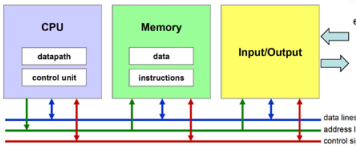
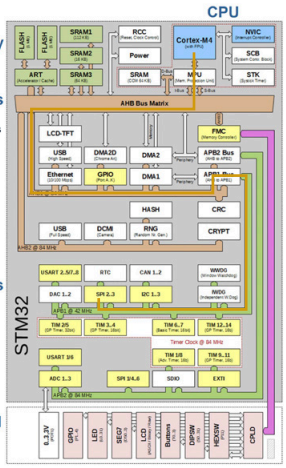
# Basics

## System Bus (Harward)

Interconnects **CPU** with **memory** and **peripherals.**




**CPU** acts as **master.** Initiates and controls all transfers.

**Peripherals** and **memory** act as **slaves.** Responding to requests from the CPU.

## Signal Groups

### Address Lines

Unidirectional: master to slave
32 address lines = $2^{32}$ addresses, one per Byte
**n address lines** means **n bits** per clk cycle.

**Data Lines:** 8, 16, 32 or 64 parallel lines of data. bidirectional (read/write)

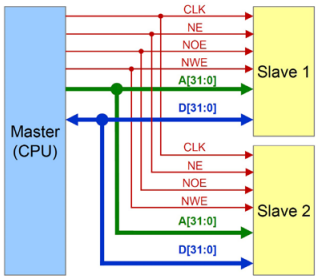
**Control Registers/Signals:** Provides **timing information.** (read/write)

Nxx = **active low,** 0 → enabled, 1 → disabled

| | |
|---|---|
| **NE:** | Not Enable (start and end of cycle) |
| **NWE:** | Not Write Enable |
| **NOE:** | Not Output Enable |
| **NBL[x:0]:** | Not Byte Line. Signals indicate valid bytes. |

Ein Control Register mit **NBL[4:0]** kann 5 Adressleitungen dekodieren.

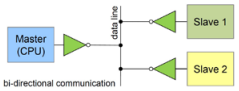
**CPU writes:** NE 0 and NWE 0 and NOE 1
**CPU reads:** NE 0 and NWE 1 and NOE 0

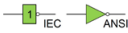



## Digital Logic Basics

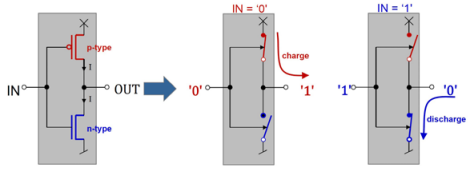
**Data Line Drivers (Treiber)** 


CPU defines which slave drives the data bus at which moment in time so that there is no electrical short, so that only one slave can write at a time.

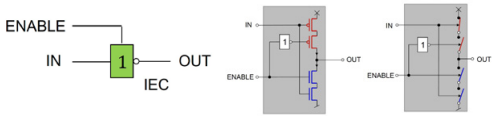



**CMOS Inverter** 

0 → 1



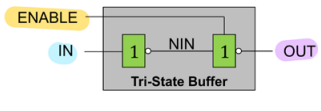

**CMOS Tri-State Inverter**
Set ENABLE = **0** to **disconnect** the driver

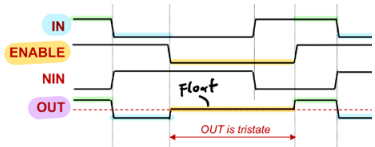



| ENABLE | OUT |
|---|---|
| '1' | ! IN |
| '0' | 'Z' |

**Z** = **floating**, high impedance

**Tri-State Buffer (Driver)**

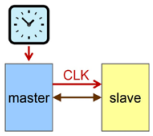



## Bus Timing

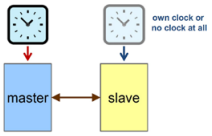
### Synchronous

Master send clk signal to slave. Base period can be encoded in a data signal. Used by most on-chip busses. Needs two lines.

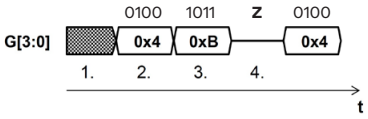



### Asynchronous

Control signals carry timing information to allow synchronization. Used for low data-rate. Only 1 line. Each node uses an individual clock. Both have to agree on a bitrate (baus).



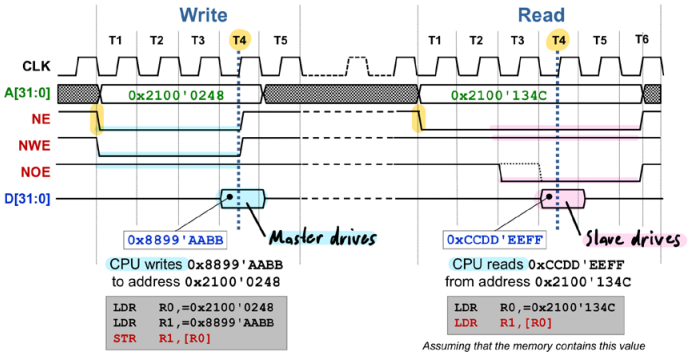

### Timing Diagram Notation for Groups of Signals




Leitung [bis : von]

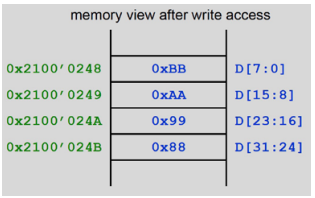
Bus G[3:0] holds 4 signals (lines).
0x4 means bus G has values G[3] = 0, G[2] = 1, G[1] = 0, G[0] = 0
Z means als signals are floaring (high impedance).

## Synchronous Bus Timing



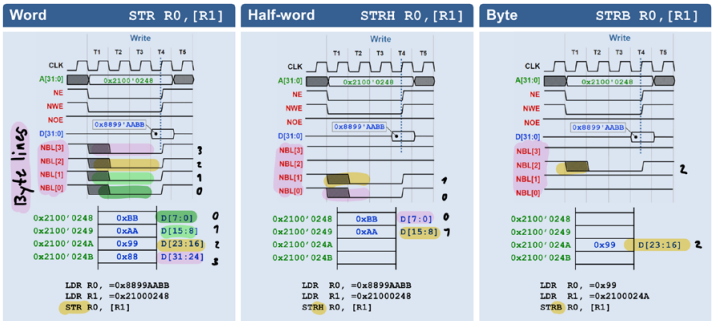

**Writes on T4** to prevent short circuit. T1 starts on **falling edge NE.**

Write


**Little Endian**

### Writing Word, Half-word and Byte → NBL[x:0]




Address must be aligned (for word and half word).

### Slow Slaves (Peripherals)

*microcontroller_basics, p.41.*

**Two possibilities:**

1. Individual **Wait States TW** depending on the address are **inserted before T4**

| T3 | TW | TW | TW | T4 |
|---|---|---|---|---|

2. Slave tells bus interface unit when it is ready.
   Used when when **access time is variable**

### Accessing Control Registers

dereference pointer → cast

```
#define LED31_0_REG     (*((volatile uint32_t *)(0x60000100)))
// Write LED register to 0xBBCC'DDEE
LED31_0_REG = 0xBBCCDDEE;
```

### Peripherals (Slaves)

Peripherals are **interfaces (protocols)** for the CPU to the outside world. Examples include **GPIO, UART, SPI, ADC.**

A peripheral is a **configurable hardware block** of a microcontroller that accepts a specific **task from the CPU,** executes this task and reports back the **status.**

Each Peripheral has a **base address** and an **offset addresses** for **control**, **status** and **data** **registers.**

### Registers

The CPU controls and monitors Peripherals through **registers:**

**Control Registers/Bits (CLK, NE, NWE, NOE, NBL):**
**CPU reads/writes, Slave reads**
Enable the CPU to configure the peripheral.
Ex: CPU controls the states of LEDs

**Status Registers/Bits (SR):**
**CPU reads, slave writes**
Enables the CPU to monitor (read) the status of the peripheral/salve.
Slaves writes status into register bit and CPU reads register bit.
Ex: CPU monitors the states of DIP switches

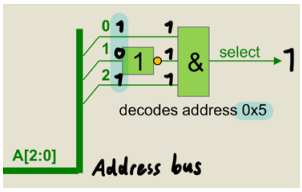
**Data Registers/Bits:**
**CPU reads/writes, slave reads/writes**
Enable the CPU to exchange data with the peripheral.

Die Adressen (Memory-Map) der Register werden vom Chip-Hersteller festgelegt.

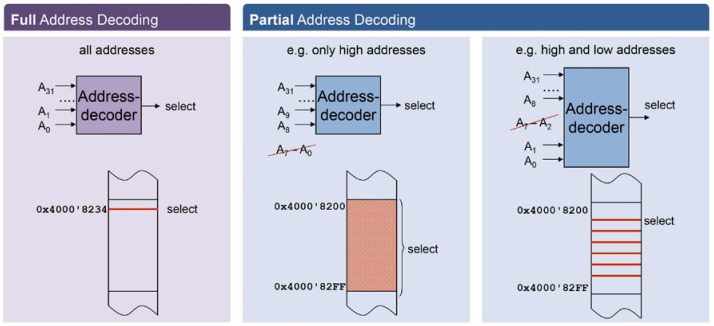
### Address Decoding

Accessing individual Peripheral Registers (physical memory location). Slave decodes, so it knows its the target.

**Full Address Decoding → 1:1 mapping**
Select if decoded address = 1.




**Partial Address Decoding → n:1 mapping**
Selects a range of addresses.




0x4000'82**xx** **(x = don't care)** → 0x4000'82**00** to 0x4000'82**FF**

Ein Slave ist unter mehreren Adressen erreichbar.

