Zusammenfassung

Embedded Systems (Hochschule Darmstadt)

messages.pdf_cover_qr_code_label

01

1. Mikrocontroller-Markt und ARM-Architekturen

- **ARM-Dominanz:** ARM (Advanced Risc Machines) definiert CPU-Kerne in Cortex-Architekturen, die den Mikrocontroller-Markt dominieren.
- **ARM-Cortex-Architekturen:**
    - **Cortex-A:** Application (betriebssystembasierte Anwendungen).
    - **Cortex-R:** Realtime (Echtzeitanwendungen).
    - **Cortex-M:** Microcontroller (Cores für Mikrocontroller, Nachfolger ARM7).
- **Verbreitung:** Über 90% aller mobilen Geräte enthalten einen ARM-Prozessor. Bis 2022 wurden 230 Milliarden ARM-Prozessoren ausgeliefert.
- **Vorgestellte Familien:** In dieser Veranstaltung werden ARM7TDMI, Cortex-M0+ und Cortex-M3 vorgestellt.

2. Mikroprozessor vs. Mikrocontroller

- **Mikroprozessor:** Ein Mikrorechner auf einem Chip, der die Bausteine des Prozessors auf einem integrierten Schaltkreis (IC) vereinigt.
- **Mikrocontroller (MC):** Zusätzlich zum Prozessor ist Peripherie integriert. Er ist für spezielle, oft einmal programmierte Steuerungs- oder Kommunikationsaufgaben zugeschnitten.
- **Aufbau eines MC (System-on-Chip/SoC):** Ein Ein-Chip-Mikrorechner mit aufgabenspezifischer Peripherie. Typische Bestandteile sind Prozessor-kern, RAM, ROM/EPROM/EEPROM, Takt, Ein-/Ausgabe-steuerung, Unterbrechungs-steuerung, Zähler/Zeitgeber und Erweiterungsbusschnittstelle.
- **Fehlende Komponenten in MCs:** Oft fehlen Speicherverwaltungseinheiten (MMU), Virtualisierungsfunktionen oder Spezialbefehle für Vektorverarbeitung.
- **Designziele:** Möglichst wenige externe Bausteine für eine Steuerungsaufgabe (Idealfall: MC, Quarz, Stromversorgung, ggf. Treiber/Bedienfeld).

3. Anwendungsfelder von Embedded Systems

- **Klassische Anwendungen:** Haushalt (Kaffeemaschine, Waschmaschine, Fernseher) und KFZ-Technik (Motormanagement, ABS, Fahrassistenten).
- **Automatisierung:** Steuern, Regeln und Überwachen von Prozessen, Steuerung von Fertigungsanlagen.

- **Weitere Bereiche:** Mobile Devices (Multitouch, GPS, Funkprotokolle) und IT-Infrastruktur (RAID-Controller, Netzwerk-Router, Verschlüsselungskomponenten).
- **Wechselwirkung:** Ein eingebettetes System benötigt Sensoren und Aktoren, um mit seiner Umgebung wechselwirken zu können.

4. Wichtige Mikrocontroller-Komponenten

- **Zähler und Zeitgeber (Counter/Timer):** Wichtig für Echtzeitanwendungen, z.B. zum Zählen von Ereignissen, Messen von Zeiten oder Pulsweitenmodulation.
    - **Echtzeitbedingungen:** Harte (garantierte Einhaltung der Reaktionszeit) oder weiche (statistische Garantie).
- **Watchdog:** Überwacht die Programmaktivitäten; bleibt ein Lebenszeichen aus, erfolgt ein Reset.
- **Interrupts (Unterbrechungen):** Unterbrechen den Programmablauf bei Ereignissen für eine schnelle, vorhersagbare Reaktion (wichtig bei Echtzeit).
- **DMA (Direct Memory Access):** Direkter Datentransfer zwischen Peripherie und Speicher ohne Beteiligung des Prozessorkerns, zur Erhöhung der Datenrate und Entlastung des Kerns.

5. Labor-Hardware und Peripherieprogrammierung

- **Verwendete Mikrocontroller/Boards:**
    - Atmel SAM3X8E auf dem **Arduino Due** (ARMv7, ARM Cortex M3).
    - Raspberry RP2040 auf dem **Raspberry Pi Pico** (ARMv6, ARM Cortex M0+).
    - (Ältere) Atmel AT91M63200 auf dem AT91EB63.
- **Programmierung der Peripherie:** Erfolgt meist über **Memory Mapped I/O** (z.B. ARM-Architektur). Die Peripherie wird in den Adressraum des Hauptspeichers eingeblendet und über das Lesen/Schreiben der Register-Adressen gesteuert. Die Übersicht liefert die Memory Map.
- **Board Support Package (BSP):** Liefert alle spezifischen Angaben (Adressen, Datenstrukturen, Konstanten) und den Startupcode zur Initialisierung des Boards.

6. Sichtbarkeit von HW-Eigenschaften in C

- **volatile:** Typqualifikator, der dem Compiler mitteilt, dass sich der Wert einer Variablen außerhalb des aktuellen Programmkontextes (z.B. durch Hardware) ändern kann. Dies verhindert eine fehlerhafte Optimierung durch den Compiler.

- **static:** Reduziert die Sichtbarkeit einer globalen Variablen auf Modulgrenzen.
- **Bitweise Operatoren:** Operatoren wie &, |, ~, ^ sind für die bitweise Verknüpfung von Operanden wichtig.
- **Hardwarenahe Deklarationen:** Hardwarebausteine werden oft als zusammengesetzte Datenstruktur (struct) deklariert, wobei die Adressen als Konstanten (Präprozessordirektiven) bekannt gemacht werden.

7. Compiler-Optimierung

- **Herausforderung:** Optimierungsanalysen erfolgen rein auf Quelltextbasis und erkennen keine externen Einflüsse (wie Hardware-Änderungen), weshalb Optimierungen bei hardwarenaher Programmierung vorsichtig eingesetzt werden sollten.
- **GCC-Optimierungsstufen:**
    - **-O0:** Keine Optimierung, schnell zu kompilieren, perfekt zum Debuggen.
    - **-O1:** Leichte Optimierung, meist ohne Risiko.
    - **-O2:** Standard für Releases; gute Balance aus Geschwindigkeit/Größe/Stabilität.
    - **-O3:** Aggressiv; bessere Laufzeit, aber längere Kompilierzeit, u. U. größere Binaries.
    - **-Os:** Optimiert für kleine Codegröße (wichtig bei Flash-Budget).

01a

Zusammenfassung: ARM Cortex-M3 Architektur und Programmiermodell

Dieses Dokument befasst sich mit der Architektur und dem Programmiermodell des **ARM Cortex-M3** Prozessorkerns, der häufig in eingebetteten Systemen verwendet wird. Es enthält auch einen Vergleich mit der älteren **ARMv4/ARM7TDMI**-Architektur.

Cortex-M3 Pipeline (3 Stufen)

Der Cortex-M3 verwendet eine **3-stufige Pipeline**, bestehend aus:

- **F** - Fetch (Holen des Befehls)
- **D** - Decode (Dekodieren des Befehls)
- **E** - Execute (Ausführen des Befehls)

Flushing (Löschen) der Pipeline: Ein "Flush" der Pipeline kann durch ein **Branch** (Sprung), eine **Exception** (Ausnahme) oder einen **Breakpoint** ausgelöst werden.

Cortex-M3 Programmiermodell (Modi und Privilegien)

Prozessormodi

Der Cortex-M3 kennt zwei Haupt-Prozessormodi:

- **Thread Mode:** Wird zur Ausführung der **Anwendungssoftware** genutzt. Der Prozessor tritt beim Reset in diesen Modus ein.
- **Handler Mode:** Wird zur **Behandlung von Exceptions** (Ausnahmen) verwendet. Nach Abschluss der Exception-Verarbeitung kehrt der Prozessor in den Thread Mode zurück.

Privilegien-Levels

Die Software-Ausführung kann auf zwei Privilegien-Levels erfolgen:

- **Privileged (Privilegiert):** Die Software kann alle **Instruktionen** nutzen und hat **Zugriff auf alle Ressourcen**. Im Handler Mode ist die Ausführung immer privilegiert.
- **Unprivileged (Unprivilegiert):**
    - Hat **eingeschränkten Zugriff** auf MSR- und MRS-Instruktionen und kann CPS nicht verwenden.
    - Kann **nicht** auf den System-Timer, NVIC oder den System Control Block zugreifen.
    - Kann **eingeschränkten Zugriff** auf Speicher oder Peripherie haben.
- **Steuerung:** Im Thread Mode steuert das **CONTROL-Register** den Privilegien-Level. Nur privilegierte Software kann in das CONTROL-Register schreiben, um den Level im Thread Mode zu ändern. Unprivilegierte Software kann die SVC-Instruktion (Supervisor Call) nutzen, um die Kontrolle an privilegierte Software zu übertragen.

Register-Sets

Cortex-M3 Register-Set

Der Cortex-M3 verwendet 32-Bit-Register:

- **R0 bis R12:** Allzweckregister für Datenoperationen.

    - **Low Registers (R0-R7):** Zugänglich durch alle 16-Bit-Thumb-Instruktionen und alle 32-Bit-Thumb-2-Instruktionen.
    - **High Registers (R8-R12):** Zugänglich durch alle Thumb-2-Instruktionen, aber nicht durch alle 16-Bit-Thumb-Instruktionen.
- **R13 (SP - Stack Pointer):** Besteht aus zwei **banked** (umschaltbaren) Stack Pointern, von denen immer nur einer sichtbar ist:
    - **MSP (Main Stack Pointer):** Standard-Stack-Pointer, verwendet vom OS-Kernel und Exception Handlern.
    - **PSP (Process Stack Pointer):** Wird vom Benutzer-Anwendungscode verwendet.
    - Die niedrigsten zwei Bits der Stack Pointer sind immer 0 (Word-Aligned).
- **R14 (LR - Link Register):** Wird verwendet, um die Rücksprungadresse zu speichern.
- **R15 (PC - Program Counter):** Enthält die aktuelle Programm-Adresse und kann zur Steuerung des Programmflusses beschrieben werden.
- **Special Registers (xPSR):** Dazu gehören Program Status Registers (PSRs), Interrupt Mask Registers (PRIMASK, FAULTMASK, BASEPRI) und das Control Register (CONTROL). Sie werden nur über die Instruktionen **MSR** (Write) und **MRS** (Read) angesprochen, da sie **keine Speicheradressen** haben.

ARMv4/ARM7TDMI Register-Set (Vergleich)

Die ältere ARMv4/ARM7TDMI-Architektur, wie im AT91 verwendet , verfügt über **mehrere Modi** (User, FIQ, SVC, Abort, IRQ, Undefined) , die für das Exception Handling relevant sind.

- Viele Register sind **modusspezifisch** (banked).
- **CPSR (Current Program Status Register):** Enthält die Status- und Kontroll-Flags:
    - **Condition Code Flags (NZCV):** Negatives Ergebnis (), Ergebnis ist Null (), Carry (), Overflow ().
    - **T Bit:** bedeutet ARM State, bedeutet Thumb State.
    - **Q Bit:** Für DSP-Kontext (Cumulative Saturation Bit).
    - **Interrupt Disable Bits (IFT):** deaktiviert IRQ, deaktiviert FIQ.
    - **Mode Bits:** Spezifizieren den Prozessor-Modus.

- **SPSR (Saved Program Status Register):** Ein modusspezifisches Register, das den Status des CPSR beim Eintreten einer Exception speichert.

02

-

Power Management, auch als Power Saving Modi oder Stromsparmodi bezeichnet, ist für eingebettete Systeme, insbesondere für **batteriebetriebene Anwendungen**, extrem wichtig, um die **Stromaufnahme zu senken**.

Zweck und Ziele

Das Hauptziel ist, den **Energieverbrauch zu reduzieren**, indem nicht benötigte Komponenten so weit wie möglich abgeschaltet werden.

- Nicht alle Peripherieeinheiten (wie Timer/Counter, USART, etc.) auf einem Mikrocontroller werden in jeder Anwendung benötigt.
- Auch **CPU-Kerne** können abgeschaltet werden, wenn kein Rechenbedarf besteht (z. B. beim Warten auf einen Interrupt).

