Mikroprozessortechnik

Prof. Dr. Michael Lipp

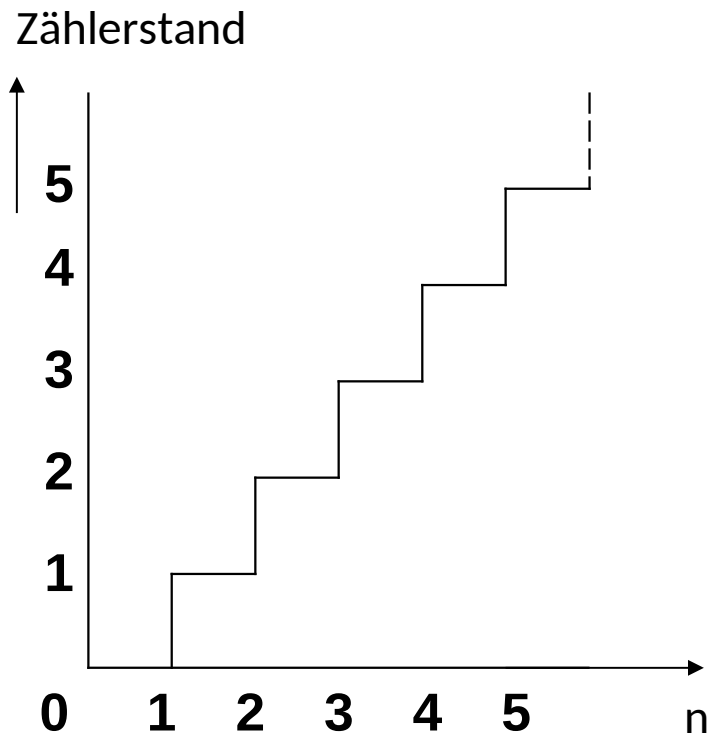
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Timer

Timer

- **Timer sind**
  - in Hardware realisierte Zähler, die
  - den Zählerstand mit einer vorgegebenen Taktrate (f_{timer}) ändern und
  - auf bestimmte Weise reagieren, wenn Zählerstände vordefinierte Bedingungen erfüllen.



$$t = n \cdot \frac{1}{f_{timer}}$$

Timer

- **Generierung verlässlicher Zeitbasis zur Steuerung z.B.:**
  - Wartezeiten mit befriedigender Genauigkeit einstellen
  - Einlesen von Daten von externen (über Ports) oder internen (ADC, USART, ...) IO-Bausteinen in bestimmten Zeitintervallen
- **Timer generiert Ereignisse in genau zu bestimmenden**

Zeitintervallen $\Delta t_{Ereignis} = x \cdot \frac{1}{f_{Timer}}$ **z. B.**

  - Setzen von Flags in bestimmten Registern
  - Generieren von Interrupts
  - Ein-/Ausschalten von Ausgängen

Timer im STM32F401RE

- **Verfügbare Timer**
  - 1 „Advanced Control Timer“ (TIM1)
  - 7 „General Purpose Timer“ in unterschiedlicher Ausprägung (TIM2 – TIM5, TIM9 – TIM11)
- **Unterschiede**
  - Breite der Register
  - Fähigkeit zum wahlweisen Aufwärts-/Abwärtszählen
  - …

Timer im STM32F401RE

Table 4. Timer feature comparison

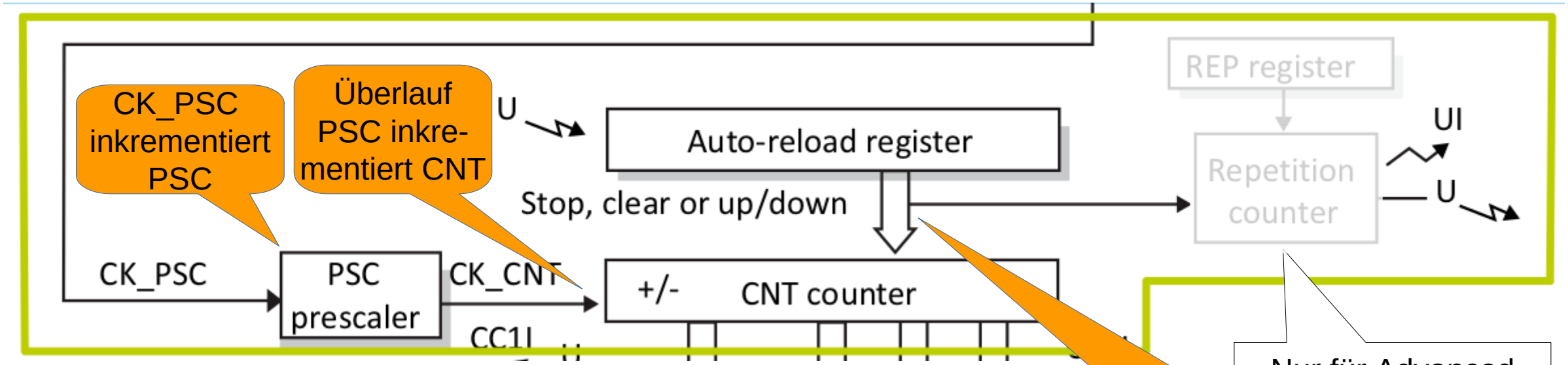
Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	84
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	84
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	84

Quelle: [F401-DS]

