

Embedded Systems

Prof. Dr. Christian Jakob

Hochschule Darmstadt
University of Applied Sciences
Fachbereich EIT

christian.jakob@h-da.de



Einführung

Inhaltlich baut die Kurseinheit „Embedded Systems“ auf dem Modul Hardwarenahe Programmierung des vierten Semesters auf. Die dort beschriebenen Konzepte und Inhalte werden im vorliegenden Lehrbrief weiter vertieft und diskutiert. Beide Lehrbriefe bieten somit eine vollständige Diskussion einer modernen Mikrocontroller-Architektur.

Der vorliegende Lehrbrief bildet die Basis dieser Kurseinheit und vermittelt praxisnah in aufeinander aufbauenden Lehreinheiten Schritt für Schritt die Grundlagen elementarer Bestandteile moderner eingebetteter Systeme. Stand im ersten Lehrbrief noch der Mikrocontroller selbst im Fokus, so liegt der Schwerpunkt in dieser Kurseinheit auch auf der Applikations-Implementierung.

Komplexe, theoretische Zusammenhänge werden in jedem Kapitel anhand zahlreicher Beispielprogramme näher erläutert. Umfangreiche Übungsaufgaben am Ende der einzelnen Abschnitte ermöglichen die unmittelbare Überprüfung und Vertiefung des gelernten Stoffes.

Eine zur Verfügung gestellte Mikrocontroller-Hardware dient im Rahmen dieser Kurseinheit als Plattform für die Erprobung der im Lehrbrief vorgestellten Beispielprogramme (*learning-by-doing*) sowie für die Umsetzung praktischer Übungsaufgaben. Eine Kontrolle des Lernerfolgs kann somit zusätzlich zu den eigentlichen Theoriefragen unmittelbar durch die Programmierung der eingesetzten Hardware-Plattform erfolgen.

Neben den in dieser Kurseinheit beschriebenen Projekten kann die eingesetzte Hardwareplattform für eine ganze Reihe weiterer Aufgaben eingesetzt werden. Alles was Sie benötigen ist ein gewöhnlicher PC, die Bereitschaft zu programmieren, sowie Ideen und Vorstellungskraft. Ich hoffe, dass Sie aufbauend auf dem Studium dieser Kurseinheit nicht nur viele spannende Projekte umsetzen, sondern auch eine Menge Spaß haben werden.

Für Kritik, Anregungen und Verbesserungsvorschläge bin ich jederzeit dankbar.

Prof. Dr. Christian Jakob, Darmstadt, September 2015

Lernziele und Kompetenzen

Das Modul „Embedded Systems“ soll den Studierenden fortgeschrittene Kenntnisse über den Aufbau, die Funktionsweise sowie den Entwurf einfacher Mikrocontroller-Systeme vermitteln. Neben einem guten Verständnis der grundlegenden Mikrocontroller-Hardware und -Peripherie lernen die Studierenden das Zusammenspiel zwischen Hard- und Software kennen und werden in die Lage versetzt anforderungsgerechte Software für eine gegebene Zielplattform zu entwickeln.

Nach dem Studium dieser Kurseinheit sollten Sie

- eine grundsätzliche Vorstellung über die Struktur und Funktionsweise von μ C-internen Komparator-Architektur sowie deren Einsatz zur Charakterisierung externer Signale,
- die kennen die grundsätzliche Arbeitsweise und Charakteristik des μ C-internen EEPROM-Speichers und können diesen im Rahmen gegebener Applikationen konfigurieren und programmieren,
- mit der Konfigurierung und Programmierung der μ C-internen Timer-Einheiten vertraut sein,
- das Prinzip der Timer/PWM basierten Signalerzeugung kennen und dieses mittels der ATmega328p internen Timer-Peripherie im Rahmen gegebener Applikationsszenarien umsetzen können.

Die Lernziele werden zu Beginn jedes einzelnen Kapitels weiter konkretisiert.

Inhaltsverzeichnis

1. Analog Komparator	7
1.1 Einführung	8
1.2 ATmega328p - Analog-Komparator - Steuerung und Konfiguration	14
1.3 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #1	17
1.4 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #2	21
1.5 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #3	27
1.5 Übungsaufgaben	34
2. EEPROM Datenspeicher	39
2.1 Einführung	40
2.2 ATmega328p - EEPROM Speicher - Steuerung und Konfiguration	43
2.3 ATmega328p - EEPROM Speicher - Praxisbeispiele - Teil #1	47
2.4 ATmega328p - EEPROM Speicher - Praxisbeispiele - Teil #2	51
2.5 Übungsaufgaben	54
3. Timer	57
3.1 Einführung	58
3.2 ATmega328p - Timer/Counter 0 - Steuerung und Konfiguration	65
3.3 ATmega328p - Timer/Counter 1 - Steuerung und Konfiguration	72
3.4 ATmega328p – Timer/Counter – Praxisbeispiele	79
3.4.1 ATmega328p - Timer T/C0 Zähler Demo - Polling Modus	79
3.4.2 ATmega328p - Timer T/C0 Zähler Demo - Interrupt Modus	80
3.4.3 ATmega328p - Timer T/C0 - Overflow Detection - Polling Modus	81
3.4.4 ATmega328p - Timer T/C0 - Overflow Detection - Interrupt Modus	82
3.4.5 ATmega328p - Timer T/C0 - Overflow Interrupt mit Timer-Reload	83
3.4.6 ATmega328p - Timer T/C0 - CTC - Polling Modus	84
3.4.8 ATmega328p - Timer T/C0 - CTC - Interrupt Modus	85
3.4.9 ATmega328p - Timer T/C1 - CTC - Interrupt Modus	86
3.4.10 ATmega328p - Timer T/C0 - nicht inv. Fast PWM Modus #1	87
3.4.11 ATmega328p - Timer T/C0 - nicht inv. Fast PWM Modus #2	88
3.5 ATmega328p – Timer/Counter – Weiterführende Beispiele	89
3.6 Übungsaufgaben	97

1. Analog Komparator

Neben dem Analog-Digital-Wandler verfügt der Atmel ATmega328p über eine vollintegrierte analoge Komparator-Stufe. Dieses Peripheriemodul ist zentrales Thema des folgenden Kapitels.

Schaltungstechnisch handelt es sich bei einer analogen Komparator-Einheit um einen Differenzverstärker der ein analoges Eingangssignal mit einer Referenzspannung vergleicht (*engl. to compare: vergleichen*). Das Ausgangssignal des Komparators ist binär und wird gesetzt, wenn die zu messende Spannung den Referenzwert überschreitet. Unterschreitet das Messsignal hingegen den Referenzwert, so wird der Ausgang zurückgesetzt. Anwendung finden μC interne Analog-Komparatoren überall da, wo es darum geht, die Ausgangsspannung einer Quelle mit einer vorgegebenen Referenz zu vergleichen und gegebenenfalls in Abhängigkeit des Resultats bestimmte Abläufe zu initiieren. So könnte beispielsweise ein lichtempfindlicher Fotowiderstand benutzt werden, um beim Unterschreiten eines gewissen Helligkeitswertes den Komparator auszulösen, wie es z.B. bei Dämmerungsschaltern der Fall ist. In Kombination mit der internen Timer Peripherie des Mikrocontrollers eignet sich der Analog-Komparator zudem zur Messung der Zeitdifferenz zweier externer Impulse und somit zur Bestimmung der Frequenz oder dem Tastverhältnis eines extern anliegenden Signals. Beide Applikationsszenarien werden im nachfolgenden Kapitel vorgestellt und ausführlich diskutiert.

Nach einer allgemeinen Einführung in die Thematik wird ausführlich auf den Aufbau, die Konfiguration sowie die Programmierung der Atmega328p internen Komparator-Peripherie eingegangen. Abschließend werden die zuvor diskutierten Konzepte anhand konkreter Beispiele anschaulich aufgezeigt.

Nach dem Studium dieses Kapitels und dem Bearbeiten der Übungsaufgaben sollten Sie

- die grundlegende Funktion sowie den Aufbau der Atmel ATmega328p internen Analog-Komparator-Stufe detailliert beschreiben können,
- in der Lage sein, den Atmel ATmega328p internen Analog-Komparator gemäß gegebener Vorgaben zu konfigurieren und in Betrieb zu nehmen,
- in der Lage sein, den ATmega328p internen Analog-Komparator sowohl im Polling- als auch im Interrupt-Modus im Rahmen vorgegebener Anwendungsbeispiele zu programmieren,
- die grundlegende Funktion sowie den Aufbau der Atmel ATmega328p internen Input-Capture-Stufe detailliert beschreiben können,
- den Komparator in Kombinationen mit μC internen Timer-Peripherie zur Frequenzmessung eines externen Wechselsignals einsetzen können.

1.1 Einführung

Neben dem im Lehrbrief „Hardwarenahe Programmierung“ beschriebenen 10-bit SAR Analog-Digital-Wandler verfügt der Atmel ATmega328p Mikrocontroller über eine vollintegrierte Komparator-Stufe zum Vergleich zweier analoger Spannungswerte. Der nichtinvertierende Eingang (+) des Komparators ist hier intern mit dem Pin AIN0 (ATmega328p – Pin PD6) des Mikrocontrollers verbunden. Der invertierende Eingang (-) ist in der Standardkonfiguration mit dem externen Pin AIN1 (ATmega328p – Pin PD7) verschaltet. Die prinzipielle Funktionsweise dieses Peripheriemoduls ist in Abbildung 1.1 dargestellt. Die Spannung am nichtinvertierenden Eingang dient hier als Referenzspannung (Sollwert), die mit einer Vergleichsspannung (Istwert) am invertierenden Eingang kontinuierlich verglichen wird.

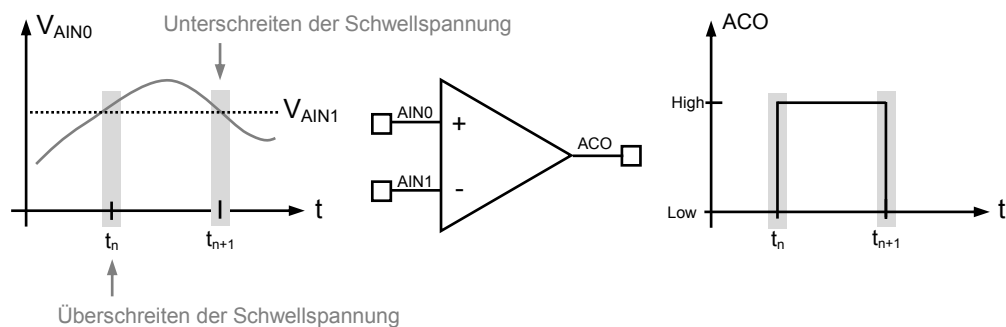


Abbildung 1-1 Verlauf der Ausgangsspannung eines analogen Komparators bei gegebener Referenzspannung und zeitlich variierender Vergleichsspannung

In Abhängigkeit davon, welcher der beiden Eingänge ein höheres Potential aufweist, wird das binäre Ausgangssignal des Komparators gesetzt bzw. gelöscht. Ist die Differenzspannung positiv ($U_{AIN0} - U_{AIN1} > 0 \text{ V}$), so führt der Ausgang einen High-Pegel. Ist die Differenzspannung hingegen negativ ($U_{AIN0} - U_{AIN1} < 0 \text{ V}$) so besitzt der Komparator-Ausgang folglich einen Low-Pegel.

Anwendung finden μC interne Analog-Komparatoren überall da, wo es darum geht, die Ausgangsspannung eines Sensors mit einer vorgegebenen Referenz zu vergleichen und gegebenenfalls in Abhängigkeit des Results bestimmte Abläufe zu initiieren. Beispiele hierfür sind unter anderem

- die Temperaturüberwachung mit LM35 Temperatursensoren¹
- die Prüfung auf Überschreitung/Unterschreitung einer vorgegebenen Potentiometerstellung
- die Auswertung von Opto-Reflex-Kopplern des Typs CNY70²
- die Unterspannungserkennung in batteriebetriebenen Systemen
- ...

Die Atmel ATmega328p interne Komparator-Stufe verfügt über vielfältigste Konfigurationsmöglichkeiten und ist somit flexibel in einer Vielzahl verschiedenster Applikationsszenarien einsetzbar.

¹ LM35 Precision Centigrade Temperature Sensors, Texas Instruments. Datenblatt: www.ti.com/product/lm35

² Reflective Optical Sensor with Transistor Output, Vishay Semiconductors. Datenblatt: www.vishay.com/docs/83751/cny70.pdf

Die Komparator-Stufe im ist nach jedem Systemstart standardmäßig aktiviert und arbeitet völlig autark und unabhängig vom eigentlichen AVR-Prozessorkern. Im Fall von batteriebetriebenen Anwendungen gilt es somit, die Komparator-Stufe bei Nichtbenutzung abzuschalten, um unnötigen Stromverbrauch (ca. 60µA) zu verhindern.

Die schaltungstechnische Realisierung dieses Peripheriemoduls des Atmel Atmega328p ist in Abbildung 1.2 dargestellt.

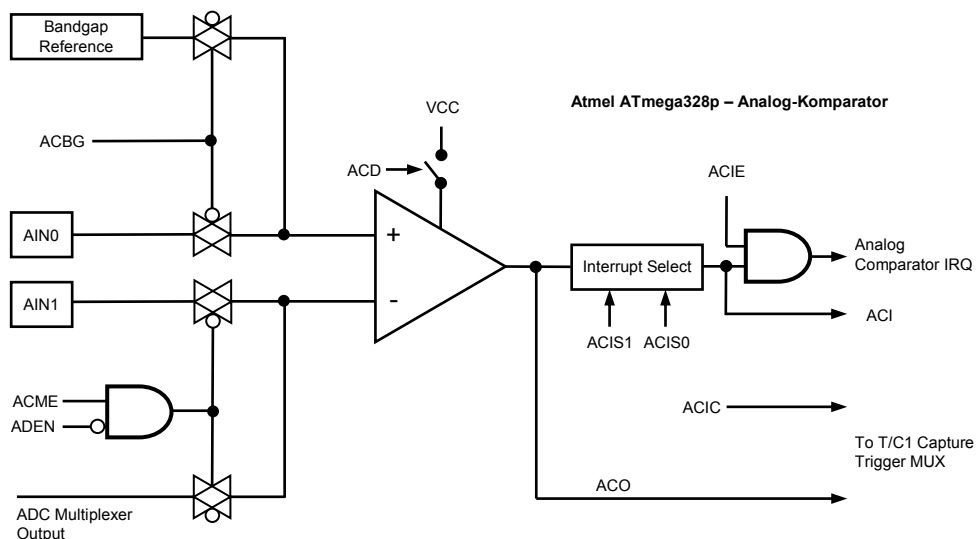


Abbildung 1-2 Blockschaltbild der Atmel ATmega328p internen analogen Komparator-Stufe

Wie hier zu sehen, ist es möglich, die Referenzspannung an Pin AIN0 durch eine stabile, temperaturunabhängige Bandgap-Referenz von ca. 1.1V zu ersetzen. Die Auswahl des jeweiligen Signals erfolgt über zwei invers angesteuerte Analogschalter. In gleicher Weise kann der nichtinvertierende Eingang des Komparators AIN1 mit dem Ausgang des ADC Multiplexers und somit wahlweise mit den Pins ADC7 bis ADC0 verschaltet werden. Diese Option ist jedoch nur bei einer Deaktivierung des Analog-Digital-Wandlers möglich (ADEN in ADCSRA) und muss über das Setzen des ACME-Bits im ADC Steuerregister ADCSRB ausgewählt werden.

Bit	7	6	5	4	3	2	1	0	
(0x7B)	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 1-3 Atmel ATmega328p – ADC – ADCSRB Steuerregister

Abbildung 1.4 zeigt exemplarisch das Messen einer Wechselspannung gegen die interne Bandgap-Referenz des ATmega328p. Die an Pin PD7 anliegende Wechselspannung besitzt einen dreieckförmigen Verlauf und variiert in ihrer Amplitude zwischen 0V und 5V. Wie hier zu sehen wird der Komparator-Ausgang ACO gesetzt, sobald das Eingangssignal an Pin PD7 unter der Referenzspannung von 1.1V liegt. Die Frequenz des ACO Signals ist folglich identisch mit der der externen Wechselspannung.

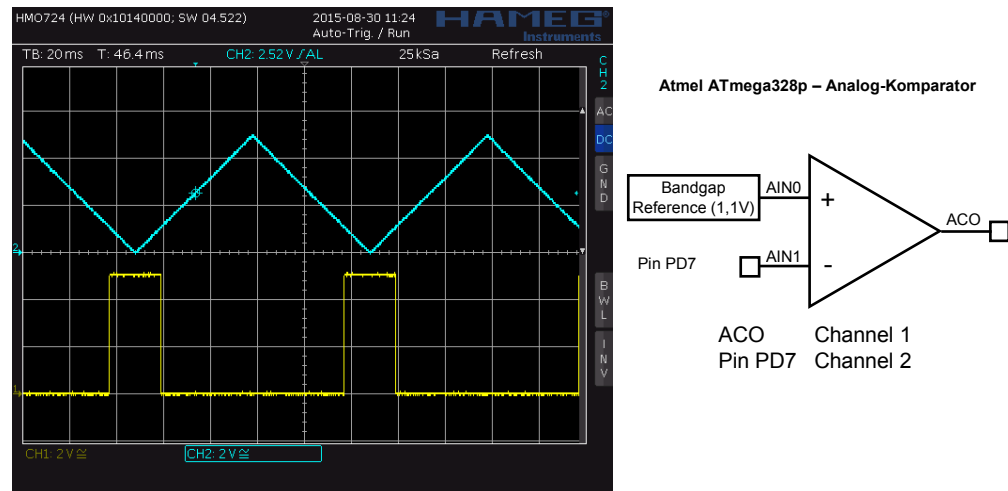


Abbildung 1-4 Verlauf der Ausgangsspannung eines analogen Komparators bei gegebener Referenzspannung und zeitlich variierender dreieckförmiger Vergleichsspannung

Der momentane Ausgangszustand des Komparators kann über das ACO-Bit im ACSR Steuerregister wie ein digitaler Eingang ausgelesen werden. Die Detektion eines Über- bzw. Unterschreitens des Referenzwertes kann sowohl durch regelmäßiges Abfragen (*engl. active polling, to poll: abfragen*) des ACO-Bits erfolgen, sowie rein interruptgesteuert. Kommt es zum Beispiel zu einem Unterschreiten der Referenzspannung, so schaltet der Komparator seinen Ausgang, und somit das ACO-Bit, auf logisch '1'. Das Setzen des ACO-Bits löst wiederum, bei gegebener Konfiguration, den Komparator-Interrupt³ aus. Die beiden zuvor genannten Betriebsarten des Analog-Komparators (Polling- bzw. Interrupt-Betrieb) werden in der nachfolgenden Sektion anhand konkreter Programmbeispiele ausführlich erläutert.

Befindet sich der Controller im „Idle Mode“, d.h. es wird kein Programm ausgeführt und es erfolgen keine Zugriffe der CPU auf den Flash-Speicher des ATmega328p, so kann der Komparator-Interrupt als Wake-Up Signal (*engl. to wake-up: aufwachen*) genutzt werden. Anwendung findet diese Konfiguration häufig bei der Implementierung von Schwellwert-Detektionen zur Reduzierung der Gesamtstromaufnahme. In den weitaus energieeffizienteren Sleep Modes des Controllers (ADC Noise Reduction, Power Save, Stand By und Power Down) ist der Komparator-Interrupt leider nicht als Wake-Up Signal verfügbar.

³ ANALOG_COMP_vect

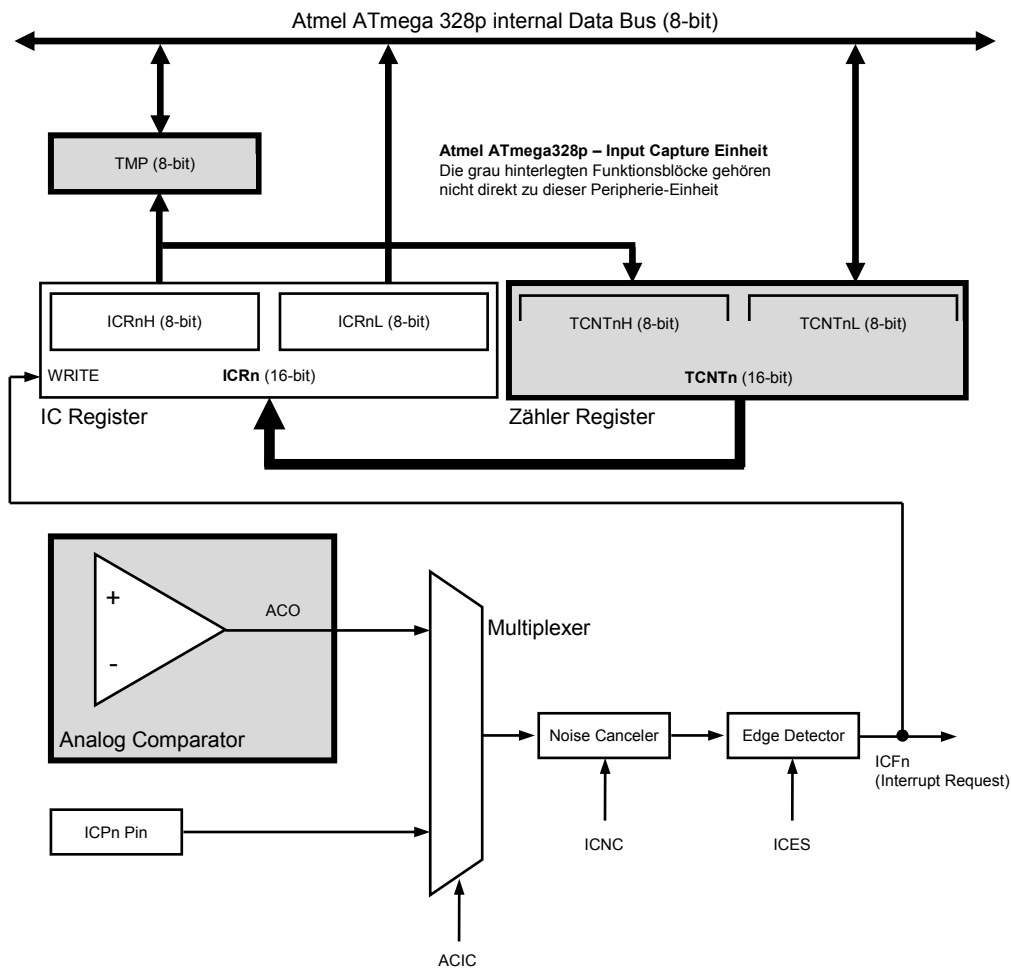


Abbildung 1-5 Blockschaltbild der Input-Capture Einheit des Timers T/C1

Vielfältigste Anwendungsmöglichkeiten ergeben sich durch die Benutzung des Analog-Komparators in Verbindung mit der Atmel ATmega328p internen Timer-Peripherie. Wie in Abbildung 1.5 dargestellt, ist der Komparatorausgang ACO neben der eigentlichen Verschaltung mit dem Steuerregister ACSR zudem intern über einen Multiplexer mit dem Eingang der Input-Capture-Einheit des Timers T/C1 verbunden. In dieser Schaltungskonfiguration wird der 16-bit Zählerstand des Timers T/C1 bei einer zuvor definierten Änderung des Komparatorausgangs ACO (Low-High, bzw. High-Low) ausgelesen und ins ebenfalls 16-bit breite Register ICR1 (kurz für engl.: *Input Capture Register*) übertragen. Gleichzeitig erfolgt das Setzen des Input Capture Flags ICF1, was bei gegebener Konfiguration zu einem Auslösen des Input Capture Interrupts führt⁴.

⁴ TIMER1_CAPT_vect

Der Timer T/C1 arbeitet in dieser Schaltungskonfiguration im „Normal Mode“, d.h. er wird schrittweise inkrementiert und läuft nach Erreichen des Maximalwertes einfach über. Die Zählgeschwindigkeit des Timers wird über das Setzen des Prescalers bestimmt⁵ und ist applikationsabhängig zu wählen.

Gilt es das 16-bit das nachfolgend dargestellte Register ICR1 softwareseitig auszulesen, so müssen die Zugriffe auf die Teilregister ICR1L und ICR1H in einer festen und vorgegebenen Reihenfolge erfolgen.

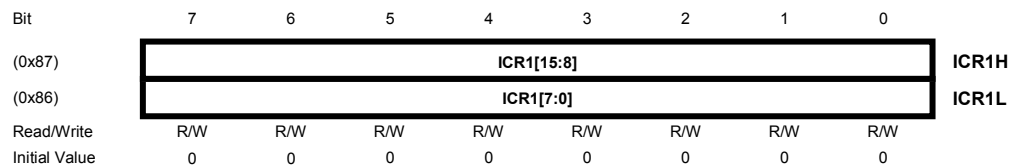


Abbildung 1-6 Atmel ATmega328p – Input Capture Register ICR1

Zuerst muss das Register ICR1L ausgelesen werden. Im Anschluss daran kann erst auf das Register ICR1H zugegriffen werden. Hintergrund ist, dass mit dem Zugriff auf das Register ICR1L eine Kopie des Registers ICR1H im temporären Sicherungsregister TEMP abgelegt wird. Kommt es seitens der CPU zu einem Auslesen des ICR1H Registers, so wird automatisch auf das TEMP Register zugegriffen. Hierdurch wird sichergestellt, dass die Inhalte der beiden Register immer demselben Input Capture Event zuzuordnen sind.

Trotz dieser Sicherungsfunktion ist der Zugriff auf das ICR1 Register nicht gänzlich unkritisch. Steht dem Prozessor nicht genügend Rechenleistung zur Verfügung um das ICR1 Register vor dem nächsten Capture Event auszulesen, so wird das Register ICR1 mit dem neuen Wert des Zählers T/C1 überschrieben. Das Auslesen des Registers ICR1 sollte somit immer unverzüglich nach Eintritt in die dem Input Capture Event zugeordnete Interrupt Service Routine erfolgen.

Kritisch ist die Verwendung der Input Capture Einheit vornehmlich in einem Multi-Interrupt System. Aufgrund der Tatsache, dass der Atmel ATmega328p die Unterbrechung einer Interrupt-Routine durch einen höher priorisierten Interrupt nicht erlaubt, ist ein rechtzeitiges Auslesen des ICR1 Registers nicht immer sichergestellt.

⁵ Der Vorteiler dient dazu, den Prozessor-Takt vorerst um einen einstellbaren Faktor zu reduzieren

Der prinzipielle Ablauf zweier aufeinanderfolgender Input Capture Events ist nachfolgend noch einmal grafisch aufgezeigt.

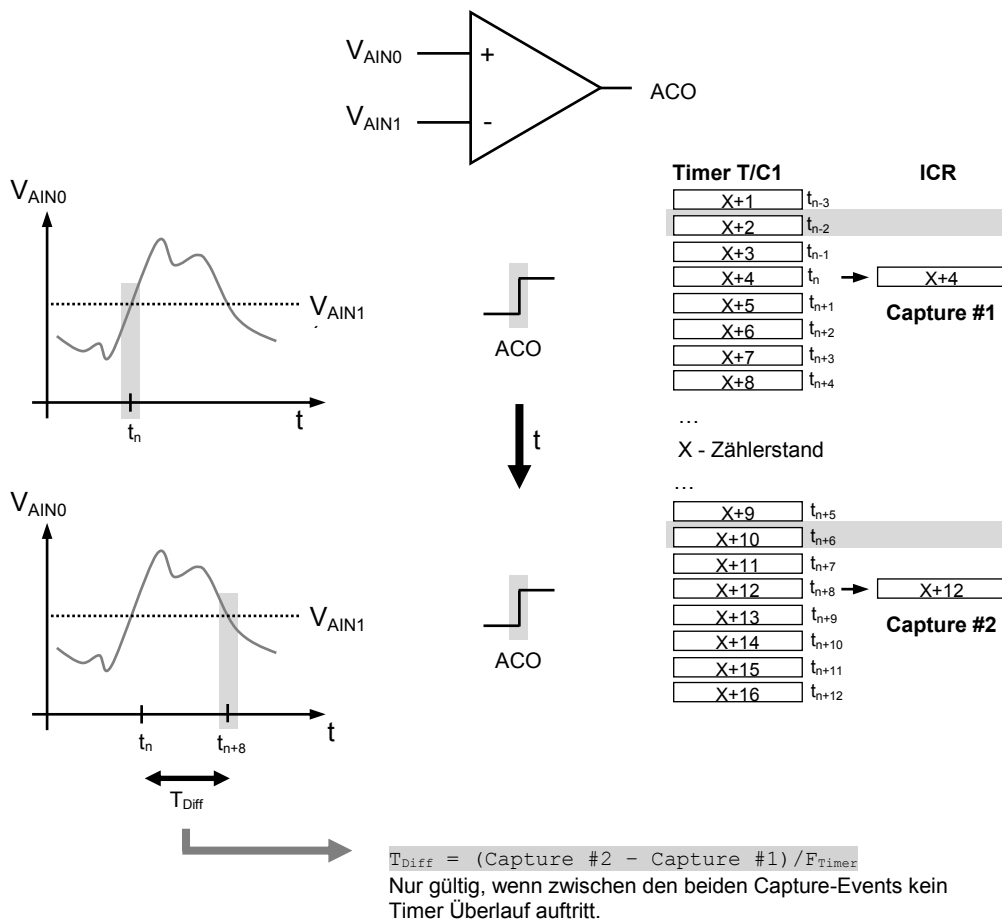


Abbildung 1-7 Prinzipielle Funktion der Input Capture Einheit in Kombination mit dem Atmel ATmega328p internen Analog-Komparator

Wie hier dargestellt wird in dieser Konfiguration einem externen Ereignis, wie zum Beispiel das Überschreiten der Referenzspannung an Eingang AIN1, ein sogenannter timestamp (*engl. für Zeitstempel*) zugewiesen. Hierbei handelt es sich um den Zählerstand des Timers T/C1 zum Zeitpunkt des Auslösens des Komparators. Über die Analyse, d.h. über eine Differenzbildung aufeinanderfolgender timestamps, kann die Frequenz oder das Tastverhältnis des externen Signals bestimmt werden.

Das zuvor beschriebene Vorhaben gestaltet sich relativ einfach, solange sich die zu bestimmende Zeitdifferenz auf das 16-bit Register des Timers T/C1 abbilden lässt. Etwas komplizierter wird es, wenn die 16-bit Auflösung nicht mehr ausreicht, d.h. Zeiten gemessen werden müssen die länger sind als eine Überlaufperiode des Timers⁶. Diese Thematik wird an späterer Stelle dieses Kapitels ausführlich diskutiert.

⁶ Überlaufperiode des T/C1 im „Normal-Mode“: $2^{16} \times PRESCALER / F_{Prozessor}$

1.2 ATmega328p - Analog-Komparator - Steuerung und Konfiguration

Die Konfiguration und Steuerung der Komparator-Stufe erfolgt über das in Abbildung 1.8 dargestellte ACSR Register (*kurz für engl.: Analog Comparator Control and Status Register*).

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

Abbildung 1-8 Atmel ATmega328p – Analog Komparator – ACSR Steuerregister

Im folgenden Abschnitt soll nun die Bedeutung sowie die Funktion der verschiedenen Steuer- und Konfigurationsbits des ACSR Register kurz erläutert werden. Eine ausführliche Beschreibung der Registerfunktion ist der Herstellerdokumentation zu entnehmen⁷.

- **Bit 7 – ACD:** Analog Comparator Disable

Über das Setzen dieses Bits erfolgt das Abschalten der Komparator-Stufe. Dies wird in der Regel dazu verwendet, um den Stromverbrauch während des Ruhe-Modus (Standby) noch weiter zu reduzieren. Vor der eigentlichen Abschaltung muss zudem die Interrupt-Funktionalität der Komparator-Stufe deaktiviert werden (Siehe ACIE-Bit – Register ACSR).

- **Bit 6 – ACBG:** Analog Comparator Bandgap Select

Ist dieses Bit gesetzt, so wird die positive Eingangsspannung von AIN0 durch eine interne Spannungsreferenz am Eingang der Komparator-Stufe ersetzt. Ist dieses Bit hingegen nicht gesetzt, so liegt die Spannung von AIN0 am positiven Eingang des Komparators.

- **Bit 5 – ACO:** Analog Comparator Output

Dieses Bit stellt eine synchronisierte Abbildung des Komparatorausgangs dar.

$$\text{IST} < \text{SOLL} \rightarrow \text{ACO} = 1$$

$$\text{IST} > \text{SOLL} \rightarrow \text{ACO} = 0$$

Durch die Synchronisation erfolgt die Aktualisierung dieses Bits ca. zwei Prozessortaktzyklen verzögert.

- **Bit 4 – ACI:** Analog Comparator Interrupt Flag

Das Setzen dieses Bits erfolgt automatisch durch den Prozessor, wenn ein Ereignis am Ausgang des Komparators einen Interrupt auslöst (konfiguriert durch die Bits ACI1 und ACI0).

⁷ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.247 ff.

Die entsprechende Komparator-Interrupt-Routine⁸ wird ausgeführt, wenn das ACIE-Bit sowie das globale Interrupt-Bit (I-Bit im SREG) gesetzt sind. Das ACI-Bit wird durch den Prozessor zurückgesetzt, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das Bit auch gelöscht werden, indem man eine logische '1' in das Flag schreibt.

▪ **Bit 3 – ACIE:** Analog Comparator Interrupt Enable

Wenn das ACIE-Bit sowie das globale Interrupt-Bit gesetzt sind, ist der Komparator-Interrupt freigeschaltet, andernfalls ist er deaktiviert.

▪ **Bit 2 – ACIC:** Analog Comparator Input Capture Enable

Durch das Setzen dieses Bits wird der Ausgang der Komparator-Stufe direkt mit der Input Capture Eingangs-Logik des Timers T/C1 verbunden. Um das Ausgangssignal ACO zum Triggern des Timer/Counter1 Interrupts zu verwenden, muss zudem das TICIE1-Bit im Timer Interrupt Mask Register (TIMSK) gesetzt sein.

▪ **Bit 1,0 - ACIS1, ACIS0:** Analog Comparator Interrupt Mode Select

Über das Setzen der Bits ACIS1 und ACIS0 wird festgelegt, welches Ereignis am Ausgang der Komparator-Stufe den Komparator-Interrupt auslöst. Die verschiedenen Konfigurationsmöglichkeiten sind in der folgenden Tabelle aufgeführt.

ACIS1	ACIS0	Analogkomparator Interrupt Modus
0	0	Interrupt bei jeder Änderung des Ausgangszustandes ACO (Toggle) auslösen
0	1	(Reserviert)
1	0	Interrupt bei fallender Flanke am Ausgang des Analogkomparators auslösen
1	1	Interrupt bei steigender Flanke am Ausgang des Analogkomparators auslösen

Tabelle 1-1 Konfigurationsmöglichkeiten der Atmel ATmega328p internen Komparator-Stufe

Um ein versehentliches Auslösen des Komparator-Interrupts zu verhindern, muss vor einer Änderung des Interrupt-Modes das ACIE-Bit im Steuerregister ACSR gelöscht werden.

Im Falle einer Benutzung der Pins AIN0 und AIN1 gilt es, die digitalen Eingangsstufen dieser Pins zu deaktivieren. Bei einer Eingangsspannung von ca. $V_{CC}/2$, d.h. bei der Hälfte der Versorgungsspannung, wären ansonsten beide Transistoren der digitalen Eingangsstufe leitfähig, was wiederum zu einem signifikanten Stromfluss führen würde. Das Abschalten der digitalen Eingangsstufen erfolgt über das in Abbildung 1.9 dargestellte DIDR1 Register (Digital Input Disable Register 1).

⁸ ANALOG_COMP_vect

Bit	7	6	5	4	3	2	1	0	
(0x7F)	-	-	-	-	-	-	AIN1D	AIN0D	DIDR1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 1-9 Atmel ATmega328p – Analog Komparator – DIDR1 Steuerregister

Wie bereits zu Beginn erwähnt besteht die Möglichkeit die Pins ADC7 bis ADC0 als negativen Eingang für den Analog-Komparator zu verwenden. Die Auswahl erfolgt intern über die Multiplexer-Stufe des Analog-Digital-Wandlers, so dass diese Peripherie-Einheit folglich deaktiviert sein muss. Wenn das ACME-Bit im ADCSRB-Register gesetzt ist und der ADC ausgeschaltet ist (ADEN in ADCSRA ist nicht gesetzt), selektieren die Bits MUX2 bis MUX0 im ADMUX Register den Pin, der als negativer Eingang für den Analog-Komparator verwendet wird. Ist das ACME-Bit hingegen nicht gesetzt bzw. das ADEN-Bit gesetzt, so wird immer der Pin AIN1 als negativer Eingang für die Komparator-Stufe verwendet. Alle Beschaltungsmöglichkeiten des negativen Komparator-Eingangs sowie die hierfür notwendigen Konfigurationseinstellungen sind in der nachfolgenden Tabelle aufgeführt.