Implementierung des Power Managements

Die Reduzierung des Energieverbrauchs kann auf verschiedene Weisen erfolgen:

- **Reduzierung der Taktfrequenz** des Mikrocontrollers.
- **Taktabschaltung** von CPU-Kernen und/oder Peripherie.
- **Tiefschlafzustände**, bei denen weitere Komponenten des Mikrocontrollers abgeschaltet werden, wie z. B. Teile des RAMs (erfordert Rekonstruktionsmöglichkeit des Inhalts bei Aufwachen).
- **Einzelne Funktionsblöcke** lassen sich auch ganz von der Clock trennen (CMOS: kein Umschalten, minimaler Stromverbrauch).

Konkrete Steuerung

- Das **Einschalten von Clocks (Taktgeber)** erfolgt durch **Setzen/Löschen von Bits in Registern**.
- Man kann **unterschiedliche Clocks** wählen und **Clockteiler (Prescaler)** definieren (langsamere Clock weniger Energieverbrauch).
- **Beim Systemstart** sind typischerweise die Peripheriefunktionen per default aus, während die CPU eingeschaltet ist, um die Funktionen einschalten zu können.

- Ein **genaues Verständnis des hierarchischen Aufbaus** der Clocks und des Power Managements ist notwendig, da eine fehlerhafte Einstellung zur Nichtfunktion einer Komponente führen kann.

Power Management in der Laborhardware

Mikrocontroller	Fähigkeiten
AT91	Ein- und Ausschalten einzelner Peripheriekomponenten sowie der CPU mittels **Power Management Unit**.
SAM3X	Ein- und Ausschalten einzelner Peripheriekomponenten sowie der CPU mittels **Power Management Unit**. Taktfrequenzen können gewählt werden. Eine **Peripheral ID (PID)** unterscheidet Peripheriekomponenten.
RP2040	Keine spezifischen Eigenschaften zum Power Management

02a

-

Power Management, auch als Power Saving Modi oder Stromsparmodi bezeichnet, ist für eingebettete Systeme, insbesondere für **batteriebetriebene Anwendungen**, extrem wichtig, um die **Stromaufnahme zu senken**.

Zweck und Ziele

Das Hauptziel ist, den **Energieverbrauch zu reduzieren**, indem nicht benötigte Komponenten so weit wie möglich abgeschaltet werden.

- Nicht alle Peripherieeinheiten (wie Timer/Counter, USART, etc.) auf einem Mikrocontroller werden in jeder Anwendung benötigt.
- Auch **CPU-Kerne** können abgeschaltet werden, wenn kein Rechenbedarf besteht (z. B. beim Warten auf einen Interrupt).

Implementierung des Power Managements

Die Reduzierung des Energieverbrauchs kann auf verschiedene Weisen erfolgen:

- **Reduzierung der Taktfrequenz** des Mikrocontrollers.

- **Taktabschaltung** von CPU-Kernen und/oder Peripherie.
- **Tiefschlafzustände**, bei denen weitere Komponenten des Mikrocontrollers abgeschaltet werden, wie z. B. Teile des RAMs (erfordert Rekonstruktionsmöglichkeit des Inhalts bei Aufwachen).
- **Einzelne Funktionsblöcke** lassen sich auch ganz von der Clock trennen (CMOS: kein Umschalten, minimaler Stromverbrauch).

Konkrete Steuerung

- Das **Einschalten von Clocks (Taktgeber)** erfolgt durch **Setzen/Löschen von Bits in Registern**.
- Man kann **unterschiedliche Clocks** wählen und **Clockteiler (Prescaler)** definieren (langsamere Clock weniger Energieverbrauch).
- **Beim Systemstart** sind typischerweise die Peripheriefunktionen per default aus, während die CPU eingeschaltet ist, um die Funktionen einschalten zu können.
- Ein **genaues Verständnis des hierarchischen Aufbaus** der Clocks und des Power Managements ist notwendig, da eine fehlerhafte Einstellung zur Nichtfunktion einer Komponente führen kann.

Power Management in der Laborhardware

Mikrocontroller	Fähigkeiten
AT91	Ein- und Ausschalten einzelner Peripheriekomponenten sowie der CPU mittels **Power Management Unit**.
SAM3X	Ein- und Ausschalten einzelner Peripheriekomponenten sowie der CPU mittels **Power Management Unit**. Taktfrequenzen können gewählt werden. Eine **Peripheral ID (PID)** unterscheidet Peripheriekomponenten.
RP2040	Keine spezifischen Eigenschaften zum Power Management

02a

Power Management beim AT91

Beim AT91-Board gibt es zwei Haupttakte : den System-Takt (CPU) und den Peripherie-Takt.

- **Grundeinstellung beim Reset:**
    - Takt für **CPU** ist **eingeschaltet**.
    - Takt für **Peripherie** ist **ausgeschaltet**.

PMC-Register zum Power Management

Die Steuerung erfolgt über die Register der Power Management Controller (PMC) Unit:

Register	Name	Zweck	Zugriff	Reset-Zustand
`PMC_SCER` (Offset 0×00)	System Clock Enable Register	Schaltet CPU-Takt ein	Write only	-
`PMC_SCDR` (Offset 0×04)	System Clock Disable Register	Schaltet CPU-Takt aus	Write only	-
`PMC_PCER` (Offset 0×10)	Peripheral Clock Enable Register	Schaltet Peripherie-Takt ein	Write only	-
`PMC_PCDR` (Offset 0×14)	Peripheral Clock Disable Register	Schaltet Peripherie-Takt aus	Write only	-
`PMC_PCSR` (Offset 0×18)	Peripheral Clock Status Register	Status des Peripherie-Takts	Read only	0×0

Programmierung

Der Peripherie-Takt wird durch Setzen des entsprechenden Bits im PMC_PCER Register aktiviert. Beispielsweise schaltet *PMC_PCER = 0x4200 den Peripheral Clock für **PIOB** ein.

Power Management beim SAM3X

Der SAM3X (auf dem Arduino Due) verwendet ebenfalls PMC-Register für das Power Management.

PMC-Register und Peripheral IDs (PID)

Beim SAM3X werden die Peripherie-Takte über **Peripheral Clock Enable Register 0** (PMC_PCER0) und **Peripheral Clock Enable Register 1** (PMC_PCER1) gesteuert.

- Der SAM3X verwendet **Peripheral Identifiers (PID)**, das sind eindeutige Nummern, die jeder Peripheriekomponente zugeordnet sind.
- Die Bits in den PMC_PCERx Registern korrespondieren mit diesen PIDs.

- Das Setzen eines Bits (1) im entsprechenden PMC_PCERx aktiviert den zugehörigen Peripherie-Takt.

Register	Adresse	Zweck	Zugriff
PMC_PCER0	$0 \times 400E0610$	Peripheral Clock Enable Register 0	Write-only
PMC_PCER1	$0 \times 400E0700$	Peripheral Clock Enable Register 1	Write-only

Wichtige Peripheral IDs (Ausschnitt)

PID	Instance Name	Instance Description
11	PIOA	Parallel I/O Controller A
12	PIOB	Parallel I/O Controller B
17	USARTO	Universal Synchronous Asynchronous Receiver Transmitter 0
18	USART1	Universal Synchronous Asynchronous Receiver Transmitter 1
27	TC0	Timer Counter Channel 0
36	PWM	Pulse Width Modulation Controller
37	ADC	ADC Controller

Programmierung (Beispiel)

Um beispielsweise den Takt für **USART1** (PID **18**) einzuschalten, wird das 18. Bit in PMC_PCER0 gesetzt:

```
// turn on clock for USART1 (PID 18)

PMC->PMC_PCER0 = 1 << ID_USART1;
```

Takthierarchie

Der SAM3X verwendet einen komplexen **Clock Generator** und einen **Master Clock Controller** (PMC_MCKR), um den **Master Clock (MCK)** zu erzeugen. Der MCK wird dann an verschiedene Controller verteilt:

- **Processor Clock Controller** (erzeugt HCLK/Processor clock).
- **Peripherals Clock Controller** (PMC_PCERx) (erzeugt periph_clk [..]).

- **Programmable Clock Controller** (PMC_PCKX) (erzeugt pck [..]).
- **USB Clock Controller** (PMC_USB) (erzeugt USB-Takte).

03

1. Grundlagen der Parallelen I/O

Parallele I/O (Input/Output) stellt die einfachste Methode dar, um einen Mikrocontroller mit der Außenwelt zu verbinden, Signale zu steuern oder Zustände zu messen.

- **Funktionsweise:** Die Kommunikation erfolgt rein digital.
    - **Zustände:** Ein/Aus, High/Low, 1/0 .
    - **Richtung:** Pins können als **Eingang** (z. B. Taster abfragen) oder **Ausgang** (z. B. LED einschalten) konfiguriert werden .
- **Struktur:**
    - **Portpins:** Dies sind die physischen elektrischen Anschlüsse am Chip.
    - **Ports:** Mehrere Pins werden oft logisch zu einem Port zusammengefasst (z. B. 32 Pins bilden einen Port).
    - **GPIO:** Die Bezeichnung "General Purpose Input Output" unterstreicht die vielseitige Nutzbarkeit.

2. Pins als Ressource und Multiplexing

Da die Anzahl der physischen Pins an einem Mikrocontroller begrenzt ist, sind sie eine knappe Ressource. Um mehr Funktionen als Pins bereitzustellen, nutzen moderne Controller **Pin Multiplexing (PinMux)**.

- **Konfigurierbarkeit:** Ein Pin kann entweder als einfacher I/O-Pin oder für spezielle Schnittstellen (USART, I2C, Timer) genutzt werden.
- **Steuerung:** Die Konfiguration erfolgt über Register (z. B. Richtung, Pull-Up/Pull-Down Widerstände, Sonderfunktionen) .

3. Anschluss von Peripherie

LEDs

Beim Anschluss von LEDs wird zwischen zwei Logiken unterschieden:

- **Ausgang Low-aktiv:** Die LED leuchtet, wenn der Ausgang auf "Low" (Masse) liegt. Dies ist die häufigere Variante, da der Prozessor den Strom nicht aktiv liefern muss (Sink Current) .
- **Ausgang High-aktiv:** Die LED leuchtet, wenn der Ausgang auf "High" (Spannung) liegt (Source Current).

Taster und Schalter

Taster können "Active high" oder "Active low" angeschlossen werden und benötigen Pull-Up- oder Pull-Down-Widerstände für definierte Pegel.

- **Entprellen:** Mechanische Taster müssen entprellt werden, um Fehlinterpretationen zu vermeiden. Dies kann über **Hardware** (FlipFlop, Kondensator) oder **Software** (Zeitmessung/Timer) geschehen .

7-Segment-Anzeigen

Zur Darstellung von Ziffern werden 7 Segmente (a bis g) angesteuert.

- **Ohne Decoder:** Jeder Segmentstrich benötigt einen eigenen Pin. Bei 4 Ziffern wären das z. B. 32 Pins (oder 12 im Multiplex).
- **Mit BCD-Decoder:** Ein Decoder wandelt binäre Zahlen (BCD) in die 7-Segment-Signale um. Dies spart Pins (z. B. nur 9 Pins für 4 Ziffern).
- **Zeitmultiplex:** Mehrere Stellen werden nacheinander sehr schnell an- und ausgeschaltet, um Leitungen zu sparen.

Tastaturmatrix

Um bei vielen Tasten Pins zu sparen, werden diese in einer Matrix (Reihen und Spalten) angeordnet.

- **Funktionsweise:** Die Spalten werden nacheinander aktiviert (Output), während die Reihen gemessen werden (Input), um gedrückte Tasten zu identifizieren.

4. Laborhardware im Vergleich

Die Vorlesung vergleicht die I/O-Fähigkeiten verschiedener Mikrocontroller:

Mikrocontroller	Eigenschaften der Parallelen I/O
AT91	• Zwei PIO-Ports (je 32 Pins).
	• Zustandsänderung durch Enable/Disable-Konzept.
	• Interrupt-fähig.
SAM3X (Arduino Due)	• Vier PIO-Ports.
	• Ähnlich AT91, aber mit Port Multiplexing für erweiterte Funktionen.
	• Zusätzliche Funktionalitäten.
RP2040 (Raspberry Pi Pico)	• **Programmable IO (PIO):** State Machines erlauben flexible Verwendung unabhängig von der CPU.
	• Digitale Schnittstellen (UART, I2C) können per Software über State Machines gebildet werden.
	• Register für Set/Clear/Toggle.

