Timer

Table of Contents

Begriffe

Timer/Counter

- Timer zählt interne Takte
- Counter zählt externe Takte (fallende/steigende Flanke ist wählbar)

Prescaler

- Teilerbaustein teilt den Takt herunter, damit die Timer nicht zu schnell zählen
- Teilungsfaktoren: 1024, 256, 64, 8, 1 usw.
- Auswahl des geteilten Taktes durch Multiplexer

Multiplexer

- Auswahlbaustein
- für die Auswahl 1-aus-8 werden 3 Steuerbits benötigt CS02, CS01, CS00 (clock select flags)
- die Auswahl 0 wählt den Takt 0Hz d.h. der Timer bleibt stehen

Output Compare Units

- vergleicht den Zählerstand mit einem Register und setzt eine Interrupt-Anfrage und/oder toggelt einen Pin

Double Buffered Register

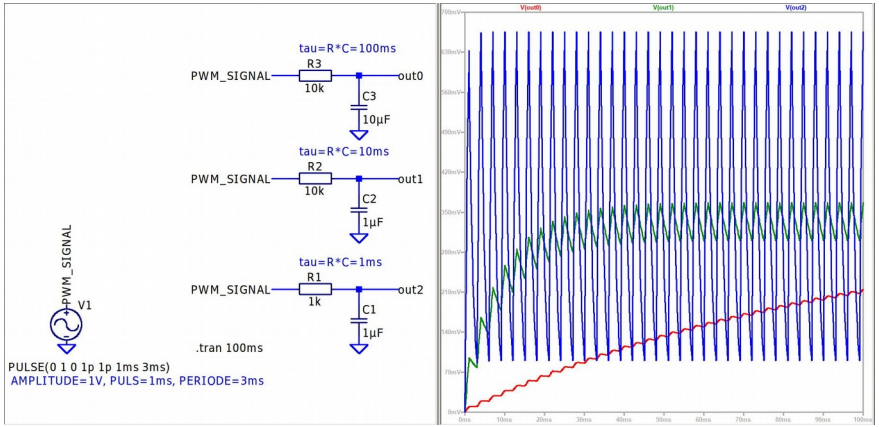
- der Inhalt wird nicht in das Original-Register sondern in ein Schattenregister geschrieben
- z.B. 16Bit Zähler soll gesetzt werden: zuerst die beiden Bytes getrennt in das Schattenregister schreiben und dann mit einem Schlag in das Zählregister übertragen

Clear Timer on Compare Match

- wenn der Zählerstand mit dem Vergleichsregister matcht wird der Zählerstand automatisch auf Null zurückgestellt

PWM (pulse width modulation)

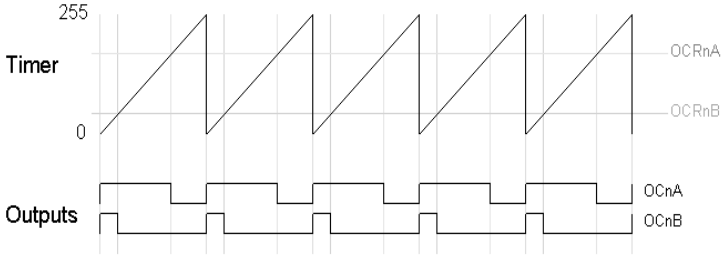
Mit PWM lassen sich analoge Spannungen erzeugen. Das Verhältinis von Pulsdauer zu Periodendauer bestimmt die erzeugte Gleichspannung. Die Gleichspannung ist der Mittelwert des Digitalsignals und kann durch ein einfaches RC-Glied erzeugt werden. Je geringer die Welligkeit, desto länger dauert es, bis sich der Gleichspannungswert einstellt.



- die Simulation zeigt: je höher die Zeitkonstante tau im Verhältinis zur Frequenz der PWM, desto geringer die Welligkeit des Gleichsspannungs-Signals, desto länger dauert es aber, bis sich die Gleichspannung einstellt

→ höhere Frequenz der PWM ist erwünscht, dies ergibt geringere Anforderungen an den Ausgangstiefpass.

