Mikrocontroller - Tipps & Tricks

Mikrocontroller vs. CPU

- CPU
  - ”alles” RAM, viel RAM
  - Keine On-Chip Peripherie
  - Viele Chips, Motherboard
- Mikrocontroller
  - Wenig RAM, Flash im Chip mit drin
  - Peripherie an Board (Timer, ADC, UART, etc)
  - Ein Chip
  - Software läuft aus dem Flash, sofort...

Mikrocontroller - Tipps & Tricks

EEPROM vs FLASH

- EEPROM
  - Byteweise löschen / programmieren
  - Einfache Schnittstelle (schreiben = programmieren)
- Flash
  - Sektorweise löschen, wortweise schreiben
  - Flashbefehle über Adreß / Datenbus, Status zurück

Mikrocontroller - Tipps & Tricks

FPGA vs CPLD

- FPGA
  - RAM-basiert,
  - Viel RAM, wenig Logikgatter, undefiniertes Timing, Look-up-tables im RAM
  - Boot-ROM notwendig
- CPLD
  - Flashbasiert, kein Boot-ROM
  - Viele Logikgatter, schnelles wohldefiniertes Timing
  - Wenig Speichermöglichkeiten

Mikrocontroller - Tipps & Tricks

Timer/Counter

- ”Zählen” von Ereignissen
  - Flanken
  - Spezialfall ”Quadruppel” Eingang (Odometrie)
- ”Messen” von Zeit
  - Capture Eingang
    - Freuenz messen
    - Pulsweite messen
  - Compare Ausgang
    - PWM ausgeben
    - Puls, definierter Länge ausgeben

Timer / Counter

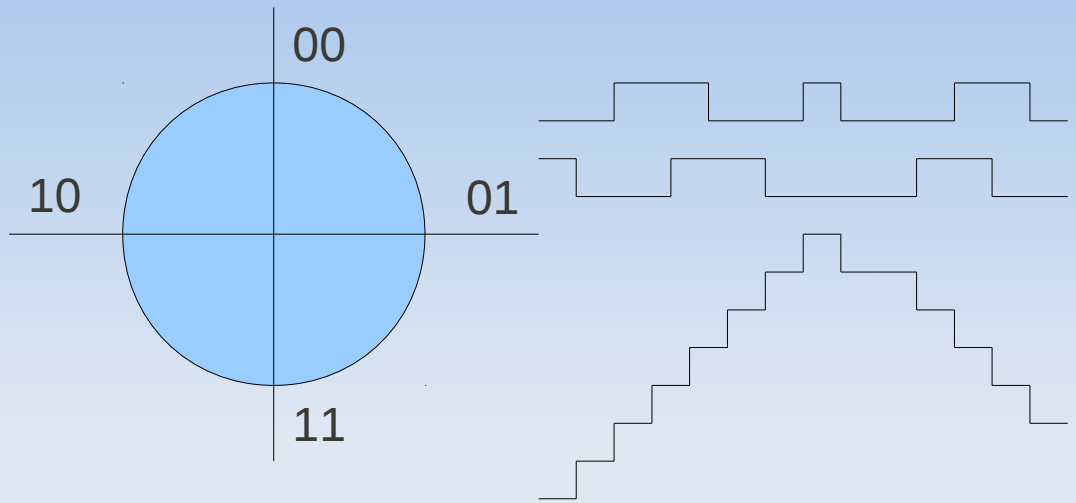
Eingang

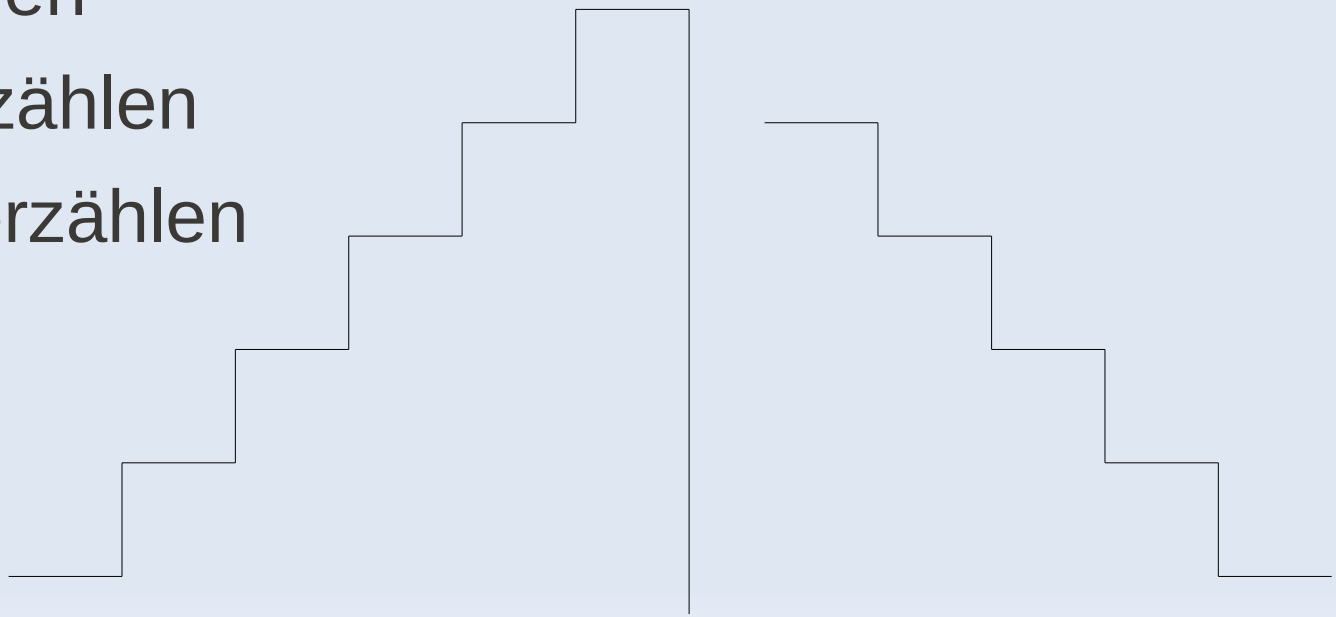
- Zähler
  - Steigende Flanken
  - fallende Flanken
  - Beide Flanken
- Vorteiler
  - CPU-Takt / 2er-Potenz (8 ,32 , 128)
  - Takt durch variablen Timer (anderer Timerüberlauf)