ACME Bit	ADEN Bit	MUX[2:0]	Beschaltung des negativer Komparator-Eingangs
0	X	XXX	AIN1 – PD7 (Pin 11)
1	1	XXX	AIN1 – PD7 (Pin 11)
1	0	000	ADC0 – PC0 (Pin 23)
1	0	001	ADC1 – PC1 (Pin 24)
1	0	010	ADC2 – PC2 (Pin 25)
1	0	011	ADC3 – PC3 (Pin 26)
1	0	100	ADC4 – PC4 (Pin 27)
1	0	101	ADC5 – PC5 (Pin 28)
1	0	110	ADC6 – (Pin 19) ⁹
1	0	111	ADC7 – (Pin 22)

Tabelle 1-2 Beschaltungsmöglichkeiten des negativen Komparator-Eingangs¹⁰

Wird der Analog-Komparator zusammen mit der Input Capture Einheit des Timers T/C1 im Interrupt Mode betrieben, so gilt es das Input Capture Interrupt Enable Bit ICIE1 im nachfolgend dargestellten Timer Register TIMSK1 (Timer/Counter 1 Interrupt Mask Register) zu setzen.

Bit	7	6	5	4	3	2	1	0	
(0x6F)	-	-	ICIE1-	-	-	OCIE1B-	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 1-10 Atmel ATmega328p – Timer/Counter 1 Interrupt Mask Register

Weitere Relevanz besitzt in diesem Kontext das Steuerregister TCCR1B (Timer/Counter1 Control Register B).

⁹ Die ADC Pins 6 & 7 sind auf dem Arduino Uno SMD R3 nicht verfügbar

¹⁰ Die hier angegebene Pinbelegung bezieht sich auf die Arduino Uno SMD R3 Plattform – ATmega328p - MLF – 32 Gehäuse

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 1-11 Atmel ATmega328p – Timer/Counter1 Control Register B

Über das Bit ICES1 im zuvor aufgeführten TCCR1B Register wird bestimmt, ob die steigende (ICES1 gesetzt) oder fallende (ICES1 nicht gesetzt) Flanke zur Auswertung des Input Capture Signals heran gezogen wird. Ist die Signalquelle zudem stark verrauscht, kann die Rauschunterdrückung über das Input Capture Noise Canceller Bit ICNC1 eingeschaltet werden. In diesem Fall wird bei der Detektion der konfigurierten Flanke über 4 Taktzyklen das Signal überwacht und nur dann, wenn alle 4 Messungen identisch sind, wird die entsprechende Aktion ausgelöst. Folglich erfolgt die Übernahme des aktuellen Zählerstands von Timer T/C1 in das ICR1 Register um vier Taktzyklen verzögert.

1.3 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #1

Im folgenden Abschnitt soll nun die Benutzung der Atmel ATmega328p internen Komparator-Stufe an konkreten Beispielen demonstriert werden. Betrachten wir im ersten Fall folgende Situation: Über den Analogkomparator des ATmega328p soll eine an Pin AIN1 anliegende, zeitlich veränderliche Spannung fortlaufend überwacht werden. Ist die an diesem Pin anliegende Spannung kleiner als die interne Bandgap-Referenz von ca. 1.1V am nichtinvertierenden Eingang des Komparators, so soll eine an Pin PB5 angeschlossene rote LED eingeschaltet werden. Ist die an Pin AIN1 anliegende Spannung größer als die interne Referenzspannung am nichtinvertierenden Eingang des Komparators, so soll die LED ausgeschaltet bleiben.

Wie in Abbildung 1.12 dargestellt, soll die variierende Spannung an Pin AIN1 zu Testzwecken mit einem Potentiometer vorgegeben werden. Wie hier illustriert wird bei einem Potentiometer ein beweglicher Schleifkontakt über ein begrenzt leitfähiges Trägermaterial geführt. Standard ist ein Drehbereich von 270°. Die Position des Schleifers teilt den Gesamtwiderstand des Trägers in zwei Teilwiderstände auf. Legt man nun an beiden Enden des Widerstandsträgers eine Spannung an, so bestimmt die Position des Schleifers das Widerstandsverhältnis und damit die Höhe der abzugreifenden Spannung am Schleifkontakt. Ist der Widerstandswert des Widerstandsträgers über seine Länge konstant, dann steigt folglich mit dem Drehwinkel des Schleifkontakts der abgegriffene Spannungswert linear an.

Durch die Veränderung der Position des Schleifkontakts, lässt sich folglich eine Messspannung am ADC des Mikrocontrollers im Bereich von 0V und der Versorgungsspannung VCC (5V) des Mikrocontrollers stufenlos einstellen. Der Widerstandswert des Potentiometers ist aufgrund des hohen Eingangswiderstandes des ADC unkritisch. Eine reale Anwendung könnte zum Beispiel die Erkennung einer Batterie-Unterspannung darstellen.

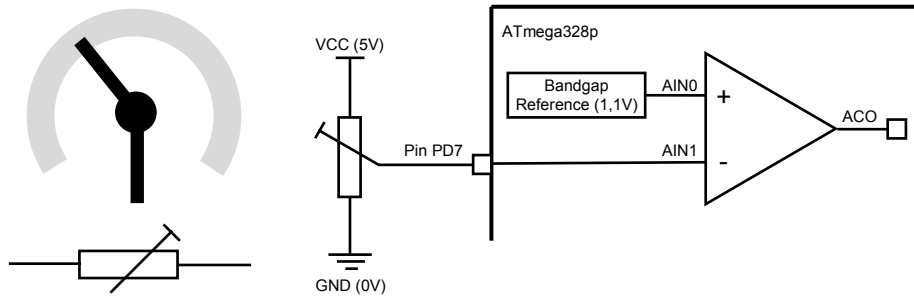


Abbildung 1-12 Schaltungsaufbau – Atmel ATmega328p mit Potentiometer an Pin AIN1

Die Detektion eines Über- bzw. Unterschreitens des Referenzwertes an Pin AIN1 soll zuerst durch regelmäßige Abfragen des ACO-Bits erfolgen (Polling). Eine mögliche Implementierung der zuvor beschriebenen Funktionalität ist in Quellcode 1.1 dargestellt. Eine Beschreibung der einzelnen Programmschritte ist den Kommentaren zu entnehmen.

```

1. // Atmel ATmega328p - Analog Komparator Demo - Polling Modus
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6.
7. #include <avr/io.h>
8. #include <util/delay.h>
9.
10. int main(void){
11.     // -- Konfiguration Status-LED --
12.     DDRB |= (1 << PB5);           // PIN PB5 als Ausgang
13.     PORTB &= ~(1 << PB5);         // LED ausschalten
14.     // -- Konfiguration Komparator-Stufe --
15.     DDRD &= ~(1 << PD7);          // AIN1 als Eingang konfigurieren...
16.     PORTD &= ~(1 << PD7);         // ... und Pull-Up-Widerstand an diesem
17.                                     // Pin deaktivieren
18.
19.     ACSR |= (1 << ACBG);           // Vref an AIN0 auswählen
20.     _delay_ms(250);               // Warten bis sich Vref stabilisiert hat
21.
22.     DIDR1 = (1 << AIN1D);          // Digitale Eingangsstufe an PIN AIN1
23.                                     // deaktivieren
24.
25.                                     // Endlosschleife, ...
26.     while(1){
27.         if((ACSR & (1 << ACO)) == 0){
28.             PORTB &= ~(1 << PB5);
29.         }
30.         else{
31.             PORTB |= (1 << PB5);
32.         }
33.     }
34.     return 0;
35. }

```

Quellcode 1-1 Atmel ATmega328p - Analog Komparator Demo – Polling Modus

Die delay Routine in Programmzeile '19' ist notwendig, da die interne Bandgap-Referenz nach Aktivierung ca. 70µs zur Stabilisierung benötigt.

Wie in Zeile '27' zu sehen erfolgt im Hauptprogramm eine permanente Überprüfung des Komparator-Ausgangs. Ist die Spannung U_{AIN0} (in diesem Fall die interne Band-Gap Referenz) größer als die extern anliegende Spannung U_{AIN1} , so ist das ACO-Bit gesetzt, was folglich zum Anschalten der LED an Ausgang PB5 führt. Alternativ kann, wie in Quellcode 1.2 gezeigt, diese Funktionalität auch über eine Interrupt basierte Programmstruktur realisiert werden.

```
1. // Atmel ATmega328p - Analog Komparator Demo - Interrupt Modus
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6.
7. #include <avr/io.h>
8. #include <avr/interrupt.h>
9. #include <util/delay.h>
10.
11. // ISR Routine - ANALOG Komparator
12. ISR(ANALOG_COMP_vect){
13.     if(ACSR & (1 << ACO)){           // Komparatorausgang ACO gesetzt?
14.         PORTD &= ~(1 << PB5); // wenn ja, LED ausschalten
15.     }
16.     else
17.         PORTB |= (1 << PB5); // wenn nicht, LED einschalten
18. }
19.
20. int main(void){
21.     // -- Konfiguration Status-LED --
22.     DDRB |= (1 << PB5); // PIN PB5 als Ausgang konfigurieren
23.     PORTB &= ~(1 << PB5); // LED ausschalten
24.     // -- Konfiguration Komparator-Stufe --
25.     DDRD &= ~(1 << PD7); // AIN1 als Eingang konfigurieren...
26.     PORTD &= ~(1 << PD7); // ... und Pull-Up-Widerstand an diesem
27.     // Pin deaktivieren
28.     ACSR = (1 << ACPBG); // Vref an AIN0 auswählen
29.     _delay_ms(250); // Warten bis sich Vref stabilisiert hat
30.     ACSR |= (1 << ACIE); // Komparator-Interrupt aktivieren
31.     ACSR &= ~(1 << ACFI0); // Interrupt-Mode - Interrupt bei jeder
32.     ACSR &= ~(1 << ACFI1); // Änderung von ACO auslösen
33.     DIDR1 = (1 << AIN1D); // Digitale Eingangsstufe an PIN AIN1
34.     // deaktivieren
35.     sei(); // Interrupts freischalten
36.
37.     while(1){ // Endlosschleife, ...
38.     }
39.     return 0;
40. }
```

Quellcode 1-2 Atmel ATmega328p - Analog Komparator Demo – Interrupt Modus

An dieser Stelle sei angemerkt, dass die Programmzeilen '31' und '32' im oben aufgeführten Quellcode lediglich der Verbesserung der Lesbarkeit dienen.

Laut Datenblatt¹¹ stellt der selektierte Komparator-Interrupt-Mode (Interrupt bei jeder Änderung des Ausgangszustandes von ACO auslösen) bereits die Default-Konfiguration dar und müsste somit nicht mehr explizit ausgewählt werden.

Geht man weiter und ersetzt das an Pin PD7 angeschlossene Potentiometer durch eine Wechselspannungsquelle, so erfolgt das Blinken der LED mit der gleichen Frequenz wie die Wechselspannung. Abbildungen 1.13 und 1.14 zeigen exemplarisch das Messen einer Wechselspannung. Die an Pin PD7 anliegende Wechselspannung besitzt einen dreieckförmigen bzw. sinusförmigen Verlauf und variiert in ihrer Amplitude zwischen 0V und 5V. Die an Pin PD7 anliegende Wechselspannung besitzt einen dreieckförmigen bzw. sinusförmigen Verlauf und variiert in ihrer Amplitude zwischen 0V und 5V. Wie in beiden Fällen zu sehen wird der Komparator-Ausgang ACO gesetzt, sobald das Eingangssignal an Pin PD7 unter der Referenzspannung von 1.1V liegt. Die Frequenz des ACO Signals und somit die LED Blinkfrequenz ist folglich identisch mit der der externen Wechselspannung.

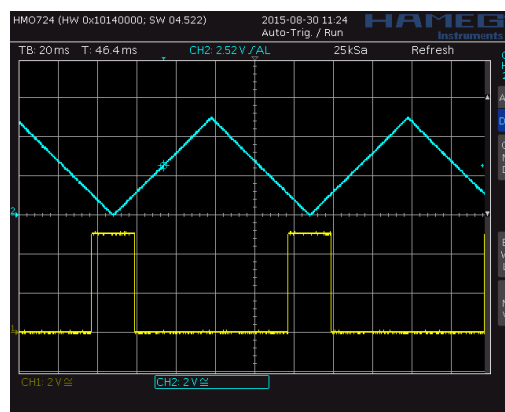


Abbildung 1-13 Verlauf der Ausgangsspannung des ATmega328p internen Analog-Komparators bei gegebener Referenzspannung und dreieckförmiger Vergleichsspannung

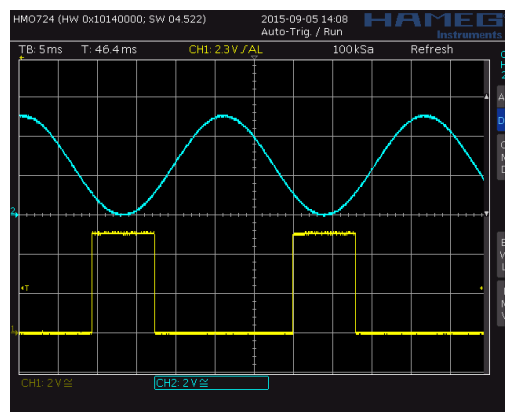
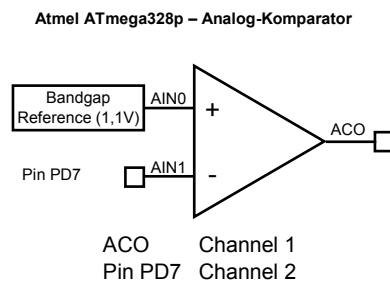
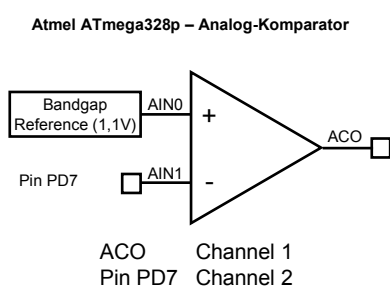


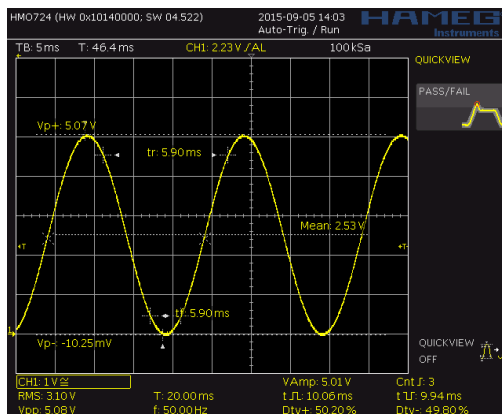
Abbildung 1-14 Verlauf der Ausgangsspannung des ATmega328p internen Analog-Komparators bei gegebener Referenzspannung und sinusförmiger Vergleichsspannung



¹¹ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.247

1.4 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #2

Der folgende Abschnitt diskutiert die Frequenzmessung eines sinusförmigen Wechselspannungssignals auf Basis der Atmel ATmega328p internen Komparator- sowie Input-Capture-Peripherie. Folgende Situation soll hierzu betrachtet werden: An Pin PD7 des ATmega328p liegt eine sinusförmige Wechselspannung an deren Frequenz es zu bestimmen gilt. Das Sinussignal besitzt, wie in Abbildung 1.15 dargestellt, einen Offset von 2.5V und variiert in seiner Amplitude zwischen 0V und 5V.



Atmel ATmega328p – Analog-Komparator

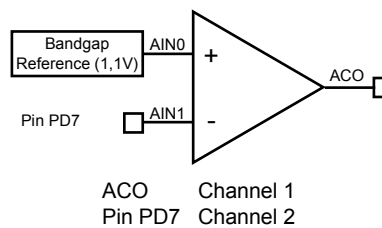


Abbildung 1-15 Verlauf der Ausgangsspannung des ATmega328p internen Analog-Komparators bei gegebener Referenz- und sinusförmiger Vergleichsspannung (2.5V Offset)

Die an Pin PD7 anliegende Spannung soll intern mit dem Eingang AIN1 des Analog-Komparators verbunden und ist fortlaufend mit der internen Bandgap-Referenz von ca. 1.1V am nichtinvertierenden Eingang des Komparators zu vergleichen. Ist die Spannung U_{AIN0} (in diesem Fall die interne Band-Gap Referenz) größer als die extern anliegende Spannung U_{AIN1} , so wird, wie im vorangegangenen Abschnitt beschrieben, das ACO-Bit gesetzt. Der Ausgang des Komparators soll zudem intern mit dem Eingang der Input-Capture-Einheit des Timers T/C1 verbunden, so dass der 16-bit Zählerstand des Timers T/C1 bei einer zuvor definierten Änderung (Low-High) des Komparatorausgangs ACO ausgelesen und ins 16-bit breite Register ICR1 übertragen wird. Die Input-Capture-Einheit des ATmega328p gilt es zudem so zu konfigurieren, dass dieses Ereignis (ACO: Low-High) zum Auslösen des Input-Capture-Interrupts führt.

Wie in Abbildung 1.16 dargestellt wird in dieser Schaltungskonfiguration dem Unterschreiten der Referenzspannung an Eingang AIN1 ein sogenannter timestamp zugewiesen. Hierbei handelt es sich um den momentanen Zählerstand des Timers T/C1 zum Zeitpunkt des Auslösens des Komparators. Über die Differenzbildung aufeinanderfolgender timestamps im Input-Capture-Interrupt, kann folglich auf die Frequenz des externen Wechselsignals geschlossen werden. Der gemessene Frequenzwert soll anschließend über die serielle Schnittstelle des ATmega328p and den PC übertragen werden.

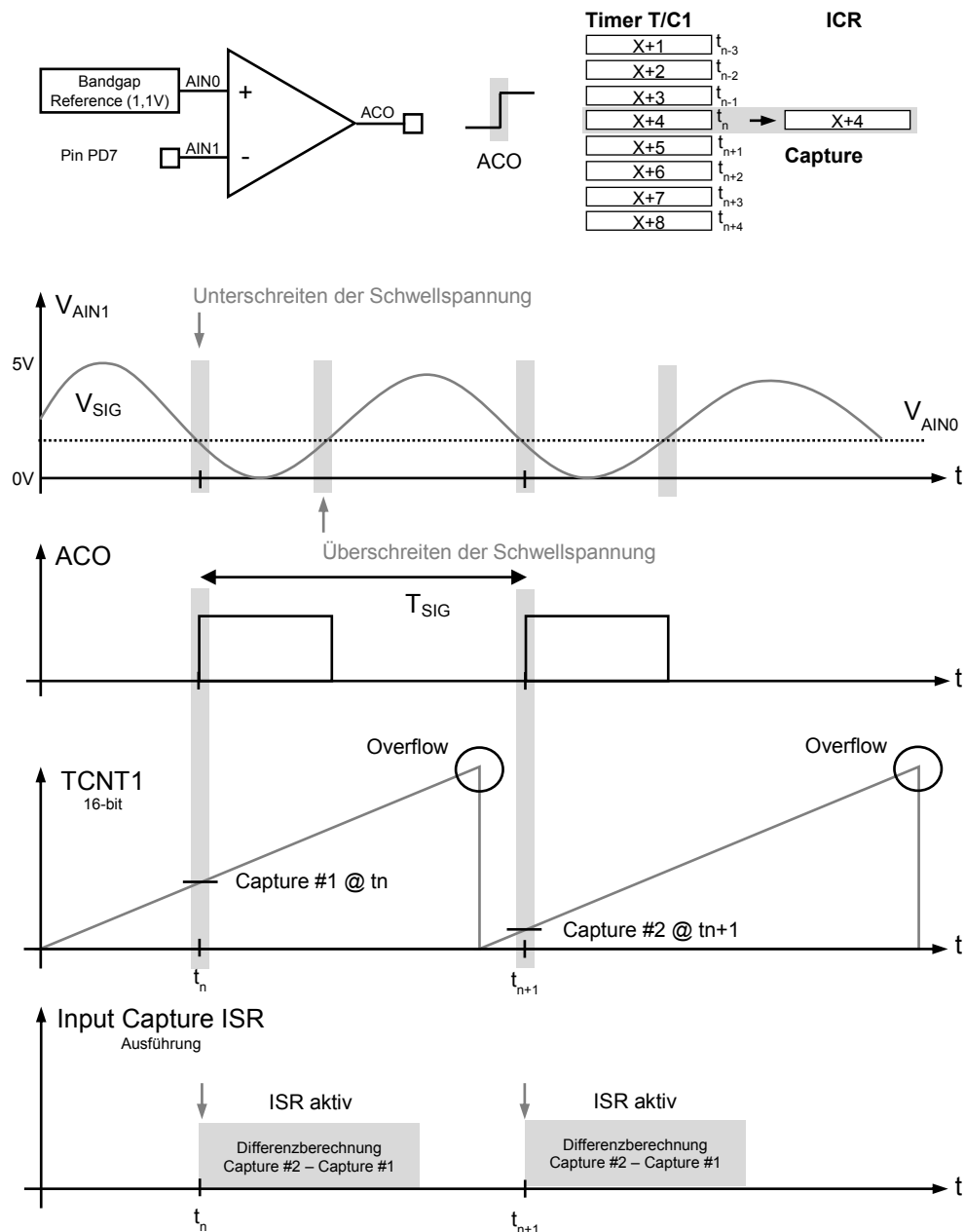


Abbildung 1-16 Einfache Frequenzmessung auf Basis der Input Capture Einheit in Kombination mit dem Atmel ATmega328p internen Analog-Komparator

Abbildung 1.16 zeigt neben der zu messendem Signalform und dem daraus resultierenden ACO Signal, den Verlauf von Timer T/C1 sowie die in diesem Zeitraum auftretenden Capture Events. Festzuhalten gilt hier, dass es zwischen den beiden Erfassungszeitpunkten zu einem Überlauf ($65535_{16} - 0_{16}$) des Timer T/C1 kommt. Dies stellt, wie nachfolgend gezeigt, kein Problem dar, da der Zahlenbereichsüberlauf durch die vorzeichenlose Modulo 2^{16} Berechnung der Differenz kompensiert wird.

Differenzberechnung – Frequenzmessung Analog-Komparator/Input-Capture-Unit

Die Zeitdifferenz zwischen t_n und t_{n+1} muss über die Differenz der beiden Timer T/C1 Werte ermittelt werden. Die hier aufgeführten Zählerstände sind frei gewählt:

Capture #2 = 9836

Capture #1 = 43600

Wie in Abbildung 1.16 zu sehen besteht folgender Zusammenhang:

$$\text{capture \#2 @}t_{n+1} < \text{capture \#1 @}t_n$$

Die Berechnung der Zeitdifferenz erfolgt Modulo 2^{16} , da als Datentyp der vorzeichenlose 16-bit Typ unsigned int verwendet wird:

$$t_{n+1} - t_n = (\text{capture \#2} - \text{capture \#1}) \text{ MOD } 2^{16}$$

Für die oben aufgeführten Beispielwerte ergibt sich folgende Zeitdifferenz:

$$\text{capture \#2} - \text{capture \#1} = 0x266C - 0xAA50 = 0xFFFF7C1C \text{ (unsigned long 32-bit)}$$



$$\text{capture \#2} - \text{capture \#1} = 0xFFFF7C1C \text{ MOD } 2^{16} = 0x7C1C \text{ (unsigned int 16-bit)} \\ = 31772$$

Dies entspricht exakt der gesuchten Zeitdifferenz:
 $65536 - 43600 + 9836 = 31772$

Abbildung 1-17 Differenzberechnung – Frequenzmessung Analog-Komparator/Input-Capture-Unit – Teil #1

Der in Abbildung 1.17 ermittelte Differenzwert gibt die Anzahl der Timer Inkremente zwischen den beiden Capture Events wieder. Den eigentlichen Zeitwert erhält man folglich durch Multiplikation mit dem Kehrwert der Timer-Frequenz.

Das zuvor beschriebene Vorhaben ist valid, solange sich die zu bestimmende Zeitdifferenz auf das 16-bit Register des Timers T/C1 abbilden lässt. Etwas komplizierter wird es, wenn die 16-bit Auflösung nicht mehr ausreicht, d.h. Zeiten gemessen werden müssen die länger sind als eine Überlaufperiode. Dieses Szenario ist in Abbildung 1.19 illustriert. Das zu messende Signal besitzt in diesem Fall eine im Vergleich zur Zeitbasis des Timers T/C1 kleine Frequenz. Hieraus resultiert, dass zwischen zwei Capture Events mehrere Überläufe des Timers T/C1 zu verzeichnen sind, die es folglich mitzuzählen gilt. Dies wird in der Regel durch den Overflow Interrupt des Timers realisiert. Das in Abbildung 1.17 eingeführte Verfahren gilt es entsprechend erweitern:

Differenzberechnung – Frequenzmessung Analog-Komparator/Input-Capture-Unit

$$t_{n+1} - t_n = (\text{capture \#2} - \text{capture \#1}) \text{ MOD } 2^{16} + 2^{16} \cdot \text{OverflowCnt}$$

OverflowCnt: Anzahl der Timer Überläufe innerhalb der Messung

Abbildung 1-18 Differenzberechnung – Frequenzmessung Analog-Komparator/Input-Capture-Unit – Teil #2

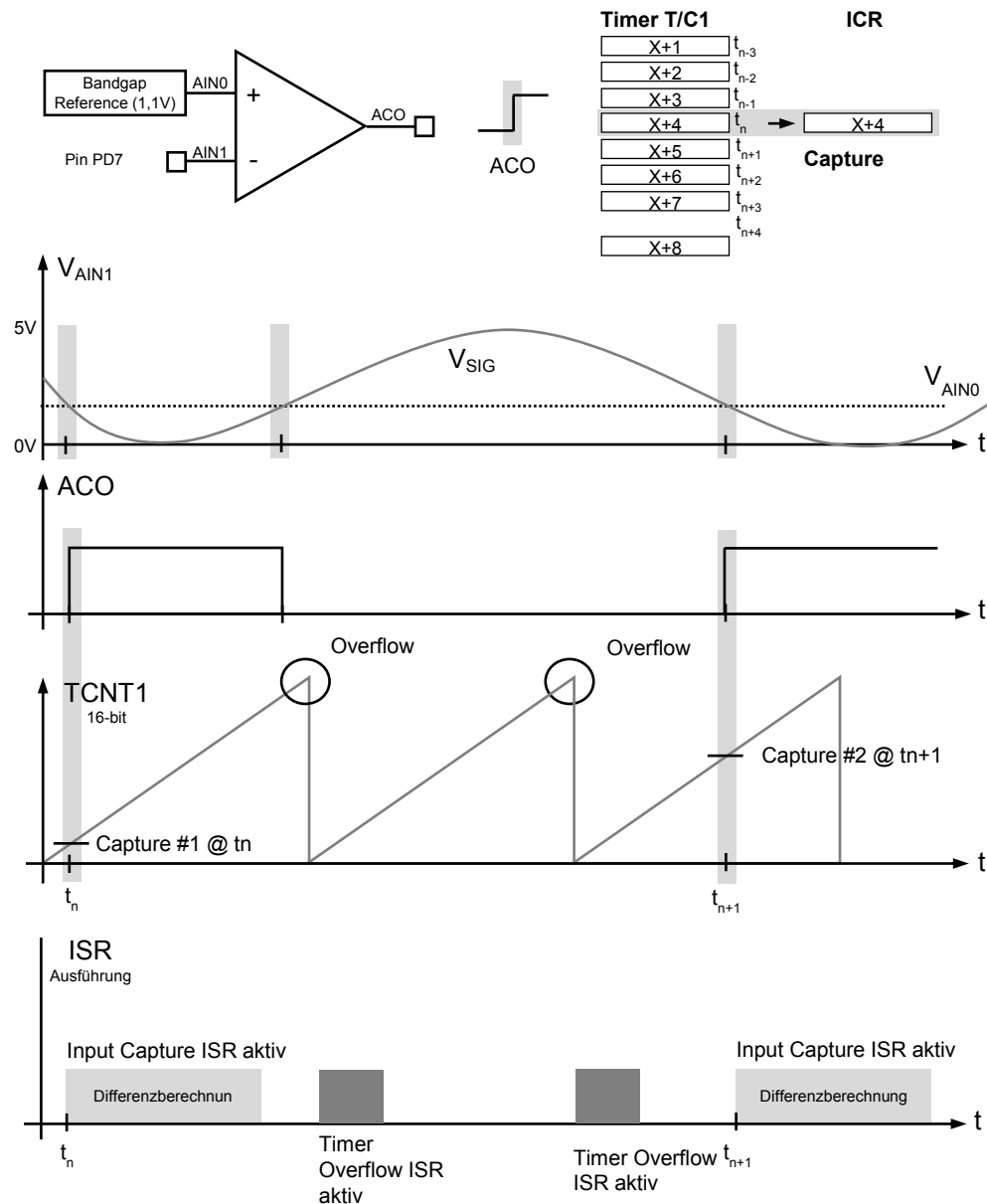


Abbildung 1-19 Erweiterte Frequenzmessung auf Basis der Input Capture Einheit in Kombination mit dem Atmel ATmega328p internen Analog-Komparator

Eine mögliche Implementierung der zuvor beschriebenen Funktionalität ist in Quellcode 1.3 dargestellt. Eine Beschreibung der einzelnen Programmschritte ist den Kommentaren zu entnehmen.

```
1. // Atmel ATmega328p - Frequenzmessung auf Basis des Analog Komparator und
2. // der Input-Capture Unit. Formatierte Ausgabe via USART - PART#1#
3. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
4. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
5.
6. #define F_CPU 16000000UL
7. #define BAUDRATE 9600
8. #define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)
9.
10. #include <avr/io.h>
11. #include <avr/interrupt.h>
12. #include <util/delay.h>
13. #include <stdlib.h>
14.
15. #ifndef TRUE #define TRUE 1 #define FALSE 0 #endif
16. // Anzahl der Timer Overflows während der Messung
17. volatile unsigned char OverflowCnt = 0;
18. // ICR-Wert bei der 1.High-Flanke speichern
19. volatile unsigned int Start = 0;
20. // ICR-Wert bei der 2.High-Flanke speichern
21. volatile unsigned int End = 0;
22. // Job Flag
23. volatile unsigned char locked = 0;
24.
25. void usart_init(void){
26.     UBRROH = (uint8_t)(BAUD_PRESCALLER>>8);
27.     UBRROL = (uint8_t)(BAUD_PRESCALLER);
28.     UCSROB = (1<<RXEN0)|(1<<TXEN0);
29.     UCSROC = (3<<UCSZ00);
30. }
31. void usart_send_byte( unsigned char data ){
32.     while(!(UCSR0A & (1<<UDRE0)));
33.     UDR0 = data;
34. }
35.
36. void usart_putstring(char* string_ptr ){
37.     while(*string_ptr != 0x00) {
38.         usart_send_byte(*string_ptr);
39.         string_ptr ++;
40.     }
41. }
42.
43. ISR( TIMER1_CAPT_vect ){
44.     static unsigned char First_Edge = TRUE;
45.
46.     if(locked) return;
47.
48.     if(First_Edge){
49.         Start      = ICR1;
50.         OverflowCnt = 0;
51.         First_Edge = FALSE; // Die nächste Flanke ist das Ende der Messung
52.     } else {        // Zweite Flanke. Messung stoppen
53.         End        = ICR1;
54.         locked      = TRUE; // Vollständige Messung. Auswertung starten
55.         First_Edge  = TRUE; // Mit der nächsten Flanke beginnt der nächste
56.                               // Messzyklus
57.     }
```

Quellcode 1-3 Analog-Komparator/Input Capture-Einheit - Frequenzmessung Teil#1

```

58. // Atmel ATmega328p - Frequenzmessung auf Basis des Analog Komparator und
59. // der Input-Capture Unit. Formatierte Ausgabe via USART - PART#2#
60. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
61. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
62.
63. ISR(TIMER1_OVF_vect){
64.     OverflowCnt++;
65. }
66.
67. int main(void){
68.     char usart_string[8];
69.     double freq_result = 0.0;
70.     // -- USART -
71.     usart_init();
72.     // -- Komparator-Stufe -
73.     DDRD  &= ~(1 << PD7); // AIN1 als Eingang konfigurieren...
74.     PORTD &= ~(1 << PD7); // ... und Pull-Up-Widerstand deaktivieren
75.     DIDR1 = (1 << AIN1D); // Digitale Eingangsstufe an AIN1 deaktivieren
76.     ACSR |= (1 << ACBG); // Vref an AIN0 auswählen
77.     _delay_ms(250); // Warten bis Vref stabil ...
78.     ACSR |= (1 << ACIC); // ACO - Input Capture Event
79.     // -- Input Capture Unit - T/C1 -
80.     TCCR1B = (1<<ICES1); // Input Capture Edge
81.     TCCR1B |= (1<<CS10); // No Prescaler
82.     TIMSK1 = (1<<ICIE1); // Capture Interrupt aktivieren
83.     TIMSK1 |= (1<<TOIE1); // Overflow Interrupt aktivieren
84.
85.     sei();
86.
87.     while(1){
88.         if(locked){ // liegt eine vollständige Messung vor?
89.             // Ja, dann Anzahl der Takte berechnen
90.             freq_result = (OverflowCnt * 65536) + End - Start;
91.             // f=1/t Signalfrequenz (kHz)
92.             freq_result = F_CPU / freq_result;
93.             // ...
94.             itoa(freq_result, usart_string,8);
95.             // 3 Nachkommastellen ausgeben
96.             dtostrf(freq_result,5,3, usart_string);
97.             // Ausgabe via USART
98.             usart_putstring(usart_string);
99.             usart_putstring(" kHz");
100.            usart_send_byte(10);
101.            // nächste Messung kann starten!
102.            locked = FALSE;
103.        }
104.    }
105. }

```

Quellcode 1-4 Analog-Komparator/Input Capture-Einheit - Frequenzmessung Teil#2

1.5 ATmega328p - Analog-Komparator - Praxisbeispiele - Teil #3

Im vorangegangenen Abschnitt wurde ein Verfahren zur Frequenzbestimmung eines externen Signals auf Basis des Analog-Komparators sowie der ATmega328p internen Timer-Peripherie vorgestellt. Eine weitere Applikation in diesem Kontext stellt das Messen von Pulsweiten dar.

In der Messtechnik kommt es relativ häufig vor, dass Sensoren die Messgröße in eine äquivalente Pulsweite konvertieren. Hier wird die zu übertragende Information (d.h. eine dem Messsignal äquivalente Spannung) auf ein zeitliches Äquivalent, in diesem Fall die Pulsdauer (auch als Pulsbreite bezeichnet), abgebildet. Die Pulsgenerierung erfolgt intern, wie in Abbildung 1.20 gezeigt, durch einen Vergleich der internen Messspannung mit einem dreieckförmigen Trägersignal. Hieraus ergibt sich ein Ausgangssignal, dessen Pulsweite in Relation zur (konstanten) Pulsperiode proportional zur momentanen Messspannung ist. Dieses Verfahren erscheint auf den ersten Blick wie eine unnötige Verkomplizierung der Messkette, besitzt jedoch in der praktischen Anwendung vielerlei Vorteile. Da sich eingekoppelte Störungen vor allem auf die Amplitude eines Signals auswirken, sollte dieser Parameter zur Informationskodierung eines Sensorsignals nicht herangezogen werden. Signalparameter wie Pulsweite bzw. Frequenz sind im Vergleich relativ störresistent und ermöglichen eine unkritische Übertragung des Messsignals selbst über längere Distanzen hinweg. Die Dekodierung gestaltet sich zudem unkritisch, da sich die Pulsbreite eines Signals mit Hilfe eines Mikrocontrollers sehr einfach und präzise bestimmen lässt.