Fast PWM (Asymmetric PWM)



- Einfaches pulsweitenmoduliertes Signal; hohe Frequenz
- bei den dabei entstehenden höheren Frequenzen werden die externen Komponenten (C,L,R) billiger
- beim Umschalten des Puls/Pausen-Verhältnisses entstehen Störungen (Glitches)
- geeignet für Leistungsregelung, Gleichrichtung, DAC Anwendungen;
- single Slope; Überlauf bei **TOP** (entweder 0xff oder OCR0A)
- Nicht-invertierender Output: bei BOTTOM OCx Pin= 1, bei MATCH wird OCx Pin=0;
- $f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$

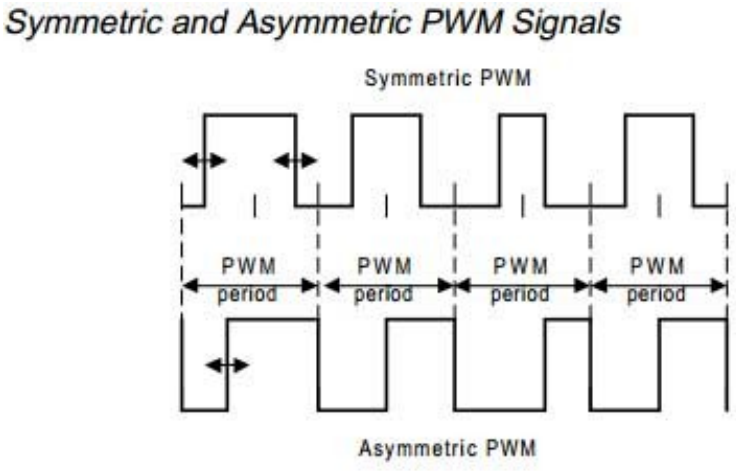
  Die Frequenz ergibt sich aus den Überläufen des Timers der mit $f_{osc}=f_{clk\_I/O}$ zählt und dem Prescaler N z.B: f_{osc}=16MHz → $f_{OCnxPWM}$=62,5kHz

Glitch Free, Phase Correct PWM (Symmetric PWM)

- Symmetrisch zur Mitte der Periode

man sieht: symmetrisch/asymmetrisch bezogen auf die Mitte der Periode.

Beim Umschalten auf ein neues Puls/Pause-Verhältnis erzeugt die symmetrische Variante weniger Oberwellen (Störungen, Glitches) weil der Abstand der Pulse konstant bleibt, bei der asymmetrischen PWM ändert sich der Abstand der Pulse, das macht sich als Störung im Analogsignal bemerkbar.



8-Bit Timer

Definition

BOTTOM: Zählerstand 0

MAX: Zählerstand 0xFF

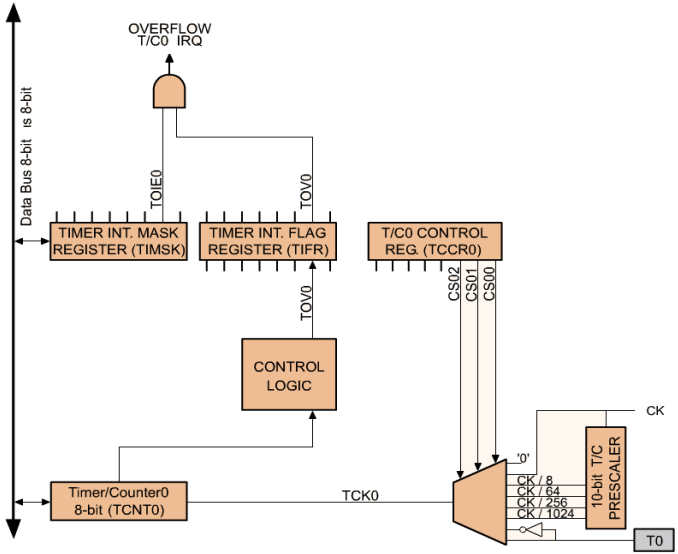
TOP: höchster Zählerstand; normalerweise TOP=MAX,
aber es kann auch TOP=OCR0A eingestellt werden.