03a

1. Grundlagen der PIO-Architektur

Die parallele Ein-/Ausgabe (PIO) wird über Register gesteuert, die in den Speicher eingeblendet sind (Memory Mapped I/O).

- **Struktur:** Pins werden zu Ports zusammengefasst (z.B. 32 Pins pro Port).
- **Verfügbarkeit:**
    - **AT91M63200:** Hat zwei Ports (A und B).
    - **SAM3X8E (Arduino Due):** Hat vier Ports (A, B, C, D) mit erweiterter Funktionalität.

2. Das Enable/Disable/Status-Konzept

Atmel verwendet ein spezielles Register-Konzept, um atomare Operationen zu ermöglichen (vermeidet Read-Modify-Write-Probleme):

- **Enable Register (PER):** Schreiben einer 1 aktiviert eine Funktion (z.B. Pin als PIO nutzen).
- **Disable Register (PDR):** Schreiben einer 1 deaktiviert die Funktion.

- **Status Register (PSR):** Lesen gibt den aktuellen Zustand zurück (1 = aktiv, 0 = inaktiv) .

Dieses Konzept zieht sich durch fast alle Funktionen (Output Enable, Interrupt Enable, Input Filter, Multi-Driver etc.).

3. Register-Übersicht (Wichtige Beispiele)

Die Steuerung eines Ports erfolgt über diverse Registeroffsets. Die wichtigsten sind:

- **PIO_PER/PDR:** PIO Enable/Disable (Pin wird von PIO-Controller oder Peripherie gesteuert).
- **PIO_OER/ODR:** Output Enable/Disable (Pin ist Ausgang oder Eingang).
- **PIO_SODR/CODR:** Set/Clear Output Data (Setzt Ausgang auf High oder Low).
- **PIO_PDSR:** Pin Data Status Register (Liest den aktuellen Pegel am Pin).
- **PIO_PUER/PUDR:** Pull-Up Enable/Disable (Internen Pull-Up-Widerstand schalten).
- **PIO_ABSR (nur SAM3X):** Wählt zwischen Peripherie-Funktion A oder B.

4. Struktur eines Portpins

Die Dokumente zeigen detaillierte Blockdiagramme für die interne Verschaltung eines Pins.

- **Ausgang (Output):** Datenfluss geht vom PIO_SODR/CODR über Multiplexer zum Pad. Wenn Peripherie aktiviert ist, kommen die Daten direkt vom Peripheriemodul (z.B. PWM, USART) .
- **Eingang (Input):** Das Signal vom Pad läuft durch einen optionalen Glitch-Filter und/oder Debouncing-Filter (nur SAM3X) in das PIO_PDSR. Zusätzlich kann eine Flankenerkennung (Event Detection) Interrupts auslösen .

5. Multiplexing (Besonderheit SAM3X)

Da Mikrocontroller mehr interne Funktionen als externe Pins haben, müssen Pins mehrfach belegt werden (Multiplexing).

- **AT91:** Einfachere Struktur, meist feste Zuordnung.
- **SAM3X:** Jeder Pin kann als **GPIO**, **Peripheral A** oder **Peripheral B** konfiguriert werden.
    - Gesteuert durch das Register PIO_ABSR (Peripheral AB Select Register).

- Um eine Peripheriefunktion (z.B. USART TXD) zu nutzen, muss der PIO-Controller für diesen Pin deaktiviert (PIO_PDR) und die korrekte Peripherie (A oder B) gewählt werden .

6. Code-Beispiele

Das Skript liefert C-Code-Beispiele für den Zugriff über Memory Mapping:

- **Output setzen (Register-Zugriff):** Man schreibt direkt in die Adressen von PIO_SODR (Set) oder PIO_CODR (Clear), um Pins zu schalten, ohne den Zustand anderer Pins zu beeinflussen .
- **Peripherie konfigurieren (SAM3X):**
  1. Clock für Peripherie einschalten (Power Management PMC_PCER).
  2. PIO-Steuerung für die Pins deaktivieren (PIO_PDR).
  3. Peripherie A oder B auswählen (PIO_ABSR).

Beispiel: Aktivieren von USART1 RX/TX auf Port A .

Zusammenfassung der Unterschiede

Feature	**AT91**	**SAM3X (Arduino Due)**
Ports	2 (A, B)	4 (A, B, C, D)
Peripherie-Wahl	Fest verdrahtet (wenn PIO disable)	Wählbar A oder B (PIO_ABSR)
Input Filter	Einfacher Glitch-Filter	Glitch- oder Debouncing-Filter
Pull-Ups	Vorhanden	Vorhanden + Pull-Downs konfigurierbar

04

I^2C-Bus (Inter-Integrated Circuit)

1. Einführung und Entwicklung

- **Herkunft:** Wurde 1982 von Philips eingeführt, um Schaltungsteile innerhalb eines Geräts zu verbinden.
- **Entwicklung:** Startete mit 100 kbit/s, entwickelte sich über den Fast Mode (400 kbit/s) bis hin zum Ultra-Fast Mode (5 Mbit/s) im Jahr 2012.

- **Begrifflichkeiten:** Atmel bezeichnet diese Schnittstelle als **TWI (Two-Wire Interface)**. Obwohl die IEEE neutrale Begriffe wie "Controller/Target" empfiehlt, verwendet die technische Dokumentation aus Konsistenzgründen weiterhin die hierarchischen Begriffe **Master/Slave**.

2. Aufbau des Busses

- **2-Draht-Übertragung:**
    - **SCL (Serial Clock):** Die Taktleitung (bei Atmel TWCK).
    - **SDA (Serial Data):** Die Datenleitung (bei Atmel TWD).
- **Elektrische Eigenschaften:**
    - Beide Leitungen sind über **Pull-up-Widerstände** mit der Versorgungsspannung () verbunden.
    - **Open-Collector/Open-Drain Logik:** Teilnehmer können die Leitung nur aktiv auf "Low" (Masse) ziehen. Ein "High"-Zustand entsteht passiv durch den Pull-up-Widerstand, wenn kein Gerät die Leitung auf Low zieht (Idle High).
- **Topologie:** Es handelt sich um einen Multi-Master-Bus, an den mehrere Teilnehmer (Master und Slaves) angeschlossen werden können.

3. Protokoll und Kommunikation

- **Synchron:** Der Takt (SCL) wird während der Übertragung immer vom aktiven Master erzeugt.
- **Datenformat:**
    - Daten werden byteweise (8 Bit) übertragen, beginnend mit dem MSB (Most Significant Bit).
    - Jedes Byte muss vom Empfänger mit einem **ACK-Bit** bestätigt werden.
- **Adressierung:** Slaves werden über eine 7-Bit-Adresse angesprochen.
- **Übertragungsablauf:**
    1. **Start-Bedingung:** Der Master zieht SDA auf Low, während SCL noch High ist.
    2. **Adressierung:** Master sendet Slave-Adresse + R/W-Bit (0 = Schreiben, 1 = Lesen).
    3. **Datentransfer:** Bytes werden gesendet und bestätigt (ACK = Low). Antwortet der Empfänger nicht, wird ein NACK (High) signalisiert.

4. **Stopp-Bedingung:** Der Master lässt SDA auf High steigen, während SCL High ist.

4. Spezielle Mechanismen

- **Bus Idle:** Wenn SCL und SDA beide High sind, ist der Bus frei.
- **Arbitration (Bus-Zugriffsregelung):** Falls zwei Master gleichzeitig senden wollen:
    - Beide Master überwachen während des Sendens die SDA-Leitung.
    - Sendet ein Master eine "1" (High), liest aber eine "0" (Low), hat er die Arbitrierung verloren (weil ein anderer Master die Leitung auf Low zieht).
    - Der Verlierer stoppt sofort das Senden und wartet, bis der Bus wieder frei ist.
- **Gültigkeit:** Pegelübergänge auf der SDA-Leitung dürfen nur stattfinden, wenn SCL Low ist (außer bei Start/Stopp).

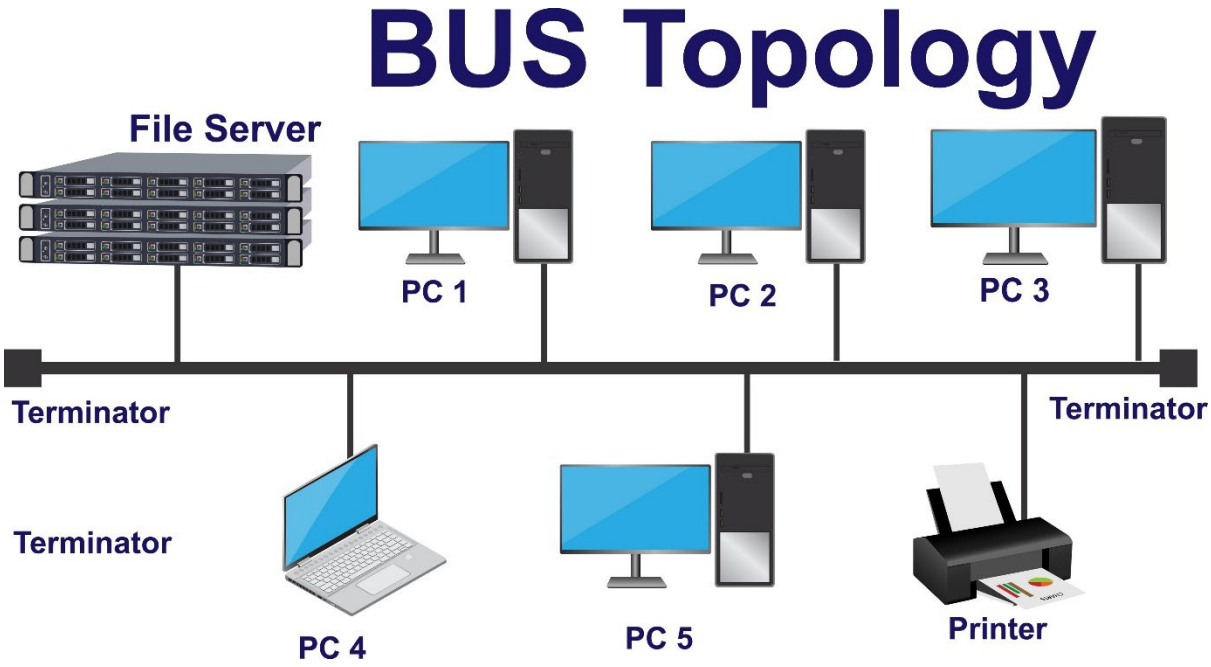
5. Übertragungsgeschwindigkeiten

Die Modi reichen von:

- **Standard Mode:** 100 kBit/s
- **Fast Mode:** 400 kBit/s
- **Fast Mode Plus:** 1 MBit/s
- **High Speed Mode:** 3,4 MBit/s
- **Ultra Fast Mode:** 5 MBit/s (nur unidirektional).

6. Hardware-Vergleich

- **AT91:** Keine Hardware-Unterstützung für I2C.
- **SAM3X (Arduino Due):** Zwei TWI-Schnittstellen, unterstützt Standard und Fast Mode, 7/10-Bit-Adressierung, Interrupt-fähig.
- **RP2040:** Zwei I2C-Schnittstellen, unterstützt bis zu Fast Mode Plus, 7/10-Bit-Adressierung, Interrupt-fähig.



04a

I²C-Bus (TWI) auf dem **SAM3X**-Mikrocontroller (Arduino Due).

1. Architektur und Hardware

Das Two-Wire Interface (TWI) ist die Atmel-Bezeichnung für den I²C-Bus.

- **Blockdiagramm:** Das TWI-Modul ist über eine APB-Bridge mit dem System verbunden und nutzt zwei Leitungen: **TWD** (Data) und **TWCK** (Clock). Diese werden über den PIO-Controller auf die physischen Pins gemultiplext (TWD1=PB12, TWCK1=PB13) .
- **Systemintegration:**
    - **Taktung:** Der Takt (MCK) kommt vom Power Management Controller (PMC).
    - **Interrupts:** Das Modul ist mit dem Nested Vector Interrupt Controller (NVIC) verbunden.
    - **DMA:** Ein Peripheral DMA Controller (PDC) ermöglicht Datentransfers ohne ständige CPU-Last .

