Kapitel 1 Modular Coding/Linking

Low Module Coupling: Die Kopplung zwischen Modulen soll möglichst klein sein, d.h. die Abhängigkeit zwischen Modulen soll minimiert werden. Ein Modul verwendet wenige Funktionen anderer Module.

High Module Cohesion: Der funktionale Zusammenhang innerhalb eines Moduls soll hoch sein. Das Modul ist für eine klar definierte Aufgabe zuständig und hat eine schmale Schnittstelle nach aussen. Die im Modul zusammengefassten Funktionen haben einen klaren Zusammenhang und bilden eine Einheit.

Information Hiding – Geheimnisprinzip: Die innere Struktur eines Moduls ist vor anderen Modulen verborgen. Interface und Implementation werden getrennt. Die Implementation eines Moduls kann jederzeit geändert werden ohne andere Module zu beeinflussen.

Deklaration: Spezifiziert wie ein Name verwendet werden kann.

```
uint32_t square(uint32_t v); // square function defined elsewhere
extern uint32_t counter;     // counter variable defined elsewhere
struct S;                    // struct S type defined elsewhere
```

Definition: Funktion wird mit ihrem Rumpf definiert. Speicher wird einer Variable zugewiesen. Ein Strukturtyp mit seinen Mitgliedern.

```
uint32_t square(uint32_t v) { ... } // square function definition
uint32_t counter;                   // counter variable definition
struct S { ... };                   // struct S type definition
```

Header Files: Deklaration von Header Files und Verwendung in anderen c Programmen.

```
// square.h
#ifndef _SQUARE_H_ // incl.-
#define _SQUARE_H_ // guard

// declaration of square
uint32_t square(uint32_t v);

#endif // end of incl.-guard
```

```
// square.c
#include "square.h"

// definition of square
uint32_t square(uint32_t v)
{
  return v*v;
}
```

```
// program_A.c
#include "square.h"
int main(void) {
  res = square(a) + b;
  ...
}
```

```
// program_B.c
#include "square.h"
int main(void) {
  y = square(x);
  ...
}
```

Konzept der Verknüpfungen in C (linkage):

External linkage: - Funktion oder globale Variable; - Der globale Name steht extern zur Verwendung in verschiedenen Modulen zur Verfügung.

Internal linkage: - Funktion oder eine globale Variable (mit dem Schlüsselwort „**static**"); - Der globale Name ist nur intern für die Verwendung in diesem Modul verfügbar.

No linkage: - Jeder Name, der sich nicht im globalen Raum befindet.

```
// square.c
...
uint32_t square(uint32_t v) {
  return v*v;
}
```

```
// main.c
#include "square.h"
static uint32_t a = 5;
static uint32_t b = 7;
int main(void) {
  uint32_t res;
  res = square(a) + b;
  ...
}
```

square = external linkage

a = internal linkage
b = internal linkage
main = external linkage
res = no linkage
square = external linkage[1)]

Import und Export Symbole

- Linkage control
  - EXPORT declares a symbol for use by other modules
  - IMPORT declares a symbol from another module for use in this module
- Internal symbols
  - Neither EXPORT nor IMPORT
  - Defined in this module
  - Can only be used within this module

```
// main.c
#include "square.h"
static uint32_t a = 5;
static uint32_t b = 7;
int main(void) {
  uint32_t res;
  res = square(a) + b;
  ...
}
```

usable outside of module main

```
; main.s
      AREA myCode,CODE,READONLY
      EXPORT main
      IMPORT square
main  PROC
      LDR     r0,a_adr
      LDR     r0,[r0,#0]   ; a
      BL      square
      ...
      ENDP
a_adr DCD     a
b_adr DCD     b

      AREA myData, DATA
a     DCD     0x00000005
b     DCD     0x00000007
```

from module square

internal symbols

Von Assembler Symbolen zu Object-File Symbolen

Referenzen - Importierte Symbole aus dem Assembler-Code werden in globale Referenzsymbole in der Objektdatei übersetzt.

Global - Exportierte Symbole aus Assembler-Code werden in globale Symbole in der Objektdatei übersetzt.

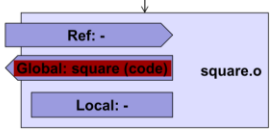
Lokal - Interne Symbole aus dem Assembler-Code werden übersetzt zu lokalen Symbolen in der Objektdatei übersetzt.

Beispiel Assembler-Code zu Object-File

- Exported
  - Code symbol **square**
- Referenced/Imported
  - None
  - No external symbol used
- Local
  - None
  - No internal symbol defined

```
; square.s
      AREA myCode,CODE,READONLY
      EXPORT square
square PROC
      MOV     r1,r0
      MULS    r0,r1,r0
      BX      lr
      ENDP
      END
```

Assemble

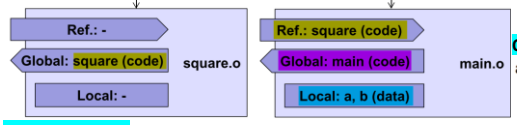


Beispiel C-Code zu Object-File

```
// square.c
uint32_t square(uint32_t v)
{
  return v*v;
}
```

```
// main.c
uint32_t square(uint32_t v);
static uint32_t a = 5;
static uint32_t b = 7;
int main(void) {
  uint32_t res; //stack or reg
  res = square(a) + b;
}
```

Preprocess, compile and assemble



Linker Aufgaben

Fügt mehrere Objectfiles zu einem ausführbaren Objektfile zusammen (Merge Sections), löst die Symbol Referenzen zwischen den einzelnen Objektfiles auf (Resolution) und passt Zugriffe auf Symbole an (Relocation).

Ergebnis: Ausführbares Objektfile (Executable)

- Codeabschnitte (CODE) zusammenführen; - Datenabschnitte (DATA) zusammenführen; - Symbolauflösung Verweise auf andere Module; - Adressverschiebung Anpassung an neue Positionen von Symbolen

Linker Input → Object-Files

Code - Code und konstante Daten des Moduls, basierend auf der Adresse 0x0

Data – Alle globalen Variablen des Moduls, basierend auf der Adresse 0x0

Symbol Tabelle – Alle Symbole mit den Attributen global / lokal, Referenz, etc.

Relocation Tabelle – Welche Bytes der CODE und DATA Sektion, müssten nach dem Zusammenführen angepasst werden.

ELF = Executable and Linkable Format

ELF enthält die oben genannten Abschnitte sowie weitere Abschnitte (z. B. Zeichenfolgentabellen, Debugging-Informationen usw.)

Beispiel square.o ELF Object File

- File section #1: code section, at base address 0x00000000
- File section #5: symbol table: `square` = `global` `code symbol`
- No data section (has no global variables)
- No relocation section (no referenced symbols in code/data)

```
File Type: ET_REL (Relocatable object) (1)
...
** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]
   Address: 0x00000000
 square
        0x00000000:   4601        .F      MOV     r1,r0
        0x00000002:   4608        .F      MOV     r0,r1
        0x00000004:   4348        HC      MULS    r0,r1,r0
        0x00000006:   4770        pG      BX      lr
...
** Section #5 '.symtab' (SHT_SYMTAB)
   #  Symbol Name                  Value     Bind  Sec  Type  Vis  Size
=============================================================================
   6  square                       0x00000001  Gb    1  Code  Hi   0x8
```

square.o
0000 CODE ...

Beipsiel main.o ELF Object-File

- **main.o (part I)**
  - File section #1: code section, at base address 0x00000000
    - 0x00000002: LDR r0, =a (address a stored at 0x14)
    - 0x0000000a: LDR r1, =b (address b stored at 0x18)
    - BL square calls a dummy address until linked
  - File section #4: data section, at base address 0x00000000

```
** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]
   Address: 0x00000000
main
        0x00000000:   b510        ..      PUSH    {r4,lr}
        0x00000002:   4804        .H      LDR     r0,[pc,#16] ; LDR r0, =a
        0x00000004:   6800        .h      LDR     r0,[r0,#0]  ; r0 = value at a
        0x00000006:   f7fffffe    ....    BL      square      ; BL needs adjustment
        0x0000000a:   4903        .I      LDR     r1,[pc,#12] ; LDR r1, =b
        0x0000000c:   6809        .h      LDR     r1,[r1,#0]  ; r1 = value at b
        0x0000000e:   1844        D.      ADDS    r4,r0,r1
        0x00000010:   2000        .       MOVS    r0,#0
        0x00000012:   bd10        ..      POP     {r4,pc}
        0x00000014:   00000000    ....    DCD     0 ; address a will be stored here
        0x00000018:   00000000    ....    DCD     0 ; address b will be stored here
** Section #4 '.data' (SHT_PROGBITS) [SHF_ALLOC + SHF_WRITE]
   Address: 0x00000000
        0x00000000:   00000005    ....    DCD     5 ; value at a = 5
        0x00000004:   00000007    ....    DCD     7 ; value at b = 7
```

square requires resolution

addresses of a and b require relocation

main.o
0000 CODE ...
0000 DATA ...