Betriebsarten

- Normaler Zählerbetrieb
- CTC (clear timer on compare)
- PWM mode

Register

- Zählerstand: TCNT
- Kontrolle: TCCR
- Interrupt-Request: TIFR
- Interrupt-Enable: TIMSK
- Output Compare: OCR



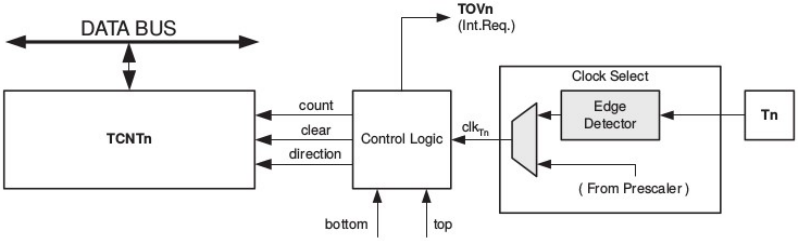
Mögliche Interrupt Flags:

- Overflow TOV0
- Compare Matched OCnA, OCnB

Signale an den Zähler:

- Clear
- Direction up/down
- Clock: von Prescaler oder externem Pin

Figure 14-2. Counter Unit Block Diagram



Der Zähler zählt bis 255 und läuft dann auf 0 über. Dabei setzt er ein Overflow-Flag und fordert einen Interrupt an. Wenn der Interrupt erlaubt ist wird die zugehörige ISR (Interrupt Service Routine) angesprungen.

Getaktet wird der Zähler (Timer1) extern oder intern und über die Flags TCCR1.CS12, TCCR1.CS11 und TCCR1.CS10 geteilt.

CS12	CS11	CS10	Clock Select
0	0	0	Timer/Counter stopped
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}$ / 8
0	1	1	$clk_{I/O}$ / 64
1	0	0	$clk_{I/O}$ / 256
1	0	1	$clk_{I/O}$ / 1024
1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
int main(void)
{
 TIMSK |= (1<<TOIE0);  // Timer 0 overflow erlauben
 TCCR0 = (1<<CS01); // Prescaler 8, Timer starten
 sei();

 while(1)
 {

 }
}

ISR (TIMER0_OVF_vect)
{
  // ein Overflow ist aufgetreten
}
```

Beispiel: erzeuge einen Takt von ca. 1ms

mit dem Prescaler von 64 ergeben sich alle 1,024ms ein Overflow

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
      TIMSK0 |= (1<<TOIE0);  // Timer 0 overflow erlauben
      TCCR0B = (1<<CS01)|(1<<CS00); // Prescaler 64
      sei();
      while(1)
      { volatile int dummy;
      }
}

ISR (TIMER0_OVF_vect)
{
      // ein Overflow ist aufgetreten
```

Beispiel: Toggle ca. alle 1ms den Pin PC0 (fosc=1MHz)

Tosc=1us → 1000 Zyklen → Prescaler = 8 → 1000/8= 125 Zyklen

T_{reload} = 256-125=131

In der Praxis ergibt sich ein Fehler von ca. 15us dadurch, dass der Einsprung in die Serviceroutine sowie die Abarbeitung der Serviceroutine jeweils ein paar Zyklen brauchen. (Reloadwert=130 passt besser)

```
#include <avr/io.h>
#include <avr/interrupt.h>
int main(void)
{
      TIMSK0 |= (1<<TOIE0);  // Timer 0 overflow erlauben
      TCCR0B = (1<<CS01); // Prescaler 8, Timer starten
      sei();
      while(1);
}

ISR (TIMER0_OVF_vect)
{
      TCNT0 = 131;
}
```