2. Register und Steuerung

Die Steuerung erfolgt über Register, die in den Speicher eingeblendet sind. Die wichtigsten sind:

- **Control Register (TWI_CR):** Steuert den Bus (Start/Stop-Bedingung, Reset) und aktiviert Master- oder Slave-Modus (MSEN, SVEN) .

- **Master Mode Register (TWI_MMR):** Konfiguriert die Slave-Adresse (**DADR**), die Lese-/Schreibrichtung (**MREAD**: 0=Write, 1=Read) und die interne Adresslänge (**IADRSZ**) .
- **Status Register (TWI_SR):** Zeigt den Status an, z.B. **TXRDY** (Transmit Ready), **RXRDY** (Receive Ready), **TXCOMP** (Transmission Completed) und Fehler wie **NACK** .
- **Clock Waveform Generator (TWI_CWGR):** Legt die Busgeschwindigkeit fest durch Teiler (**CKDIV**) und Clock-Low/High-Perioden (**CLDIV**, **CHDIV**) .
- **Transmit/Receive Holding Register (THR/RHR):** Dienen zum Senden bzw. Empfangen von Datenbytes .

3. Betriebsmodi

Der SAM3X unterstützt verschiedene Modi, wobei der Fokus auf dem Master-Betrieb liegt:

1. **Master Transmitter:** Sendet Daten an einen Slave. Ablauf: Start -> Adresse+Write -> Daten -> Stop .
2. **Master Receiver:** Empfängt Daten von einem Slave. Ablauf: Start -> Adresse+Read -> Daten -> Stop .
3. **Multi-Master:** Mehrere Master können den Bus nutzen. Eine **Arbitration**-Logik verhindert Kollisionen. Wenn ein Master eine "1" sendet, aber "0" liest, verliert er die Kontrolle und stoppt .
4. **Slave Modes:** Der Due kann auch als Slave fungieren und auf Anfragen eines anderen Masters antworten (Transmitter oder Receiver) .

4. Programmierung und Ablauf

Die Folien beschreiben detaillierte Abläufe für die Initialisierung und Datenübertragung:

Initialisierung (Master + DMA)

1. **PMC:** Peripherie-Clock für TWI einschalten.
2. **PIO:** Pins (PB12, PB13) freigeben und auf Peripherie A multiplexen.
3. **TWI Reset:** Softwarereset und Master-Mode aktivieren.
4. **Clock:** Busfrequenz einstellen (z.B. 400 kHz).
5. **Ziel:** Slave-Adresse in MMR setzen.
6. **DMA:** PDC-Register (TPR, TCR, PTCR) konfigurieren, aber noch nicht starten .

Senden (Polling/CPU)

- Für jedes Byte: Warten bis TXRDY gesetzt ist, dann Daten in THR schreiben.
- Beim letzten Byte die **STOP**-Bedingung im CR setzen.
- Am Ende auf TXCOMP warten .

Empfangen (Polling/CPU)

- **Einzelbyte:** Start und Stop gleichzeitig im CR setzen. Warten auf RXRDY, Byte aus RHR lesen.
- **Mehrere Bytes:** Start setzen. Bytes lesen wenn RXRDY. Vor dem letzten Byte Stop setzen .

DMA vs. CPU

- **DMA (PDC):** Entlastet die CPU bei großen Datenmengen. Pointer- und Counter-Register steuern den Transfer automatisch.
- **CPU (Polling):** Einfacher für einzelne Bytes oder Registerzugriffe, blockiert aber die CPU während des Wartens auf Flags .

05

Serielle Schnittstelle

1. Grundlagen und Überblick

Die serielle Schnittstelle ist eine grundlegende Kommunikationsform in der Computertechnik und eingebetteten Systemen.

- **Seriell:** Bits werden nacheinander über eine Leitung übertragen. Vorteile: Weniger Leitungen, geringere Kosten, keine Laufzeitunterschiede zwischen parallelen Leitungen .
- **Parallel:** Mehrere Bits (z.B. 8 oder 32) werden gleichzeitig über mehrere Leitungen übertragen. Vorteile: Höhere Datenrate bei gleichem Takt. Nachteile: Teurer, Platzbedarf, Laufzeitunterschiede (Skew) begrenzen die Geschwindigkeit .

Typische serielle Schnittstellen: RS-232, I^2C, SPI, CAN, USB, Ethernet, Firewire .

2. Synchron vs. Asynchron

- **Synchrone Übertragung:**
    - Eine zusätzliche Taktleitung (Clock) gibt den Zeitpunkt an, zu dem ein Bit gültig ist.

    - Sender und Empfänger sind durch dieses Signal synchronisiert.
    - Beispiele: SPI, I^2C .
- **Asynchrone Übertragung:**
    - Keine Taktleitung.
    - Sender und Empfänger müssen sich auf Parameter (Baudrate) einigen.
    - Ein Startbit signalisiert den Beginn einer Übertragung; der Empfänger synchronisiert sich darauf.
    - Beispiele: RS-232, CAN .

3. UART und USART

- **UART (Universal Asynchronous Receiver/Transmitter):** Ein Hardware-Baustein für asynchrone serielle Kommunikation (z.B. RS-232).
- **USART (Universal Synchronous/Asynchronous Receiver/Transmitter):** Eine Erweiterung des UART, die auch synchrone Kommunikation unterstützt. Dies ermöglicht Protokolle wie I^2C oder SPI mit derselben Hardware .

4. Das Protokoll (Zeichenrahmen)

Ein Protokoll definiert, wie Bits interpretiert werden. Ein typischer **Zeichenrahmen** (Frame) für asynchrone Übertragung (z.B. 9600 8O1) besteht aus:

1. **Ruhezustand:** Leitung ist auf High (Mark).
2. **Start-Bit:** Ein Low-Pegel (Space) signalisiert den Beginn.
3. **Datenbits:** Meist 7 oder 8 Bit (LSB zuerst) .
4. **Paritätsbit (optional):** Zur Fehlererkennung (Even/Odd).
5. **Stopp-Bit(s):** Ein oder zwei High-Pegel schließen das Zeichen ab und stellen den Ruhezustand wieder her.

Abtastung: Der Empfänger tastet das Signal mit einem Vielfachen der Baudrate (z.B. 16x) ab, um das Startbit sicher zu erkennen und die Bits in ihrer Mitte auszulesen ("Sampling") .

5. Elektrische Schnittstellen (Beispiel RS-232)

- **Logikpegel:** RS-232 verwendet negative Logik und höhere Spannungen (z.B. -15V für logisch 1, +15V für logisch 0), um Störsicherheit über längere Kabel zu gewährleisten. Mikrocontroller (TTL/CMOS) benötigen Pegelwandler (z.B. MAX232), um diese Spannungen zu erzeugen .

- **Leitungen:** Neben TXD (Senden) und RXD (Empfangen) gibt es Handshake-Leitungen wie RTS/CTS (Request to Send / Clear to Send) zur Flusssteuerung.

6. Programmierung (API)

Eine einfache Software-Schnittstelle (Treiber) bietet Funktionen wie:

- serial_init(): Initialisierung (Baudrate, Modus).
- putchar(char c): Senden eines Zeichens.
- getchar(): Empfangen eines Zeichens.
- write() / read(): Senden/Empfangen ganzer Puffer .

7. Laborhardware im Vergleich

Mikrocontroller	UART/USART-Fähigkeiten
AT91	3 USARTs, programmierbare Baudrate, Paritäts-/Fehlererkennung, Echo/Loopback-Modi.
SAM3X (Arduino Due)	Wie AT91, zusätzlich konfigurierbar als SPI, IrDA oder LIN. Fractional Baud Rate Generator für präzisere Takte.
RP2040	2 UARTs, große FIFO-Puffer, Hardware-Flusssteuerung, PIO für weitere serielle Protokolle nutzbar.

05a

seriellen Schnittstelle (USART) auf den Mikrocontrollern AT91 und SAM3X (Arduino Due).

1. Hardware-Architektur

Die USART-Module (Universal Synchronous/Asynchronous Receiver/Transmitter) sind komplexe Peripherieblöcke, die über den **APB** (Advanced Peripheral Bus) angebunden sind.

- **Komponenten:** Ein USART-Kanal besteht aus einem **Empfänger** (Receiver) und einem **Sender** (Transmitter). Beide besitzen eigene Steuerwerke und Taktgeneratoren.
- **Register-Pufferung:** Um Datenverluste zu vermeiden, arbeiten Sender und Empfänger mit doppelter Pufferung:
    - **Shift Register (TSR/RSR):** Hier findet die eigentliche Serialisierung/Deserialisierung statt.

    - **Holding Register (THR/RHR):** Hier schreibt die CPU das nächste zu sendende Byte (US_THR) oder liest das letzte empfangene Byte (US_RHR) .

- **Integration:** Die Module sind mit dem **Interrupt Controller** (für IRQs), dem **Power Management Controller** (PMC, für den Takt) und dem **PIO Controller** (für die Pin-Belegung) verbunden .
- **DMA-Unterstützung:** Es besteht eine direkte Anbindung an den **Peripheral Data Controller (PDC)** für Direct Memory Access (DMA), was die CPU entlastet.

2. Register-Mapping und Steuerung

Die Steuerung erfolgt über *Memory Mapped I/O*. Die Registeroffsets sind bei AT91 und SAM3X weitgehend identisch. Wichtige Register sind:

- **Control Register (US_CR):** Write-only. Dient zum Zurücksetzen (Reset) und Aktivieren/Deaktivieren von Sender (TXEN/TXDIS) und Empfänger (RXEN/RXDIS) .
- **Mode Register (US_MR):** Konfiguriert das Protokoll. Hier werden Parität (PAR), Zeichenlänge (CHRL, z.B. 8 Bit) und Anzahl der Stoppbits (NBSTOP) eingestellt .
- **Status Register (US_CSR):** Read-only. Zeigt den aktuellen Zustand an:
    - TXRDY: Sender bereit für neues Byte.
    - RXRDY: Neues Byte empfangen und bereit zum Abholen.
    - Fehlerflags wie OVRE (Overrun), FRAME (Framing Error), PARE (Parity Error) .
- **Baud Rate Generator Register (US_BRGR):** Legt die Übertragungsgeschwindigkeit fest.

3. Baudratenerzeugung

Die Baudrate wird vom Systemtakt (MCK) abgeleitet.

- **AT91:** Nutzt einen einfachen Clock Divisor (CD). Die Formel für asynchronen Betrieb lautet: $$Baudrate = \frac{SelectedClock}{16 \times CD}$$
- **SAM3X (Arduino Due):** Nutzt einen erweiterten **Fractional Baud Rate Generator**. Neben dem Teiler (CD) gibt es einen **Fractional Part (FP)**. Dies erlaubt eine präzisere Einstellung der Baudrate durch feinere Abstufung.

4. Initialisierung und Konfiguration

Ein typischer Initialisierungsablauf umfasst folgende Schritte :

1. **Power Management:** Takt für das USART-Modul einschalten (PMC_PCER).
2. **PIO Konfiguration:** Die entsprechenden Pins (z.B. PA12/PA13 für USART1 beim SAM3X) müssen vom PIO-Controller freigegeben und an die Peripherie übergeben werden (PIO_PDR, PIO_ABSR).
3. **Reset:** Sender und Empfänger zurücksetzen (US_CR).
4. **Konfiguration:** Baudrate (US_BRGR) und Modus (US_MR) setzen.
5. **Aktivierung:** Sender und Empfänger freigeben (TXEN, RXEN im US_CR).

5. Datenübertragungsmethoden

Polling (Programmgesteuert)

Die CPU fragt aktiv das Statusregister ab.

- **Senden:** Warten bis TXRDY im US_CSR gesetzt ist, dann Daten in US_THR schreiben .
- **Empfangen:** Warten bis RXRDY im US_CSR gesetzt ist, dann Daten aus US_RHR lesen .

DMA (Direct Memory Access)

Für größere Datenmengen wird der PDC genutzt. Dies geschieht über Pointer- und Counter-Register:

- **Pointer (US_TPR/US_RPR):** Startadresse des Datenpuffers im Speicher.
- **Counter (US_TCR/US_RCR):** Anzahl der zu übertragenden Bytes.
- Die Hardware dekrementiert den Counter automatisch. Wenn er 0 erreicht, wird ein Interrupt (ENDTX bzw. ENDRX) ausgelöst .

