CT2 Summary (beliebig lang)

22. Februar, 2024; rev. 21. Juni 2025

Linda Riesen (rieselin)

1 From Labs

NE-Signal	Startadresse	Größe
NE1	0x60000000	64 MB
NE2	0x64000000	64 MB
NE3	0x68000000	64 MB
NE4	0x6C000000	64 MB

Abbildung 1: NE1, NE2 NE3 NE4 Basisadressen

2 Vorlesung 01: Microcontroller Basics

2.1 Microcontroller

Single Chip Solution => CPU with integrated peripherals and memory

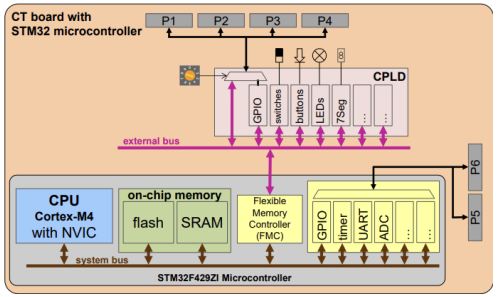

Abbildung 2: CT Board with STM32 Microcontroller and Busses

2.1.1 Signalbedeutung für Diagramm

- CLK: Takt – einfach um zu sehen, wann etwas passiert.
- A[31:0]: Adressbus – zeigt an, welche Adresse gerade angesprochen wird.
- D[31:0]: Datenbus – hier stehen die Daten, entweder lesend oder schreibend.
- NE: Chip Enable / Not Enable → aktiv bei Low (0), aktiviert das Gerät.
- NWE: Not Write Enable → aktiv bei Low (0), Schreiben
- NOE: Not Output Enable → aktiv bei Low (0), Lesen

Calculations

- Jede Adressleitung kann 2 Zustände haben (0 oder 1). => $2^{n=anz.adressleitungen}$ Bytes die Adressiert werden können

- **Peripheral:** configurable hardware block of a microcontroller (controlled through registers), accepts a specific task from the CPU, executes this task and reports back the status (e.g., task completion, error). Many peripherals are interfaces to the outside world. Examples include GPIO, UART, SPI, ADC ...

- **System Bus** connecction between cpu (acts as master (initiating and controlling all transfers), on-chip memory, peripherals (act as slaves)
  Bus Specification:
  - Protocol and Operations
  - Signals (number of signals, descriptions)
    - Address lines (unidirectional, #lines = size of adress space)
    - Data lines (8, 16, 32 or 64 parallel lines of data bidirectional (read/write))
      - if single data line for several slaves, cpu defines who drives data bus at which moment in time
    - Control signals (control direction, provide timing info)

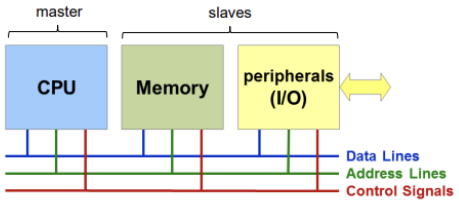

Abbildung 3: Signal Groups System Bus

- Timing (frequency, setup and hold times) (synchron (common clock)/asynchron (slaves have no access to clock, controls contain timing info))
- Electrical properties (drive strength, load)
- Mechanical requirements

- **Register**
  - Control Registers Enable the CPU to configure the peripheral (bsp: LEDs)
  - Status Registers Enable the CPU to monitor the status of the peripheral (bsp DipSW.)
  - Data Registers Enable the CPU to exchange data with the peripheral

2.2 Digital Logic Basics

CMOS **Inverter**: Transistors (CMOS = p-type (PMOS transistor) + n-type (NMOS transistor)) (Inverts 0 => 1 and 1=>0

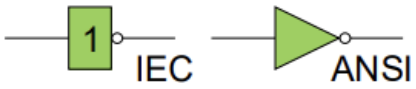

Abbildung 4: CMOS Inverter

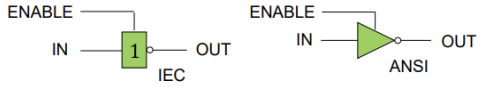

Abbildung 5: CMOS Tri-State Inverter

tristate- floating"(ist schwache, überschreibbare 1) (= no one is driving the bus) = Linie im Bus Timing Diagram
Letter 'N' prefix in signal name (Nxxx) means active-low signal
NBL (Not Byte Line) signals indicate valid bytes. NBL0 bis 3

2.3 Control and Status Registers

- Control Bits
  - Allow the CPU to configure slave
  - CPU (software) writes to register bit
  - Slave hardware uses the output of the register bit
  - Usually read/write

- Status Bits
  - Allow the CPU to monitor the state of a slave
  - Slave writes status into register bit
  - CPU (software) reads register bit
  - Usually read-only

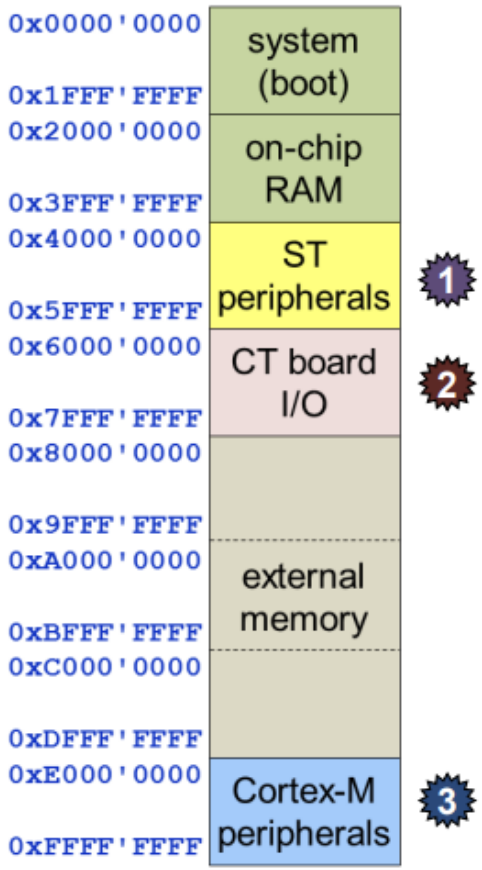

Abbildung 6: Control and Status Registers on CT Board

1. ST peripherals: e.g. Timers, ADC, UART, SPI, ...
2. CT board I/O LEDs, dip switches, LCD, ...
3. ARM Cortex-M NVIC, ...

