HTL Leoben

Rohstoffe • Metalle • Logistik • IT

5IT / 5LIT

NTVS – Netzwerktechnik und verteilte Systeme

Embedded Systems

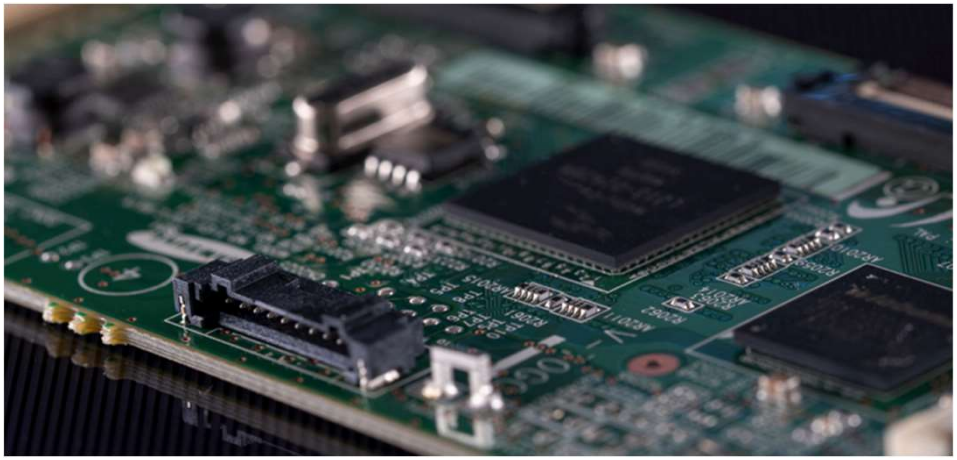
DI(FH) Markus Zacharias

Was ist ein Embedded System

Ein Embedded System oder auch eingebettetes System ist Teil eines viel größeren Systems. Es übernimmt eine oder mehrere bestimmte Funktionen innerhalb dieses übergeordneten Systems.

Beispiele für übergeordnete Systeme:

- Smartphone
- Tablets
- Auto
- Waschmaschine
- Flugzeug



Embedded Systems - Geschichte

Eingebettete Systeme gibt es tatsächlich schon sehr lange. Sie fanden wie so viele technische Errungenschaften ihren Ursprung in der Luftfahrt bzw. der Raumfahrt.

Bereits 1961 entwickelte der amerikanische Ingenieur Charles Stark Draper die erste integrierte Schaltung, die auf dem Apollo Guidance Computer des Apollo Command Module und dem Lunar Module installiert und verwendet wurde. Damit konnten Astronauten Flugdaten in Echtzeit sammeln.

Embedded Systems - Geschichte

Im Jahr 1968 dann ging es von der Luft auf den Boden. Der Volkswagen 1600 verwendete einen Mikroprozessor zur Steuerung seines elektronischen Kraftstoffeinspritzsystems. Er sorgte damit für den Einzug von Embedded Systems ins Fahrzeug.

1971 dann kam es zum ersten im Handel erhältlichen Prozessor durch Intel. 4004, der 4-Bit-Mikroprozessor, wurde für Taschenrechner und andere elektronische Kleingeräte entwickelt.

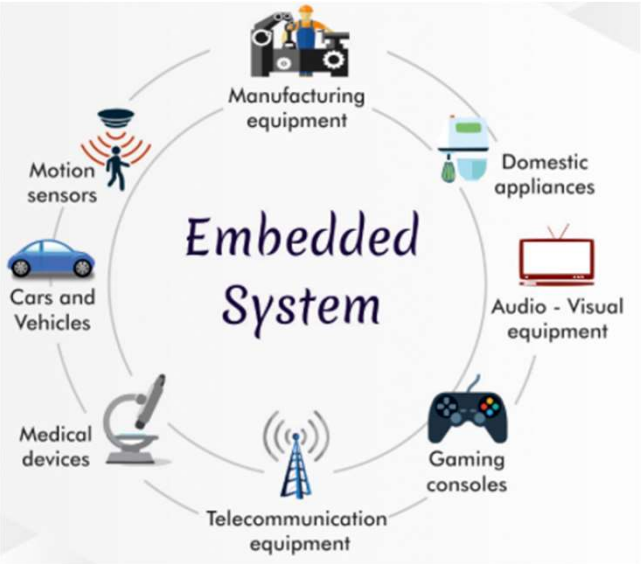
1996 folgte der inzwischen noch bekanntere Hersteller Microsoft. In den 1990ern kamen die ersten eingebetteten Linux-Produkte, die auch heute noch in den meisten Geräten verwendet werden, auf den Markt.

Embedded Systems – Einsatzgebiete

Embedded Systems spielen in vielen Bereichen eine wichtige Rolle.

In der

- Industrie
- Medizintechnik
- Telekommunikation
- Militär
- Landwirtschaft
- usw.



Diese eingebetteten Systems verrichten weitestgehend unbeachtet ihren Dienst.

Embedded Systems - Allgemein

Ein Embedded Systems ist eine Kombination aus Hardware und Software, das für diese speziellen Aufgaben entwickelt wurde.

Die Systeme können programmierbar oder mit einer vorab festgelegten Funktion ausgestattet sein.

Ein eingebettetes System ist ein Computer, jedoch haben einige davon gar keine Benutzeroberfläche. Das betrifft vor allem Systeme die für die Ausführung einer einzigen Aufgabe entwickelt wurden.

Es gibt aber auch Systeme die sehr komplexe grafische Benutzeroberflächen (GUIs) aufweisen. z.B. in mobilen Geräten

Embedded Systems – Beispiel Smartphone

Ein Mobiltelefon ist **kein** Embedded System, sondern eine Kombination von unterschiedlichsten eingebetteten Systemen, die es ihm ermöglichen, eine Vielzahl von Aufgaben zu erfüllen.

Beispiele von Embedded Systems in einem Smartphone?

Embedded Systems – „klassischer PC“

Ein „klassischer“ PC wäre grundsätzlich in der Lage, die Aufgabe eines Embedded Systems zu übernehmen.

Der PC wäre was die Leistungsfähigkeit betrifft „unterfordert“, jedoch „überfordert“ hinsichtlich der der Schnelligkeit.

Ein schlankes Embedded System bringt Echtzeitfähigkeit mit.

Ein auf seine Anwendung angepasstes Embedded System ist nicht nur schneller, es braucht auch deutlich weniger Platz, Energie und kostet nur einen Bruchteil.

Embedded Systems vs. PC

Ein Embedded System hat im Vergleich zu einem PC stark reduzierte Ressourcen.

PC:

- Festplatte
- Betriebssystem
- Tastatur
- Bildschirm

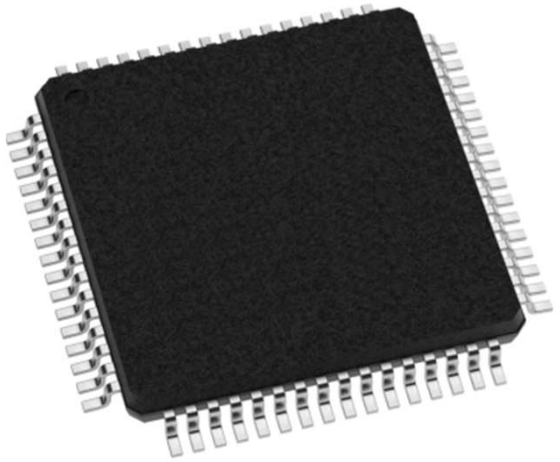
Embedded System:

- ROM oder Flash
- Firmware
- Tastenfeld (opt.)
- LCD (opt.)

Embedded Systems – Hardware

Die Hardware basiert auf einem Microprozessor oder einen Microcontroller. Dieser übernimmt mit Hilfe von Sensoren und Aktoren Überwachungs-, Steuerungs- oder Regelfunktionen oder ist für die Daten- bzw. Signalverarbeitung zuständig.

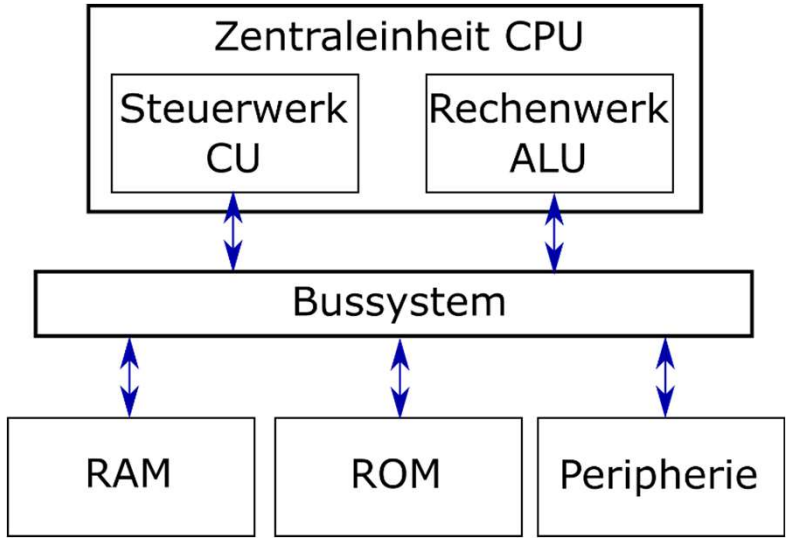
Microprozessor und Microcontroller sind visuell nicht von einander zu unterscheiden.



Embedded Systems – Hardware Übersicht

Der Mikroprozessor enthält jedoch nur eine CPU und erfordert daher zusätzliche Komponenten, wie zum Beispiel Speicherchips.

Zu den Mikrocontrollern gehören nicht nur eine CPU, sondern auch Speicher und Peripheriegeräte wie GPIOs, Flash-Speicher, RAM, usw.



Embedded Systems – Hardware Übersicht

CPU

CPU steht für Central Processing Unit => zentrale Verarbeitungseinheit => unverzichtbar, Herzstück

Sie besteht primär aus Steuer- (CU) und Rechenwerk (ALU)

ALU (Algorithmic Logic Unit)

Arithmetisch_Logsiche Einheit ist der Teil der CPU , der arithmetische-logische Operationen mit binären Daten ausführt.

Sie kann addieren, subtrahieren, multiplizieren, …; sie führt logische und, oder, … Verknüpfungen aus und ist zu Schiebeoperationen in der Lage.

Embedded Systems – Hardware Übersicht

CU (Control Unit)

Die Kontrolleinheit übt die Basisfunktion in der CPU aus. Unter Benutzung des internen Oszillators steuert sie die Ereignisse. Nachdem ein Befehl geholt oder decodiert wurde, gibt die Kontrolleinheit geeignete Befehle aus, um die richtigen Aktionen zu starten. Diese Befehle werden über den Bus aus dem Arbeitsspeicher geholt.

Bussystem

Über dieses ist die CPU mit den anderen Komponenten verbunden.

z.B. Datenbus, Adressbus und Steuerbus

Je nach Prozessor werden unterschiedlich viele Bit gleichzeitig übertragen. 8, 16, 32 oder 64 Bit

Embedded Systems – Hardware Übersicht

RAM (Random Access Memory)

Im Arbeitsspeicher o. Hauptspeicher werden die gerade auszuführenden Programmbefehle und benötigen Daten gemeinsam abgelegt. Leistungsfähigkeit und Größe beeinflussen maßgeblich die Leistungsfähigkeit des gesamten Systems.

ROM (Read Only Memory)

Die CPU holt sich den Befehlscode aus Programmspeicher. Datenspeicher auf den nur lesend zugegriffen werden kann und nicht flüchtig ist.

Vermehrt werden jedoch stattdessen Flash-Speicher eingebaut, dessen Inhalt nachträglich geändert werden kann.

Embedded Systems – Hardware Übersicht

Register

Register sind temporäre Speicher, davon haben einige einen festen Verwendungszweck wie Befehlsregister, Statusregister, Programmzähler, andere sind mehr für allgemeine Zwecke gedacht.