6. Test-Modi

Zur Fehlersuche bietet das US_MR Register spezielle Loopback-Modi:

- **Automatic Echo:** Empfangene Daten werden direkt zurückgesendet.
- **Local Loopback:** Sender ist intern direkt mit dem Empfänger verbunden (gut für Softwaretests ohne externe Hardware).
- **Remote Loopback:** RXD Pin ist intern direkt mit TXD Pin verbunden.

06

Timer

1. Einleitung und Komponenten

Timer und Counter sind wesentliche Peripheriekomponenten, die in praktisch jedem modernen Mikrocontroller integriert sind (On-Chip). Sie ermöglichen zeitliche Operationen unabhängig vom laufenden Programm der CPU (paralleles Verhalten).

- **Funktionsweise:**
    - **Timer (Zeitgeber):** Erzeugt Signale oder Interrupts in genau definierten Zeitperioden (z.B. für Betriebssystem-Scheduler).
    - **Counter (Zähler):** Misst die Zeit zwischen Signalen oder zählt externe Ereignisse.
- **Hardware:** Enthält Register und konfigurierbare Logik, um Bedingungen zu prüfen (z.B. Zählerwert entspricht Registerwert) und Aktionen in Hardware auszulösen, ohne dass Software eingreifen muss.

2. Basiskomponenten

- **Counter/Zählregister:** Wird durch einen Takt kontinuierlich inkrementiert. Der Takt wird meist vom Systemtakt abgeleitet. Das Register hat eine feste Breite (z.B. 16 oder 32 Bit). Bei Erreichen des Maximalwerts erfolgt ein Überlauf .
- **Äquivalenz von Zähler und Zeit:** Da die Taktfrequenz bekannt ist, entspricht der Zählerstand direkt einer Zeitdauer (Periodendauer T = 1 / f).

3. Flexibilisierung und Konfiguration

Um den Timer an verschiedene Zeitintervalle anzupassen, gibt es zwei Hauptmechanismen:

- **Reset-Register (RC):** Der Zähler wird mit einem Register (RC) verglichen. Bei Gleichheit (Counter == RC) wird der Zähler auf 0 zurückgesetzt. Dies definiert die Periodendauer T_A unabhängig von der Bitbreite des Zählers $(T_A = RC/f_{CLK})$.
- **Prescaler (Vorteiler):** Der Systemtakt wird durch einen Teiler (z.B. 2, 8, 32, 1024) dividiert, bevor er den Zähler erreicht. Dies verlangsamt das Zählen und ermöglicht so die Messung oder Erzeugung längerer Zeiträume .
- **Kombination:** $f_A = \frac{f_{CLK}}{Prescaler \cdot RC}$.

4. Betriebsmodi

A. Compare Mode (Periodische Interrupts)

- **Ziel:** Aufgaben in regelmäßigen Abständen ausführen (z.B. OS-Scheduler).
- **Mechanismus:** Der Zähler wird mit einem Register (RC) verglichen. Bei Übereinstimmung wird ein Interrupt ausgelöst und der Zähler zurückgesetzt .

B. Waveform Mode (Signalerzeugung)

- **Ziel:** Erzeugung von Signalen an Ausgängen (z.B. für Motoren).
- **Mechanismus:** Nutzt Vergleichsregister (RA, RC).
    - Übereinstimmung mit RA Ausgang auf Low/High setzen.
    - Übereinstimmung mit RC Ausgang auf High/Low setzen und Zähler-Reset.
- **Anwendung - PWM (Pulsweitenmodulation):** Steuerung des Verhältnisses von High- zu Low-Zeit (Tastverhältnis).
    - **Dimmen von LEDs:** High-Anteil bestimmt Helligkeit.
    - **Motorsteuerung:** High-Anteil bestimmt Drehmoment/Geschwindigkeit .
    - **Servosteuerung:** Länge des High-Impulses (z.B. 1,5ms in 20ms Periode) bestimmt die Position .

C. Capture Mode (Messung)

- **Ziel:** Messen von Zeitdauern oder Zählen von Ereignissen.
- **Mechanismus:** Speichert den aktuellen Zählerstand in Registern (RA, RB) bei externen Ereignissen (z.B. steigende/fallende Flanke).
- **Anwendung:** Messung der Impulsbreite eines Ultraschallsensors zur Entfernungsbestimmung $(Distanz = \frac{340m/s}{Zeit})$.

5. Erweiterte Konzepte

- **Kaskadierung:** Hintereinanderschalten von Zählern, um den Zählbereich zu erweitern (z.B. Sekunden Minuten Stunden) .
- **Entprellen:** Timer können genutzt werden, um mechanische Taster per Software-Zeitmessung zu entprellen.
- **Hardware-Vergleich:**
    - **AT91:** Zwei Blöcke mit je drei 16-Bit Kanälen.

- **SAM3X:** Ähnlich AT91, aber mit 32-Bit Kanälen.
- **RP2040:** 8 PWM-Slices (16-Bit), ein globaler 64-Bit Mikrosekunden-Timer.

06a

Timer auf AT91 und SAM3X Mikrocontrollern

1. Architektur der Timer/Counter-Blöcke

Die Timer/Counter (T/C)-Blöcke sind bei beiden Atmel-Familien (AT91 und SAM3X) prinzipiell ähnlich aufgebaut, unterscheiden sich jedoch in Details wie der Registerbreite.

- **Struktur:** Jeder Block enthält **drei unabhängige Channels**, die jeweils einen eigenen Timer bilden.
- **Registerbreite:**
  - **AT91:** 16-Bit Zähler.
  - **SAM3X (Arduino Due):** 32-Bit Zähler .
- **Kaskadierung:** Channels können verbunden werden, um Zähler mit größerer Bitbreite zu realisieren (z.B. für Uhren: Sekunden -> Minuten -> Stunden) .

2. Taktversorgung (Clock Selection)

Jeder Channel kann eine eigene Taktquelle wählen. Dies geschieht über Multiplexer, die den Systemtakt (Master Clock, MCK) über verschiedene Teiler (Prescaler) zuführen.

- **Prescaler:**
  - **AT91:** MCK geteilt durch 2, 8, 32, 128, 1024.
  - **SAM3X:** MCK geteilt durch 2, 8, 32, 128 (hier TIMER_CLOCK1 bis TIMER_CLOCK4 genannt) .
- **Externe Takte:** Zusätzlich können externe Signale (XC0, XC1, XC2) als Taktquelle dienen .

3. Betriebsmodi

A. Waveform Mode (Signalerzeugung)

Dient zur Erzeugung von Signalen (z.B. PWM) an den Ausgängen TIOA und TIOB.

- **Vergleichsregister:** Es gibt drei Vergleichsregister: **RA, RB, RC**.
- **Funktionsweise:** Der Zähler (Counter) wird kontinuierlich inkrementiert.

    - Bei Übereinstimmung mit RA/RB/RC können die Ausgänge gesetzt, gelöscht oder getoggelt werden.
    - RC dient oft als Periodendauer: Bei Counter == RC wird der Zähler zurückgesetzt (CPCTRG gesetzt).
- **Beispiel PWM:** Eine PWM mit 30% Duty Cycle an TIOA und 50% an TIOB kann durch Setzen von RC (Periode), RA (30% von RC) und RB (50% von RC) realisiert werden. Der Zähler wird bei RC zurückgesetzt, TIOA/TIOB bei 0 gesetzt und bei RA/RB gelöscht .

B. Capture Mode (Messung)

Dient zur Messung von externen Signalen an den Eingängen TIOA und TIOB.

- **Funktionsweise:** Externe Ereignisse (Flankenwechsel) laden den aktuellen Zählerstand in die Register **RA** und **RB** (LDRA, LDRB).
- **Anwendung:** Messung von Impulsdauern oder Periodendauern.
    - Beispiel: Messung der Periodendauer T_A. Start bei steigender Flanke (Reset), Laden von RB bei nächster steigender Flanke. $T_A = RB \cdot T_{Clock}$
- **Besonderheit:** Der Zähler kann gestoppt werden, wenn ein Register geladen wurde (LDBSTOP), um den Messwert zu sichern.

4. Register und Konfiguration

Die Steuerung erfolgt über diverse Register. Wichtige Beispiele:

- **Channel Mode Register (TC_CMR):** Wählt den Modus (Wave/Capture), den Takt (Prescaler) und die Aktionen bei Vergleichen (Set/Clear/Toggle) .
- **Channel Control Register (TC_CCR):** Ermöglicht Software-Trigger (SWTRG) zum Starten/Resetten und das Aktivieren/Deaktivieren des Takts (CLKEN/CLKDIS) .
- **Status Register (TC_SR):** Zeigt Ereignisse an, z.B. ob RA/RB geladen wurden (LDRAS/LDRBS) oder ein Überlauf (COVFS) stattfand .

5. Periodendauer und Überlauf

Die maximal messbare Zeit oder generierbare Periode hängt von der Zählerbreite und dem Takt ab.

- **AT91 (16-Bit):** Bei 33 MHz und Teiler 1024 erfolgt ein Überlauf nach ca. 2 Sekunden.
- **SAM3X (32-Bit):** Bei 84 MHz (Arduino Due) und Teiler 128 dauert es bis zum Überlauf über 6500 Sekunden (ca. 1,8 Stunden) .

07

Interrupt Handling

1. Ereignisgesteuerte Programmierung

Moderne Programme, insbesondere in eingebetteten Systemen, sind oft **ereignisgesteuert**. Das bedeutet, der Ablauf wird primär durch externe Ereignisse (z.B. Tastendruck, Sensorwerte, Kommunikation) bestimmt und nicht nur durch interne Daten .

- **Typische Ereignisquellen:** Tastatur/Maus, Schnittstellen (USB, Ethernet, I2C, SPI), Timer, Sensoren .

2. Mechanismen der Ereignisverarbeitung

Es gibt verschiedene Ansätze, wie ein System auf Ereignisse reagieren kann:

- **Busy Waiting:** Die einfachste Form. Der Prozessor wartet in einer Endlosschleife aktiv auf das Eintreten eines Ereignisses. Er ist blockiert und kann nichts anderes tun.
- **Polling:** Der Prozessor fragt Ereignisquellen zyklisch (z.B. in einer Schleife oder timergesteuert) ab.
    - *Vorteil:* Einfach zu implementieren; deterministisch.
    - *Nachteil:* Gefahr, Ereignisse zu verpassen; Verschwendung von Rechenzeit durch ständiges Abfragen ("Sind wir schon da?") .
- **Interrupt (Unterbrechung):** Das Ereignis löst ein Hardware-Signal aus, das den Prozessor unterbricht ("Klingel").
    - *Vorteil:* Echte Nebenläufigkeit; Prozessor kann andere Aufgaben erledigen oder schlafen (Sleep-Mode), bis das Ereignis eintritt; bessere Programmstruktur .
    - *Nachteil:* Komplexer (Kontextwechsel nötig); Gefahr von Race Conditions (Shared Data Problem).

3. Funktionsweise von Interrupts

Ein Interrupt unterbricht den aktuellen Befehlsstrom asynchron.

1. **Anforderung:** Ein Peripheriebaustein sendet einen *Interrupt Request* an den Interrupt Controller.
2. **Unterbrechung:** Wenn der Interrupt erlaubt und priorisiert ist, unterbricht die CPU das laufende Programm.

3. **Kontextwechsel:** Der aktuelle Zustand (Register, Program Counter, Statusregister) wird gesichert (auf dem Stack oder in speziellen Registern), damit das Programm später nahtlos fortgesetzt werden kann .
4. **ISR:** Die *Interrupt Service Routine* (ISR) wird ausgeführt, um das Ereignis zu behandeln.
5. **Rücksprung:** Der gesicherte Kontext wird wiederhergestellt und das ursprüngliche Programm fortgesetzt .

4. Interrupt Controller

Um die CPU zu entlasten, verwalten spezielle Hardware-Bausteine (**Interrupt Controller**) die verschiedenen Interrupt-Quellen und deren Prioritäten.

