABLE);
```

Tick, tack...

Aktuellen Tickwert holen:

```
TIM_GetCounter(TIM4);
```

Timer konfigurieren

Als letztes noch den Timer starten:

```
TIM_Cmd(TIM4, ENABLE);
```

Tick, tack...
Aktuellen Tickwert holen:

```
TIM_GetCounter(TIM4);
```

PWM

- Jetzt, wo wir wissen, wie Timer generell funktionieren, lassen wir mal LEDs flackern
- Helligkeit von LEDs wird über PWM (= Pulse Width Modulation = Pulsweitenmodulation) ermöglicht
- Zu gut Deutsch: LED ist X% der Gesamtzeit an und 100%-X% der Gesamtzeit aus

PWM

- Fürs messen von Zeiten nehmen wir immer Timer auf einem Mikrocontroller
- Wir machen die LED an, fangen bei 0 an zu Zählen und warten etwas
- Wenn der Timer einen gewissen Wert erreicht hat (= gewisse Zeit verstrichen ist), machen wir die LED wieder aus ...
- ... und warten, bis der Timer überläuft
- Wenn man das schnell genug macht, sieht man das als Mensch durch die Trägheit der Augen nichtmal
- Repeat...
- Easy, oder?

PWM

- Fürs messen von Zeiten nehmen wir immer Timer auf einem Mikrocontroller
- Wir machen die LED an, fangen bei 0 an zu Zählen und warten etwas
- Wenn der Timer einen gewissen Wert erreicht hat (= gewisse Zeit verstrichen ist), machen wir die LED wieder aus ...
- ... und warten, bis der Timer überläuft
- Wenn man das schnell genug macht, sieht man das als Mensch durch die Trägheit der Augen nichtmal
- Repeat...
- Easy, oder?

PWM

Rechenbeispiel:

- Eine Timerperiode ist 50µS lang
- Angenommen der Timer zählt innerhalb dieser 50µS immer bis 50 und wird dann wieder auf 0 zurückgesetzt ...
- ... und wir setzen den OutputCompare auf 25, ist 50% der Zeit die LED an, sonst aus
- Bei 10 wären das 5µS an und den die Restlichen 45µS aus

Output Compare Config

Ich kopier mal wieder Code (*05_ledpwm*):

```
//PD12 auf Alternate Function legen (Achtung! Bei GPIO Config auch AF als Mode angeben!)
GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);

//Timer konfigurieren, so das er passend tickt
<...>

TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

// PWM1 Mode configuration: Channel1 (GPIOD Pin 12)
TIM_OC1Init(TIM4, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

// Comparewert setzen
TIM_SetCompare1(TIM4, 10);
```

GPIO-Aufgaben

1. Versucht mal selbst die LEDs zum Blinken zu bringen (also selbst Pinmodi konfigurieren etc.)
2. Versucht mal den Input für den Userbutton richtig zu konfigurieren und Dinge damit zu tun
3. Probiert auch mal die Interrupts aus
4. Was fällt euch auf, wenn ihr den Taster drückt? Wie oft wird eure programmierte Aktion pro Druck ausgelöst?
5. Behebt das festgestellte Problem (entprellen)! Wie ist mir egal.

Timer-Aufgaben

1. Konfiguriert mal einen Timer im TimeBase-Mode und lasst ihn mit einer von euch festgelegten Frequenz ticken (also ohne Output Compare oder son Kram, nur der Timer)
2. Wenn das geht, könnt ihr in einer Schleife regelmäßig den Wert auslesen und ab einem Schwellwert ne LED an- und ausschalten -> Blinkerei! (siehe auch: `http://visualgdb.com/tutorials/arm/stm32/timers/`)
3. Spielt mal mit den Comparewerten in dem LED-PWM example und guckt was sich ändert. Ich komm dann mal mit nem Logicanalyzer rum und zeig auch live wie das aussieht.

Timer-Aufgaben

4. Wenn es euch interessiert, stellt den Timer mal so ein, dass er alle x Millisekunden überläuft und probiert mal in einem zweiten Schritt einen Interrupt bei einem Timer-Overflow zu erzeugen (Hints: *TIM_ITConfig()*, *TIMx_IRQHandler()*, *TIM_ClearITPendingBit()*, *TIM_GetITStatus()*). Damit könnt ihr jetzt relativ genau Zeit messen und wisst, wie die Delay()-Funktion funktioniert

Sonstiges

- Letztes mal wollte jemand wissen, was das EVENTOUT das bei den meisten Pins als auxiliary function zu sehen war bedeutet. Hier stehts:
  `http://electronics.stackexchange.com/questions/28740/what-is-the-stm32-event-eventout`
- Spielt mal damit, wenn ihr Lust habt, das hört sich lustig an!