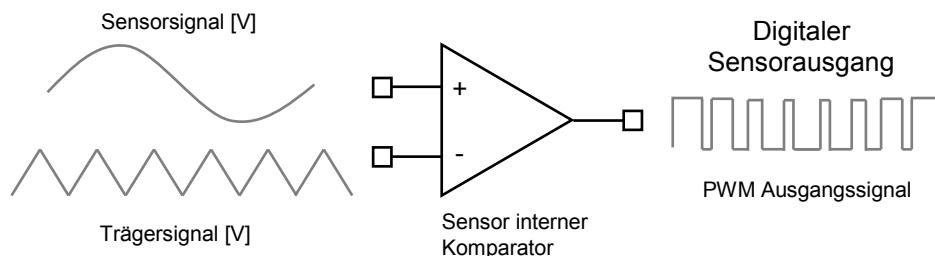


Abbildung 1-20 Sensor mit digitalem PWM Ausgang

Das Messen von Pulsweiten auf Basis des Atmel ATmega328p wird im nachfolgenden Abschnitt am Beispiel eines HC-SR04 Ultraschall Messmoduls ausführlich diskutiert. Dieser setzt eine gemessene Entfernung zwischen 2cm und 3m in eine proportionale Pulslänge um.

Als Ultraschall bezeichnet man akustische oder mechanische Schwingungen oberhalb von 20 kHz. Der technische Anwendungsbereich von Ultraschall bei Übertragungen im Medium Luft reicht von ca. 30 kHz bis etwa 300 kHz. Ultraschall-Sensoren werden primär als Abstands- oder Entfernungs-Sensoren eingesetzt. Sie bestehen aus einem Ultraschall-Lautsprecher und einem darauf abgestimmten Ultraschall-Mikrofon, beides meistens in Piezo-Technik. Der Lautsprecher sendet einen kurzen Schallimpuls aus. Dessen Laufzeit durch den Raum, bis er vom Mikrofon aufgenommen wird, ist proportional zum Weg, den das Ultraschallsignal zurückgelegt hat.

Prinzipiell wird bei der Entfernungsmessung die Zeit zwischen dem Senden eines kurzen Ultraschall-Bursts bis zum Eintreffen des Echos gemessen, woraus die Entfernung zu einem Objekt ermittelt wird. Da der Schall in der Luft bei 20°C ca. 343 m/s zurücklegt, kann man die Entfernung eines Objekts aus der Laufzeit berechnen:

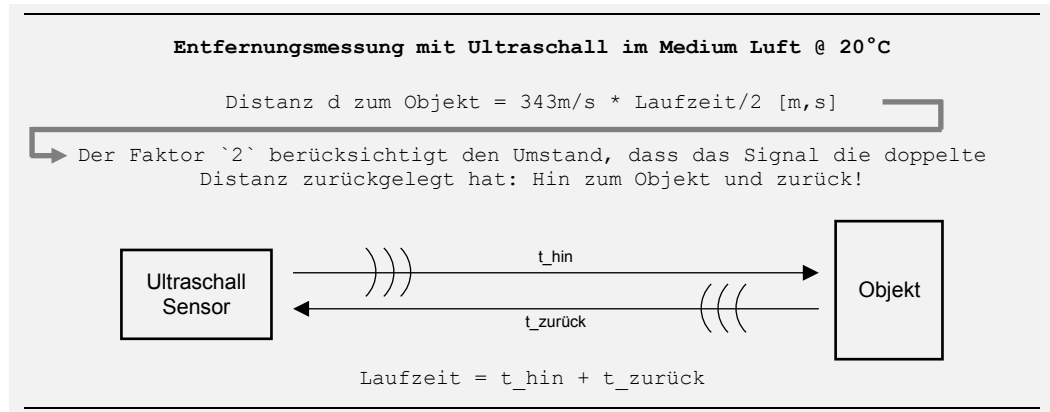


Abbildung 1-21 Entfernungsmessung mit Ultraschall im Medium Luft @ 20°C

Die Schallgeschwindigkeit (etwa 320 bis 350 m pro Sekunde; 343 m/s bei einer Lufttemperatur von 20°C) ist relativ langsam im Vergleich zur Verarbeitungsgeschwindigkeit moderner elektronischer Schaltkreise. Sie lässt sich daher relativ leicht elektronisch bestimmen. Probleme entstehen durch mangelhafte Reflexion am Messobjekt, durch störende Echos und durch Ultraschall-Signale fremder Quellen, weshalb die Ultraschall-Entfernungsmessung als nicht besonders zuverlässig gilt.

Ein Einsatzgebiet der Ultraschall-Sensorik stellt die Einparkhilfe im modernen Automobil dar. Autos, die damit ausgestattet sind, erkennt man an den runden Sensoren, die in die Stoßstange des Fahrzeugs eingebaut sind. Die Sensoren messen den Abstand zwischen Auto und einem vermeintlichen Hindernis. Die Distanz wird entweder rein akustisch oder in einer Kombination aus optischem und akustischem Signal dem Fahrer übermittelt und warnt somit vor einer möglichen Kollision.

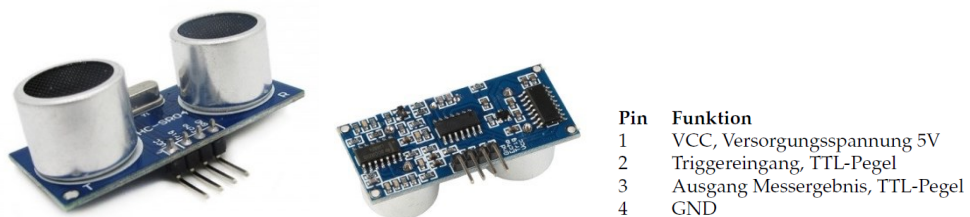


Abbildung 1-22 Ultraschall-Messmodul HC-SR04

Das hier eingesetzte Ultraschall-Modul HC-SR04 eignet sich zur Messung von Distanzen von 2cm bis ca. 3m. Die Auflösung beträgt ungefähr 3mm. Es benötigt nur eine einfache Versorgungsspannung von 5V bei einer Stromaufnahme von weniger als 2mA. Nach Triggerung mit einer fallenden Flanke (TTL - Pegel) misst das Modul selbstständig die Entfernung und wandelt diese in ein PWM Signal um welches am Ausgang zur Verfügung steht. Ein Messintervall hat eine Dauer von 20ms. Es können folglich 50 Messungen pro Sekunde durchgeführt werden.

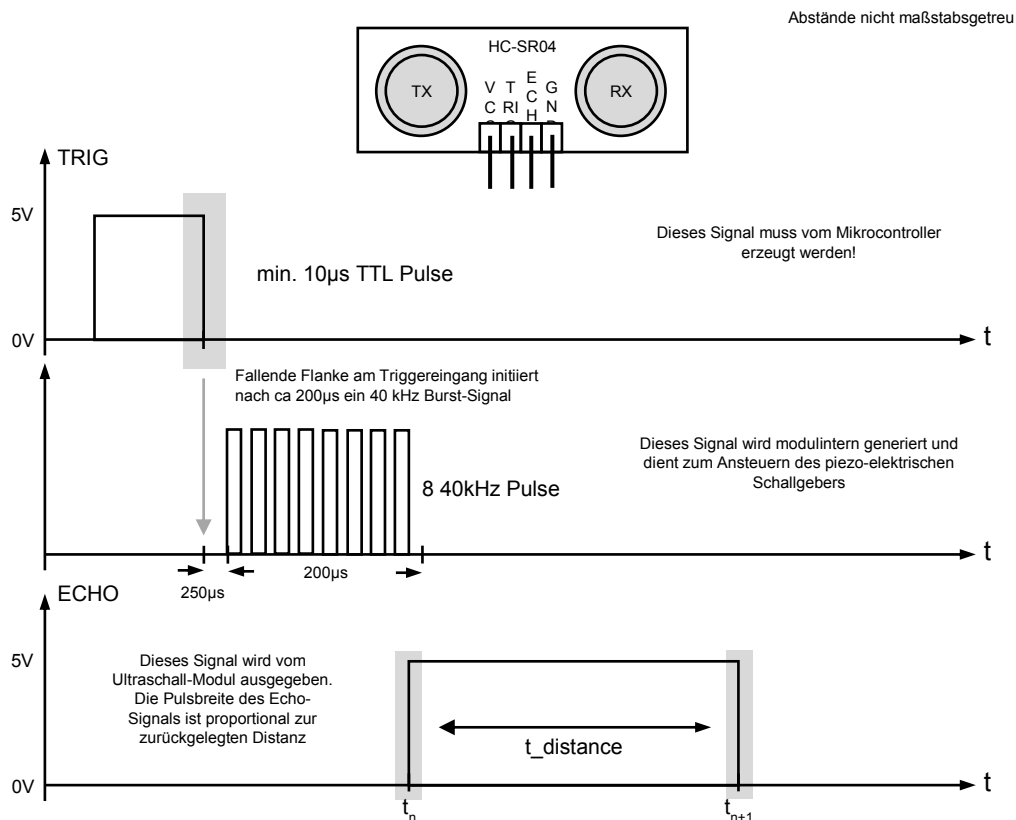


Abbildung 1-23 Signalformen - HC-SR04 Sensormoduls

Das Auslösen eines Messzyklus geschieht, wie in Abbildung 1.23 dargestellt, durch eine fallende Flanke am Trigger-Eingang (Pin 2). Hierbei gilt es zu beachten, dass das Trigger-Signal für mindestens 10µs stabil anliegt. Ist dies der Fall, so sendet das Ultraschall-Modul nach ca. 250µs ein 40 kHz und 200µs langes Burst-Signal. Danach geht der Echo-Ausgang (Pin 3) sofort auf H-Pegel und das Modul wartet auf den Empfang des Echos. Wird dieses detektiert fällt der Ausgang auf L-Pegel. 20ms nach Triggerung kann eine weitere Messung stattfinden. Wird kein Echo detektiert verweilt der Ausgang für insgesamt 200ms auf H-Pegel und zeigt so die erfolglose Messung an. Danach wartet das Modul auf die nächste fallende Flanke am Trigger-Eingang und die Messung beginnt neu.

Die besten Messergebnisse ergeben sich bei Reflektion an glatten, ebenen Flächen. Bei Distanzen bis 1m ist das Material der Fläche recht unkritisch. Der Winkel zum Objekt kann bei kurzen Distanzen von unter 1m bis etwa 45° betragen. Auch recht dünne Objekte werden zuverlässig erkannt.

Ein normaler Kugelschreiber z.B. lässt sich bis auf eine Distanz von ca. 30cm sicher erfassen. Bei der maximalen Distanz von 3m muss schon genau gezielt werden und es sollten keine anderen Gegenstände in ähnlicher Entfernung im Sendekegel von 15° vorhanden sein. Die Berechnung der Distanz erfolgt wie nachfolgend aufgezeigt.

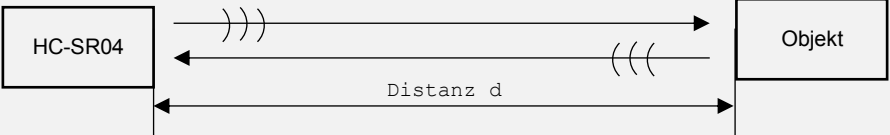
Ultraschall-Entfernungsmessung auf Basis des HC-SR04 Moduls im Medium Luft @ 20°C

$$((\text{Distanz } d \text{ zum Objekt}) * 2) / (\text{Pulsbreite in } \mu\text{s}) = 343\text{m/s}$$

Der Faktor `2` berücksichtigt den Umstand, dass das Signal die doppelte Distanz zurückgelegt hat: Hin zum Objekt und zurück!

$$34300\text{cm/s} = ((\text{Distanz } d \text{ zum Objekt}) * 2) / (\text{Pulsbreite in } \mu\text{s})$$

$$0.0343\text{cm}/\mu\text{s} = ((\text{Distanz } d \text{ zum Objekt}) * 2) / (\text{Pulsbreite in } \mu\text{s})$$

$$\text{Distanz } d \text{ zum Objekt (in cm)} = \text{Pulsbreite} / 58$$


Im vorliegenden Fall liegt die berechnete Zeitdifferenz in der Form von Timer Inkrementen (nachfolgend als **TTicks** bezeichnet) vor. Es wird angenommen, dass der Timer ohne vorgeschalteten Prescaler an der vollen Prozessorfrequenz von **16MHz** arbeitet. Hieraus ergibt sich folgende Berechnungsvorschrift:

$$((\text{Distanz } d \text{ zum Objekt}) * 2) / (\text{Pulsbreite in } \text{TTicks} * (1/16\text{MHz})) = 343\text{m/s}$$

$$((\text{Distanz } d \text{ zum Objekt}) * 2) / (\text{Pulsbreite in } \text{TTicks} * (1/16\text{MHz})) = 34300\text{cm/s}$$

Zu implementierende Gleichung

$$\text{Distanz } d \text{ zum Objekt (in cm)} = 0.001071875 * \text{TTicks}$$

Die Temperaturabhängigkeit der Schallgeschwindigkeit im Medium Luft kann im Bereich -20°C bis +40°C näherungsweise mit folgender Formel berechnet werden:

Temperaturabhängigkeit **der** Schallgeschwindigkeit im Medium Luft

$$\text{Schallgeschwindigkeit } v = 331.5 + 0.6 * T_U / ^\circ\text{C} \text{ [m, s]}$$

T_U bezeichnet die Umgebungstemperatur in °C.

Abbildung 1-24 Berechnungsbeispiel zur Entfernungsmessung mit Ultraschall im Medium Luft

Die programmtechnische Realisierung der Atmel ATmega328p basierten Pulsbreitenmessung zur Abstandsbestimmung mit dem zuvor vorgestellten HC-SR04 Ultraschall-Modul wird in Abbildung 1.25 näher beschrieben. Wie hier dargestellt erfordert die Messungen von Pulsweiten externer Signale die Umschaltung der Flankenerkennung nach jeder Erfassung. Es bietet sich zudem an, den Timer T/C1 mit Beginn einer Messung zurückzusetzen.

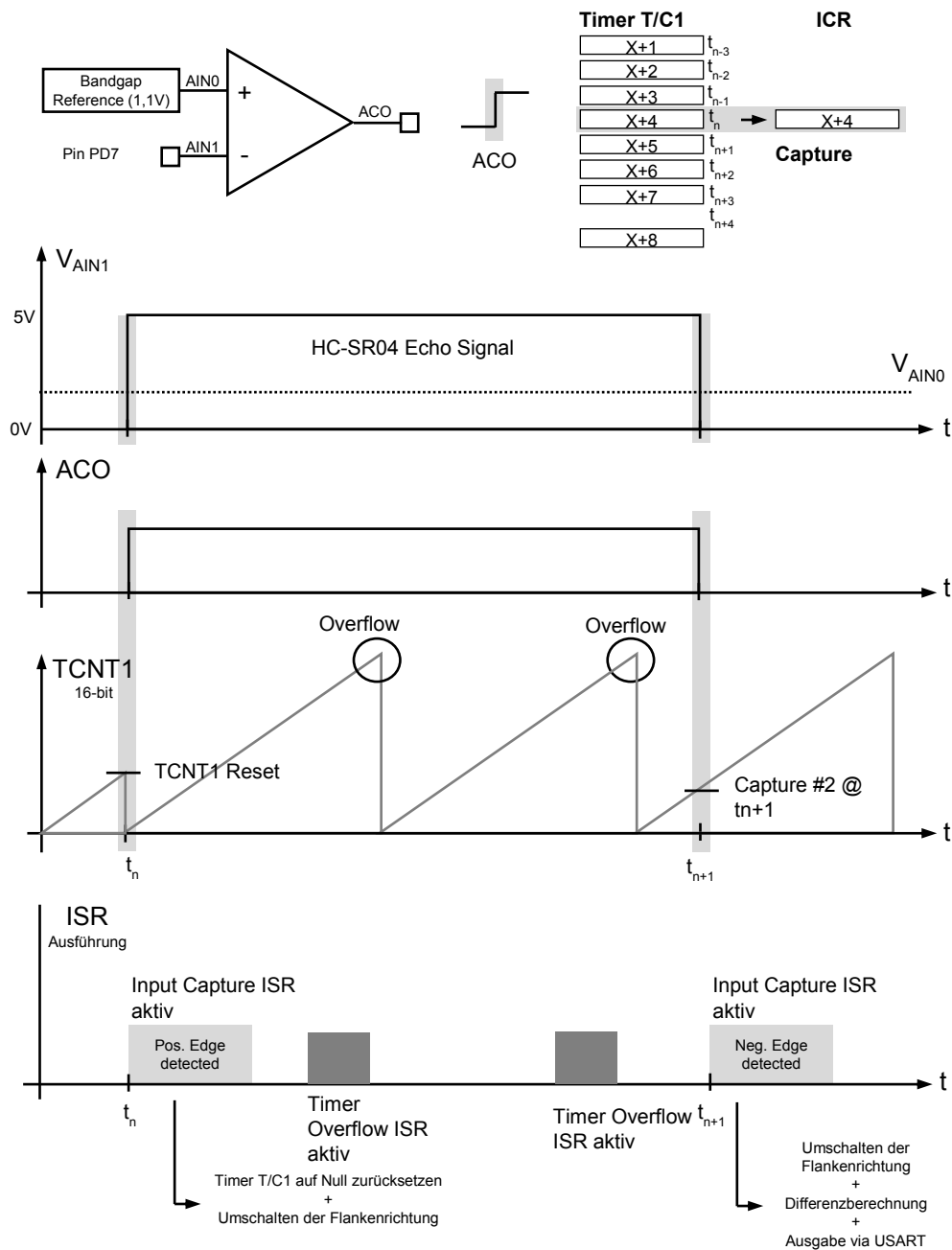


Abbildung 1-25 HC-SR04 Pulsbreitenmessung auf Basis der Input Capture Einheit in Kombination mit dem Atmel ATmega328p internen Analog-Komparator

Eine mögliche Implementierung der zuvor beschriebenen Funktionalität ist in Quellcode 1.5 dargestellt. Eine Beschreibung der einzelnen Programmschritte ist den Kommentaren zu entnehmen.


```

1. // Atmel ATmega328p - HC-SR04 Ultrasonic Distance Measurement auf Basis
2. // des Analog Komparator und der Input-Capture Unit. USART Ausgabe
3. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
4. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
5.
6. #define F_CPU 16000000UL
7.
8. #include <avr/interrupt.h>
9. #include <avr/io.h>
10. #include <util/delay.h>
11. #include <stdlib.h>
12.
13. #define BAUDRATE 9600
14. #define BAUD_PRESCALER (((F_CPU / (BAUDRATE * 16UL))) - 1)
15. // Realisierung der USART Funktionen finden Sie in den vor. Beispielen
16. void usart_init(void);
17. void usart_send(unsigned char data);
18. void usart_putstr(char* string_ptr);
19. void sonar_trigger(void);
20.
21. volatile long result = 0;
22. volatile unsigned char rising_edge = 0;
23. volatile unsigned char measurement_started = 0;
24. volatile unsigned short int Overflows = 0;
25. volatile unsigned short int stop_value = 0;
26.
27. ISR(TIMER1_CAPT_vect){
28.     if(measurement_started) {
29.         // accept interrupts only when sonar was started
30.         if((PINB & (1 << PB0)) != rising_edge){
31.             if(rising_edge == 0) {
32.                 // voltage rise, start time measurement
33.                 TCCR1B &= ~(1 << ICES1);
34.                 rising_edge = 1;
35.                 stop_value = ICR1;
36.                 Overflows = 0;
37.                 TCNT1 = 0; // reset timer counter
38.             } else {
39.                 TCCR1B |= (1 << ICES1);
40.                 rising_edge = 0;
41.                 // voltage drop, stop time measurement
42.                 stop_value = ICR1;
43.                 if(TIFR1 & (1 << TOV1)){
44.                     Overflows++;
45.                     TIFR1 &= ~(1 << TOV1);
46.                 }
47.                 result = (Overflows * 65536 + stop_value); ///58;
48.                 measurement_started = 0;
49.             }
50.         }
51.     }
52. }
53.
54. ISR(TIMER1_OVF_vect){
55.     if(rising_edge) // voltage rise was detected previously
56.         Overflows++; // count the number of overflows
57. }

```

Quellcode 1-5 Analog-Komparator/Input Capture-Einheit - Frequenzmessung Teil#1

```
58. // Atmel ATmega328p - HC-SR04 Ultrasonic Distance Measurement auf Basis
59. // des Analog Komparator und der Input-Capture Unit. USART Ausgabe
60. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
61. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
62.
63. int main(void){
64.     double erg;
65.     char message [12];
66.     char cm_unit [4] = " cm";
67.     // -- USART -
68.     usart_init();
69.     // -- Komparator-Stufe -
70.     DDRD  &= ~(1 << PD7); // AIN1 als Eingang konfigurieren...
71.     PORTD &= ~(1 << PD7); // ... und Pull-Up-Widerstand deaktivieren
72.     DIDR1 = (1 << AIN1D); // Digitale Eingangsstufe an AIN1 deaktivieren
73.     ACSR |= (1 << ACBG); // Vref an AIN0 auswählen
74.     _delay_ms(250); // Warten bis Vref stabil ...
75.     ACSR |= (1 << ACIC); // ACO - Input Capture Event
76.     // -- Input Capture Unit - T/C1 -
77.     TCCR1B = (1<<ICES1); // Input Capture Edge
78.     TCCR1B |= (1<<CS10); // No Prescaler
79.     TIMSK1 = (1<<ICIE1); // Capture Interrupt aktivieren
80.     TIMSK1 |= (1<<TOIE1); // Overflow Interrupt aktivieren
81.     // -- Trigger Output -
82.     DDRB |= (1 << PB5); // Trigger output
83.     PORTB &= ~(1 << PB5); // set to zero ...
84.
85.
86.     sei();
87.
88.     while(1){
89.         _delay_ms(1000);
90.         sonar_trigger();
91.         while(measurement_started != 0){
92.             // just wait ....
93.         }
94.         erg = (double)(result*0.0010625);
95.         dtostrf(erg,5,3,message);
96.         usart_putstring(message);
97.         usart_putstring(cm_unit);
98.         usart_send(10);
99.         // wait some time before starting the next measurement cycle ...
100.        _delay_ms(2000);
101.    }
102.    return 0;
103. }
104.
105. void sonar_trigger(void) {
106.     PORTB &= ~(1 << PB5); // clear to zero for 1 us
107.     _delay_us(1);
108.     PORTB |= (1 << PB5); // set high for 10us
109.     measurement_started = 1; // sonar launched
110.     _delay_us(10);
111.     PORTB &= ~(1 << PB5); // clear again ...
112. }
```

Quellcode 1-6 Analog-Komparator/Input Capture-Einheit - Frequenzmessung Teil#2

1.5 Übungsaufgaben



1. Erläutern Sie den Aufbau sowie die Funktion der ATmega328p internen Komparator-Stufe.
2. Wie in Abschnitt 1.2 erläutert, werden μ C-interne-Analog-Komparatoren primär eingesetzt um die Ausgangsspannung eines Sensors mit einer vorgegebenen Referenz zu vergleichen und gegebenenfalls in Abhängigkeit des Resultats bestimmte Steuerabläufe zu initiieren. Der Mikrocontroller agiert hier somit als eine Art Schwellenwertschalter. Einsatz findet dieses Verfahren in einer Vielzahl von Applikationen wie etwa
 - der Temperaturüberwachung auf Basis von LM35 Temperatursensoren
 - oder der Prüfung auf Überschreitung einer vorgegebenen Potentiometerstellung
 - ...

Prinzipiell könnte für die zuvor beschriebenen Anwendungen auch der im vorangegangenen Kapitel diskutiert 10-bit SAR Analog-Digital-Wandler eingesetzt werden. Eine fortlaufende Konvertierung der anliegenden Messspannung gefolgt, von einem Vergleich mit einem digitalen Referenzwert, ist funktional identisch mit der Nutzung der analogen Komparator-Stufe.

- Wo jedoch liegen die Nachteile dieses Verfahrens? Diskutieren sie diese Punkte am Beispiel eines Warnsystems zur Detektion eines kritischen Temperaturwertes.
3. Warum gilt es im Falle einer Benutzung der Pins `AIN0` und `AIN1`, die digitalen Eingangsstufen dieser Pins zu deaktivieren?
 4. Bauen Sie die Schaltung gemäß Abbildung 1.12 auf und implementieren Sie das in Quelltext 1.1 aufgeführte Beispielprogramm.
 5. Bauen Sie die Schaltung gemäß Abbildung 1.12 auf und implementieren Sie das in Quelltext 1.2 aufgeführte Beispielprogramm.
 6. Diskutieren Sie die beiden vorangegangenen Aufgaben eingesetzten Implementierungsformen. In welchen Applikationen bietet ihrer Meinung nach die Interrupt-basierte Programmausführung Vorteile gegenüber dem Polling-Verfahren?
 7. Ersetzen Sie das an Pin `PD7` angeschlossene Potentiometer durch eine sinusförmige Wechselspannungsquelle und führen Sie die in Aufgaben 4 und 5 umgesetzten Programme erneut aus.

8. Ändern Sie eines der beiden Beispielprogramme aus Aufgabe 4 oder 5 so ab, dass eine Unterschreitung der Referenzspannung durch eine mit 10Hz blinkende LED angezeigt wird.
9. An Pin PD7 liegt eine sinusförmige Wechselspannung an, die in ihrer Amplitude zwischen 0V und 5V variiert und eine Frequenz von 1kHz besitzt. Diese Spannung soll mit der internen 1.1V Band-Gap Referenz verglichen werden und bei Unterschreitung zu einem Auslösen des Komparator-Interrupt führen. Bestimmen Sie für dieses Szenario die resultierende Interrupt zu Interrupt Zeit.
10. Ziel dieser Aufgabe ist die Implementierung eines einfachen Dämmerungsschalters auf Basis eines lichtempfindlichen Widerstands (*LDR, engl. light dependent resistor*)¹². Beispielhaft soll beim Unterschreiten eines voreingestellten Helligkeitswertes eine rote LED eingeschaltet werden. Lichtempfindliche Widerstände, auch als Fotowiderstände bezeichnet, sind Halbleiterbauelemente, deren Widerstand sich bei Lichteinfall ändert. Bei Halbleitermaterialien führt nicht nur eine Erwärmung zur Erhöhung der Leitfähigkeit, sondern auch der Einfall von Licht. Licht aktiviert einzelne Elektronen und löst sie damit aus der molekularen Bindung, d.h. durch Lichtquanten überwinden die Elektronen die Energielücke vom Valenz- in das Leitungsband. Je höher der Lichteinfall, desto kleiner wird aufgrund des inneren fotoelektrischen Effekts sein elektrischer Widerstand.
 - Erarbeiten Sie ein Konzept für eine schaltungs- und programmtechnische Umsetzung der zuvor beschriebenen Funktionalität auf Basis des Atmel ATmega328p internen Analog-Komparators.
 - Verdeutlichen Sie ihre Ideen mit Hilfe von Aktivitäts- und Ablaufdiagrammen.
 - Wie kann die Einstellung eines bestimmten Dämmerungswertes schaltungstechnisch erfolgen?
 - Implementieren Sie das von Ihnen zuvor erarbeitete Konzept und verifizieren Sie die korrekte Funktionalität ihrer Lösung.
11. Für welche Anwendungen eignet sich die Benutzung des Analog-Komparators in Verbindung mit der Atmel ATmega328p internen Timer-Peripherie? Erläutern Sie in wenigen Worten die Funktion dieser Schaltungskonfiguration.

¹² Beispiel Datenblatt: Cadmium Sulfoselenide (CdS) Photoconductive Photocells, PHOTONIC DETECTORS INC

12. Warum wird der Einsatz einer Interrupt basierten Input-Capture-Einheit in einem Multi-Interrupt Betrieb als kritisch erachtet. Diskutieren Sie welche Punkte unter Umständen zur Verletzung der Echtzeitfähigkeit eines solchen Systems führen könnten.
13. Abbildung 1.26 zeigt die schaltungstechnische Realisierung einer einfachen Kapazitätsmessung auf Basis des ATmega328p internen Analogkomparators.

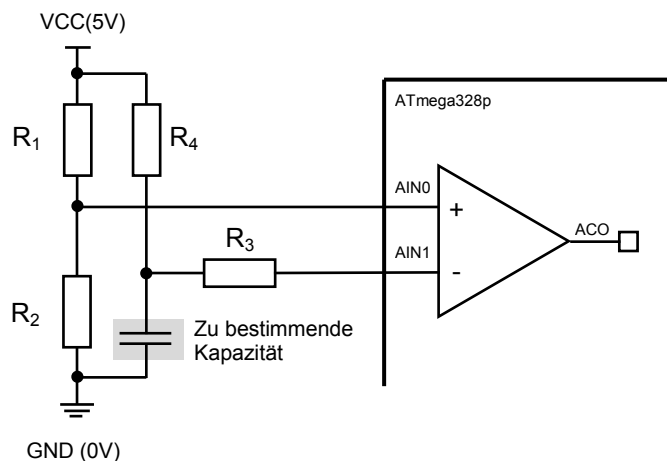


Abbildung 1-26 Atmel ATmega328p basierte Schaltung zur Kapazitätsmessung

Die Widerstände R_1 , R_2 und R_4 können von Ihnen frei bestimmt werden. Die Größe der Kapazität ist unbekannt und gilt es zu bestimmen. Ein Größenbereich von 1nF bis 100nF kann jedoch angenommen werden. Nutzen Sie für die Lösung dieser Aufgabenstellung die in diesem Kapitel diskutierten μ C-Schaltungstechniken.

- Beschreiben Sie die zugrunde liegende Idee der in Abbildung 1.26 dargestellten Schaltung.
- Beschreiben Sie die Funktion der Widerstände R_1 und R_2 innerhalb der Schaltung. **Hinweis:** Dimensionieren sie diese so, dass an Eingang AIN0 eine Spannung von $VCC/2$ anliegt.
- Welche Funktion besitzt der Widerstands R_3 innerhalb der Schaltung? **Hinweis:** Dimensionieren Sie diesen Widerstand mit 100Ω .
- Erarbeiten Sie ein Konzept für eine programmtechnische Umsetzung der hier geforderten Funktionalität. Verdeutlichen Sie Ihre Ideen mittels Aktivitäts- und Ablaufdiagrammen.
- Implementieren Sie das von Ihnen zuvor erarbeitete Konzept und verifizieren Sie die korrekte Funktionalität ihrer Lösung. Geben Sie die ermittelten Kapazitätswerte über die USART Schnittstelle auf einem PC-seitigen Terminal aus.

14. Das HC-SR04 Ultraschall-Modul wird häufig bei mobilen Roboterplattformen zur Ortung und zur Hindernisdetektion eingesetzt. Die nachfolgende Abbildung stellt exemplarisch die zeitlichen Verläufe der beiden HC-SR04 Signale TRIG (gelb, Kanal 1) und ECHO (türkis, Kanal 2) dar, die im Rahmen einer realen Messung aufgenommen wurden. Die Länge des Echo Signals beträgt 353.4 μs . Das Trigger-Signal ist hingegen für 25 μs gesetzt.

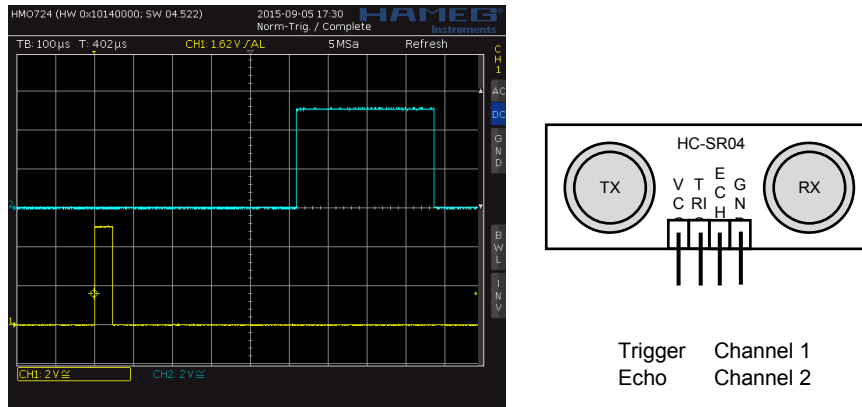


Abbildung 1-27 HC-SR04 Trigger- und Ausgangssignal

- Berechnen Sie für das zuvor dargestellte Szenario die Signallaufzeit sowie die resultierende Distanz zwischen dem HC-SR04 Ultraschall-Modul und dem detektierten Hindernis. Geben Sie die Signallaufzeit in μs sowie zusätzlich in äquivalenten Timer-Inkrementen an (Timer T/C1 @ 16MHz und ohne Prescaler). Die Distanz gilt es in cm anzugeben.
15. Ziel dieser Aufgabe ist die Entwicklung eines HC-SR04 basierten Ultraschall-Abstandswarners. Die Warnung erfolgt rein visuell über eine an Pin PB5 angeschlossene rote LED. Die Abstandsinformation soll über die Blinkfrequenz der LED dargestellt werden, d.h. die Frequenz der LED soll mit geringer werdendem Abstand zu einem detektierten Objekt zunehmen. Beträgt der Abstand weniger wie ein cm, so soll die LED statisch leuchten.
- Erarbeiten Sie ein Konzept für eine programmtechnische Umsetzung der hier geforderten Funktionalität. Verdeutlichen Sie Ihre Ideen mittels Aktivitäts- und Ablaufdiagrammen.
 - Implementieren Sie das von Ihnen zuvor erarbeitete Konzept und verifizieren Sie die korrekte Funktionalität ihrer Lösung.

16. Schreiben Sie eine Software zur Bestimmung der zeitlichen Länge eines an Pin PD7 anliegenden Rechtecksignals. Die ermittelte Pulsweite soll nach erfolgter Messung über die serielle Schnittstelle des ATmega328p an den PC übertragen werden. Als einstellbare Signalquelle soll ein Funktionsgenerator dienen.

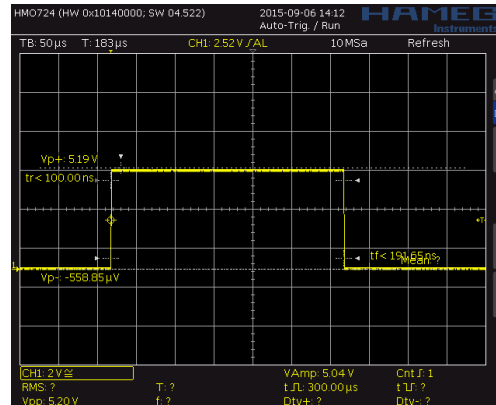
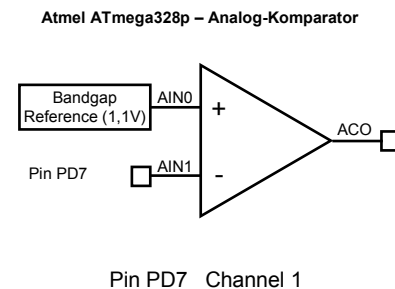


Abbildung 1-28 HC-SR04 Ausgangssignal



Das generierte Signal soll, wie in Abbildung 1.28 dargestellt, in seiner Amplitude zwischen 0V und 5V variieren. Die Pulsweite sei frei einzustellen.

- Erarbeiten Sie ein Konzept für eine programmtechnische Umsetzung der hier geforderten Funktionalität. Verdeutlichen Sie Ihre Ideen mittels Aktivitäts- und Ablaufdiagrammen.
- Die Messung soll über ein vom PC gesendetes Kommando gestartet werden.
- Nach erfolgter Messung soll die ermittelte Pulsweite in Timer Inkrementen an den PC übertragen werden.
- Implementieren Sie das von Ihnen zuvor erarbeitete Konzept und verifizieren Sie die korrekte Funktionalität ihrer Lösung.
- Ermitteln Sie die minimale Pulsbreite, die von ihrem System noch zuverlässig gemessen werden kann. Was limitiert die Performance ihrer Realisierung?
- Bestimmen Sie abschließend den relativen/absoluten Fehler ihres Messsystems.