Timer im STM32F401RE

- **Unterlagen**
  - Alle Details zu den Timern einschließlich Registerbeschreibungen finden Sie in [F401-RM]
  - Die Dokumentation der Register ist zu umfangreich, um sie (wie bei den zuvor behandelten Peripherieeinheiten) in die Folien zu kopieren, sie werden nur referenziert
    - Die vollständige Beschreibung der General-purpose Timers TIM2 bis TIM5 [F401-RM] S. 316–375 gilt daher als Anhang zu den Folien und darf in der Klausur wie die Folien als Hilfsmittel verwendet werden.
- **Basisfunktionalität**
  - Up-/Downcounting Mode

Upcounting Mode



The update event period is calculated as follows:

Update_event = CK_PSC /((PSC + 1)*(ARR + 1)*(RCR + 1))

Where: CK_PSC = timer clock input

PSC = 16-bit prescaler register

ARR = 16/32-bit Autoreload register

RCR = 16-bit repetition counter

Beispiel:

CK_PSC = 72 MHz

Prescaler = 1

Auto reload = 65535

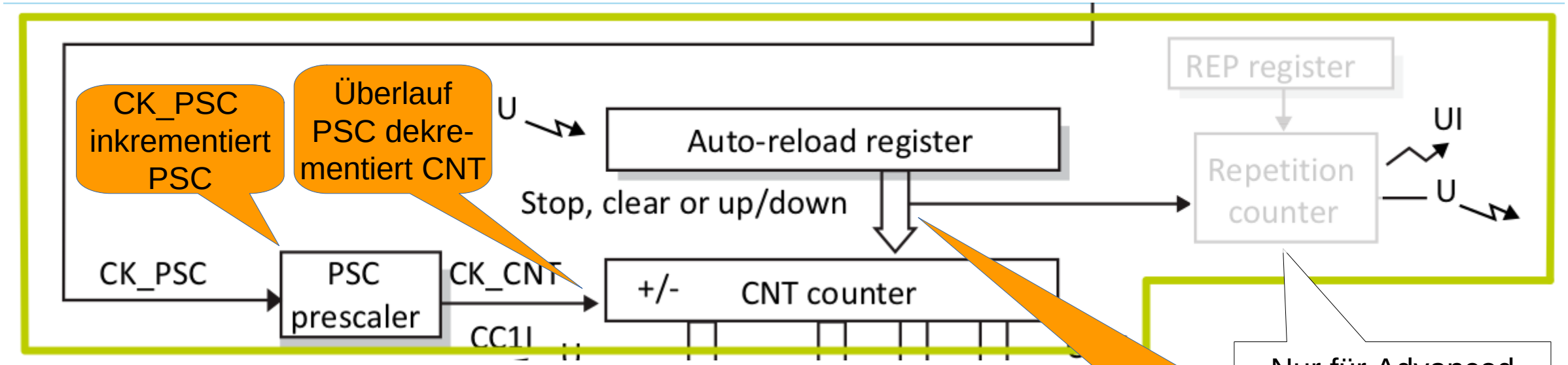
No repetition counter RCR = 0

Update_event = 72*(10^6)/((1 + 1)*(65535 + 1)*(1))

Update_event = 549.3 Hz

Erreichen von Wert in ARR lädt CNT als nächstes mit 0 und löst Update-Event aus

Downcounting Mode



The update event period is calculated as follows:

Update_event = CK_PSC /((PSC + 1)*(ARR + 1)*(RCR + 1))

Where: CK_PSC = timer clock input

PSC = 16-bit prescaler register

ARR = 16/32-bit Autoreload register

RCR = 16-bit repetition counter

Beispiel:

CK_PSC = 72 MHz

Prescaler = 1

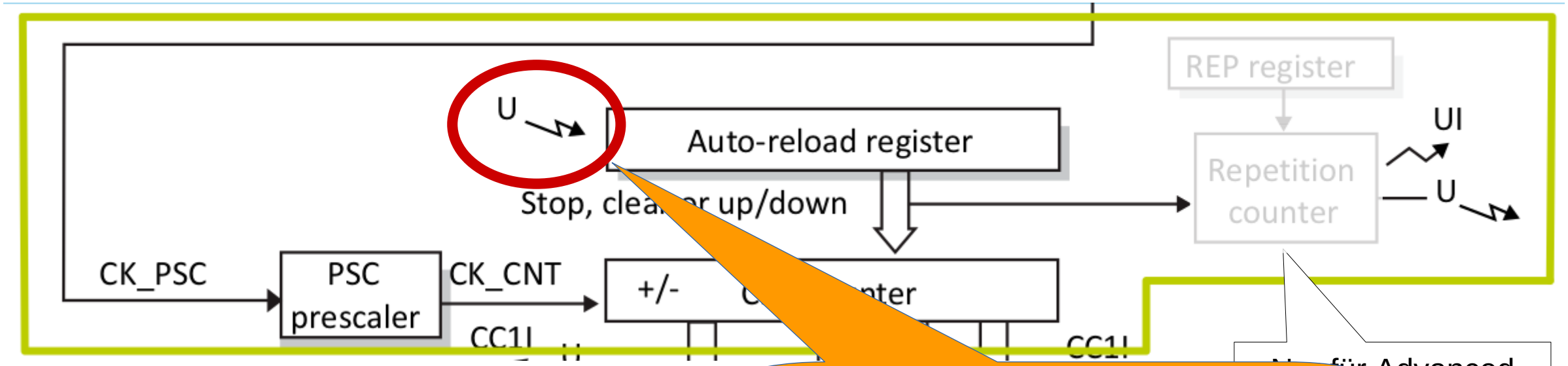
Auto reload = 65535

No repetition counter RCR = 0

Update_event = 72*(10^6)/((1 + 1)*(65535 + 1)*(1))