Output Compare Match / Waveform Generator

Begriffe:

- OCRn … Output Compare Value Register n (OCR0)
- OCFn … Request Flag Output Compare; wird automatisch gelöscht, wenn ISR exekutiert wird (oder softwaremäßig durch Schreiben einer 1)
- FOCn … Force Output Compare (löst keinen Interrupt aus, setzt auch den Timer nicht zurück, wirkt aber auf den Ocnx Pin)
- Ocnx … an diesem Pin wird das Timing erzeugt
- WGMn1:0 … Waveform Generator Mode Normal, PWM (Fast, Phase Correct)
- COMnx1:0 Waveform-Generator: definieren den Zustand von Ocnx beim nächsten Match und die Quelle für den OC0x Pin

Figure 14-3. Output Compare Unit, Block Diagram

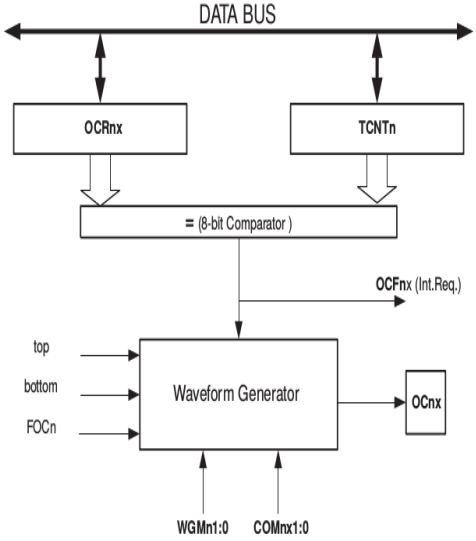


Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

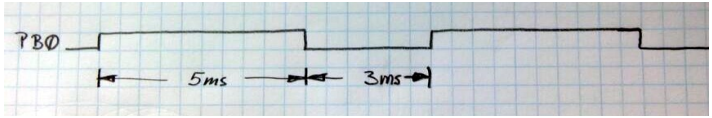
Double Buffering

Doppelte Pufferung: die CPU greift auf das Schattenregister von OCR0x.
Normalbetrieb: die CPU greift direkt auf OCR0x zu.

Doppelte Pufferung synchronisiert das Update der OCR0x Register zu TOP bzw. BOTTOM der Zählsequenz. Dies verhindert unsymmetrische PWM Signale => Glitch-Free Output d.h. wenn man das OCR0x Register ändert wird es nicht sofort gesetzt, sondern erst nachdem z.B TOP erreicht wurde.

Erzeugen von Timings

Am Pin PB0 soll folgendes Timing erzeugt werden:



Idee: der Timer wird mit einem Wert geladen, so dass er nach 5ms überläuft. Dies löst einen Interrupt aus. In der ISR wird der Wert auf 3ms umgestellt usw.

Taktrate des Oszillators: 16MHz T_{osc}=62.5ns: Überlauf beim Zählerstand 255 d.h. nach 256 Zyklen, also alle 256*62.5ns = 16µs. Viel zu schnell. Wenn man den Takt herunterteilt ergeben sich:

Teiler	Periode des Taktes	Überlauf nach 256 Zyklen
1	62.5ns	16µs
8	500ns	128µs
64	4µs	1.024ms
256	16µs	4.096ms
1024	64µs	16.384ms

Man sieht: für ein Timing von 5ms eignet sich bei 8Bit und 16MHz nur der Teiler 1024. Ein Zählschritt dauert dann 64µs. Die gewünschten 5ms entsprechen daher 78.125 Taktperioden. Das Timing ist also nicht genau erreichbar.

Der Zähler wird nicht bei Null, sondern so gestartet, dass er nach 5ms überläuft. Der Wert dafür (preload value) ist 256-78 = 178.