- **main.o (part II)**
  - File section #6: symbols:
    - a: local data section symbol, at offset 0x00000000
    - b: local data section symbol, at offset 0x00000004
    - main: global code section symbol, at offset 0x00000000 (LSB set: Thumb code)
    - square: global code section symbol, referenced (no definition in main.o)

```
...
** Section #6 '.symtab' (SHT_SYMTAB)
   #  Symbol Name                  Value     Bind  Sec  Type  Vis  Size
=============================================================================
   7  a                            0x00000000  Lc    4  Data  De   0x4
   8  b                            0x00000004  Lc    4  Data  De   0x4
  11  main                         0x00000001  Gb    1  Code  Hi   0x14
  12  square                       0x00000000  Gb  Ref  Code  Hi
...
```

- **main.o (part III)**
  - File section #7: relocation table:
    - Relocation at code address 0x00000006:
      - ► Modify the BL call to branch to the symbol square
    - Relocation at code address 0x00000014:
      - ► Set the absolute 32 bit value of the symbol a
    - Relocation at code address 0x00000018:
      - ► Set the absolute 32 bit value of the symbol b

Relocation table section

```
...
** Section #7 '.rel.text' (SHT_REL)
   #   Offset     Relocation Type                Wrt Symbol
=============================================================
   0  0x00000006   10 R_ARM_THM_CALL             12 square
   1  0x00000014    2 R_ARM_ABS32                 7 a
   2  0x00000018    2 R_ARM_ABS32                 8 b
...
```

Affected code section locations

```
** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]
...
0x00000006:   f7fffffe    ....    BL      square      ; BL needs adjustment
...
0x00000014:   00000000    ....    DCD     0 ; address a will be stored here
0x00000018:   00000000    ....    DCD     0 ; address b will be stored here
```

Relocate sections and usage of symbols

Relocation calculations

- new value = global base + merge offset + module relative offset
- E.g. symbol `b`:
  - ► global base = internal SRAM = 0x20000000
  - ► merge offset = 1st in merged data section = 0x00000000
  - ► module relative offset = b is the 2nd variable after a = 0x00000004
  - ► new value for symbol b = 0x20000004
- E.g. symbol `square` if user code (like main) starts at 0x08000254
  - ► 0x08000254 + 0x0000001C + 0x00000000 = 0x08000270

QUIZ_V02

a) Ordnen Sie die Namen `a, b, main, cube, res` und `i` im folgenden «C» Programm zu! (Antworten in menti.com/1k65ivcs1i)

```
// main.c

static uint32_t a = 5;
uint32_t b = 7;
int main(void) {
  uint32_t result;
  int i;
  result = cube(a) + b;
  for (i=0; i < result; i++) {
  ...
}
```

External linkage, export: main, b
External linkage, import: cube
Internal linkage: a
No linkage: result, i

b) Der obige Code verletzt eine Grundregel von "C". Welche ist dies? Was sollte geändert werden? (Antwort ins Plenum)

Verletzte Regel: **Declare before use**. Die extern definierte Funktion `uint32_t cube (uint32_t v)` muss erklärt werden. Idealerweise durch `#include "cube.h"`

c) In welchem physikalischen / logischen Adressbereich wird der Assembler Speicherplatz für die folgenden Symbole angelegt? (Antworten in menti.com/1k65ivcs1i)

main	a	cube
main:	CODE, READ-ONLY (z. B. internal Flash) (ebenda: Literals für die Adressen von a, b, result)	
cube:	NIRGENDS - wird vom Compiler nicht für dieses Modul angelegt (Externer, importierter Name)	
a, b, result:	DATA, READ-WRITE (z. B. internal SRAM)	
i:	STACK (oder lediglich in einem Register)	

Wie werden im ARM Assembler die verschiedenen Memory Regionen gekennzeichnet?

AREA – Anweisung:

AREA <name>, <type>, <access>

```
; ------------------------------------------
; -- Variables
; ------------------------------------------
        AREA MyAsmVar, DATA, READWRITE

; ------------------------------------------
; -- Constants
; ------------------------------------------
        AREA MyAsmConst, DATA, READONLY

; ------------------------------------------
; -- MyCode
; ------------------------------------------
        AREA MyCode, CODE, READONLY
```

Uebung01

Aufgabe 4

Beim Linken werden Object Files zu einem Executable File zusammengebunden. Damit dieses Binden erfolgreich ausgeführt werden kann, muss jedes Objekt File entsprechende Informationen enthalten.

Nennen Sie die Object File Sections welche die entsprechenden Informationen enthalten.

Object File Section Name	Enthaltene Information
Code Section	Instruktionen
Data Section	Globale Daten
Symbol Table	Liste der internen, importierten und exportierten Symbole
Relocation Table	Adressen der Instruktionen und Daten deren Zugriff auf Symbole im Laufe des Linkens angepasst werden müssen

Aufgabe 7

Was ist eine Native-Compiler Tool Chain, was eine Cross-Compiler Tool Chain?

1) Native-Compiler Tool Chain:

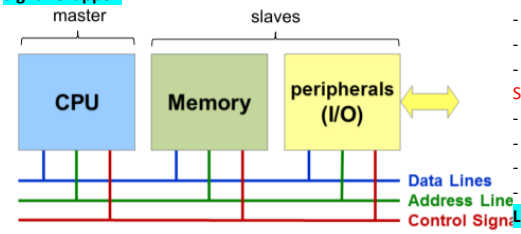
Erzeugt Programme für die gleiche Architektur/Umgebung in welcher die Tool-Chain ausgeführt wird.

2) Cross-Compiler Tool Chain:

Erzeugt Programme für eine andere Architektur/Umgebung als jene in welcher die Tool-Chain ausgeführt wird.

Kapitel 2 Microcontroller Basics

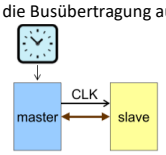
Signal Gruppen



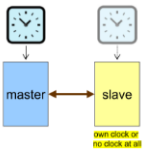
Adressen Leitung: **Unidirektional**, Vom Master zu Slaven
Daten Leitung: 8, 16, 32 oder 64 parallele Daten Leitungen, **bidirektional** (lesen, schreiben)
Control Signale: Kontrollierte Lese- und Schreibrichtung, enthält Zeitinformationen. **Meist Undirektional**.

Bus-Timing-Optionen

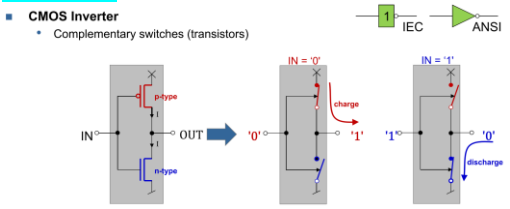
Synchron: Master und Slave haben denselben Takt. Takt steuert die Busübertragung auf beiden Seiten.



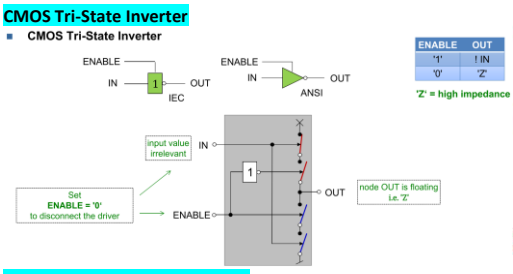
Asynchron: Slave haben keinen Zugriff auf den Takt vom Master. Steuersignale enthalten Zeitinformationen, um eine Synchronisation zu ermöglichen.