2.4 Address Decoding (How Does a Slave Know, That It Is the Target of an Access)

1. SELECT = 1 ausgelöst von Address Decoder

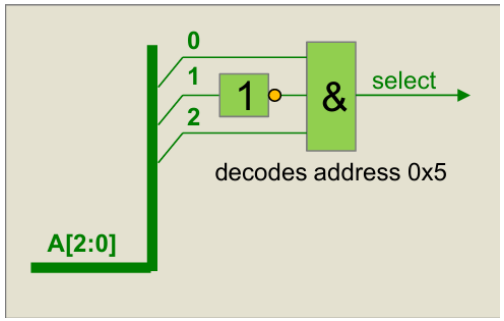

Abbildung 7: Address Decoding

2. OE / WE read_enable write_enable
3. OE / WE cycle counter muss auf 4 gezählt haben

- Full Address Decoding
  - All address lines are decoded (checked)
  - A control register can be accessed at exactly one location
  - 1 : 1 mapping (unique address maps to a single hardware)
- Partial Address Decoding
  - Only a sub-set of the address lines is decoded
  - Detects an address range or a set of addresses
  - n : 1 mapping (n unique addresses map to the same hardware register)
  - simpler decoding
  - aliasing (map a hardware register to several addresses)

2.5 Slow Slaves (Peripherals)

Problem: slowest slave defines bus cycle time
Use **Individual wait states** for slow slaves (depending on address of access / slave tells bus interface unit when its ready)

2.6 Bus Hierarchies

On chip / off chip bus (IRL even more busses)

2.7 Accessing Control Registers in C

Compiler may remove statements that have no effectfrom compilers POV (e.g. hardware access) => use volatile in variable declaration or use pointers

3 Vorlesung 02 + 03 General Purpose I/O (GPIO)

Begriffe

- Pads: the **internal** small metal contacts on the chip used for connecting the IC to external components (= doors for signals)
- Pins: the **external** contacts that connect the IC to the outside world
- Chip: a small flat piece of semiconductor material
- integrated circuit (IC): set of electronic circuits on a chip

3.1 GPIO

- often grouped in Ports ((A . . . K). Each port has 16 lines, but K has 8 lines. independent lines)
- They can be programmed to allow various possibilities
- Programming is done by reading/writing at appropriate addresses for configuration and for data
- Anleitung in F4 datasheets (documents):
  - Pin out: Pad func. connection to package pins
  - Block diagram
  - etc.
- Reference Manuals F4:
  - Register address = Base address (memory map) + Offset (ref. manual)
- GPIO have limited number of IO => needs pin sharing
  - Not all functions externally available at the same time
  - Programming of internal registers defines pin use
  - Pin / function configuration is usually static, i.e. set once at startup

3.2 Pin Functions / Configuration

- **Signal Types**:
  - **Digital**: Binary signal (1/0). (or if electrical divided into 2 regions)
  - **Analog**: Continuous signal with multiple levels (regions)
- **Configuration Registers**:
  - **Mode (GPIOx_MODER)**: Defines input/output mode.
    - 00: Input
    - 01: General purpose output mode
    - 10: Alternate function mode
    - 11: Analog mode
  - **Output Type (GPIOx_OTYPER)**:
    - 0: Push-pull (default).
    - 1: Open-drain.
  - **Pull-up/Pull-down (GPIOx_PUPDR)**: Configures pull-up/down resistors.
    - 00: No pull-up, no pull-down
    - 01: Pull-up
    - 10: Pull-down
    - 11: Reserved
  - **Speed (GPIOx_OSPEEDR)**: Controls output speed. (00 Low, 01 Med, 10 Hi, 11 Very Hi)

- **Data Registers**:
  - **Input (GPIOx_IDR)**: Read-only, holds input values.
  - **Output (GPIOx_ODR)**: Read/write, holds output values.
- **Basic I/O Configuration**:
  - Identify pin numbers.
  - Configure using GPIOx registers.

3.3 I/O Port Configurations (jeder pin kann konfiguriert werden)

- GP = general-purpose
- PP = push-pull
- PU = pull-up
- PD = pull-down
- OD = open-drain
- AF = alternate function

Data Operations :

- Read input from GPIOx_IDR.
- Write output to GPIOx_ODR or GPIOx_BSRR.

3.4 Hardware Abstraction Layer (HAL)

- Provides base addresses, register structs, and macros.

4 Vorlesung 04 Serial Data Transfer – SPI

4.1 Stuff from exercises

$$T(= Periode) = \frac{1}{Baudrate}$$

Bei **UART** der seriellen Kommunikation wird jedes Bit in der Reihenfolge von rechts nach links übertragen, d.h., zuerst das Least Significant Bit (LSB) **SPI** ist MSB

4.1.1 Instructions for Taktabweichung in UART Transmission

- Sender transmits 1 Start-Bit, 7 Data-Bits, and 1 Stop-Bit.
- Empfänger reads bits at the midpoint of each bit interval.
- The maximum allowed deviation for proper bit recognition: 0.5 bits.
- Total bits from Start-Bit to midpoint of D6: 7.5 bits (1 Start-Bit + 7 Data-Bits + 1/2 Stop-Bit).
- Calculate the maximum allowable clock deviation as:

$$\text{Taktabweichung in \%} = \frac{0.5}{7.5} \times 100 = 6.67\%$$

4.2 Serial Communication

- Connections (wires)
  - Serial data line(s)
  - Optional control lines
- Communication Modes
  - Simplex: Unidirectional, one-way only
  - Half-duplex: Bidirectional, only one direction at a time
  - Full-duplex: Bidirectional, both directions simultaneously
- Timing

  - Synchronous: Both nodes use the same clock Clock often provided by master
  - Asynchronous: Each node uses an individual clock

4.3 SPI –Serial Peripheral Interface

- Serial bus for on-board connections (short dist. communication)
- Connects micro-controller and ext. devices
- Synchronous: Master distr. clock to slaves
- im vergleich mit parallel bus: saves board area, lower pin count, simplifies Electromagnetic compatibility (EMC)
- No defined Addressing Scheme (Use of SS (KISS))
- No acknowledge/Erro detection
- Data rate highly flexible
- no flow-control available (master can delay clock edge)
- suspectible to spikes on clock line

4.3.1 SPI Basics

- Master (single master)
  - Generates clock (SCLK)
  - Initiates data transfer by setting $\bar{SS} = 0$ (Slave Select)
- MOSI -Master Out Slave In
  - Data from master to slave
- MISO -Master In Slave Out
  - Data from slave to master