Bsp: Die Adresse 0x6**BFF**'0000 liefert das identische Byte wie unter 0x6**800**'0000 → Die Bits ADDR[25:16] (0xBFF – 0x800 + 1 = 1024 = $2^{10}$) sind nicht angeschlossen und werden vom slave deshalb nicht dekodiert.

Unter wie vielen 64K Byte Adressblöcken kann auf den Baustein zugegriffen werden? A[25:16] = 10 Adresslinien → $2^{10}$ = 1024 Adressblöcke (Bytes)

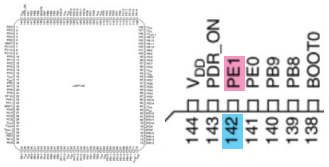
Beispiel:


2 Adressleitungen werden nicht dekodiert.

Adresslinien A[**0**] und A[**5**] werden vom Slave nicht dekodiert.
Es gibt $2^2$ = 4 mögliche Adressen (Adressblöcke) den Save anzusprechen.

## General Purpose I/O (GPIO Peripheral)

General Purpose I/O pins are **configurable** by software to select the needed function: **Input/Output, UART, SPI, ADC,** etc. For instance, multiple pins can be used togehter and act as USB port. **Pinouts:**

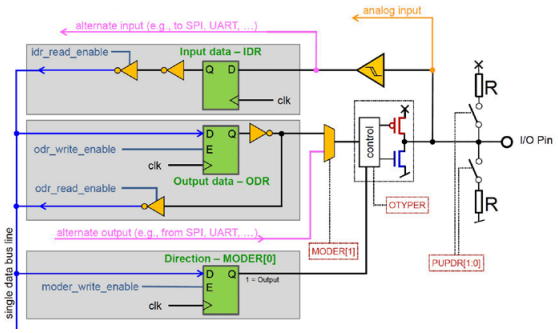



Port E.1 (PE1)
Pin 142

PE[15:0] → GPIO Port E
has 16 Pins

### Feature Registers (Alternate Functions)

**Configuration** is done through **registers:** MODER, OTYPER, PUPDR, etc.




**Register address = Base address + Offset**

Example: What is the **address** of the **register** GPIOA_OTYPER?
→ Check **offset** of **register** OTYPER

base + offset = address
0x4002 0000 + 0x4 = 0x4002 0004

**Manual:**
Base addesss → check **Memory Map, p.65**

| Boundary address | Peripheral | Register map |
|---|---|---|
| 0x4002 0000 - 0x4002 03FF | GPIOA | *Section 8.4.11: GPIO register map on page 290* |

Offset → check **Register Map** of Peripheral, e.g. GPIO Register Map, p.290

| Offset | Register |
|---|---|
| 0x04 | GPIOx_ OTYPER (where x = A..I/J/K) |
| | Reset value |

**Port Mode (Input/Ouput) Register: GPIOx_MODER**, p.284

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| port x.7 | | port x.6 | | port x.5 | | port x.4 | | port x.3 | | port x.2 | | port x.1 | | port x.0 | |

**MODERx[1:0]** (2 bits per pin)

00: Input
01: General purpose output mode
10: Alternate function mode
11: Analog mode

```
// set ports 0, 1 and 2 as output
// 1. reset register bits
GPIOB->MODER &= ~0x3F; // 0b11'1111
// 2. set register bits
GPIOB->MODER |= 0x15;  // 0b01'0101
```

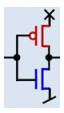
**Output Type Register: GPIOx_OTYPER**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OT15 | OT14 | OT13 | OT12 | OT11 | OT10 | OT9 | OT8 | OT7 | OT6 | OT5 | OT4 | OT3 | OT2 | OT1 | OT0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

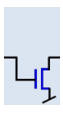
**OTx** (1 bit per pin)

0: Output **push-pull**
1: Output **open-drain**

```
// set ports 0, 1 and 2 to 0, 1, 1
// 1. reset
GPIOB->OTYPER &= ~0x7;
// 2. set
GPIOB->OTYPER |= 0x6;
```




**Push-Pull**
Can drive output **high** or **low.**




**Open-Drain**
Can **only** drive output **low.** Open drain is like an open switch.

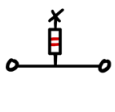
Push-pull ist wegen **Kurzschlussgefahr** nicht geeignet für den Betrieb an einem Bus mit mehreren Teilnehmern.

**Pull-Up/Down Register: GPIOx_PUPDR**

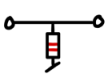
**PUPDRx[1:0]** (2 bits per pin)

00: No pull-up, no pull-down
01: Pull-up
10: Pull-down
11: Reserved

```
// set ports 0, 1 and 2
// 1. reset
GPIOB->PUPDR &= ~0x3F;
// 2. set
GPIOB->PUPDR |= 0x10;
```

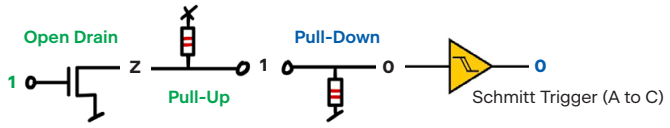


**Pull-Up**
Pulls a Z to high.



**Pull-Down**
Pulls a Z to low.

**Critical combinations:**
- Output (write) **Open Drain** + **No Pull** to input (read) **No Pull**: write **1** → read **Z**
- Output (write) **Open Drain** (pull/no pull) to Input **Pull-Down**: write **1** → read **0**
- Pull-Up with Pull-Down in generall




| | | Inputs (Read) | | |
|---|---|---|---|---|
| | | PA0 / LED16: Pull-Down | PA1 / LED17: Pull-Up | PA2 / LED18: No Pull |
| Outputs (write) | PB0 / S8: Push-Pull, NO pull-up/down | 0: 0 1: 1 | 0: 0 1: 1 | 0: 0 1: 1 |
| | PB1 / S9: Open Drain, NO pull-up/down | 0: 0 1: 0 | 0: 0 1: 1 | 0: 0 1: 0/1 |
| | PB2 / S10: Open Drain, pull-up | 0: 0 1: 0 | 0: 0 1: 1 | 0: 0 1: 1 |

## Pin Features
- Input or output
- High-current-capable
- Speed selection (how fast a signal can change, e.g. 0 → 1)
- Maximum I/O toggling (up to 90 MHz)

**GPIOx_OSPEEDR:** Output Speed Register
How fast a signal changes from 0 to 1 and vose versa.

**GPIOx_IDR:** Input Data Register
Port input data. Contains input value of corresponding I/O port. Read only.

**GPIOx_ODR:** Output Data Register
Port output data. Read and write by software.

**GPIOx_BSRR:** Clearing Bits Register
Ensures atomic access in software (no interruption possible). Write only.

**Example:** On a STM32F429 LQFP144 **Pin 37** shall be configured as **low-speed output** with **open drain** and **pull-up**:

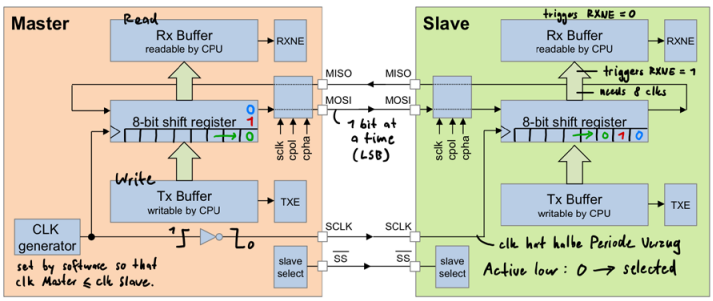
Pin 37 → Port **A.3**

| | | |
|---|---|---|
| GPIOA_MODER[7:6] | to '01' | Ouput |
| GPIOA_OTYPER[3] | to '1' | Open Drain |
| GPIOA_OSPEEDR[7:6] | to '00' | Low speed |
| GPIOA_PUPDR[7:6] | to '01' | Pull-up |

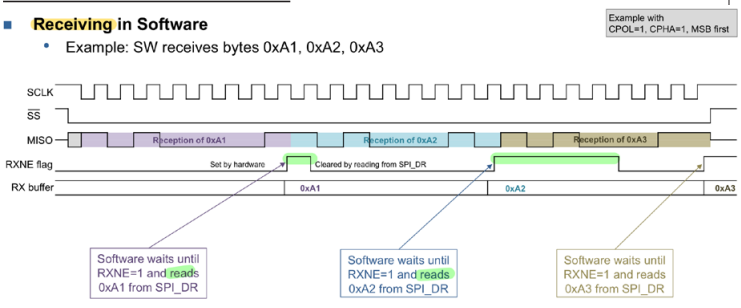
*Kapitel: SPI (Serial Peripheral Interface)*

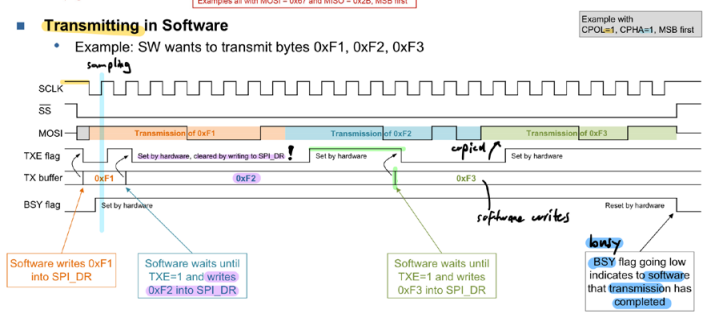
**Status bits with Interrupt**
TXE Tx Buffer Empty
RXNE Rx Buffer Not Empty

### Implementation using Shift Registers




### Transmitting and Receiving




## SPI (Serial Peripheral Interface)

### Properties

SPI is a **full duplex, synchronous** (separate clk signal) and **serial** data connection. Needs at least 4 wires. No official standards (no legally binding specs). Many different variants. **LSB/MSB** first is configurable.