Update_event = 549.3 Hz

Downcounting Mode



The update event period is calculated as follows:

Update_event = CK_PSC /((PSC + 1)*(ARR + 1)*(RCR + 1))

Where: CK_PSC = timer clock input

PSC = 16-bit prescaler register

ARR = 16/32-bit Autoreload register

RCR = 16-bit repetition counter

Das Auto-reload Register (und andere) sind gepuffert. D. h. neue Werte werden nicht sofort in das Register übernommen sondern erst nach dem nächsten Update-Event.

für Advanced ementiert

No repetition counter RCR = 0

Update_event = 72*(10^6)/((1 + 1)*(65535 + 1)*(1))

Update_event = 549.3 Hz

Up-/Downcounting Mode – Register

13.4.11 TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

13.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 **CEN**: Counter enable

0: Counter disabled
1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

13.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.
Refer to the *Section 13.3.1: Time-base unit* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

13.4.6 TIMx event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

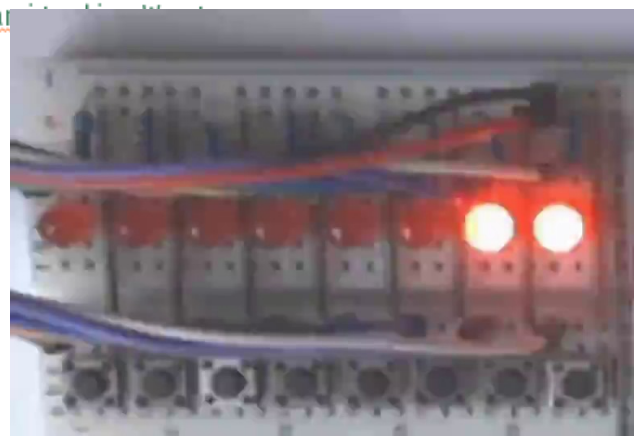
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.
0: No action
1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

Up-/Downcounting Mode – Anzeige

```
16 /**
17  * Beispielprogramm für die Verwendung eines General-purpose Timers
18  * im Aufwärts(oder Abwärts)zähler-Modus. Der Prescaler teilt den
19  * Systemtakt (84 MHz) durch 41016. Damit wird das Zählregister mit
20  * ca. 2048 Hz angesteuert. Die oberen 8-Bit des Zählers werden auf den
21  * LEDs ausgegeben, d.h. es ist ein Zählen mit ca. 8 Hz sichtbar. (Zur
22  * Kontrolle: Led 2 blinkt mit ca. 1 Hz.)
23  *
24  * Das ARR hat den default Wert 0xffff, angezeigt werden dan
25  * 0x00 bis 0xff (bzw. 0xff bis 0x00 beim Abwärtszählen).
26  */
27 void task1() {
28     // Timer mit Takt versorgen
29     SET_BIT(RCC->APB1ENR, RCC_APB1ENR_TIM3EN);
30
31     // Prescaler auf angegebenen Wert setzen
32     TIM3->PSC = 41016 - 1;
33
34     // Optional: Zählrichtung auf Downcounter umstellen
35     // SET_BIT(TIM3->CR1, TIM_CR1_DIR);
36
37     // Update-Event zum Aktualisieren der gepufferten Register erzwingen
38     // und Timer starten
39     SET_BIT(TIM3->EGR, TIM_EGR_UG);
40     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
41
42     while (true) {
43         leds = TIM3->CNT >> 8;
44     }
45 }
```



Live Coding

Up-/Downcounting Mode

- **Typische Nutzung**
  - Erzeugen eines Ereignis mit bestimmter Periode
  - Nur der „Überlauf“ (Neuladen des CNT-Registers) ist interessant
  - Timer setzt beim Neuladen das Update Interrupt Flag im Statusregister
    - Kann mit Polling abgefragt werden
    - Kann einen Interrupt auslösen

Up-/Downcounting Mode – Statusregister

13.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

If channel CC1 is configured as output:
This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.
0: No match
1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

If channel CC1 is configured as input:
This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.
0: No input capture occurred
1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

- This bit is set by hardware on an update event. It is cleared by software.
  0: No update occurred.
  1: Update interrupt pending. This bit is set by hardware when the registers are updated:
- At overflow or underflow (for TIM2 to TIM5) and if UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

Upcounting mit UIF-Polling

```
47 /**
48  * Beispielprogramm für die Verwendung eines General-purpose Timers im
49  * Polling-Modus. Der Prescaler teilt den Systemtakt (84 MHz) durch 41016,
50  * so dass das Zählregister mit ca. 2048 Hz getaktet wird. Dem ARR wird
51  * 2048-1 zugewiesen, so dass letztlich einmal pro Sekunde das UIF gesetzt
52  * wird. Darauf wird in einer while-Schleife gewartet und die LED getoggelt.
53  */
54 void task2() {
55     // Timer mit Takt versorgen.
56     SET_BIT(RCC->APB1ENR, RCC_APB1ENR_TIM3EN);
57
58     // Periode mit Hilfe von Prescaler und Reload-Register einstellen.
59     TIM3->PSC = 41016 - 1;
60     TIM3->ARR = 2048 - 1;
61
62     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
63     // das dadurch im SR gesetztes UIF löschen und Timer starten.
64     SET_BIT(TIM3->EGR, TIM_EGR_UG);
65     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
66     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
67
68     while (true) {
69         // Warten, bis Update-Flag gesetzt ist (ARR neu geladen)
70         while(!READ_BIT(TIM3->SR, TIM_SR_UIF)) {
71         }
72
73         // Update-Flag wieder zurücksetzen und LED toggeln.
74         TIM3->SR = 0;
75         leds[0] = leds[0] ^ 1;
76     }
77 }
```