Peripherie

Peripheriemodule können sehr unterschiedlich sein. Z.B. Digitale Ein- und Ausgänge, Timer, Buscontroller, ADC usw.

Embedded Systems – Firmware

Firmware

Die Firmware, ist eine Software, die fest in einem elektronischen Gerät implementiert ist. Meisten wird die Firmware auf einem Flash-Speicher, EPROM, EEPROM, gespeichert und ist vom Benutzer nicht ohne Weiteres auswechselbar.

Die auf diese Weise gespeicherte Software ist dauerhaft und unveränderlich, daher der Name Firmware (firm => securely or solid fixed in place).

Die Firmware ist im Gegensatz zur Software funktional fest mit der Hardware verbunden. Ohne die Firmware ist die Hardware nicht nutzbar. Sie nimmt eine Art Zwischenstellung zwischen Hardware und Anwendungssoftware ein.

Embedded Systems – Embedded Software

Wie die Firmware, wird auch die Embedded Software für ein bestimmtes Device erstellt. Sie läuft typischer Weise auf einem Mikrocontroller oder Microprozessor.

Im Gegensatz zur Firmware ist die eingebettete Software eher wie die Anwendungssoftware, die auf einem PC ausgeführt wird.

Eingebettete Software implementiert typischerweise Merkmale und Funktionen auf höherer Ebene des Geräts.

Die Firmware übernimmt hingegen Low-Level Aufgaben, wie z.B. die Umwandlung analoger Sensor Signale in digitale Daten.

Embedded System – Recheneinheit

Recheneinheit

In den meisten eingebetteten Systemen werden heutzutage softwarebasierte Recheneinheiten wie General-Purpose-Prozessoren oder Mikrocontroller verwendet, die für eine Vielzahl verschiedener Anwendungen genutzt werden können und die wir schon in einem früheren Zeitpunkt kurz besprochen haben.

In einigen Anwendungsfällen werden jedoch bestimmte Algorithmen oder Eigenschaften benötigt, die die Entwickler dazu veranlassen die Recheneinheit entsprechend anzupassen bzw. für diese Anwendungen zu optimieren.

Zur Auswahl stehen digitale Signalprozessoren (DSPs), Application-specific Integrated Circuits (ASICs) aber auch hardware- programmierbare Recheneinheiten wie z.B. Field-Programmable Gate-Arrays (FPGAs) und uvm.

Embedded System – Recheneinheit

General-Purpose-Prozessoren

- Die Rechengeschwindigkeit ist wesentlich höher als die von Mikrocontrollern.
- Caches mit mehrschichtigem, schnellen Speicherzugriff.
- Die hohen Rechengeschwindigkeiten sind vor allem beim Einsatz in Echtzeitanwendungen von Vorteil.
- Das gesamte Silizium wird für allgemeine Anwendungen verwendet. Keine Timer oder größere Speicher
- Kein Augenmerk auf die Integration von Peripherie sondern auf Leistung.

Embedded System – Recheneinheit

Mikrocontroller

- Prozessor und wichtige Peripherieelemente wie z.B. Bus-Treiber, PWM-Units, A/D u. D/A-Wandler, Programm- und Datenspeicher, Schnittstellen etc. sind in einem Chip vereinen.
- Der Vorteil ist, dass diese Peripherieelemente optimal an den Prozessor angebunden sind.
- Durch die große Palette an erhältlichen Mikrocontrollern kann die Hardware bereits durch die Auswahl stark optimiert werden.
- Mikrocontroller benötigen nicht einmal einen *externen Taktgeber,* sondern Können für einfache Aufgaben intern einen Takt erzeugen.
- Um diese Vielfalt etwas überschaubarer zu machen, werden die Mikrocontroller meist in Familien geordnet, die bestimmte Subtypen haben.

Embedded System – Recheneinheit

Digitale Signalprozessoren (DSPs)

- DSPs sind im Grunde genommen Mikrocontroller mit spezieller Peripherie und einem erweiterten Befehlssatz.
- Optimiert für Aufgaben in der Signalverarbeitung, wie digitale Filter oder andere Signalmanipulationen. Sind häufig im Audio- und Videobereich im Einsatz, dienen aber auch als Beschleuniger für die Kommunikation in verteilten Systemen.
- Enthalten meist eingebaute D/A- und A/D-Wandler, die direkt an den Prozessor angebunden sind.
- Mehrerer Rechenwerke, unter anderem eines Multiply-Accumulate-Rechenwerkes (MAC). Dieses ermöglicht die zeitgleiche Ausführung von Addition und Multiplikation, wie sie für Faltungen und Fast-Fourier-Transformationen (FFTs) benötigt werden. Das spielt vor allem bei der Signalübertragung und -kodierung eine wichtige Rolle.

Embedded System – Recheneinheit

Application Specific Integrated Circuits (ASICs)

- Sind speziell für eine Aufgabe gefertigte Chips
- Optimiert bzgl. den aufgabenspezifischen Anforderungen z.B. Geschwindigkeit, Energieeffizienz, Baugröße und/oder Zuverlässigkeit,
- In der Regel sehr unflexibel
- Entwicklung und Fertigung in kleinen Stückzahlen sehr teuer und zeitaufwändig. Lohnt sich nur bei Massenfertigung.
- ASICs oft als frei verkäufliche Standardteile bei entsprechenden Händlern. Man spricht dann von *Application-Specific Standard Products (ASSP)*.

Embedded System – Recheneinheit

Field-Programmable Gate-Arrays (FPGAs)

- Spezielle ICs, die noch kein konkretes Verhalten eingeprägt haben. Das bedeutet, dass man die Hardware selbst „programmieren“ kann.
- Im Gegensatz zu einem Mikrocontroller gibt man nicht per Software einer festgelegten Hardware ihr Verhalten vor, sondern verändert die Hardware selbst.
- FPGAs können im Gegensatz zu ASICs mehrfach rekonfiguriert werden, was insbesondere bei der Entwicklung ein nicht zu unterschätzender Vorteil ist.
- FPGAs werden oft zur Entwicklung von ASICs eingesetzt, da auf ihnen bereits das zeitliche Zusammenspiel der Komponenten des ASICs simuliert werden kann.
- Im Vergleich zum Mikrocontroller sind FPGAs allerdings recht teuer und die Performance ist etwas schlechter bei gleicher Technologie.

Embedded Systems – Übersicht ROM Arten

Arten von ROMs

- ROM:

  Beim ROM werden Daten direkt verewigt.

- PROM:

  Beim „Programmable“ Read-Only Memory sind auf dem Chip quasi Sicherungen eingebaut. Beim Beschreiben des PROM werden selektiv Sicherungen durchgebrannt. Das bedeutet dieser Speichertyp ist nur einmal beschreibbar.

- EPROM:

  „Erasable Programmable ROM“ => Dieser Speicher kann nur mit UV-Licht gelöscht werden.

Embedded Systems – Übersicht ROM Arten

- OTP-EPROM:

  Diese „One Time Programmable“ – EPROM Variante kann deshalb nicht mit UV_Licht gelöscht werden, da sie sicher vor jedem Licht in ein Package eingeschweißt ist.

- Flash:

  Flash oder Flash-EEPROM ist ein Speicher, der elektronisch gelöscht werden kann. Es gibt auch FLASH Varianten, die in mehrere Bereiche unterteilt sind und bei denen man selektiv nur einen Bereich löschen kann. Löschen und Schreiben geht relativ langsam, das auslesen geht jedoch sehr schnell.

Embedded Systems – Übersicht ROM Arten

- EEPROM:

  Das Elektrische – EPROM kommt mit wenigen Pins aus, wird deshalb auch seriell beschrieben. Sie sind meist sehr kleine Speicher und werden verwendet zur Speicherung kleiner Datenmengen in elektrischen Geräten, bei denen die Information auch ohne anliegende Versorgungsspannung erhalten bleiben muss. Z.B. für Seriennummer, MAC-Adresse, …

Embedded System – Übersicht RAM

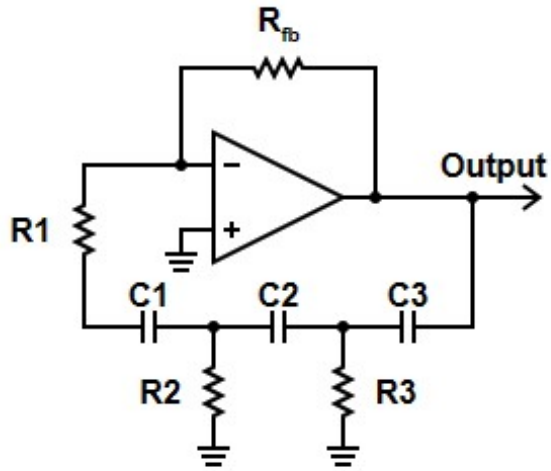
Das Kapitel RAM wird selbst erarbeitet.

Embedded System – Takterzeugung

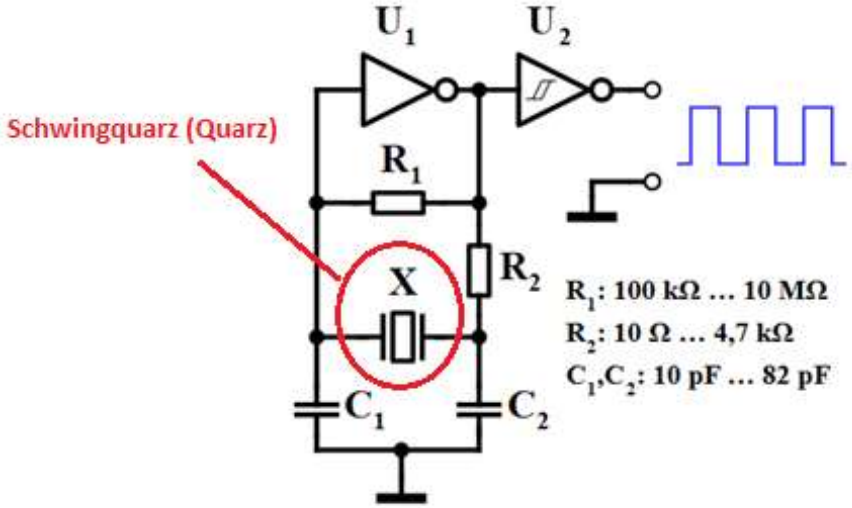
Takterzeugung

Zur Kontrolle aller Abläufe zw. CPU und Peripherie ist eine Taktquelle notwendig. Im einfachsten Fall ist dies ein **RC-Oszillator**.

Für exakte Zeitabläufe oder Zeitmessungen werden **Quarzoszillatoren** eingesetzt.



RC Phase-Shift Oszillator (Quelle: Wikipedia)



Quarzoszillator (Quelle: Wikipedia)

Embedded System – Takterzeugung

Mit den Anforderungen Strom zu sparen (Strom vs. Frequenz), die Peripherie mit unterschiedlichen Taktfrequenzen zu versorgen und Störungen möglichst kleine zu halten kann die Takterzeugung und Verteilung sehr schnell eine komplizierte Angelegenheit werden.

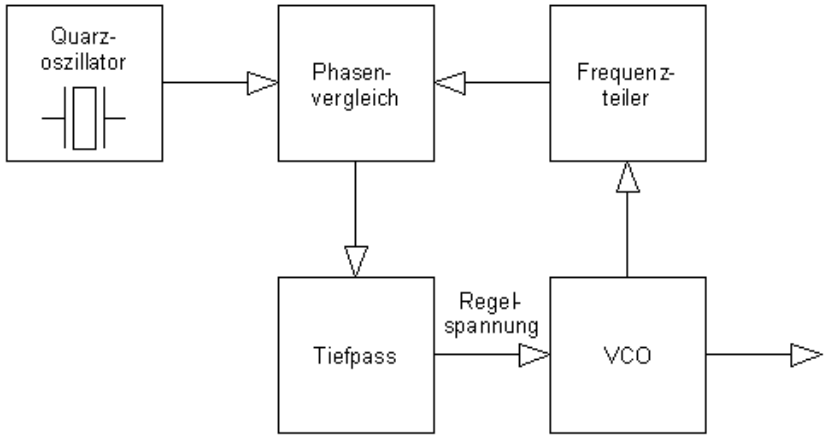
Daher ist bei der Konfiguration des RCC-Moduls (reset and clock control, RCC) Vorsicht geboten, da das sehr schnell zu einem Systemabsturz führen kann.