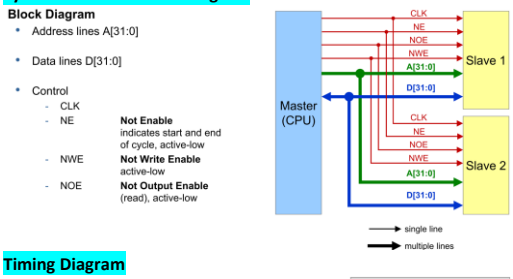
CMOS Inverter



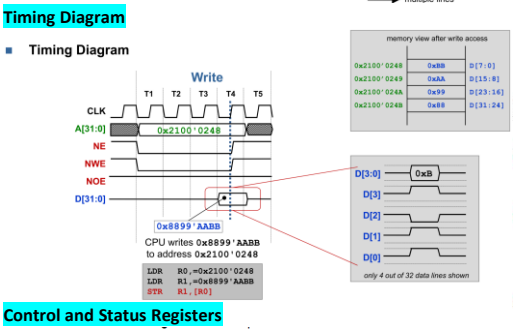
CMOS Tri-State Inverter



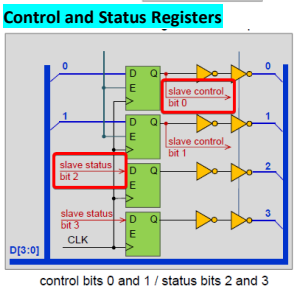
Synchroner Bus - Block Diagram



Timing Diagram



Control and Status Registers



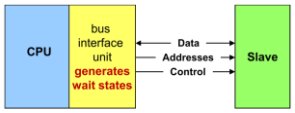
control bits 0 and 1 / status bits 2 and 3

Kontrollbits
- Lassen Sie die CPU den Slave konfigurieren
- CPU (Software) schreibt in das Registerbit
- Die Slave-Hardware verwendet die Ausgabe des Registerbits
- Normalerweise lesen / schreiben

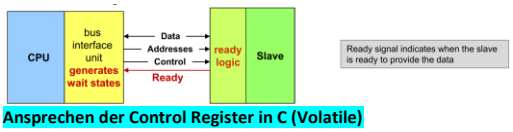
Statusbits
- Lassen Sie die CPU einen Slave überwachen
- Der Slave schreibt den Status in das Registerbit
- CPU (Software) liest Registerbit
- Normalerweise schreibgeschützt

Langsame Slaves – Zwei Möglichkeiten

1 Einzelne Wartezustände können an einer Busschnittstelleneinheit programmiert werden.



2 Der Slave teilt der Busschnittstelleneinheit mit, wann sie bereit ist.



Ansprechen der Control Register in C (Volatile)

Mit dem Schlüsselwort **Volatile** kann dem Compiler mitgeteilt werden, dass Statements nicht wegkompiliert werden sollen, da die Variablen durch die Hardware oder einen Interrput Handler geändert werden könnten. Der Compiler darf den schreib/lese Zugriff nicht weg-optimieren.

Quiz_V04

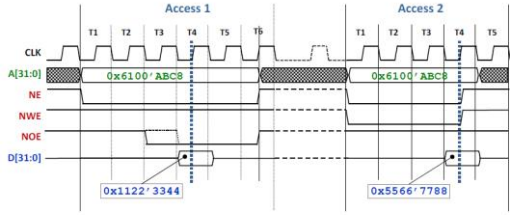
c) Wann und wo im Zeitverlauf werden allenfalls benötigte «Wait States» eingebaut? Wozu dienen diese?

Wait states werden benötigt, wenn ein Slave die Daten nicht schnell genug nach der fallenden Flanke des «NOE» Signals bereitstellen kann. Diese zusätzlichen Clock-Perioden werden zwischen dem fallenden NOE und dem Lesezeitpunkt (Steigende Flanke in T4) eingeschoben. Kriterium: Maximale Zeit des Slave zwischen fallender NOE Flanke und Gültigkeit der Datenleitungen. (→ Datenblatt).

Uebung02

Aufgabe 1

Gegeben sind die folgenden Buszugriffe



a) Geben Sie für beide Zugriffe jeweils die Richtung an (write oder read) sowie die Adresse des Zugriffs und den geschriebenen bzw. gelesenen Wert.

access 1: ***read von Adresse 0x6100'ABC8 mit Wert 0x1122'3344***
access 2: ***write nach Adresse 0x6100'ABC8 mit Wert 0x5566'7788***

b) Was enthält der Speicher **vor** dem Zugriff "Access 1"? Tragen Sie die Bytes, auf welche zugegriffen wird, mit ihren Adressen in der Memory Map ein. Der Prozessor ist little endian.

Adresse	Inhalt (Byte)
0x6100'ABC8	0x44
0x6100'ABC9	0x33
0x6100'ABCA	0x22
0x6100'ABCB	0x11

c) Was enthält der Speicher **nach** dem Zugriff "Access 2"? Tragen Sie die Bytes, auf welche zugegriffen wird, mit ihren Adressen in der Memory Map ein. Der Prozessor ist little endian.

Adresse	Inhalt (Byte)
0x6100'ABC8	0x88
0x6100'ABC9	0x77
0x6100'ABCA	0x66
0x6100'ABCB	0x55

Aufgabe 2

Gegeben ist ein System Bus mit den 6 Adresslinien `A[5:0]`.

a) Wie viele Bytes können damit adressiert werden?

2^6 = 64 → Es können 64 Bytes adressiert werden.

b) Unter wie vielen Adressen kann ein 8-bit Control Register angesprochen werden, wenn dafür 4 dieser Adressleitungen dekodiert werden?

$2^{(6-4)}$ = 4 → Es können 4 Bytes adressiert werden.

c) Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden, wenn nur die oberen 4 Adressleitungen wie folgt dekodiert werden:

```
select = A[5] & A[4] & !A[3] & !A[2]
```

1100XXb → 0x30, 0x31, 0x32, 0x33

d) Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden, wenn nur die unteren 4 Adressleitungen wie folgt dekodiert werden:

```
select = !A[3] & A[2] & A[1] & !A[0]
```

XX0110b → 0x06, 0x16, 0x26, 0x36

e) Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden, wenn nur die mittleren 4 Adressleitungen wie folgt dekodiert werden:

```
select = !A[4] & !A[3] & A[2] & !A[1]
```

X0010Xb → 0x04, 0x05, 0x24, 0x25

f) Wie müssen die Adressen dekodiert werden, wenn das Control Register genau unter der Adresse `0x28` angesprochen werden soll ?

```
select = A[5] & !A[4] & A[3] & !A[2] & !A[1] & !A[0]
```

Aufgabe 3

Schreiben Sie Codesequenzen in C für die folgenden Fälle. Verwenden Sie unsigned integer Typen aus `stdint.h`

a) Lesen Sie den Wert eines 8-bit Control Registers an der Adresse `0x6100'0007` in eine von Ihnen zu definierende Variable ein.

```
#define MY_BYTE_REG    (*((volatile uint8_t *)(0x61000007)))

uint8_t my_var;

my_var = MY_BYTE_REG;
```

b) Setzen Sie sämtliche Bits eines 16-Bit Control Registers an Adresse `0x6100'0008` auf ‚1'.

```
#define MY_HALFWORD_REG (*((volatile uint16_t *)(0x61000008)))

MY_HALFWORD_REG = 0xFFFF;
```

c) Warten Sie in einer Schleife, bis Bit 15 im 32-bit Control Register an der Adresse `0x6100'000C` auf ‚1' gesetzt ist.

```
#define MY_WORD_REG    (*((volatile uint32_t *)(0x6100000C)))

while (!(MY_WORD_REG & 0x00008000)){
}
```

d) Setzen Sie Bit 16 im Control Register an Adresse `0x6100'0010` auf ‚1' ohne die anderen Bits des Registers zu verändern.

```
#define MY_WORD_REG2    (*((volatile uint32_t *)(0x61000010)))

MY_WORD_REG2 |= 0x00010000;
```

Kapitel 3 General Purpose I/O (GPIO)

Push-pull vs. Open-drain

Push-pull vs. Open-drain

- Case output i.e. MODER[0] = '1'

OT = '0' → push-pull

Output stage can drive output 'high' or 'low'

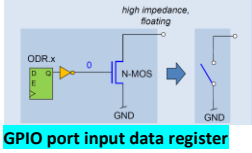
OT = '1' → open-drain

Output stage can only drive output 'low'

Open-drain

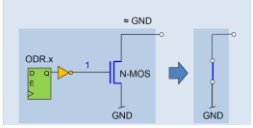
ODR.x = '1'