Live Coding

C/C++ Wissen – Function Pointer

- **Interrupt Handler für Beispielprojektstruktur mit „Tasks“**
  - Nicht alle Tasks (verschiedene Aufgaben) können den gleichen Interrupt Handler verwenden
  - „Umschalten“ des im Interrupt Handler ausgeführten Codes muss möglich sein
  - Verschiedene Möglichkeiten denkbar
    - `#ifdef/#define`
    - `if` mit globaler Variable
    - **Function-Pointer**

C/C++ Wissen – Function Pointer

- **Pointer auf Variablen sind bekannt**
- **Pointer können auch auf Funktionen zeigen**
  - Definition syntaktisch wie eine Funktionsdeklaration bei der der Name durch „(*Zeigername)“ ersetzt ist
  - Beispiel:
    - Funktion: `void f()`
    - Zeiger auf Funktion mit Rückgabetyp `void` und leerer Parameterliste:

      ```
      void (*myFunctionPointer)()
      ```
    - Zeiger auf Funktion zeigen lassen:

      ```
      myFunctionPointer = f;
      ```
    - Aufruf:

      ```
      (*myFunctionPointer)();
      ```

C/C++ Wissen – Function Pointer

- **Anwendung für Timer-Beispiele**

```
/**
 * Zeiger auf die Funktion, die der Timer 3 IRQ Handler aufrufen soll.
 * Ermöglicht die Konfiguration unterschiedlicher IRQ Handler
 * für verschiedene Tasks.
 *
 * Der Wert des Funktionszeigers muss zu Beginn der Task gesetzt werden,
 * die einen Timer 3 Interrupt aktiviert.
 */
static void (*activeTIM3_IRQHandler)();

/**
 * Interrupt-Handler für Timer 3. Damit in den unterschiedlichen Beispielen
 * (tasks) unterschiedliche Interrupt-Handler aufgerufen werden können,
 * führt dieser Handler die Funktion aus, auf die tim3IrqHandler zeigt.
 */
extern "C" void TIM3_IRQHandler() {
    (*activeTIM3_IRQHandler)();
}
```

Upcounting mit Update Event Interrupt

```
109 /**
110  * Beispielprogramm für die Verwendung eines General-purpose Timers im
111  * Interrupt-Modus. Der Prescaler teilt den Systemtakt (84 MHz) durch 41016,
112  * so dass das Zählregister mit ca. 2048 Hz getaktet wird. Dem ARR wird
113  * 2048-1 zugewiesen, so dass letztlich einmal pro Sekunde der Update-
114  * Interrupt ausgelöst wird. In der zugehörien ISR wird das UIF
115  * zurückgesetzt und die LED getoggelt.
116  */
117 void task3() {
118     // Task-spezifischen IRQ Handler einstellen.
119     activeTIM3_IRQHandler = task3_TIM3_IRQHandler;
120
121     // Timer mit Takt versorgen
122     SET_BIT(RCC->APB1ENR, RCC_APB1ENR_TIM3EN);
123
124     // Periode mit Hilfe von Prescaler und Reload-Register einstellen.
125     TIM3->PSC = 41016 - 1;
126     TIM3->ARR = 2048 - 1;
127
128     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
129     // das dadurch im SR gesetztes UIF löschen.
130     SET_BIT(TIM3->EGR, TIM_EGR_UG);
131     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
132
133     // Interrupt-getrieben: Interrupts einschalten ...
134     __NVIC_EnableIRQ(TIM3_IRQn); // ... TIM3-Interrupts allgemein und ...
135     SET_BIT(TIM3->DIER, TIM_DIER_UIE); // ... speziell den Update-Interrupt
136
137     // Timer einschalten
138     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
139
140     while (true) {
141     }
142 }
```

```
/**
 * Interrupt-Handler für Timer 3. Togglet die LED 0, wenn das Update
 * Flag gesetzt ist (und setzt es zurück).
 */
extern "C" void TIM3_IRQHandler() {
    // Update-Interrupt-Flag wieder zurücksetzen und LED toggeln.
    // (// Makro, Alternative zu "TIM3->SR &= ~TIM_SR_UIF;")
    CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
    leds[0] = leds[0] ^ 1;
}
```