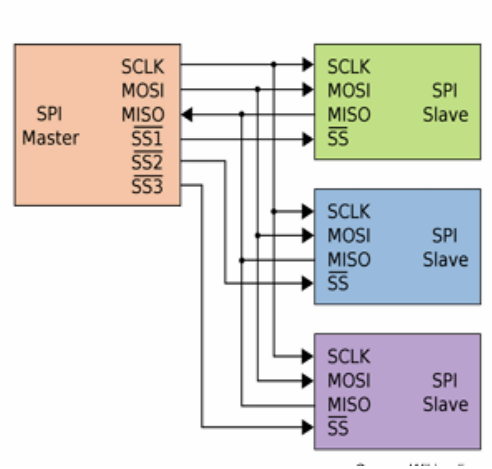

Abbildung 8: Single Master Multiple Slaves Tri-state Output

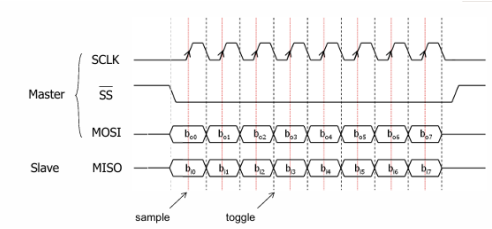

Abbildung 9: Timing: Toggle output on 1 clock edge, sample on other

4.3.2 SPI Modes

TX: provides data on 'Toggling Edge'
RX: takes over data with 'Sampling Edge'

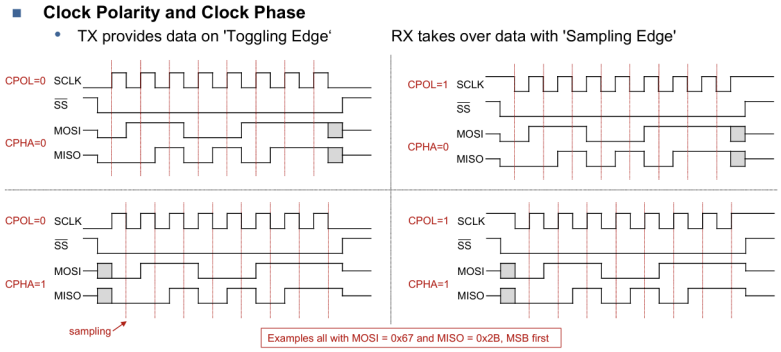

Abbildung 10: Clock Polarity and Clock Phase

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

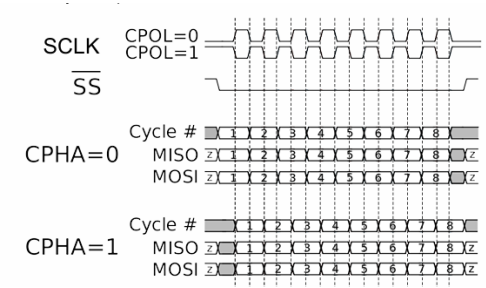

Abbildung 11: Data and Clock: All possible combinations

4.3.3 SPI – STM32F4xxx

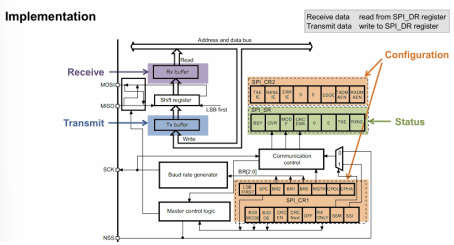

Abbildung 12: Implementation

- TXE: TX Buffer Empty: Software can write next TX byte to register SPI_DR
- RXNE RX Buffer Not Empty: A byte has been recieved. Software can read it from SPI_DR

4.3.4 SPI – Flash Devices

Signal Name	Function	Direction
C	Serial clock	Input
DQ0	Serial data	Input
DQ1	Serial data	Output
S#	Chip select	Input
W#	Write Protect	Input
RESET#	Reset	Input
V_{CC}	Supply voltage	–
V_{SS}	Ground	–

Abbildung 13: Flash Drives: Save board Area Signals

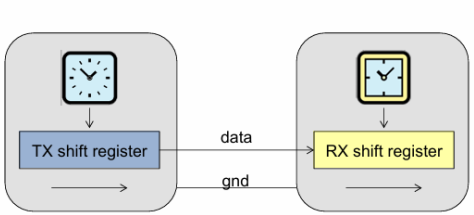

Abbildung 15: TX and RX

5 Vorlesung 05 Serial Data Transfer - UART / I2C

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
The three interfaces provide the lowest layer of communication and require higher level protocols to provide and interpret the transferred data.		

Abbildung 14: 3 simple serial Communication Protocols (in Microcontrollers)

5.1 Asynchronous Serial Interface

5.1.1 Universal Asynchronous Receiver Transmitter (UART)

UART is asynchronous. No clock is transmitted, which saves pin/cable costs. However, it is more prone to drift and resynch is needed after every item (several bits)

- Shift registers with diverging clockc sources: Require synchronization at start of each data item in receiver
- no clock is transmitted but each device has own clock => leads to drifts
- To sync: use start and stop bits (which generates an overhead)
- Each data item (5-8 bits) requires synchronization

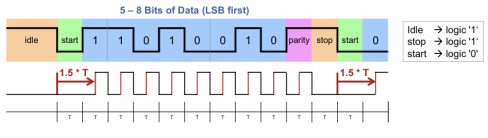

Abbildung 16: UART Timing

- Transition stop (’1’) => start (’0’)
- Reciever detects edge at the start of each data block
- allows reciever to sample data in the middle of bits
- clocks have to be accurate enough to allow sampling up to parity bit

5.1.2 Longer Distances: RS-232 / RS-485 - Electrical Characteristics

Longer connections require line drivers

RS-232 (Interconnecting Equipment by Cable) Use **Ground** as common reference level for all signals, use cable for simple bidirectional point-to-point interface, allows transmissions up to ca. 10m

RS-485 (Differential Tramsmission based on UART) Noise in the line can lead to many errors => use differential signals (less susceptible to disturbances => longer distances)

- Transmit and receive share the same lines
- Multi-point communication
- Half-duplex

Differential Signal: Form difference of 2 line signals (noise affects both and in difference mostly cancels itself out)

5.1.3 U(S)ART - STM32F4xxx

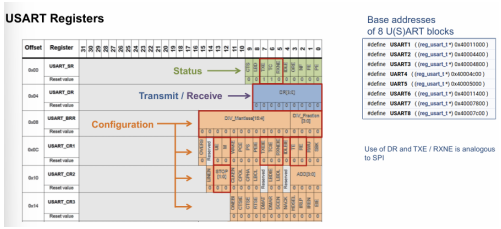

Abbildung 17: USART Registers

5.2 I2C Bus (Inter-Integrated Circuit (I-squared-C)) MSB

5.2.1 Protocol / Operation

The Inter-Integrated Circuit (I2C) bus is a synchronous, half-duplex, multi-master (is possible), multi-slave communication protocol that uses two bidirectional lines:

- **SCL (Serial Clock Line)**: Provides the clock signal. Driven through Master
- **SDA (Serial Data Line)**: Transfers data between devices. Data is driven by Master or by addressed slave, change of data is only allowed when scl is low