Nach einem POR (Power on reset) benötigt die CPU einen Takt noch bevor überhaupt das RCC konfiguriert wurde.

Daher startet das Taktmodule mit einem internen Takt, damit die CPU arbeiten kann. Jedoch ist hier die Toleranz relativ groß.

Embedded System – Takterzeugung

Für höhere Genauigkeiten wir eine externe Taktquelle oder Quarzkristall verwendet. Meist verwendet man einen genauen niederfrequenten externen Takt und eine interne PLL-Schaltung (phase-locked loop, PLL) um einen genauen höher frequenten Takt zu erzeugen.



Prinzipschaltung eines PLL-Regelkreises (Quelle: Wikipedia)

Nach dem hochstarten und konfigurieren kann auf diesen genaueren Takt umgeschaltet werden.

Embedded System – Busankopplung

Bussystem

Kommunikationsendpunkte, die Daten austauschen, müssen miteinander verknüpft sein. Die Summer aller dafür notwendigen Leitungen wird als „BUS“ bezeichnet. Die Anzahl der gleichzeitig verfügbaren Busleitungen ist die Busbreite. Konform mit der Datenbusbreite der CPU.

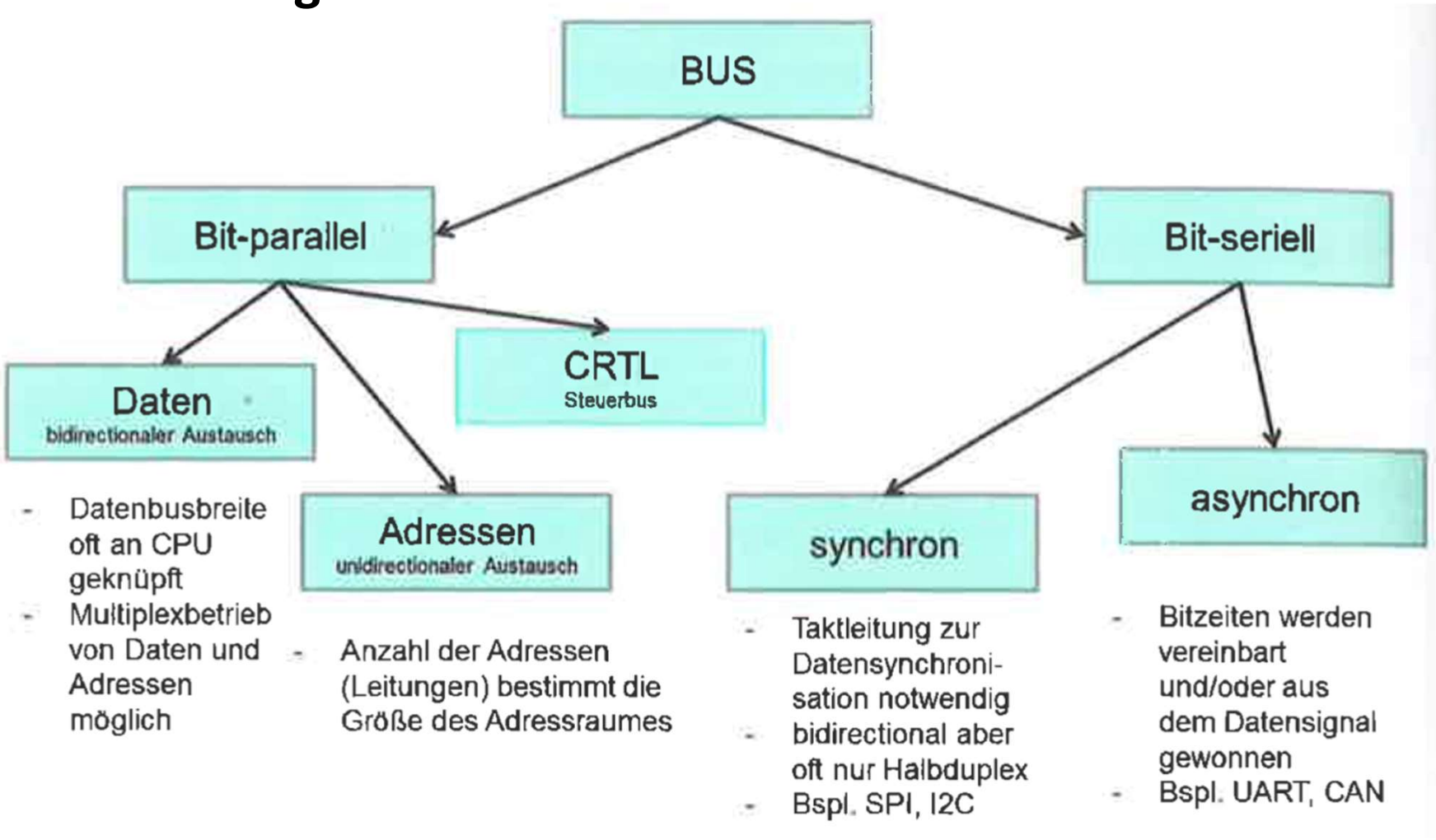
Sind alle Informationen gleichzeitig am Bus verfügbar, Spricht man von einem Parallelbus.

Ein zweiseitiger Datenaustausch wird als bidirektionaler Bus bezeichnet.

Von einem unidirektionalen Bus spricht man wenn Daten nur in eine Richtung ausgetauscht werden.

Embedded System – Busankopplung

Unterscheidung

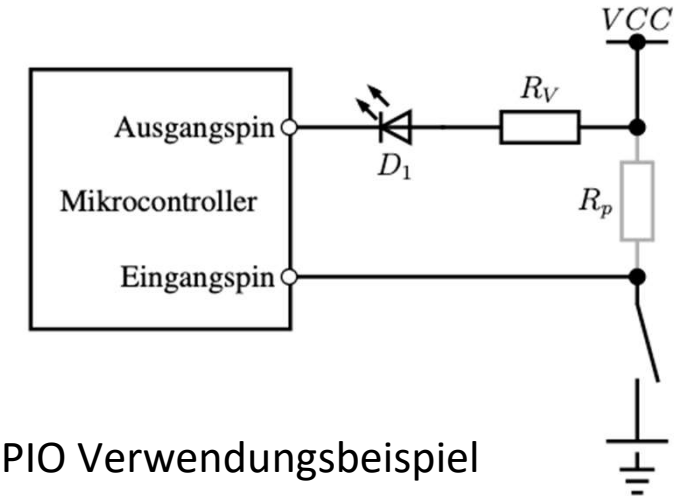


Unterschiedliche Bussysteme

Embedded System – GPIO

Unabhängig von der Wahl der tatsächlichen Recheneinheit muss diese am Ende auch in die Umgebung eingebunden werden.

Wie im Bild unterhalb zu sehen ist erfolgt das durch frei nutzbare Pins (*General-Purpose-I/O-Pins (GPIOs)*), die in diesem Beispiel eines Mikrocontrollers als Eingangs- und Ausgangspins konfiguriert sind.



GPIO Verwendungsbeispiel

Embedded System – GPIO

Die GPIO (General Purpose Input Output) Pins sind ein wichtiger Teil der Peripherie eines Microcontrollers. Sie können je nach Konfiguration sowohl als digitaler Eingang als auch als digitaler Ausgang verwendet werden. Manche GPIO Pins können auch für andere Aufgaben verwendet werden (z.B. ADC, Counter, ...).

Wird ein GPIO Pin als Eingang konfiguriert kann er verwendet zum

- Einlesen eines Tasters
- Einlesen eines Sensors, usw.

Wird ein GPIO Pin als Ausgang konfiguriert kann er verwendet zum

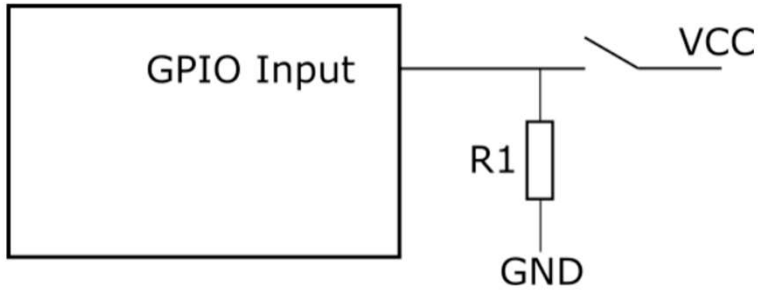
- Ansteuern einer LED
- Ansteuern eines Motors, usw.

Embedded System – GPIO

Ein Eingangspin kann auf zwei unterschiedliche Arten beschaltet werden.

Pull-Down

In diesem Fall wird der Ausgang über eine Pull Down Widerstand mit GND verbunden. Damit ist sicher gestellt, dass sich bei nicht betätigtem Taster ein Potential von GND am Eingang befindet. Durch betätigen des Tasters wird der Eingang auf eine Spannung von VCC gezogen. Damit der Strom, der bei betätigtem Taster fließt, nicht zu groß wird ist der Pull Down Widerstand im hochohmigen Bereich (ca. 10kΩ).

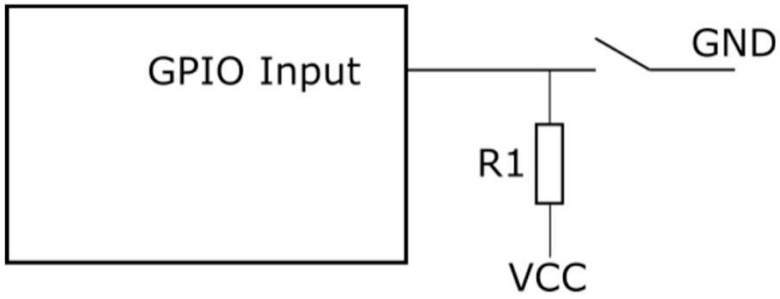


Prinzipschaltung Pull-Down

Embedded System – GPIO

Pull-UP

In diesem Fall wird der Ausgang über eine Pull Up Widerstand mit VCC verbunden. Damit ist sicher gestellt, dass sich bei nicht betätigtem Taster ein Potential von VCC am Eingang befindet. Durch betätigen des Tasters wird der Eingang auf eine Spannung von 0V gezogen. Damit der Strom, der bei betätigtem Taster fließt, nicht zu groß wird ist der Pull Up Widerstand im hochohmigen Bereich (ca. 10kΩ). Die Logik beim der Pull Up Beschaltung ist invertiert im Vergleich zur Pull Down Beschaltung.