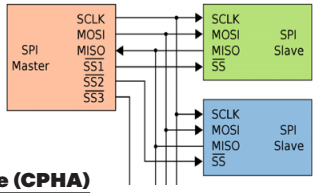
Mainly used for **on-board** connections and **short distance** communication. Connects microcontroller and external devices (sensors, displays, flash memories, etc.)

- No defined addressing scheme
- Transmission-receive-acknowledge and error-detection has to be implemented in higher protocols (software)
- No flow-control (no wait-states, slave cannot influence the data rate and has to be ready)
- Susceptible to spikes on clock line:
  spike → slave shifts bit → Fehlerübertragung

**Single Master, Multiple Slaves**

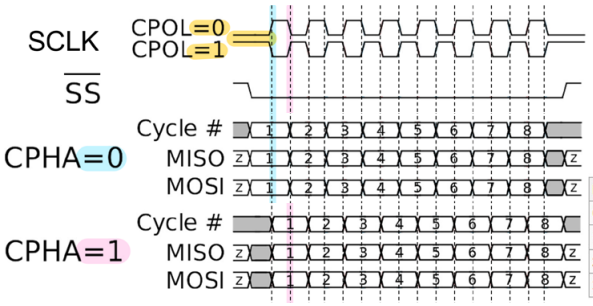
Master generates clk signal for all slaves.

MOSI: Master Out, Slave In
MISO: Slave Out, Master In




### Clock Polarity (CPOL) and Clock Phase (CPHA)

Bsp: TX provides data on toggling edge. RX takes data with sampling edge.




| Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

### Synchronizing Hardware and Software

Software has no direct acces to shift register → Software has to check status register **SPI_SR** and flags/bits **TXE** (for sending) or **RXNE** (for receiving):

**TXE=1** Software can write next Byte to data register **SPI_DR**
**RXNE=1** A byte has been received. Software can read it from **SPI_DR**

**TXE** bit/flag is **set by hardware** and **cleared** by **writing to SPI_DR** in software
**RXNE** bit/flag is **set by hardware** and **cleared** by **reading from SPI_DR**

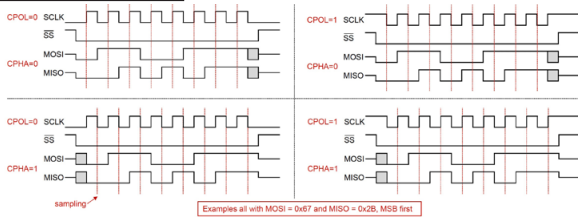
Master oder Slave müssen immer **gleichzeitig senden und lesen.** Möchte der Master nichts senden, der Slave aber schon, sendet der Master nur '0's.

### SPI Control Register (SPI_CR)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIDI MODE | BIDI OE | CRC EN | CRC NEXT | DFF | RX ONLY | SSM | SSI | LSB FIRST | SPE | BR [2:0] | | | MSTR | CPOL | CPHA |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

- **SSI:** Internal slave select. This bit has an effect only when the **SSM** (Software slave management) bit is set.
- **SPE:** SPI enable
- **BR:** Baud rate control
- **MSTR:** Master selection. 0: Slave configuration. 1: Master configuration

### Timing Diagramm



## UART (Universal Asynchronous Receiver-Transmitter)

5–8 Bits of Data Transfer (always **LSB first**)

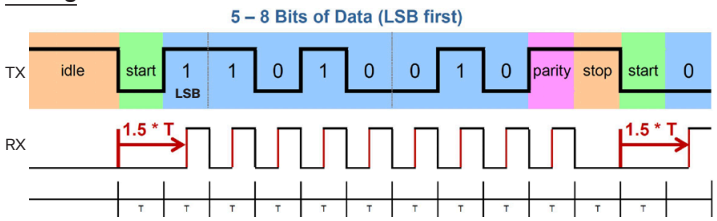
### Synchronization

**Problem:** TX (transmitter) and RX (receiver) have diverging clock sources and different divider rations. **Clock is not transmitted** but requires **synchronization at start of each data item.**

Transmitter and receiver use **closest integer** to divide **(Prescaler) clock**:

Example: **9600** Target Baud (Hz)
**2 MHz** ÷ **208** = **9615 Hz** → slightly too fast  **prescaler** = round(**clk**/baud)
**5 MHz** ÷ **521** = **9597 Hz** → slightly too slow

### Timing




Receiver (RX) samples data in middle of bits on rising edge.
**Synchronization** happens between each **falling edge** after **last stop bit**.

$$Baud = Symbols^{*} / s \quad {}^{*}Bits$$

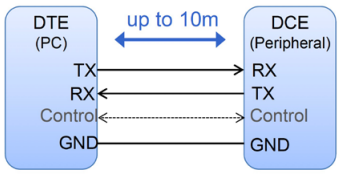
$$T = \frac{1}{Baud}\ [s]$$ Every *T* second a bit can be sent

Maximal erlaubte **Taktabweichung (% von T)**,
damit ein Zeichen fehlerfrei empfangen werden kann.

$$Max.\ Abweichung = \frac{0.5}{Bits^{*} - 0.5}\ \%$$

*Start bis Parity Bits

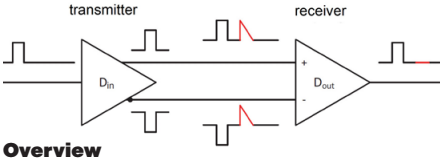
### Electrical Charasteristics




Logic '1' -3V to -15V
Logic '0'  3V to  15V

### Differential Transmission

Allows 100+ Meters. Noise on the two lines cancels itself (to a large extent).




### Overview

**SPI, I2C, UART** provide the **lowest layer** of communication and **require higher level protocols** to provide and interpret the transferred data.

When to use which one:
- **UART:** **Long distance** communication (~10m wires) but **slowest**
- **SPI:** **Fastest** but only **on-board** and requires **more pins**
- **I2C:** **Multi-point-to-multi-point** but only **board-to-board** (med. dist.)

| UART | SPI | I2C |
|---|---|---|
| serial ports (RS-232) | 4-wire bus | 2-wire bus |
| *TX, RX* opt. control signals | *MOSI, MISO, SCLK, SS* | *SCL, SDA* |
| point-to-point | point-to-multipoint | (multi-) point-to-multi-point |
| full-duplex | full-duplex | half-duplex |
| asynchronous | synchronous | synchronous |
| only higher layer addressing | slave selection through $\overline{SS}$ signal | 7/10-bit slave address |
| parity bit possible | no error detection | no error detection |
| chip-to-chip, PC terminal program | chip-to-chip, on-board connections | chip-to-chip, board-to-board connections |

### Serial Communication

Provides simple, low-lewel physical connection and reduces number of switching lines. But requires a **higher level protocol,** usually **implemented in software.**

**Communication Modes:**
- **Simplex:** Unidirectional, one-way only
- **Half-duplex:** Bidirectional, only one direction at a time
- **Full-duplex:** Bidirectional, both directions simultaneously

**Communication Interfaces:**
**SPI, I2C, UART:** These three interfaces provide the **lowest layer** of communication and **require higher level protocols** to provide and interpret the transferred data.

## I²C (Inter-Integrated Circuit)

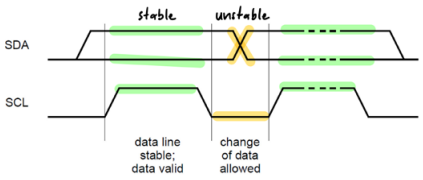
Pronounced «i squared c». Always **MSB first.**

8-bit oriented data transfer **(MSB)**. Lines **Clock (SCL)** and **Data (SDA).**
**Each device** (masters ans slaves) on bus **addressable** through **unique address.** Multiple-Master possible.
Well suited for connection of multiple boards (HDMI).

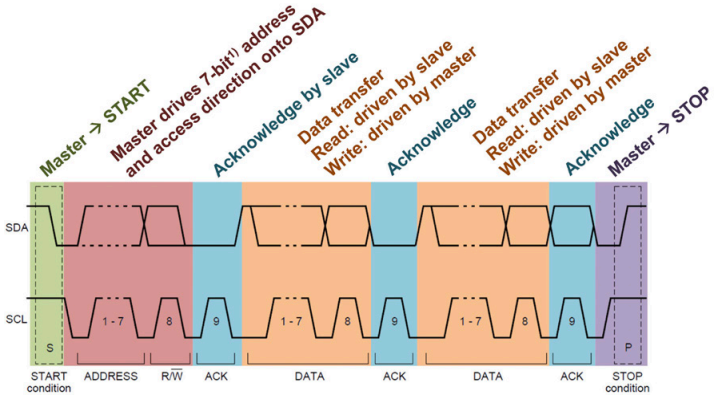
### Data Transfer

- Master drives clock (SCL)
- Master initiates transaction through **START condition** (falling edge on SDA when SCL high)
- Master terminates transaction through **STOP condition** (rising edge on SDA when SCL high)
  Stop can be sent anytime (it's like an interrupt)

**Change of data** only allowed when **SCL is low.**

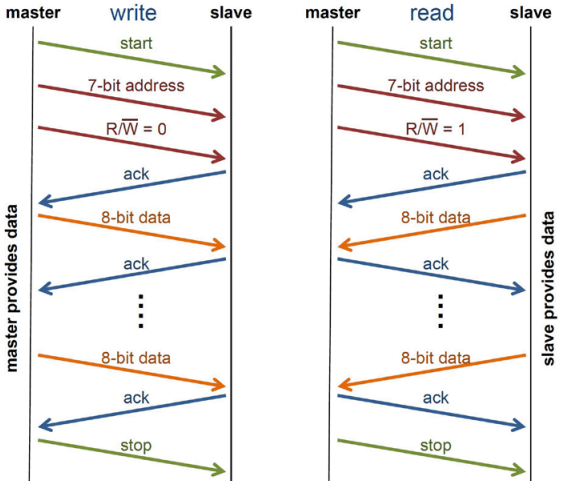



- Bit numbering starts with 1 not with 0.
- **Bit 9: Receiver** acknowledges by **driving SDA low.** ACK is active low.
- **R/W:** 1 = Master Read, 0 = Master Write