Well suited for connection of multiple boards

- 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)

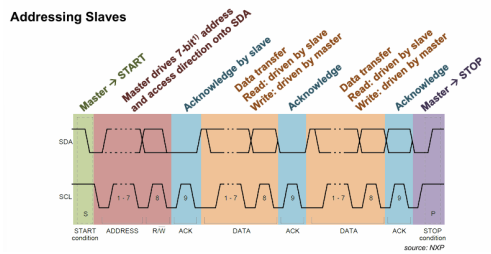

Abbildung 18: Addressing Slaves Example

5.2.2 I2C - STM32F4xxx

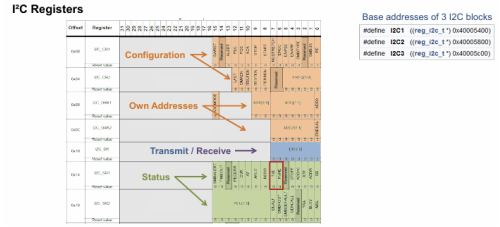

Abbildung 19: I2C Registers

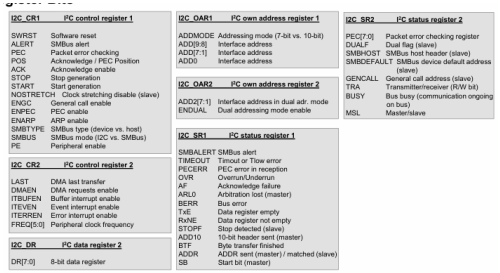

Abbildung 20: I2C Register Bits

5.3 Comparision

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
The three interfaces provide the lowest layer of communication and require higher level protocols to provide and interpret the transferred data.		

Abbildung 21: UART vs SPI vs I2C

6 Vorlesung 06 Timer / Counter

Timer: counting clock cycles or processor cycles (periodic) Counter: counting events

App Examples

- Trigger for periodic software tasks
- Count number of pulses on input pin
- Measure time between rising edges of an input pin
- Generate defined sequence of pulses on an output pin
- A state machine that moves to the next stage at each event. The stages represent (say) successive numbers. The number of stages is limited, therefore there is a limit to the counter (max number)
- Also for counting other peripheries

6.1 Function

16-bit / 32-bit counter register: Increment / decrement at every tick

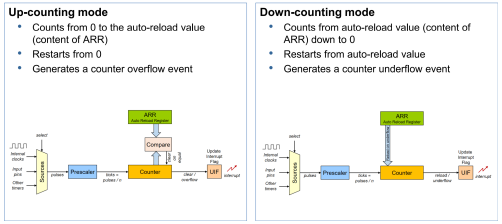

Abbildung 22: Up-Counting / Down-Counting Mode

- 16-bit counter 0, 1, 2, ... 65’535
- 32-bit counter 0, 1, 2, ... 4’294’967’295

Counter und ARR immer glecih Lang

6.2 Prescaler

Increase counting range: Count only every n-th event e.g. n = 1, 2, 4, 8, 32, 64 , . . .

6.3 ST32F4xx Timers

microcontroller has several timers

Timers TIM2-TIM5

- **Bit-width:**
  - 16-bit: TIM3 and TIM4
  - 32-bit: TIM2 and TIM5
- **Counting Modes:**
  - Up, down, up/down

  - Auto-reload

- **Prescaler:**
  - 16-bit programmable prescaler
  - Divides counter clock frequency by a factor between 1 and 65536
- **Independent Channels:** (Up to 4)
  - Input capture
  - Output compare
  - PWM generation
  - One-pulse mode output
- **Synchronization Circuit:**
  - Control timer with external signals
  - Interconnect several timers together
- **Interrupt/DMA Generation:**
  - Based on several events

6.4 Timer Configuration

See Manual (also Slides p.17-29)

6.5 Input Capture

Measuring intervals => pulse lengths and periods: Count ticks between timer start and an event

6.6 Pulse-Width-Modulation (PWM)

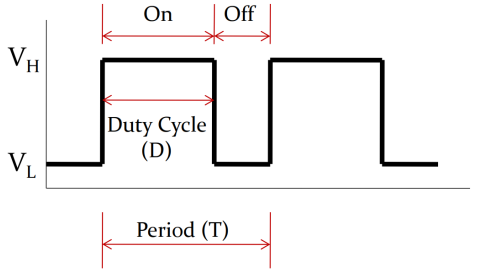

Abbildung 23: Duty Cycle - Definition

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

Average signal: $V_{avg} = D * V_H + (1 - D) * V_L$

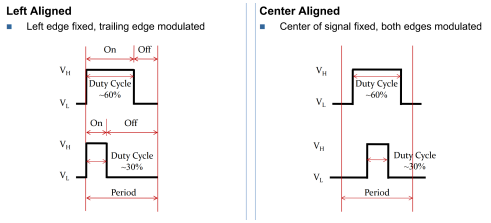

Abbildung 24: PWM Left Aligned / Right Aligned

6.7 Output Compare – Generating PWM Signals

Toggle output pin when counter reaches CCR

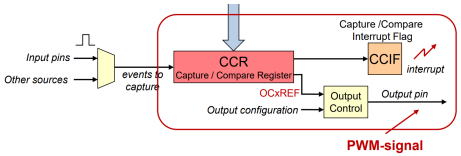

Abbildung 25: Output COmpare (Below Counter)

6.8 Capture / Compare Configuration

- Select counter clock (internal, external, prescaler)
- Write desired data to TIMx_ARR register => defines common period of PWM signals
- Write desired data to TIMx_CCRx registers => defines duty cycles of PWM signals
- Set CCxIE bits if interrupts are to be generated (in TIMx_DIER register)
- Select the output mode (registers CCMRx / CCER)
- Enable counter by setting the CEN bit in the TIMx_CR1 register

7 Vorlesung 07: ADC / DAC

DAC: Kein Prüfungsstoff Ziel: Digital zu Analog übersetzen und umgekehrt

7.1 Analog zu Digital: Samplen: ADC

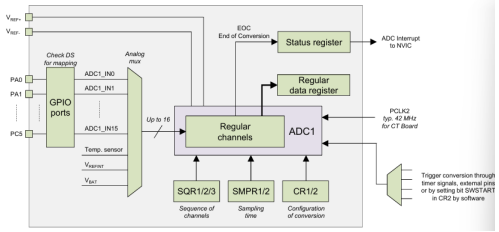

Abbildung 26: ADC Diagram Simplified

- Converts input signal (voltage) to a digital value (N-bit)
- Conversion results in one of 2N (N = Resolution = Size of Digital Word) possible numerical levels: Quantisierung
- Raw input signal can be dynamic1) or static2)
  - Dynamic signal (green) sampled at specific time intervals
  - Samples transformed into series of discrete values (blue)