Live Coding

Upcounting mit Update Event Interrupt

13.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

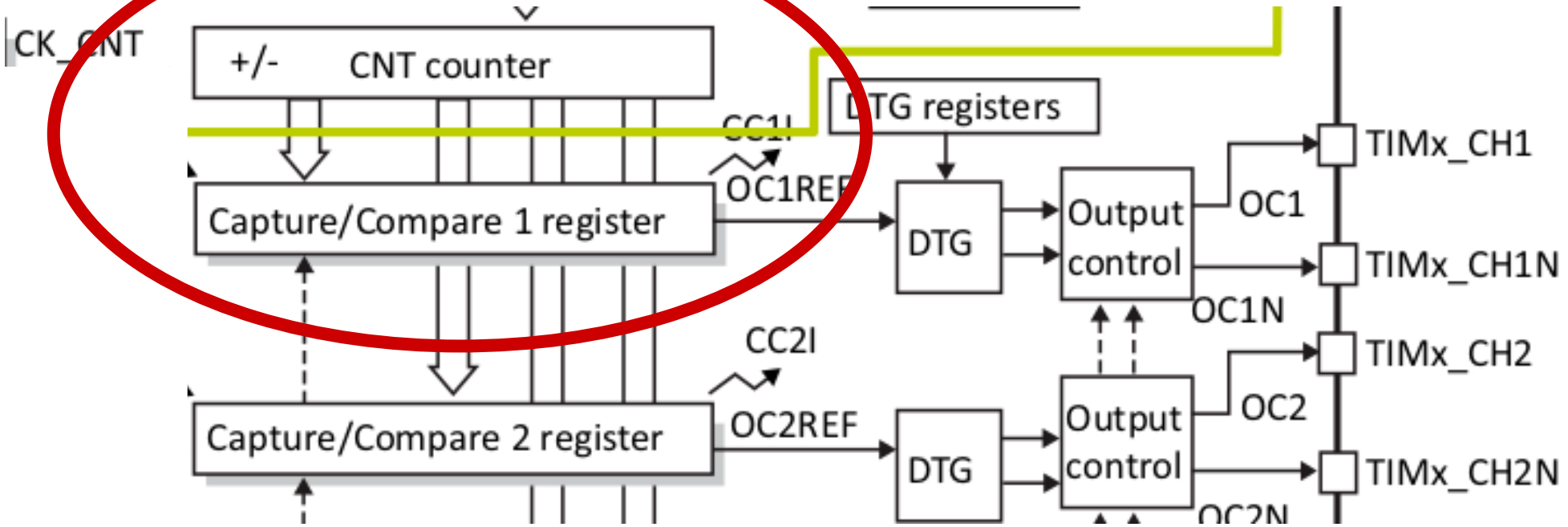
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable
0: CC1 interrupt disabled
1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled
1: Update interrupt enabled

Zähler/Timer-Erweiterung

- **Die Timer verfügen zusätzlich über 1-4 Capture/Compare-Register**
  - Wir betrachten nur die Compare-Funktionalität

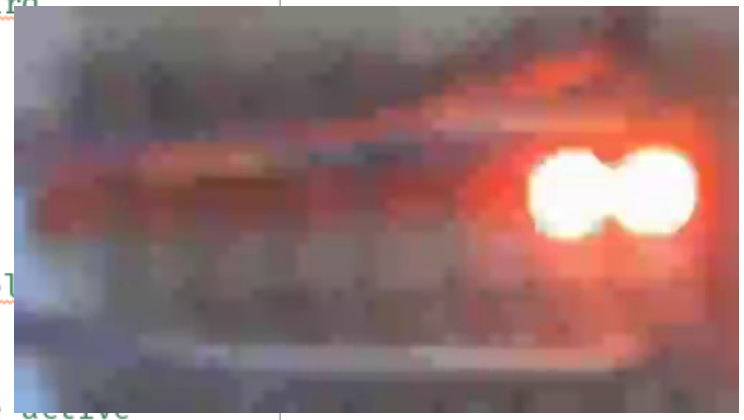


(Capture) Compare Register

- **Nutzung durch die Software**
  - Flag im SR wird gesetzt (CCxIF) wenn CNT == CCRx

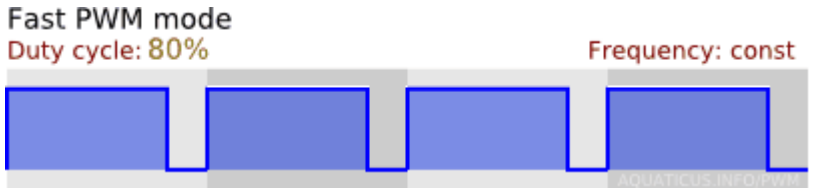
Compare Register – Abfrage CC1IF

```
144 /**
145  * Beispielprogramm für CC1IF-Abfrage. CNT wird mit ca. 2048 Hz angesteuert.
146  * Wenn der Wert 10 << 11 ereicht wird (also nach 10 Sekunden) wird
147  * LED 7 angeschaltet.
148  */
149 void task4() {
150     // Timer mit Takt versorgen
151     SET_BIT(RCC->APB1ENR, RCC_APB1ENR_TIM3EN);
152
153     // Periode mit Hilfe von Prescaler und Reload-Register einstel
154     TIM3->PSC = 41016 - 1;
155
156     // Compare register 1 auf 10 * 2048 setzen und output compare
157     // konfigurieren.
158     TIM3->CCR1 = 10 << 11;
159
160     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
161     // das dadurch im SR gesetztes UIF löschen und Timer starten.
162     SET_BIT(TIM3->EGR, TIM_EGR_UG);
163     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
164     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
165
166     while (true) {
167         // Bits 15-10 von CNT auf LEDs 5-0 und CC1IF auf LED 7 anzeigen.
168         leds = ((TIM3->CNT >> 10) & 0x3f)
169                 | ((READ_BIT(TIM3->SR, TIM_SR_CC1IF) >> TIM_SR_CC1IF_Pos) << 7);
170     }
171 }
```



Compare Register – PWM

- **Häufige Aufgabenstellung: Erzeugung PWM-Signal**
  - Pulse Width Modulation – Pulsbreitenmodulation
  - Innerhalb einer konstanten Periode variiert das Verhältnis zwischen High- und Low-Pegel