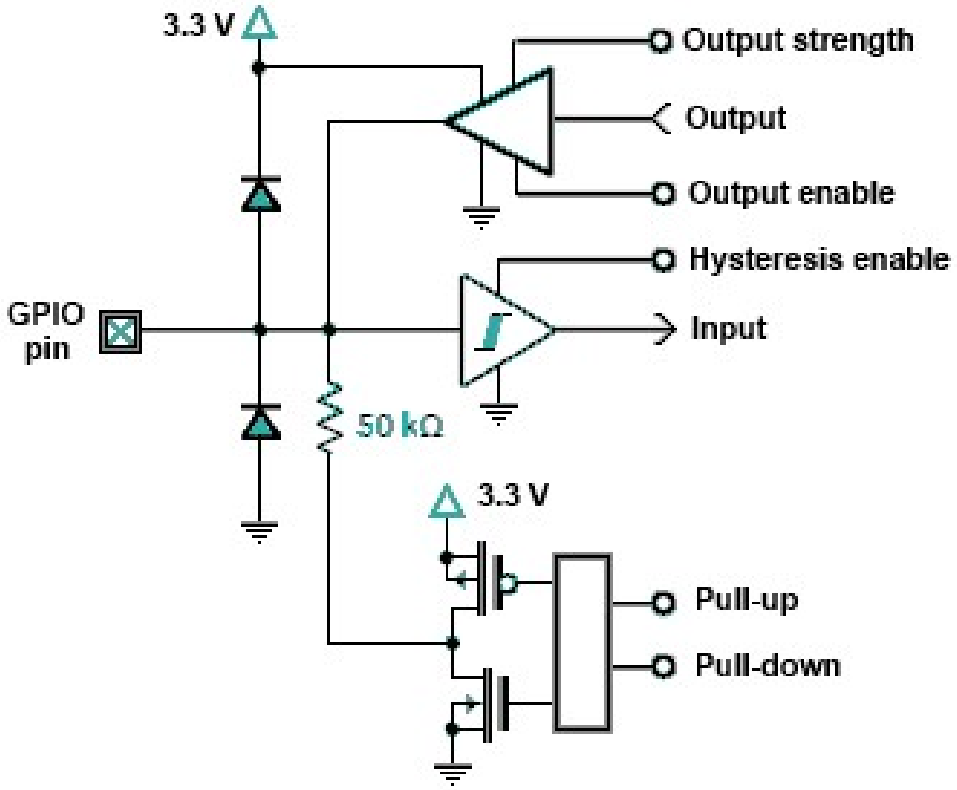


Prinzipschaltung Pull-UP

Embedded System – GPIO

GPIO Pin – Beispiel 1

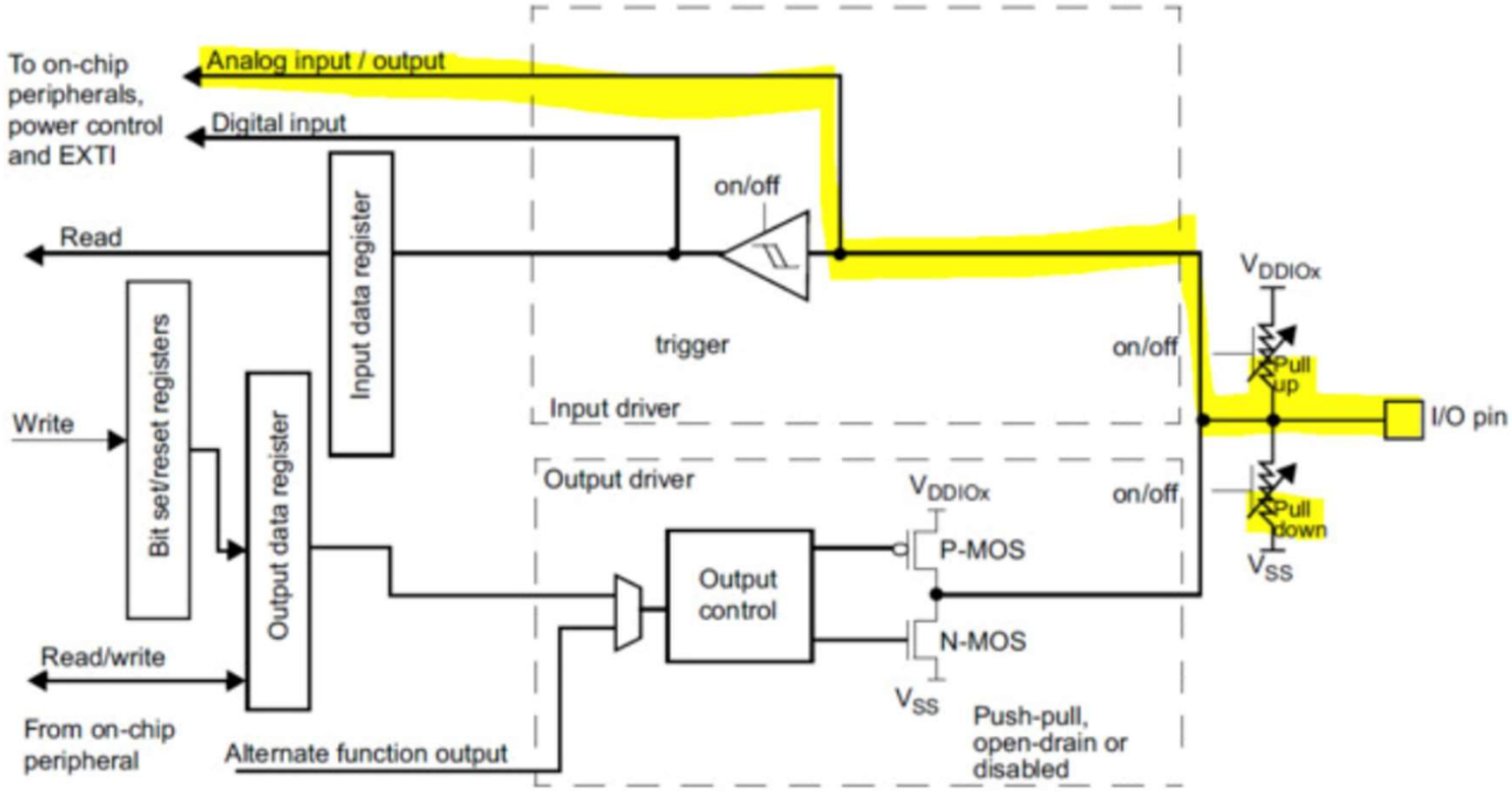
- Ausgangsstufe mit unterschiedlichen Treiberstärken
- Eingangsstufe mit einschaltbarer Hysterese
- Interne Pull-UP/DOWNs
- Schutzstruktur (Diode)



Beispiel eines GPIO Pins

Embedded System – GPIO

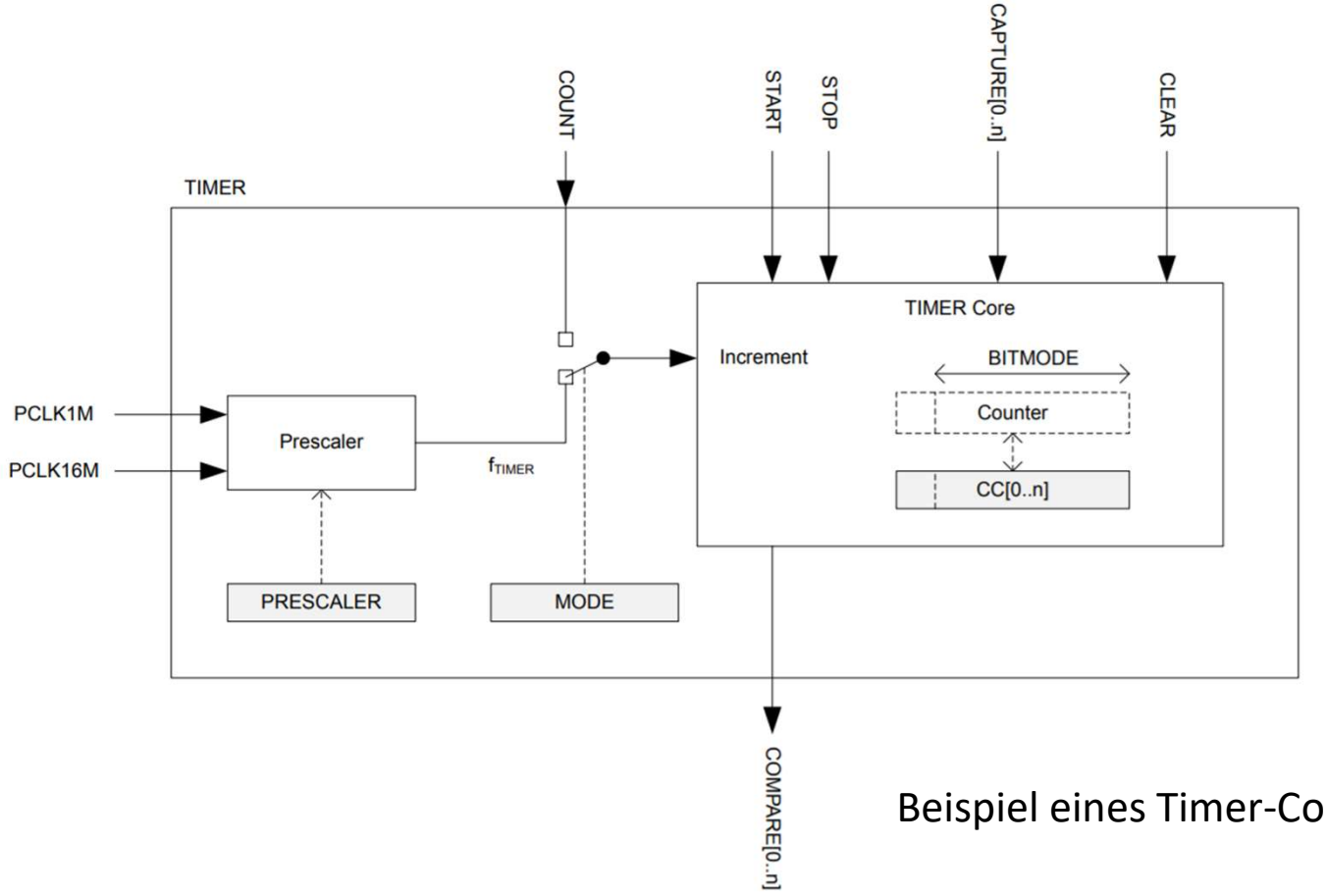
GPIO Pin – Beispiel 2



GPIO mit eine zusätzliche Pfad für analoge Eingangs- oder Ausgangsspannungen.

Embedded Systems – Timer Modul

In einem Embedded System stellt das Timer-Counter Modul einen wichtigen Teil dar.



Beispiel eines Timer-Counter Moduls

Embedded Systems – Timer

TIMER Core

Das Timer Module besteht im Kern aus dem Timer-Core der technisch gesehen eigentlich ein Zähler ist.

Er hat folgende Ein- und Ausgänge bzw. Einstellmöglich:

- **Start** aktiviert den Zähler
- **Stop** deaktiviert den Zähler
- **Increment** Eine steigende Flanke auf diesem Eingang erhöht den Zählwert um Eins.
- **Clear** setzt den Zählwert auf Null zurück.
- **Bitmode** legt den maximalen Zählwert fest. 8 Bit bedeutet, dass der maximale Zählwert 2^8-1 = 255 ist. Wird der Wert überschritten fängt der Zähler wieder bei Null an.
- **Capture[0...n]** Hier können mehrere Werte hinterlegt werden bei deren Erreichen der Timer einen Interrupt auslöst. Siehe Kapitel Timer Interrupts.

Anmerkung: CC[0..n] => Capture/Compare Register 0 bis n

TIMER Module

Da das Timer Module sowohl als Timer als auch als Counter verwendet werden kann, gibt es außerhalb des Timer Cores noch weitere Elemente und Parameter:

- **Mode** legt fest ob der Timer als Timer oder Counter verwendet wird. Technisch wird dabei der Increment Eingang auf Count verbunden (Counter Mode) oder der Incrment Eingang wird auf die Clock (Prozessortakt) verbunden (Timer Mode).
- **Prescaler** Da der Prozessortakt für Anwendungen oft zu schnell ist kann er für den Timer im sogenannten Prescaler skaliert werden. Dabei wird der Takt durch 2 hoch den Prescaler Wert geteilt. Bei einer Prozessorfrequenz von 16Mhz bedeuetet ein Prescaler 3 eine Timerfrequenz von 16 / 2^3 = 2Mhz.