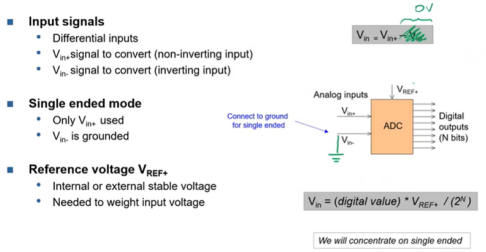

Abbildung 27: Single Ended ADC

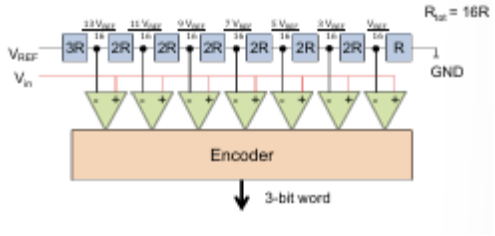

Abbildung 28: Flash ADC

- LSB (Least Significant Bit): 1 LSB $\triangleq \frac{V_{REF}}{2^N}$
- Full Scale Range (FSR): Range between analog levels of minimum an dmaximum digital codes = V_{FSR} is one LSB Less than V_{REF}

7.2 DAC: Digital to Analog Converter

$V_{Out} = (\text{digital value}) * \frac{V_{REF}}{2^N}$

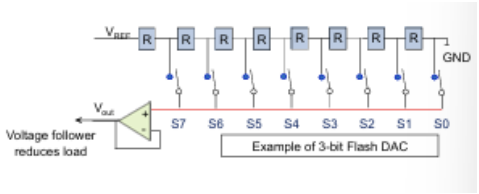

Abbildung 29: Flash DAC

7.3 Successive Approximation Register (SAR) ADC

SAR ADC as a (cheaper!) alternative to the previously introduced Flash ADC

- Approach V_{in} with successive division by 2
- Binary search: Start with half the digital value $MSB = 1$, all other bits at 0
- DAC generates analog value V_{DAC} that is compared to V_{in} If $V_{DAC} < V_{in}$ => keep MSB at 1, otherwise set MSB to 0
- Continue with other N bits in same way (N steps)

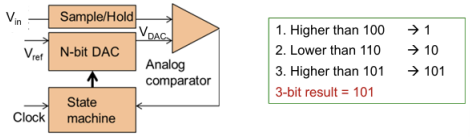

Abbildung 30: SAR ADC

Abbildung 31: ADC Characteristics

Begriffe

- Sampling rate
  - Input signal sampled at discrete points in time => discontinuities
  - Should be at least twice the highest frequency component of input signal (Nyquist–Shannon sampling theorem)
- Conversion time
  - Time between start of sampling and digital output available
  - Programming a higher resolution may increase conversion time
- Monotonicity
  - Increase of Vin results in increase or no change of digital output and vice-versa

7.4 ADC Types of Errors

7.4.1 Quantization error

- Analog input is continuous: Infinite number of states
- Digital output is discrete: Finite number of states
- Introduces an error between -0.5 LSB and +0.5 LSB

Nicht reversibel!

7.4.2 Offset Error (= zero-scale error)

- Deviation of real N-bit ADC from ideal N-bit ADC at input point zero
- For an ideal N-bit ADC, the first transition occurs at 0.5 LSB above zero
- Can be corrected using the microcontroller

7.4.3 Gain Error

- Indicates how well the slope of an actual transfer function matches the slope of the ideal transfer function
- Expressed in LSB or as a percent of full-scale range (%FSR)
- Calibration with hardware or software possible
- full-scale error = offset error + gain error

7.5 ADC Conversion 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.

Abbildung 32: ADC Conversion Modes

7.6 ADC Timing

Depends on time for sampling and conversion: $T_{total} = T_{sample} + T_{conv}$ [mit conv: Pro bit res. 1 cycle, sample: einstellbar zw. 3 und 480 cycles]

given: APB2 clock = 48 MHz
Prescaler 2 → ADCCLK = 24 MHz
3 cycles sampling time
12 bit resolution

T_{total} = (3 + 12) * 1/ 24 MHz = 0.625 us

sampling rate < 1 / T_{total} = 1.6 Msps

Abbildung 33: ADC Rechnungsbeispiel Timing

8 Vorlesung 09: Memory (p.1-24)

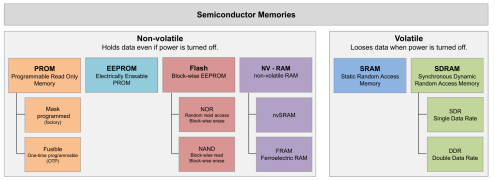

Abbildung 34: Memory Technology Comparison

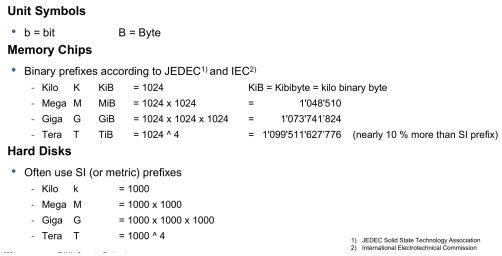

Abbildung 35: Units Symbols, Memory Chips, Hard Disks

Memories Are Arrays of Bit Cells Memory Architecture => n x m array: n words with m data bits

8.1 PROM - Programmable Read Only Memory

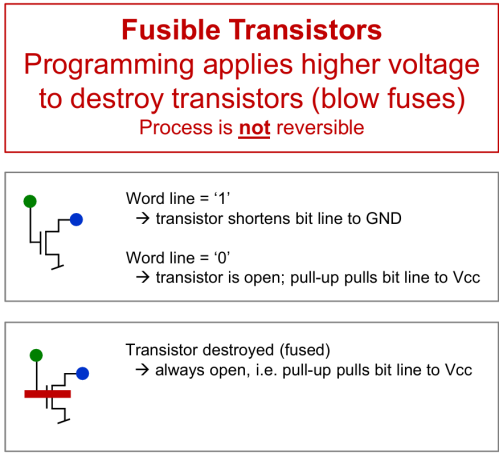

Abbildung 36: PROM Logic

8.1.1 EEPROM and Flash Memory (Making PROMs Reprogrammable)

Replace Fusing by Floating Gate Transistor

- Write cell to '0' => ON
  - High voltage (Up) deposits charge on the floating gate (Isolated by e.g. SiO2 (Silicon Dioxide)
  - Transistor ON (conducting) if control gate = '1'
- Write cell to '1' => OFF
  - Discharge floating gate (with negative Up)
  - Transistor is OFF = blocking independent of value on the control gate