- R_{DS} is high
- Transistor is blocking → switch is open
- Output high impedance, floating



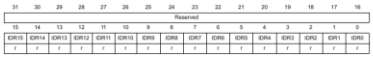
ODR.x = '0'

- R_{DS} is low
- Transistor is conducting → switch closed
- Output pulled to GND

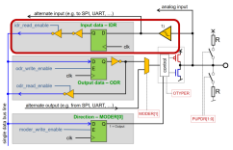


GPIO port input data register

GPIO port input data register (GPIOx_IDR)

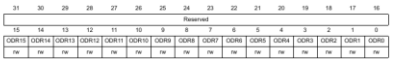


- IDR: Port input data
  - contain input value of corresponding I/O port
  - read-only

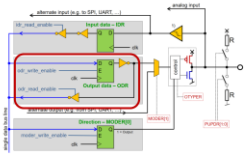


GPIO port output data register

GPIO port output data register (GPIOx_ODR)



- ODR: Port output data
  - read and write by software



GPIO port bit set/reset register

GPIO port bit set/reset register (GPIOx_BSRR)

- Clear port bit by writing a '1' to BSRR[bit+16]
- Set port bit by writing a '1' to BSRR[bit]
- Ensures atomic access in software (no interruption possible)
  - Setting a bit through ODR requires 'read ODR', 'OR operation with bit mask' and 'write ODR'

GPIO Configuration

Write the code to configure the bits from the last exercise

- GPIOA_MODER[7:6] → MODER3 = 01 → GP output
- GPIOA_OTYPER[3] → OT3 = 1 → open-drain
- GPIOA_OSPEEDR[7:6] → OSPEEDR3 = 00 → low speed
- GPIOA_PUPDR[7:6] → PUPDR3 = 01 → pull up

- Use the base addresses and structs in `reg_stm32f4xx.h`
- Do not change the other bits

```
#include "reg_stm32f4xx.h"

void config_gpioa_pin3(void)
{
    GPIOA->MODER &= 0xFFFFFF3F;
    GPIOA->MODER |= 0x00000040;

    GPIOA->OTYPER |= 0x00000008;

    GPIOA->OSPEEDR &= 0xFFFFFF3F;

    GPIOA->PUPDR &= 0xFFFFFF3F;
    GPIOA->PUPDR |= 0x00000040;
}
```

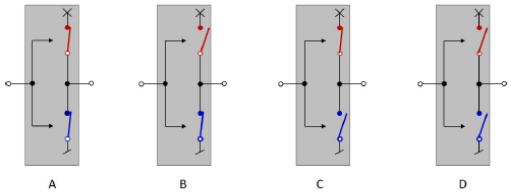
or

```
#include "reg_stm32f4xx.h"

void config_gpioa_pin3(void)
{
    GPIOA->MODER &= ~(0x03<<6);
    GPIOA->MODER |= (0x01<<6);

    GPIOA->OTYPER |=(0x01<<3);

    GPIOA->OSPEEDR &=~(0x03<<6);

    GPIOA->PUPDR &= ~(0x03<<6);
    GPIOA->PUPDR |= (0x01<<6);
}
```

Quiz_V05

Aufgabe 2: CMOS Basics

a) Welche der folgenden Schalterstellungen sind bei einem CMOS Inverter möglich? Antwort auf Mentimeter!



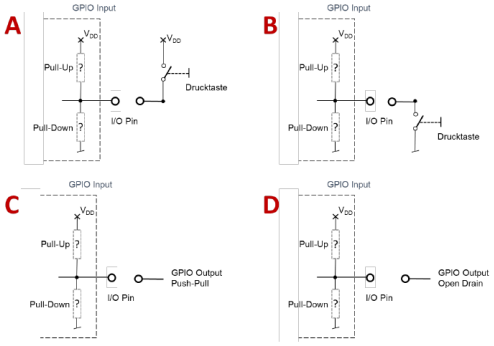
A B C D

Korrekte Antworten:
B (Input «1», output «0»)
C (Input «0», Output «1»)

Quiz_V06

Aufgabe 1: GPIO (Antworten auf Mentimeter)

a) Für welche der folgenden Input-output Kombinationen werden am Eingang Pull-Widerstände benötigt, und wenn ja welche?



	Pull-Up	No Pull	Pull-Down
A	-	-	X
B	X	-	-
C	-	X	-
D	X	-	-

Aufgabe 2: Registerzugriff

a) In einem Register `(uint32_t) (MYDEVICE->CONFREG)` sollen die Bits B29:B27 auf den binären Wert «011» gesetzt werden, ohne dass die anderen Bits des Registers geändert werden. Geben Sie den C-Code an.

B31	B30	B29	B28	B27	B26	B25	B24	B23	B22	B21	B20	B29	B18	B17	B16
		0	1	1											
B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

```
MYDEVICE->CONFREG &= ~(0x07 << 27)  //( oder &= 0xC7ff'ffff)
MYDEVICE->CONFREG |= (0x03 << 27)  //( oder |= 0x1800'0000)
```

Kapitel 4 Serial Data Transfer

Serial Communication

Communication Modes

- Simplex — Unidirectional, one way only
- Half-duplex — Bidirectional, only one direction at a time
- Full-duplex — Bidirectional, both directions simultaneously

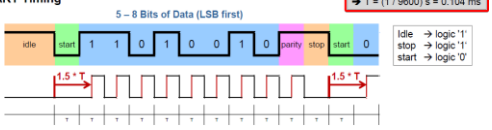
Timing

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

UART Timing

UART Timing

e.g. 9'600 Baud = symbols / s → T = (1 / 9600) s = 0.104 ms



- Transition stop ('1') → start ('0')
  - Receiver detects edge at the start of each data block (5 to 8 bits)
  - Allows receiver to sample data "in middle of bits" → red edges
  - Clocks have to be accurate enough to allow sampling up to parity bit, i.e. max. +/- 0.5 bit times aggregated deviation until last bit

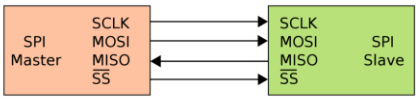
SPI Basics

SPI – Serial Peripheral Interface

- Serial bus for on-board connections
  - Used for short distance communication
- Connects microcontroller and external devices
  - Sensors, A/D converters, displays, flash memories, codecs
  - IOs, Real Time Clocks (RTC), wireless transceivers...
- Synchronous
  - Master distributes the clock to slaves
- Compared to a parallel bus
  - Saves board area
  - Lowers pin count on both chips (TX and RX) → smaller, low-cost
  - Simplifies EMC (Electromagnetic compatibility)

Synchronous Serial Data Connection

- Full duplex

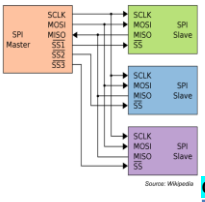


- Master (single master)
  - Generates clock (SCLK)
  - Initiates data transfer by setting $\overline{SS}$ = 0 (Slave Select)
- MOSI – Master Out Slave In
  - Data from master to slave
- MISO – Master In Slave Out
  - Data from slave to master

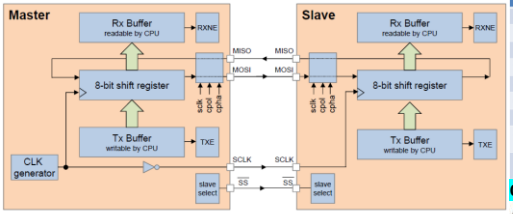
Single Master – Multiple Slaves

Single Master – Multiple Slaves

- Master generates a common clock signal for all slaves
- MOSI
  - From master output to all slave inputs
- MISO
  - All slave outputs connected to single master input
- Slaves
  - Individual select $\overline{SS1}$, $\overline{SS2}$, $\overline{SS3}$
  - $\overline{SSx}$ = '1' → Slave output MISOx is tri-state



Implementation Using Shift Registers



'LSB first' vs. 'MSB first' is configurable in most microcontrollers. Slaves are often hard-wired.