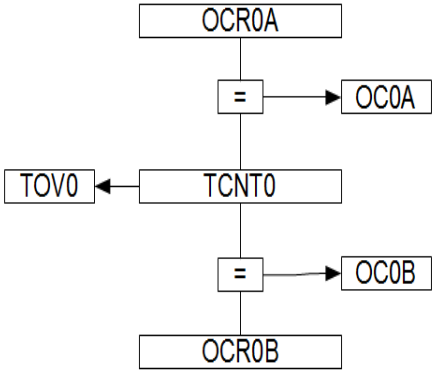
Nachteil dieser Methode: das Laden der Timer in der ISR benötigt zusätzliche Rechenzeit und ändert daher das Timing leicht.

```
#define F_CPU 16000000
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED 0
#define PRELOAD_5MS 0x100 - 78
#define PRELOAD_3MS 0x100 - 47     //Timer overflow after 3 ms @16MHz

void setup();
int main(void){
      setup();
      sei();
      while(1);
}
void setup(){
 // div 1024 ==> CS0=5
 TCCR0B |=_BV(CS02)|_BV(CS00);
 TIMSK0 |= _BV(TOIE0); //int enable
 DDRB |= _BV(LED);
}

ISR(TIMER0_OVF_vect){
 PORTB ^= _BV(LED); //Toggle Pin
 if (PORTB & _BV(LED))
```

```
  TCNT0 = PRELOAD_5MS;   //no need to stop counter (8bit operation)
 else
  TCNT0 = PRELOAD_3MS;
}
```

Auto Reload: CTC (clear timer on compare match)

1. Register OCR0A setzen
2. Wenn der Zählerstand den Wert OCR0A erreicht hat und überläuft, wird wahlweise ein Pin (OC0A) gesetzt (Null, Eins, Toggle) und/oder ein Interrupt ausgelöst
3. Die Pins OC0A (bzw. OC0B) müssen als OUTPUT konfiguriert werden.



Beispiel: alle 8192us soll der Ausgang OC0A toggeln: fosc=16MHz

```
#include <avr/io.h>
#include <avr/interrupt.h>

void setup(){
      DDRD=(1<<PORTD6);  //OC0A pin is output
      TCCR0A |= (1<<COM0A0);  // toggle OC0A Pin (PIND6)
      TCCR0A |= (1<<WGM01);   // CTC Mode
      TCCR0B=(1<<CS02) | (1<<CS00); // prescal 1024
      TCNT0=0x00;
      OCR0A=0x7F;  // nach T=62.5ns*1024*128 = 8192ms compare match
      // Timer/Counter 0 Interrupt(s) initialization
      TIMSK0 |= (1<<OCIE0A) | (1<<TOIE0);
}

int main(void){
      setup(); sei(); while (1);
}

ISR(TIMER0_OVF_vect) {
      // never reached
      volatile static int i; i++;
}

ISR(TIMER0_COMPA_vect){
      // count the compare matches
      volatile static int i; i++;
}
```