Timer / Counter

Timer Register

- 8, 16 oder 32 Bit
- Reload-Register
- Manuel laden
- Löschen
- Hochzählen
- Runterzählen





Timer / Counter

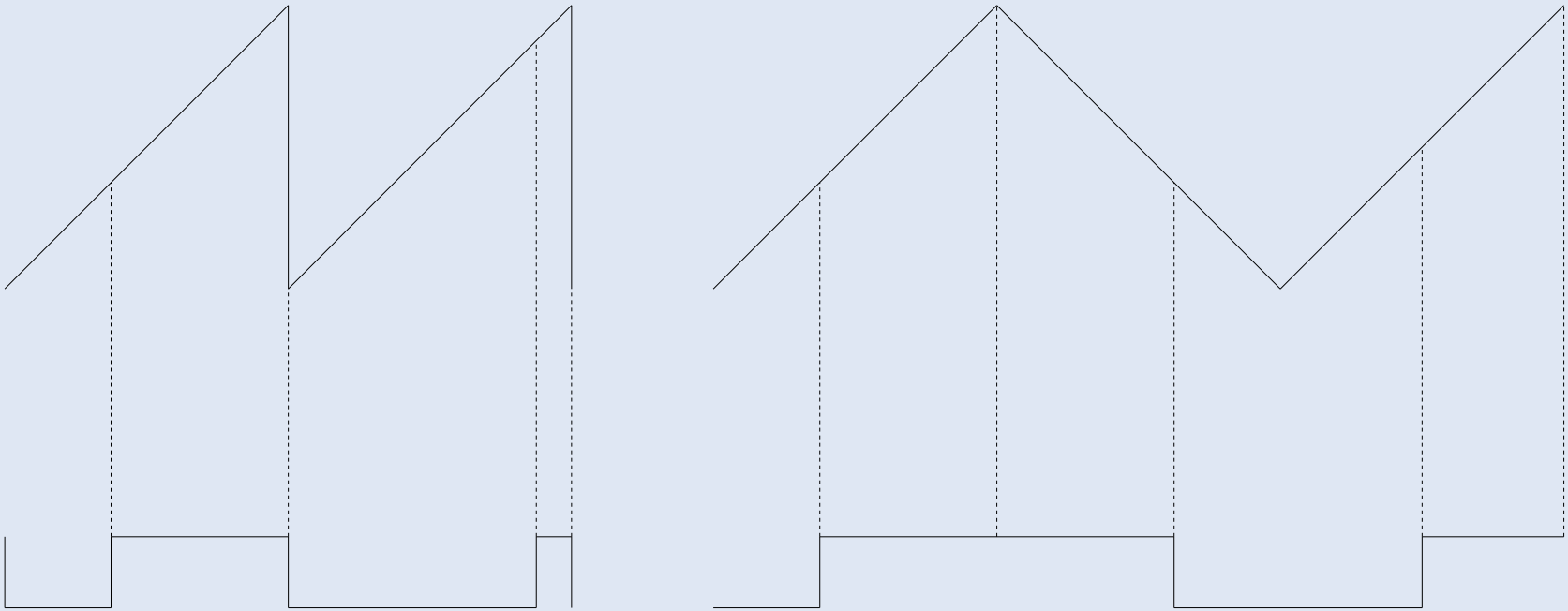
Ereignisse

- Interrupts
  - Überlauf (Raster Aufruf, Timeout)
  - Compare (Compare-Update)
  - Capture (Capture-Wert abholen)
  - Start A/D-Wandler etc.
- Ausgaben
  - Pin togglen, setzen oder löschen (Pulse, PWM, Frequenzausgabe)

Timer / Counter

PWM ausgeben (z.B. Dimmer, Motor)