2. EEPROM Datenspeicher

In vielen Mikrocontroller basierten Applikationen besteht die Notwendigkeit Daten aus einem Programm heraus zu speichern, so dass diese auch nach dem Abschalten der Versorgungsspannung erhalten bleiben und bei einem erneuten Systemstart wieder zur Verfügung stehen. Zwar verfügen Mikrocontroller der Atmel AVR Serie über mehrere vollintegrierte Speichermodule, jedoch eignen sich diese nur bedingt zur dauerhaften Speicherung von Laufzeitvariablen. Der interne 2 kbyte große SRAM-Speicher des Atmel ATmega328p ist ein sogenannter volatiler bzw. flüchtiger Speicher. Er verliert folglich mit dem Abschalten der Versorgungsspannung alle in ihm gespeicherten Daten (Laufzeitvariablen etc.) und ist somit als dauerhafter Datenspeicher denkbar ungeeignet. Gleiches gilt für den 32 kbyte großen ATmega328p internen Flash-Speicher¹³. Hierbei handelt es sich zwar um einen nichtflüchtigen Speichertyp, d.h. er behält auch nach Abschalten der Versorgungsspannung seinen Speicherinhalt, jedoch kann ein laufendes Programm¹⁴ auf diesen Bereich nicht schreibend zugreifen. Dies ist naheliegend, da andernfalls ein Programm sich selbst überschreiben könnte. Neben dem Flash- und SRAM-Speicher verfügen alle Derivate der Atmel ATmega Serie daher über einen integrierten EEPROM (*engl. für Electrically Erasable Programmable Read Only Memory*) Speicher zur dedizierten Ablage nichtflüchtiger Daten. Der Einsatz dieses 1024 byte großen Speichertyps ist zentrales Thema des aktuellen Kapitels und wird nachfolgend ausführlich behandelt.

Nach einer allgemeinen Einführung wird im zweiten Abschnitt dieses Kapitels auf die Konfiguration sowie die Programmierung des ATmega328p internen EEPROM-Speichers eingegangen. Neben der direkten Programmierung der AVR-Hardware werden in diesem Zusammenhang die EEPROM-Routinen der AVR-LibC Bibliothek des `avr-gcc` C-Compilers vorgestellt. Die zuvor beschriebenen Konzepte werden abschließend anhand konkreter Beispiele anschaulich aufgezeigt.

Nach dem Studium dieses Kapitels und dem Bearbeiten der Übungsaufgaben sollten Sie

- die grundlegende Funktion sowie den Aufbau des Atmel ATmega328p internen EEPROM-Speichers detailliert beschreiben können,
- in der Lage sein, den Atmel ATmega328p internen EEPROM-Speicher gemäß gegebener Vorgaben zu konfigurieren und in Betrieb zu nehmen,
- den EEPROM-Speicher des ATmega328p sowohl durch direkten Registerzugriff wie auch durch Benutzung der EEPROM-Routinen der AVR-LibC Bibliothek ansprechen können.

¹³ Der Flash-Speicher des ATmega328p dient als Programmspeicher

¹⁴ Hiervon ausgenommen ist der Zugriff auf die Boot Sektion des Flash-Speichers

2.1 Einführung

Neben einem vollintegrierten Befehlsspeicher in Flash-Technologie verfügt der Atmel ATmega328p über einen separaten 1024 byte großen EEPROM-Speicher. Hierbei handelt es sich um einen nichtflüchtigen Speichertyp, d.h. er hält die in ihm gespeicherte Information auch ohne anliegende Versorgungsspannung. Die englische Bezeichnung EEPROM steht für *“Electrically Erasable Programmable Read-Only Memory“* was auf Deutsch so viel bedeutet wie *“Elektronisch löschbarer sowie programmierbarer Festwertspeicher“*. Historisch und technologisch betrachtet stellt der EEPROM-Speicher den Nachfolger des EPROM-Speichers (*engl. für Erasable Programmable Read-Only Memory*) dar, den es noch mittels UV-Licht zu löschen galt.

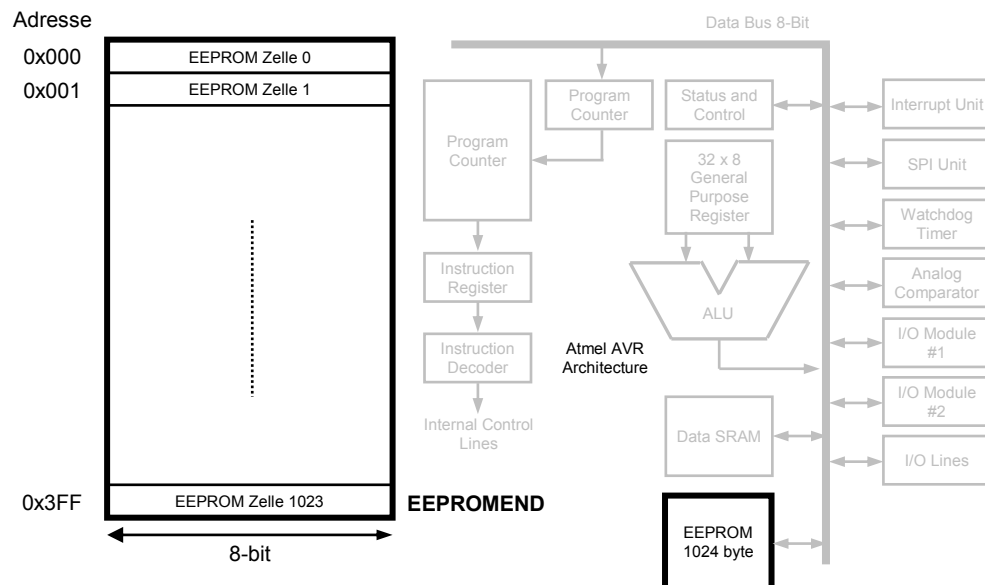


Abbildung 2-1 Atmel ATmega328p interner EEPROM-Speicher

Wie in Abbildung 2.1 dargestellt handelt es sich bei dem EEPROM im Atmel ATmega328p um einen byte-orientierten Speicher. Der Speicher kann folglich byteweise beschrieben und ausgelesen werden. Das Programmieren und Löschen des EEPROM-Speichers erfolgt ohne externes Programmiergerät in der Zielschaltung. Die für die Programmierung der Zellen notwendige Programmiervspannung wird intern über dedizierte Spannungswandler erzeugt. Gleiches gilt für die interne Ablaufsteuerung. Die Programmierzeit pro Byte beträgt in der Regel zwischen 3 und 4 Millisekunden und wird über speziell hierfür vorgesehene Timer festgelegt. Zudem verfügt der EEPROM-Speicher über eine sogenannte „Brown-Out“ Erkennung, die einen Schreibzugriff verhindert, falls die Betriebsspannung des Mikrocontrollers den notwendigen Mindestwert unterschreitet.

Der Zugriff auf den EEPROM-Speicher gestaltet sich aus Anwendersicht relativ einfach und erfolgt aus dem laufenden Programm heraus. Für die Programmierung eines Bytes muss die Adresse sowie das zu schreibende Byte selbst in spezielle Register geschrieben werden. Anschließend gilt es den Schreibvorgang über das Setzen dedizierter Steuerbits im Kontrollregister des EEPROM zu initiieren. Zu Beginn des Schreibzugriffs wird zuerst der Inhalt der adressierten Zelle gelöscht. Anschließend erfolgt das Beschreiben der Zelle mit dem neuen Wert. Beide Vorgänge laufen autonom ab und sind aus Anwendersicht nicht zugänglich. Kritisch ist in diesem Zusammenhang folgender Punkt: Kommt es während des Schreibvorgangs auf den EEPROM-Speicher zu einem System-Reset, so ist das Ergebnis undefiniert, da das Adressregister während des Zugriffs zurückgesetzt wird. Sowohl das Byte an der zu beschreibenden Adresse sowie an Adresse '0' des EEPROMs können somit fehlerhaft sein.

Zu beachten ist darüber hinaus, dass der EEPROM-Speicher nur eine begrenzte Anzahl von Schreibzugriffen erlaubt. Überschreitet man die im Datenblatt¹⁵ angegebene Anzahl von 100.000 Schreib-/Löschzyklen, wird die korrekte Funktion der EEPROM-Zelle nicht mehr garantiert. Diese Angabe bezieht sich auf jede einzelne Zelle des Speichers. Lesezugriffe auf den EEPROM-Speicher des ATmega328p können hingegen beliebig oft durchgeführt werden.

Aus Anwendersicht gilt es folglich die Anzahl der Schreib- bzw. Löschzyklen auf den EEPROM-Speicher des Atmel ATmega328p auf ein Minimum zu reduzieren. Ein sinnvoller Ansatz ist hier, die im Registersatz befindlichen Daten vor dem eigentlichen Abschalten des Mikrocontrollers in den EEPROM-Speicher zu kopieren. Hierfür ist jedoch eine Erweiterung der externen Beschaltung notwendig, um sicherzustellen, dass die Versorgungsspannung selbst nach Abschalten dergleichen für einen geraumen Zeitraum weiterhin stabil anliegt. Das Puffern der Versorgungsspannung kann z.B. durch große Elektrolyt- bzw. Doppelschicht-Kondensatoren oder einfache Lithiumzellen erfolgen. Die schaltungstechnische Auslegung der externen Beschaltung muss in allen Fällen gewährleisten, dass dem Mikrocontroller nach Abschalten der Versorgungsspannung noch genügend Zeit für das Kopieren der Daten in den EEPROM-Speicher zur Verfügung steht. In Anbetracht der Zugriffszeiten von mehreren Millisekunden stellt diese Aufgabe eine besondere Herausforderung dar, die sich mit steigender Anzahl zu speichernder Datenbytes weiter verschärft. Neben dem eigentlichen Kopiervorgang gilt es zudem das Abschalten der Versorgungsspannung verlässlich und zeitnah zu detektieren. Ein möglicher Lösungsansatz wird in Abbildung 2.2 aufgezeigt.

Die hier gezeigte externe Beschaltung basiert auf einem einfachen Elektrolytkondensator, der nach Abschalten der Versorgungsspannung den Mikrocontroller weiter mit Strom versorgt. Kommt es zum Abschalten der Versorgungsspannung, verhindert die Diode *D1* ein sofortiges Entladen des Kondensators *C1*.

¹⁵ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.20

Externe Beschaltung

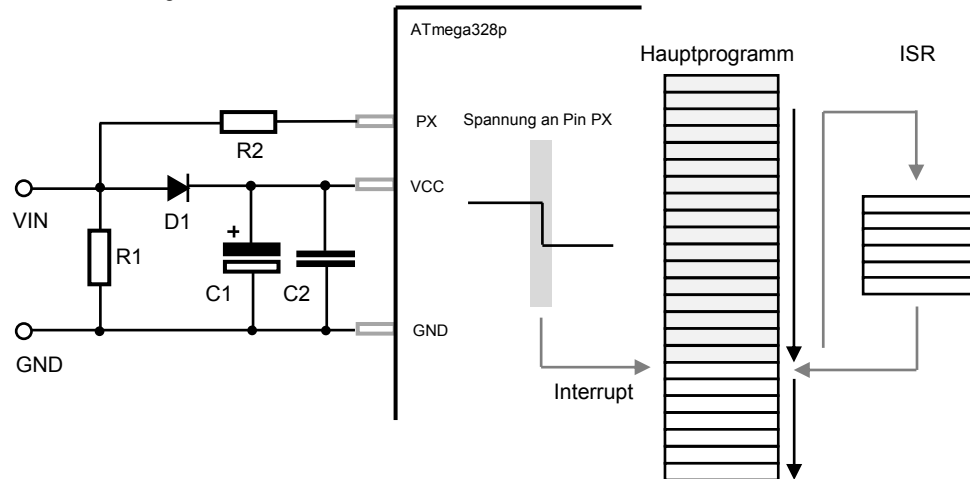


Abbildung 2-2 Externe Pufferschaltung zur Abschaltverzögerung und Sicherung möglicher Registerinhalte im EEPROM-Speicher des Atmel ATmega328

Durch den Wegfall der Versorgungsspannung kommt es zu einem negativen Flankenwechsel an Pin PX. Dieses Ereignis löst einen Pin-Change-Interrupt, der den eigentlichen Kopiervorgang initiiert. Der Widerstand $R1$ ist notwendig, um den Leckstrom der Diode in dieser Situation gegen Masse abzuführen. Der Widerstand $R2$ wird benötigt um die Spannung an Pin PX auf VCC abzusenken. Andernfalls käme es zu einem dauerhaften Stromfluss über die interne, nach VCC geschaltete Schutzdiode des Controllers was mit einer Zerstörung dergleichen einhergehen würde. Bei der Kapazität $C2$ handelt es sich um einen einfachen 100nF Koppelkondensator.

Nach Auslösen eines Pin-Change-Interrupts gilt es folglich umgehend alle relevanten Registerinhalte ins EEPROM zu kopieren. Die hierfür zur Verfügung stehende Zeit wird unter anderem durch die Größe des Elektrolytkondensators bestimmt und kann wie folgt berechnet werden:

$$Q = CU \rightarrow \frac{dQ}{dt} = C \frac{dU}{dt} \rightarrow I = C \frac{\Delta U}{\Delta t} \rightarrow \Delta t = T_P = C \frac{\Delta U}{I} \text{ bzw. } C = \frac{IT_P}{\Delta U}$$

Der Ausdruck T_P steht hier für die zur Verfügung stehende bzw. benötigte Zeit für den Kopiervorgang, I für die Stromaufnahme des Controllers und ΔU für den gemäß Datenblatt erlaubten Spannungsabfall. Bei einer Stromaufnahme von 10mA (5V @8MHz), einem möglichen Spannungsabfall ΔU von 2V und einem zu kopierenden Datensatz von 32 byte, ergibt sich bei einer durchschnittlichen EEPROM-Zugriffsdauer von 3.4ms eine Minimalkapazität von ungefähr 500µF! Mögliche Toleranzen und etwaige Temperaturabhängigkeiten des Elektrolytkondensators sowie eine erhöhte Stromaufnahme im Falle eines EEPROM-Schreibzugriffs blieben bei dieser Rechnung nicht berücksichtigt.

2.2 ATmega328p - EEPROM Speicher - Steuerung und Konfiguration

Für die Programmierung des EEPROM-Speichers des Atmel ATmega328p sind im Wesentlichen folgende Register relevant:

- Das EEPROM Adressregister **EEAR**,
- das EEPROM Datenregister **EEDR**,
- sowie das EEPROM Steuerregister **EECR**.

Alle drei Register besitzen eine Länge von 8-bit, mit Ausnahme des **EEAR** Adressregisters. In dieses Register muss die Adresse der EEPROM-Zelle eingetragen werden, die es zu lesen bzw. zu beschreiben gilt. Da der ATmega328p über einen 1024 byte großen EEPROM-Speicher verfügt, handelt es sich bei dem **EEAR** Register um ein 16-bit Register bei dem lediglich die unteren 10-bit zur Adressierung der EEPROM-Zelle genutzt werden. Wie in Abbildung 2.3 gezeigt kann das **EEAR** Register über die beiden 8-bit Doppel-Register **EEARH** und **EEARL** beschrieben werden.

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	-	-	-	-	-	-	EEAR9	EEAR8	EEARH
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	X	X	

Bit	7	6	5	4	3	2	1	0	
0x21 (0x41)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	

Abbildung 2-3 Atmel ATmega328p – EEPROM - Adressregister **EEAR**

Festzuhalten gilt hier, dass der Zustand der unteren 10-bit des **EEAR** Adressregisters nach einem Reset unbestimmt ist. Gilt es auf Adresse Null des EEPROM-Speichers nach einem Systemstart zuzugreifen, so muss folglich dieser Adresswert explizit ins **EEAR** Register geschrieben werden. Der eigentliche Datenaustausch erfolgt über das in Abbildung 2.4 dargestellte **EEDR** Datenregister. Bei einem Schreibzugriff auf den EEPROM-Speicher gilt es das zu schreibende Byte ins **EEDR** Register zu laden. Bei einem Lesezugriff wird hingegen der Inhalt der betreffenden EEPROM-Zelle in diesem Register abgelegt. In beiden Fällen wird die jeweilige Zelle durch den Inhalt des **EEAR** Register adressiert.

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

Abbildung 2-4 Atmel ATmega328p – EEPROM – Datenregister **EEDR**

Die Steuerung eines Schreib- bzw. Lesezugriffs auf den EEPROM-Speicher des Atmel ATmega328p erfolgt über das in Abbildung 2.5 abgebildete **EECR** Register.

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	-	-	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

Abbildung 2-5 Atmel ATmega328p – EEPROM – Steuerregister EECR

Wie hier dargestellt, handelt es sich hierbei um ein 8-bit breites Steuerregister bei dem lediglich die unteren 6 Bits belegt sind. Die Bedeutung sowie die Funktion dieser Bits soll nachfolgend kurz erläutert werden. Eine ausführliche Beschreibung der Registerfunktion ist der Herstellerdokumentation zu entnehmen¹⁶.

▪ Bit 5:4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits

Über das Setzen der sogenannten EEPROM Programming Mode Bits wird die Art des EEPROM-Schreibzugriffs festgelegt. Es ist zum einen möglich, Daten in einer sogenannten atomaren Operation in den Speicher zu schreiben. Das Löschen der adressierten Zelle sowie das nachfolgende Beschreiben werden hier zusammenhängend, d.h. in einer einzigen Operation ausgeführt. Darüber hinaus besteht, wie nachfolgend gezeigt, die Möglichkeit, diese beiden Vorgänge auf zwei unabhängige Operationen abzubilden. Die verschiedenen Konfigurationseinstellungen sind wie folgt kodiert:

EEPM1	EEPM0	Programmierzeit	Operation
0	0	3.4ms	Löschen & Schreiben in einer Operation
0	1	1.8ms	Lediglich Löschen
1	0	1.8ms	Lediglich Schreiben
1	1	-	Ohne Funktion

Tabelle 2-1 Atmel ATmega328p – EEPROM – Programming Modes

Die über die Bits EEPM1 und EEPM0 selektierte Aktion wird durch das Setzen des EEPROM Write Enable Signal Bits EEPE bei gleichzeitig gesetztem EEPROM Master Write Enable Bit EEMPE ausgeführt. Zudem sei vermerkt, dass bei gesetztem EEPE Bit jegliche Änderung der EEPROM Programming Mode Bits unberücksichtigt bleibt. Den Zustand dieses Bits gilt es folglich vor einer Änderung des EEPROM Programming Modes zu prüfen.

▪ Bit 3 – EERIE: EEPROM Ready Interrupt Enable

Ist das EEPROM Ready Interrupt Enable Bit EERIE gesetzt und sind zudem globale Interrupts im zentralen Statusregister SREG der AVR CPU freigeschaltet, so wird nach Beendigung eines Schreibzugriffs (mit dem Zurücksetzen des EWE Bits) ein Interrupt ausgelöst.

¹⁶ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.22

▪ **Bit 2 – EEMPE:** EEPROM Master Write Enable

Durch das Setzen des EEPROM Master Write Enable Bit `EEMPE` wird der EEPROM-Speicher des Atmel ATmega328p für einen nachfolgenden Schreibzugriff freigeschaltet. Das `EEMPE` Bit wird durch die interne EEPROM-Steuerung nach vier Prozessortakten zurückgesetzt. Innerhalb dieses Zeitraums muss folglich der Schreibzugriff auf den EEPROM-Speicher erfolgen. Dies geschieht wiederum durch das Setzen des `EEPE` Bits.

▪ **Bit 1 – EEPE:** EEPROM Write Enable

Durch das Setzen des EEPROM Write Enable Bit `EEPE` erfolgt bei gleichzeitig gesetztem `EEMPE` Bit der eigentliche Schreibzugriff auf den EEPROM-Speicher. Nach Beendigung des Speicherzugriffs wird das `EEPE` Bit durch die ATmega328p interne EEPROM-Steuerung automatisch zurückgesetzt. Mit dem Zurücksetzen des `EEPE` Bits wird zudem bei gesetztem `EERIE` Bit ein Interrupt ausgelöst¹⁷. Das `EEPE` Bit ist nach einem Systemstart undefiniert und muss vor der Freigabe des Schreibzugriffs gelöscht werden. Für das Schreiben von Daten in den EEPROM-Speicher wird gemäß Datenblatt¹⁸ folgende Abfolge empfohlen:

1. Möglichen Flash-Schreibzugriff abwarten¹⁹: `SPMEN` Bit im `SPMCSR` Register gleich Null?
2. EEPROM-Status prüfen: Ist das `EEPE` Bit gesetzt? Wenn ja, warten!
3. Zu beschreibende Adresse ins `EEAR` Register übertragen.
4. Zu schreibendes Datenbyte ins `EEDR` Register übertragen.
5. EEPROM für Schreibzugriffe freigeben: `EEMPE` Bit setzen.
6. Schreibvorgang innerhalb der nächsten vier Prozessortaktzyklen auslösen: `EEPE` Bit setzen.
7. (Optional) Auf das Ende des Schreibvorgangs warten (3ms - 4ms).

Wie hier zu sehen ist, nimmt ein schreibender Zugriff auf den EEPROM-Speicher relativ viel Zeit in Anspruch. Es gilt daher immer zu prüfen, dass ein vorangegangener EEPROM-Zugriff abgeschlossen wurde bevor der nächste Schreibzugriff gestartet wird.

▪ **Bit 0 – EERE:** EEPROM Read Enable

Durch das Setzen des EEPROM Read Enable Bit `EERE` erfolgt ein lesender Zugriff auf den EEPROM-Speicher des Atmel ATmega328p. Die zu lesende EEPROM-Zelle wird hierbei durch das `EEAR` Register adressiert. Der Inhalt der betreffenden Zelle wird abschließend ins `EEDR` Register übertragen und kann von dort nachfolgend ausgelesen werden.

¹⁷ `EE_READY_vect`

¹⁸ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.20

¹⁹ Nur relevant, wenn die Software Zugriffe auf die Boot Sektion des Flash-Speichers erlaubt

Ein Lesezugriff ist nicht möglich, wenn bereits zuvor eine Schreiboperation auf den EEPROM-Speicher gestartet wurde. Werden während eines noch laufenden Schreibvorgangs neue Daten sowie Adressen in die entsprechenden EEPROM-Steuerregister geschrieben, so wird der noch laufende Schreibzyklus unterbrochen und sein Ergebnis ist undefiniert. Es gilt folglich vor jedem Lesezugriff den Status des EEPROM-Speichers zu prüfen. Auch hier kann der Status des EEPROM-Speichers durch einfaches Auslesen des `EEPE` Bits ermittelt werden. Ein Lesevorgang ist folglich möglich wenn das `EEPE` Bit nicht gesetzt ist.

Nach Abschluss des Lesevorgangs wird das `EERE` Bit automatisch zurückgesetzt und der EEPROM-Speicher des ATmega328p steht für weitere Lese- bzw. Schreibvorgänge bereit. Im Vergleich zu einem Schreibzugriff ist hier ein Abfragen des `EERE` Bits nicht notwendig, da der Lesevorgang lediglich einen Systemtakt in Anspruch nimmt. Für das Lesen von Daten aus dem EEPROM-Speicher des ATmega328p wird seitens des Herstellers folgende Abfolge empfohlen²⁰:

1. EEPROM-Status prüfen: Ist das `EEPE` Bit gesetzt? Wenn ja, warten!
2. Zu lesende Adresse ins `EEAR` Register übertragen.
3. Lesevorgang auslösen: `EERE` Bit setzen.
4. EEPROM Byte aus dem `EEDR` Datenregister auslesen.

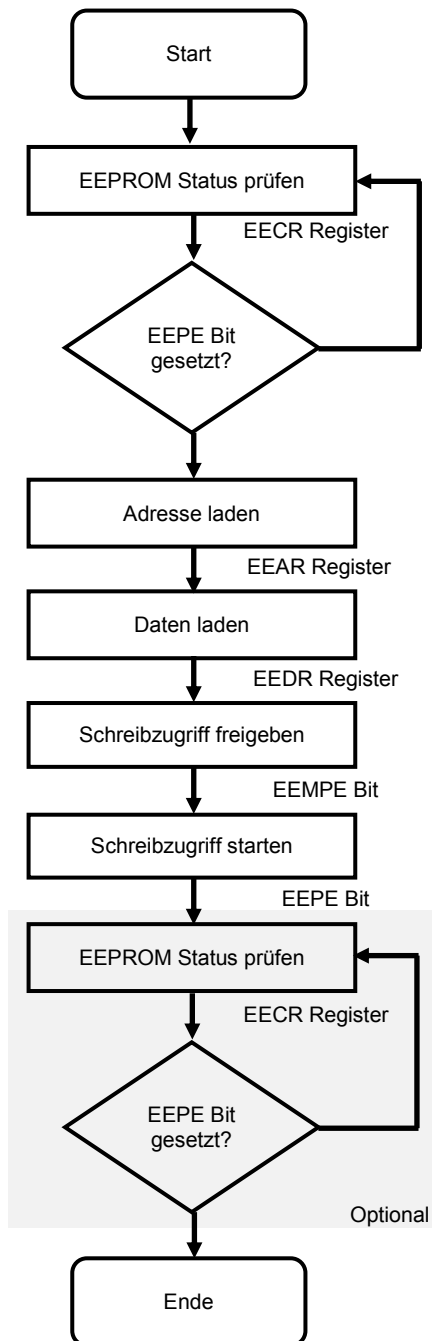
Die zuvor beschriebenen Verfahren zum Lesen und Beschreiben des EEPROM-Speichers setzen voraus, dass während der Schreib- bzw. Lese-Phase keine Interrupts auftreten. Hintergrund ist, dass einzelne Phasen des EEPROM-Zugriffs innerhalb einer definierten Anzahl von Prozessortakten erfolgen müssen. So muss nach Freischaltung der EEPROM-Schreibzugriffe durch das Setzen des `EEPE` Bit der eigentliche Schreibvorgang innerhalb der nächsten vier Prozessortaktzyklen durch das Setzen des `EEPE` Bit ausgelöst werden. Tritt innerhalb dieser Zeit ein Interrupt auf, schlägt der Schreibzugriff auf den EEPROM-Speicher fehl. Daher gilt es Interrupts vor Ausführung der zuvor beschriebenen EEPROM-Sequenzen global zu deaktivieren.

Des Weiteren muss sichergestellt werden, dass mit dem Start einer EEPROM-Operation jegliche Schreibzugriffe auf den Bootbereich des Flash-Speichers abgeschlossen sind. Andernfalls kann, ähnlich wie bei einem unterbrechenden Interrupt, ein fehlerfreier Zugriff auf den EEPROM-Speicher nicht garantiert werden. Dieser Punkt ist jedoch lediglich relevant, wenn die implementierte Software eine Boot-Loader-Funktionalität besitzt und somit Schreibzugriffe seitens der CPU auf den Boot-Bereich des Flash-Speichers erfolgen können.

²⁰ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.20

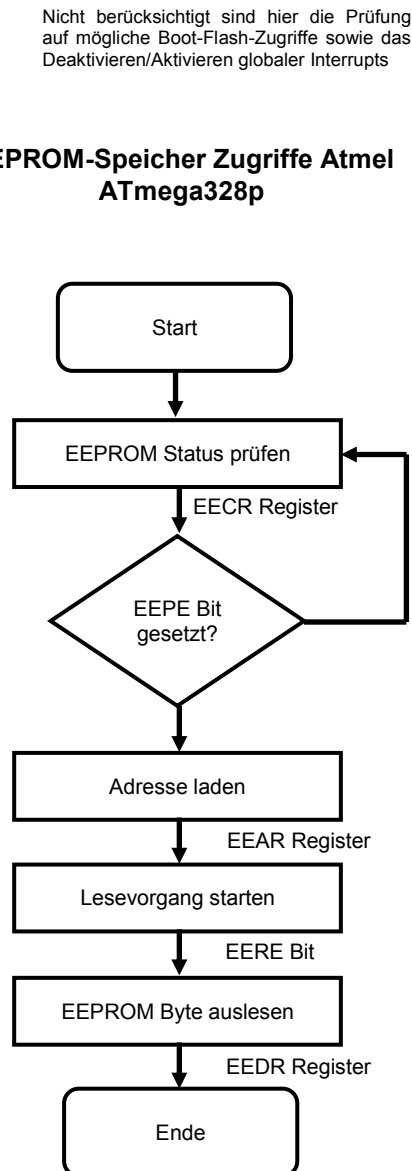
2.3 ATmega328p - EEPROM Speicher - Praxisbeispiele - Teil #1

Im folgenden Abschnitt soll nun die Benutzung des Atmel ATmega328p internen EEPROM-Speichers anhand konkreter Beispiele demonstriert werden. Im Wesentlichen gilt es Funktionen für den schreibenden bzw. lesenden Zugriff auf den EEPROM-Speicher zu entwerfen. Betrachten wir hierzu die nachfolgend aufgeführten Ablaufdiagramme.



EEPROM Schreibzugriff

EEPROM-Speicher Zugriffe Atmel ATmega328p



EEPROM Lesezugriff

Abbildung 2-6 Atmel ATmega328p – EEPROM – Ablaufdiagramme Schreib-/Lesezugriff

Eine mögliche Implementierung der in Abbildung 2.6 beschriebenen Funktionalität ist im nachfolgenden Quellcode dargestellt. Eine Beschreibung der einzelnen

Programmschritte ist den Kommentaren zu entnehmen. Es sei lediglich angemerkt, dass es sich bei der EEPROM-Schreibroutine um eine blockierende Funktion handelt, d.h. sie wartet aktiv auf das Ende eines Schreibvorgangs. Diese Prüfung könnte alternativ auch durch die aufrufende Funktion erfolgen. Die hier gezeigten Assembler-Routinen setzen zudem global deaktivierte Interrupts voraus. Für die Ausführung der Funktionen ist zudem das Einbinden der Definitionsdatei `avr/io.h` notwendig.

```

1. // Atmel ATmega328p - EEPROM Read/Write Routines in AVR Assembler
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. ; --- ATmega328p EEPROM Schreibroutine ---
6. ;     - Zu beschreibende EEPROM Adresse: r18:r17
7. ;     - Zu schreibendes Datenwort:      r16
8.
9. EEPROM_WRITE_ACCESS:
10.    ; Auf den Abschluss eines vorangegangenen EEPROM Zugriffs warten ...
11.    sbic    EECR, EEPE
12.    rjmp    EEPROM_WRITE_ACCESS
13.    ; Zu beschreibende Adresse laden (r18:r17)
14.    out     EEARH, r18
15.    out     EEARL, r17
16.    ; Zu schreibende Daten ins EEDR Register laden (r16)
17.    out     EEDR
18.    ; Schreibzugriff freischalten
19.    sbi     EECR, EEMPE
20.    ; EEPROM Schreibzugriff starten ...
21.    sbi     EECR, EEPE
22. EEPROM_WAIT:
23.    ; Auf den Abschluss des EEPROM Zugriffs warten (OPTIONAL) ...
24.    sbic    EECR, EEPE
25.    rjmp    EEPROM_WAIT
26.    ret
27.
28. ; --- ATmega328p EEPROM Leseroutine ---
29. ;     - Zu lesende EEPROM Adresse:      r18:r17
30. ;     - Rückgabe des gelesenen Datenwort: r16
31.
32. EEPROM_READ_ACCESS:
33.    ; Auf den Abschluss eines vorangegangenen EEPROM Zugriffs warten ...
34.    sbic    EECR, EEPE
35.    rjmp    EEPROM_READ_ACCESS
36.    ; Zu lesende Adresse laden (r18:r17)
37.    out     EEARH, r18
38.    out     EEARL, r17
39.    ; EEPROM Lesezugriff starten ...
40.    sbi     EECR, EERE
41.    ; gelesene Daten ins AVR CPU Register r16 transferieren
42.    in      r16, EEDR
43.    ret

```

Quellcode 2-1 Atmel ATmega328p – EEPROM – Lese- und Schreibroutine in AVR Assembler

Alternativ kann die Kodierung der in Quellcode 2.1 gezeigten Funktionen auch in ANSI C erfolgen. Eine mögliche Realisierung eines einfachen Schreib- und Lesezugriffs auf den EEPROM-Speicher des ATmega328p ist nachfolgend dargestellt. Wie zuvor handelt es sich bei der EEPROM-Schreibroutine um eine blockierende Funktion, d.h. sie wartet aktiv auf das Ende eines Schreibvorgangs. Auch an dieser Stelle könnte diese Prüfung durch die aufrufende Funktion erfolgen. Zudem wird auch hier die Deaktivierung globaler Interrupts vorausgesetzt. Für die Ausführung der Funktionen wird hier ebenfalls das Einbinden die Definitionsdatei `avr/io.h` vorausgesetzt.

```
1. // Atmel ATmega328p - EEPROM Read/Write Routines in ANSI C
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. void eeprom_write_access(unsigned int address, unsigned char data){
6.     // Auf den Abschluss eines vorangegangenen EEPROM Zugriffs warten ...
7.     while(EECR & (1 << EEPE));
8.     // Zu beschreibende Adresse laden
9.     EEAR = address;
10.    // Zu schreibende Daten laden
11.    EEDR = data;
12.    // Schreibzugriff freischalten
13.    EECR |= (1 << EEMPE);
14.    // Schreibzugriff starten
15.    EECR |= (1 << EEPE);
16.    // Auf den Abschluss des EEPROM Zugriffs warten (OPTIONAL)...
17.    while(EECR & (1 << EEPE));
18. }
19.
20. unsigned char eeprom_read_access(unsigned int address){
21.    // Auf den Abschluss eines vorangegangenen EEPROM Zugriffs warten ...
22.    while(EECR & (1 << EEPE));
23.    // Zu lesende Adresse laden
24.    EEARH = address;
25.    // EEPROM Lesezugriff starten ...
26.    EECR |= (1 << EERE);
27.    // gelesene Daten zurückgeben ...
28.    return EEDR;
29. }
```

Quellcode 2-2 Atmel ATmega328p – EEPROM – Lese- und Schreibroutine in ANSI C

Soll vor dem Aufruf der EEPROM-Schreibroutine sichergestellt werden, dass diese nicht intern in einer Warteschleife auf den Abschluss des vorherigen Zugriffs wartet, kann vorher der Status des EEPROM ermittelt werden, und somit geprüft werden, ob ein EEPROM-Schreibzugriff umgehend möglich ist. Eine Realisierung dieser Funktionalität ist in Quellcode 2.3 dargestellt.

```

1. // Atmel ATmega328p - EEPROM 'Check State' Routine in ANSI C
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. unsigned char eeprom_check_state (void){
6.     // Findet momentan ein Speicherzugriff statt?
7.     return (EECR & (1 << EEPE));
8. }
9.

```

Quellcode 2-3 Atmel ATmega328p- EEPROM – Status-Prüfroutine

Die zuvor gezeigten Funktionen können nun genutzt werden, um komplexere Schreib- bzw. Lesezugriffe auf den EEPROM-Speicher des ATmega328p zu realisieren. Nachfolgend ist in Quellcode 2.4 die Implementierung zweier Funktionen zum Schreiben bzw. Lesen eines Charakter-Strings gezeigt. Beide Funktionen setzen einen nullterminiert String sowie das Einbinden der in Quellcode 2.2 erstellten Funktionen voraus.

```

1. // Atmel ATmega328p - EEPROM String Read/Write Routines in ANSI C
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. void eeprom_write_string(unsigned int address, unsigned char *ptr){
6.     // Nullterminierter String!
7.     while(*ptr){
8.         eeprom_write_access(address,*ptr);
9.         address++;
10.        ptr++;
11.    }
12. }
13.
14. void eeprom_read_string(unsigned int address, unsigned char *ptr,
15.                          unsigned int string_length){
16.     // Ende erreicht?
17.     while(string_length){
18.         *ptr= eeprom_read_access(address);
19.         if(*ptr != '\0'){
20.             address++;
21.             string_length--;
22.             ptr++;
23.         }
24.         else
25.             return;
26.     }
27.

```

Quellcode 2-4 Atmel ATmega328p- EEPROM – Character String Lese- und Schreibroutine

Wie im vorangegangenen Abschnitt erläutert, lassen EEPROM-Speicher nur eine begrenzte Anzahl an Schreibzugriffen zu. Laut Angaben des Herstellers liegt dieser Wert im Falle des ATmega328p bei rund 100.000 Schreib/Löschzyklen. Ein Ziel ist somit immer aus Anwendersicht die Anzahl der schreibenden EEPROM-Zugriffe auf ein Minimum zu beschränken.