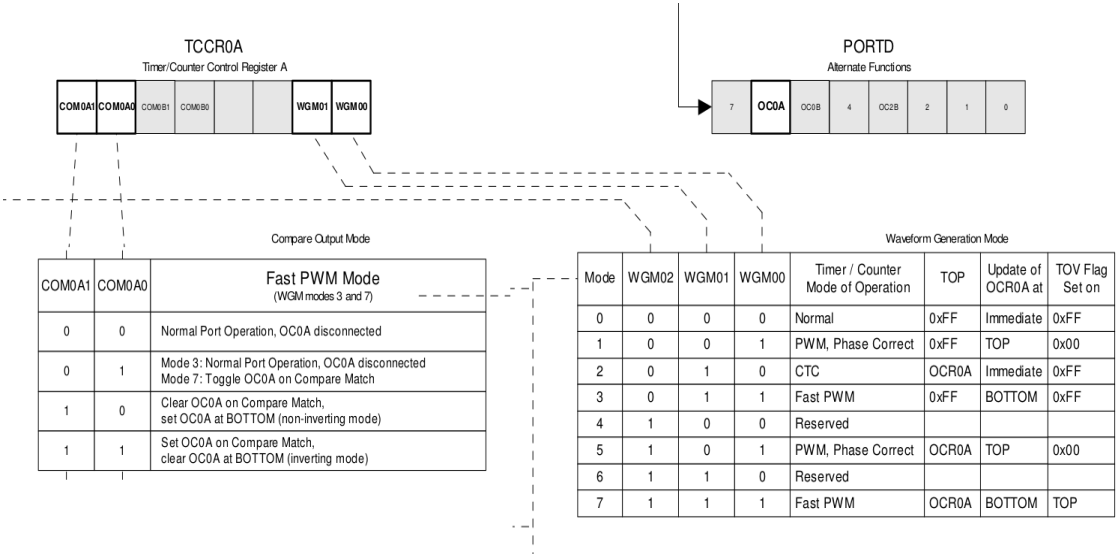
PWM

1. FAST PWM 3 (WGM01=WGM00=1 in Register TCCR0A)

2. Non-Inverting Mode (COM0A1=1 in Register TCCR0A)

3. Compare Register (OCR0A) als Schaltschwelle setzen

4 Timer starten (Bit CS00 in Register TCCR0B setzen)



Interrupt

1. erlauben (Bit OCIE0A in Register TIMSK0 setzen)

2. ISR(TIMER0_COMPA_vect){}

3. sei()

```
#include <avr/io.h>
#include <avr/interrupt.h>

void setup(){
      DDRD=(1<<PORTD6);  //OC0A pin is output
      TCCR0A |= (1<<WGM01)|(1<<WGM00);   // Fast PWM Mode
      TCCR0A |= (1<<COM0A1);  // noninverting PWM (PIND6)
      TCCR0B=(1<<CS00); // prescal 1
      OCR0A=256/2;  // 50% duty cycle
      // Timer/Counter 0 Interrupt(s) initialization
      TIMSK0 |= (1<<OCIE0A) | (1<<TOIE0);
}

int main(void){
      setup(); sei(); while (1);
}

ISR(TIMER0_OVF_vect) {
      // timer overflow interrupt service routine
}

ISR(TIMER0_COMPA_vect){
      // compare match
}
```

Timer1 16Bit

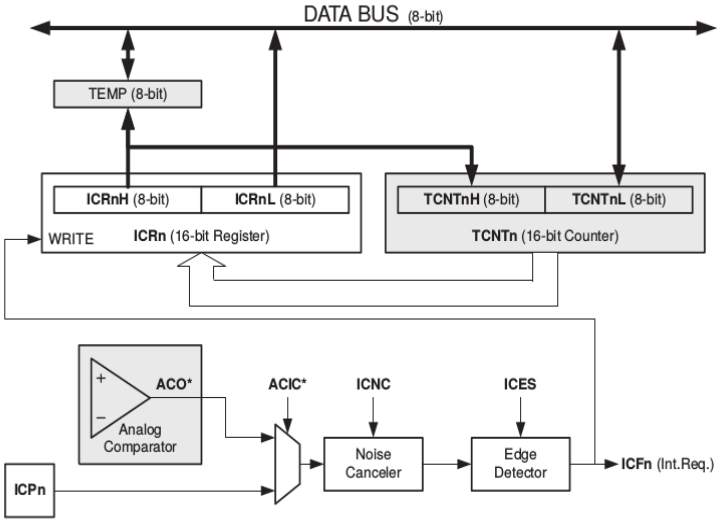
- Count und Compare Register sind 16 bit
- Lesen und Schreiben aber sind 8 Bit Operationen
  - High-Byte wird in Schattenregister zwischengespeichert
  - beim Schreiben/Lesen auf CNT Register wird das Schattenregister verwendet um z.B: beim Lesen den richtigen Zählerstand zwischenzuspeichern.
  - Schreiben: zuerst das High-Byte, dann das Low-Byte
  - Lesen: Zuerst das Low-Byte, dann das High-Byte
  - nur in Assembler wichtig; in C wird das durch den Compiler automatisch richtig gemacht ...

```
; Write TCNT1 (0x03FA)
ldi r17,0x03
ldi r16,0xFA
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H ...
```