Embedded Systems – Timer Interrupt

Ein Timer (und auch andere Module) können in einem Microcontroller Interrupts auslösen. Ein Timer kann einen Interrupt bei einem bestimmten Wert auslösen. Dabei spielen sich im Hauptprogramm folgende Schritte ab:

- Das Hauptprogramm wird gestoppt.
- Der nächste Befehl des Hauptprogramms wird gespeichert.
- Ein eigener Programmteil, die Interrupt-Service-Routine (ISR), wird gestartet.
- Die ISR wird abgearbeitet.
- Das Hauptprogramm wird beim gespeicherten Befehl fortgesetzt.

Embedded Systems – Timer Interrupt

Beispiel Timer / Timer-Interrupt

Embedded System – Interrupt Handling

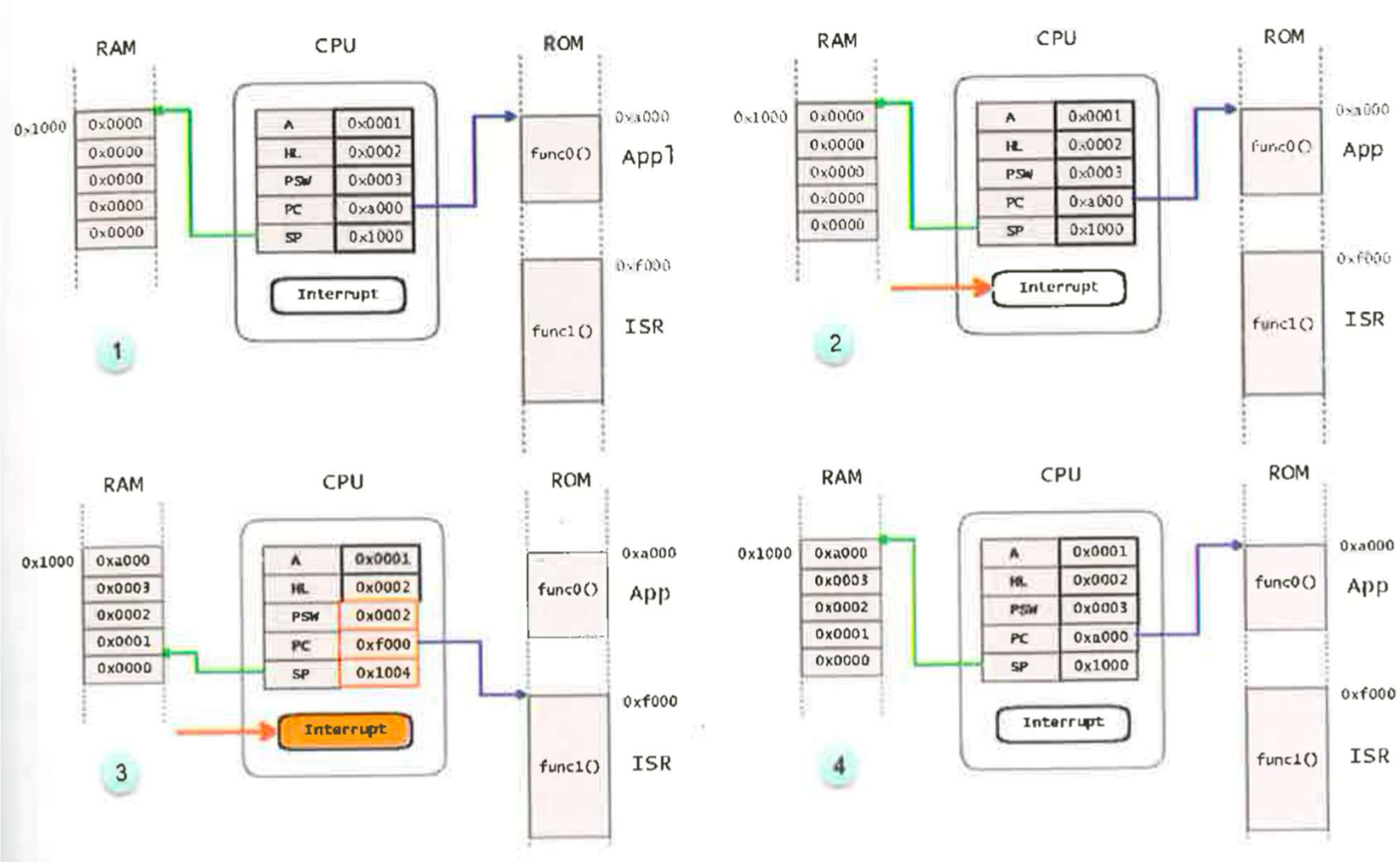
Interrupt-Handling

- Interrupt-Handling wesentliches Leistungsmerkmal einer MCU.
- Interrupt ist eine Programmunterbrechung der aktuellen Applikation.
- Interrupt Service Routine, ist das Einschieben und Ausführen eines anderen Programmabschnittes.

Die folgende Abbildung zeigt die Annahme und Ausführung einer Interrupt Service Routine am Beispiel einer einfachen CPU.

Embedded System – Interrupt Handling

Interrupt Service Routine



Embedded System – Interrupt Handling

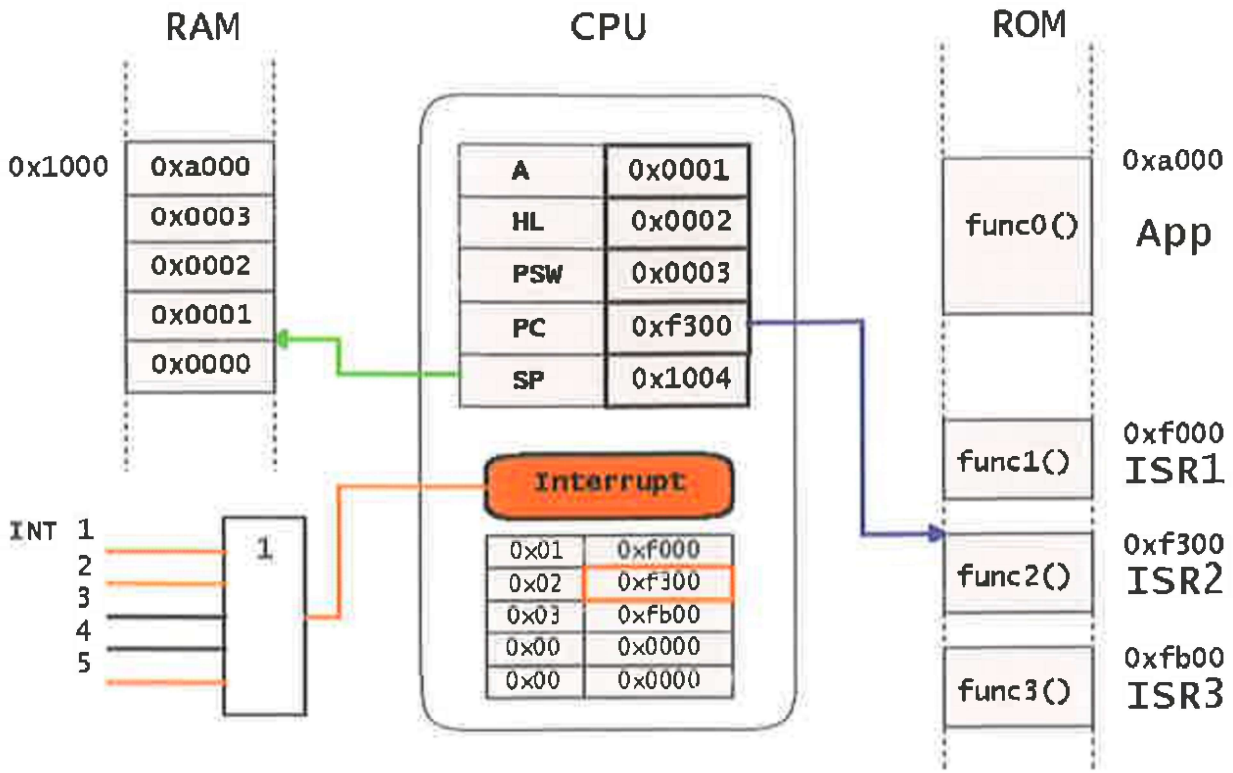
Interrupt Vektoren und Interrupt Vektor Tabelle

- Die meisten MCUs verfügen über mehrere Interrupt-Quellen
- Für alle vorhandenen Interrupt-Events wird eine Liste mit den Adressen der ISRs angelegt
- Liste wird als Interrupt-Vektor-Tabelle bezeichnet
- Tabelleneinträgen können mit Prioritäten versehen werden.

Die folgende Abbildung zeigt den Interrupot Controller mit priorisierbare Interrupt-Vektoren.

Embedded System – Interrupt Handling

Interrupt Vektoren



Embedded System – Pulsweitenmodulation

Ein PWM-Ausgangssignal ist ein Rechtecksignal bei dem in konstanten Zeitabständen ein- und ausgeschalten wird.

In der Microcontrollertechnik kann es sehr gut als Ersatz für analoge Ausgänge genutzt werden.

Es kann damit z.B.

- Die Helligkeit einer LED gesteuert werden.
- Die Drehzahl eines Motors gesteuert werden.
- Die Leistung eines Heizelementes gesteuert werden.
- …

Embedded System – Pulsweitenmodulation

Periodendauer T

Die Zeitdauer von einmal ausschalten über einschalten bis zum nächsten ausschalten. Sie kann aus der Frequenz berechnet werden.

$$T = t_{Ein} + t_{Aus} = \frac{1}{f}$$

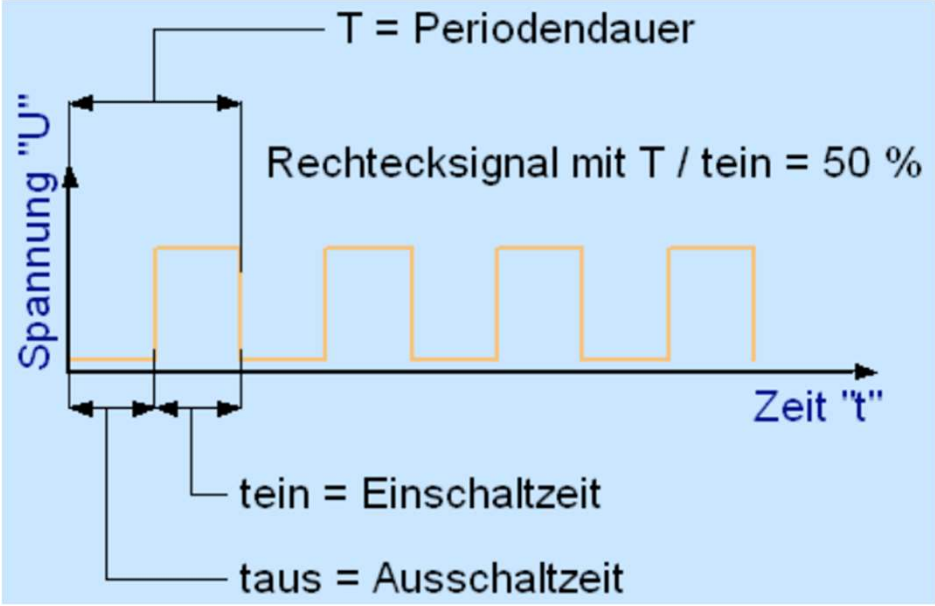
Bei einer Frequenz von 1kHz ergibt das eine Periodendauer von 1ms.

Einschaltdauer bzw. Pulsbreite t_{Ein}

Zeitdauer die das Signal auf High ist.

Ausschaltdauer t_{Aus}

Zeitdauer die das Signal auf Low ist.