Quelle: http://www.aquaticus.info/pwm-modes/

  - Mit „nachgeschaltetem Tiefpass“ sehr günstiger Digital-Analog-Wandler
    - Helligkeitssteuerung LED (Tiefpass ist das menschliche Auge)
    - Geschwindigkeitssteuerung Motor (Tiefpass durch mechanischen Aufbau)
    - …

Compare-Register – PWM mit Polling

```
173 /**
174  * Beispielprogramm für Software PWM mit CCR ("Slow Motion"). CNT wird mit
175  * ca. 2048 Hz angesteuert. UIF schaltet die LED7 an, CC1IF schaltet die LED7
176  * aus, d.h. solange der Wert in CNT kleiner dem Wert in CCR1 ist,
177  * leuchtet die LEDs 7.
178  */
179 void task5() {
180     // Timer mit Takt versorgen
181     RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
182
183     // Periode mit Hilfe von Prescaler und Reload-Register einstellen.
184     TIM3->PSC = 41016 - 1;
185     TIM3->ARR = 16 * 2048 - 1;
186
187     // Compare register 1 auf 10 * 2048 setzen
188     TIM3->CCR1 = 10 << 11;
189
190     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
191     // das dadurch im SR gesetztes UIF löschen und Timer starten.
192     SET_BIT(TIM3->EGR, TIM_EGR_UG);
193     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
194     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
195
196     bool isActive = true;
197     while (true) {
198         // Bits 15-10 von CNT auf LEDs 5-0 und isActive auf LED 7 anzeigen.
199         if(READ_BIT(TIM3->SR, TIM_SR_CC1IF)) {
200             isActive = false;
201             CLEAR_BIT(TIM3->SR, TIM_SR_CC1IF);
202         }
203         if(READ_BIT(TIM3->SR, TIM_SR_UIF)) {
204             isActive = true;
205             CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
206         }
207         leds = ((TIM3->CNT >> 10) & 0x3f) | (isActive << 7);
208     }
209 }
```

Compare-Register – PWM mit Interrupt

```
227 /**
228  * Beispielprogramm für Software PWM mit CCR ("Slow Motion"). Wie task5,
229  * aber das Toggeln der LED wird über den Interrupt gesteuert.
230  */
231 void task6() {
232     // Task-spezifischen IRQ Handler einstellen.
233     activeTIM3_IRQHandler = task6_TIM3_IRQHandler;
234
235     // Timer mit Takt versorgen
236     RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
237
238     // Periode mit Hilfe von Prescaler und Reload-Register einstellen.
239     TIM3->PSC = 41016 - 1;
240     TIM3->ARR = 16 * 2048 - 1;
241
242     // Compare register 1 auf 10 * 2048 setzen
243     TIM3->CCR1 = 10 << 11;
244
245     // Interrupt-getrieben: Interrupts einschalten ...
246     __NVIC_EnableIRQ(TIM3_IRQn); // ... TIM3-Interrupts allgemein und ...
247     SET_BIT(TIM3->DIER, TIM_DIER_UIE); // ... speziell den Update-Interrupt
248     SET_BIT(TIM3->DIER, TIM_DIER_CC1IE); // ... und den CC1-Interrupt
249
250     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
251     // das dadurch im SR gesetztes UIF löschen und Timer starten.
252     SET_BIT(TIM3->EGR, TIM_EGR_UG);
253     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
254     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
255
256     while (true) {
257         __disable_irq();
258         leds = (leds & 0xc0) | ((TIM3->CNT >> 10) & 0x3f);
259         __enable_irq();
260     }
261 }
```

```
/**
 * Interrupt-Handler für Timer 3/task 6. Schaltet LED7 ein, wenn das
 * Update-Flag gesetzt ist, schaltet LED7 aus, wenn das CC1IF gesetzt
 * ist.
 */
void task6_TIM3_IRQHandler() {
    if(READ_BIT(TIM3->SR, TIM_SR_UIF)) {
        CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
        leds = leds | 0x80;
    }
    if(READ_BIT(TIM3->SR, TIM_SR_CC1IF)) {
        CLEAR_BIT(TIM3->SR, TIM_SR_CC1IF);
        leds = leds & ~0x80;
    }
}
```

Ergänzung zu Interrupts

- **Vorsicht wenn Ressourcen im Hauptprogramm und im Interrupt Handler genutzt werden**

```
leds = (leds & 0xc0) | ((TIM3->CNT >> 10) & 0x3f);
```

Im Hauptprogramm wird der berechnete Wert leds zugewiesen. Da der Wert mit noch nicht gesetztem Bit 7 berechnet wurde, wird die Änderung durch die ISR „sofort“ wieder überschrieben