Input Capture (Timer1 only)

- ICP Pin oder Analoger Komparator löst Capture des 16Bit Timers aus
- Ein Edge-Detektor erkennt steigende/fallende Flanke und löst aus
- das 16Bit Register ICR1 nimmt den Zählerstand auf
- ein Interrupt-Requestflag wird gesetzt und automatisch beim Einstieg in die ISR gelöscht

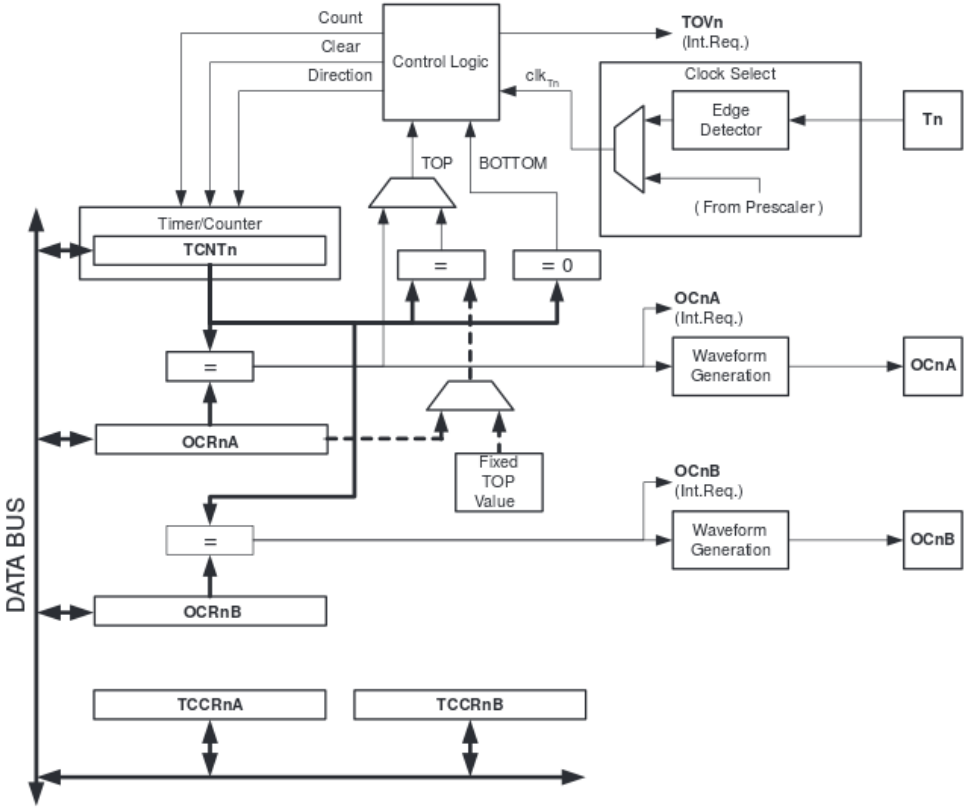
Figure 15-3. Input Capture Unit Block Diagram



- Noise Cancellation verzögert das Capture Signal um 4 Systemtaktzyklen (4x das gleiche Signal hintereinander bevor eine Änderung akzeptiert wird).

Timer Counter ATMEGA328p

Figure 14-1. 8-bit Timer/Counter Block Diagram



Qellen

http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Die_Timer_und_Z%C3%A4hler_des_AVR

Template für ATMEL Studio 6.2 Counter0.zip