First **Bits 1–8** defines address and wether master is reading (1) or writing (0).
Example: **0x66** = **011 0011** 0 → address = **011 0011** (0x33), R/W = 0

### Accesses



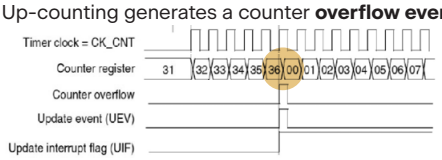
## Counters

### Use

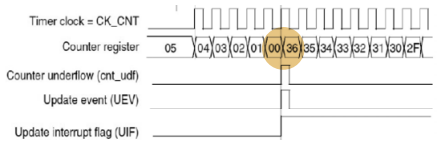
- Count of events (clock pulses or external signals)
  e.g. count number of pulses on input pin
- Measure time / frequencies, phases, periods
  e.g. time between risin edges of an input pin
- Intervals, pulses, interrupts
  e.g. generate sequence of pulses on an ouput pin

### Modes

- **Up- or down-counting**
  Up-counting generates a counter **overflow event** (Compare)




  Down-counting generates a counter **underflow event**




- **16-bit / 32-bit counter register**
- **Set interrupt flag to trigger interrupt**

### Prescaler

Wird zum heruntertackten der zeitliche Basis (internal clock) benötigt.
**= Divisor für Eingangssignal.** Der Prescaller lässt nur jeder n-te Puls durch.
Optionally but optimal the prescaler **should be divisor** of the source (Hz).
Grundsätzlich kann der Prescaler **frei gewählt** werden, so dass < $2^{16}$
The greater the prescaler, the less ticks.

Prescaler = $\lceil (pulse_{Hz} / reload_{Hz}) / 2^{16} \rceil$  PSC = Prescaler – 1

### Auto Reload Register (ARR)

Wird ausgelösst, sobald ein gewisser Wert erreicht ist.
0 <= Counter <= ARR

ARR = (clk / interrupt) / (prescale + 1) - 1

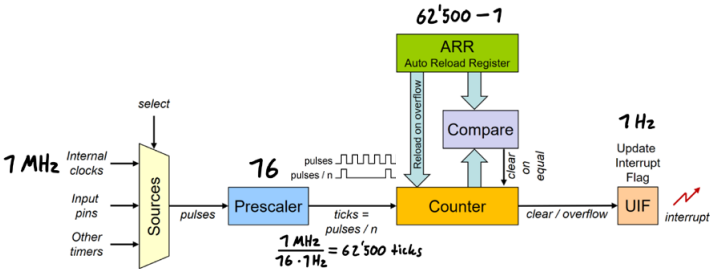
### Example

Source 1 MHz → period T = 1 / 100 MHz = **0.01 us**
**16-bit counter**

We want an interrupt (= reload) every: **1 s (1 Hz)**
1 MHz / 1 Hz **> 65'536**
Prescaler = (1 MHz / 1 Hz) / $2^{16}$ = 15.2... → **16** → PSC = 15

Optionally but optimal the prescaler **should be divisor of the pulse (Hz),** set prescaler to next higher divisor.

**ARR** = (1 MHz / 1 Hz) / **16** = 62'500 ticks **– 1**
Validate: 62'500 ticks · **0.01 us** · **16** = **1 s**




Counter wird mit jedem n-ten Tick um eins erhöht oder erniedrigt.

Umso kleiner der Prescaler, desto mehr Ticks und desto besser wird der Wertebereich des Auto-Reload-Registers (ARR) ausgeschöpft.

## Timers

Jeder Timer braucht eine Clock.

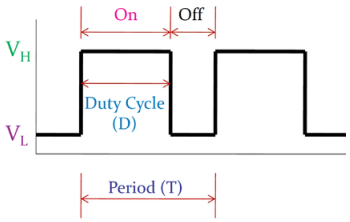
**Modes:** Input Capture, Pulse-Width-Modulation (PWM),
Output Compare (Generating PWM), Capture/Compare Configuration

### Input Capture (CCR Register)

Measures intervals (pulse lengths and periods) by counting ticks between timer start and an event.

### Pulse Width Modulation (PWM)

Voltage is proportional to duty cycle of PWM-Signal.

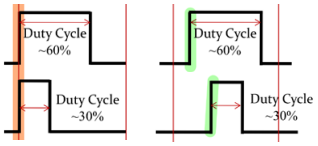



$$Duty\ Cycle = \frac{On\ Time}{Period} \times 100\%$$

Average signal

$$V_{avg} = D \cdot V_H + (1 - D) \cdot V_L$$

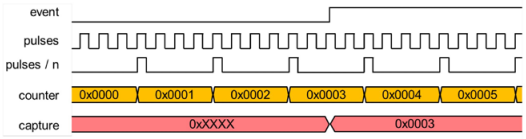
**Alignment**
Left-alignment löst elektromagnetischen Pulse aus, ist zu vermeiden
→ **Center Aligned**




### Capture/Compare Register (CCRx)

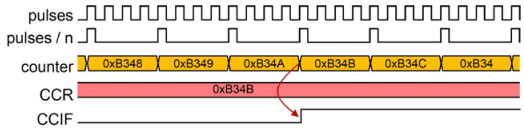
CNT (counter) threshold.
**Function Capture:** Speichert bei Event **CNT (counter)** in **CCR.**

Bei einem Event wird der Inhalt des Counter Registers (CNT) als Vergleichswert in das Capture / Compare Register kopiert. Der Counter läuft weiter.




**Function Compare:** Löst **Event (CCIF)** aus wenn CNT = CCR

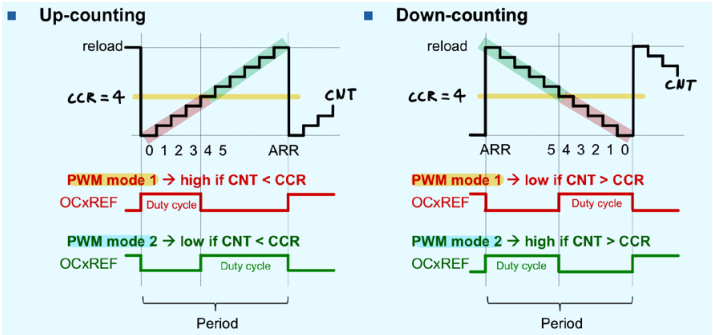
Sobald der **Counter** den Wert des Capture/Compare Register (CCR) erreicht hat, wird ein Event oder ein Interrupt ausgelöst. Der Counter läuft weiter.



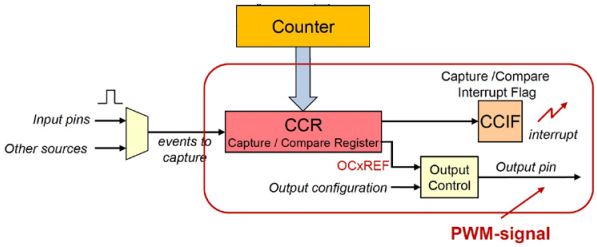

### Generating PMW Signal

**Compare function** generates PWM Signals.

**Reload (=ARR):** Wert für Timerüberlauf
Upcounter: Timer zählt bis zu diesem Wert, dann Überlauf.
Downcounter: Startwert für Timer




PWM Mode 1 (CCMR): CCR = (ARR + 1) · Duty-Cycle
PWM Mode 2 (CCMR): CCR = (ARR + 1) · (1 - Duty-Cycle) - 1



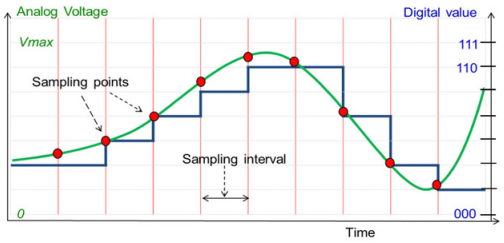
## ADC (Analog to Digital Converter)

Converts input voltage signal to a digital N-bit value.
Result is one if **$2^N$ possible level.**

**Sampling rate:** At least twice the highest frequency → Nyquist–Shannon

### 3-bit ADC Example

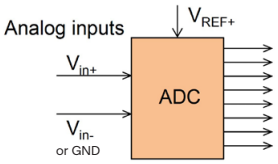
$2^3 = 8$ possible levels/resolution (000, 001, ..., 110, 111)




### ADC Converter

**Reference Voltage $V_{REF+}$**

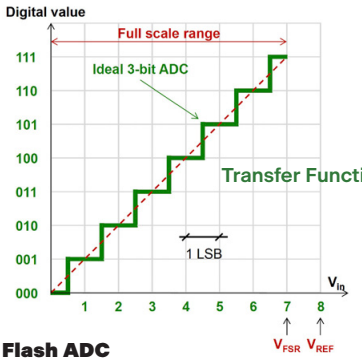
- Internal or external stable voltage
- Needed to **weight input voltage**




$V_{in} = (digital\ value) * V_{REF+} / (2^N)$

$V_{in} = V_{in+} - V_{in-}$

### Full Scale Range ($V_{FSR}$)




Was ist die **absolute Adresse** des Registers, in dem die Werte des ACD2 gelesen werden können?