Wert von leds wird in Register geladen, aber neuer Wert noch nicht geschrieben

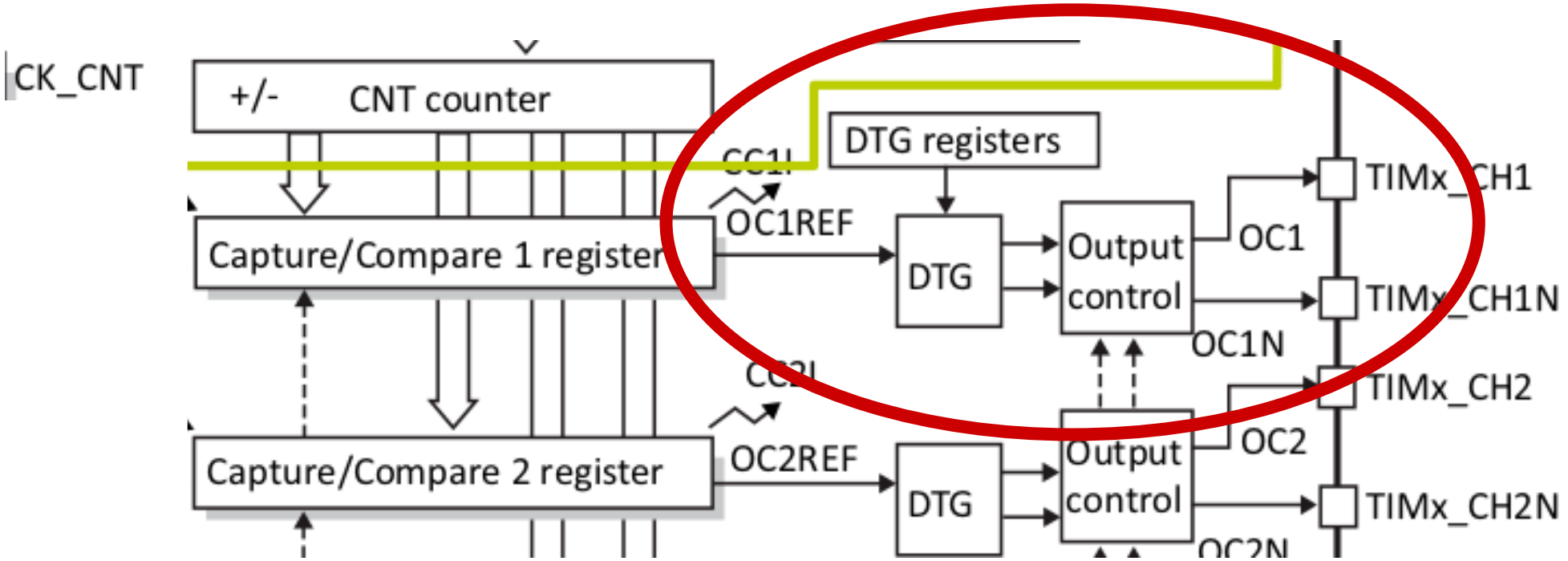
```
leds = leds | 0x80;
```

Interrupt unterbricht und Handler setzt Bit 7 in leds

Lösung: Interrupts im Hauptprogramm temporär ausschalten

Output Compare – Signal auf GPIO-Pin

- **Der Vergleich von CNT und CCR kann auch (praktisch verzögerungsfrei) zur Generierung von Signalen durch die Hardware verwendet werden**