Embedded System – Pulsweitenmodulation

Duty Cycle

Der Duty Cycle ist die wichtigste Größe bei einer PWM. Damit kann der analoge Ersatzwert in Prozent gesteuert werden. Er wird wie folgt berechnet:

$$\text{Duty Cycle} = \frac{t_{Ein}}{T} * 100\% = \frac{t_{Ein}}{t_{Ein} + t_{Aus}} * 100\%$$

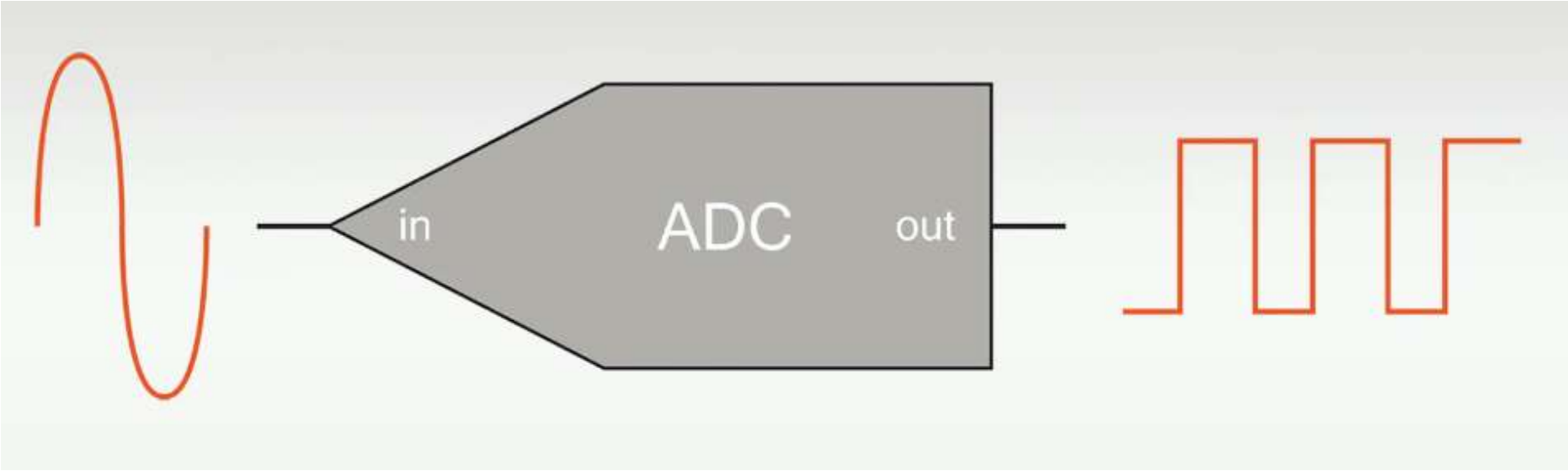
Bei einer Periodendauer von 1ms und einer Einschaltdauer von 200us ergibt sich somit ein Duty Cycle von 20%.

Embedded System – Pulsweitenmodulation

Beispiel Pulsweitenmodulation (PWM)

Embedded Systems – A/D Wandler

Der A/D-Wandler überführt die analogen Eingangsgrößen in digitale Ausgangsgrößen, die dann weiter verarbeitet werden können.



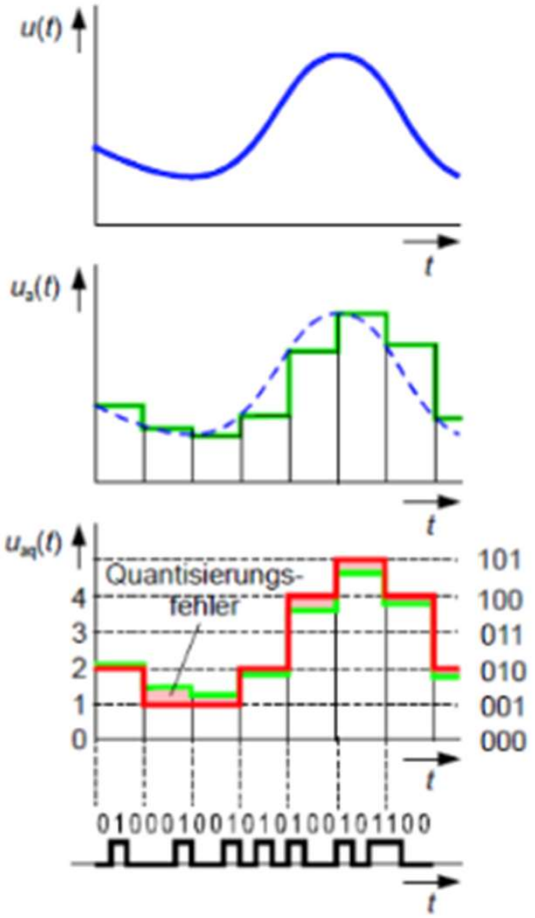
Embedded Systems – A/D Wandler

Analog zu digital:

Durch Abtastung des zeitkontinuierlichen Signals einer analogen Quelle ergibt sich ein zeitdiskretes Signal.

Durch Quantisierung des zeitdiskreten und wertkontinuierlichen Signals ergibt sich ein digitales Signal.

Die Umwandlung eines analogen in ein digitales Signal erfolgt mit Hilfe eines Analog/Digital Wandlers (ADC – Analog to Digital Converter).



Embedded Systems – A/D Wandler

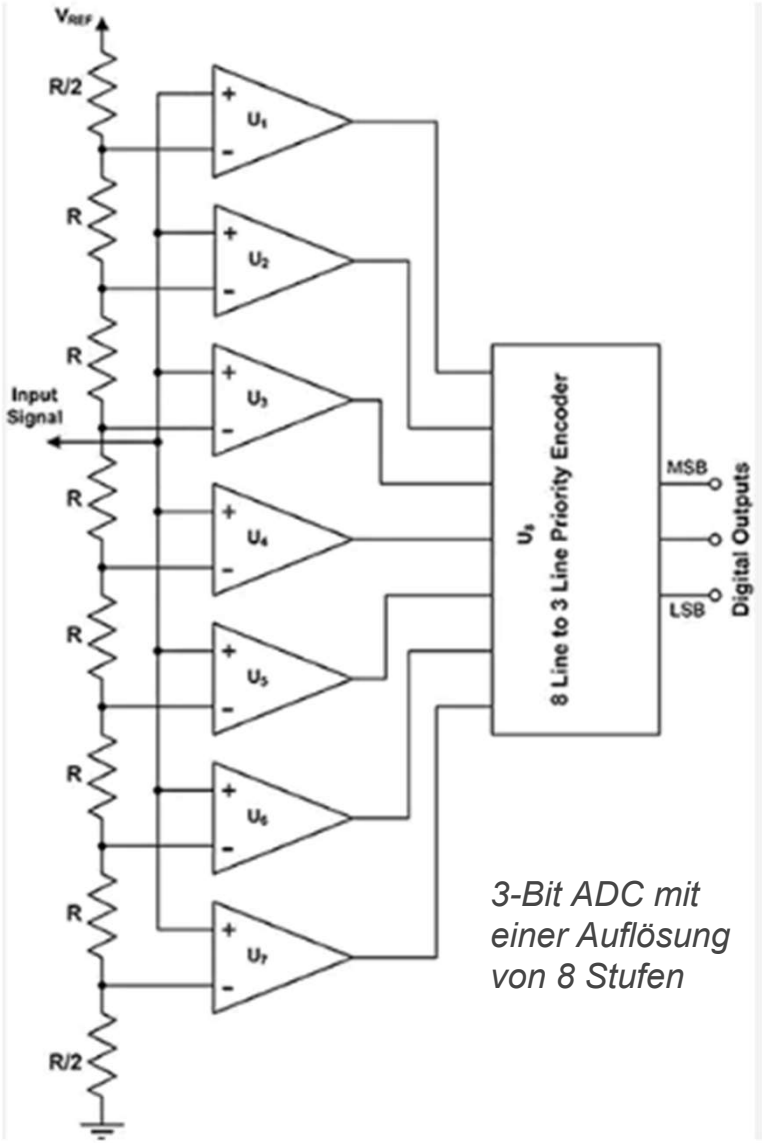
Der A/D-Wandler überführt die analoge Eingangsgröße in eine digitale Ausgangsgröße, die dann weiter verarbeitet werden können.

Analog zu digital Wandlung:

Der anliegende Spannungswert wird als Bruchteil einer zur Verfügung gestellten Referenzspannung (V_{REF}) dargestellt.

Wichtige Kenngrößen:

- Abtastrate
- Auflösung
- Genauigkeit
- Eingangsspannungsbereich
- Linearität



3-Bit ADC mit einer Auflösung von 8 Stufen

Analog/Digital Wandler

Zu den Kenngrößen und der Funktion des ADC:

Die erste Operation eines ADC ist das Abtasten des Analogsignals. Die dazugehörige Kenngröße eines ADC ist die Abtastrate, die die maximale Signalfrequenz angibt, die vom ADC gewandelt werden kann.

Der ADC muss das abgetastete Signal quantisieren. Diese Kenngröße wird in der Regel als die Anzahl der Auflösungsbits bzw. Auflösung bezeichnet. Wird ein Signal zum Beispiel in 8 Bits aufgeteilt, bedeutet dies, dass es 2^8 oder 256 diskrete Pegel (Quantisierungsstufen) gibt.

Sowohl die Auflösung als auch die Abtastrate wird von der ADC-Hardware bestimmt. Je höher die Auflösung des ADC, desto begrenzter ist in der Regel die maximale Abtastrate.

Die Genauigkeit eines ADC ist von der Auflösung und der Abtastrate abhängig. Die Auflösung beeinflusst die Amplitudengenauigkeit. Die Abtastrate bestimmt die Zeitgenauigkeit.

Analog/Digital Wandler

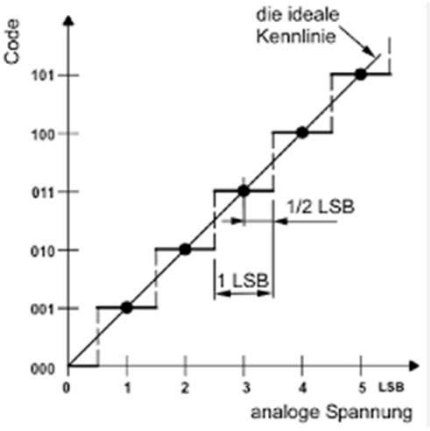
Beispiel A/D Wandlung:

ADC: Auflösung = 8 Bit, Referenzspannung U_{ref} = 5V; Eingangsspannung U_e = 3,3V

Zahl: $Z = \frac{U_e}{U_{LSB}} = \frac{U_e}{U_{ref}/2^8} = \frac{3.3}{5/256}$ = 169_{dez} => 10101001_{dual}

LSB: $U_{LSB} = \frac{U_{ref}}{2^{Auflösung}}$ U_{LSB} … ist die Spannungseinheit für das niedrigste Bit.

Dies entspricht umgekehrt: $\frac{U_{ref}}{2^8} * Z = \frac{5}{256} * 169 = 3{,}30078\ \text{V}$



Hiermit wird sichtbar, dass

- die Auflösung (n = 8 Bit => Auflösung von 2^n = 256 Schritten),
- als auch die Höhe der Referenzspannung die Genauigkeit des Wandlers beeinflussen.

Bei der Quantisierung einer Eingangsspannung in einen Zahlenwert, hat ein Bereich von Eingangsspannungen den selben Zahlenwert.
=> Quantisierungsfehler +/- 0,5 LSB.

Embedded Systems – A/D Wandlerverfahren

Abhängig von den Anforderungen und den Wandler-Eigenschaften werden unterschiedliche A/D-Wandler in der Praxis eingesetzt.