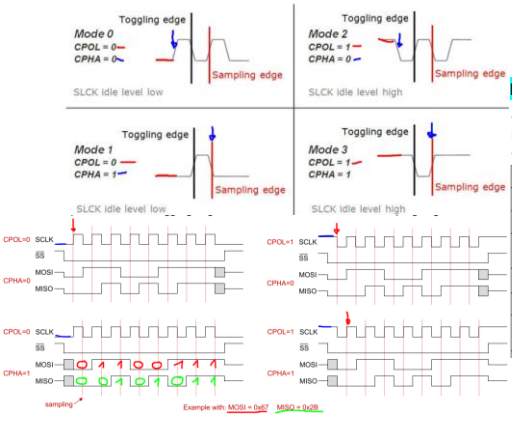
Status bits with interrupt
TXE Tx Buffer Empty
RXNE Rx Buffer Not Empty

Clock Polarity and Clock Phase

Clock Polarity and Clock Phase

CPOL ist Ruhezustand des CLOCK-Signals
CPHA ist die CLOCK-PHASE -->
CPHA = 0 erste Flanke ist Sampling Edge
CPHA = 1 zweite Flanke ist Sampling Edge

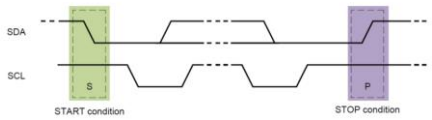
- TX provides data on 'Toggling Edge'
- RX takes over data with 'Sampling Edge'



I2C Bus

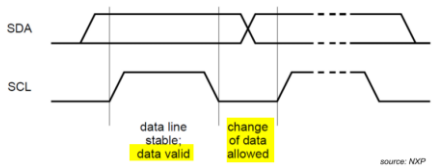
Operation

- Master drives clock line (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



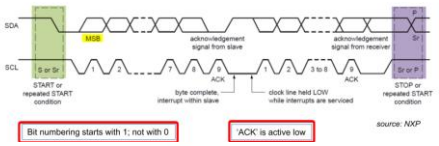
Driving Data on SDA

- Data driven onto SDA by master or by addressed slave
  - Depending on transaction (read/write) and point in time
  - Change of data only allowed when SCL is low
  - Allows detection of START and STOP condition



Data Transfer on I2C

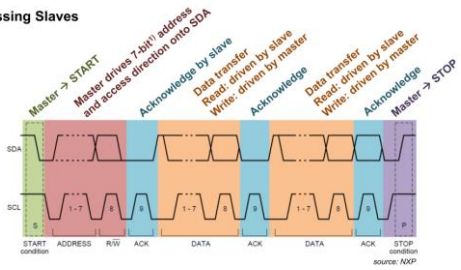
- 8-bit oriented transfers
- Bit 9: Receiver acknowledges by driving SDA low
- Master defines number of 8-bit transfers (STOP)



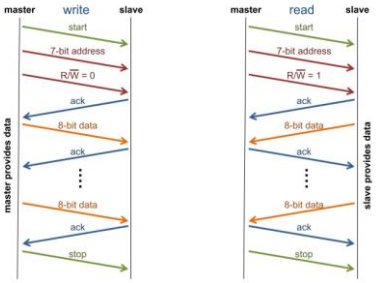
Bit numbering starts with 1; not with 0

'ACK' is active low

Addressing Slaves



Accesses



Comparison UART, SPI & 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.		

Quiz_V07

Aufgabe 1: SPI

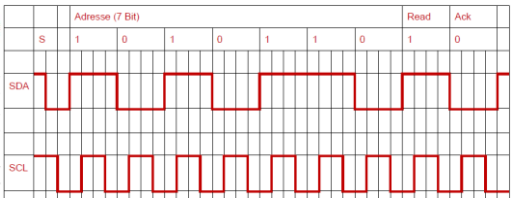
a) SPI ist ein De-Facto Standard. Geben Sie mindestens 4 Parameter an, die Sie für eine SPI Verbindung konfigurieren bzw. überprüfen müssen.

CPOL Clock Polarity (Ruhezustand der SCLK Leitung)
CPHA Clock Phase (sampling edge: Erste oder zweite SCLK Flanke)
Bit Order (MSB / LSB)
Wortgrösse (8 / 16 Bit)
Baudrate (heruntergeteilt von der Peripheriebus Clockrate)

PP

Aufgabe 6 — 6 Punkte

Mittels I²C soll ein Baustein zum Lesen adressiert werden. Die 7-Bit Adresse lautet 0x56. Zeichnen Sie die übertragenen Signale vom Start bis zum Acknowledge auf.



Uebung04

Aufgabe 1

Auf einer seriellen asynchronen Übertragungsleitung (UART) mit 19'200 Bit/s, 7 Daten-Bits, und einem Stop-Bit (ohne Parity Bit) soll die Zeichenfolge "AC" übertragen werden.

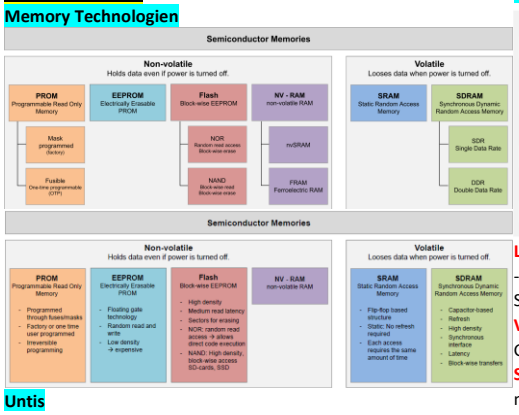
ASCII('A') = 0x41 = 100 0001b
ASCII('C') = 0x43 = 100 0011b

a) Wie lange dauert die Übertragung eines Bits (Periode T)?

1/19'200 s = 52.1 us

Kapitel 5 Memory

Memory Technologien



Untis

Unit Symbols

- b = bit B = Byte

Memory Chips

- Binary prefixes according to JEDEC[1] and IEC[2]
  - Kilo K = 1024
  - Mega M = 1024 x 1024 = 1'048'510
  - Giga G = 1024 x 1024 x 1024 = 1'073'741'824

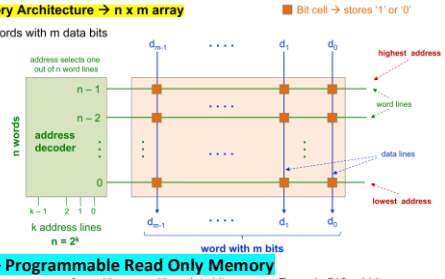
Hard Disks

- Often use SI (or metric) prefixes
  - Kilo k = 1000
  - Mega M = 1000 x 1000
  - Giga G = 1000 x 1000 x 1000

Memory Architektur

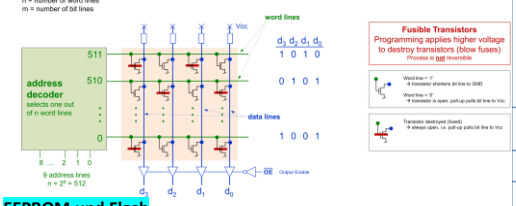
Memory Architecture → n x m array

- n words with m data bits



PROM – Programmable Read Only Memory

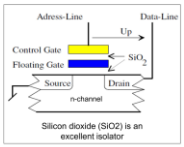
n x m array → n addresses with m data bits Example 512 x 4 bit



EEPROM und Flash

Making PROMs Reprogrammable

- "Floating Gate" transistor
  - Replace fusing by reprogrammable "Floating Gate"
- Write cell to '0' → ON
  - High voltage Up deposits charge on floating gate (isolated by SiO2)
  - Transistor ON (conducting) if control gate equal '1'
- Erase cell to '1' → OFF
  - Discharge floating gate with negative Up
  - Transistor is OFF, i.e. blocking independent of value on control gate
- EEPROM
  - High cell area → low density, high cost per bit
- Flash
  - Erasing can only be done for whole sectors → small cell area, high density, low cost per bit



Flash

Write Operations (Programming)

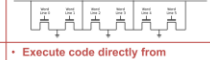
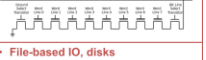
- Can only change bits from '1' to '0'
  - Otherwise an erase operation is required
- Word, half-word or byte access possible
- Writing a double word ~16 us
  - I.e. around 1000 times slower than SRAM

Erase Operations

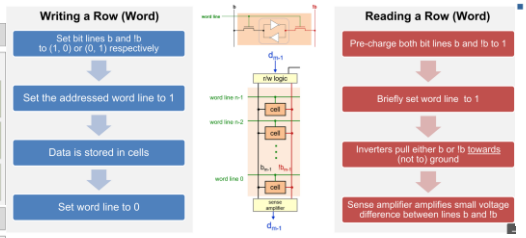
- Change all bits from '0' to '1'
  - Only possible by sector or by bank, not on a word
  - Typical sector sizes of 16
- Erase of a 128 Kbytes sector takes between 1 and 2 seconds [1]
- Endurance: 10'000 erase cycles [2]
- Sector may not be accessed (write or read) during erase
  - I.e. execute program from another sector or from SRAM during erase