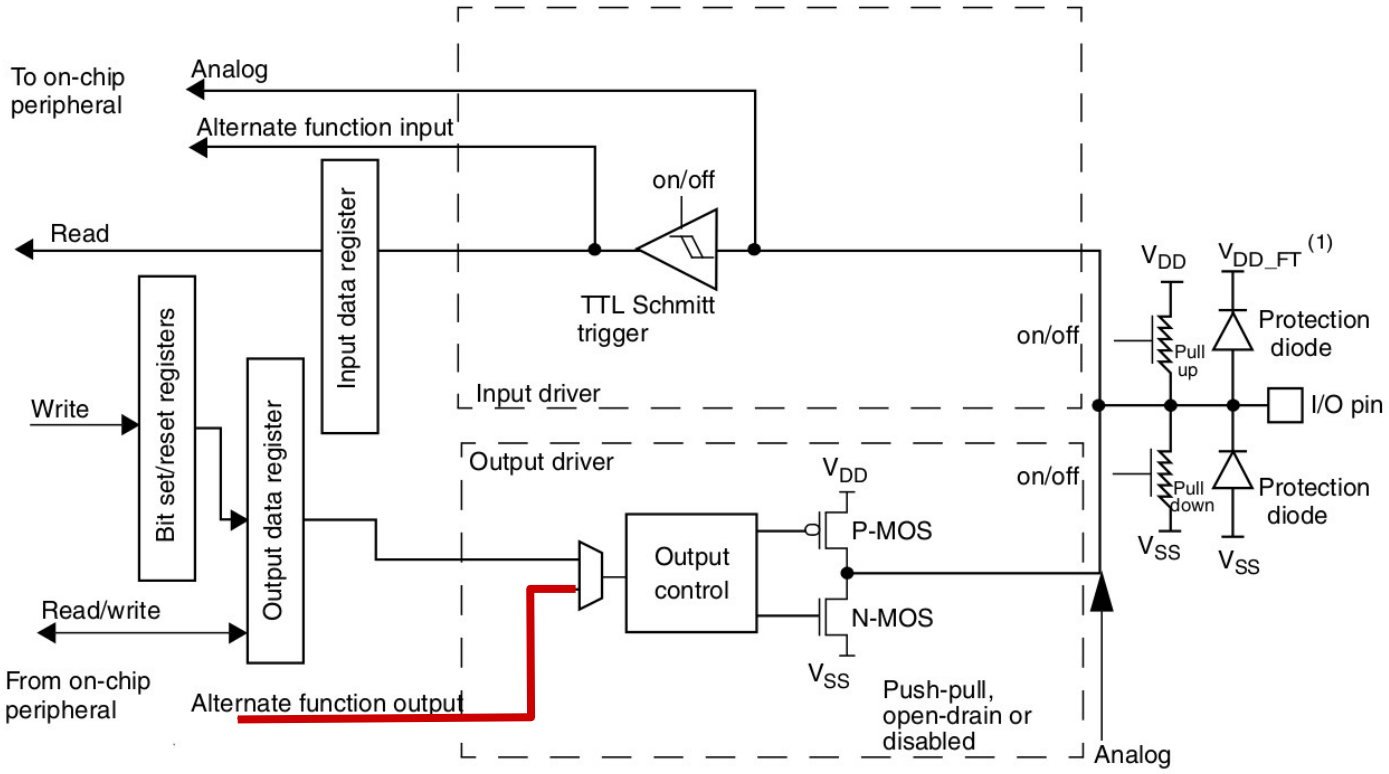


Output Compare – Generiertes Signal

- **Verschiedene Modi**
  - Output compare timing
    - Keine Generierung (bislang „genutzt“)
  - Output compare active
    - OCxREF für Kanal wird gesetzt, wenn CNT und CCR übereinstimmen
  - Output compare inactive
    - OCxREF für Kanal wird zurück gesetzt, wenn CNT und CCR übereinstimmen
  - Output compare toggle
    - OCxREF für Kanal ändert den Zustand, wenn CNT und CCR übereinstimmen
  - Output compare forced active/inactive
    - OCxREF wird zwangsweise auf 1 oder 0 gesetzt
  - Output compare PWM1 (s.u.)

Output Compare – Signal auf GPIO-Pin

- **Voraussetzung: Konfiguration des „Alternate Function“-Modus des GPIO-Pins**



Output Compare – Beispiel Active

Port		AF00	AF01	AF02
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5
	PC0	-	-	-
	PC1	-	-	-
	PC2	-	-	-
	PC3	-	-	-
	PC4	-	-	-
	PC5	-	-	-
	PC6	-	–	TIM3_CH1

S. [F410-DS] Table 9

```
/**
 * Beispielprogramm für OC1REF-Ausgabe. CNT wird mit ca. 2048 Hz angesteuert.
 * Wenn der Wert 10 << 11 ereicht wird (also nach 10 Sekunden) wird
 * LED6 angeschaltet.
 */
void task7() {
    // GPIO-Pin PC6 auf alternate function 2 konfigurieren.
    MODIFY_REG(GPIOC->MODER, GPIO_MODER_MODE6, 2 << GPIO_MODER_MODE6_Pos);
    MODIFY_REG(GPIOC->AFR[0], GPIO_AFRL_AFSEL6,
            GPIO_AF2_TIM3 << GPIO_AFRL_AFSEL6_Pos);

    // Timer mit Takt versorgen
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    // Prescaler auf berechneten Wert für ca. 2048 Hz output setzen.
    TIM3->PSC = 41016 - 1;

    // Compare register 1 auf 10 * 2048 setzen, output compare active
    // konfigurieren und Ausgabe des OC1REF einschalten.
    TIM3->CCR1 = 10 << 11;
    MODIFY_REG(TIM3->CCMR1, TIM_CCMR1_OC1M, 1 << TIM_CCMR1_OC1M_Pos);
    SET_BIT(TIM3->CCER, TIM_CCER_CC1E);

    // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
    // das dadurch im SR gesetztes UIF löschen und Timer starten.
    SET_BIT(TIM3->EGR, TIM_EGR_UG);
    CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
    SET_BIT(TIM3->CR1, TIM_CR1_CEN);

    while (true) {
        // Bits 15-10 von CNT auf LEDs 5-0 anzeigen.
        leds = (TIM3->CNT >> 10) & 0x3f;
    }
}
```

Live Coding

Output Compare – PWM1

```
298 /**
299  * Beispielprogramm für OC1REF-Ausgabe im PWM1-Modus. CNT wird mit ca. 2048 Hz
300  * angesteuert und zählt Modulo 16*2048. So lange der Wert von CNT >= CCR1
301  * (10 << 11) ist, bleibt LED6 angeschaltet.
302  */
303 void task8() {
304     // GPIO-Pin PC6 auf alternate function 2 konfigurieren.
305     MODIFY_REG(GPIOC->MODER, GPIO_MODER_MODE6, 2 << GPIO_MODER_MODE6_Pos);
306     MODIFY_REG(GPIOC->AFR[0], GPIO_AFRL_AFSEL6,
307             GPIO_AF2_TIM3 << GPIO_AFRL_AFSEL6_Pos);
308
309     // Timer mit Takt versorgen
310     RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
311
312     // Prescaler auf berechneten Wert für ca. 2048 Hz output setzen.
313     TIM3->PSC = 41016 - 1;
314     TIM3->ARR = 16 * 2048 - 1;
315
316     // Compare register 1 auf 10 * 2048 setzen, PWM1 konfigurieren
317     // und Ausgabe des OC1REF einschalten.
318     TIM3->CCR1 = 10 << 11;
319     MODIFY_REG(TIM3->CCMR1, TIM_CCMR1_OC1M, 6 << TIM_CCMR1_OC1M_Pos);
320     SET_BIT(TIM3->CCER, TIM_CCER_CC1E);
321
322     // Update-Event zum Aktualisieren der gepufferten Register erzwingen,
323     // das dadurch im SR gesetztes UIF löschen und Timer starten.
324     SET_BIT(TIM3->EGR, TIM_EGR_UG);
325     CLEAR_BIT(TIM3->SR, TIM_SR_UIF);
326     SET_BIT(TIM3->CR1, TIM_CR1_CEN);
327
328     while (true) {
329         // Bits 15-10 von CNT auf LEDs 5-0 anzeigen.
330         leds = (TIM3->CNT >> 10) & 0x3f;
331     }
332 }
```

Live Coding