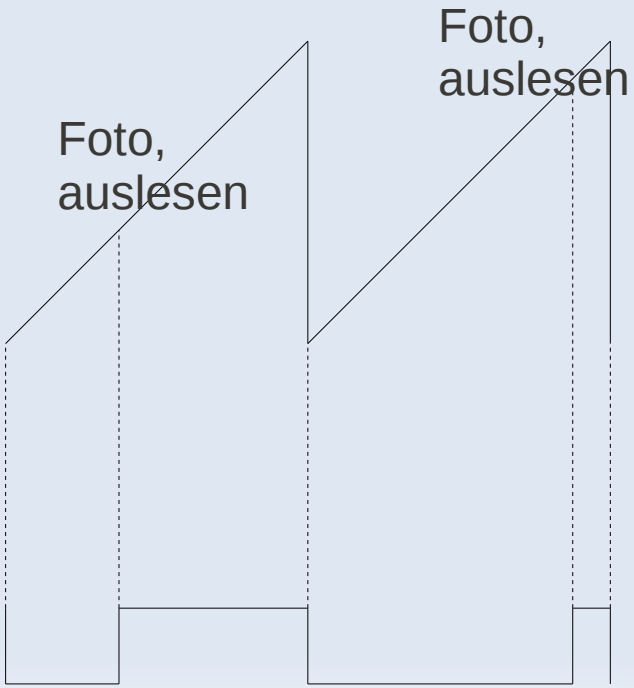
- Timerüberlauf = Frequenz
- Compare-Wert = Tastverhältnis



Timer / Counter

PWM messen

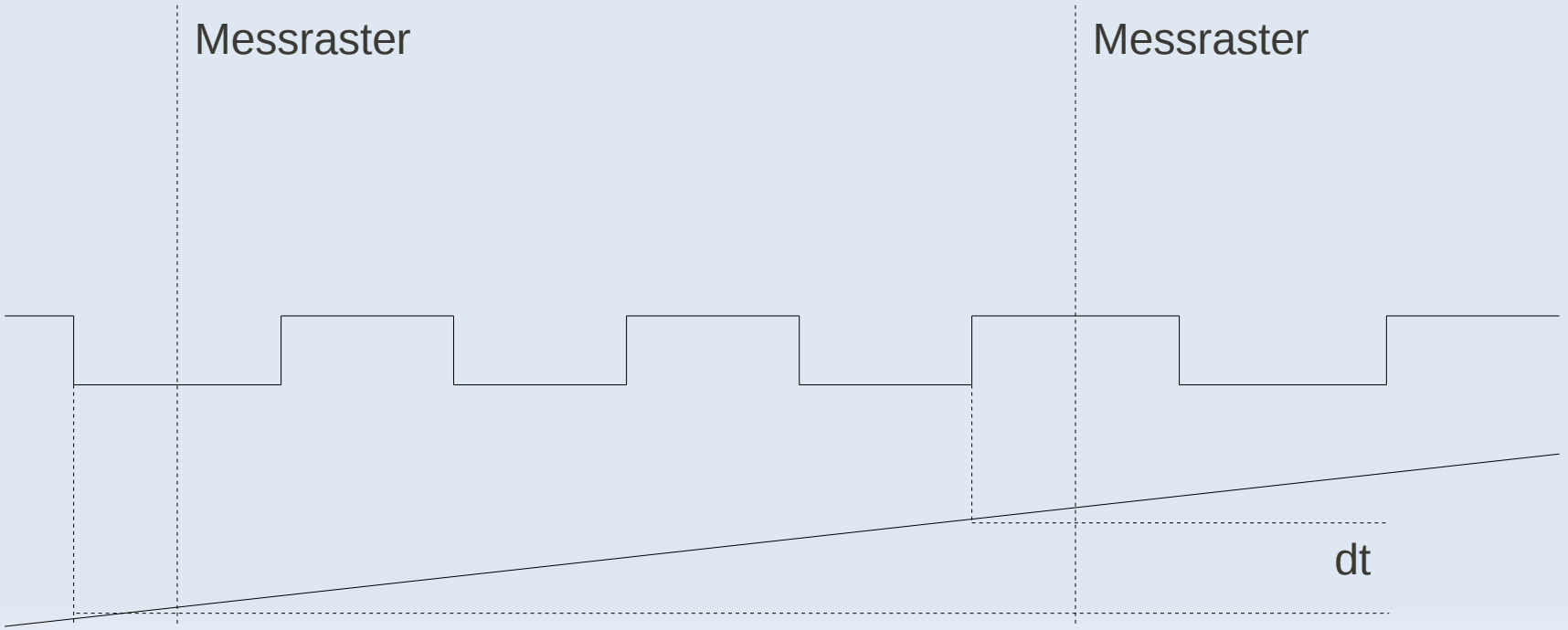
- Capture-Event ”fotografiert” Timer
- z.B. Timer Reset bei fallender Flanke, Foto, auslesen bei steigender Flanke



Timer / Counter

Frequenzmessung (Radgeschwindigkeit)