1) Depending on supply voltage and configuration parameters
2) Value from STM32F42923 datasheet

Flash – NOR und NAND Topologie

	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
Interface	• Read same as asynchronous SRAM • Types with serial interface 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

SRAM – Static Random Access Memory



Lesen und schreiben: Alle Zugriffe dauern ungefähr gleich lange; -Zugriffszeit unabhängig vom Speicherort des Datenelements im Speicher

Volatile: Der Speicherinhalt bleibt nur so lange erhalten, wie das Gerät mit Strom versorgt wird

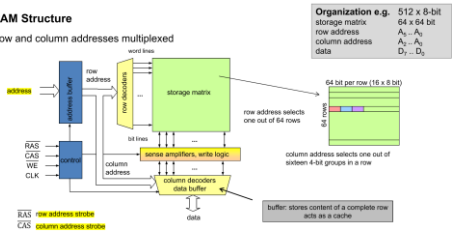
Statisch: -Keine Aktualisierung erforderlich; -Aktualisieren: Regelmässiges Lesen und Umschreiben der Speicherzelle, um den Inhalt zu erhalten

SDRAM – Synchronous Dynamic RAM

-Informationen, die als **Ladung im Kondensator** gespeichert sind
-Der Kondensator hält die Ladung nur einige Millisekunden lang
-Die Ladung muss regelmäßig aktualisiert werden
-Aktualisierungslogik befindet sich normalerweise auf dem SDRAM-Gerät

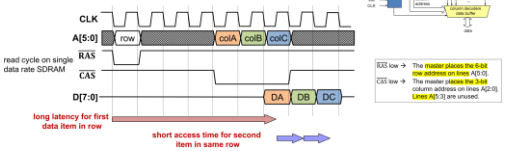
SDRAM Structure

- Row and column addresses multiplexed

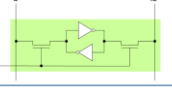
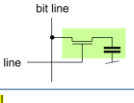


Synchronous Interface

- Multiplexed row and column addresses
- Clocks up to 1200 MHz



Static RAM (SRAM) vs Synchronous Dynamic RAM (SDRAM)

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

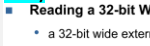
Externe Memory – Schreibe 32-Bit Word (Flexible Memory Controller FMC)

Writing a 32-bit Word from System Bus (32 Data Lines)

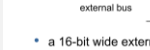
- Word stored in FMC-FIFO
- System bus is released for other accesses
- FMC-FIFO content is transferred to external memory using 1 to 4 bus cycles

- to a 32-bit wide external memory (32 data lines)

- to a 16-bit wide external memory (16 data lines)



- to an 8-bit wide external memory (8 data lines)

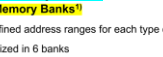


Externe Memory – Lese 32-Bit Word (Flexible Memory Controller FMC)

Reading a 32-bit Word from

- a 32-bit wide external memory (32 data lines)

- a 16-bit wide external memory (16 data lines)

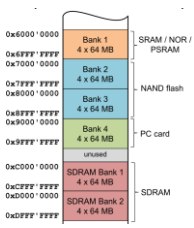


- an 8-bit wide external memory (8 data lines)

FMC – Memory Banks

FMC – Memory Banks[1]

- 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

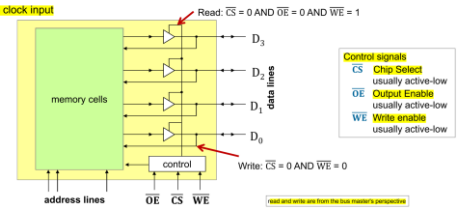


1) An organizational unit of memory. Bank size is architecture dependent

Asynchronous SRAM Device

Asynchronous SRAM Device

- No clock input



Alternatively the control logic can be represented with a truth table

CS	OE	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

FMC – Signale für SRAM

FMC Signals for SRAMs

- 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
NBL[3:0]	OUT	Byte enable

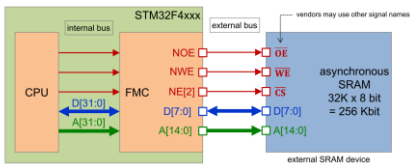
see "Synchronous Bus" in slide set "Microcontroller Basics"

- Write accesses: NBL[3:0] indicate which bytes shall be updated (see lab)
  - Example 32-bit data bus D[31:0]
    - Word access → all four bytes NBL[3:0] = 0000b
    - Half-word access → two out of four bytes e.g. NBL[3:0] = 0011b
    - Byte access → one out of four bytes e.g. NBL[3:0] = 1011b

Beispiel 32K x 8Bit SRAM

Example 32K x 8 bit SRAM

- Connecting an external 8-bit asynchronous SRAM device

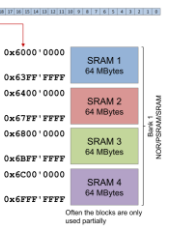


FMC – SRAM (Bank 1)

FMC – SRAM (Bank 1)

- Select one out of four SRAM devices
  - Address bits 27:26 → Encoded in signals NE[4:1]

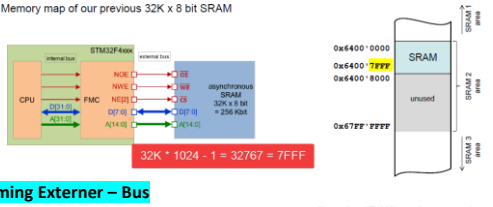
A[27:26]	Enable	Memory Device
00	NE[1]	SRAM 1
01	NE[2]	SRAM 2
10	NE[3]	SRAM 3
11	NE[4]	SRAM 4



- Data bus configured in control registers
  - Example
    - SRAM1 as 32-bit → D[31:0]
    - SRAM2 as 8-bit → D[7:0] only
    - SRAM3 as 16-bit → D[15:0] only
    - SRAM4 as 32-bit → D[31:0]

Example (revisited)

- Memory map of our previous 32K x 8 bit SRAM

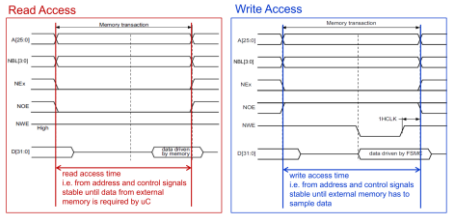


Timing Externer – Bus

Timing on External Bus

Figures from STM32F4xxx reference manual p. 1591, chapter 37, Flexible Memory Controller

- As seen from the microcontroller



Quiz_V07

Aufgabe 2: Memory

Im Datenblatt eines Speicherbausteines finden Sie folgende Angabe:
128k x 16 Bit SRAM
n • m

a) Wie viele Bytes kann der Baustein speichern?
16 Bit --> 2 Byte Speicherwortbreite
128k * 2 = 256 kByte

b) Wie viele Datenleitungen werden benötigt?
m = 16 also 16 Datenleitungen notwendig
D[15:0]

c) Wie viele Adressleitungen werden benötigt?
n = 2^x
128k = 2^x
128K * 1024 = 131072
log2(131072) = 17 --> Somit sind 17 Adressleitungen notwendig
A[16:0]

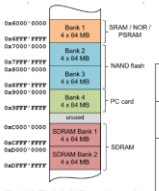
Aufgabe 2: Flexible Memory Controller

a) Welcher Speichertyp kann unter den folgenden Adressen angesprochen werden?

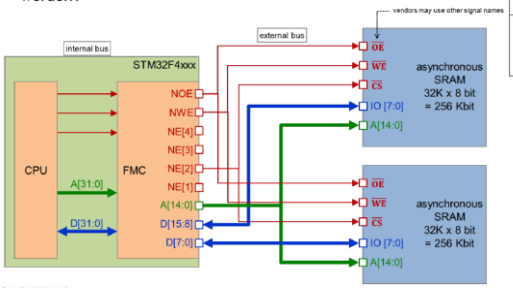
0xC864'2000 → SDRAM (Bank 1, Device 3)

0x7324'1478 → NAND – Flash (Bank 2, Device 1)

0x643A'89BC → SRAM (Device 2)



b) Zwei 8-Bit Bausteine werden zusammen als 16-Bit SRAM Device 2 an den FMC angeschlossen. Wie müssen diese angeschlossen werden?