Eine Möglichkeit, Schreibzyklen einzusparen, besteht in der Prüfung, ob der zu speichernde Wert bereits in der adressierten Zelle enthalten ist. Ist dies der Fall, so gilt es den Schreibzugriff zu verwerfen. Durch dieses Verfahren wird sichergestellt, dass nur im Falle unterschiedlicher Werte ein Schreibzugriff erfolgt. Der damit einhergehende zeitliche Overhead ist minimal und kann in Anbetracht der Dauer eines EEPROM-Schreibzugriffs vernachlässigt werden. Eine mögliche Realisierung dieses Verfahrens ist in Quellcode 2.5 gezeigt.

```
1. // Atmel ATmega328p - EEPROM Update Funktion in ANSI C
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. void eeprom_update_access(unsigned int address, unsigned char data){
6.     unsigned char current_eeprom_byte;
7.     // Auf den Abschluss eines vorangegangenen EEPROM Zugriffs warten ...
8.     while(EECR & (1 << EEPE));
9.     // Betreffende EEPROM Adresse laden
10.    EEARH = address;
11.    // EEPROM Lesezugriff starten ...
12.    EECR |= (1 << EERE);
13.    // gelesene Daten zurückgeben ...
14.    current_eeprom_byte = EEDR;
15.    // betreffendes EEPROM byte prüfen ...
16.    if(current_eeprom_byte != data) {
17.        // Zu beschreibende Adresse laden (eigentlich nicht notwendig)
18.        EEAR = address;
19.        // Zu schreibende Daten laden
20.        EEDR = data;
21.        // Schreibzugriff freischalten
22.        EECR |= (1 << EEMPE);
23.        // Schreibzugriff starten
24.        EECR |= (1 << EEPE);
25.        // Auf den Abschluss des EEPROM Zugriffs warten (OPTIONAL)...
26.        while(EECR & (1 << EEPE));
27.    }
28. }
```

Quellcode 2-5 Atmel ATmega328p- EEPROM – Update-Funktion

2.4 ATmega328p - EEPROM Speicher - Praxisbeispiele - Teil #2

Im folgenden Abschnitt soll kurz auf die Benutzung der, in der AVR-Libc definierten, EEPROM-Routinen eingegangen werden. Bei der AVR-Libc handelt es sich um die weitverbreitetste Laufzeitbibliothek für die Atmel AVR Architektur. Neben Controller-spezifischen Headerdateien stellt diese Bibliothek eine Reihe von Funktionen für den effizienten Zugriff auf die ATmega interne Peripherie zur Verfügung. Im Folgenden liegt der Fokus auf dem Bibliotheksmodul `avr/eeprom.h` der AVR-Libc. Eine ausführliche Beschreibung der weiteren Bibliotheksfunktionen ist der offiziellen Dokumentation zu entnehmen²¹.

²¹ Homepage: <http://www.nongnu.org/avr-libc/>

Die AVR-Libc Bibliothek verfügt über eine Reihe von Zugriffsroutinen für den ATmega328p internen EEPROM-Speicher. Um diese in einer Applikation zu nutzen, gilt es die entsprechende Headerdatei über die Präprozessordirektive `#include <avr/eeprom.h>` einzubinden. Dem Anwender stehen dann die nachfolgend aufgeführten Funktionen für den EEPROM-Zugriff.

```

1.
2.  // --- AVR-Libc EEPROM-Routinen (avr/eeprom.h) ---
3.
4.  // #1 Byte: 8-bit EEPROM access
5.  uint8_t eeprom_read_byte(const uint8_t *addr);
6.  void eeprom_write_byte(uint8_t *addr, uint8_t value);
7.  void eeprom_update_byte(uint8_t *addr, uint8_t value);
8.
9.  // #2 Word: 16-bit EEPROM access
10. uint16_t eeprom_read_word(const uint16_t *addr);
11. void eeprom_write_word(uint16_t *addr, uint16_t value);
12. void eeprom_update_word(uint16_t *addr, uint16_t value);
13.
14. // #3 DWord: 32-bit EEPROM access
15. uint32_t eeprom_read_dword(const uint32_t *addr);
16. void eeprom_write_dword(uint32_t *addr, uint32_t value);
17. void eeprom_update_dword(uint32_t *addr, uint32_t value);
18.
19. // #4 Float: 32-bit EEPROM access
20. float eeprom_read_float(const float *addr);
21. void eeprom_write_float(float *addr, float value);
22. void eeprom_update_float(float *addr, float value);
23.
24. // #5 Block: user defined EEPROM access
25. void eeprom_read_block(void *ptr_ram, const void *ptr_eeprom, size_t n);
26. void eeprom_write_block(const void *ptr_ram, void *ptr_eeprom, size_t n);
27. void eeprom_update_block(const void *ptr_ram, void *ptr_eeprom, size_t n);
28.
29. // EEPROM Memory Attribute
30. #define EEMEM attribute((section(".eeprom")))

```

Quellcode 2-6 Atmel ATmega328p– EEPROM – AVR-Libc EEPROM Routinen

Jede der zuvor aufgeführten fünf Befehlsklassen stellt drei verschiedene Zugriffsvarianten zur Verfügung: Eine lesende, eine schreibende sowie eine Update-Funktion. Die Länge der verwendeten Variablentypen ist durch den `avr-gcc` Compiler definiert und entspricht den oben gemachten Angaben.

Neben den EEPROM-Zugriffsroutinen stellt die AVR-Libc Bibliothek das `EEMEM` Makro zur Verfügung, um Variablen im EEPROM-Speicher anzulegen. Der in Quellcode 2.6 hierfür definierte Ausdruck ist wie folgt zu interpretieren: Das `_attribute_ ((section (".eeprom")))` ist eine Besonderheit der AVR gcc Toolchain. Mit einem `_attribute_` werden Bezeichner im Programm mit einem Kennzeichen (einem Attribut) versehen. Das in Quellcode 2.6 verwendete `section(".eeprom")` Attribut bezeichnet offensichtlich einen Bereich im EEPROM-Speicher des Mikrocontrollers.

Einer mit diesem Attribut versehenen Variablen wird folglich ein Bereich im EEPROM-Speicher zugewiesen. Andere Sektionen sind `.text` für Programmcode im Flash sowie `.data` und `.bss` für Programmdaten im RAM. Diese Attribute können jedoch bei der Erstellung des Quelltextes weggelassen werden, da in diesem Fall die Zuweisung durch den Compiler selbst erfolgt. Für eine ausführliche Behandlung dieses Themas wird an dieser Stelle auf die Compiler-Dokumentation verwiesen²².

Zudem ist es möglich im EEPROM angelegte Variablen zu initialisieren und mit Standardwerten zu belegen. Hierfür muss der `gcc` Compiler eine sogenannte `.eep` Datei (Intel Hex-Format) erzeugen, die gemeinsam mit dem eigentlichen Programm (`.hex` Datei) an den Mikrocontroller übertragen wird. Setzt man das Programm `avrdude` zum Programmieren des Controllers ein, so gilt es den Programmaufruf mit dem nachfolgenden Zusatz zu versehen:

```
-U eeprom:w:"$(ProjectDir)Debug\$(ItemFileName).eep":i
```

An dieser Stelle soll jedoch darauf hingewiesen werden, dass der Arduino Uno Bootloader keine Zugriffe auf das interne EEPROM unterstützt. Eine Umsetzung der zuvor diskutierten Konzepte ist im nachfolgenden Quellcode aufgezeigt.

```
1. // Atmel ATmega328p - Short EEPROM Demo
2. // Zielplattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, August 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #define BAUDRATE 9600
7. #define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)
8. #define EEMEM __attribute__((section(".eeprom")))
9.
10. #include <avr/eeprom.h>
11. #include <util/delay.h>
12. #include <avr/io.h>
13. #include "usart_routines.h"
14.
15. unsigned char test_byte EEMEM = 0xCD;
16.
17. int main(void){
18.     unsigned char test;
19.     usart_init();
20.     while(1){
21.         eeprom_write_byte((unsigned char*)0x02,0xAB);
22.         test = eeprom_read_byte((unsigned char *)0x02);
23.         usart_send(test);
24.         test = eeprom_read_byte(&test_byte);
25.         usart_send(test);
26.     }
27.     return 0;
28. }
```

Quellcode 2-7 Atmel ATmega328p– EEPROM – Demo Programm #1

²² Homepage: <http://www.nongnu.org/avr-libc/>

2.5 Übungsaufgaben



17. Erläutern Sie in wenigen Worten die grundlegende Charakteristik der drei, im ATmega328p verwendeten Speichertypen.
18. Warum sollten unnötige Schreibzugriffe auf den EEPROM-Speicher unbedingt vermieden werden.
19. Erläutern Sie das in Abschnitt 2.1 vorgestellte Konzept zur Sicherung der Registerinhalte im Falle des Abschaltens der Versorgungsspannung. Berechnen Sie zudem die notwendige Kapazität für folgendes Szenario: ATmega328p@1MHz, 5V, 1mA Stromaufnahme. Mit welchem Spannungsabfall können Sie gemäß Datenblatt maximal rechnen? Es sollen alle Register gesichert werden.
20. Wo sehen Sie bei dem Vorgehen aus Aufgabe '3' mögliche Limitierungen bzw. Nachteile? Durch welche schaltungstechnischen Maßnahmen könnte diese behoben werden?
21. Erklären Sie den Unterschied zwischen einem EEPROM-Schreibzugriff und einem EEPROM-Speicherupdate.
22. Starten Sie das Programm „Atmel Studio 6“ und erstellen Sie ein neues „GCC Executable“ Projekt mit dem Namen „test_eeprom“. Wählen sie als Zielpattform den ATmega328p Mikrocontroller aus der „megaAVR 8-bit“ Device-Kategorie.

```

1. // Atmel ATmega328p - Short EEPROM Demo
2. // Zielpattform: Atmel ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, christian.jakob@h-da.de
4.
5. #define EEMEM __attribute__((section (".eeprom")))
6.
7. #include <avr/eeprom.h>
8. #include <avr/io.h>
9.
10. unsigned char test_byte EEMEM = 0xCD;
11.
12. int main(void) {
13.     unsigned char test;
14.     uart_init();
15.     while(1) {
16.         eeprom_write_byte((unsigned char*)0x02, 0xAB);
17.         test = eeprom_read_byte((unsigned char *)0x02);
18.         test = eeprom_read_byte(&test_byte);
19.     }
20.     return 0;
21. }

```

Quellcode 2-8 Atmel ATmega328p– EEPROM – Demo-Programm #2

- Kopieren Sie anschließend den zuvor aufgeführten Quellcode in die neu erzeugte C-Datei. Schalten Sie mögliche Optimierungen durch den Compiler ab. Aktivieren Sie zudem die Zeilennummerierung im Editor²³.
 - Da der EEPROM-Speicher vom Atmel Studio internen Simulator unterstützt wird, soll nachfolgend die Funktionalität des Programms mittels Simulation näher analysiert werden. Kompilieren Sie dazu das Programm mit der Taste **F7** und wechseln Sie anschließend mittels **Alt+F5** in den Debug-Modus.
 - Wählen Sie nun im Menü View die Ansichten Processor und Memory aus. Selektieren Sie Im Fenster Memory den EEPROM-Speicher des ATmega328pt.
 - Wenn ein C-Programm mittels des Atmel Studio Simulator getestet wird, kann das Watch Window zur Überwachung der Symbole verwendet werden. Nutzen Sie dieses Feature zur Überwachung der Variablenwerte im Programm.
 - Ordnen Sie die einzelnen Fenster gut sichtbar an und simulieren Sie anschließend das Programm Befehl für Befehl im Einzelschritt-Mode (**F11**). Beachten Sie, dass der nächste auszuführende Befehl im Programm immer gelb hinterlegt wird.
- 23.** Evaluieren Sie die in der AVR-LibC Bibliothek bereitgestellten Funktionen für den Zugriff auf den EEPROM-Speicher. Schalten Sie mögliche Compiler-Optimierungen (-O0) ab und aktivieren Sie zudem die Zeilennummerierung im Editor. Kompilieren Sie ihr Programm mit der Taste **F7** und wechseln Sie anschließend mittels **Alt+F5** in den Debug-Modus. Überprüfen Sie die korrekte Funktionalität ihres Programms mittels Simulation.
- 24.** Legen Sie ein Character-Array im EEPROM des ATmega328p an und initialisieren Sie es mit dem Satz „Hello World!“. Nach Systemstart soll auf diese Variable mittels der Funktion `eeeprom_read_block` in den RAM Bereich des Controllers übertragen. Verifizieren Sie ihre Lösung mittels Simulation.
- 25.** Bauen Sie die Schaltung gemäß Abbildung 2.2 auf und implementieren Sie das in Abschnitt 2.1 vorgestellte Konzept zur Registersicherung. Ermitteln Sie mit Hilfe des Datenblatts die benötigten Werte hinsichtlich des möglichen Spannungsabfalls sowie der ungefähren Stromaufnahme. Realisieren Sie einen Sicherungsmechanismus für einen Registerwert und prüfen Sie anschließend ihre Implementierung auf der Zielhardware.

²³ Atmel Studio 6 – Tools – Options – Text Editor – All languages – General: Display line numbers

3. Timer

Viele Prozesse, die durch einen Mikrocontroller gesteuert werden, basieren auf präzisen zeitlichen Rahmenbedingungen. So müssen z.B. Zündzeitpunkte bei Verbrennungsmotoren genau eingehalten oder Ventile für eine bestimmte Dauer geöffnet werden, um eine möglichst genaue Menge des Kraftstoff-Luftgemischs in den Zylinder des Motors einzuführen.

Für Aufgaben dieser Art verfügen moderne Mikrocontroller wie der Atmel ATmega328p über eine Vielzahl von sogenannten Timer-Einheiten, die bei gegebener Konfiguration und Programmierung das zyklische Ausführen von Programmteilen in zeitlich konstanten und periodischen Abständen erlauben. Eine solche Programmstruktur ermöglicht die Generierung einer exakten Abtastzeitbasis und wird somit vornehmlich zur Implementierung von digitalen Regel- und Signalverarbeitungsalgorithmen eingesetzt. Ein weiteres populäres Einsatzgebiet von Timern ist die Generierung von pulsweitenmodulierten Rechteckspannungen oder etwa das Zählen externer Ereignisse. Darüber hinaus eignet sich die Timer-Peripherie eines Mikrocontrollers zudem zur Messung der Zeitdifferenz zweier externer Impulse und somit zur Bestimmung der Frequenz oder dem Tastverhältnis eines extern anliegenden Signals.

Der Einsatz der ATmega328p internen Timer-Peripherie im Kontext dieser Anwendungen ist Thema des folgenden Kapitels. Nach einer allgemeinen thematischen Einführung werden im zweiten Abschnitt des Kapitels der Aufbau, die Konfiguration sowie die Programmierung der ATmega328p internen Timer-Peripherie ausführlich diskutiert. Die zuvor diskutierten Konzepte werden abschließend anhand konkreter Beispiele anschaulich aufgezeigt.

Nach dem Studium dieses Kapitels und dem Bearbeiten der Übungsaufgaben sollten Sie

- die grundlegende Funktionsweise sowie den Aufbau der ATmega328p internen Timer kennen und beschreiben können,
- in der Lage sein für eine gegebene Konfiguration die resultierenden Timer-Diagramme zu zeichnen ,
- in der Lage sein den ATmega328p internen Timer gemäß gegebener Vorgaben zu konfigurieren und in Betrieb zu nehmen,
- einfache Timer-Routinen, sowohl im Polling-Modus als auch im Interrupt-Mode im Rahmen verschiedener Anwendungsszenarien programmieren können,
- den Timer-Interrupt einsetzen können, um Anwendungsfunktionen periodisch auszuführen,
- die verschiedenen PWM Modes kennen und erklären können sowie pulsweitenmodulierte Signale mit Hilfe der ATmega328p internen Timer-Peripherie generieren können.

3.1 Einführung

Zu den wichtigsten Peripheriekomponenten des Atmel ATmega328p gehören die internen Timer/Counter-Stufen. Sie erlauben zum einen die Steuerung zeitlich komplexer Abläufe (*Timer, engl. Zeitgeber*). Zum anderen agieren Sie bei gegebener Konfiguration als externer Ereigniszähler (*Counter, engl. Zähler*). Neben diesen beiden Grundfunktionen können Sie zudem in vielfältigster Weise zur Generierung pulswidenmodulierter Signale eingesetzt werden.

Der Atmel ATmega328 verfügt die folgenden unabhängigen Timer-Einheiten:

Timer T/C0 8-bit (0-255)

- Binärer 8-bit Aufwärtzähler
- Interner sowie externer Taktbetrieb. Der interne Takt (Systemtakt) kann mittels Vorteiler weiter reduziert werden.
- Mögliche Interrupt-Quellen: Overflow sowie Compare Match.
- Durch die Nutzung des externen Takteingang kann der Timer als Ereigniszähler betrieben werden.
- Generierung pulswidenmodulierter Signale (fast PWM sowie phase correct PWM).
- Zwei unabhängige Compare Match Einheiten
- Clear Timer On Compare Match Funktionalität

Timer T/C1 16-bit (0-65535)

Zusätzlich zum zuvor beschriebenen Funktionsumfang verfügt die 16-bit Timer-Einheit T/C1 über:

- Eine Input-Capture-Einheit mit integrierter Rauschunterdrückung zur Frequenz- bzw. Pulsbreitenmessung externer Signale.

Timer T/C2 8-bit (0-255)

- Binärer 8-bit Aufwärtzähler
- Ausschließlich interner Taktbetrieb. Der Takt (Systemtakt) kann zudem mittels Vorteiler weiter reduziert werden. Die Anzahl an möglichen Vonteilern ist größer wie
- Mögliche Interrupt-Quellen: Overflow sowie Compare Match.
- Generierung pulswidenmodulierter Signale (fast PWM sowie phase correct PWM).
- Clear Timer On Compare Match Funktionalität

Im einfachsten Fall arbeitet einer der drei zuvor beschriebenen Timer-Einheiten im sogenannten Normal Mode. In dieser Betriebsart wird der Zähler bis zum Erreichen seines Maximalwerts (8-bit: 0xFF) inkrementiert und läuft mit dem folgenden Zählimpuls über (8-bit: 0x00). Mit dem Erreichen des Maximalwertes (0xFF) wird zudem das Overflow-Bit (*overflow: engl. für Überlauf*) im Statusregister des Timers gesetzt.

Erreicht der Zählerstand einen zuvor festgelegten Vergleichswert wird darüber hinaus das sogenannte Compare Match Bit gesetzt. Beide Ereignisse können bei gegebener Konfiguration einen Interrupt auslösen. In der nachfolgenden Abbildung ist beispielhaft der zeitliche Verlauf des 8-bit Timers T/C0 im Normal Mode aufgezeigt.

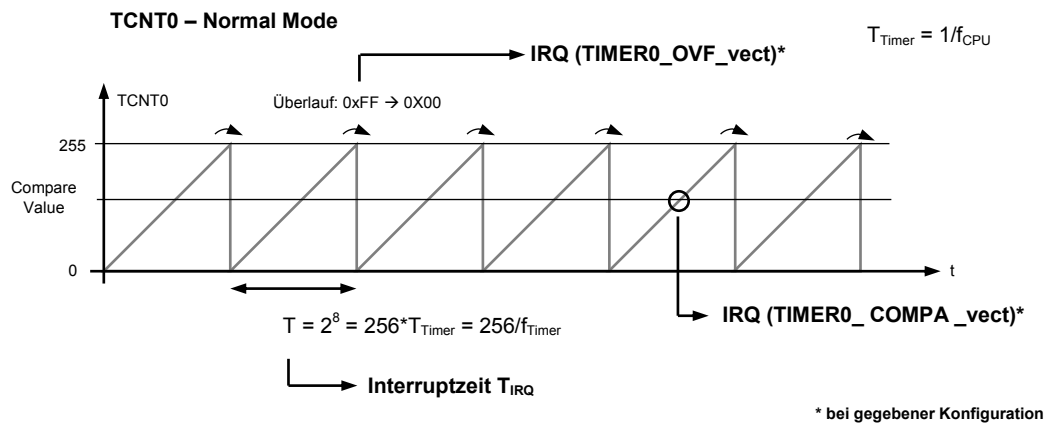


Abbildung 3-1 Zeitlicher Verlauf eines 8-bit Timer im Normal Mode (0xFF – 0x00)

Der zeitliche Abstand zwischen zwei Überläufen wird maßgeblich durch die Taktfrequenz sowie die Bitbreite des Timers bestimmt. Wird, wie in Abbildung 3.2 gezeigt, mit dem Erreichen des maximalen Zählwerts ein Overflow-Interrupt ausgelöst, so kann in der nachfolgend ausgeführten Interrupt-Routine der Timer mit einem Wert ungleich Null initialisiert werden.

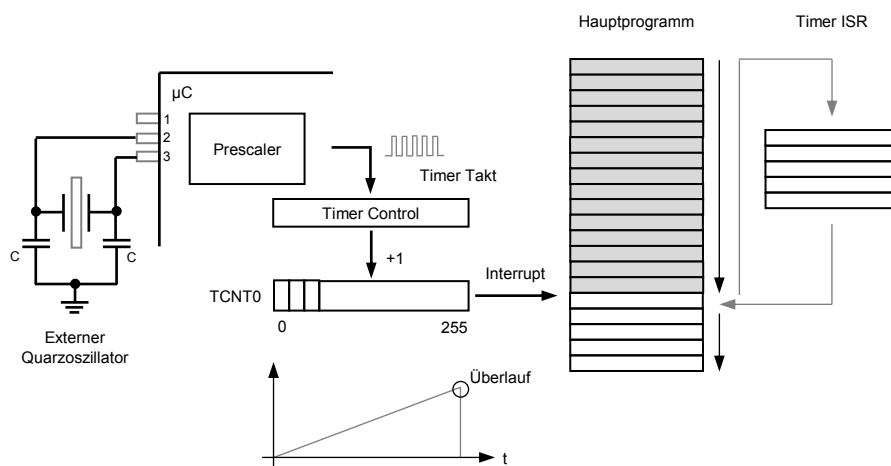
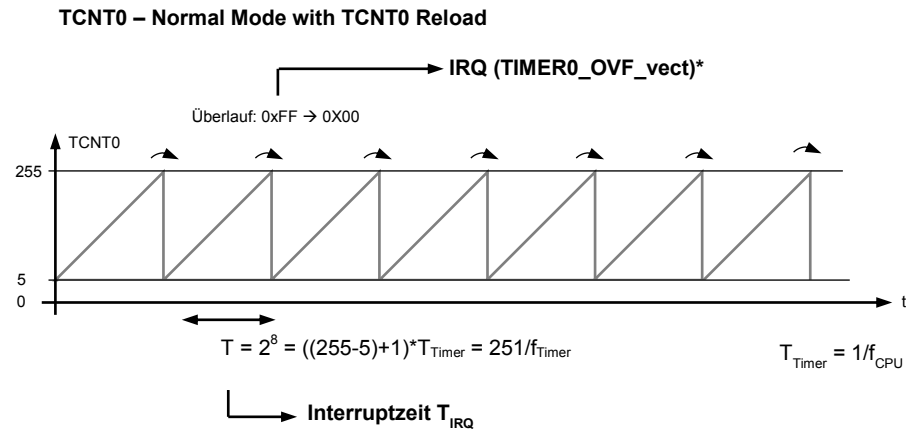


Abbildung 3-2 8-bit Timer konfiguriert im Normal-Mode (0xFF – 0x00) mit aktiviertem Overflow Interrupt

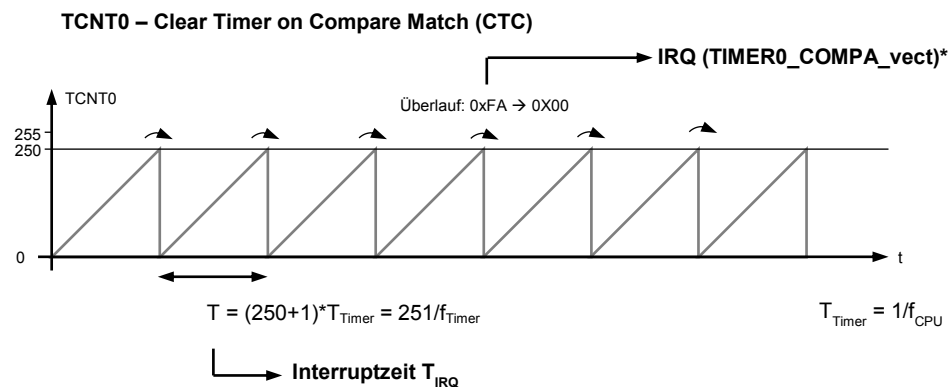
Dieses Szenario ist in Abbildung 3.3 dargestellt und bietet den Vorteil die Interrupt-Periode weiter zu reduzieren. Wie hier zu sehen beginnt der Zähler nach einem Überlauf nicht mit dem Wert Null, sondern mit dem in der Interrupt-Routine geladenen Wert (hier: 5).



* bei gegebener Konfiguration

Abbildung 3-3 Zeitlicher Verlauf eines 8-bit Timer im Normal Mode (0xFF – 0x00) mit Reload des initialen Zählerwerts

Eine weitere Möglichkeit die Interrupt-Frequenz eines Timers zu erhöhen bietet der sogenannte Clear Timer on Compare Match. In dieser Betriebsart wird der Zähler kontinuierlich bis zum Erreichen des festgelegten Vergleichswerts inkrementiert und läuft folglich mit dem nachfolgenden Zählimpuls über. Oder anders ausgedrückt: Der Timer wird mit dem Erreichen des neu definierten Maximalwerts auf Null zurückgesetzt. Mit dieser Funktionalität ist es folglich möglich, sehr präzise Interrupt-Perioden zu erzeugen, ohne dabei programmtechnisch eingreifen zu müssen (Timer-Initialisierung innerhalb der Overflow-ISR). Darüber hinaus kann, wie in Abbildung 3.4 illustriert, mit dem Erreichen des Vergleichswerts ein entsprechender Interrupt ausgelöst werden.



* bei gegebener Konfiguration

Abbildung 3-4 Zeitlicher Verlauf eines 8-bit Timer im 'Clear-Timer-On-Compare' Mode

Wenn der Wert im Vergleichsregister erreicht wird, wird zusätzlich zum möglichen Interrupt ein zur Compare-Unit gehörender Pin getoggelt. Dieses Szenario ist in Abbildung 3.5 gezeigt.

TCNT0 – Clear Timer on Compare Match (CTC)

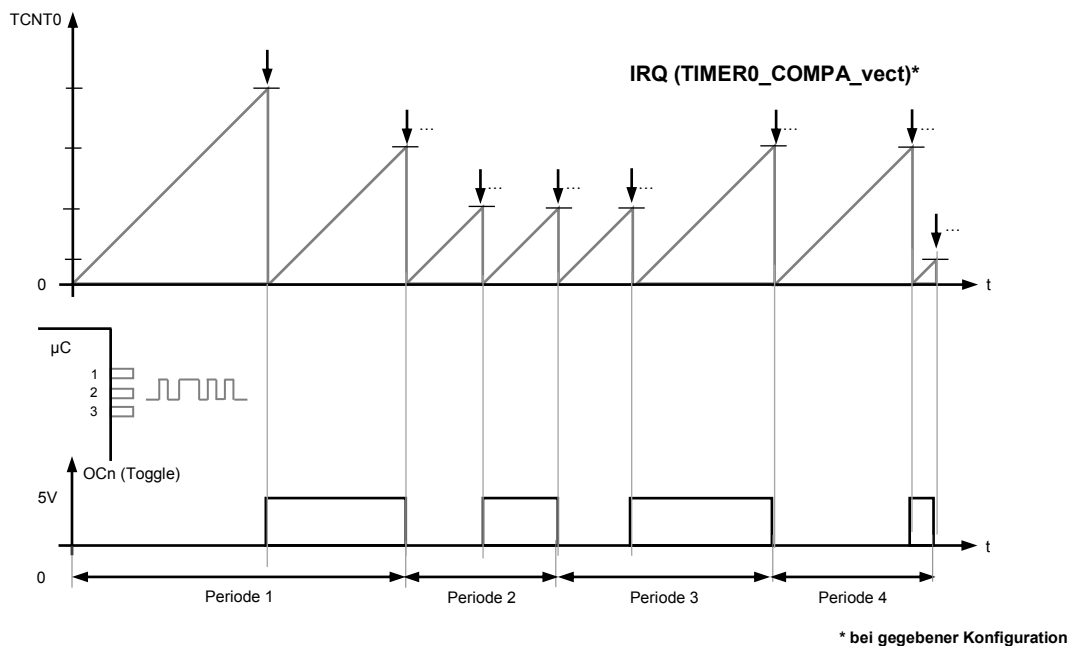
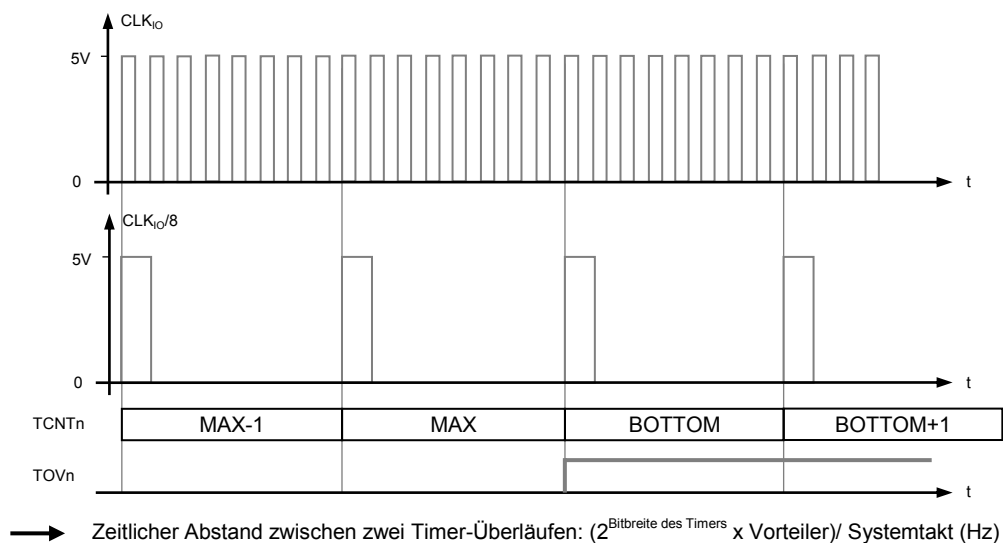


Abbildung 3-5 Zeitlicher Verlauf eines Timers im 'Clear-Timer-On-Compare' Mode mit zeitlich ändernden Compare-Werten unter Einbeziehung des OCn Pins

Gilt es die Interrupt-Frequenz des Timers zu reduzieren, so kann die Basisfrequenz des Controllers, wie in Abbildung 3.6 gezeigt, über das Setzen eines Prescaler-Wertes (*prescaler*, eng. für *Vorteiler*) heruntergeteilt werden.



Beispiel: Systemtakt 4MHz

Vorteile	Overflows/Sekunde	Zeit T _O zwischen zwei Überläufen [s]
1	15625	0.000064 = 64 µs
8	1953.125	0.000512 = 512 µs
64	244.1406	0.004096 = 4.1 ms
256	61.0351	0.016384 = 16.4 ms
1024	15.2587	0.065536 = 65,5 ms

Abbildung 3-6 Zeitlicher Verlauf eines Timers mit aktiviertem DIV-8 Prescaler

Als Eingangstakt für den Timer kann entweder der Systemtakt sowie ein an einen I/O-Pin angelegtes Signal verwendet werden. Wenn ein externes Signal verwendet wird, so darf dessen Frequenz nicht höher sein als die Hälfte des CPU-Taktes. In dieser Konfiguration agiert der Timer, wie nachfolgend illustriert als reiner Impuls-Zähler.

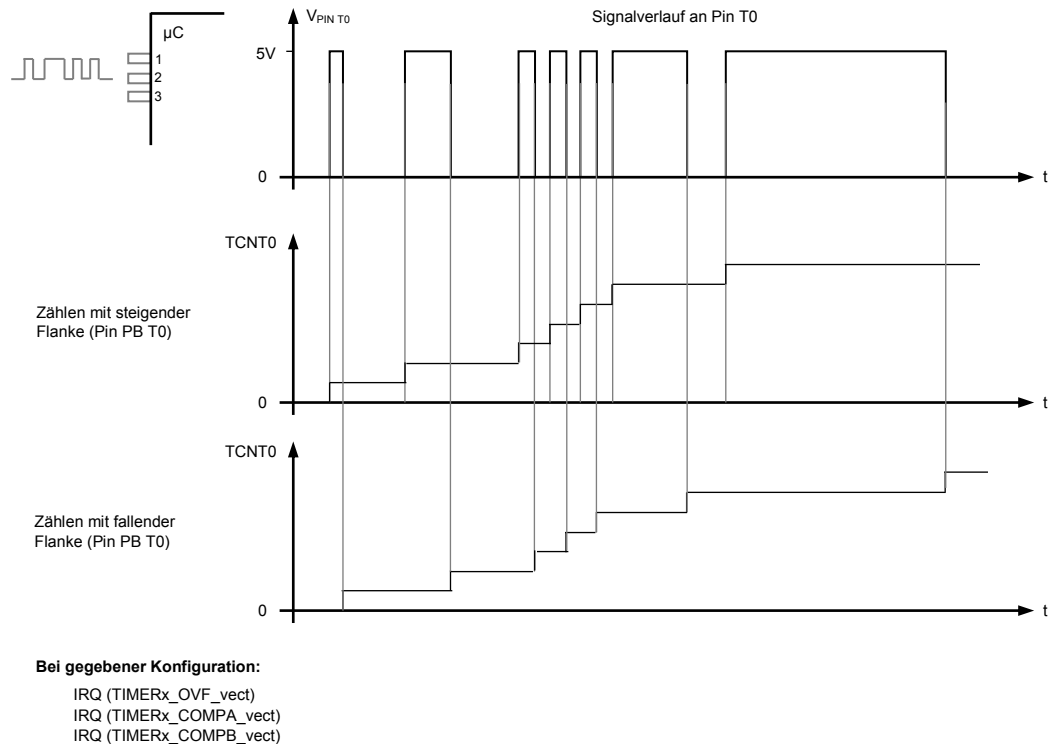


Abbildung 3-7 Timer als externer Ereigniszähler. Das Zählen erfolgt hier mit steigender bzw. fallender Flanke des externen Signals an Pin PB T0

Ein weiteres populäres Anwendungsfeld von Timer-Einheiten ist die in Abbildung 3.8 gezeigte Generierung von pulswidenmodulierten Signalen.

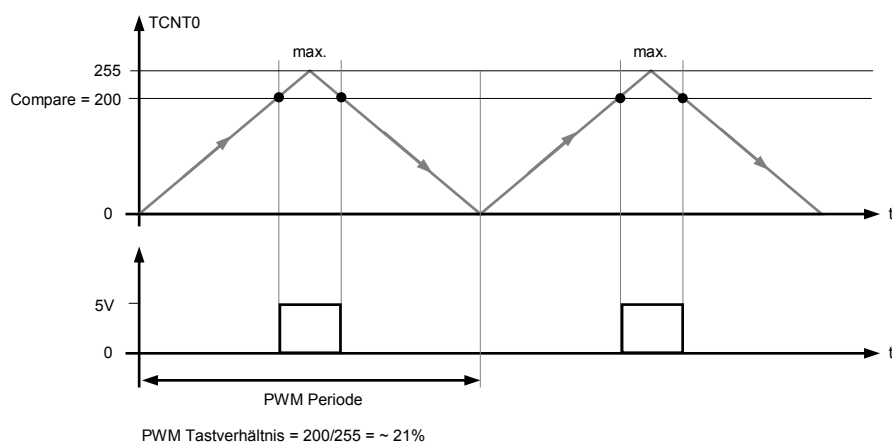


Abbildung 3-8 Generierung eines PWM-Signals

Nachfolgend ist der sogenannte 'Fast PWM Mode' dargestellt. In dieser Betriebsart wird der Zähler bis zum Erreichen seines Maximalwerts (8-bit: 0xFF) inkrementiert und läuft mit dem folgenden Zählimpuls über (8-bit: 0x00). Mit dem Überschreiten des Maximalwertes (0xFF – 0x00) wird zudem das Overflow-Bit (*overflow: engl. für Überlauf*) im Statusregister des Timers gesetzt, was bei gegebener Konfiguration zum Auslösen eines Interrupts führt.

Darüber hinaus wird mit dem Eintreten des Überlaufs der sogenannte Output Compare Pin der jeweiligen Timer-Einheit gesetzt. Dieser Pin wird beim nachfolgenden Hochzählen wieder zurückgesetzt, sobald der Zählerstand dem Inhalt des Output Compare Registers entspricht. Bei gegebener Konfiguration kann dieses Ereignis zum Auslösen eines Interrupts führen. Durch den zuvor beschriebenen Mechanismus wird, wie in Abbildung 3.9 gezeigt, ein Ausgangssignal mit konstanter Frequenz generiert, dessen Pulsweite über den Wert des Compare Registers festgelegt wird.