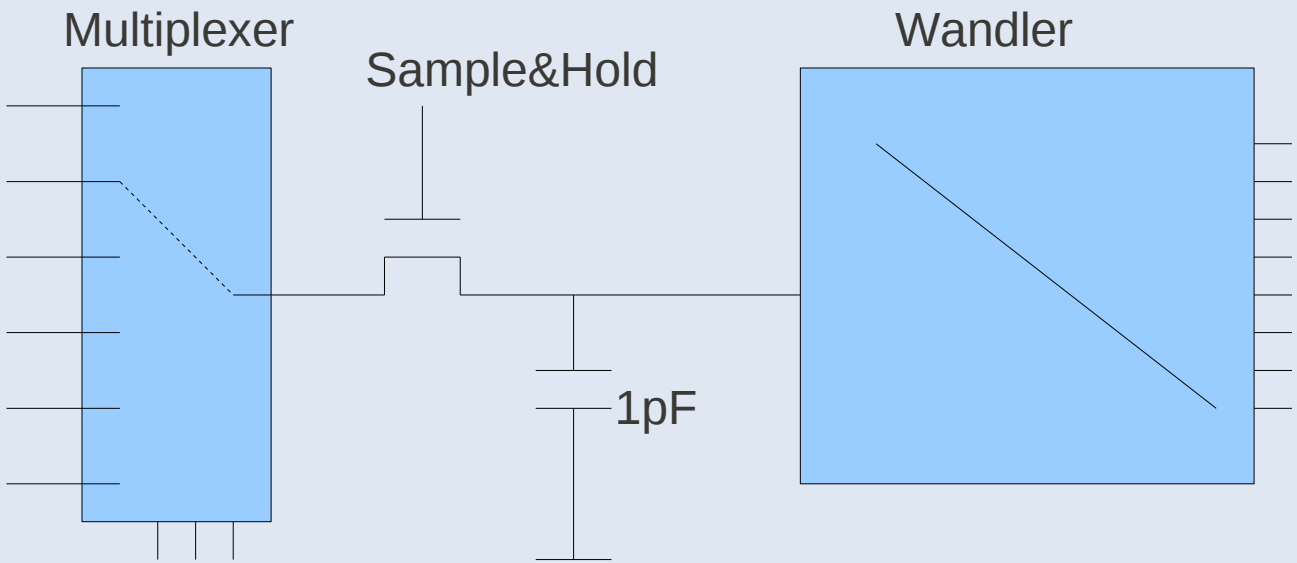
- ”Durchschnitt” vom Messraster
  - Anzahl der Flanken / dt (Differenz der zuletzt gemessenen Zeitpunkte : Capture) (... / 2)



A/D - Wandler

ADC

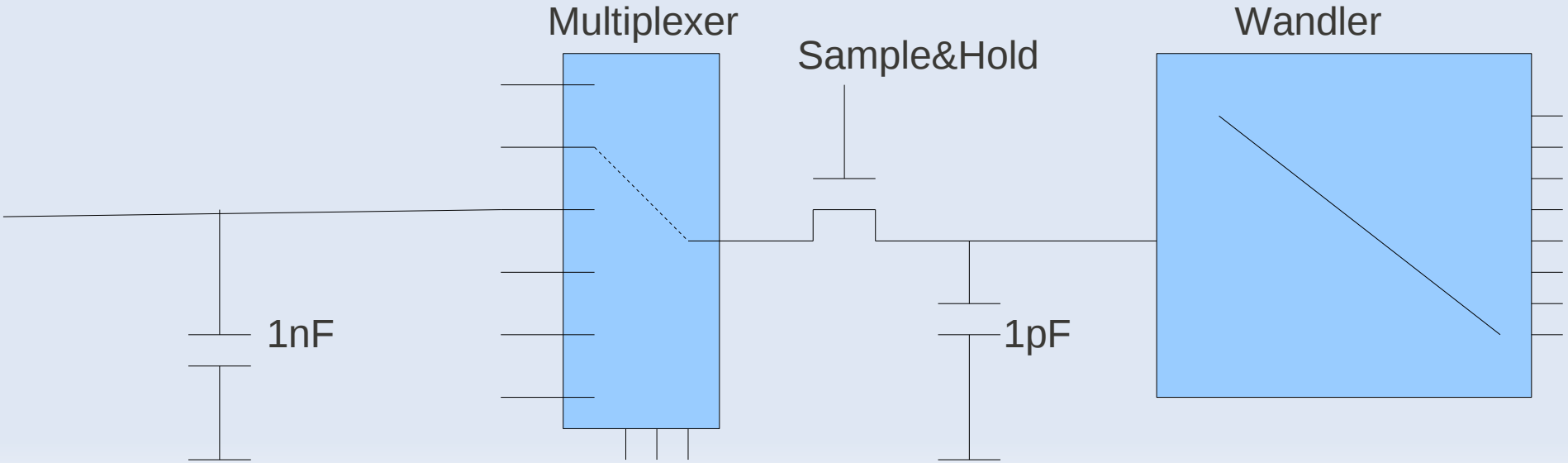
- Analog-Multiplexer
- Sample & hold
- Wandler