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

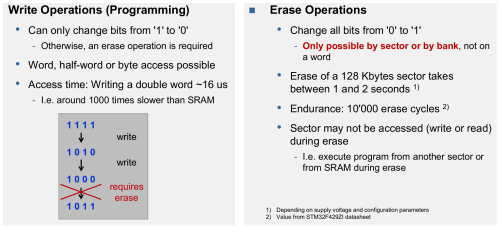

Abbildung 37: Write / Erase Operations

	NOR Flash	NAND Flash
Topology		
Applications	• Execute code directly from memory • Persistent device configurations (replacement of EEPROM)	• File-based IO, disks • Large amounts of sequential data (images, SD cards, SSD) • 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 • Types with serial interface (SPI) available	• Special NAND flash interface • Error correction for defective blocks
Access	• Random access read ~0.12 µs • **Writing individual bytes possible** • Slow writes ~180 µs / 32 Byte	• **Slow random access** read: 1. Byte 25 µs, then 0.03 µs each • **Writing of individual bytes difficult** • **Fast block write** ~300 µs / 2'112 Bytes

Abbildung 38: NOR vs NAND Topology

8.2 SRAM –Static Random Access Memory

flip-flop (latch) based cells

Abbildung 39: SRAM Reading and Writing

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	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 $\overline{CS}$

Abbildung 40: SRAM Logic Async

8.3 SDRAM – Synchronous Dynamic RAM

Information stored as charge in capacitor, Row and column addresses multiplexed

- High integration
  - Large memories at low cost
  - Allows to store large amounts of data
- Leakage current => Loss of charge
  - Capacitor holds charge only for a few milliseconds
  - Charge has to be refreshed periodically => dynamic
  - Refresh logic usually located on SDRAM device

8.4 SRAM vs SDRAM

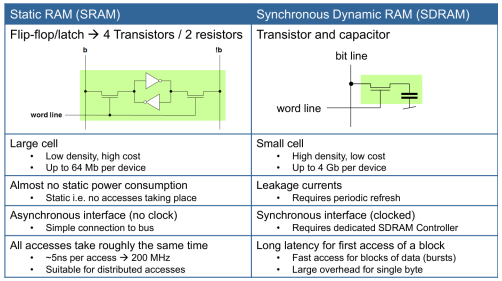

Abbildung 41: SRAM vs SDRAM

9 Vorlesung 08: Memory (p.32-63)

9.1 Von Praktika

- **Stuck-at Fault**: Kurzschluss zu Ground oder der Versorgungsspannung
- **Bridge Fault (shorted)**: Kurzschluss zwischen zwei nebeneinanderliegenden Leitungen, d.h. die Werte auf beiden Leitungen sind identisch
- **Open**:Ein Unterbruch in der Signalleitung. Ein vom Microcontroller ausgegebener Wert erreicht den Speicher nicht. Die betreffende Leitung liegt auf einem undefinierten Pegel.

9.2 Mirroring

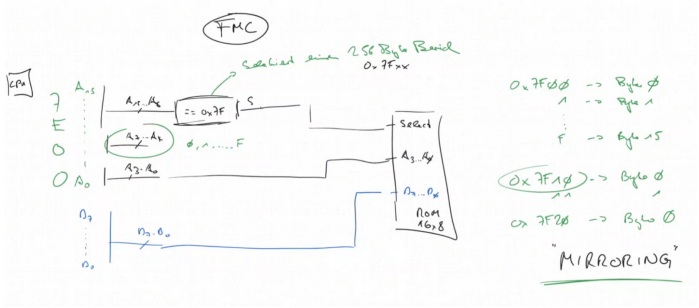

Abbildung 42: Mirroring: Ensteht aus ignorierten addresslinien

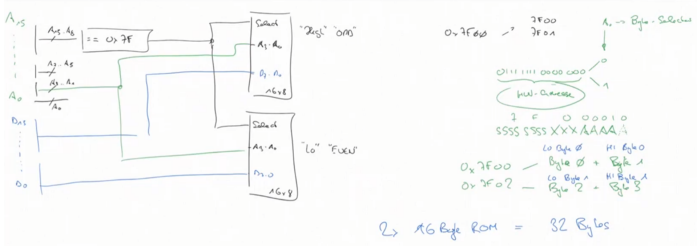

Abbildung 43: Adressing High and Low (Odd and Even) at the same time

9.3 FMC: Flexible Memory Controller:

Internen Bus nach aussen anzeigen

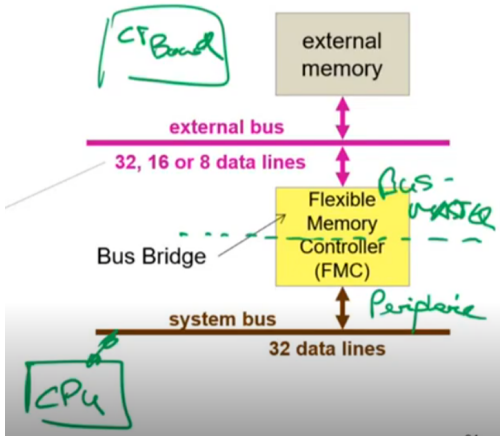

Abbildung 44: FMC Bridge between system bus and external bus (slave on system, master on ext.)

9.3.1 FMC for SRAM

Prefix 'N' → active-low signal

FMC signal name	I/O	Function
A[25:0]	OUT	Address bus
D[31:0]	INOUT	Data bidirectional bus
NE[4:1]	OUT	Four enable lines [1)]
NOE	OUT	Output enable
NWE	OUT	Write enable

Abbildung 45: Signale die FMC genereiert wenn SRAM angehängt wird

Different Number of Data Lines Causes Bottleneck: E.g. 32-bit (word) access to 8-bit external memory

Implementation FMC

- CPU write to memory
  - Address and data stored in FMC FIFO buffer

  - Avoids wait for slow memory
  - Free system bus for other accesses

- CPU read from memory
  - System bus has to wait until external memory device provides data = Called **Wait State**

9.4 Memory Banks

- ST defined address ranges for each type of memory
- Organized in 6 banks
- Each bank allows connection of 4 devices
- Pins are multiplexed: Not possible to fully use all the banks simultaneously

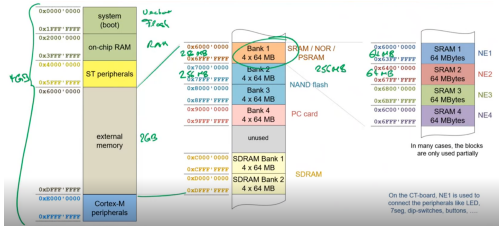

Abbildung 46: Memory Banks Zoom in

9.5 Configuration FMC

Timing, size of bus, etc.