1. Memory Map, p.66
   Base Adress ADC 0x4001 2000
2. ADC Register Map, p.433
   Offset ADC2 + 0x100
3. Offset Data Register, p.428
   ADC_DR + 0x4C

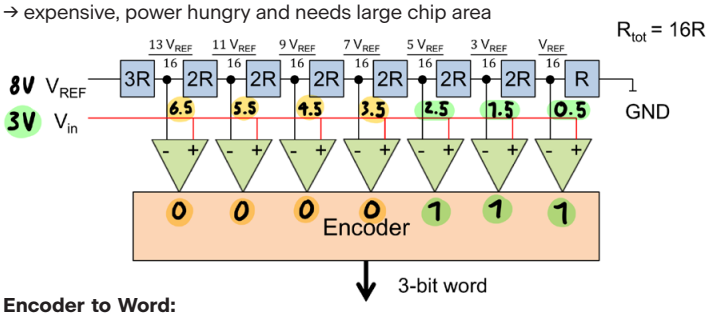
   = 0x4001 214C

**LSB** = $V_{REF} / 2^N$ [V]

$V_{FSR} = V_{REF} - 1$ LSB [V]

### Flash ADC

Network of $2^N$ resistors
→ expensive, power hungry and needs large chip area



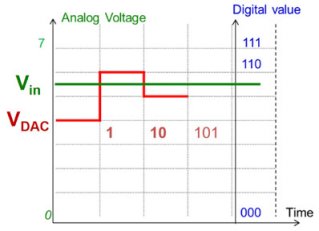

**Encoder to Word:**

0000000 → 000 0000**11** → 010 ...
0000**001** → 001 0000**111** → 011 **1111111** → 111

### Successive Approximation Register (SAR) ADC

Better alternative to the Flash ADC.
Used on most microcontrollers. Resolution from 8 to 16 bits.
Good trade-off between speed, power and cost.

Approach $V_{in}$ with successive division by 2 = **Binary search**
Start with half the digital value and MSB first.



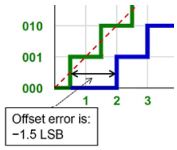

1. Higher than 100 → 1
2. Lower than 110 → 10
3. Higher than 101 → 101

3-bit result = 101

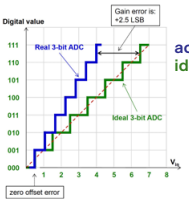
### Types of Error

**Quantization (Precision) error:** Error between −0.5 LSB and +0.5 LSB.
The lower the resolution, the higher the error.
Solution: Increase number of bits (resolution) or reduce $V_{REF}$

**Offset error**
Can be corrected using the microcontroller.




**Gain error**
full-scale error = offset error + gain error
Calibration with hardware or software.




### Convertion Modes

| | Single channel | Multi-channel (scan mode) |
|---|---|---|
| **Single conversion** | Convert 1 channel, then stop. This is the simplest mode. | Convert all channels in group, one after the other, then stop. The group of channels is in a sequence that can be programmed. |
| **Continuous conversion** | Continuously convert 1 channel until stop order is given. Minimal CPU intervention. | Continuously convert a group of several channels until stop order is given. The group of channels is in a sequence that can be programmed. |

**Single Conversion:** External trigger.
Sensor tied to ADC input or on a push button you read the value.

**Continuous Conversion:** Wert wird vorzu in data register geschrieben.
Continuously monitor a single sensor.

**Multi-Channel:** Group of sensors that need to be monitored.
ADC input for each sensor.

### ADC Timing

**Total Conversion Time:** $T_{total} = T_{sample} + T_{conv}$ [s]

$T_{sample}$ Sample and hold time. Between 3 and 480 cycles
$T_{conv}$ Resolution. 6/8/10/12 ADCCLK cycles

Example:
clock = 48 MHz with Prescaler 2 → ADCCLK = clock / Prescaler = 24 MHz
3 cycles sampling time, 12 bit resolution

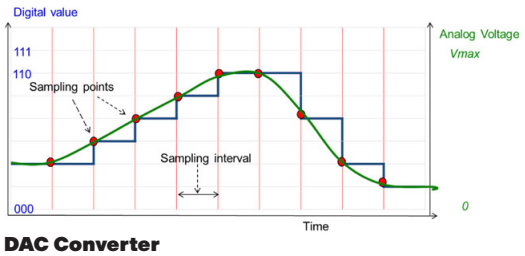
$T_{total} = (3 + 12) \cdot (1 / 24\ Mhz) = (3 + 12) \cdot (2 / 48\ MHz) = 0.625$ us

Sampling Rate < $1 / T_{total}$ = 1.6 Msps (MHz)

## DAC (Digital to Analog Converter)

Converts N-bit digital input to analog voltage level

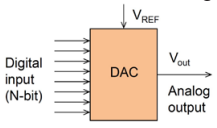
### 3-bit DAC Example




### DAC Converter

**Reference Voltage $V_{REF}$**

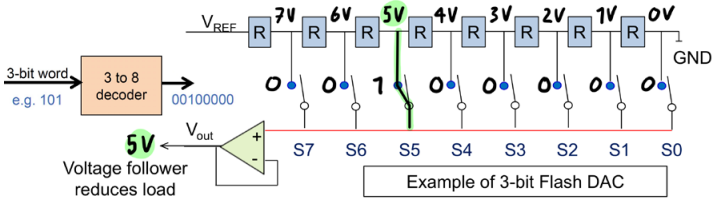
- Needed to relate digital value to a voltage



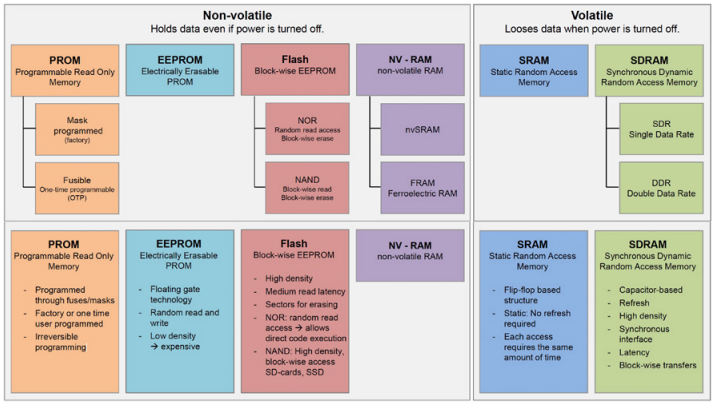

$V_{out} = (digital\ value) * V_{REF} / (2^N)$

### Flash DAC

Network of resistors (of same value) creates $2^N$ voltage levels.
Decoder controls **switches**. One switch for one bit.



# Memory




**ROM (Read Only Memory) → Non-volatile** (persistent data storage)
PROM (Programmable ROM), Flash Memory,
EEPROM (Electrically Erasable Programmable ROM)

**RAM (Random Access Memory) → Volatile** (flüchtig)
Memory content retained only as long as device is powered.
SRAM (Static RAM), SDRAM (Synchronous Dynamic RAM)

**On-Chip Memory**
- **Flash** (persistent): Programms are stored here
- **SRAM** (volatile): Variables used by programms are stored here

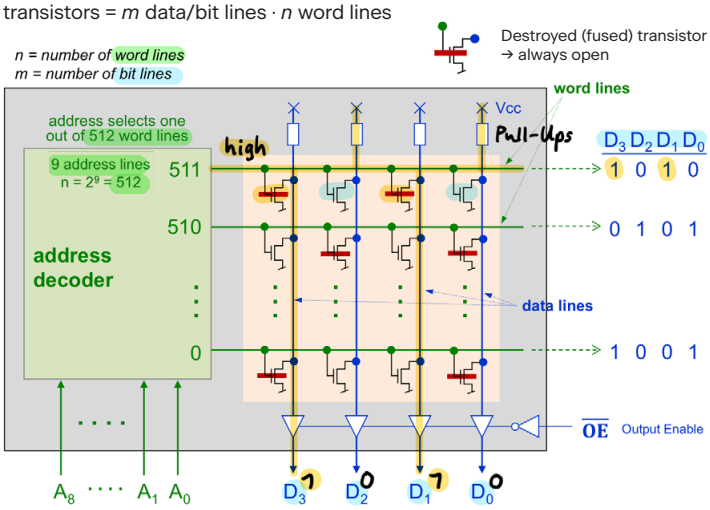
## Units for Memory Chips

| | | | |
|---|---|---|---|
| KiB | $= 1024^1 = 2^{10} =$ | 1024 | **KibiByte** (**Kilo** **bi**nary **Byte**) |
| MiB | $= 1024^2 = 2^{20} =$ | 1'048'510 | MebiByte |
| GiB | $= 1024^3 = 2^{30} =$ | 1'073'741'824 | GibiByte |
| TiB | $= 1024^4 = 2^{40} =$ | 1'099'511'627'776 | TebiByte |

## PROM (Programmable Read Only Memory)

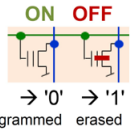
**Fusible Transistors**
Programming applies higher voltage to destroy transistors (blow fuses).
Process is **not reversible** → Reprogrammable PROMs: EEPROM and Flash

word lines (Addressblöcke) = $2^{\text{address lines}}$
transistors = *m* data/bit lines · *n* word lines




## EEPROM and Flash

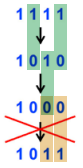
Reprogrammable PROM.
Uses **Floating Gates** instead of Fusible Transistors.




**EEPROM:** High cell area, low density, high cost per bit

**Flash Memory:** Erasing can only be done for whole sectors.
Small cell area, high density, low cost per bit

**Operations**




**Write (Programming):**
Can only change bits from '1' to '0'

**Erase:**
Change **all bits** from '0' to '1'

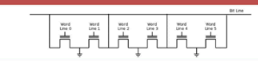
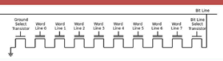
→ **1111**
→ 1011

**0 → 1** requires erase of sector

## Flash Memory

Volatile (flüchtig). On-Chip. High latency.