A/D - Wandler

Anforderungen am Eingang

- Signal muß niederohmig sein, wegen schneller Laden des ”Hold”-Kondensator
- z.B. Kondensator (1nF) direkt am A/D-Eingang (Umladen 1nF => 1pF,
  Fehler max. 1/1000)



A/D - Wandler

ADC – Abtasten - Theorie

- Filter mit Grenzfrequenz < halbe Abstastfrequenz
- Sonst Aliasing
- RC-Glied, PT1-Filter einfach aber schlecht
- Oversampling 5-10 fach
- Shannon, Nyquist

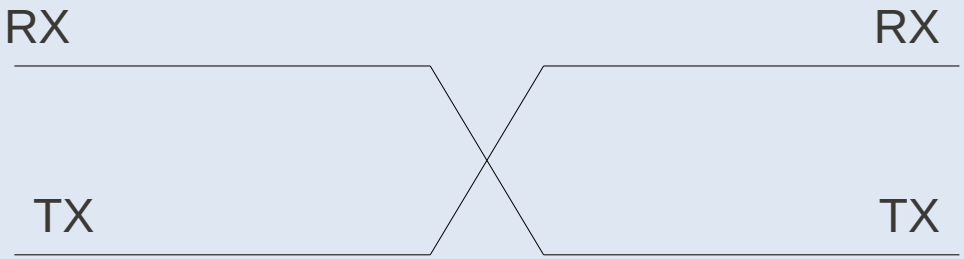
Mikrocontroller - Tipps & Tricks

Schnittstellen

- U(S)ART (seriell)
- I2C (TWI)
- SPI
- CAN

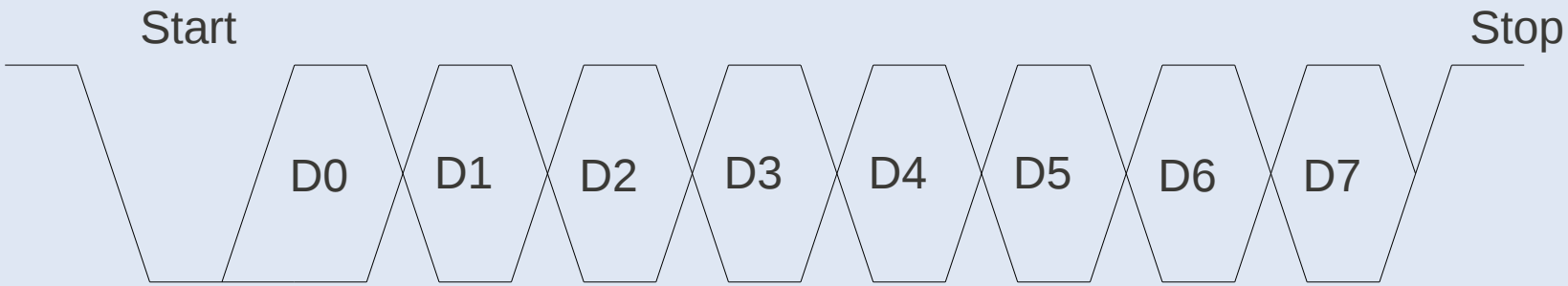
UART (SCI)

- Baudrate auf beiden Seiten bekannt, gleich
- Baudratenteiler vom CPU-Takt abgeleitet, Fehler < 5%
- Kreuzung TX – RX, bidirektionale Kommunikation
- Bis 115200 Baud



UART (SCI)

- 1 Startbit (Low), 7-8 Daten Bits, (Parität), 0-2 Stoppbits (High)
- Baudraten: 9600, 19200, 38400, 115200
- Beispiel: 8n1, 8 Datenbits, keine Parität, 1 Stoppbit



UART (SCI)

Nachteil:

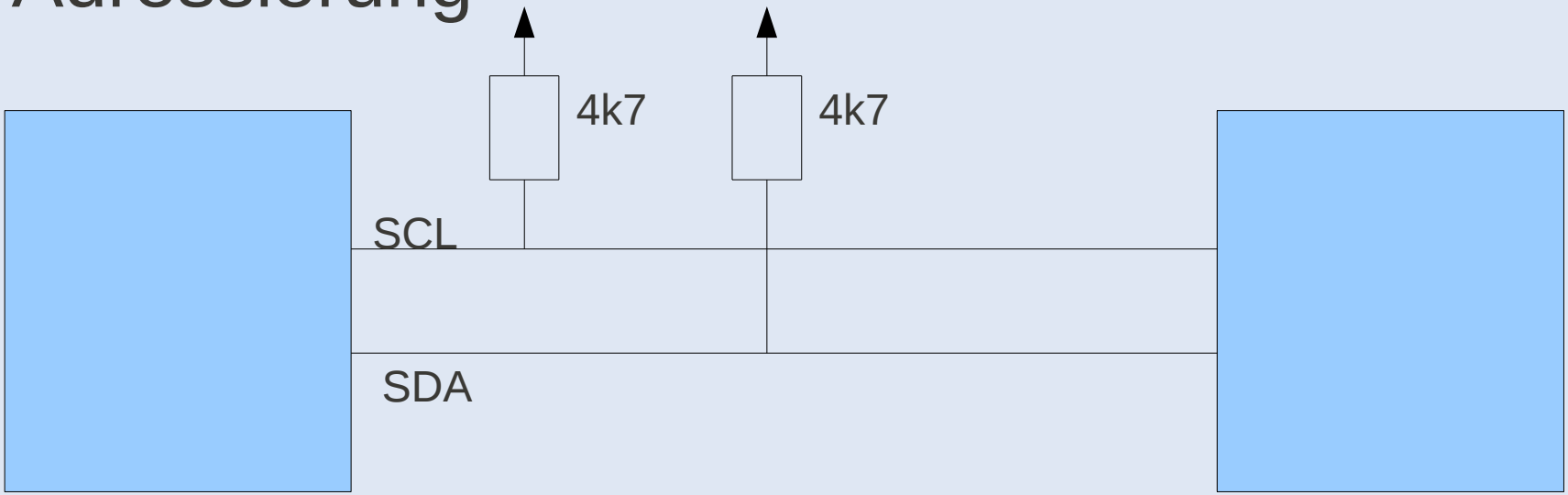
- Anforderung an den Takt (Quartz)
- Hoher HW-Aufwand im Controller
- Kein Bus... i.d.R. Punkt zu Punkt

Vorteil

- Frameüberwachung möglich
- Einfach und sicher
- Direkt an PC ist kein Problem
- Weit verbreitet, einfache Verdrahtung

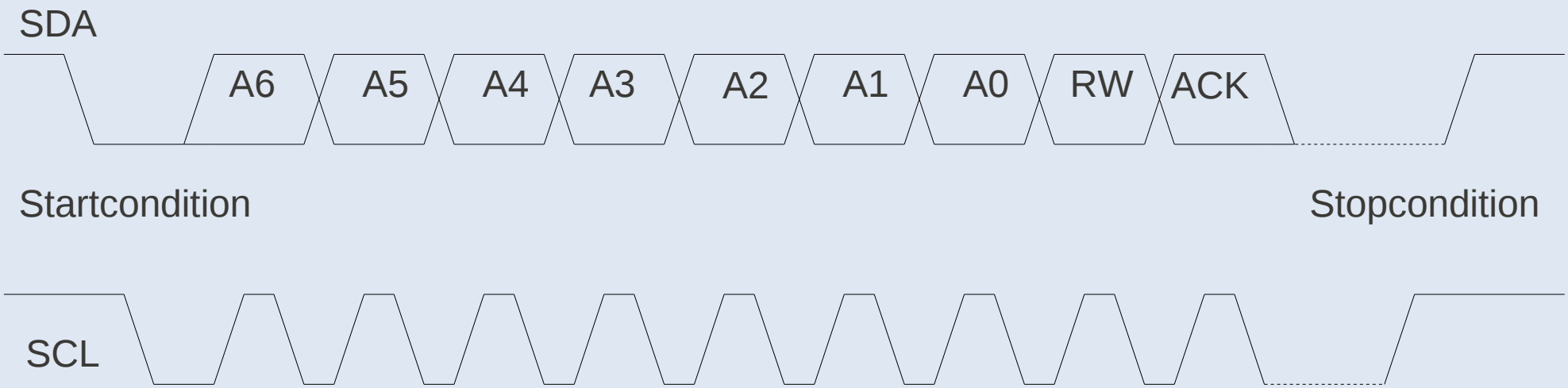
I2C (TWI)

- Baudrate auf SCL vorgegeben
- Framing, Start-Condition, Stop-Condition Acknowledge
- Bus von je 2 Open-Collector Aus-/Eingängen
- Adressierung



I2C (TWI)

- Startcondition, 7 Adress-Bits, R/W-Bit, Acknowlegde, Daten, Stopcondition



I2C (TWI)

Verwendung

- Sensoren (Naviboard2...)
- AD-Wandler
- Speicher (EEPROM)
- Latches
- etc.

I2C (TWI)