FMC Möglichkeiten

- SRAM and NOR flash
- Synchronous DRAM
- NAND flash

10 Vorlesung 10: Quiz

11 Vorlesunng 11: Cache

DRAM ist nur schnell wenn Block zugriffe (ganze Memory zeilen) => use cache! Cache ist unsichtbar für CPU => denkt es kommt direkt vom Memory => kein Umschreiben nötig Cache nutzt Blöcke = N bytes (= N anz. blockzeilen) Cache is "guessing"which blocks the CPU will need next

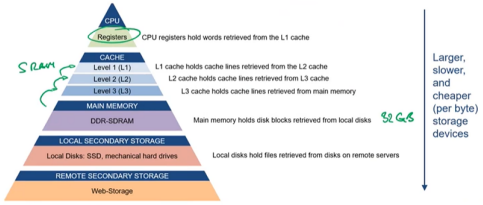

Abbildung 47: The Memory Hierarchy

Je näher am Register desto kleiner der Cache => Auswahl nötig!

11.1 Principle of Locality:

Programs usually access small regions of memory in a given interval of time: "Örtliche Nachbarschaft"

- current data (also instruction!) location is likely being close to next accessed location
- current data (also instruction!) location is likely being accessed again in near future

11.2 Cache Mechanics

11.2.1 Memory Blocks

- Blocks of main Memory copied to faster cache memory
- Use principle of locality :

- temporal: supported by default
- spatial: supported because of block transfers

11.2.2 Cache hit

Daten verlangt von Block der bereits im Cache ist => kann sofort bedient werden = Ziel!

11.2.3 Cache miss

Daten verlangt von Block der nicht bereits im Cache ist => wait state bis block geholt wird => welcher Block wird im Cache ersetzt? => depends on Cache strategy

- Cold miss
  - First access to a block
- Capacity miss
  - Working set larger than cache => the cache cannot hold all the data required by the system
- Conflict miss
  - Multiple data blocks map to same line or set => blocks compete for the same line or set in the cache

11.3 Cache Organization

Chache is organized in m lines

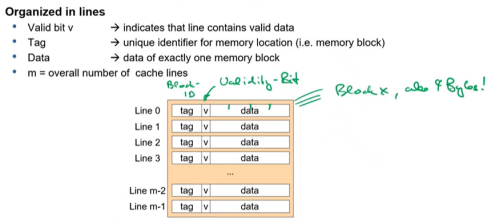

Abbildung 48: Cache Organization

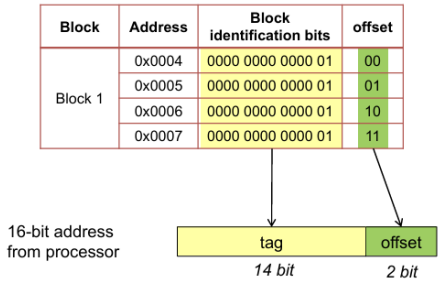

Abbildung 49: Fully Associative Addressing => offset depends on cache size

11.3.1 Fully Associative

Any memory block can be at any line = fully associative (tag contains complete block identification)

11.3.2 Direct Mapped

Reduces flexibility but simplifies hardware => billig Each memory block is mapped to exactly on cache line (multiple blocks mapped to the same line)=> $LineNr = BlockNr \mod m$ => multiples können nie gleichzeitig in cache sein

Abbildung 50: Direct Mapped Addressing

11.3.3 N-Way Set Associative (mischung aus above)

Partition into sets: s= m/n number of sets, n = lines per set, b = bytes per line

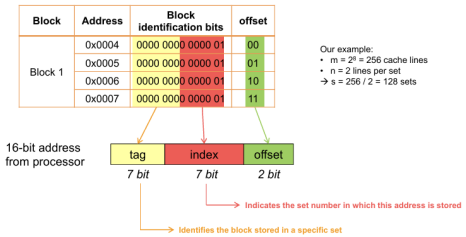

Abbildung 51: N-Way Set Associative Addressing (index = set number)

11.4 Performance

- Hit rate / miss rate: Fraction of memory references found / not found in cache
  - hit rate = nr_of_hits / nr_of_accesses
  - miss rate = nr_of_misses / nr_of_accesses = 1 – hit rate
- Hit time: Time to deliver a block in the cache to the processor
- Miss penalty: Additional time required to fetch data from memory because of a cache miss

11.5 Replacement Strategies

typically **hardcoded in hardware** für LRU/LFU/FIFO => statistic führen => oft is random sogar gleich gut

- **LRU (Least Recently Used)**: Replace the line that was used least recently.
- **LFU (Least Frequently Used)**: Replace the line used least often.
- **FIFO (First In–First Out)**: Replace the oldest cache line.
- **Random Replacement**: Randomly select a line to replace.

11.6 Write Strategies

11.6.1 On Write Hit (data already in cache)

- **Write-through**: Write immediately to main memory.
- **Write-back**: Write to memory only when the cache line is replaced (requires a *dirty* or *valid* bit).

11.6.2 On Write Miss (data not in cache)

- **Write-allocate**: Load block from memory into cache, then write to cache.
- **No-write-allocate**: Write directly to memory, without loading data into cache.

12 Vorlesung 12: Software State Machine

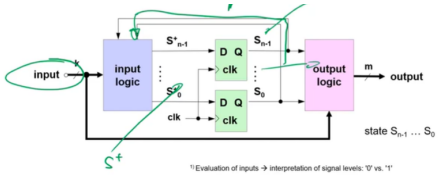

Abbildung 52: Finite State Machine (FSM) (begrenzte Anz. zustände = Finite)

Types of FSM:

- Moore: Input signals influence state only
- Mealy: Input signals influence state AND output signals

Hardware is intrinsically parallel (software sequential) => FSM in Software:

12.1 Reactive System => State-event model

- **responds to external events** (= event driven, only evaluate if input changes (hardware immer evalutate))

- **internal state** => memory of what happened before
- **Actions** (influence outside world)
- **Event** kann / kann nicht internal state ändern / actions triggern
- Every State Machine must have an inital state!
- Inputs without effect are omitted (e.g. in diagram)
- alle states müssen mind. 1x erreichbar sein

12.2 Interaction of 2... FSMs

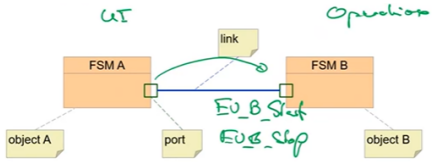

Abbildung 53: Interaction of FSMs

Event queue

- Collect events generated by different objects
- Buffered in event queue: Avoids losing events
- FSM processes one event after the other
- Events are deleted after processing