**NOR Flash** ist mit Schnittstelle von asynchronous SRAM für Lesezugriffen kompatibel. **Solid-State-Disks (SSD)** are built from **NAND Flash.**

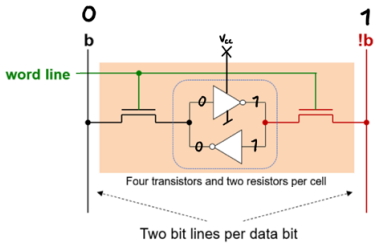
| | NOR Flash | NAND Flash |
|---|---|---|
| Topology |  |  |
| Applications | • Execute code directly from memory<br>• Persistent device configurations (replacement of EEPROM) | • File-based IO, disks<br>• Large amounts of sequential data (images, SD cards, SSD)<br>• Load programs into RAM before executing |
| Density | • **Medium** Up to 2 GBit = 256 MByte | • **High** Up to 1 Tbit = 128 GByte |
| Interface | • Read same as asynchronous SRAM<br>• Types with serial interface (SPI) available | • Special NAND flash interface<br>• Error correction for defective blocks |
| Access | • Random access<br>read ~0.12 µs<br>• **Writing individual bytes possible**<br>• Slow writes<br>~180 µs / 32 Byte | • **Slow random access**<br>read: 1. Byte 25 µs,<br>then 0.03 µs each<br>• **Writing of individual bytes difficult**<br>• **Fast block write**<br>~300 µs / 2'112 Bytes |

Für grosse Speicher geeignet, günstiger

## SRAM (Static Access Memory)

Programmable Flip-Flop (latch) based cells. Volatile. On-Chip.
**Two bit lines per data bit.**

**Static** = No refresh required (DRAM needs regular refreshing).
Lower power usage than DRAM.
SRAM is faser than DRAM but more complex, larger and more expensive.




**Asynchronous Interface**

| $\overline{CS}$ | $\overline{OE}$ | $\overline{WE}$ | I/O | Function |
|---|---|---|---|---|
| L | L | H | DATA OUT | Read Data |
| L | X | L | DATA IN | Write Data |
| L | H | H | HIGH-Z | Outputs Disabled |
| H | X | X | HIGH-Z | Deselected |

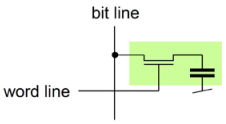
Some memory vendors call the signal $\overline{CE}$ (chip enable) instead of $\overline{CS}$

**Control signals**
**CS** Chip Select, usually active-low
**OE** Output Enable, usually active-low
**WE** Write enable, usually active-low

**Aufgabe:** Wieviel Adresspins (word lines) werden für 64K x 8 Bit benötigt?
$\log_2(64K) = 15.9... \rightarrow$ 16 Adresspins → ADDR[15:0]

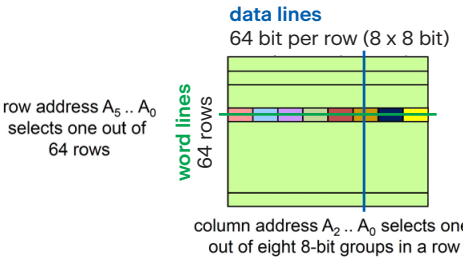
## SDRAM (Synchronous Dynamic RAM)

Volatile. Information stored as charge in **capacitor**
→ Leakage current → Loss of charge
Capacitor holds charge only for a few milliseconds.
**Dynamic** = Has to be refreshed periodically.




Allows to store large amounts of data at low cost.

**SDRAM Structure**
Row and column addresses multiplexed (buffer acts as a cache and stores a complete row). **Long latency** for first data item in row but then short access time for second item in same row.




**Synchronous Interface**
RAS Row address strobe
CAS Column address strobe

| Static RAM (SRAM) | Synchronous Dynamic RAM (SDRAM) |
|---|---|
| Flip-flop/latch → 4 Transistors / 2 resistors | Transistor and capacitor |
| Large cell<br>• Low density, high cost<br>• Up to 64 Mb per device | Small cell<br>• High density, low cost<br>• Up to 4 Gb per device |
| Almost no static power consumption<br>• Static i.e. no accesses taking place | Leakage currents<br>• Requires periodic refresh |
| Asynchronous interface (no clock)<br>• Simple connection to bus | Synchronous interface (clocked)<br>• Requires dedicated SDRAM Controller |
| All accesses take roughly the same time<br>• ~5ns per access → 200 MHz<br>• Suitable for distributed accesses | Long latency for first access of a block<br>• Fast access for blocks of data (bursts)<br>• Large overhead for single byte |

**Zugriffszeit:**
**SRAM:** alle Zugriffzeiten gleich lang.
**SDRAM:** Hohe Latenz für ersten Zugriff, kurze Zugriffszeit für Folgeadressen.
Eignet sich für Blockzugriff.

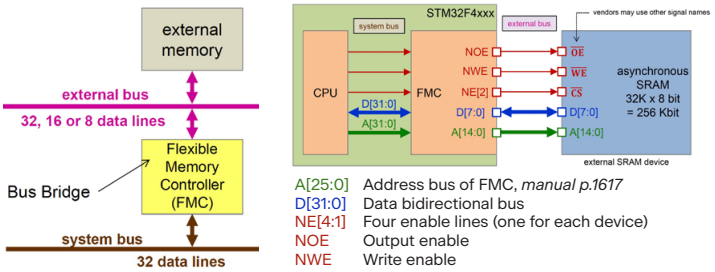
## Off-Chip Memory

**Issues when connecting external Memory:**
- Different Bus Widths than on System Bus (32/16/8 Bits on External Bus)
- Different Address Ranges
- Different Timing (fast on System Bus, Slow on External Bus)

→ Flexible **Memory Controller (FMC)** for Address-Translation.

## Flexible Memory Controller (FMC)

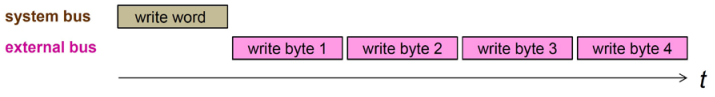
**Configurable Bus Bridge** between system bus and external bus.
Translate between internal and external Adresses → **Address-Translation.**



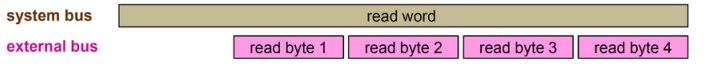

A[25:0] Address bus of FMC, *manual p.1617*
D[31:0] Data bidirectional bus
NE[4:1] Four enable lines (one for each device)
NOE Output enable
NWE Write enable

### Different Number of Data Lines causes Bottleneck

Example: **Writing** a 32-bit Word from System Bus (32 Data Lines)
to an 8-bit wide external memory (8 data lines)



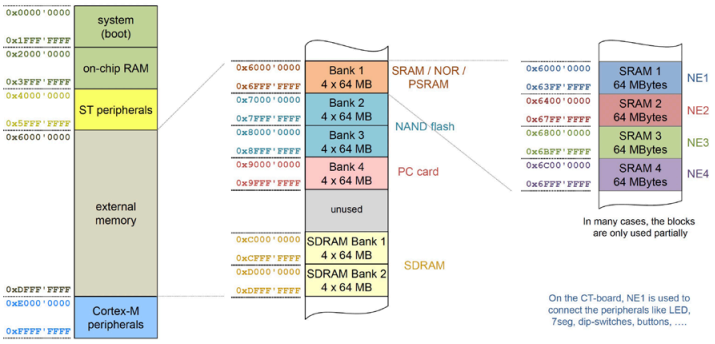

Example: **Reading** a 32-bit Word from an
8-bit wide external memory (8 data lines)




Reading: System bus **always has to wait** until all data is available.

### Memory Banks

Vordefiniert **Speicherbereich pro Speicherart.**
Each bank allows connection of 4 devices.

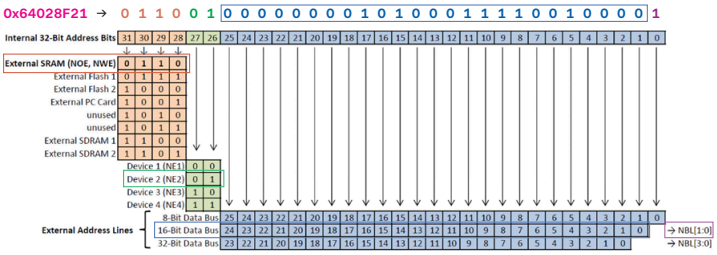



Manual: FMC memory banks p.1610

### Example: Address De-Composition

Write Byte to External SRAM, Device 2 (NE2) on 16-bit wide data bus
to address 0x64028F21 (in software)

Select Byte Line 1 → NBL[0] = 1, NBL[1] = 0 (active low)
Memory Address on Device → 0x00014790




### Example: Addess Range

Geg. SRAM mit 16 Adresspins (word lines) ADDR[15:0]
Tiefste Address des Bausteins is 0x6800'0000.
Was ist die höchste Adresse? 0x6800'0000 + $2^{16}$ – 1 = 0x6800'FFFF

# Cache

**Problem:** DRAM is fast for blocks of data (bursts) but slow for single byte.

**Idea:** Access slower main memory in bursts and maintain a fast cache for single accesses for frequently or recently used instructions or data.