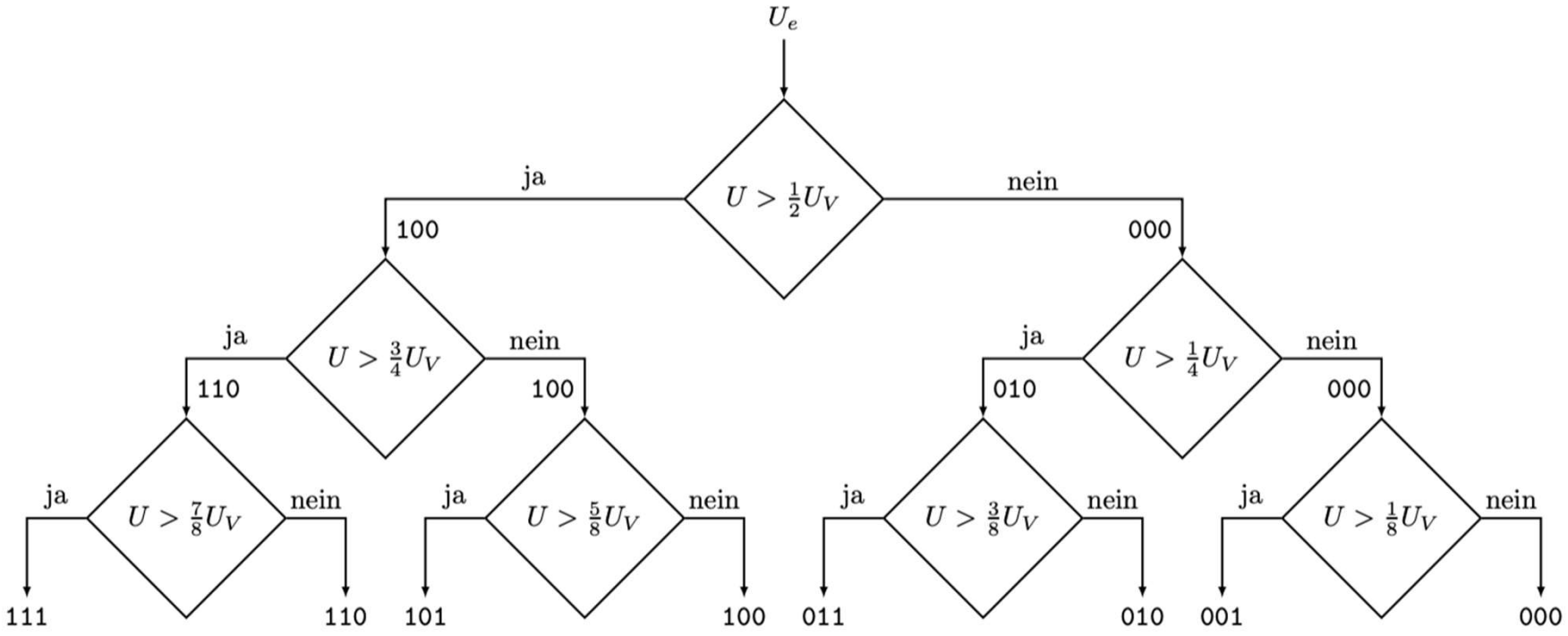
Es wird zwischen drei prinzipiell verschiedenen Verfahren unterschieden

- Parallelverfahren
- Wägeverfahren
- Zählverfahren

Embedded Systems – A/D Wandler

Wägeverfahren

Das Wägeverfahren (*Successive Approximation Register (SAR) ADC*) gehört zur Klasse der direkten Wandler. Er vergleicht die Eingangsgrößen schrittweise mit bestimmten Referenzgrößen.

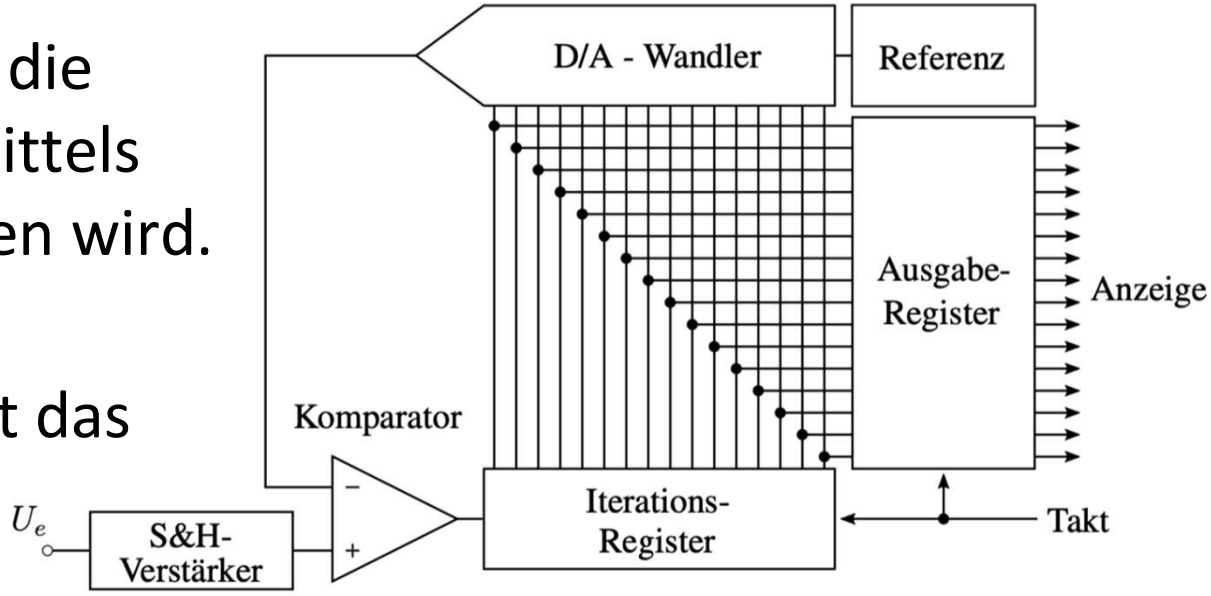


Embedded Systems – A/D Wandler

Aufbau A/D Wandler mit Wägeverfahren

Der D/A-Wandlers erzeugt die Vergleichsspannung, die mittels eines Komparator verglichen wird.

Das Vergleichsergebnis im Iterationsregister bestimmt das entsprechende Bit im Ausgaberegister.



Die schrittweise Bestimmung des digitalen Wortes ist zeitabhängig von der gewünschten Auflösung. D.h. je höher die Auflösung, desto länger dauert die Umsetzung.

Parallele Wandler

Um die Wandlungszeit deutlich zu verkürzen, werden A/D-Wandler nach dem Parallelverfahren eingesetzt.

Diese sind beispielsweise für Videoanwendungen notwendig.

Während der SAR-ADC mehrere Vergleiche ausführt, kommt der Parallel-Wandler mit nur einem Vergleich aus.

Möchte man jedoch eine n-Bit Auflösung eines digitalen Wortes erzeugen, so sind 2^n-1 analoge Komparatoren notwendig.

Beispielsweise ein 8-Bit Parallel Wandler benötigt somit $2^8-1=255$ Komparatoren.

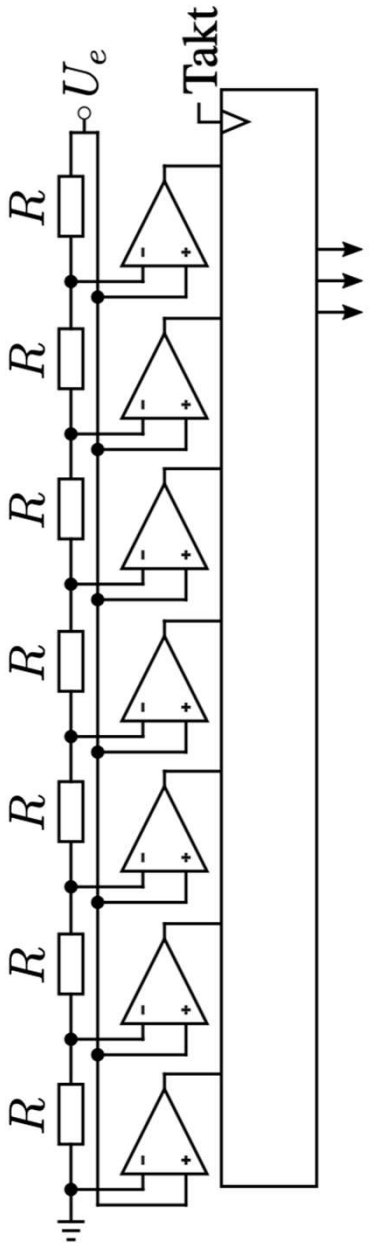
Embedded Systems – A/D Wandler

Parallele Wandler – Flash Converter

Bei parallelen Wandlern (engl.: *flash converters*) wird die zu messende Spannung gleichzeitig mit mehreren Referenzspannungen verglichen.

Dazu wird der gültige Messbereich in 2^n Intervalle unterteilt und die Eingangsspannung mit jedem dieser Intervalle verglichen. n ist die Genauigkeit in Bits.

Die $2^n - 1$ Vergleichsergebnisse werden von einem Prioritätsdekodierer in einen Binärwert umgewandelt.



3-Bit Wandler

Embedded Systems – A/D Wandler

Zählverfahren

A/D Wandler nach dem Zählverfahren erfordern den geringsten Schaltungsaufwand.

Allerdings ist die Wandlungszeit wesentlich größer als bei den anderen Verfahren.

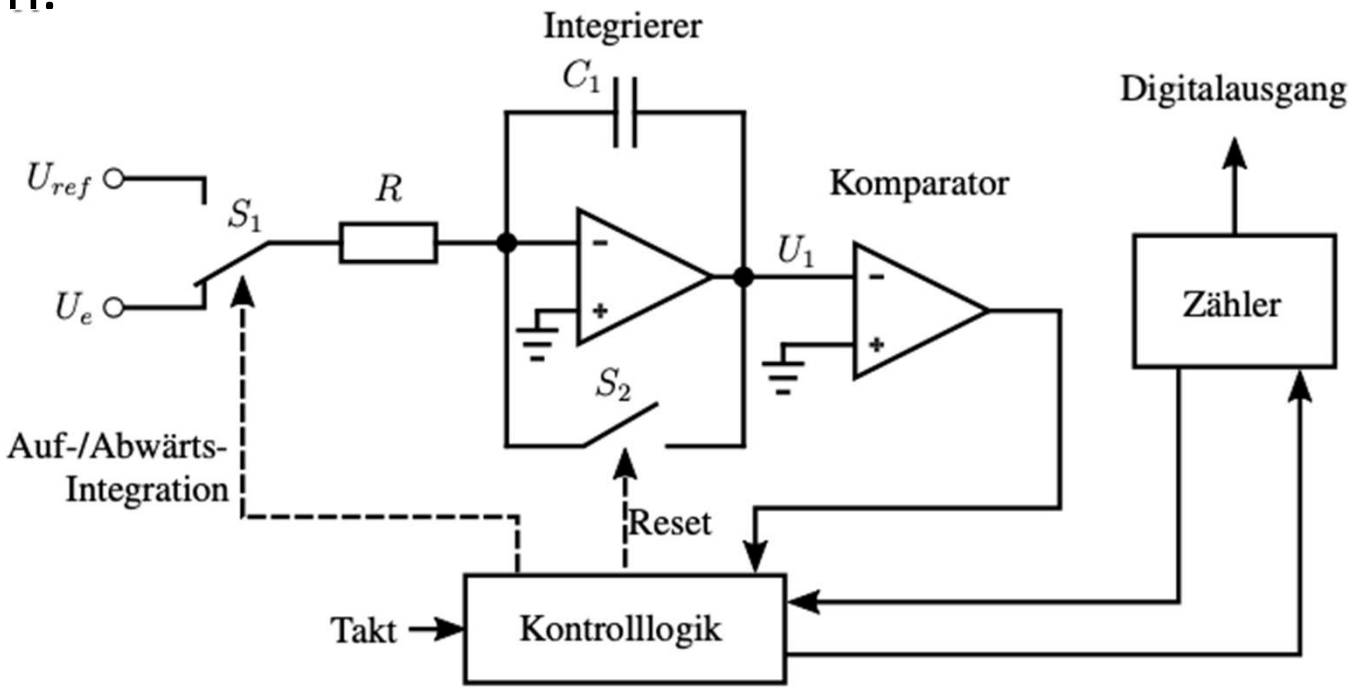
Sie genügt jedoch bei langsam veränderlichen Signalen, die z.B. bei einer Temperaturmessung auftreten.