13 Vorlesung 13: Interrupt Performance

13.1 Polling

Synchronous with main program

Advantages

- Simple and straightforward
- Implicit synchronization
- Deterministic
- No additional interrupt logic required

Disadvantages

- Busy wait => wastes CPU time
- Reduced throughput
- Long reaction times
- in case of many I/O devices or if the CPU is working on other tasks

13.2 Interrupt Driven I/O

NVIC Peripherie, enabled mit Bits (Enable), has Priority levels

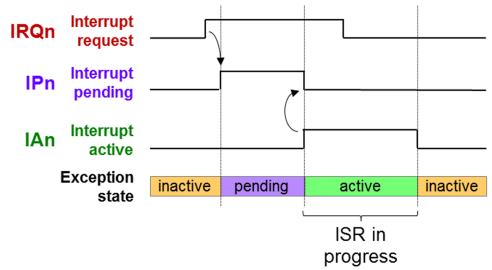

Abbildung 54: Interrupt Process

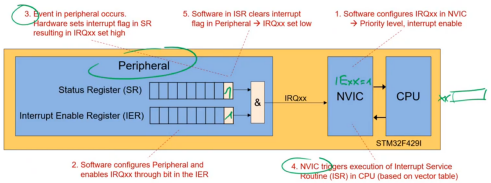

Abbildung 55: Interrupt Handler: Hardware set, software reset

13.2.1 Interrupt Performance

Interrupt frequency: $f_{INT} = 1/s = Hz = s^{-1} = Baud$ How often does an interrupt occur, Varies from source to source

Interrupt service Time: t_{ISR} Required time to process an interrupt: Depends on: #of instructions in ISR, Required #of clock cycles per instructions, clock frequency, switching time

Impact on system performance Percentage of CPU time used to service interrupts

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

$t_{\text{ISR}} > t_{\text{INT}}$ **— Time between two interrupt events**

- Some interrupt events will not be serviced (=> data loss)
- f_{INT} as well as t_{ISR} may vary over time => even if average is ok interrupts may still be lost
- Caution in case of several interrupt sources
  - Interrupts can occur simultaneously
  - Computing power required for both ISRs
- => strive for short ISRs: Execute only time-critical tasks in ISR
  - Move tasks with relaxed time-constraints to main loop
  - Makes ISR available for other time-critical tasks
  - Often simply feed interrupt to queue; dispatch queue in main loop

13.3 Interrupt Latency

Time between interrupt event and start of servicing by ISR

- Varies a lot: from about 50 nanoseconds (ns) up to several milliseconds (ms)
- From about 50 nanoseconds (ns) up to several milliseconds (ms)
- Latency influenced by **hardware** (CPU): versch. instructions have different exec. times (some are abandoned & restarted after ISR, some are interrupted & resumed after ISR)
- Latency influenced by **software**: saving addit. registers on stack, process ongoing or higher prio ISRs, disabled Interrupts => nur kurze abschnitte!
- f_{INT} too high => zu viele interrupts => keine CPU mehr übrig für data processing
- Certain peripherals require fast response (< 1us)
- Pre-Emption: temporary interruption and suspension of a task without asking for its cooperation => Priority levels programmed in NVIC
- Some applications can tolerate considerable interrupt latency as long as it is **consistent**

13.4 Managing Latency

- High priority interrupts may cause high latencies for lower priority interrupts
  - Remedy: Move 'waiting loop' to main program => use FIFO queue
  - Move non-time-critical work from ISRs to main loop

13.5 Interrupt Driven FSM

Events trigger interrupts => enter events into queue => FSM in main loop reads events from queue

14 Interrupt-Latenz: Definition, Ursachen und Messung

14.1 Begriff der Interrupt-Latenz

Die **Interrupt-Latenz** ist die Zeitspanne zwischen dem Eintreten eines Interrupt-Ereignisses (z. B. ein externer Signalwechsel) und dem Beginn der Ausführung der zugehörigen Interrupt-Service-Routine (ISR). Diese Latenz ist ein entscheidender Parameter in zeitkritischen Echtzeitsystemen, da sie bestimmt, wie schnell das System auf externe Ereignisse reagieren kann.

14.2 Ursachen für Interrupt-Latenz

Mehrere Faktoren können die Interrupt-Latenz beeinflussen oder verlängern:

- **Globale Interrupts deaktiviert:** Wenn Interrupts im System global deaktiviert sind (z. B. durch eine CLI-Instruktion), kann ein eintreffender Interrupt erst bearbeitet werden, wenn Interrupts wieder aktiviert werden.
- **Laufende ISR eines anderen Interrupts:** Falls gerade eine andere ISR ausgeführt wird, muss der neue Interrupt warten, bis die laufende ISR abgeschlossen ist (sofern keine verschachtelten Interrupts erlaubt sind).
- **Interrupt-Priorität:** Interrupts mit niedrigerer Priorität müssen warten, wenn ein höher priorisierter Interrupt bereits in Bearbeitung ist.
- **CPU-Zustand:** Pipelining, Caching, Kontextwechsel oder CPU-Sperren (z. B. durch Mutexe) können ebenfalls die Latenz erhöhen.

14.3 Messung der Interrupt-Latenz

Um die Interrupt-Latenz zu messen, kann ein einfaches Programm realisiert werden, das wie folgt funktioniert:

1. Ein externer Trigger (z. B. ein GPIO-Pin) löst einen Interrupt aus.
2. Zeitmarke t_0 wird bei Eintreffen des Triggers gesetzt (z. B. mithilfe eines Oszilloskops oder eines Timers).
3. Die erste Instruktion in der ISR setzt eine weitere Zeitmarke t_1 (z. B. durch Umschalten eines anderen GPIO-Pins).
4. Die Differenz $\Delta t = t_1 - t_0$ stellt die Interrupt-Latenz dar.

14.4 Einfluss höher priorisierter Interrupts

Wenn während der Wartezeit ein anderer Interrupt mit höherer Priorität ausgelöst wird, wird dessen ISR zuerst ausgeführt. Das bedeutet, dass die eigentliche ISR des ursprünglich ausgelösten Interrupts verzögert startet. Dies verlängert die effektive Interrupt-Latenz für diesen Interrupt. In Systemen mit vielen gleichzeitig aktiven Interruptquellen ist es daher wichtig, eine geeignete Prioritätssteuerung und Interruptverschachtelung zu implementieren.

Zusammenfassung: Die Interrupt-Latenz ist ein kritisches Zeitmaß in Echtzeitsystemen. Sie kann durch Systemzustände, Interruptprioritäten und ISR-Ausführungszeiten beeinflusst werden. Eine sorgfältige Systemgestaltung und präzise Messung sind essenziell, um diese Zeitspanne zu minimieren und vorhersagbar zu machen.