Aufgabe 3: Speichertechnologien

Ordnen Sie die folgenden Speichertechnologien zu:

Prozessor-nahes Memory, geeignet als Cache	SRAM
Prozessor-nahes Memory, geeignet für Code	NOR-Flash
Prozessor-fernes Memory, geeignet für Code und Daten, flüchtig	SDRAM
Prozessor-fernes Memory, geeignet für Code und Daten, nichtflüchtig	NAND-Flash
Prozessor-fernes Memory, geeignet für grosse Mengen Code und Daten, nichtflüchtig	Hard Disk

Aufgabe 1

a) Nennen Sie drei Unterschiede zwischen einem 'asynchronous SRAM' (statisches RAM) und einem SDRAM (Synchronous Dynamic RAM).

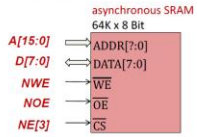
1. ***statisch (Inhalt bleibt bestehen solange Speisung anliegt) vs. dynamisch (permanenter refresh notwendig)***
2. ***Speicherelement: Flip-flop vs. Kondensator***
3. ***Schnittstelle: asynchron (NWE,NOE) vs. synchron (RAS,CAS)***
4. ***Zugriffszeit:***
   SRAM alle Zugriffzeiten gleich lang
   SDRAM Hohe Latenz für ersten Zugriff, kurze Zugriffszeit für Folgeadressen
5. ***SDRAM: kleinere Speicherzellen → höhere Anzahl von Speicherzellen pro Fläche***

b) Was ist die typische Funktion des Pins $\overline{OE}$ bei einem asynchronen SRAM?

Der Pin kontrolliert, ob das Memory Daten auf den Bus schreibt (treibt) oder ob sich die SRAM Ausgangstreiber der Datenleitungen im Floating Zustand befinden. Der Pin wird beim Auslesen des RAMs durch den Prozessor aktiviert (low gesetzt).

Aufgabe 2

Gegeben ist der folgende 'asynchronous SRAM' Baustein.



a) Wie viele Adresspins benötigt der Baustein?

64K = 2^{16} → 16 Adresspins → ADDR[15:0] 64 * 1024 = 65536 log2(65536) = 16

b) Der Baustein wird an den 'Flexible Memory Controller' (FMC) des STM32F4xxx angeschlossen. Adresse **0x6800'0000** soll die tiefste Adresse sein, unter welcher der Baustein angesprochen werden kann. Tragen Sie die anzuschliessenden FMC–Signale direkt links neben den Pfeilen ein.

c) Unter welcher Adresse greifen Sie aus der Software heraus auf das Byte an der höchsten Adresse des Bausteines zu? 0x6800'FFFF

d) Beim Entwickeln der Software stellen Sie fest, dass Sie unter der Adresse **0x6BFF'0000** auf das identische Byte des Bausteins wie unter **0x6800'0000** zugreifen. Was ist die Erklärung?

Partial Address Decoding: Die Bits A[25:16] sind nicht angeschlossen und werden deshalb nicht decodiert.

e) Unter wie vielen 64KByte Adressblöcken kann auf den Baustein zugegriffen werden?

A[25:16] → 10 Adresslinien → 2^{10} = 1024 Adressblöcke

0x68XX'0000
0x69XX'0000
0x6AXX'0000
0x6BXX'0000

Aufgabe 3

Welche der folgenden Aussagen treffen für ein DRAM (Dynamisches RAM) zu?
Bezeichnen Sie das entsprechende Feld mit einem ‚X'.

	trifft zu	*trifft nicht zu*
Die Daten werden in einer RS Flip-flop artigen Zelle gespeichert.		X
Zugriffe auf einzelne Speicherstellen haben eine hohe Latenz.	X	
Auf Grund der Leckströme ist ein periodischer Refresh notwendig.	X	
Der Preis pro Speicherzelle ist hoch.		X
Ist ein flüchtiger (volatiler) Speicher.	X	
Eignet sich gut für Blockzugriffe.	X	
Falls keine Zugriffe erfolgen, ist der Leistungsverbrauch sehr klein.		X

Quiz_V09

Aufgabe 1: Welche Zugriffsdiagramme sind hier gezeichnet?

Abbildung 1

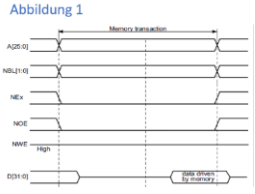
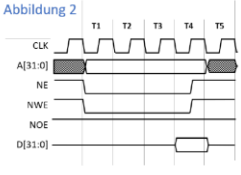


Abbildung 2



a) Asynchroner / synchron Zugriff
b) Speicherbaustein vermutlich on-chip / off-chip

Abbildung 1: Asynchrones SRAM/NOR-Flash, Off-Chip (26 Adress-Bits)

Abbildung 2: Synchrones SRAM/NOR-Flash, On-Chip (32 Adress-Bits)

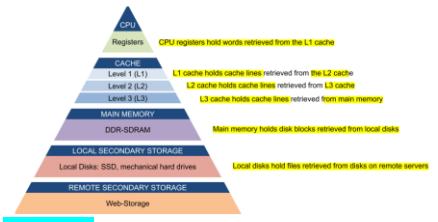
Kapitel 6 Cache

Prinzip der Loyalität

Räumliche Lokalität: Der aktuelle Datenstandort befindet sich wahrscheinlich in der Nähe des nächstgelegenen Speicherorts

Zeitliche Lokalität: Auf den aktuellen Datenstandort wird wahrscheinlich in naher Zukunft erneut zugegriffen

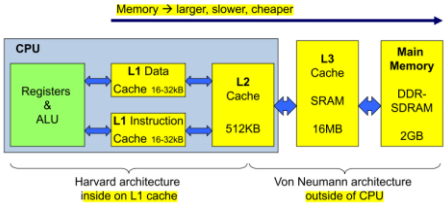
Speicher Hierarchie



Cache Level

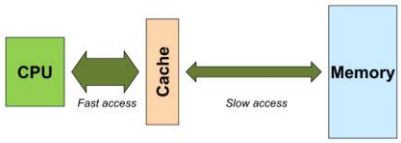
Cache Levels

- Typical Cache Architecture



Definition Cache

Computerspeicher mit kurzer Zugriffszeit. Speicherung häufig oder kürzlich verwendeter Anweisungen oder Daten.

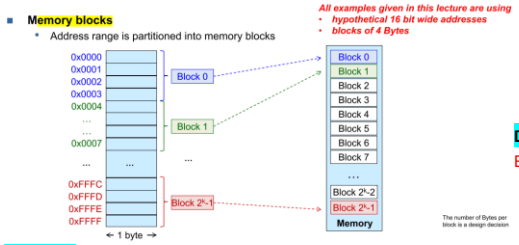


CPU
Very small
Registers (Word)

Cache
Small, fast, expensive
Caches a subset of memory blocks

Memory
Large, slow, cheap
Partitioned into memory blocks

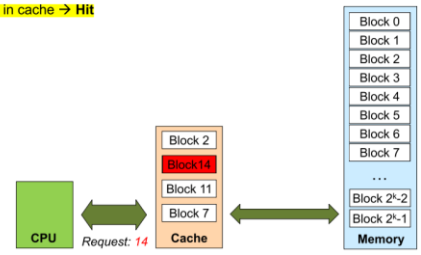
Memory blocks



Cache hit

Cache hit

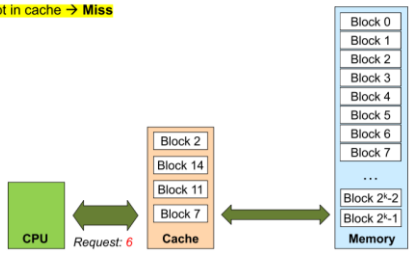
- Data in block 14 is needed
- Block 14 in cache → **Hit**



Cache miss

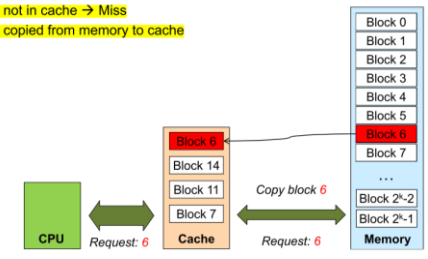
Cache miss (I)