- **AT91/SAM3X:** Nutzen den *Advanced Interrupt Controller (AIC)* (AT91) bzw. *Nested Vector Interrupt Controller (NVIC)* (SAM3X/Cortex-M). Der NVIC unterstützt verschachtelte Interrupts und automatische Kontextsicherung effizient.
- **Priorisierung:** Wichtig, um bei gleichzeitigen Ereignissen das wichtigere zuerst zu bearbeiten (z.B. kritische Fehler vor Tastenanschlägen) .

5. Direct Memory Access (DMA)

Für sehr schnelle Datenübertragungen (z.B. Gigabit Ethernet) sind reine Software-Interrupts oft zu langsam.

- **Lösung:** Ein DMA-Controller übernimmt den Datentransfer zwischen Peripherie und Speicher selbstständig.
- **Rolle des Interrupts:** Der Interrupt dient hier nur noch zur Ablaufkontrolle (z.B. Meldung "Transfer abgeschlossen"), nicht mehr für jedes einzelne Datenbyte .

6. Kritische Aspekte

- **Shared Data Problem:** Wenn Hauptprogramm und ISR auf dieselben Daten zugreifen, kann es zu Inkonsistenzen kommen. Kritische Abschnitte müssen geschützt werden (z.B. durch kurzzeitiges Sperren von Interrupts) .
- **Reaktionszeit:** Die Latenz bis zum Start der ISR hängt von der aktuellen Tätigkeit der CPU und der Kontextwechsel-Dauer ab.

07a

Interrupts auf dem AT91 (ARMv4/ARM7TDMI)

1. Advanced Interrupt Controller (AIC)

Der AT91 verwendet den **Advanced Interrupt Controller (AIC)**, um Interrupts zu verwalten.

- **Aufgabe:** Verwaltet interne und externe Interrupt-Quellen, priorisiert diese und leitet sie als **IRQ** (Normal Interrupt) oder **FIQ** (Fast Interrupt) an den ARM-Core weiter .
- **Quellen:** Es gibt 32 Interrupt-Quellen (ID 0-31), darunter Timer, USART, PIO (Parallel I/O) und externe IRQ-Pins. Quelle 0 ist reserviert für FIQ, Quelle 1 für den System-Timer (SWIRQ).

2. Register des AIC

Der AIC wird über Memory Mapped I/O gesteuert (Basisadresse 0xFFFFF000). Wichtige Register sind:

- **SMR (Source Mode Register):** Konfiguriert den Trigger-Typ (Flanke/Pegel) und die Priorität (0-7) für jede Quelle .
- **SVR (Source Vector Register):** Enthält die Adresse der Interrupt Service Routine (ISR) für die jeweilige Quelle.
- **IVR (IRQ Vector Register):** Enthält die Adresse der ISR des aktuell aktiven Interrupts (wird vom AIC automatisch geladen).
- **EOICR (End of Interrupt Command Register):** Muss am Ende der ISR beschrieben werden, um dem AIC den Abschluss zu signalisieren.
- **IECR/IDCR:** Interrupt Enable/Disable Command Register (zum Aktivieren/Deaktivieren von Interrupts).

3. Ablauf einer Interrupt-Behandlung

Wenn ein Interrupt auftritt, passiert Folgendes:

1. **Hardware-Ablauf:**
    - Die CPU beendet den aktuellen Befehl.
    - Der Modus wird gewechselt (z.B. User -> IRQ).
    - Der aktuelle PC wird im Link Register (LR_irq = PC - 4) gesichert.
    - Das Statusregister (CPSR) wird im SPSR gesichert.

    - Die CPU springt zur Vektoradresse (0x18 für IRQ) .
2. **Vektortabelle:** An Adresse 0x18 steht ein Sprungbefehl, der den PC mit dem Inhalt des AIC_IVR lädt. Damit springt die CPU direkt zur spezifischen ISR .
3. **Software-Ablauf (ISR):**
    - Register retten (Kontext sichern).
    - Interrupt bearbeiten.
    - Interrupt im AIC bestätigen (Schreiben in EOICR) .
    - Kontext wiederherstellen.
    - Rücksprung mit SUBS pc, lr, #4 (stellt gleichzeitig PC und CPSR wieder her).

4. Statusregister (CPSR)

Das **Current Program Status Register (CPSR)** enthält wichtige Informationen:

- **Flag-Bits:** N, Z, C, V (Ergebnisse arithmetischer Operationen).
- **I/F-Bits:** Sperren (Maskieren) von IRQ (I=1) bzw. FIQ (F=1).
- **Mode-Bits:** Zeigen den aktuellen Prozessormodus an (User, IRQ, FIQ, Supervisor, Abort, Undefined, System) .
- **Zugriff:** Nur über spezielle Befehle MSR (Register -> Status) und MRS (Status -> Register) modifizierbar .

5. Nested / Reentrant Interrupts

Standardmäßig sind Interrupts auf dem ARM7 nicht verschachtelt (nested), da beim Eintritt in den IRQ-Modus das I-Bit im CPSR gesetzt wird. Um einen Interrupt durch einen höher priorisierten unterbrechbar zu machen (**Reentrant**), muss die ISR:

1. In den **System-Mode** wechseln.
2. Das LR (Rücksprungadresse) und SPSR auf dem Stack sichern.
3. Interrupts wieder freigeben (I-Bit löschen) .

6. Spurious Interrupts

Ein "Spurious Interrupt" (Geister-Interrupt) kann auftreten, wenn eine Interrupt-Anforderung verschwindet, bevor die CPU sie verarbeiten kann (z.B. durch Glitches oder Software-Löschung). Der AIC erkennt dies und liefert stattdessen die Adresse aus dem **Spurious Vector Register (SPU)**. Eine ISR dafür ist zwingend erforderlich, um das System nicht abstürzen zu lassen .

07b

Interrupt Handling auf dem SAM3X (Cortex-M3)

1. Nested Vectored Interrupt Controller (NVIC)

Der SAM3X (basierend auf der ARM Cortex-M3 Architektur) verwendet den **NVIC** (Nested Vectored Interrupt Controller), der direkt in den Prozessorkern integriert ist. Im Gegensatz zum AT91 (ARM7) ist keine externe Hardware nötig.

- **Eigenschaften:**
    - 30 externe Interrupt-Quellen (IRQ0...IRQ29).
    - Unterstützt verschachtelte (nested) Interrupts: Ein höher priorisierter Interrupt kann einen niedrigeren unterbrechen.
    - 16 Prioritätsstufen (0 = höchste, 15 = niedrigste) .
    - Konfigurierbar über **CMSIS** (Cortex Microcontroller Software Interface Standard).

2. Ausnahmen (Exceptions)

Der Cortex-M3 unterscheidet verschiedene Arten von Ausnahmen, die alle über die Vektortabelle verwaltet werden:

- **System Exceptions:** Reset, NMI (Non Maskable Interrupt), Hard Fault, Memory Management Fault, Bus Fault, Usage Fault, SVCall, PendSV, SysTick .
- **Interrupts (IRQs):** Asynchrone Ausnahmen, die von Peripheriegeräten (z.B. Timer, USART) ausgelöst werden. Der Chiphersteller definiert die Anzahl (beim SAM3X bis zu 30) .

3. Vektortabelle

Die Vektortabelle enthält die Startadressen der Exception-Handler (ISRs).

- **Position:** Standardmäßig an Adresse 0x00000000. Kann über das Register VTOR (Vector Table Offset Register) verschoben werden.
- **Inhalt:**
    - Eintrag: Initialer Stack Pointer (MSP).
    - Folgende Einträge: Startadressen für Reset, NMI, Hard Fault, ..., IRQ0, IRQ1, etc.
- **Besonderheit:** Das niederwertigste Bit jeder Adresse muss 1 sein, um den Thumb-Modus zu signalisieren .

4. Priorisierung und Gruppierung

- **Präemption:** Ein Interrupt höherer Priorität verdrängt einen laufenden Interrupt niedrigerer Priorität. Gleiche Prioritäten verdrängen sich nicht.
- **Sub-Prioritäten:** Prioritäten können in Gruppen eingeteilt werden (Group Priority vs. Sub-Priority). Nur die Gruppenpriorität entscheidet über Präemption (Unterbrechung), die Sub-Priorität entscheidet, welcher von zwei gleichzeitig anstehenden Interrupts zuerst bearbeitet wird .

5. Optimierte Interrupt-Verarbeitung

Der Cortex-M3 bietet Hardware-Features für sehr schnelle Interrupt-Reaktionen:

- **Automatisches Stacking:** Beim Eintritt in eine ISR werden Register (R0-R3, R12, LR, PC, PSR) automatisch auf den Stack gesichert (Push) .
- **Tail-Chaining:** Wenn am Ende einer ISR ein weiterer Interrupt ansteht, wird *kein* Kontextwechsel (Pop/Push) durchgeführt. Der Prozessor springt direkt in die nächste ISR. Das spart Zyklen .
- **Late-Arrival:** Wenn während des Sicherns (Stacking) für einen Interrupt ein *höher* priorisierter Interrupt auftritt, wird direkt zu diesem gewechselt (der bereits begonnene Stacking-Vorgang wird genutzt) .

6. Programmierung (CMSIS)

Die Programmierung erfolgt über standardisierte C-Funktionen aus dem CMSIS-Layer:

- NVIC_EnableIRQ(IRQn): Aktiviert einen Interrupt.
- NVIC_DisableIRQ(IRQn): Deaktiviert ihn.
- NVIC_SetPriority(IRQn, priority): Setzt die Priorität.
- NVIC_ClearPendingIRQ(IRQn): Löscht den Pending-Status.

Beispiel für das Aktivieren eines Timer-Interrupts:

NVIC_EnableIRQ(TC8_IRQn); // Aktiviert den Interrupt für Timer 8

08

Signalverarbeitung im Kontext von Eingebetteten Systemen:

1. Grundlagen der Signalverarbeitung

Mikrocontroller interagieren über Sensoren und Aktuatoren mit der physischen Welt. Dabei spielen Analog-Digital- (ADC) und Digital-Analog-Wandler (DAC) eine zentrale

Rolle, um zwischen der analogen Außenwelt und der digitalen Verarbeitung zu vermitteln .

Die Signalverarbeitung beschäftigt sich mit analogen Größen wie:

- **Messgrößen:** Temperatur, Abstand, Geschwindigkeit.
- **Audiosignalen:** Sprache, Musik.
- **Stellgrößen:** Motordrehzahl, Servowinkel .

2. Zeit- und Frequenzbereich

Signale können auf zwei Arten dargestellt und analysiert werden:

- **Zeitbereich:** Darstellung der Amplitude über der Zeit (z. B. Oszilloskop). Zeigt die direkte Veränderung einer Größe .
- **Frequenzbereich (Spektralbereich):** Darstellung der Amplitude über der Frequenz. Dies ist besonders für periodische Signale nützlich .

Mathematischer Hintergrund:

- **Fourier-Transformation:** Jedes periodische Signal kann in eine Summe von Sinus- und Kosinusfunktionen (Grundfrequenz und Obertöne) zerlegt werden. Dies ermöglicht die Umwandlung vom Zeit- in den Frequenzbereich .
- Reale Signale sind meist Mischungen vieler Frequenzen und besitzen eine bestimmte Bandbreite .

3. Abtastung (Sampling)

Um analoge Signale digital zu verarbeiten, müssen sie abgetastet werden.

- **Abtasttheorem (Nyquist-Shannon):** Die Abtastfrequenz f_s muss mehr als doppelt so hoch sein wie die höchste im Signal vorkommende Frequenz f_{max} $(f_s > 2 \cdot f_{max})$, um das Signal korrekt rekonstruieren zu können .
- **Aliasing:** Wird das Abtasttheorem verletzt (zu langsame Abtastung), entstehen "Geisterfrequenzen" (Aliasing), und das ursprüngliche Signal kann nicht mehr korrekt wiederhergestellt werden .
- **Oversampling (Überabtastung):** Abtastung mit einem Vielfachen der notwendigen Frequenz. Dies erhöht den Signal-Rausch-Abstand (SNR) und vereinfacht analoge Filter am Eingang (Anti-Aliasing-Filter) .