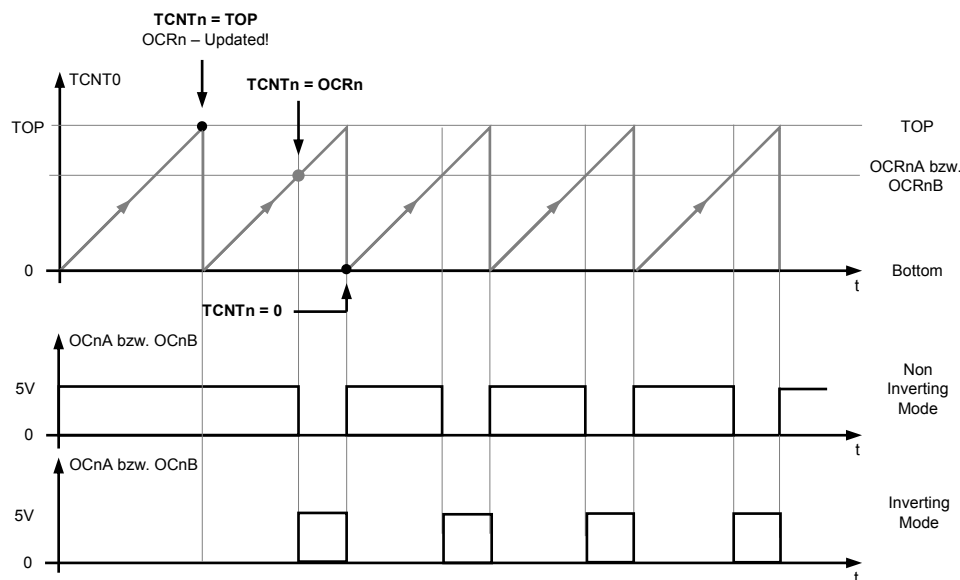


Abbildung 3-9 Atmel ATmega328p – Timer - Fast PWM Mode

Das 'Fast-PWM' Verfahren besitzt einen gravierenden Nachteil: Ändert man den Vergleichswert, dann bleibt eine Flanke stehen (die aus dem Overflow resultiert, wenn der Zähler wieder bei 0 beginnt), und die andere verschiebt sich (wegen des geänderten Vergleichswertes). Hieraus resultiert, wie nachfolgend gezeigt, eine Mittelpunktverschiebung zwischen zwei Flanken. Das PWM-Signal ist folglich nicht phasenrichtig.

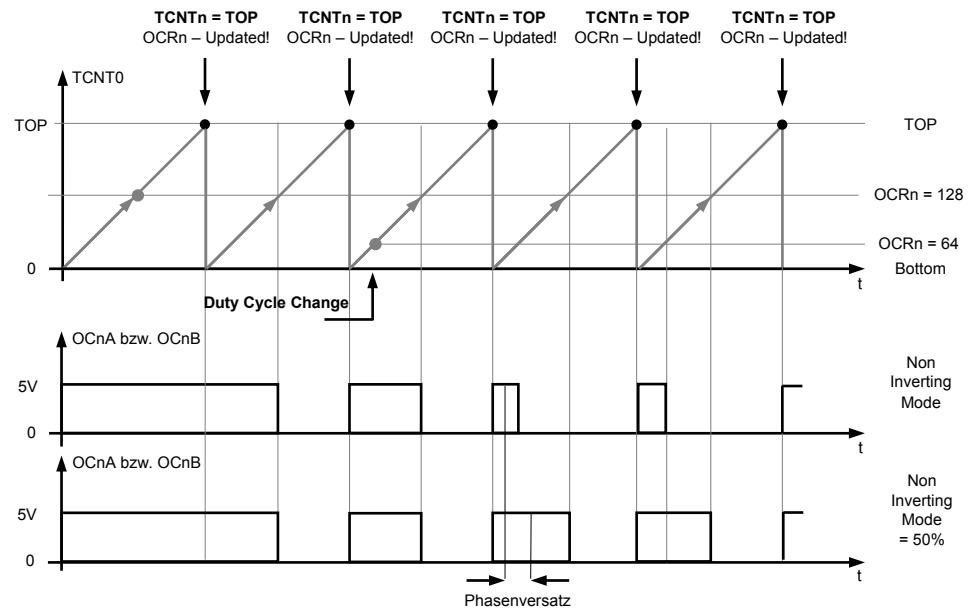


Abbildung 3-10 Atmel ATmega328p – Timer – Effekt des Phasenschiebens im Falle des Fast PWM Konfigurations-Mode

Phasenrichtige PWM-Signal-Erzeugung erlaubt der sogenannte Phase-Correct-PWM-Mode. In dieser Betriebsart arbeitet der Zähler im dual-slope Betrieb, d.h. der Zähler läuft erst rauf, und anschließend wieder runter. Jeweils beim Erreichen des Vergleichswertes wird das PWM-Signalausgang umgeschaltet. Ändert man den Vergleichswert, geht die steigende Flanke nach vorn und die fallende nach hinten, bzw. umgekehrt. Die Mitte des PWM-Signals bleibt, wie nachfolgend gezeigt, damit gleich, d.h. phasenkorrekt.

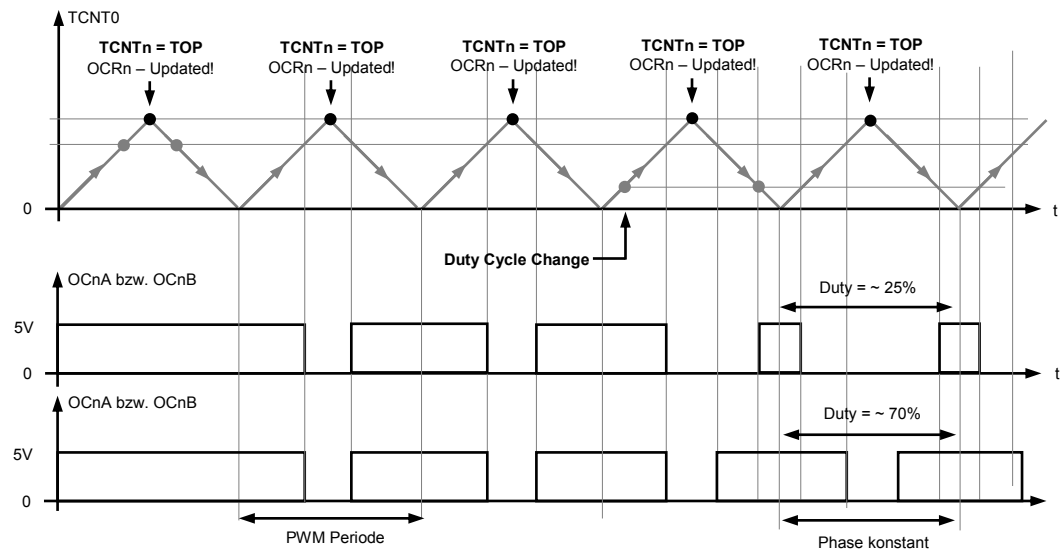


Abbildung 3-11 Atmel ATmega328p – Timer – Phase Correct PWM Mode

3.2 ATmega328p - Timer/Counter 0 - Steuerung und Konfiguration

Die nachfolgende Abbildung zeigt eine vereinfachte Darstellung der 8-bit Timer-Einheit T/C0 im Atmel ATmega328p.

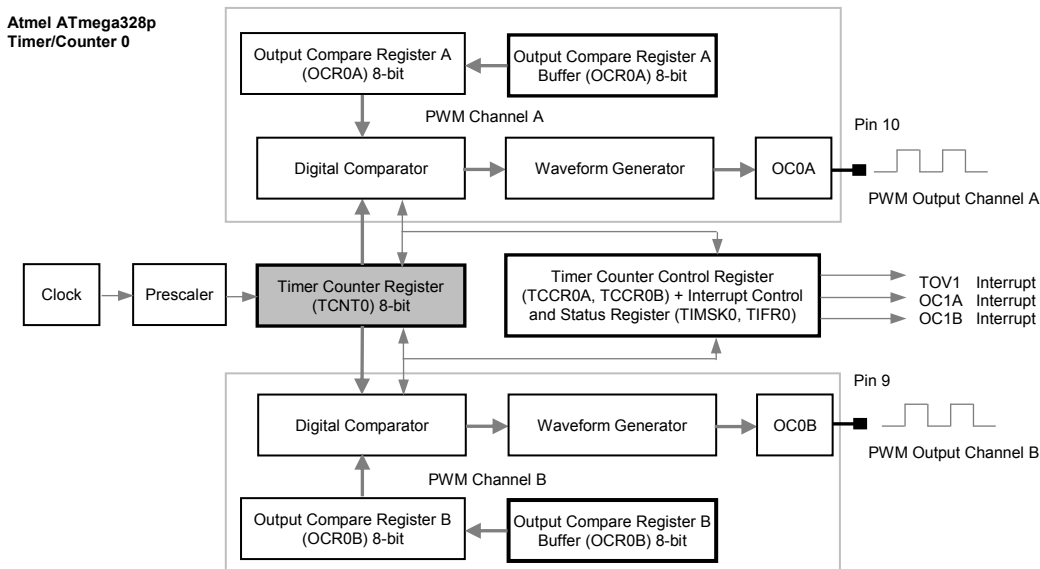


Abbildung 3-12 Vereinfachtes Blockschaltbild des Atmel ATmega328p Timer/Counter 0 Peripheriemoduls. Alle dick umrandeten Register sind über den internen Prozessorbus zugänglich

Die Programmierung dieses Peripheriemoduls erfolgt im Wesentlichen über die Register:

- Timer/Counter 0 Control Register TCCR0A
- Timer/Counter 0 Control Register TCCR0B
- Timer/Counter 0 Interrupt Mask Register TIMSK0
- Timer/Counter 0 Interrupt Flag Register TIFR0
- Timer/Counter 0 Register TCNT0
- Timer/Counter 0 Output Compare Register OCR0A

Nachfolgend wird nun die Funktion dieser Register vorgestellt und diskutiert. Für eine ausführliche Beschreibung der einzelnen Registerfunktionen wird an dieser Stelle auf die Dokumentation des Herstellers²⁴ verwiesen.

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-13 Atmel ATmega328p – Timer/Counter 0 Control Register A

Abbildung 3.13 zeigt das Steuerregister TCCR0A des Timer-Moduls T/C0.

²⁴ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.93 ff.

Über diese Registereinheit wird die Reaktion der Ausgangspins OC0A und OC0B im Falle einer Übereinstimmung des Zählerstands von Register TCNT0 mit den beiden Vergleichsregistern OCR0A und OCR0B festgelegt. Darüber hinaus erfolgt über das Register TCCR0A die Konfiguration der Betriebsart der Timer-Einheit. Die einzelnen Bits dieses Registers besitzen hier folgende Funktion:

Bit 7:6 – COM0A1:0: Compare Match Output A Mode

Über das Setzen der Bits COM0A1 und COM0A0 erfolgt die Konfiguration der Signaländerung an Pin OC0A für den Fall einer Übereinstimmung von Vergleichswert (OCR0A) und Zählerstand (TCNT0). Den Pin OC0A gilt es folglich über das zugeordnete Datenrichtungsregister als Ausgang zu konfigurieren. Zudem sei darauf hingewiesen, dass die Konfiguration des Compare Match Output Modes durch den zugrundeliegenden Betriebsmodus des Timer-Einheit T/C0 bestimmt wird.

COM0A1	COM0A0	Compare Output Mode A, non-PWM Mode
0	0	Normal Operation, OC0A disconnected
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

COM0A1	COM0A0	Compare Output Mode A, Fast-PWM Mode
0	0	Normal Operation, OC0A disconnected
0	1	WGM02 = 0 Normal Port Operation, OC0A disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode)
1	1	Set OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode)

COM0A1	COM0A0	Compare Output Mode A, Phase Correct-PWM Mode
0	0	Normal Operation, OC0A disconnected
0	1	WGM02 = 0 Normal Port Operation, OC0A disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match when up-counting Set OC0A on Compare Match when down counting
1	1	Set OC0A on Compare Match when up-counting Clear OC0A on Compare Match when down counting

Tabelle 3-1 Atmel ATmega328p – Timer/Counter 0 Compare Mode A

▪ Bit 5:4 – COM0B1:0: Compare Match Output B Mode

Über das Setzen der Bits COM0B1 und COM0B0 erfolgt die Konfiguration der Signaländerung an Pin OC0B für den Fall einer Übereinstimmung von Vergleichswert (OCR0B) und Zählerstand (TCNT0). Den Pin OC0B gilt es folglich über das zugeordnete Datenrichtungsregister als Ausgang zu konfigurieren. Zudem sei darauf hingewiesen, dass die Konfiguration des Compare Match Output Modes durch den zugrundeliegenden Betriebsmodus des Timer-Einheit T/C0 bestimmt wird.

COM0B1	COM0B0	Compare Output Mode B, non-PWM Mode
0	0	Normal Operation, OC0B disconnected
0	1	Toggle OC0B on Compare Match
1	0	Clear OC0B on Compare Match

1	1	Set OC0B on Compare Match
---	---	---------------------------

COM0B1	COM0B0	Compare Output Mode B, Fast-PWM Mode
0	0	Normal Operation, OC0B disconnected
0	1	ohne Funktion
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)

COM0B1	COM0B0	Compare Output Mode B, Phase Correct-PWM Mode
0	0	Normal Operation, OC0B disconnected
0	1	ohne Funktion
1	0	Clear OC0B on Compare Match when up-counting Set OC0B on Compare Match when down counting
1	1	Set OC0B on Compare Match when up-counting Clear OC0B on Compare Match when down counting

Tabelle 3-2 Atmel ATmega328p – Timer/Counter 0 Compare Mode B Betrieb

▪ Bit 1:0 – WGM01:0: Waveform Generation Mode

Das Setzen der Waveform Generation Mode Bits WGM02²⁵, WGM01 und WGM00 bestimmt den Betriebsmodus in dem der Timer T/C0 operiert. Die einzelnen Betriebsmodi sowie die hierfür notwendigen Einstellungen sind in der folgenden Tabelle aufgeführt.

WGM02	WGM01	WGM00	Beschreibung	TOP
0	0	0	Normal Mode	0xFF
0	0	1	PWM, Phase Correct	0xFF
0	1	0	CTC	OCR0A
0	1	1	Fast PWM	0xFF
1	0	0	Ohne Funktion	-
1	0	1	Fast PWM, Phase Correct	OCR0A
1	1	0	Ohne Funktion	-
1	1	1	Fast PWM	OCR0A

Tabelle 3-3 Atmel ATmega328p – Timer/Counter 0- Waveform Generation Modes

Abbildung 3.14 zeigt das zweite Steuerregister des Peripheriemoduls T/C0.

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-14 Atmel ATmega328p – Timer/Counter 0 Control Register B

²⁵ Siehe Steuerregister TCCR0B

Die einzelnen Bits dieses Registers besitzen folgende Funktion:

▪ **Bit 7 – FOC0A: Force Output Compare A**

Über das Setzen des Force Output Compare A Bits wird ein Compare Match Event erzwungen, obwohl das Zählerregister TCNT0 sowie das Vergleichsregister OCR0A unterschiedliche Werte aufweisen. Jedoch kommt es durch das Setzen dieses Bits weder zum Auslösen eines Interrupts, noch wird der Wert des TCNT0 Registers in irgendeiner Weise beeinflusst. Diese Funktionalität steht im PWM-Betrieb nicht zur Verfügung.

▪ **Bit 6 – FOC0B: Force Output Compare B**

Über das Setzen des Force Output Compare B Bits wird ein Compare Match Event erzwungen, obwohl das Zählerregister TCNT0 sowie das Vergleichsregister OCR0B unterschiedliche Werte aufweisen. Jedoch kommt es durch das Setzen dieses Bits weder zum Auslösen eines Interrupts, noch wird der Wert des TCNT0 Registers in irgendeiner Weise beeinflusst. Diese Funktionalität steht im PWM-Betrieb nicht zur Verfügung.

▪ **Bit 3 – WGM02: Waveform Generation Mode**

Siehe Beschreibung zu Steuerregister TCCR0A.

▪ **Bit 2:0 – CS02:0: Clock Select**

Über das Setzen der Clock Select Bits CS02, CS01 und CS00 erfolgt die Einstellung des Timer-Takts. Der Timer T/C0 kann zum einen über den internen Systemtakt des Controllers betrieben werden. Dieser kann zudem über das Setzen eines Vorteilers (*engl. Prescaler*) weiter reduziert werden. Zum anderen besteht die Möglichkeit den Timer mit einem an Pin T0 anliegenden externen Takt zu betreiben. In diesem Fall operiert der Timer als Zähler (*engl. Counter*). Die möglichen Konfigurationen sowie die hierfür notwendigen Einstellungen sind in der folgenden Tabelle aufgeführt.

CS02	CS01	CS00	Beschreibung
0	0	0	Timer/Counter 0 abgeschaltet
0	0	1	Timer mit Systemtakt betreiben
0	1	0	Systemtakt/8
0	1	1	Systemtakt/64
1	0	0	Systemtakt/256
1	0	1	Systemtakt/1024
1	1	0	Betriebsart - Zähler Ext. Clock Source (T0 Pin). Fallende Flanke
1	1	1	Betriebsart - Zähler Ext. Clock Source (T0 Pin). Steigende Flanke

Tabelle 3-4 Atmel ATmega328p – Timer/Counter 0- Clock Select Modes

Werden die Bits CS02, CS01 und CS00 mit einem Wert ungleich 0b000 überschrieben, beginnt der Timer T/C0 mit jedem an seinem Takteingang eintreffenden Taktimpuls seinen Zählerstand (Register TCNT0) um eins zu inkrementieren.

Wird die Timer-Einheit T/C0 als Zähler betrieben, so erfolgt intern eine Synchronisierung des externen Signals mit dem Systemtakt des Controllers. Hierzu wird das an Pin T0 anliegende Signal kontinuierlich mit jeder steigenden Flanke des Systemtaktes abgetastet. Um eine sichere Signalerkennung zu gewährleisten muss sichergestellt sein, dass die Periodendauer des externen Signals an Pin T0 mindestens der doppelten Periodendauer des Systemtaktes entspricht.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-15 Atmel ATmega328p – Timer/Counter 0 Interrupt Mask Register

Die Freigabe der einzelnen Interrupts der Timer-Einheit T/C0 erfolgt über das in Abbildung 3.15 dargestellte Timer/Counter0 Interrupt Mask Register TIMSK0.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

Durch das Setzen dieses Bits erfolgt die Aktivierung des Compare Match B Interrupts der Timer-Einheit T/C0 (vorausgesetzt Interrupts sind global freigeschaltet).

- **Bit 1 – OCIE0A: Timer/Counter Output Compare Match A Interrupt Enable**

Durch das Setzen dieses Bits erfolgt die Aktivierung des 'Compare Match A' Interrupts der Timer-Einheit (vorausgesetzt Interrupts sind global freigeschaltet).

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

Durch das Setzen dieses Bits erfolgt die Aktivierung des Overflow Interrupts der Timer-Einheit (vorausgesetzt Interrupts sind global freigeschaltet).

Der Status der einzelnen Interrupts der Timer-Peripherie T/C0 wird in Register TIFR0 vermerkt.

Bit	7	6	5	4	3	2	1	0	
0x15(0x35)	-	-	-	-	-	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-16 Atmel ATmega328p – Timer/Counter 0 Interrupt Flag Register

- **Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag**

Das OCF0B Bit wird gesetzt, wenn der aktuelle Zählerstand des Registers TCNT0 dem Wert im Register OCR0B entspricht. Ist zudem das OCIE0B Bit gesetzt und sind globale Interrupts im zentralen Statusregister SREG der

AVR CPU freigeschaltet, so erfolgt mit dem Setzen des OCF0B Bits das Auslösen des zugehörigen Interrupts²⁶. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das OCF0B Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

▪ **Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag**

Das OCF0A Bit wird gesetzt, wenn der aktuelle Zählerstand des Registers TCNT0 dem Wert im Register OCR0A entspricht. Ist zudem das OCIE0A Bit gesetzt und sind globale Interrupts im zentralen Statusregister SREG der AVR CPU freigeschaltet, so erfolgt mit dem Setzen des OCF0A Bits das Auslösen des zugehörigen Interrupts²⁷. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das OCF0A Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

• **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

Das Timer/Counter0 Overflow Bit TOV0 wird gesetzt, wenn ein Überlauf des Registers TCNT0 stattfindet. Ist zudem das TOIE0 Bit gesetzt und sind globale Interrupts im zentralen Statusregister SREG der AVR CPU freigeschaltet, so erfolgt mit dem Setzen des TOV0 Bits das Auslösen des zugehörigen Interrupts²⁸. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das TOV0 Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

Im Zentrum des aktuellen Peripheriemoduls steht das in Abbildung 3.17 dargestellte Zählregister TCNT0. Hierbei handelt es sich um ein 8-bit breites Register, dessen Wert in Abhängigkeit der gewählten Konfiguration (Taktquelle, Prescaler) kontinuierlich um eins inkrementiert wird.

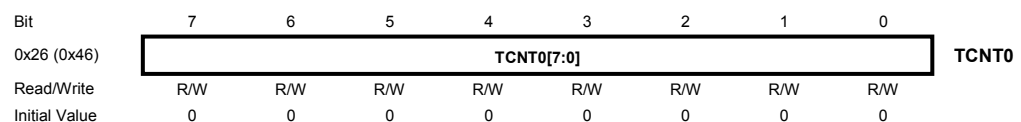


Abbildung 3-17 Atmel ATmega328p – Timer/Counter 0 Register

Abbildung 3.18 zeigt nachfolgend die Struktur des Output Compare Register OCR0A des Timers T/C0. Der hier gespeicherte 8-bit Wert wird intern fortlaufend mit dem aktuellen Zählerstand des Registers TCNT0 verglichen.

²⁶ TIMER0_COMPB_vect

²⁷ TIMER0_COMPA_vect

²⁸ TIMER0_COMPA_vect

Besitzen beide Register identischen Inhalt, so kann dieses Ereignis zum Auslösen des Compare Match Interrupts `TIMER0_COMPA_vect` sowie zum Triggern einer Signaländerung an Pin `OC0A` genutzt werden.

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-18 Atmel ATmega328p – Timer/Counter 0 Output Compare Register

Identische Funktionalität bietet das in Abbildung 3.19 dargestellte Output Compare Register `OCR0B`. Der Inhalt dieses Registers wird durch dedizierte Hardware in jedem Taktzyklus mit dem Inhalt des Registers `TCNT0` verglichen. Besitzen beide Register identischen Inhalt, so kann dieses Ereignis zum Auslösen eines Compare Match Interrupts `TIMER0_COMPB_vect` sowie zum Triggern einer Signaländerung an Pin `OC0B` genutzt werden.

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-19 Atmel ATmega328p – Timer/Counter 0 Output Compare Register

3.3 ATmega328p - Timer/Counter 1 - Steuerung und Konfiguration

Für die Programmierung des Timers T/C1 des Atmel ATmega328p sind im Wesentlichen die folgenden sieben Register relevant:

- Timer/Counter 1 Control Register TCCR1A
- Timer/Counter 1 Control Register TCCR1B
- Timer/Counter 1 Control Register TCCR1C
- Timer/Counter 1 Interrupt Mask Register TIMSK1
- Timer/Counter 1 Interrupt Flag Register TIFR1
- Timer/Counter 1 Register TCNT1H/L (16-bit)
- Timer/Counter 1 Output Compare Register OCR1AH/L (16-bit)
- Timer/Counter 1 Output Compare Register OCR1BH/L (16-bit)
- Timer/Counter 1 Input Capture Register ICR1H/L (16-bit)

Ein Blockschaltbild dieser Peripherie-Einheit ist in Abbildung 3.20 dargestellt.

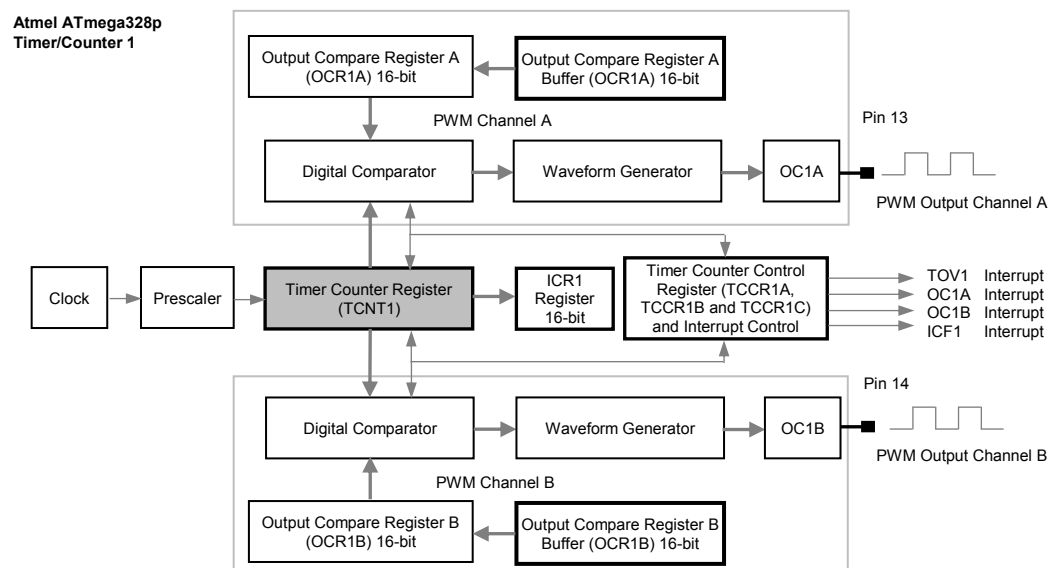


Abbildung 3-20 Vereinfachtes Blockschaltbild des Atmel ATmega328p Timer/Counter 1 Peripheriemoduls. Alle dick umrandeten Register sind über den internen Prozessorbuss zugänglich

Nachfolgend wird nun die Funktion dieser Register vorgestellt und diskutiert. Für eine ausführliche Beschreibung der einzelnen Registerfunktionen wird an dieser Stelle auf die Dokumentation des Herstellers²⁹ verwiesen.

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0			WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-21 Atmel ATmega328p – Timer/Counter 1 Control Register A

²⁹ Datenblatt Atmel ATmega328p: ATmega48PA/88PA/168PA/328P - doc8161, S.93 ff.

Die Steuerung der 16-bit Timer-Einheit T/C1 des ATmega328p erfolgt wie zuvor gezeigt über die Steuerregister TCCR1A, TCCR1B und TCCR1C. Ähnlich wie im Falle des 8-bit Timer T/C0 wird über die Registereinheit TCCR1A die Reaktion der Ausgangspins OC1A und OC1B im Falle einer Übereinstimmung des Zählerstands von Register TCNT1 mit den beiden Vergleichsregistern OCR1A und OCR1B festgelegt. Darüber hinaus erfolgt über das Register TCCR1A die Konfiguration der Betriebsart der Timer-Einheit. Die einzelnen Bits dieses Registers besitzen folgende Funktion:

▪ **Bit 7:4 – COM1X1:0: Compare Match Output Mode**

Über das Setzen der Bits COM1B1 und COM1B0 bzw. COM1A1 und COM1A0 erfolgt die Konfiguration der Signaländerung an den Pins OC1B und OC1A für den Fall einer Übereinstimmung von Vergleichswert (OCR1B/OCR1A) und Zählerstand (TCNT1). Die Pins OC1B und OC1A gilt es folglich über das zugeordnete Datenrichtungsregister als Ausgänge zu konfigurieren. Zudem sei darauf hingewiesen, dass die Konfiguration des Compare Match Output Modes durch den Betriebsmodus der Timer-Einheit T/C1 bestimmt wird.

COM1A1/ COM1B1	COM1A0/ COM1B0	Compare Output Mode A/B, non-PWM Mode
0	0	Normal Operation, OC1A/OC1B disconnected
0	1	Toggle OC1A/OC1B on Compare Match
1	0	Clear OC1A/OC1B on Compare Match
1	1	Set OC1A/OC1B on Compare Match
COM1A1/ COM1B1	COM1A0/ COM1B0	Compare Output Mode A/B, Fast-PWM Mode
0	0	Normal Operation, OC1A/OC1B disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
COM0A1	COM0A0	Compare Output Mode A/B, Phase Correct-PWM Mode
0	0	Normal Operation, OC1A/OC1B disconnected
0	1	WGM13:0 = 9 or 11: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when down counting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when down counting.

Tabelle 3-5 Atmel ATmega328p – Timer/Counter 1 Compare Mode A/B

▪ Bit 1:0 – WGM01:0: Waveform Generation Mode

Das Setzen der Waveform Generation Mode Bits WGM13, WGM12³⁰, WGM11 und WGM00 bestimmt den Betriebsmodus, in dem der 16-bit Timer T/C1 operiert. Die einzelnen Betriebsmodi sowie die hierfür notwendigen Einstellungen sind in der folgenden Tabelle aufgeführt.

WGM 13	WGM 12	WGM 11	WGM 10	Beschreibung	TOP
0	0	0	0	Normal Mode	0xFFFF
0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF
0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF
0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF
0	1	0	0	CTC	OCR1A
0	1	0	1	Fast PWM, 8-bit	0x00FF
0	1	1	0	Fast PWM, 9-bit	0x01FF
0	1	1	1	Fast PWM, 10-bit	0x03FF
1	0	0	0	PWM, Phase and Freq. Correct	ICR1
1	0	0	1	PWM, Phase and Freq. Correct	OCR1A
1	0	1	0	PWM, Phase Correct	ICR1
1	0	1	1	PWM, Phase Correct	OCR1A
1	1	0	0	CTC	ICR1
1	1	0	1	ohne Funktion	-
1	1	1	0	Fast PWM	ICR1
1	1	1	1	Fast PWM	OCR1A

Tabelle 3-6 Atmel ATmega328p – Timer/Counter 1 - Waveform Generation Modes

Das zweite Steuerregister der Timer-Peripherie T/C1 ist in Abbildung 3.22 dargestellt.

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-22 Atmel ATmega328p – Timer/Counter 1 Control Register B

▪ Bit 7 – ICNC1: Input Capture Noise Canceler

Ist das Signal zur Triggerung des Input Capture Einheit stark verrauscht, kann die Rauschunterdrückung über das Input Capture Noise Canceller Bit ICNC1 eingeschaltet werden. In diesem Fall wird bei der Detektion der konfigurierten Flanke über 4 Taktzyklen das Signal überwacht und nur dann, wenn alle 4 Messungen identisch sind, wird die entsprechende Aktion ausgelöst. Dementsprechend erfolgt die Übernahme des aktuellen Zählerstands von Timer T/C1 in das ICR1 Register um vier Taktzyklen verzögert.

³⁰ Das Setzen der Bits WGM13 und WGM12 erfolgt über das Steuerregister TCCR1B

▪ Bit 6 – ICES1: Input Capture Edge Select

Über das Bit ICES1 im zuvor aufgeführten TCCR1B Register wird bestimmt, ob die steigende (ICES1 gesetzt) oder fallende (ICES1 nicht gesetzt) Flanke zur Auswertung des Input Capture Signals heran gezogen wird.

▪ Bit 4 – WGM13: Waveform Generation Mode

Siehe Beschreibung zu Steuerregister TCCR1A

▪ Bit 4 – WGM12: Waveform Generation Mode

Siehe Beschreibung zu Steuerregister TCCR1A

▪ Bit 2:0 – CS02:0: Clock Select

Über das Setzen der Clock Select Bits CS12, CS11 und CS10 erfolgt die Einstellung des Timer-Takts. Der Timer T/C1 kann zum einen über den internen Systemtakt des Controllers betrieben werden. Dieser kann zudem über das Setzen eines Vorteilers (*engl. Prescaler*) weiter reduziert werden. Zum anderen besteht die Möglichkeit den Timer mit einem an Pin T1 anliegenden externen Takt zu betreiben. In diesem Fall operiert der Timer als Zähler (*engl. Counter*). Die möglichen Konfigurationen sowie die hierfür notwendigen Einstellungen sind in der folgenden Tabelle aufgeführt.

CS12	CS11	CS10	Beschreibung
0	0	0	Timer/Counter 1 anhalten
0	0	1	Timer 1 mit Systemtakt betreiben
0	1	0	Clock/8
0	1	1	Clock/ 32
1	0	0	Clock/ 256
1	0	1	Clock/ 1024
1	1	0	Ext. Clock Source (T1 Pin). Fallende Flanke
1	1	1	Ext. Clock Source (T1 Pin) Steigende Flanke

Tabelle 3-7 Atmel ATmega328p – Timer/Counter 1 - Clock Select Modes

Über das dritte Steuerregister TCCR1C wird ausschließlich die Force Output Compare Funktionalität der Timer-Einheit T/C1 konfiguriert.

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	-	-	-	-	-	-	TCCR1C
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-23 Atmel ATmega328p – Timer/Counter 1 Control Register C

▪ Bit 7 – FOC1A: Force Output Compare A

Über das Setzen des FOC1A Bits wird ein Compare Match Event erzwungen, obwohl das Zählerregister TCNT1 sowie das Vergleichsregister OCR1A unterschiedliche Werte aufweisen. Jedoch kommt es durch das Setzen dieses Bits weder zum Auslösen eines Interrupts, noch wird der Wert des TCNT1 Registers in irgendeiner Weise beeinflusst. Diese Funktionalität steht im PWM-Betrieb nicht zur Verfügung.

▪ Bit 6 – FOC1B: Force Output Compare B

Über das Setzen des FOC1B Bits wird ein Compare Match Event erzwungen, obwohl das Zählerregister TCNT1 sowie das Vergleichsregister OCR1B unterschiedliche Werte aufweisen. Jedoch kommt es durch das Setzen dieses Bits weder zum Auslösen eines Interrupts, noch wird der Wert des TCNT0 Registers in irgendeiner Weise beeinflusst. Diese Funktionalität steht im PWM-Betrieb nicht zur Verfügung.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-24 Atmel ATmega328p – Timer/Counter 1 Interrupt Mask Register

Die Aktivierung der für die Timer-Einheit T/C1 relevanten Interrupts erfolgt über das in Abbildung 3.25 dargestellte TIMSK1 Register.

▪ Bit 5 – ICIE1: Timer/Counter 1 Input Capture Interrupt Enable

Durch das Setzen dieses Bits erfolgt die Aktivierung des Input Capture Interrupts der Timer-Einheit T/C1 (vorausgesetzt Interrupts sind global freigeschaltet).

▪ Bit 2 – OCIE1B: Timer/Counter 1 Output Compare Match B Interrupt Enable

Durch das Setzen dieses Bits erfolgt die Aktivierung des Output Compare B Interrupts der Timer-Einheit T/C1 (vorausgesetzt Interrupts sind global freigeschaltet).

▪ Bit 1 – OCIE1A: Timer/Counter 1 Output Compare Match A Interrupt Enable

Durch das Setzen dieses Bits erfolgt die Aktivierung des Output Compare A Interrupts der Timer-Einheit T/C1 (vorausgesetzt Interrupts sind global freigeschaltet).

▪ Bit 0 – TOIE1: Timer/Counter 1 Overflow Interrupt Enable

Durch das Setzen dieses Bits erfolgt die Aktivierung des Overflows Interrupts der Timer-Einheit T/C1 (vorausgesetzt Interrupts sind global freigeschaltet).

Bit	7	6	5	4	3	2	1	0	
0x16(0x36)	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-25 Atmel ATmega328p – Timer/Counter 1 Interrupt Flag Register

Der Status der einzelnen Interrupts wird im Register TIFR1 vermerkt.