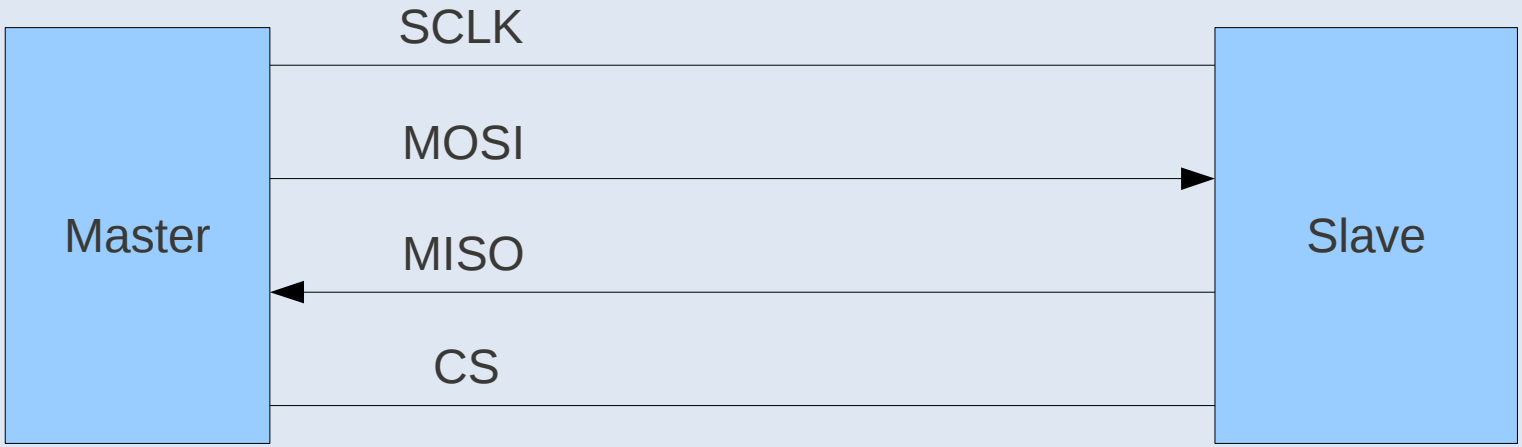
Nachteil:

- Störanfällig
- ungwollte Start/Stopconditions
- Auf Lowziehen eines Teilnehmers legt den Bus lahm (Freitakten auch nicht möglich wenn SCL auf low)

- Vorteil
  - Bus, Acknowlegdes, Überwachbarkeit
  - Geringer HW Aufwand
  - Multimaster, Adressierung sehr flexibel
  - Weit verbreitet, einfache Verdrahtung

SPI

- Takt kommt vom Master
- CS, zum Aktivieren/Deaktivieren des Slaves
- Gleichzeitige bidirektionale Kommunikation
- Prinzip Shift-Register, extrem einfache Controller HW



SPI

- Start mit Chipselect

SPI

Nachteile

- Störanfällig auf der SCLK-Leitung (kurz halten!)
- Klare Master/Slave Einteilung
- CS notwendig per Slave

Vorteile

- Billig in HW
- CS kann genutzt werden um Slave zu deaktivieren
- Kann teilweise sehr hoch getaktet werden
- Weit verbreitet

SPI

Verwendung

- SD / MMC-Karte
- Sensoren
- Speicher
- Latches
- AVR-ISP (in system programming)
- etc.

CAN

CAN

- Entwickelt von Bosch (1987)
- 2 Draht Bus, Differenzsignal
- Automotiv, robust
- Alle hören alles (jedes Bit)
- Länge < 32m, Baudrate <=1MBaud
- Eingebaute Diagnose, mit heilen, und abschalten
- Abschlusswiderstände 120 Ohm an jeder Seite (2x)

CAN

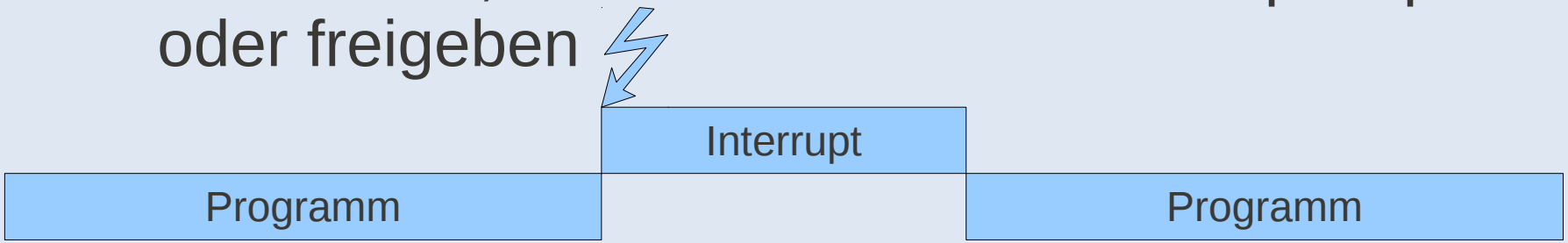
CAN

- Message-Konzept:
  - CAN Id (11 oder 29 Bit)
  - Bis zu 8 Byte Daten
  - CRC16 Checksumme
  - Acknowlegde
  - Alle dürfen meckern...
- Message Filterung
- Mailbox-Konzept