Filterung:

- **Mittelwertbildung:** Durch Oversampling und anschließende Mittelwertbildung kann das Rauschen reduziert und die effektive Auflösung erhöht werden. Dies wirkt wie ein Tiefpassfilter .
- **Gleitender Mittelwert (Moving Average):** Ein kontinuierlich berechneter Mittelwert über die letzten n Werte glättet das Signal und filtert hohe Frequenzanteile (Rauschen) heraus, reagiert aber verzögert auf schnelle Änderungen .

4. Analog-Digital-Wandlung (ADC)

Ein ADC wandelt eine kontinuierliche Eingangsspannung in ein diskretes digitales Wort um.

- **Ablauf:** Das Signal wird zu einem Zeitpunkt abgetastet und für die Dauer der Wandlung gehalten (Sample & Hold) .
- **Quantisierung:** Die Spannung wird in Stufen unterteilt. Dabei entsteht zwangsläufig ein Quantisierungsfehler (max. 1/2 LSB).
- **Auflösung:** Die kleinste erfassbare Änderung beträgt U/2^n (bei Bit Wortbreite und Spannungsbereich U). Beispiel: 10 Bit bei 10V ergibt ca. 9,8 mV Auflösung .

Wandler-Verfahren:

- **Zählverfahren:** Ein Zähler erzeugt über einen DAC eine Treppenspannung, bis diese die Eingangsspannung erreicht (einfach, aber langsam und abhängig von der Spannungshöhe) .
- **Wägeverfahren (Successive Approximation):** Die Bits werden beginnend beim MSB (Most Significant Bit) nacheinander gesetzt und geprüft (ähnlich einer Balkenwaage). Dies ist effizienter als das Zählverfahren .

5. Digital-Analog-Wandlung (DAC)

Ein DAC wandelt binäre Wörter zurück in analoge Spannungen (z. B. für Regelventile).

- **Direktes Verfahren:** Auswahl einer Spannung über einen Multiplexer (schnell, aber sehr aufwendig bei vielen Bits) .
- **R2R-Netzwerk (Paralleles Verfahren):** Ein Netzwerk aus Widerständen (nur Werte R und 2R) erzeugt durch Schalterstellung der Bits die entsprechende Ausgangsspannung .

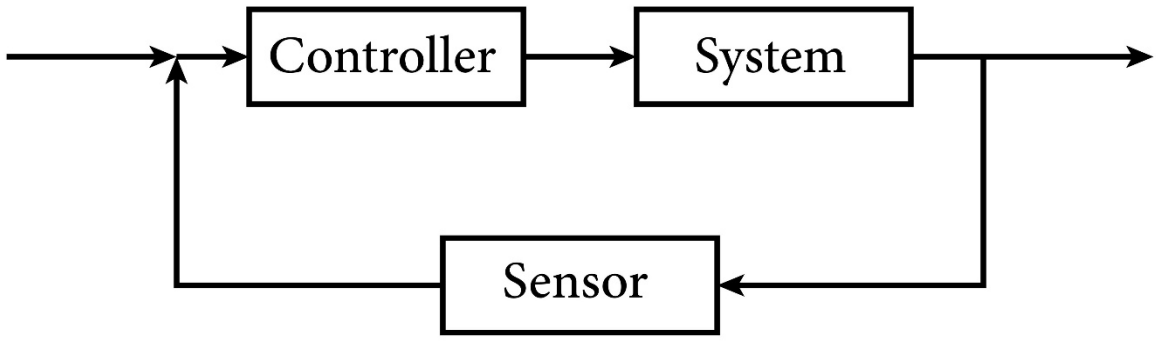
09

Regelungen in eingebetteten Systemen.

1. Grundlagen und Regelkreis

Regelungen sind zentral für eingebettete Systeme, die mit ihrer physischen Umgebung interagieren.

- **Prinzip:** Entscheidungen werden basierend auf Feedback (**Regelgröße**) getroffen. Ein Sensor misst den Ist-Zustand (z.B. Temperatur, Geschwindigkeit), der mit einem Soll-Zustand verglichen wird.
- **Der Regelkreis:**
    - **Regler:** Berechnet aus der Regeldifferenz $e(t) = w(t) - y(t)$ (Soll - Ist) eine Stellgröße u(t).
    - **Stellglied (Aktuator):** Setzt die Stellgröße physikalisch um (z.B. Heizung, Motor).
    - **Regelstrecke (Prozess):** Das physikalische System, das beeinflusst wird.
    - **Störgröße:** Externe Einflüsse, die den Prozess verändern (z.B. offenes Fenster beim Heizen).
- **Abgrenzung:** Eine **Steuerung** (Open Loop) hat kein Feedback und kann auf Störungen nicht reagieren. Eine **Regelung** (Closed Loop) misst das Ergebnis und korrigiert Fehler.



Shutterstock

2. Reglertypen

Zweipunkt- und Mehrpunktregler

Der einfachste Regler, der nur zwei Zustände kennt (Ein/Aus, z.B. Bügeleisen).

- **Problem:** Schnelles Schalten um den Sollwert.
- **Lösung:** Einsatz einer **Hysterese** (Schmitt-Trigger). Es wird erst bei Unterschreiten einer unteren Schwelle eingeschaltet und erst bei Überschreiten einer oberen Schwelle ausgeschaltet.

PID-Regler und seine Komponenten

Der PID-Regler ist der Standard in der Technik und kombiniert drei Verhaltensweisen:

1. **P-Anteil (Proportional):** Die Stellgröße ist proportional zum Fehler $(u(t) = K_p \cdot e(t))$.
    - *Vorteil:* Reagiert sofort.
    - *Nachteil:* Hat oft eine **bleibende Regelabweichung** (erreicht den Sollwert nie ganz), da bei Fehler 0 auch die Stellgröße 0 wäre (außer bei integrierenden Strecken).
2. **I-Anteil (Integral):** Summiert den Fehler über die Zeit auf.
    - *Vorteil:* Eliminiert die bleibende Regelabweichung vollständig (hohe stationäre Genauigkeit).
    - *Nachteil:* Reagiert langsamer und neigt zum Schwingen.
3. **D-Anteil (Differential):** Reagiert auf die Änderungsgeschwindigkeit des Fehlers.
    - *Vorteil:* Wirkt dämpfend und kann zukünftige Fehler "vorhersehen".
    - *Nachteil:* Verstärkt Rauschen extrem; wird daher fast nie allein eingesetzt.

- **PID-Formel:** $u(t) = K_p \cdot e(t) + K_i \int e(\tau)d\tau + K_d \frac{de(t)}{dt}$.

3. Implementierung in Software

Da Mikrocontroller zeitdiskret arbeiten, müssen die mathematischen Modelle angepasst werden.

- **Diskretisierung:** Integrale werden zu Summen, Differentiale zu Differenzenquotienten basierend auf der Abtastzeit T.

- **Windup-Effekt:** Wenn das Stellglied am Anschlag ist (Sättigung, z.B. Ventil 100% offen), wächst der I-Anteil theoretisch weiter ins Unendliche ("Windup"). Kehrt sich der Fehler um, dauert es lange, bis der I-Anteil wieder sinkt.
    - *Lösung:* **Anti-Windup**-Strategien, z.B. Begrenzung des I-Anteils (Clamping).

4. Echtzeit-Aspekte

Für eine stabile Regelung ist ein deterministisches Zeitverhalten (konstantes Abtastintervall T) essenziell.

- **Jitter:** Schwankungen in der Abtastzeit verschlechtern die Regelgüte.
- **Polling vs. Interrupt:** Polling (Busy Waiting) ist oft nicht deterministisch genug (z.B. durch andere Aufgaben blockiert). Die Implementierung über **Timer-Interrupts** wird für harte Echtzeitanforderungen empfohlen.

5. Hybride Ansätze

Um Rechenleistung zu sparen, können komplexe Regler durch **Kennfelder** (Look-up Tables) angenähert werden. Statt die PID-Formel zu berechnen, wird die Stellgröße basierend auf Fehler und Fehleränderung direkt aus einer Tabelle ausgelesen.

10

Software Interrupt (SWI), spezifisch für die ARM-Architektur.

1. Konzept und Zweck

Ein Software Interrupt (ausgelöst durch den Assemblerbefehl SWI) dient als Schnittstelle zwischen Anwendungsprogrammen und dem Betriebssystem. Er ermöglicht einen kontrollierten Wechsel vom normalen, eingeschränkten Benutzermodus (**User Mode**) in den privilegierten **Supervisor Mode**. Dies wird genutzt, um Systemdienste (System Calls) wie z.B. putchar oder getchar aufzurufen.

2. Ablauf der Ausführung (Hardware-Seitig)

Sobald der Befehl SWI ausgeführt wird, führt die CPU automatisch eine festgelegte Sequenz aus, um den Kontext zu wechseln:

1. **Status sichern:** Der Inhalt des aktuellen Statusregisters (CPSR) wird in das Saved Program Status Register des Supervisor-Modus (SPSR_SVC) kopiert. Damit bleiben Flags und der ursprüngliche Modus erhalten.

2. **Moduswechsel:** Die CPU schaltet in den **Supervisor Mode**. Dadurch werden automatisch der Stack Pointer (SP_svc) und das Link Register (LR_svc) dieses Modus aktiv.
3. **Interrupts sperren:** Das IRQ-Disable-Bit im CPSR wird gesetzt. Dies verhindert, dass der SWI-Handler sofort durch normale Hardware-Interrupts unterbrochen wird.
4. **Rücksprungadresse sichern:** Die Adresse des Befehls nach dem SWI (PC - 4) wird im Link Register (LR_svc) gespeichert, damit das Programm später dort fortgesetzt werden kann.
5. **Sprung zum Vektor:** Der Program Counter (PC) wird auf die feste Adresse **0x08** gesetzt. Dort liegt der Einsprungpunkt für den SWI-Handler (Exception Vector).

3. Der SWI-Handler (Software-Seitig)

Der Code, der an der Adresse 0x08 startet (der Handler), muss die Anfrage verarbeiten.

- **SWI-Nummer extrahieren:** Der SWI-Befehl enthält eine Nummer (z.B. SWI 0x01), die angibt, welche Funktion gewünscht ist. Diese Nummer steht nicht in einem Register, sondern im Maschinencode des Befehls selbst.
    - Der Handler lädt den Befehl, der an der Adresse [LR - 4] steht, in ein Register.
    - Die oberen 8 Bits (der Opcode für "SWI") werden mit einer Bitmaske (BIC ... #0xff000000) entfernt. Die verbleibenden 24 Bits sind die gesuchte SWI-Nummer.
- **Verzweigung:** Anhand dieser Nummer entscheidet der Handler (z.B. über eine Sprungtabelle in Assembler oder ein switch-Statement in C), welche Systemfunktion ausgeführt wird.
- **Rücksprung:** Nach Abarbeitung wird der ursprüngliche Zustand (User Mode) wiederhergestellt, indem SPSR zurück ins CPSR und LR in den PC kopiert wird.

10a

Software Interrupt (SWI), spezifisch für die ARM-Architektur.

1. Konzept und Zweck

Ein Software Interrupt (ausgelöst durch den Assemblerbefehl SWI) dient als Schnittstelle zwischen Anwendungsprogrammen und dem Betriebssystem. Er ermöglicht einen kontrollierten Wechsel vom normalen, eingeschränkten Benutzermodus (**User Mode**) in den privilegierten **Supervisor Mode**. Dies wird genutzt, um Systemdienste (System Calls) wie z.B. putchar oder getchar aufzurufen.

2. Ablauf der Ausführung (Hardware-Seitig)

Sobald der Befehl SWI ausgeführt wird, führt die CPU automatisch eine festgelegte Sequenz aus, um den Kontext zu wechseln:

1. **Status sichern:** Der Inhalt des aktuellen Statusregisters (CPSR) wird in das Saved Program Status Register des Supervisor-Modus (SPSR_SVC) kopiert. Damit bleiben Flags und der ursprüngliche Modus erhalten.
2. **Moduswechsel:** Die CPU schaltet in den **Supervisor Mode**. Dadurch werden automatisch der Stack Pointer (SP_svc) und das Link Register (LR_svc) dieses Modus aktiv.
3. **Interrupts sperren:** Das IRQ-Disable-Bit im CPSR wird gesetzt. Dies verhindert, dass der SWI-Handler sofort durch normale Hardware-Interrupts unterbrochen wird.
4. **Rücksprungadresse sichern:** Die Adresse des Befehls nach dem SWI (PC - 4) wird im Link Register (LR_svc) gespeichert, damit das Programm später dort fortgesetzt werden kann.
5. **Sprung zum Vektor:** Der Program Counter (PC) wird auf die feste Adresse **0x08** gesetzt. Dort liegt der Einsprungpunkt für den SWI-Handler (Exception Vector).