▪ Bit 5 – ICF1: Timer/Counter 1 Input Capture Flag

Wird am Input Capture Pin ICP die im Register TCCR1B definierte Flanke erkannt wird, so wird der aktuelle Inhalt des 16-bit Registers TCNT1 in das Register ICR1 kopiert und das Input Capture Flag ICF1 gesetzt. Ist zudem das ICIE1 Bit gesetzt und sind globale Interrupts freigeschaltet, so erfolgt mit dem Setzen des ICF1 Bits das Auslösen des zugehörigen Interrupts `TIMER1_CAPT_vect`. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das ICF1 Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

▪ Bit 2 – OCF1B: Timer/Counter 1 Output Compare B Match Flag

Das OCF1B Bit wird gesetzt, wenn der aktuelle Zählerstand des Registers TCNT1 dem Wert im Register OCR1B entspricht. Ist zudem das OCIE1B Bit gesetzt und sind globale Interrupts freigeschaltet, so erfolgt mit dem Setzen des OCF1B Bits das Auslösen des zugehörigen Interrupts `TIMER1_COMPB_vect`. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das OCF1B Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

▪ Bit 1 – OCF1A: Timer/Counter 1 Output Compare A Match Flag

Das OCF1A Bit wird gesetzt, wenn der aktuelle Zählerstand des Registers TCNT1 dem Wert im Register OCR1A entspricht. Ist zudem das OCIE1A Bit gesetzt und sind globale Interrupts im zentralen Statusregister SREG der AVR CPU freigeschaltet, so erfolgt mit dem Setzen des OCF1A Bits das Auslösen des zugehörigen Interrupts `TIMER1_COMPA_vect`. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das OCF1A Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

▪ Bit 0 – TOV1: Timer/Counter 1 Overflow Flag

Das Timer/Counter1 Overflow Bit TOV1 wird gesetzt, wenn ein Überlauf des Registers TCNT1 stattfindet. Ist zudem das TOIE1 Bit gesetzt und sind globale Interrupts im zentralen Statusregister SREG der AVR CPU freigeschaltet, so erfolgt mit dem Setzen des TOV1 Bits das Auslösen des zugehörigen Interrupts `TIMER1_OVF_vect`. Das Bit wird automatisch gelöscht, wenn die dazugehörige Interrupt-Routine ausgeführt wird. Alternativ kann das TOV1 Bit gelöscht werden, in dem man es mit einer `1` überschreibt.

Neben den verschiedenen Steuer- und Statusregistern verfügt die Timer-Einheit T/C1 über die nachfolgend dargestellten vier Funktionsregister.

Bit	15	14	13	12	11	10	9	8	
(0x85)	TCNT1[15:8]								TCNT1H
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-26 Atmel ATmega328p – Timer/Counter 1 Register

Bit	15	14	13	12	11	10	9	8	
(0x89)	OCR1A[15:8]								OCR1AH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x88)	OCR1A [7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-27 Atmel ATmega328p – Timer/Counter 1 Output Compare Register A

Bit	15	14	13	12	11	10	9	8	
(0x8B)	OCR1B[15:8]								OCR1BH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x8A)	OCR1B [7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-28 Atmel ATmega328p – Timer/Counter 1 Output Compare Register B

Bit	15	14	13	12	11	10	9	8	
(0x8B)	ICR1[15:8]								ICR1H
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x8A)	ICR1 [7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 3-29 Atmel ATmega328p – Input Capture Register 1

Da der Atmel ATmega328p lediglich eine interne Datenbusbreite von 8-bit besitzt gilt es beim Lesen bzw. Schreiben der zuvor gezeigten 16-bit Register eine vorgegebene Zugriffreihenfolge einzuhalten. Bei der Initialisierung der Register muss immer zuerst das High-Byte geschrieben werden. Erfolgt hingegen ein Lesezugriff auf eines der oben dargestellten Register, so gilt es immer zuerst das Low-Byte zu lesen.

3.4 ATmega328p – Timer/Counter – Praxisbeispiele

Im folgenden Abschnitt soll nun die Benutzung der Atmel ATmega328p internen Timer-Einheiten anhand konkreter Beispiele demonstriert werden. Die Beschreibung der zu implementierenden Funktionalität erfolgt in allen Fällen durch eine entsprechende grafische Problembeschreibung. Eine Erläuterung der einzelnen Programmschritte ist den jeweiligen Kommentaren zu entnehmen.

3.4.1 ATmega328p - Timer T/C0 Zähler Demo - Polling Modus

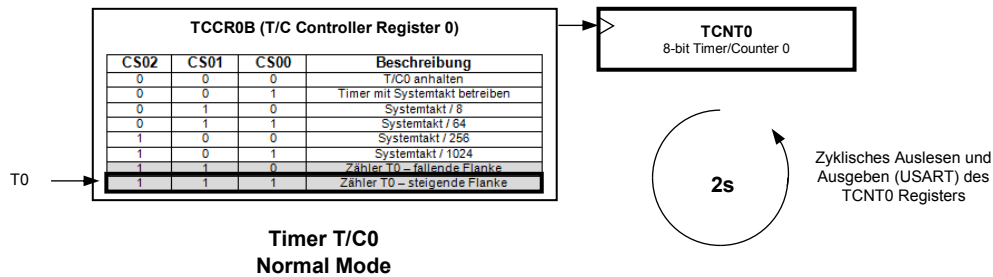


Abbildung 3-30 Problembeschreibung - Timer/Counter-0 ‚Zähler‘ Demo – Polling Modus

```

1. // ATmega328p ,Counter-0 Zähler‘ Demo – Zähleingang PD4 - Polling Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #define BAUDRATE 9600
7. #define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)
8. #define LINE_FEED 10
9.
10. #include <avr/io.h>
11. #include <util/delay.h>
12. #include 'usart.h'
13.
14. int main(void){
15.     unsigned char counter_state;
16.     // -- Initialisierung USART Schnittstelle --
17.     usart_init();
18.     // Pin PD4 als Eingang konfigurieren
19.     DDRD &= ~(1 << DDD4);
20.     // Zähler aktivieren, mit steigender Flanke zählen...
21.     TCCR0B |= (1 << CS02) | (1 << CS01) | (1 << CS00);
22.
23.     while (1){
24.         counter_state = TCNT0;           // Zähler auslesen und
25.         usart_put_byte(counter_state);    // Zählerstand via USART
26.         usart_put_byte(LINE_FEED);       // ausgeben ...
27.         _delay_ms(2000);
28.     }
29.     return 0;
30. }

```

Quellcode 3-1 Atmel ATmega328p – Timer/Counter-0 ‚Zähler‘ Demo – Polling Modus

3.4.2 ATmega328p - Timer T/C0 Zähler Demo - Interrupt Modus

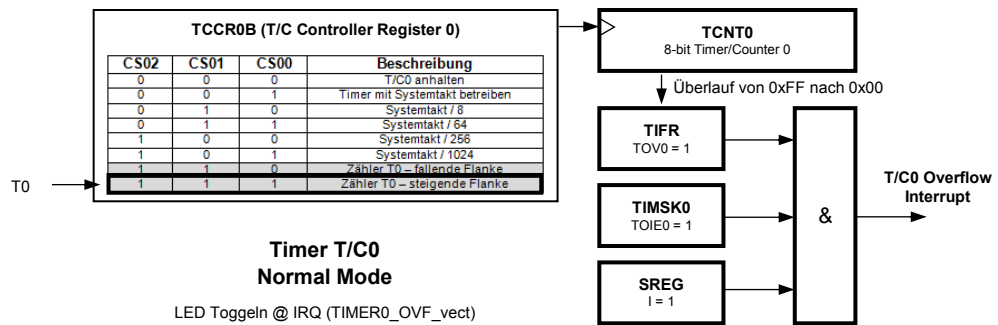


Abbildung 3-31 Problembeschreibung - Timer T/C0 ‚Zähler‘ Demo – Interrupt Modus

```

1. // ATmega328p ,Counter-0 Zähler\ Demo - Zähleingang PD4 - Interrupt Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #include <avr/io.h>
6. #include <avr/interrupt.h>
7.
8. volatile unsigned char overflow;
9.
10. int main(void) {
11.     // -- Konfiguration Status-LED --
12.     DDRB |= (1 << DDB5);          // PIN PB5 als Ausgang konfigurieren
13.     PORTB &= ~(1 << PORTB5);      // LED ausschalten...
14.     // Pin PD4 als Eingang konfigurieren
15.     DDRD &= ~(1 << DDD4);
16.     // Timer Overflow Interrupt aktivieren
17.     TIMSK0 |= (1 << TOIE0);
18.     // Zähler aktivieren, mit steigender Flanke zählen...
19.     TCCR0B |= (1 << CS02) | (1 << CS01) | (1 << CS00);
20.
21.     sei();                        // globale Interrupts aktivieren
22.
23.     while (1) {
24.         if (overflow != 0) {
25.             PORTB ^= (1 << PORTB5);
26.             overflow = 0;
27.         }
28.     }
29.     return 0;
30. }
31.
32. ISR (TIMER0_OVF_vect) {
33.     overflow = 1;
34. }

```

Quellcode 3-2 Atmel ATmega328p – Timer/Counter-0 ‚Zähler‘ Demo – Interrupt Modus

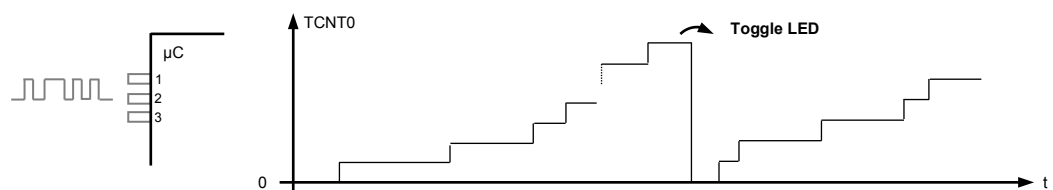


Abbildung 3-32 Atmel ATmega328p – Timer/Counter-0 ‚Zähler‘ Demo – Interrupt Modus

3.4.3 ATmega328p - Timer T/C0 - Overflow Detection - Polling Modus

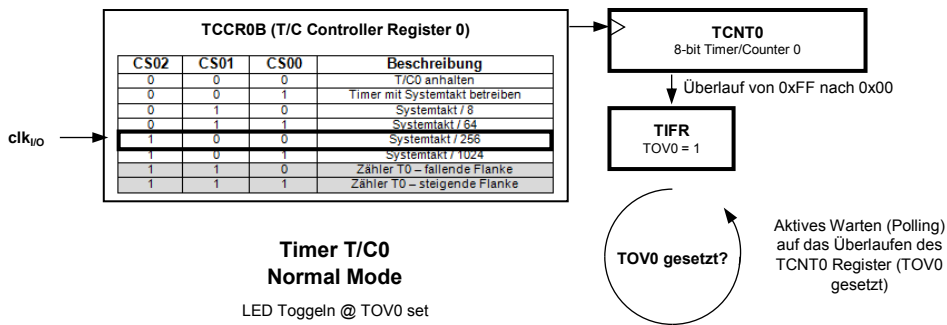


Abbildung 3-33 Problembeschreibung - Timer T/C0 – Overflow Detection – Polling Modus

```

1. // ATmega328p ,Timer T/C0` Demo - Overflow Detektion - Polling Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7.
8. int main(void){
9.     unsigned char test;
10.    // -- Konfiguration Status-LED --
11.    DDRB |= (1 << DDB5); // PIN PB5 als Ausgang konfigurieren
12.    PORTB &= ~(1 << PORTB5); // LED ausschalten...
13.    // -- Konfiguration Timer T/C0 --
14.    TCCR0B = (1 << CS02); // Prescaler (256) setzen und Timer T/C0
15.                          // starten...
16.    while(1){
17.        while ((TIFR0 & (1 << TOV0) ) == 0);
18.        TIFR0 |= (1 << TOV0); // Overflow Flag zurücksetzen
19.        if(test == 11){ // ... und LED togglen
20.            test = 0; // Blinkfrequenz ~10Hz
21.            PORTB ^= (1 << PORTB5);
22.        }
23.        else
24.            test++;
25.    }
26.    return 0;
27. }

```

Quellcode 3-3 Atmel ATmega328p – Timer T/C0 – Overflow Detection – Polling Modus

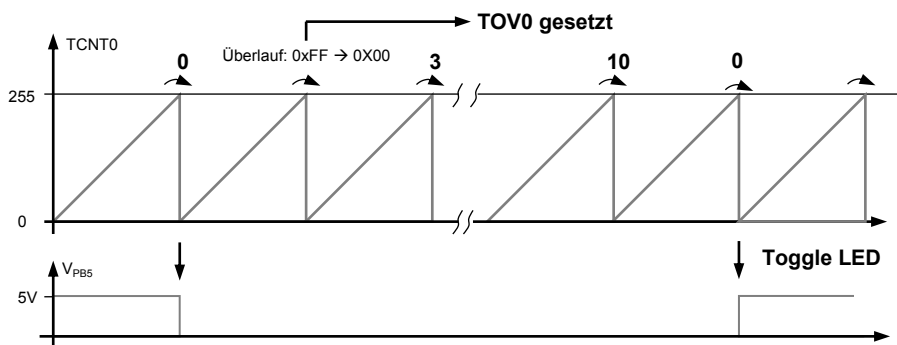


Abbildung 3-34 Atmel ATmega328p – Timer T/C0 – Overflow Detection – Polling Modus

3.4.4 ATmega328p - Timer T/C0 - Overflow Detection - Interrupt Modus

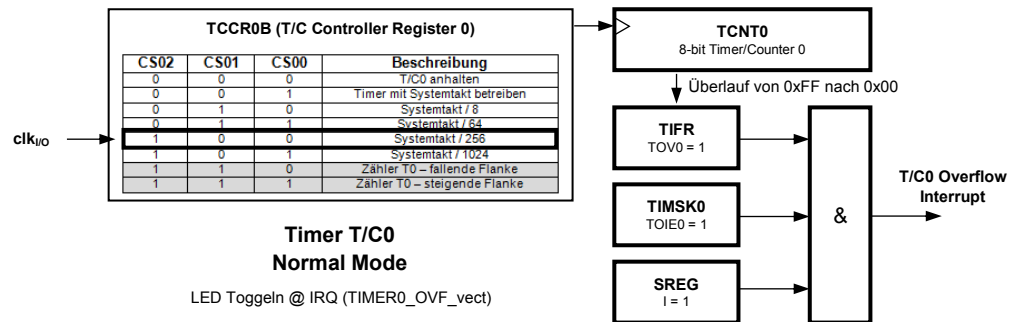


Abbildung 3-35 Problembeschreibung - Timer T/C0 – Overflow Detection – Interrupt Modus

```

1. // ATmega328p ,Timer T/C0` Demo - Overflow Detektion - Interrupt Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7. #include <avr/interrupt.h>
8.
9. volatile unsigned char test; // Interrupt Variable
10.
11. int main(void) {
12.     // -- Konfiguration Status-LED --
13.     DDRB |= (1 << DDB5); // PIN PB5 als Ausgang konfigurieren
14.     PORTB &= ~(1 << PORTB5); // LED ausschalten...
15.     // -- Konfiguration Timer T/C0 --
16.     TIMSK0 |= (1 << TOIE0); // Timer T/C0 Overflow IRQ aktivieren
17.     TCCR0B |= (1 << CS02); // Prescaler (256) setzen und Timer T/C0
18.                             // starten...
19.     sei(); // Globale Interrupts aktivieren
20.
21.     while(1){
22.         if(test == 12){ // Blinkfrequenz ~10Hz, siehe Abb. XX
23.             test = 0;
24.             PORTB ^= (1 << PORTB5); }
25.         return 0;
26.     }
27.
28. ISR (TIMER0_OVF_vect){ test++; }

```

Quellcode 3-4 Atmel ATmega328p – Timer T/C0 – Overflow Detection – Interrupt Modus

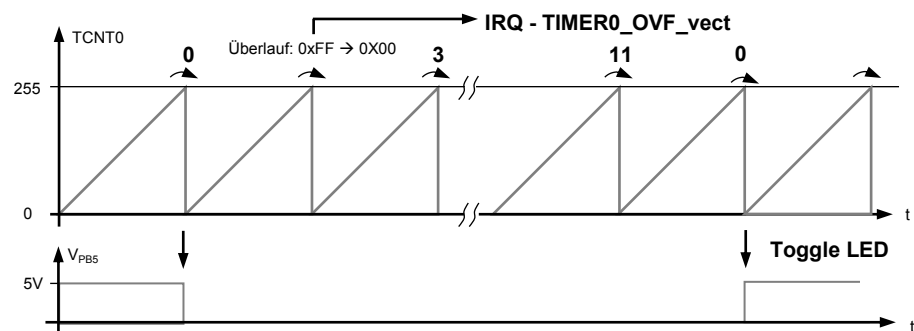


Abbildung 3-36 Atmel ATmega328p – Timer T/C0 – Overflow Detection – Interrupt Modus

3.4.5 ATmega328p - Timer T/C0 - Overflow Interrupt mit Timer-Reload

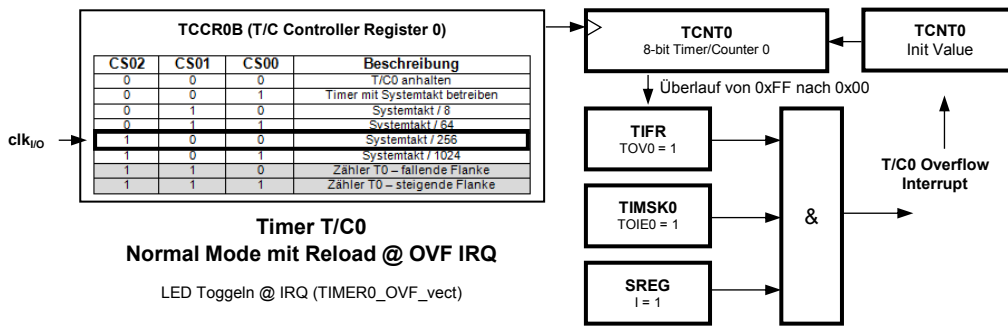


Abbildung 3-37 Problembeschreibung - Timer T/C0 – Overflow Interrupt mit Timer-Reload

```

1. // ATmega328p ,Timer T/C0` Demo - Overflow Interrupt mit Reload
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7. #include <avr/interrupt.h>
8.
9. int main(void) {
10.     // -- Konfiguration Status-LED --
11.     DDRB |= (1 << DDB5); // PIN PB5 als Ausgang konfigurieren
12.     PORTB &= ~(1 << PORTB5); // LED ausschalten...
13.     // -- Konfiguration Timer T/C0 --
14.     TIMSK0 |= (1 << TOIE0); // Timer Overflow Interrupt aktivieren
15.     TCCR0B = (1 << CS02); // Prescaler (256) setzen und Timer T/C0
16.                             // starten
17.     sei();
18.
19.     while(1) {
20.     }
21.     return 0;
22. }
23.
24. ISR (TIMER0_OVF_vect) {
25.     TCNT0 = 193; // resultierende Blinkfrequenz
26.     PORTB ^= (1 << PB5); // 497Hz
27. }

```

Quellcode 3-5 Atmel ATmega328p – Timer T/C0 – Overflow Interrupt mit Timer T/C0 Reload

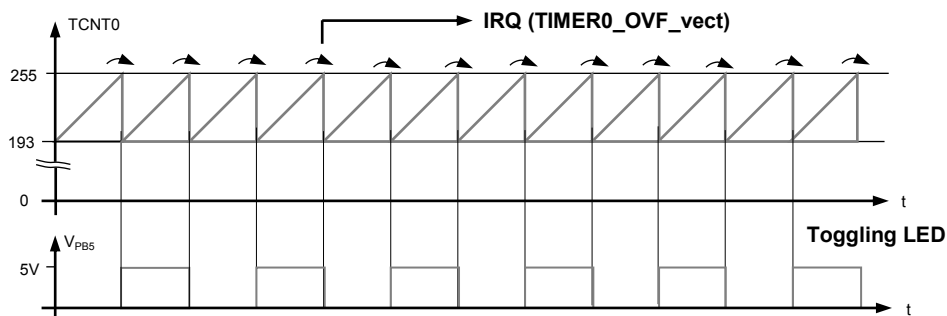


Abbildung 3-38 Atmel ATmega328p - Timer T/C0 – Overflow Interrupt mit Timer T/C0 Reload

3.4.6 ATmega328p - Timer T/C0 - CTC - Polling Modus

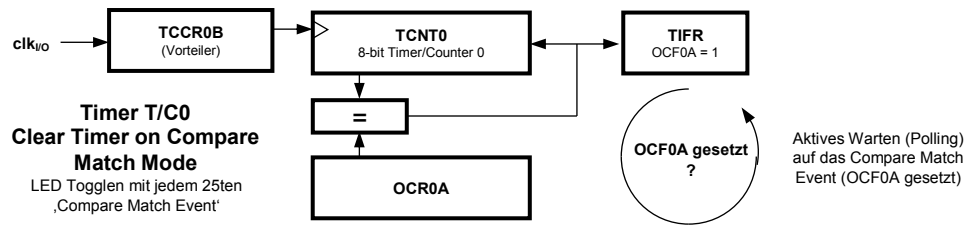


Abbildung 3-39 Problembeschreibung - Timer T/C0 – CTC – Polling Modus

```

1. // ATmega328p ,Timer T/C0` Demo - Clear Timer on Compare - Polling Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7.
8. int main(void) {
9.     unsigned char test = 0;
10.    // -- Konfiguration Status-LED --
11.    DDRB |= (1 << DDB5); // PIN PB5 als Ausgang konfigurieren
12.    PORTB &= ~(1 << PORTB5); // LED ausschalten...
13.    // -- Konfiguration Timer T/C0 --
14.    TCCR0A |= (1 << WGM01); // Clear Timer on Compare...
15.    OCR0A = 125; // Neuen max. Zählwert festlegen
16.    TCCR0B |= (1 << CS02); // Prescaler (256) setzen und
17.                           // Timer T/C0 starten...
18.    while(1) {
19.        // wait for 'compare match flag' ...
20.        while((TIFR0 & (1 << OCF0A) ) == 0);
21.        TIFR0 |= (1 << OCF0A); // compare match flag
22.                               // zurücksetzen...
23.        if(test == 24) {
24.            test = 0; // Blinkfrequenz ~10Hz,
25.            PORTB ^= (1 << PORTB5); //
26.        }
27.        else
28.            test++;
29.    }
30.    return 0;
31. }

```

Quellcode 3-6 ATMEL ATmega328p – Timer T/C0 – Clear Timer on Compare – Polling Modus

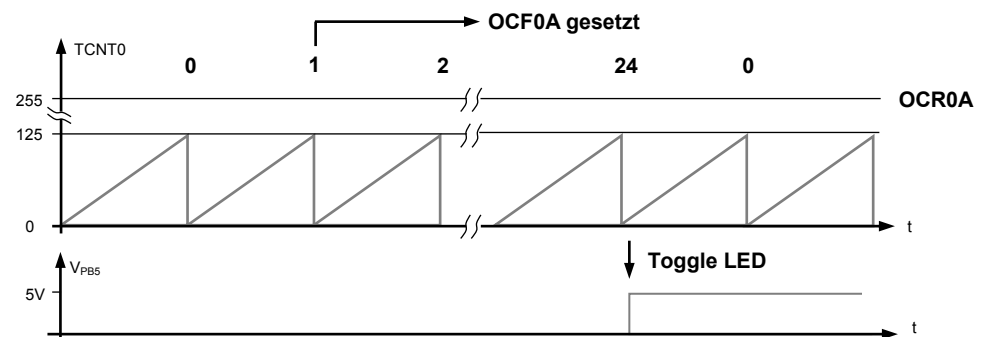


Abbildung 3-40 Atmel ATmega328p – Timer T/C0 – clear Timer on compare – Polling Modus

3.4.8 ATmega328p - Timer T/C0 - CTC - Interrupt Modus

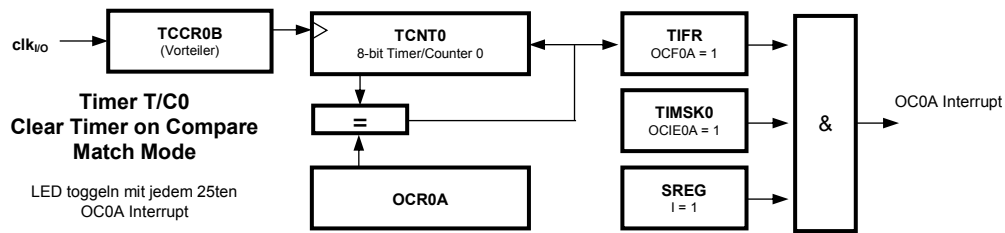


Abbildung 3-41 Problembeschreibung - Timer T/C0 –CTC – Interrupt Modus

```

1. // ATmega328p ,Timer T/C0` Demo - Clear Timer on Compare - Interrupt Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7. #include <avr/interrupt.h>
8.
9. volatile unsigned char test;      // Interrupt Variable
10.
11. int main(void){
12.     // -- Konfiguration Status-LED --
13.     DDRB |= (1 << DDB5);          // PIN PB5 als Ausgang konfigurieren
14.     PORTB &= ~(1 << PORTB5);      // LED ausschalten...
15.     // -- Konfiguration Timer T/C0 --
16.     TCCR0A |= (1 << WGM01);        // Clear Timer on Compare...
17.     OCR0A = 125;                  // Neuen max. Zählwert festlegen
18.
19.     TIMSK0 |= (1 << OCIE0A);       // 'Clear Timer on Compare' Interrupt
20.     TCCR0B |= (1 << CS02);        // aktivieren, Prescaler (256) setzen
21.                                   // und Timer T/C0 starten...
22.
23.     sei();                        // Globale Interrupts aktivieren
24.
25.     while(1){
26.         if(test == 25){            // Blinkfrequenz ~10Hz,
27.             test = 0;
28.             PORTB ^= (1 << PORTB5);
29.         }
30.         return 0;
31.     }
32.
33.     ISR (TIMER0_COMPA_vect){test++; }

```

Quellcode 3-8 Atmel ATmega328p – Timer T/C0 – Clear Timer on Compare – Interrupt Modus

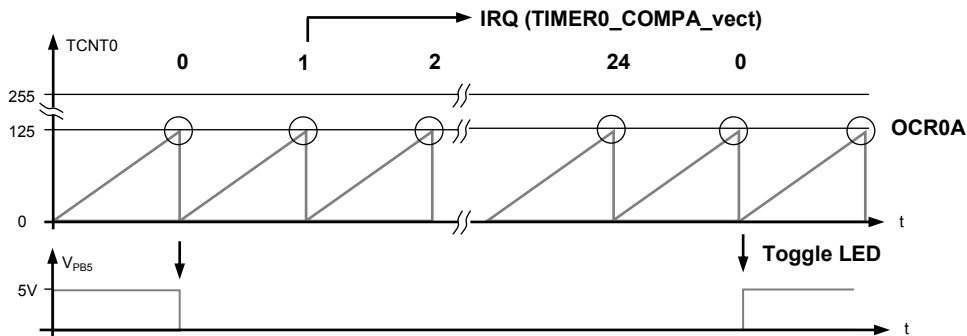


Abbildung 3-42 Atmel ATmega328p – Timer T/C0 – Clear Timer on compare – Interrupt Modus

3.4.9 ATmega328p - Timer T/C1 - CTC - Interrupt Modus

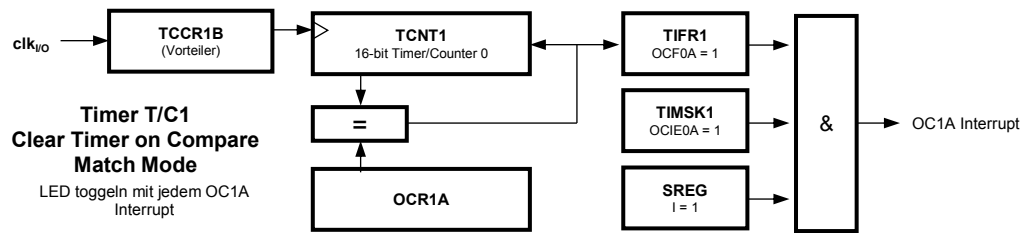


Abbildung 3-43 Problembeschreibung - Timer T/C1 – CTC – Interrupt Modus

```

1. // ATmega328p ,Timer T/C1` Demo - Clear Timer on Compare - Interrupt Modus
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6. #include <avr/io.h>
7. #include <avr/interrupt.h>
8.
9. int main(void) {
10.    // -- Konfiguration Status-LED --
11.    DDRB |= (1 << DDB5);          // PIN PB5 als Ausgang konfigurieren
12.    PORTB &= ~(1 << PORTB5);
13.    // -- Konfiguration Timer T/C1 --
14.    OCR1A = 0x3D08;                // Neuen max. Zählwert setzen,
15.    TCCR1B |= (1 << WGM12);        // Mode 4, CTC on OCR1A
16.    TIMSK1 |= (1 << OCIE1A);       // OCIE1A Interrupt aktivieren
17.    TCCR1B |= (1 << CS12);         // Prescaler (1024) setzen und
18.    TCCR1B |= (1 << CS10);         // Timer T/C1 starten
19.
20.    sei();                          // Globale Interrupts aktivieren
21.
22.    while(1) {
23.    }                                // ISR Freq. = (16MHz/1024)/(OCR1A+1)
24.    return 0;                       // = 1 Hz
25. }                                  // Blinkfrequenz LED@PB5: 0.5Hz
26.
27. ISR (TIMER1_COMPA_vect){PORTB ^= (1 << PORTB5);}

```

Quellcode 3-9 Atmel ATmega328p – Timer T/C1 – Clear Timer on Compare – Interrupt Modus

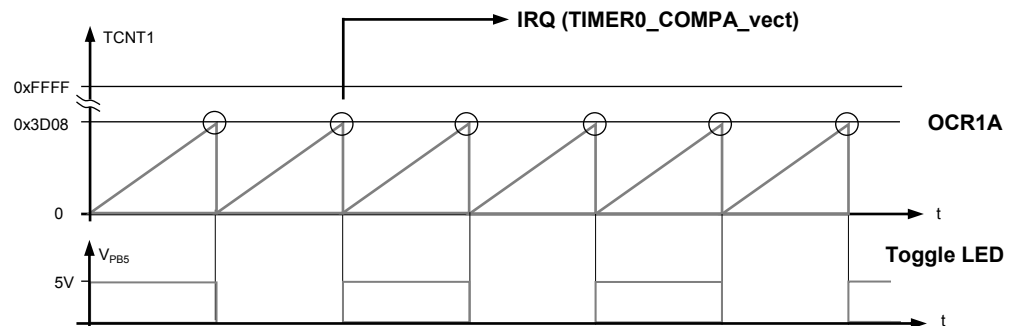


Abbildung 3-44 Atmel ATmega328p – Timer T/C1 – Clear Timer on compare – Interrupt Modus

3.4.10 ATmega328p - Timer T/C0 - nicht inv. Fast PWM Modus #1

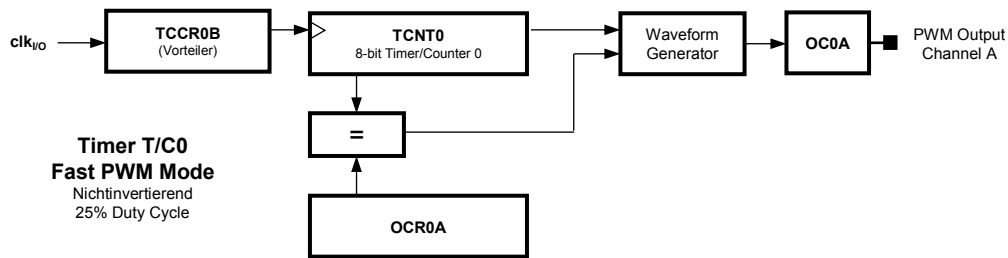


Abbildung 3-45 Problembeschreibung - Timer T/C0 – nicht inv. Fast PWM Modus #1

```

1. // ATmega328p ,Timer T/C0 PWM` Demo - 8kHz Fast PWM - 25% Duty Cycle
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. #include <avr/io.h>
6.
7. int main(void){
8.     // -- Konfiguration PWM Ausgang --
9.     DDRD |= (1 << DDD6);           // PIN PD6 als Ausgang konfigurieren
10.    // -- Konfiguration Timer T/C0 --
11.    OCR0A = 64;                     // 25% Duty Cycle einstellen
12.    TCCR0A |= (1 << COM0A1);        // ,nicht invertiertes` PWM Signal
13.    TCCR0A |= (1 << WGM01);         // `Fast PWM` Modus auswählen
14.    TCCR0A |= (1 << WGM00);
15.    TCCR0B |= (1 << CS01);          // Prescaler (8) setzen und starten...
16.
17.    while(1){                       // resultierendes PWM Signal:
18.    }                                // ~8kHz, 25% Duty Cycle
19.    return 0;
20. }
```

Quellcode 3-10 ATMEL ATmega328p – Timer T/C0 – nicht invertierender Fast PWM Modus

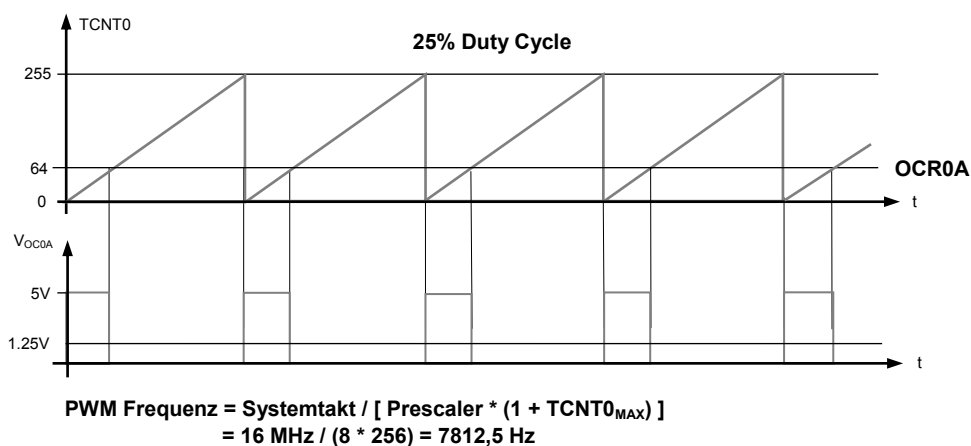


Abbildung 3-46 Atmel ATmega328p – Timer T/C0 – nicht invertierender Fast PWM Modus

3.4.11 ATmega328p - Timer T/C0 - nicht inv. Fast PWM Modus #2

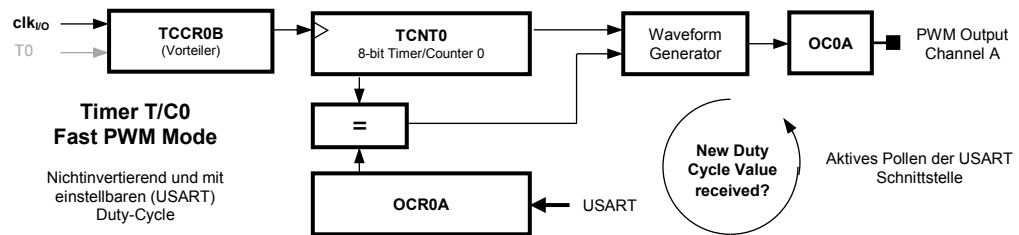


Abbildung 3-47 Problembeschreibung - Timer T/C0 – nicht inv. Fast PWM Modus #2

```

1. // ATmega328p ,Timer T/C0 PWM` Demo - 8kHz Fast PWM - PWM Duty Cycle ist
2. // über die USART-Schnittstelle frei einstellbar...
3. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
4. // by Dr. C. Jakob, fbeit, h_da, Januar 2015, christian.jakob@h-da.de
5.
6. #include <avr/io.h>
7. #include 'usart.h'
8.
9. int main(void) {
10.     unsigned char pwm_value = 64; // Default PWM Wert...
11.     // -- Konfiguration PWM Ausgang --
12.     DDRD |= (1 << DDD6);           // PIN PD6 als Ausgang konfigurieren
13.     // -- Fast PWM Konfiguration Timer T/C0 --
14.     OCR0A = pwm_value;              // Default Duty Cycle einstellen
15.     TCCR0A |= (1 << COM0A1);        // ,nicht invertiertes` PWM Signal
16.     TCCR0A |= (1 << WGM01);         // `Fast PWM` Modus auswählen
17.     TCCR0A |= (1 << WGM00);
18.     TCCR0B |= (1 << CS01);          // Prescaler (8) setzen und starten...
19.     // -- USART initialisieren --
20.     usart_init();
21.     // 8kHz PWM, über USART frei einstellbarer Duty Cycle
22.     while(1) {
23.         pwm_value = usart_receive();
24.         OCR0A      = pwm_value;
25.     }
26.     return 0;
27. }

```

Quellcode 3-11 ATMEL ATmega328p – Timer T/C0 – nicht inv. Fast PWM Modus, über USART frei einstellbarer PWM Duty Cycle

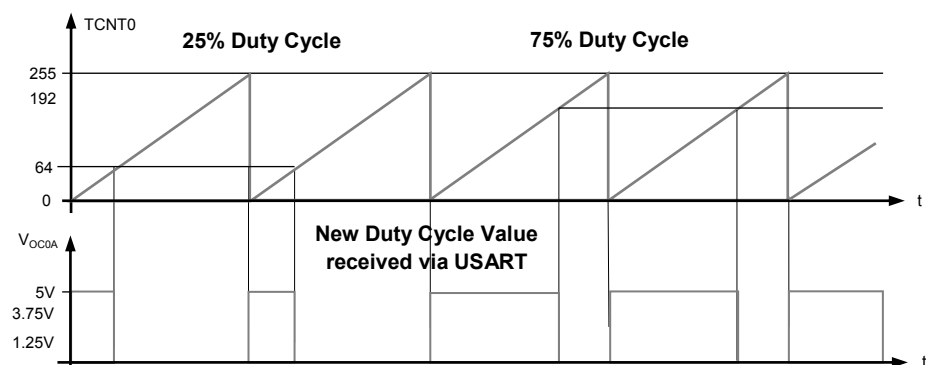


Abbildung 3-48 Atmel ATmega328p – Timer T/C0 – nicht inv. Fast PWM Modus, über USART frei einstellbarer PWM Duty Cycle

3.5 ATmega328p – Timer/Counter – Weiterführende Beispiele

Der folgende Abschnitt diskutiert den Einsatz der ATmega328p internen Timer-Peripherie zum Dimmen einer angeschlossenen LED. Betrachten wir zuerst die Ausgabe eines PWM-Signals und berechnen dessen Mittelwert:

$$V_{OUT,avg} = \frac{1}{T_{SW}} \int_0^{T_{SW}} f(t) dt$$

Der Ausdruck T_{SW} steht hier für die Periode des PWM-Signals. Allgemein kann der Verlauf des pulswiden modulierten Signals wie folgt beschrieben werden:

$$f(t) = \begin{cases} V_{IN}, & t \leq T_{ON} \\ 0, & t > T_{ON} < T_{SW} \end{cases}$$

Das Lösen des oben aufgeführten Integrals führt somit zu:

$$\begin{aligned} V_{OUT,avg} &= \frac{1}{T_{SW}} \left(\int_0^{T_{ON}} V_{IN} dt + \int_{T_{ON}}^{T_{SW}} 0 dt \right) \\ &= \frac{1}{T_{SW}} ([V_{IN}t]_0^{T_{ON}} + [0t]_{T_{ON}}^{T_{SW}}) \end{aligned}$$

was letztlich den nachfolgenden Ausdruck ergibt:

$$V_{OUT,avg} = \frac{V_{IN}T_{ON}}{T_{SW}} = DV_{IN}$$

Die eingestellte Pulsweite bestimmt folglich den Mittelwert des PWM-Signals und somit, wie nachfolgend dargestellt, die mittlere Spannung am PWM-Signalausgang des Mikrocontrollers.