Cache: 1–2 cycles, Memory: 100–1000 cycles

## Locality

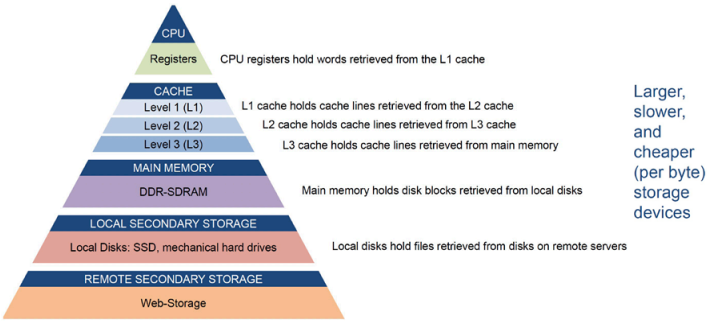
**Spacial:** Current data location is likely being close to next accessed location.
→ incremental access (loop):

```
for(i = 0; i < 100000; i++) {      // incremental access
    a[i] = b[i];                   // → spatial locality
}
```

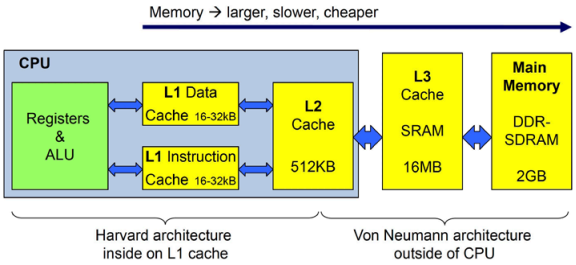
**Temporal:** Current data location is likely being accessed again in near future.
→ if-statement:

```
if (a[1234] == a[4321]) {          // → temporal locality
    a[1234] = 0;
}
```

## Memory Hierarchy




## Basic Cache Architecture

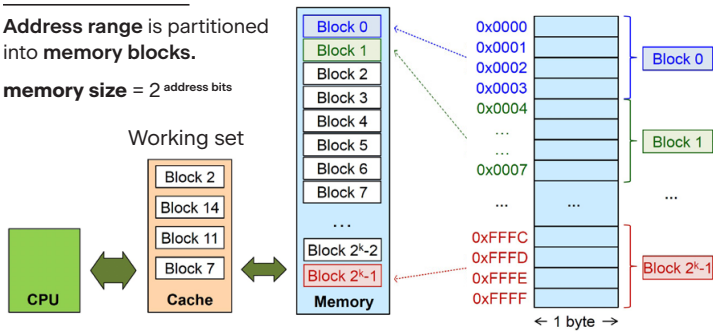



**Harvard:** 2 separate busses (lines) for data and instruction.
**Neumann:** 1 bus (line) for both dara and instruction.

## Cache Mechanics

**Address range** is partitioned into **memory blocks.**

**memory size** = $2^{\text{address bits}}$

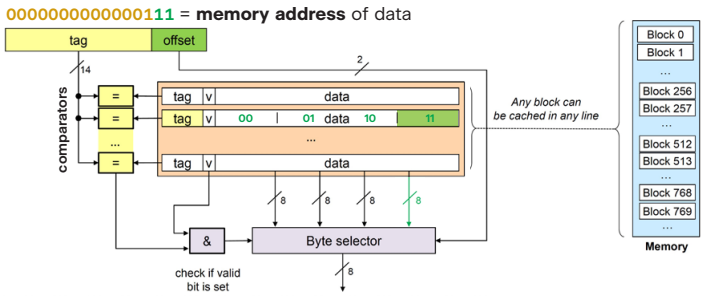



## Cache models

| Organization | Fully associative | Direct mapped | N-way set associative |
|---|---|---|---|
| **Number of sets** | 1 | m | m/n |
| **Associativity** | m(=n) | 1 | n |
| **Advantages** | • Fast, flexible<br>• Highest hit rates<br>• Advanced replacement strategies | • Simple logic<br>• Replacement strategy defined by organization | Combination of both other concepts to combine advantages and to compensate disadvantages |
| **Disadvantages** | • Complex logic: one comparator per line<br>• Requires large area on silicon<br>• Replacement can be complex | • Lower hit rates | |

### Fully Associative

Organization: **Any block can be cached in any line** → random seleciton
Comparators check if data with tag is in cache.




### Direct Mapped

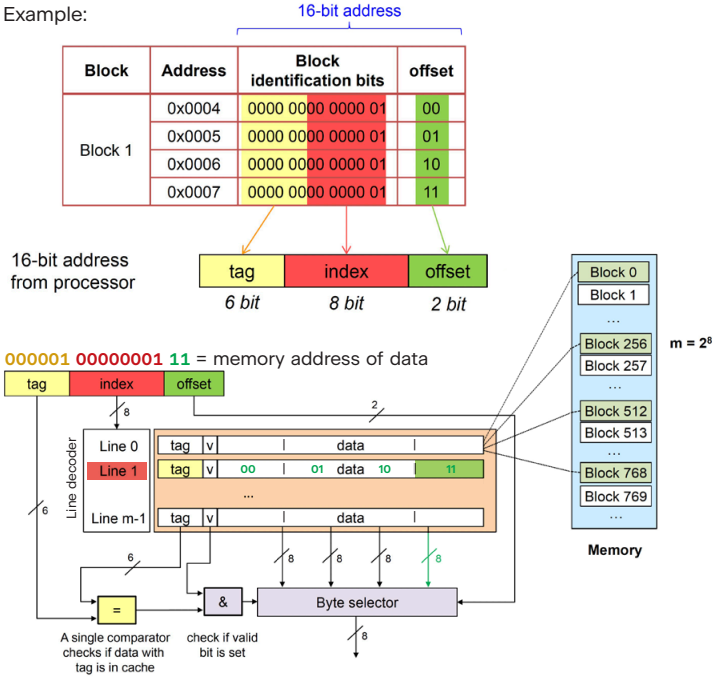
Organization: **Each memory block is mapped to exactly one cache line**
→ Multiple memory blocks will map to the same line.

- **Tag:** Identifiziert ob sich ein Block gerade im Cache befindet.
- **Index:** (1) Bestimmt die Anzahl Zeilen (Blocks) im Cache und
  (2) Selektiert die Cache-Zeile (Block) der Memory Adresse.

  **Cache lines** = $2^{\text{index-bits}}$
  **Cache size** = cache lines · blocksize [Bytes]
- **Offset:** (1) Bestimmt die Anzahl Bytes in einer Cache Zeile (Block) und
  (2) selektiert das entsprechende Byte in der Cache-Zeile.

  **Blocksize** = $2^{\text{offset-bits}}$ [Bytes]
- **v:** validity, indicates that the line contains valid data

Example:




**Cache Line** = floor(Address / Blocksize) % Blocksize

### N-Way Set Associative

Organization: Partition into sets. Folie S.24

### Performance

- **Cold miss:** First access to a block
- **Capacity miss:** The cache cannot hold all the data required by the system (Working set is larger than cache)
- **Conflict miss:** Multiple data blocks map to same line or set

**hit rate** = nr_of_hits / nr_of_accesses
**miss rate** = nr_of_misses / nr_of_accesses = 1 – hit rate

### Code Example

```
for(j = 0; j < 10000; j++){
  for(i = 0; i < 40000; i++){
    c[i][j] = a[i][j] + b[i][j];
  }
}
// a[i][j] and a[i+1][j]
// are 10'000 elements apart
```

```
for(i = 0;i < 40000; i++){
  for(j = 0;j < 10000; j++){
    c[i][j] = a[i][j] + b[i][j];
  }
}
// a[i][j] and a[i][j+1]
// are next to each other
```

→ big steps outside, small steps inside

## Finite State Machines (FSM)

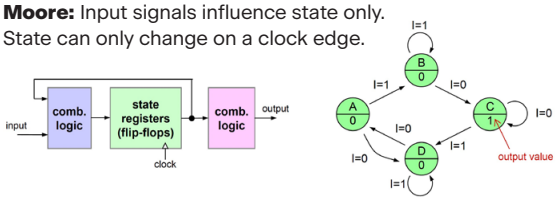
Idee: Endliche Anzahle Zustände in Software oder Hardware abbilden.
Always has a defined state.

**Run-to-completion:** Once started a transition cannot be interrupted.

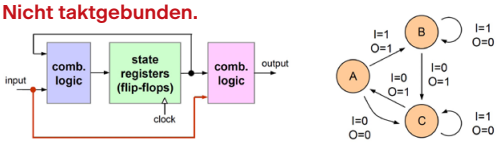
### State Machines in Hardware

Flip-flops store internal state.

**Moore:** Input signals influence state only.
State can only change on a clock edge.




**Mealy:** Input signals influence state AND output signals.
Nicht taktgebunden.




### FSM in Software vs Hardware

**Hardware** is intrinsically **parallel → clock driven**
- Several FSMs can be processed in parallel using the same clock
- Evaluate signal levels of inputs at clock edge

**Software** is intrinsically **sequential → event driven**
- CPU has to process one FSM after the other
- Uses reactive system (State-event model)

### Reative System (State-Event Model)

**Event-driven:** Only evaluate the FSM if an (external) input changes.
Depending on the current state an event may have a different effect.

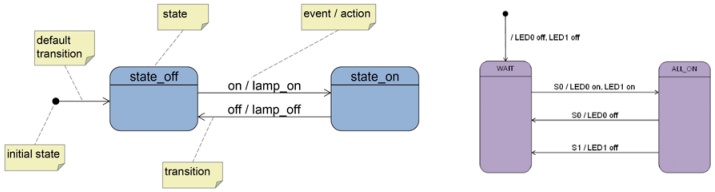
An event may or may not change the internal state and trigger actions which then influence the outside world.

#### Terminology

**State:** Internal state of the system.
**Event:** Asynchronous input that may cause a transition.
**Transition:** Triggers an action and may change the state.
Represented as a function in software.
**Action:** Output (effect) associated with a transition.

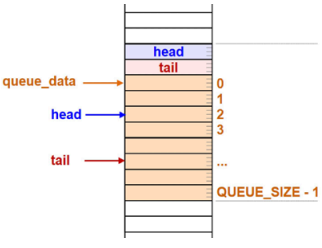
### Modeling State Machines in UML

- Every state-diagram must have an initial state.
- Each state has to be reachable through a transition.
- The state-diagram has to be deterministic




### Event queue

When incoming events are faster than processing the events.
Events are stored in a ring buffer. Avoids losing events.




```
#define QUEUE_SIZE 20

/**
 *  \brief  Defines a queue.
 */
typedef struct {
    uint32_t head;
    uint32_t tail;
    uint32_t queue_data[QUEUE_SIZE];
} queue_t;
```

## Interrupt Performance

Peripherals (Timer, SPI, UART, I2C, ADC, GPIO) signal events to firmware.
Status registers can then be addressed and read by firmware.