Es gibt verschiedene Realisierungsformen für das Zählverfahren, das bedeutendere ist das Dual-Slope Verfahren.

Man bekommt bei geringstem Aufwand die größte Genauigkeit.

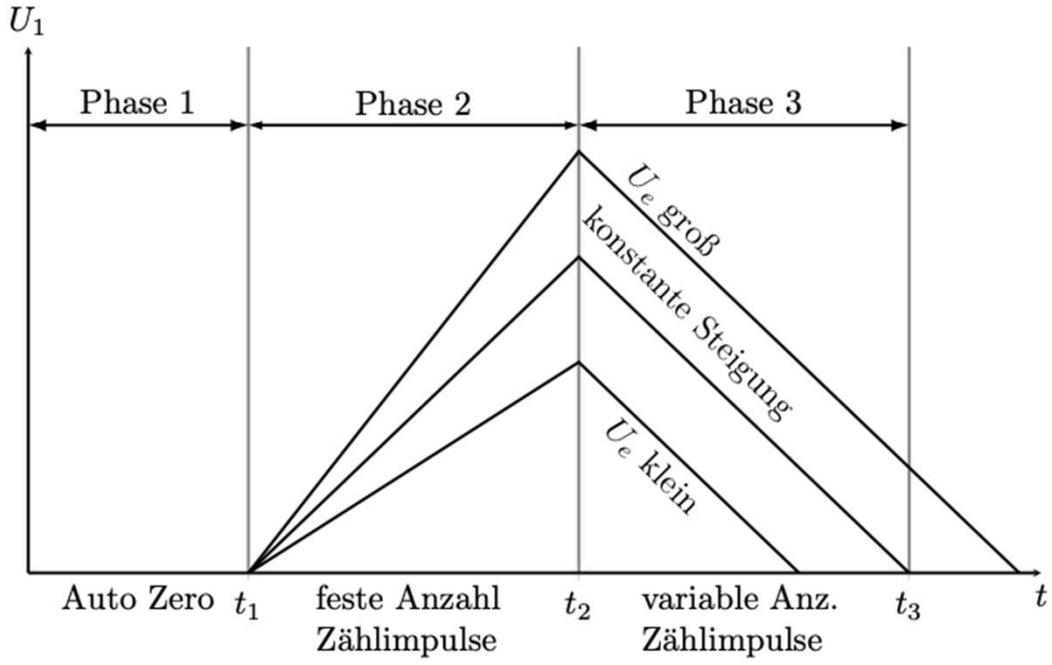
Zählverfahren – Dua-Slope Verfahren

Bei den integrierenden A/D-Wandlern werden Spannungen schrittweise auf- bzw. abintegriert, um die Eingangsspannung anzunähern.



Embedded Systems – A/D Wandler

Zählverfahren – Dua-Slope Verfahren



$$U_e = \frac{t_3}{t_2} * U_{ref} = U_{ref} * \frac{nT}{NT} = \frac{U_{ref}}{N} * n$$

T … Periodendauer

1. Phase = Reset Phase, Kondensator C wird entleert, Zähler auf Null.

2. Phase => Eingangsspannnung wird über eine feste Zeit (N .. Takte) aufintegriert, so dass die Spannung am Komparator proportional zur Eingangsspannung ist.

3. Phase => Es wird Schrittweise mit der Referenzspannung abintegriert, bis am Komparator 0V anliegt. Die Zeit bzw. Anzahl n der hierfür notwendigen Takte ist proportional zur Eingangsspannung.

Embedded Systems – A/D Wandler

Beispiel A/D Wandler

Serielle Schnittstellen

Serielle Schnittstellen werden selbst erarbeitet.

Embedded System – erweiterte Peripherie

D/A-Wandler

Neben den bereits beschriebenen A/D-Wandlern findet man auch häufig D/A-Wandler (digital to analog converter).

Wie bei den A/D Wandlern unterscheidet man auch bei den D/A-Wandler in die drei verschiedenen Verfahren:

- Parallelverfahren
- Wägeverfahren
- Zählverfahren

Embedded System – erweiterte Peripherie

Seriellen Kommunikation

Zur weiteren seriellen Kommunikation findet man den

- CAN-Bus (controller area network),
- ein USB-Interface oder
- eine I^2S Schnittstelle (Inter-IC sound).

Das Controller Area Network (CAN) verbindet mehrere gleich-berechtigte Komponenten (Knoten, Node) über einen 2-Draht Bus miteinander. Aufgrund der hohen Störsicherheit, der geringen Kosten und der Echtzeitfähigkeit wird CAN eingesetzt.

Die I^2S Schnittstelle dient zum Austausch von Stereo-Audio-Daten zwischen ICs.

Embedded System – erweiterte Peripherie

Watchdog

Mit Hilfe eines Watchdogs wird die Sicherheit erhöht.

Der Watchdog wird in definierten Abständen von der Applikation angesprochen und wirkt damit wie ein “Totmann-Schalter“. Unterbleibt die Aktivierung, wird das System hart zurückgesetzt.

Das ist zwar nicht ungefährlich, aber möglicherweise besser, als wenn es in einem Dead-Lock hängen bleibt.

Embedded System – erweiterte Peripherie

DMA - Direct memory access

Eine sehr nützliche Funktion ist der direkte Speichertransfer DMA (direct memory access).

Unter Direct Memory Access versteht man, wenn Komponenten selbstständig ohne Beteiligung der CPU Daten übertragen können.

Peripheriekomponenten können, ohne Umweg über die CPU direkt mit dem Arbeitsspeicher kommunizieren.

Der Vorteil des DMA ist die schnellere Datenübertragung. Die Aufgabe wird dem DMA-Controller zugewiesen.

Software – Bitmanipulation in C

Allgemein

Bei Microcontrollern werden sehr viele Einstellungen der CPU und der Peripherieteile über Register gesetzt.

Register sind nichts anderen als Speicherbereiche mit fixen Adressen. Dort stehen einzelnen Bits oder Gruppen von Bits für verschiedene Parameter.

Mikrocontroller werden typischerweise in C programmiert. Deshalb sind in diesem Dokument auch alle Beispiele in der Programmiersprache C geschrieben.

Zum Setzen und Rücksetzen werden Bitweise Logikoperatoren verwendet.

Software – Bitmanipulation in C

Bitweises Oder

Ein Bitweises **Oder** wird mit dem Symbol **|** geschrieben. Dabei werden von 2 Variablen alle Bits einzeln miteinander mit einem logischen Oder verknüpft. Sobald ein Bit eine 1 ist, ist auch das Bit in der Ergebniszahl 1. Z.B.

Listing 1: Bitweises oder

```
1 uint_8 v1 = 0x3C;
2 //0010 1100
3 uint_8 v2 = 0x1;
4 //0000 0001
5 uint_8 v3 = v1 | v2;
6 //0010 1101
```

Ein Bitweises oder kann zum Setzen von einzelnen oder mehreren Bits verwendet werden.

Software – Bitmanipulation in C

Bitweises Und

Ein Bitweises **Und** wird mit dem Symbol **&** geschrieben. Dabei werden von 2 Variablen alle Bits einzeln miteinander mit einem logischen Und verknüpft. Sobald ein Bit eine 0 ist, ist auch das Bit in der Ergebniszahl 0. Z.B.

Listing 2: Bitweises und

```
1 uint_8 v1 = 0x3C;
2 //0010 1100
3 uint_8 v2 = 0xF1;
4 //1111 0001
5 uint_8 v3 = v1 & v2;
6 //0010 0000
```

Ein Bitweises und kann zum Rücksetzen von einzelnen oder mehreren Bits verwendet werden.

Software – Bitmanipulation in C

Bitweises Negation

Ein Bitweise **Negation** wird mit dem Symbol **~** geschrieben. Dabei werden alle Bits einer Variable invertiert. Aus jeder logischen 1 wird eine 0 und umgekehrt. Z.B.

Listing 3: Bitweise Negation

```
1 uint_8 v1 = 0x3C;
2 //0010 1100
3 uint_8 v2 = ~v1;
4 //1101 0011
```

Software – Bitmanipulation in C

Bit-Shifting

Mit den Bit-Shifting Operatoren können die einzelnen Bits einer Zahl um einen bestimmte Anzahl entweder nach Links << oder nach Rechts >> verschoben werden. An den freien Stellen wird automatisch eine 0 eingefügt. Z.B.

Listing 4: Bitshifting

```
1 uint_8 v1 = 0x3C;
2 //0010 1100
3 uint_8 v2 = v1 << 1;
4 //0101 1000
5 uint_8 v3 = v1 >> 2;
6 //0000 1011
```

Software – Bitmanipulation in C

Setzen von einzelnen Bits

Um bestimmte Einstellungen zu setzen müssen in einen Register einzelnen Bits auf den Wert 1 gesetzt werden. Z.B.

Listing 5: Setzen von einzelnen Bits

```
1  // Das Register das wir verändern möchten liegt auf der
       Adresse 0x50000000.
2  //Wo diese Register dann bei einem spezifischen
       Microcontroller liegen kann dem Handbuch entnommen
       werden.
3  #define GPIO0 ((uint_8*)0x50000000)
4
5  //Das Register hat jetzt den Binärwert 0000 0000
6  //Liegt z.B. auf dem zweiten Bit von rechts (Bit Nr. 1)
       eine Einstellung die wir setzen wollen definieren wir
       uns dafür eine Bitmaske.
7
8  uint_8 mySetting = 1;
9  uint_8 mask = 1 << mySetting;
10 //0000 0010
11
12 //Wollen wir jetzt das Bit setzen können wir das Register
       mit der Maske bitweise verodern.
13
14 GPIO0 |= mask;
15
16 //1.) GPIO = 0000 0000
17 //2.) mask = 0000 0010
18 //3.) GPIO = 0000 0010
19
20 Die anderen Bits dieses Registers sind davon nicht
       betroffen. Wenn z.B schon andere Bits gesetzt wurden.
21 //1.) GPIO = 0011 0100
22 //2.) mask = 0000 0010
23 //3.) GPIO = 0011 0110
```

Software – Bitmanipulation in C

Rücksetzen von einzelnen Bits

Umgekehrt ist es auch manchmal notwendig einzelne Bits rückzusetzen. Dafür kann ähnlich vorgegangen werden wie beim Setzen.

Listing 6: Rücksetzen von einzelnen Bits

```
1  // Das Register das wir verändern möchten liegt auf der
       Adresse 0x50000000.
2  //Wo diese Register dann in Wirklichkeit liegen kann dem
       Microcontroller handbuch entnommen werden.
3  #define GPIO0 ((uint_8*)0x50000000)
4
5  //Das Register hat jetzt z.B den Binärwert 0011 0010
6  //Liegt z.B. auf dem zweiten Bit von rechts (Bit Nr. 1)
       eine Einstellung die wir rücksetzen wollen können wir
       die selbe Bitmaske wie zum setzen verwenden.
7
8  uint_8 mySetting = 1;
9  //0000 0001
10 uint_8 mask = 1 << mySetting;
11 //0000 0010
12
13 //Wollen wir jetzt das Bit rücksetzen können wir das
       Register mit der invertierten Maske bitweise verunden.
14 GPIO0 &= ~mask;
15
16 //1.) GPIO = 0011 0010
17 //2.) mask = 0000 0010
18 //3.)~mask = 1111 1101
19 //4.) GPIO = 0011 0000
```

Embedded System – Notizen