Interrupts

Interrupts

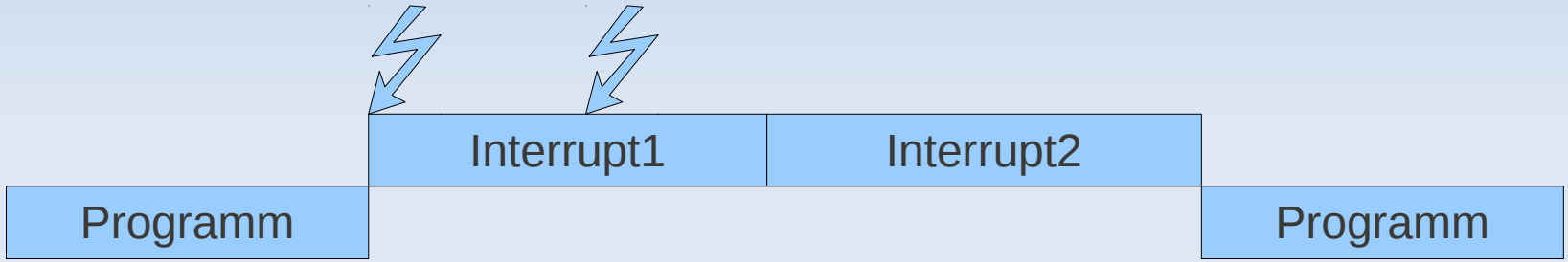
- Unterbrechung des ”normalen” Programmablaufs, auf ein Ereignis (HW oder SW) - IR-Vektor
- Sichern aller gebrauchten Register
- Ausführen der Routine
- Zurückschreiben der Sicherung, Rücksprung
- Andere Regeln während der Interruptausführung, kurz halten, keine Routinen die Interrupts sperren oder freigeben



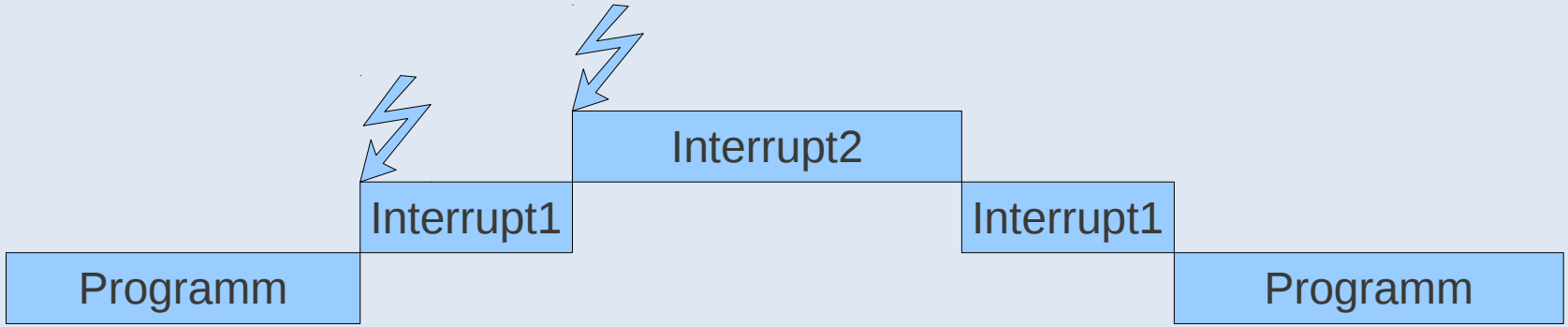
Interrupts

Nesting

- Interrupts können auch während Interrupts kommen



Interrupts können Interrupts unterbrechen bei höherer Priorität - Nesting



Interrupts

Nesting - Techniken

- Software (z.B. AVR)
  - Sperren aller Interrupts
  - Sperren nieder priorer Features
  - Interupts freigeben, am Ende restaurieren
- Hardware (z.B. Cortex-M, C167)
  - Prozessor(IR-Controller) hat im eigenen Status die aktuelle Priorität auf der gerade ausgeführt wird
  - Nur höhere Prios werden sofort ausgeführt, andere hinten angestellt

DMA

Reduktion von Interrupts durch DMA

- Abholen von n AD-Werten in eine Tabelle
- Empfang von n Bytes über SCI/SPI/I2C
- Sendem von n Bytes über SCI/SPI/I2C

Stört den Programmablauf nur wenig

- Interrupts benötigen meist viel Zeit, gerade bei großen, modernen Prozessoren (Cache, FPU, etc)

Scheduling

Scheduler

- Routine, die Taskkontexte (Stack, Register) wechselt, bei Bedarf nach Prioritäten und Status der Tasks

Wird Aufgerufen von:

- SW selber, ich will jetzt die Kontrolle aufgeben, Warten auf Resource
- Zeitscheibe (Timerinterrupt)
- Am Ende eines Tasks

Scheduling

Echtzeit-Scheduler

- Sehr harte Timing-Anforderungen, z.B. Verbrennungsmotorsteuerung(Zünden, Einspritzen)
- Garantierte Reaktionszeiten (keine langen Interrupt sperrzeiten, keine langen atomaren Operationen)

Bei ”kleinen” Mikrokontrollern

- Bearbeitung direkt im Timerinterrupt, ist immer Alternative zum Betriebssystem

Scheduling

Prioritäten-Inversion - Beispiel

- 2 Tasks, Messdaten aufzeichnen (Prio hoch), Syslogs speicher (Prio niedrig)
- Beide speichern auf Dateisystem, das Schreiben eines Sektors ist atomar
- Syslog-Task beghinnt Sektor zu schreiben, und wird von Messtask unterbrochen, der auch schreiben will 

=> Entkopplung nötig zur Vermeidung der Prioritäteninversion