3. Der SWI-Handler (Software-Seitig)

Der Code, der an der Adresse 0x08 startet (der Handler), muss die Anfrage verarbeiten.

- **SWI-Nummer extrahieren:** Der SWI-Befehl enthält eine Nummer (z.B. SWI 0x01), die angibt, welche Funktion gewünscht ist. Diese Nummer steht nicht in einem Register, sondern im Maschinencode des Befehls selbst.
    - Der Handler lädt den Befehl, der an der Adresse [LR - 4] steht, in ein Register.
    - Die oberen 8 Bits (der Opcode für "SWI") werden mit einer Bitmaske (BIC ... #0xff000000) entfernt. Die verbleibenden 24 Bits sind die gesuchte SWI-Nummer.
- **Verzweigung:** Anhand dieser Nummer entscheidet der Handler (z.B. über eine Sprungtabelle in Assembler oder ein switch-Statement in C), welche Systemfunktion ausgeführt wird.
- **Rücksprung:** Nach Abarbeitung wird der ursprüngliche Zustand (User Mode) wiederhergestellt, indem SPSR zurück ins CPSR und LR in den PC kopiert wird.

Leitungscodes (Line Codes) im Bereich Eingebettete Systeme.

1. Einordnung (OSI-Schicht 1)

Leitungscodes gehören zur **Bitübertragungsschicht** (Physical Layer) des OSI-Modells.

- **Aufgabe:** Definition der physikalischen Verbindung (Stecker, Leitungen), der physikalischen Signale (elektrisch, optisch) und der **Codierung** von Bitfolgen in übertragbare Signale .
- **Pegel:** Es wird unterschieden zwischen **High-active** (High=1, Low=0) und **Low-active** (Low=1, High=0).
- **Probleme einfacher Codierungen:** Bei langen Folgen von Nullen oder Einsen finden keine Pegelwechsel statt. Dadurch können Fehler (z. B. Leitungsbruch) schwer erkannt werden und der Empfänger kann die Synchronisation verlieren .

2. Binäre Leitungscodes (NRZ-Familie)

Diese Codes nutzen zwei Pegelzustände.

- **NRZ (Non Return to Zero):** Der Pegel repräsentiert direkt das Bit (0=Low, 1=High). Standard bei asynchroner Übertragung (z. B. RS-232), hat aber Synchronisationsprobleme bei langen identischen Folgen .
- **NRZI (NRZ Invert):** Die Information steckt im **Pegelwechsel**, nicht im Pegel selbst.
    - **NRZI-S (Space):** Pegelwechsel bei einer **0**, kein Wechsel bei einer **1**. (Wird z. B. bei USB verwendet).
    - **NRZI-M (Mark):** Pegelwechsel bei einer **1**, kein Wechsel bei einer **0**.
    - *Vorteil:* Bessere Synchronisation, aber immer noch problematisch bei langen Folgen ohne Wechsel (z. B. viele 1en bei NRZI-S) .

3. Ternäre Leitungscodes (RZ)

Der **RZ-Code (Return to Zero)** nutzt drei Pegel (-1, 0, 1).

- **Funktionsweise:** Eine logische 1 ist ein positiver Pegel, eine 0 ein negativer Pegel. Nach der Hälfte der Taktzeit kehrt das Signal immer auf **0 Volt** zurück.
- **Vorteil:** Das Signal ist **selbsttaktend**, da in jedem Bit ein Flankenwechsel stattfindet.

- **Nachteil:** Die benötigte Bandbreite verdoppelt sich (Baudrate = 2 * Bitrate). Zudem enthält das Signal einen Gleichspannungsanteil, was eine galvanische Trennung erschwert .

4. Selbsttaktende Binärcodes (Manchester)

Der **Manchester-Code** kombiniert Takt- und Dateninformation.

- **Prinzip:** Ein Bit wird durch eine **Flanke** in der Mitte des Taktintervalls definiert.
    - **IEEE 802.3 (Ethernet):** Fallende Flanke = **0**, Steigende Flanke = **1**.
- **Eigenschaften:**
    - Signal ist gleichspannungsfrei (Mittelwert ist 0).
    - Selbsttaktend (einfache Synchronisation).
    - Kodierung/Dekodierung einfach per XOR-Verknüpfung möglich.
    - **Nachteil:** Benötigt die doppelte Bandbreite im Vergleich zu NRZ .

5. Baudrate vs. Bitrate

Das Skript unterscheidet klar zwischen diesen beiden Größen:

- **Bitrate:** Anzahl der übertragenen Informationen (Bits) pro Sekunde.
- **Baudrate:** Anzahl der übertragenen Symbole (Pegelzustände auf der Leitung) pro Sekunde.
- **Vergleich:**
    - Beim **NRZI-S** entspricht 1 Symbol genau 1 Bit Baudrate = Bitrate.
    - Beim **Manchester-Code** (und RZ) besteht 1 Bit aus 2 Symbolen (z. B. High dann Low) **Baudrate = 2 * Bitrate** .

11a

Sicherung der Datenübertragung (Fehlererkennung).

1. Einfache Sicherungsverfahren

Um Übertragungsfehler zu erkennen, werden redundante Informationen hinzugefügt.

- **VRC (Vertical Redundancy Checking):** Querparität (Paritätsbit). Erkennt 1-Bit-Fehler, versagt aber, wenn sich Daten- und Prüfbit so ändern, dass die Parität wieder stimmt.
- **LRC (Longitudinal Redundancy Checking):** Längsparität. Hier wird über die Spalten eines Datenblocks ein Paritätsbit gebildet.

- **Checksum (Summenprüfung):** Bytes eines Blocks werden als Zahlen addiert (oft Modulo 256). Das Zweierkomplement der Summe wird als Prüfsumme (FCS) angehängt. Der Empfänger addiert alles inklusive FCS; das Ergebnis muss 0 sein.

2. Cyclic Redundancy Check (CRC)

CRC ist ein leistungsfähigeres Verfahren, das auf Polynomdivision basiert.

- **Mathematischer Hintergrund:** Bitfolgen werden als Koeffizienten von Polynomen interpretiert. Gerechnet wird in der **Modulo-2-Arithmetik** (Dualsystem ohne Übertrag).
    - Addition und Subtraktion entsprechen dabei der **XOR**-Verknüpfung (1+1 = 0,0-1=1).
- **Ablauf:**
    1. Die Daten werden um den Grad des Generatorpolynoms (Stellen) nach links verschoben (mit Nullen aufgefüllt).
    2. Diese Folge wird durch das Generatorpolynom geteilt.
    3. Der **Rest** der Division ist die Prüfsumme (FCS), die an die Originaldaten angehängt wird.
- **Prüfung:** Der Empfänger teilt die empfangene Nachricht (Daten + FCS) durch dasselbe Generatorpolynom. Ist der Rest **0**, war die Übertragung fehlerfrei.

3. Hardware-Implementierung (LFSR)

CRC lässt sich in Hardware sehr effizient mittels **Schieberegistern** realisieren.

- **Linear Feedback Shift Register (LFSR):** Besteht aus einer Kette von Flipflops (Schieberegister) und XOR-Gattern.
- **Funktion:** Die Rückkopplungen im Register entsprechen den Koeffizienten des Generatorpolynoms. Mit jedem Takt wird ein Bit verarbeitet, am Ende steht der Rest (CRC) im Register.

4. Gängige Standards

Die Qualität der Fehlererkennung hängt vom gewählten Generatorpolynom ab.

- **CRC-5:** USB Token.
- **CRC-16:** USB-Daten.
- **CRC-CCITT:** Bluetooth, X.25.
- **CRC-32:** Ethernet (IEEE 802.3), Internet, USB 3.0.

5. Hamming-Distanz

Die Hamming-Distanz ist ein Maß für die Unterschiedlichkeit von Code-Wörtern.

- **Definition:** Die Anzahl der Bit-Positionen, in denen sich zwei Wörter unterscheiden. Berechnet durch XOR-Verknüpfung und Zählen der Einsen.
- **Bedeutung für Fehlererkennung:**
    - Um Fehler sicher zu **erkennen**, muss gelten: $d \geq e + 1.$
    - Um Fehler zu **korrigieren**, muss gelten: $d \geq 2e + 1.$

12

Weitere Themen (Thumb, Systemstart, Assembler) im Modul Eingebettete Systeme.

1. Thumb-Befehlssatz

Der Thumb-Modus ist ein 16-Bit-Befehlssatz, der entwickelt wurde, um die Codedichte zu verbessern.

- **Eigenschaften:**
    - Befehle sind 16 Bit breit, was ca. 65% der Codegröße im Vergleich zum 32-Bit ARM-Code entspricht .
    - Der Prozessor verfügt über einen zusätzlichen Ausführungszustand (T-Bit im CPSR gesetzt:), der Strom umschaltet (z. B. durch den BX-Befehl) .
    - **Vorteile:** Höhere Codedichte, energieeffizienter bei externem Speicher, schneller auf 16-Bit-Speichersystemen .
    - **Nachteile:** Geringere Ausführungsgeschwindigkeit auf 32-Bit-Speichern (ca. 40% langsamer), eingeschränkter Registerzugriff (meist nur R0–R7), keine bedingte Ausführung (außer Branches) .
- **Hardware-Umsetzung:** Ein "Thumb Decompressor" in der Pipeline wandelt die 16-Bit-Befehle zur Laufzeit in native ARM-Befehle um .
- **Thumb-2:** Eine Weiterentwicklung (z. B. im Cortex-M3), die 16- und 32-Bit-Befehle mischt. Dies kombiniert die Dichte von Thumb mit der Leistung von ARM ohne explizite Umschaltung .

2. Systemstart und Remapping

Beim Einschalten (Power Up) startet der Prozessor an Adresse 0x00000000.

- **Problem:** Zum Start muss dort nicht-flüchtiger Speicher (ROM/Flash) liegen. Im laufenden Betrieb benötigt man dort jedoch oft RAM (für die flexible Interrupt-Vektortabelle) .
- **Lösung (Remapping):** Ein Memory Controller (z. B. EBI beim AT91) blendet nach dem Start den RAM-Bereich über den ROM-Bereich an Adresse 0 ein .
- **Boot-Ablauf (AT91):**
    1. Interrupt Controller (AIC) initialisieren.
    2. Vektortabelle vom Flash ins interne RAM kopieren.
    3. **Remapping** durchführen (RAM an Adresse 0 sichtbar machen).
    4. Stacks und Stackpointer für verschiedene Modi definieren.
    5. Variablen initialisieren (BSS).
    6. Sprung in die main()-Routine .

3. Assembler-Programmierung

Die Vorlesung diskutiert das Für und Wider der Assembler-Nutzung in modernen Projekten.

- **Gründe für Assembler:**
    - Notwendig für tiefste Systemroutinen (z. B. Context Switch im OS, Startup-Code).
    - Direkter Hardwarezugriff und Inbetriebnahme.
    - Zeitkritische Interrupt-Routinen oder extreme Optimierung.
    - Verständnis von Compiler-Output und Fehlersuche .
- **Gründe dagegen:**
    - Geringe Produktivität (Faktor 5–10 schlechter als Hochsprachen) und hohe Kosten .
    - Schlechte Lesbarkeit, Wartbarkeit und Portierbarkeit (prozessorspezifisch) .
- **Relevanz:** In großen Projekten (z. B. Linux Kernel, Open Office) macht Assembler nur noch einen verschwindend geringen Anteil aus (< 3% bzw. 0%).

4. Ausblick: Mikrocontroller-Landschaft

Die Programmierung erfordert Bewusstsein für Ressourcen (Speicher, Energie). Wichtige Plattformen sind:

- **Hobby/Maker:** Arduino, Raspberry Pi, ESP32.
- **Industrie/Automotive:** Infineon Aurix, NXP, Renesas.
- **Trends:** IoT (Low-Energy, Wireless) und Embedded AI (NPU-Integration) .