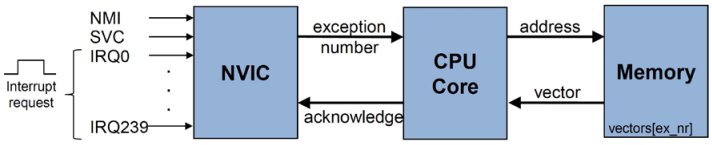
### Polling

to poll = abfragen. **Periodic Query** of Status Information in a loop,
e.g. reading status registers SPI, UART, GPIO (pins), etc.
**Synchronous** with main program.

**Advantages:** Simple, deterministic, no additional interrupt logic required
**Disadvantages:** Busy waiting, wastes CPU time,
reduced throughput, long reaction times

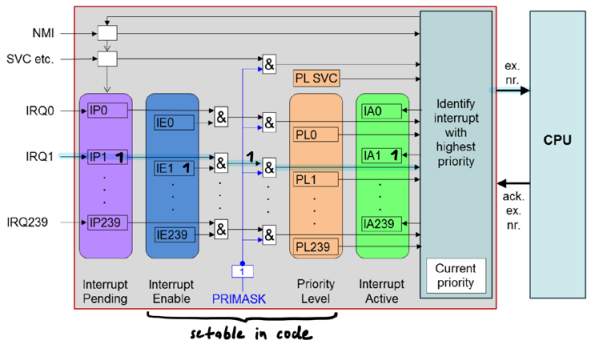
### Interrupt Driven I/O

**Interrupt System (STM32F429I)**



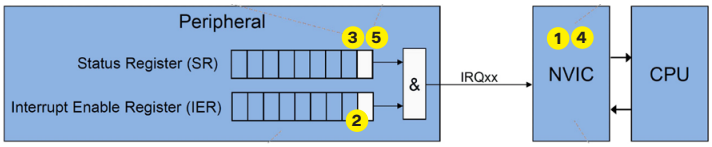

**NVIC** selects and forwards IRQ (peripheral event) with highest priority
**CPU** requests vector (ISR function pointer) from memory
**Memory** holds vector table =
addresses of ISR (Interrupt Service Routine) = function pointers
**IRQ**x = Peripherie (Timer, SPI, UART, I2C, ADC, GPIO)

### NVIC (Nested Vector Interrupt Controller)




Interrupt flags in Status Registers (SR) are hardware set and software reset.

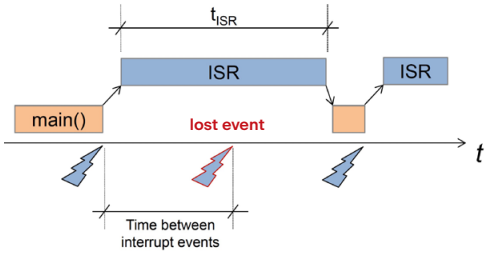
1. Software configures IRQx in NVIC (Priority level, interrupt enable)
   `NVIC->IP[IRQNUM_TIM2] = 0x10; //set priority level of timer2 to 1`
2. Software configures Peripheral and enables IRQxx through bit in the IER
   `timer->DIER |= irq; /* or disable */ timer->DIER &= ~irq;`
3. Event occurs. **Hardware** sets interrupt flag in SR → **IRQx set high**
4. NVIC triggers execution of ISR in CPU (based on vector table)
5. **Software** in ISR clears interrupt flag in Peripheral → **IRQx set low**
   `timer->SR &= (uint32_t) ~irq;`




### Interrupt Performance

Strive for short ISRs. Execute only time-critical tasks in ISR.
Move tasks with relaxed time-constraints to main loop.
Read data and do processing somewhere else.

Gründe, warum Daten verloren gehen können: Interrupt Service Time $t_{ISR}$ ist nicht immer gleich weil die Latency ist je nach Instruktion und Interrupt unterschiedlich und benötigt mehr oder weniger Clockzyklen.




**Interrupt Frequency $f_{INT}$ [Hz]**
How often an interrupt occurs.
**Interruptfrequenz = Bitrate [baud] / Zwischenspeicher [bit]**

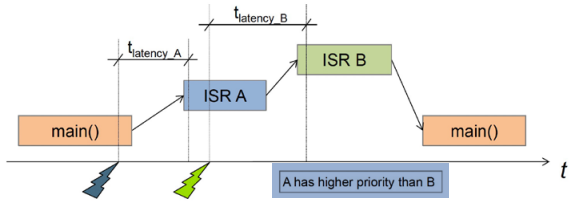
**Interrupt Service Time $t_{ISR}$ [s]**
Required time to process an interrupt (inkl. Aufruf und Rücksprung)
**Interrupt Service Time = clockcycles-per-interrupt · 1/clk [Hz]**

Depends on:
- Number of instructions
- Required number of clock cycles per instruction
- CPU clock frequency
- Context switch: Time for switching to and returning from ISR (vorheriger Kontext muss wieder hergestellt werden)

**Impact on system performance (CPU Auslastung)**

$$\text{Impact} = f_{INT} * t_{ISR} * 100\ \%$$

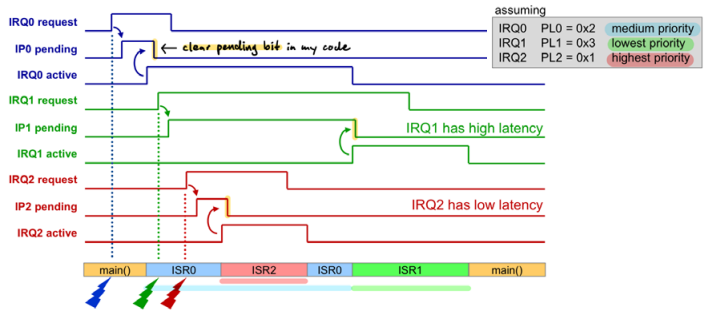
**Interrupt Latency $t_{latency}$ [s]**
Latenz (Reaktionszeit, engl. latency)
Time between interrupt event and start of servicing by ISR




Latency is influenced by
- Hardware (CPU): Herstellung Kontext
- Multi-cycle instructions on (Cortex-M3/M4)
- Wenn kleine Priorität
- Wenn interrupt kurzzeitig disabled ist (set by software)

**Low latency:** Pre-emption with appropriate priorities.
Pre-emption: Temporary interruption and suspension of a task, without asking for its cooperation, with the intention to resume that task later.

**Interrupt priorities**




$$\text{Interruptfrequenz } f_{INT} = \frac{\text{Bitrate [baud]}}{\text{Zwischenspeicher [bit]}}$$

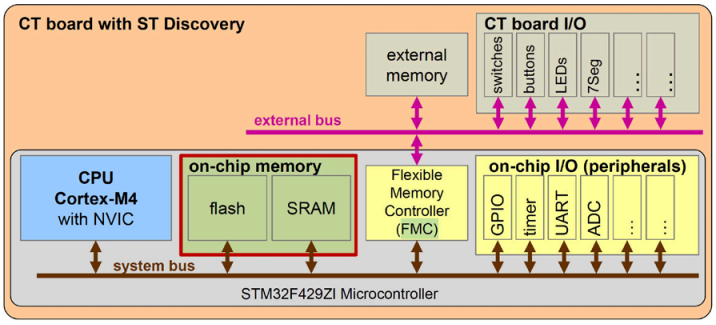
$$\text{Interrupt Service Time } t_{ISR} = \text{clockcycles-per-interrupt} \cdot \text{clk [Hz]}$$

$$\text{Impact}_{\%} = f_{INT} \cdot t_{ISR} \cdot 100\%$$

$$\text{baud [bit/s]} = \frac{\text{Zwischenspeicher [bit]} \cdot \text{Impact}}{\text{clockcycles-per-interrupt}} \cdot \text{clk [Hz]}$$

## Varia

### Simplified Model STM32F429ZI




### BIOS (Basic Input/Output System)

BIOS is a type of firmware that initializes and tests hardware components when a computer is powered on, and it loads the operating system from storage into memory. The kernel (a computer program at the core of the OS) then takes over and becomes the manager of all things.

### Complex vs Complicating

- Complex: Something being made up of many interconnected parts.
- Complicating: Something more difficult, messy, or harder to understand. Something is being made more complex / adding more complexity.

### Embedded Systems

A specialized computer system (mostly without an OS). A combination of a computer processor, computer memory, and input/output peripheral devices.

### RM0090 Reference manual

Memory Map (base adresses) : p.65
GPIO registers: p.284
GPIO register map: p.290
Index: p.1752

### Vorsätze für Masseinheiten (metric prefix)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| peta | P | $10^{15}$ | 1 000 000 000 000 000 | deci | d | $10^{-1}$ | 0.1 |
| tera | T | $10^{12}$ | 1 000 000 000 000 | centi | c | $10^{-2}$ | 0.01 |
| giga | G | $10^{9}$ | 1 000 000 000 | milli | m | $10^{-3}$ | 0.001 |
| mega | M | $10^{6}$ | 1 000 000 | micro | µ | $10^{-6}$ | 0.000 001 |
| kilo | k | $10^{3}$ | 1 000 | nano | n | $10^{-9}$ | 0.000 000 001 |
| hecto | h | $10^{2}$ | 100 | pico | p | $10^{-12}$ | 0.000 000 000 001 |
| deca | da | $10^{1}$ | 10 | femto | f | $10^{-15}$ | 0.000 000 000 000 001 |

### Hex to Dec

```
10 11 12 13 14 15
 A  B  C  D  E  F
```