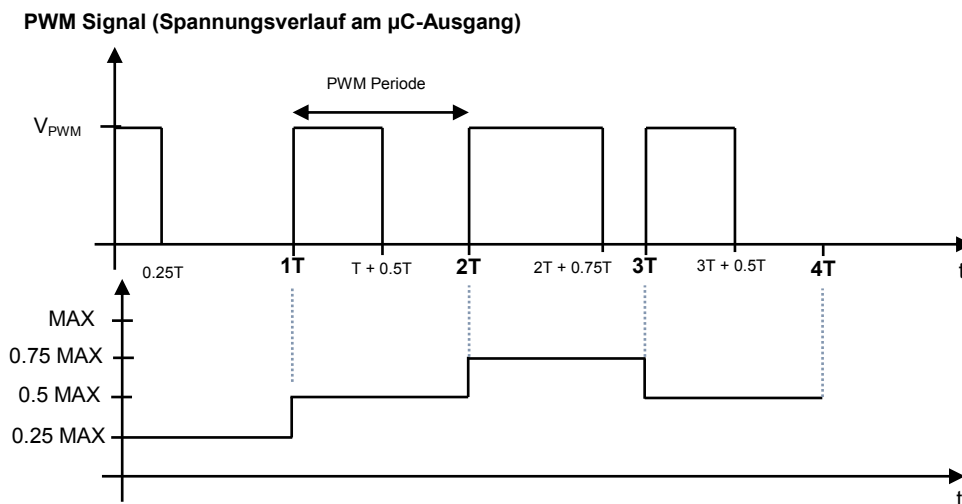


Abbildung 3-49 PWM-Signalverlauf

Steuert man eine LED mittels eines pulswidenmodulierten Signals an, so kann über das Setzen der Pulsbreite die Helligkeit der LED bestimmt werden. Die PWM-Frequenz gilt es in diesem Zusammenhang so zu wählen, dass kein Flackern mehr wahrnehmbar ist.

Betrachten wir in diesem Zusammenhang folgendes Szenario: Die an den PWM-Ausgang des μC angeschlossene LED soll in ihrer Helligkeit kontinuierlich variieren, d.h. sie soll so angesteuert werden, dass sie langsam heller und wieder dunkler wird. Dieser Vorgang nennt LED Fading und soll sich dauerhaft wiederholen. Auf den ersten Blick erscheint diese Aufgabe recht einfach. Ein naheliegender Ansatz wäre im Falle einer 8-bit PWM die Pulsbreite linear von 0 auf 255 zu erhöhen und anschließend wieder mit gleicher Schrittweite zu dekrementieren. Das Problem offenbart sich jedoch recht schnell. Ein lineares Inkrementieren der Pulsweite führt nicht zu einem linearen Anstieg der LED Helligkeit! Realisiert man diesen Ansatz stellt man fest, dass die LED bereits bei relativ kleinen Pulsbreiten ($64/255$ – 8-bit PWM) voll erleuchtet.

Der Grund hierfür liegt in der Helligkeitswahrnehmung des menschlichen Auges. Diese ist nichtlinear, genauer gesagt: sie ist nahezu logarithmisch. Das ermöglicht die Wahrnehmung eines sehr großen Helligkeitsbereichs, angefangen eines Vollmonds mit $\sim 1/4$ Lux über eine normale Schreibtischbeleuchtung mit ca. 750 Lux bis hin zu einem hellen Sommertag mit bis zu 100.000 Lux. Praktisch bedeutet dieser Umstand, dass wir dem Auge des Betrachters große physikalische Helligkeitsunterschiede präsentieren müssen, damit dieser schlussendlich eine lineare Helligkeitsteigerung wahrnimmt. Inkrementiert man folglich die Pulsbreite nicht linear, sondern exponentiell, so ergibt sich für den Betrachter eine nahezu lineare Steigerung der LED Helligkeit.

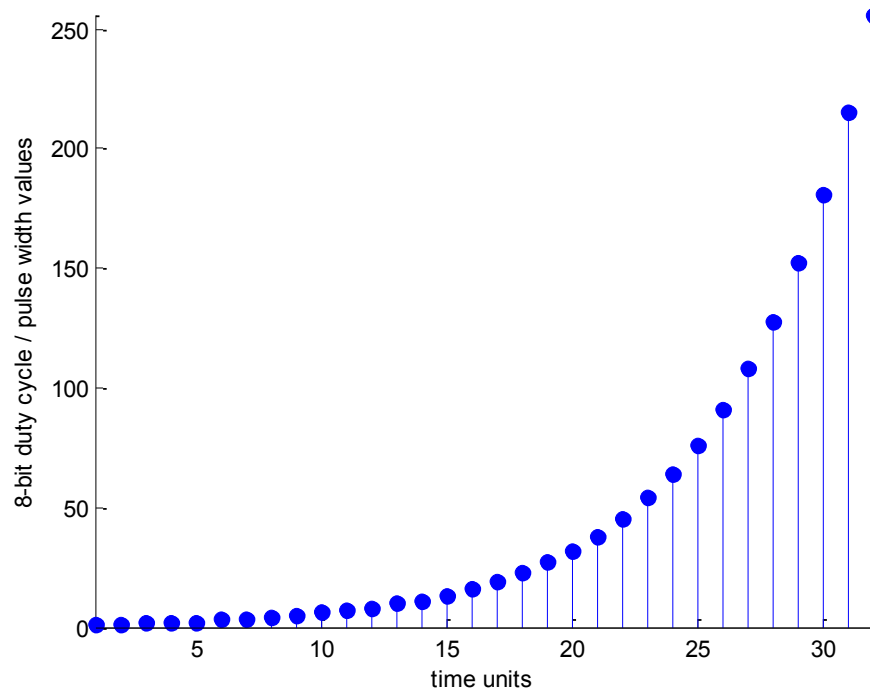


Abbildung 3-50 Exponentieller PWM Duty-Cycle Verlauf

Eine mögliche Implementierung der zuvor beschriebenen Funktionalität auf Basis des 16-bit Timers T/C1 ist in Quellcode 3.12 dargestellt. Eine Beschreibung der einzelnen Programmschritte ist den Kommentaren zu entnehmen. Die Berechnung der Duty-Cycle-Werte erfolgte mittels Microsoft Excel.

```
1. // ATmega328p ,PWM Demo` LED Fading - exp. Duty Cycle Variation
2. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
3. // by Dr. C. Jakob, fbeit, h_da, Sptember 2015, christian.jakob@h-da.de
4.
5. #define F_CPU 16000000UL
6.
7. #include <avr/io.h>
8. #include <util/delay.h>
9. #include <avr/pgmspace.h>
10.
11. const unsigned short int pwm_values[256] PROGMEM = {
12. 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
13. 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 7,
14. 7, 7, 8, 8, 8, 9, 9, 10, 10, 10, 11, 11, 12, 12, 13, 13, 14, 15,
15. 15, 16, 17, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
16. 32, 33, 35, 36, 38, 40, 41, 43, 45, 47, 49, 52, 54, 56, 59,
17. 61, 64, 67, 70, 73, 76, 79, 83, 87, 91, 95, 99, 103, 108, 112,
18. 117, 123, 128, 134, 140, 146, 152, 159, 166, 173, 181, 189, 197,
19. 206, 215, 225, 235, 245, 256, 267, 279, 292, 304, 318, 332, 347,
20. 362, 378, 395, 412, 431, 450, 470, 490, 512, 535, 558, 583, 609,
21. 636, 664, 693, 724, 756, 790, 825, 861, 899, 939, 981, 1024, 1069,
22. 1117, 1166, 1218, 1272, 1328, 1387, 1448, 1512, 1579, 1649, 1722,
23. 1798, 1878, 1961, 2048, 2139, 2233, 2332, 2435, 2543, 2656, 2773,
24. 2896, 3025, 3158, 3298, 3444, 3597, 3756, 3922, 4096, 4277, 4467,
25. 4664, 4871, 5087, 5312, 5547, 5793, 6049, 6317, 6596, 6889, 7194,
26. 7512, 7845, 8192, 8555, 8933, 9329, 9742, 10173, 10624, 11094,
27. 11585, 12098, 12634, 13193, 13777, 14387, 15024, 15689, 16384,
28. 17109, 17867, 18658, 19484, 20346, 21247, 22188, 23170, 24196,
29. 25267, 26386, 27554, 28774, 30048, 31378, 32768, 34218, 35733,
30. 37315, 38967, 40693, 42494, 44376, 46340, 48392, 50534, 52772,
31. 5108, 57548, 60096, 62757, 65535 };
32.
33. int main(void) {
34.     unsigned char tmp;
35.     DDRB |= (1 << DDB1);
36.     TCCR1A = (1 << COM1A1) | (1 << WGM11);
37.     TCCR1B = 0;
38.     ICR1 = 0xFFFF;
39.     TCCR1B |= (1 << WGM12) | (1 << WGM13) | (1 << CS10);
40.
41.     while(1) {
42.         for(tmp = 0; tmp < 254; tmp++){
43.             OCR1A = pgm_read_word(&pwm_values[tmp]);
44.             _delay_ms(4);
45.             tmp++; }
46.         _delay_ms(100);
47.         for(tmp = 0; tmp < 254; tmp++){
48.             OCR1A = pgm_read_word(&pwm_values[255-tmp]);
49.             _delay_ms(4);
50.             tmp++; }
51.     }
52.     return 0;
53. }
```

Quellcode 3-12 ATMEL ATmega328p – Timer T/C1 – LED Fading

Über die Variation der Pulsbreite ist es folglich möglich eine zeitlich variierende Spannung zu erzeugen die im Mittel einem bestimmten Funktionsverlauf entspricht. Nachfolgend soll die Generierung einer Sinusspannung betrachtet werden. Eine Implementierung dieser Funktionalität ist in Quellcode 3.13 aufgezeigt.

```

1. // ATmega328p ,Timer T/C0 PWM SINUS` Demo - 8kHz Fast PWM - Erster
2. // Abschnitt des Quellcodes, Fortsetzung auf der folgenden Seite
3. // Zielplattform: ATMEL ATmega328p, Arduino Uno R3 SMD Edition
4. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
5.
6. #define F_CPU 16000000UL
7. #define SAMPLING_RATE 8000
8.
9. #include <avr/interrupt.h>
10. #include <avr/io.h>
11. #include <avr/pgmspace.h>
12.
13. const unsigned char sine[] PROGMEM = {0x80,0x83,0x86,0x89,0x8D,0x90,0x93,
14.    0x96,0x99,0x9C,0x9F,0xA2,0xA5,0xA8,0xAB,0xAE,0xB1,0xB4,0xB7,0xBA,0xBD,
15.    0xBF,0xC2,0xC5,0xC7,0xCA,0xCD,0xCF,0xD1,0xD4,0xD6,0xD9,0xDB,0xDD,0xDF,
16.    0xE1,0xE3,0xE5,0xE7,0xE9,0xEB,0xEC,0xEE,0xF0,0xF1,0xF3,0xF4,0xF5,0xF6,
17.    0xF8,0xF9,0xFA,0xFB,0xFC,0xFC,0xFD,0xFE,0xFE,0xFF,0xFF,0xFF,0xFF,
18.    0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xFE,0xFD,0xFD,0xFC,0xFB,0xFA,
19.    0xF9,0xF8,0xF7,0xF6,0xF5,0xF3,0xF2,0xF0,0xEF,0xED,0xEC,0xEA,0xE8,0xE6,
20.    0xE4,0xE2,0xE0,0xDE,0xDC,0xDA,0xD7,0xD5,0xD3,0xD0,0xCE,0xCB,0xC9,0xC6,
21.    0xC3,0xC1,0xBE,0xBB,0xB8,0xB5,0xB3,0xB0,0xAD,0xAA,0xA7,0xA4,0xA1,0x9E,
22.    0x9B,0x98,0x94,0x91,0x8E,0x8B,0x88,0x85,0x82,0x7E,0x7B,0x78,0x75,0x72,
23.    0x6F,0x6C,0x68,0x65,0x62,0x5F,0x5C,0x59,0x56,0x53,0x50,0x4D,0x4B,0x48,
24.    0x45,0x42,0x3F,0x3D,0x3A,0x37,0x35,0x32,0x30,0x2D,0x2B,0x29,0x26,0x24,
25.    0x22,0x20,0x1E,0x1C,0x1A,0x18,0x16,0x14,0x13,0x11,0x10,0x0E,0x0D,0x0B,
26.    0x0A,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x03,0x02,0x02,0x01,0x01,0x00,
27.    0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x01,0x02,0x02,0x03,0x04,0x04,
28.    0x05,0x06,0x07,0x08,0x0A,0x0B,0x0C,0x0D,0x0F,0x10,0x12,0x14,0x15,0x17,
29.    0x19,0x1B,0x1D,0x1F,0x21,0x23,0x25,0x27,0x2A,0x2C,0x2F,0x31,0x33,0x36,
30.    0x39,0x3B,0x3E,0x41,0x43,0x46,0x49,0x4C,0x4F,0x52,0x55,0x58,0x5B,0x5E,
31.    0x61,0x64,0x67,0x6A,0x6D,0x70,0x73,0x77,0x7A,0x7D,0x80};
32.
33. volatile unsigned char sine_index;
34.
35. ISR(TIMER1_COMPA_vect) {
36.    OCR0A = pgm_read_byte(&sine[sine_index]);
37.    sine_index++;
38. }
39.
40. int main(void){
41.    // -- Konfiguration PWM Ausgang --
42.    DDRD |= (1 << DDD6); // PIN PD6 als Ausgang konfigurieren
43.    // -- Fast PWM Konfiguration Timer T/C0 --
44.    TCCR0A |= (1 << WGM01); // Fast PWM Mode einstellen
45.    TCCR0A |= (1 << WGM00);
46.    TCCR0B &= ~(1 << WGM02); // Max. Zählwert - 0xFF
47.    TCCR0A |= (1 << COM0A1); // nicht invertierender PWM Modus
48.    TCCR0B &= ~(1 << COM0B1); // OC0B nicht benutzen
49.    TCCR0B &= ~(1 << COM0B0);
50.    TCCR0B |= (1 << CS00); // Keinen Prescaler verwenden
51.    //Initiale Pulsweite definieren
52.    OCR0A = pgm_read_byte(&sine[0]);
53.
54.    // ##### FORTSETZUNG AUF DER NÄCHSTEN SEITE #####
55.

```

Quellcode 3-13 ATMEL ATmega328p – Timer T/C0 – Sinussignal-Generator Teil - 1

```

57. // -- Konfiguration Timer T/C1 --
58. TCCR1B |= (1 << WGM12); // T/C1 Clear Timer on Compare Match
59. // Mode setzen
60. TCCR1B |= (1 << WGM12); // Max. Zählerwert - OCR1A
61. TCCR1B |= (1 << CS10); // Kein Prescaler verwenden
62.
63. OCR1A = F_CPU/SAMPLING_RATE; // 16e6 / 8000 = 2000, der Timer T/C1
64. // wird somit mit einer Frequenz von
65. // 8kHz aufgerufen...
66. TIMSK1 |= (1 << OCIE1A); // Interrupt auslösen wenn Timer TCNT1
67. // gleich dem OCR1A ist
68. sine_index = 0;
69.
70. sei(); // Globale Interrupts aktivieren
71.
72. while(1){ // Resultierender Sinus besitzt eine
73. } // Frequenz von 8kHz/256 = 31.25Hz
74. return 0;
75. }

```

Quellcode 3-13 ATMEEL ATmega328p – Timer T/C0 – Sinussignal-Generator Teil - 2

Ein im Mittel sinusförmiger Spannungsverlauf an einem PWM-Ausgang kann folglich durch eine sinusförmige Variation der PWM-Pulsbreite realisiert werden. Die Änderung des PWM Duty-Cycles wird hierbei durch die Verwendung eines zweiten Timers realisiert. Die in Quellcode 3.13 verwendeten Sinuswerte wurden im vorliegenden Fall durch das nachfolgende MATLAB Skript erzeugt.

```

1. // Einfaches MATLAB Skript zur Generierung von 256 Sinuswerten...
2. // Siehe ATMEEL ATmega328p - PWM Demo - Sinusgenerator
3. // by Dr. C. Jakob, fbeit, h_da, September 2015, christian.jakob@h-da.de
4.
5. t = linspace(0,1,256);
6. f_t = (sin(2*pi*1*t)+1)*128;
7. f_t = round(cast(f_t,'uint8'));
8. plot(t,f_t);
9. fileID = fopen('C:\sine_demo','w');
10. for i = 1:256;
11.     if i == 256;
12.         fprintf(fileID,'0x%02X',f_t(i));
13.     else
14.         fprintf(fileID,'0x%02X,',f_t(i));
15.     end
16.     if mod(i,16) == 0
17.         fprintf(fileID,'\n');
18.     end
19. end
20. fclose(fileID);

```

Quellcode 3-14 MATLAB Skript zur Generierung von 256 Sinuswerten

An diesem Punkt soll noch einmal darauf hingewiesen werden, dass man mit dem oben beschriebenen Verfahren lediglich eine sinusförmige Änderung des PWM-Signals erzielt. Gilt es eine sinusförmige Spannung zu erzeugen, so muss der PWM-Ausgang mit einem Tiefpassfilter zur Mittelwertbildung beschaltet werden.

Wichtig ist in diesem Zusammenhang die richtige Wahl der Grenzfrequenz des Tiefpassfilters (wir beziehen uns nachfolgend auf einen passiven RC-Tiefpassfilter erster Ordnung). Diese Frequenz wählt man in der Regel deutlich kleiner als die PWM-Frequenz, um die im PWM-Rechtecksignal enthaltene Grundschwingung sowie all ihre Oberschwingungen weitgehend zu unterdrücken und nur den Gleichspannungsanteil des PWM-Signals zu erhalten. Zu klein darf man die Grenzfrequenz allerdings auch nicht wählen, da dies die Einschwingzeit auf einen neuen Ausgangswert deutlich verlängert. Man verbessert zwar hierdurch die Dämpfung der PWM Grund- sowie der Oberschwingungen, verschlechtert jedoch deutlich die zeitliche Dynamik des Signals. Dieser Zusammenhang soll nachfolgend näher betrachtet werden.

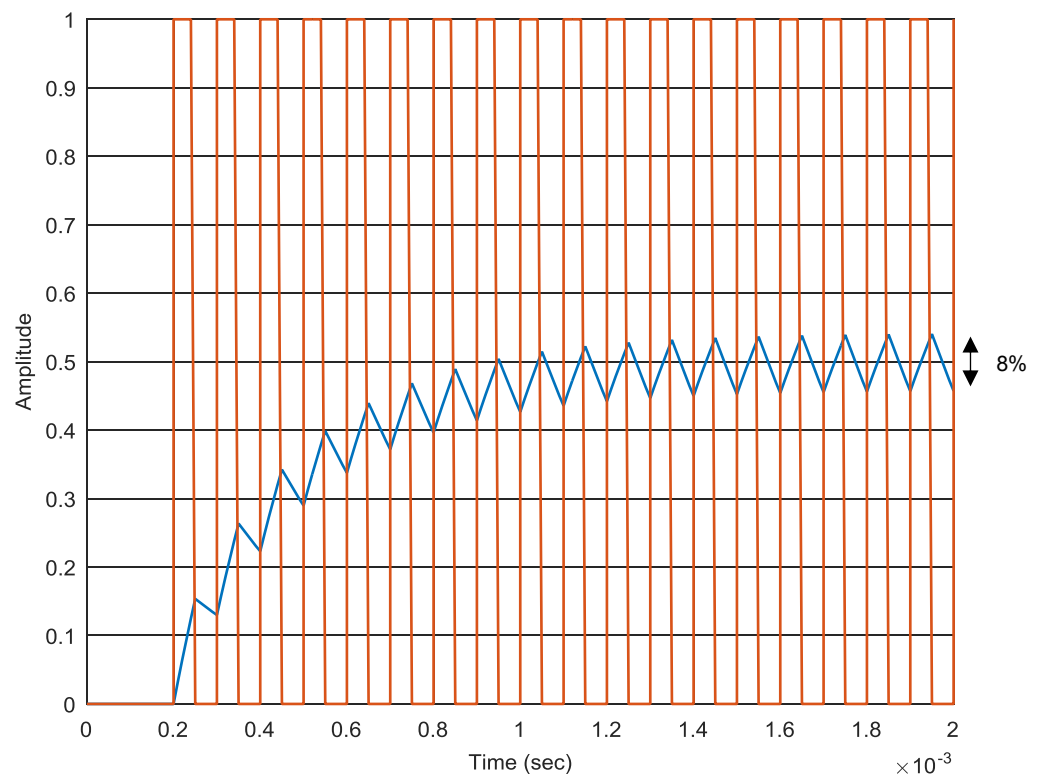


Abbildung 3-50 PWM-Signal mit dargestelltem mittlerem Spannungsverlauf

Abbildung 3.50 zeigt in roter Farbe das PWM-Signal. Dieses wird auf den Eingang eines einfachen RC-Tiefpassfilters erster Ordnung gegeben. Das resultierende Ausgangssignal dieses Filters ist in Abbildung 3.50 in blauer Farbe dargestellt und entspricht dem mittleren Spannungsverlauf des PWM-Eingangssignals. Der Filter besitzt einen RC-Wert von $300\mu\text{s}$ sowie eine Grenzfrequenz von 500Hz . Das dargestellte PWM-Signal besitzt eine Frequenz von 10kHz und ändert nach 0.2ms seine Pulsbreite von 0% auf 50% . Das resultierende Ausgangssignal besitzt eine recht geringe Restwelligkeit, d.h. die hohen Frequenzanteile des Eingangssignals wurden ausreichend gut gedämpft. Jedoch dauert der Einschwingvorgang, wie in Abbildung 3.50 zu sehen, in Summe 10 PWM-Perioden.

Reduziert man hingegen den RC-Wert des Tiefpassfilters ($30\mu\text{s}$) und vergrößert somit die Grenzfrequenz (5kHz), so verbessert sich zwar, wie in Abbildung 3.51 gezeigt, das Einschwingverhalten des Filters deutlich (erfolgt in zwei PWM-Perioden), jedoch besitzt das Ausgangssignale eine erhebliche Restwelligkeit.

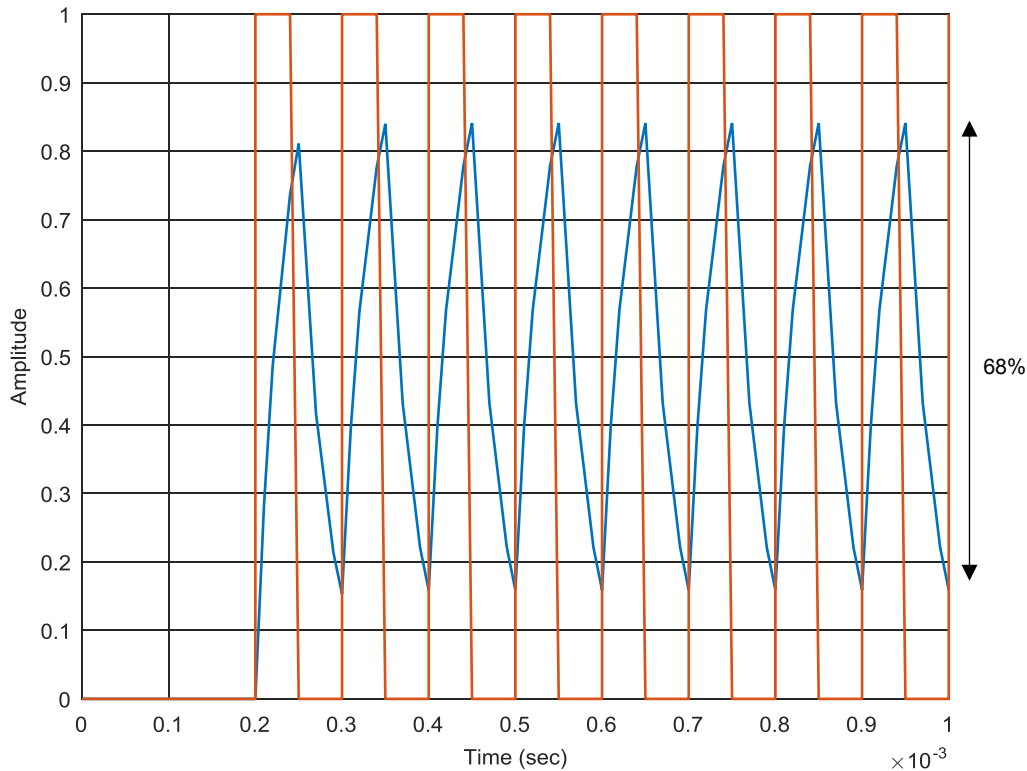


Abbildung 3-51 PWM-Signal mit dargestelltem mittlerem Spannungsverlauf

In Abbildung 3.52 ist ein PWM-Signal mit einer Frequenz von 100kHz in roter Farbe dargestellt. Die Pulsbreite dieses Signals variiert sinusförmig mit einer Frequenz von 1000Hz (schwarzer Kurvenverlauf in Abbildung 3.52). Dieses Signal wird nun auf einen RC-Tiefpassfilter mit einem RC-Wert von $40\mu\text{s}$ gegeben. Das resultierende Ausgangssignal ist in blauer Farbe dargestellt. Wie hier zu sehen folgt das Ausgangssignal (blau) dem mittleren Spannungsverlauf der PWM-Eingangsspannung (schwarz) hinreichend gut. Erhöht man hingegen nun bei gleichbleibender Filterkonfiguration die Frequenz der sinusförmigen Pulsbreitenvariation auf 8000Hz so kommt es, wie in Abbildung 3.53 zu sehen, zu deutlichen Signalverzerrungen.

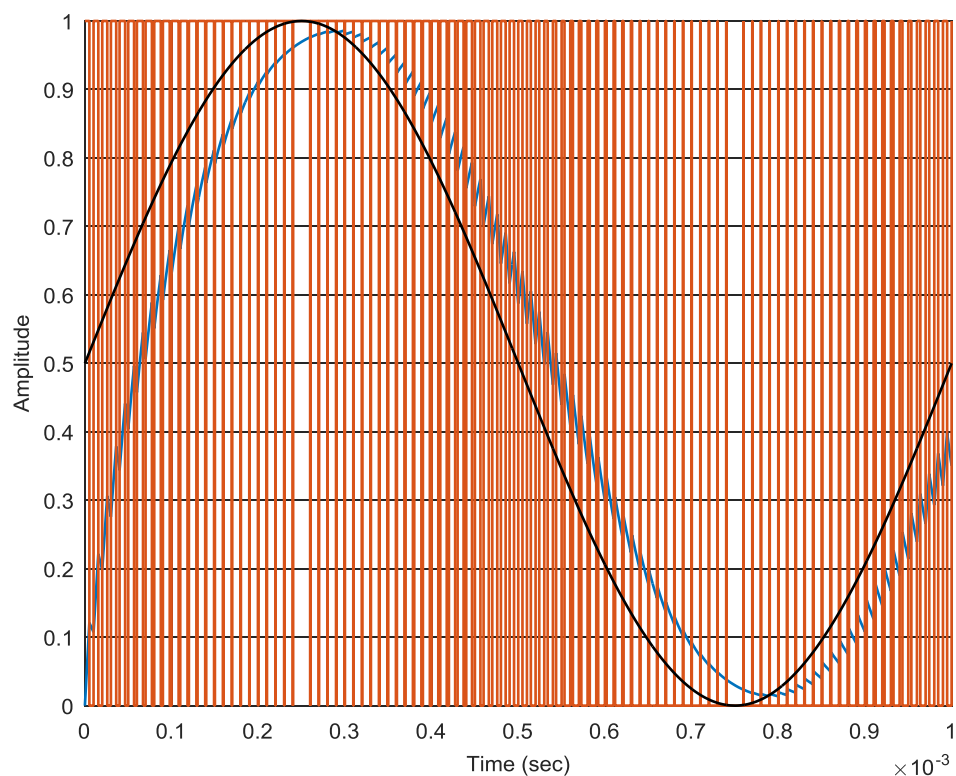


Abbildung 3-52 PWM-Signal (mit sinusförmiger Duty-Cycle Änderung) mit dargestellter mittl. Spannung (RC-Wert: $40\mu\text{s}$, PWM-Freq. – 100kHz, PWM-Duty-Cycles Freq.: 1000Hz)

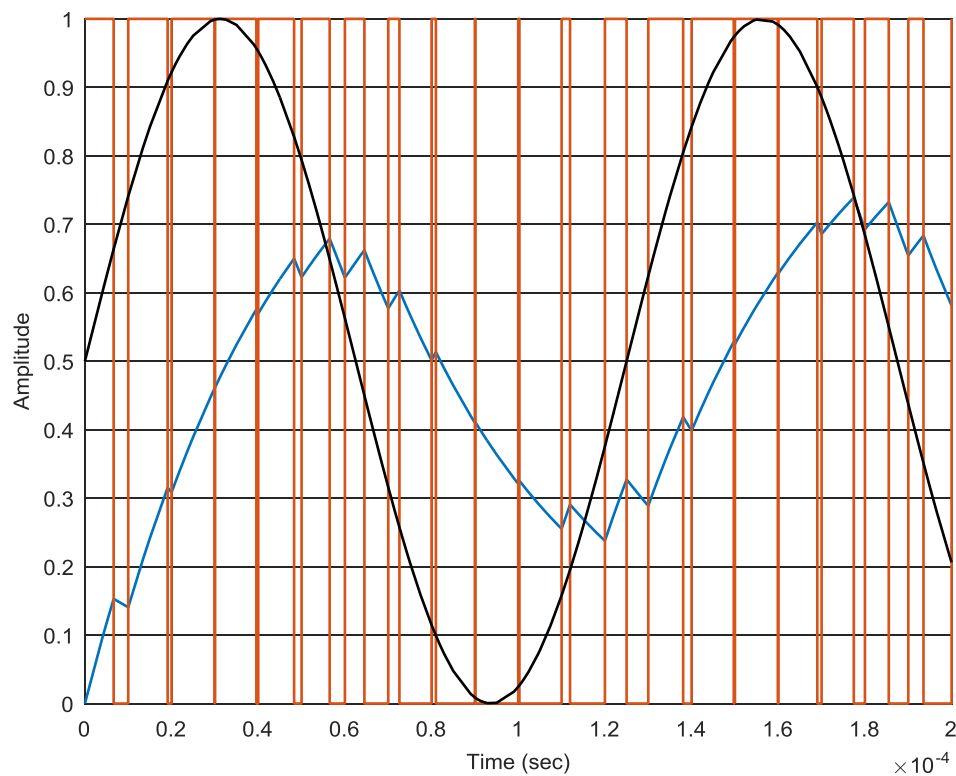


Abbildung 3-53 PWM-Signal (mit sinusförmiger Duty-Cycle Änderung) mit dargestellter mittl. Spannung (RC-Wert: $40\mu\text{s}$, PWM-Freq. – 100kHz, PWM-Duty-Cycles Freq.: 8000Hz)

3.6 Übungsaufgaben

1. Erläutern Sie die grundlegenden Funktionen der einzelnen Timer-Einheiten im Atmel ATmega328p und diskutieren Sie mögliche Einsatzgebiete.
2. Wie arbeitet ein Timer im 'Normal Mode'?
3. Wie arbeitet ein Timer im 'Normal Mode mit Reload Funktion'?
4. Erläutern Sie die Funktion des Timers im 'Clear-Timer-On-Compare-Match' Mode.
5. Welche Einstellung bestimmt im 'Fast PWM' Mode die Pulsbreite des resultierenden Signals? Erläutern Sie zudem den Unterschied der beiden PWM Modes 'Fast PWM' sowie 'Phase correct PWM'.
6. Wie kann die Frequenz eines PWM-Signals (Fast PWM Mode) berechnet werden? Wie berechnet sich die PWM-Frequenz im 'Phase correct' Mode?
7. Realisieren Sie die in Abschnitt 3.4 vorgestellten Beispielprogramme und verifizieren Sie ihre korrekte Funktionalität.
8. Realisieren Sie die folgenden Programmfunktionen unter Zuhilfenahme der Atmel ATmega328p internen Timer-Peripherie. Sollte eine bestimmte Konfiguration nicht möglich sein, erklären Sie die Gründe hierfür und berechnen Sie die jeweiligen Abweichungen von der vorgegebenen Konfiguration. Gehen Sie in allen Fällen von einem 16MHz Systemtakt aus.
 - Blinken einer an Pin PB5 angeschlossenen LED mit einer Frequenz von 1Hz (Timer T/C0 + Overflow-Interrupt).
 - Blinken einer an Pin PB5 angeschlossenen LED mit einer Frequenz von 1Hz (Timer T/C0 + Overflow-Interrupt + Timer Reload).
 - Blinken einer an Pin PB5 angeschlossenen LED mit einer Frequenz von 1Hz (Timer T/C1 + Overflow-Interrupt).
 - Blinken einer an Pin PB5 angeschlossenen LED mit einer Frequenz von 1Hz (Timer T/C1 + Overflow-Interrupt + Timer Reload).
 - Generierung eines periodischen Timer-Interrupts mit einer Zeitbasis von 30ms (freie Timer- sowie Konfigurationswahl).
 - Generierung eines periodischen Timer-Interrupts mit einer Zeitbasis von 250µs (freie Timer sowie Konfigurationswahl).
 - Generierung eines periodischen Timer-Interrupts mit einer Zeitbasis von 30ms (freie Wahl des Timer + CTC-Interrupt).
 - Generierung eines periodischen Timer-Interrupts mit einer Zeitbasis von 1s (freie Timer sowie Konfigurationswahl).