- Data in block 6 is needed
- Block 6 not in cache → **Miss**



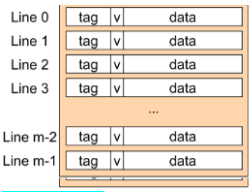
Cache miss (II)

- Data in block 6 is needed
- Block 6 not in cache → Miss
- Block 6 copied from memory to cache



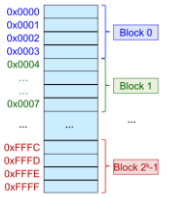
Cache Organisation

Valid bit v = zeigt an, dass die Zeile gültige Daten enthält
Tag = eindeutige Kennung für den Speicherort
Data = Daten von genau einem Speicherblock
m = Gesamtzahl der Cache-Zeilen



Adressierung

Addressing

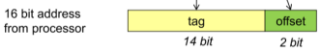


Block	Address	Block identification bits	offset
Block 0	0x0000	0000 0000 0000 00	00
	0x0001	0000 0000 0000 00	01
	0x0002	0000 0000 0000 00	10
	0x0003	0000 0000 0000 00	11
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11
...			
Block 2k -1	0xFFFC	1111 1111 1111 11	00
	0xFFFD	1111 1111 1111 11	01
	0xFFFE	1111 1111 1111 11	10
	0xFFFF	1111 1111 1111 11	11

Voll Assoziative (Fully Associative)

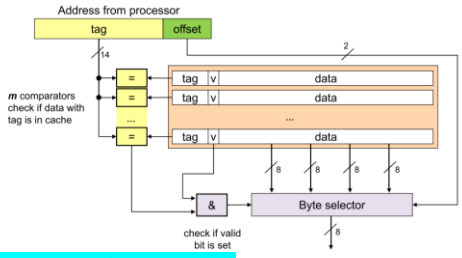
Tag enthält vollständige Blockidentifikation.

Block	Address	Block identification bits	offset
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11



Jede Cache-Zeile kann jeden Block laden

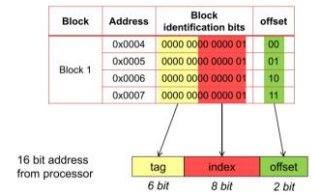
Architecture



Direkt Zugeordnet (Direct Mapped)

Blockidentifikation aufgeteilt in Tag und Index

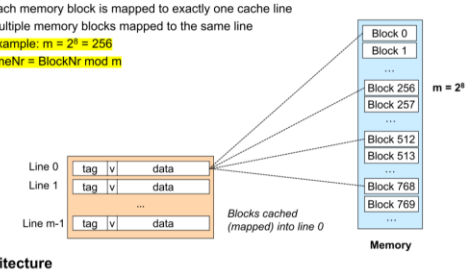
Block	Address	Block identification bits	offset
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11



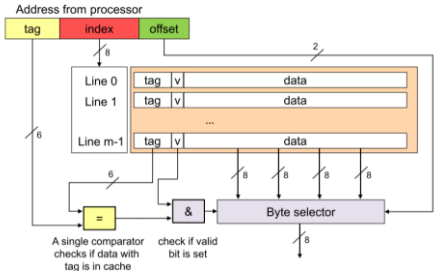
Jeder Speicherblock ist genau einer Cache-Zeile zugeordnet.
Mehrere Speicherblöcke sind derselben Zeile zugeordnet.

Organization

- Each memory block is mapped to exactly one cache line
- Multiple memory blocks mapped to the same line
- Example: m = 2^8 = 256
- LineNr = BlockNr mod m



Architecture

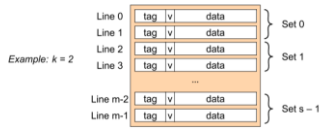


N-Wege Satz Assoziative (N-Way Set Assoicative)

N-Way Set Associative
Finding a compromise between simple logic (Direct Mapped) and high hit rates (Fully Associative)

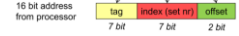
Organization

- Partition into sets
  - s = m/n number of sets
  - n lines per set („N-way")
  - b Bytes per line
- s x n x b data Bytes



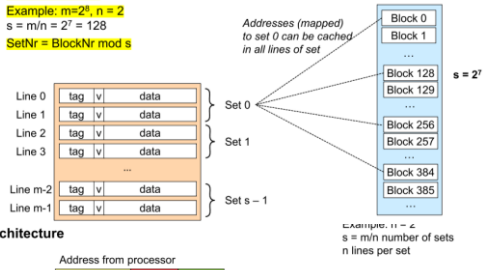
- Example: n = 2
  - Maximum index corresponds to number of sets (s = m/n)

Block	Address	Block identification bits	offset
Block 1	0x0007	0000 0000 0000 01	11
	0x0006	0000 0000 0000 01	10
	0x0005	0000 0000 0000 01	01
	0x0004	0000 0000 0000 01	00

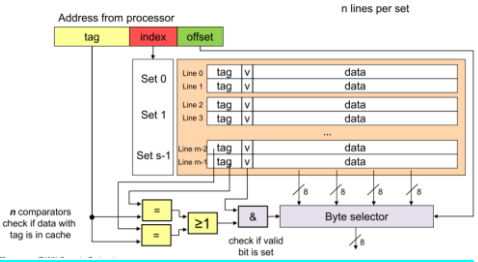


Organization

- Partition into sets
- Example: m=2^8, n = 2
  s = m/n = 2^7 = 128
- SetNr = BlockNr mod s



Architecture



Vergleich Voll assoziativ, direkt zugeordnet und N-Wege Satz assoziativ

Comparison

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

Performance

Cold miss: Erster Zugriff auf einen Block.
Capacity miss: Arbeitssatz größer als Cache.
Conflict miss: Mehrere Datenobjekte werden demselben Steckplatz zugeordnet.
Hit rate / miss rate: Bruchteil der im Cache gefundenen / nicht gefundenen Speicherreferenzen
hit rate = nr_of_hits / nr_of_accesses
miss rate = nr_of_misses / nr_of_accesses = 1 – hit rate
Hit time: Zeit, um einen Block im Cache an den Prozessor zu liefern
Miss penalty: Zusätzliche Zeit zum Abrufen von Daten aus dem Speicher aufgrund eines Cache-Fehlers

Ersatzstrategien

Auswahl der zu ersetzenden Cache-Zeile:
LRU: Least recenty used (Zuletzt benutzt)
LFU: Least frequently used (Am wenigsten verwendet)
FIFO: Fist In – First Out (ältetes)
Random Replace: zufällig gewählt

Schreibstrategien

Was tun bei einem Schreibhit?
Schreibe durch: Schreiben Sie sofort in den Speicher
Schreib zurück: Verzögerung des Schreibvorgangs in den Speicher bis zum Ersetzen der Zeile (benötigt ein gültiges Bit)
Was tun bei einem Schreibfehler?
Write-Allocate: Laden Sie die Zeile in den Cache (aus dem Speicher) und aktualisieren Sie die Zeile im Cache
No-Write-Allocate: Schreibt sofort in den Speicher

Aufgabe 2: Direct Mapped Cache

Bestimmen Sie aus den Angaben die Anzahl Adress-Bits sowie die Aufteilung auf Tag, Index und Offset

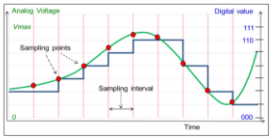
Grösse des Caches	32 kByte
Block-Grösse	64 Byte
Grösse des Arbeitsspeichers	2MByte

Anzahl Adress-Bits	21	(Schritt 1: 2 Mbyte = 2 * 2^{20} Byte) 2^1 * 2^20 = 2^21
Anzahl Bits für Tag	6	(Schritt 4: übrige Adressbits: 21 - 6 - 9 = 6)
Bits für Index	9	(Schritt 3: Grösse Cache (2^{15}) dividiert durch Blockgrösse) 0x2C 0x21 = 0x7D8 32768 / 64 = 512 512 = 2^x --> x = 9
Bits for Offset	6	(Schritt 2: 64 Byte = 2^6 Byte pro Block/Zeile)

Kapitel 7 ADC / DAC

Analog to Digital Converter

- **ADC – Analog to Digital Converter**
  - Converts input signal (voltage) to a digital value (N-bit)
  - Conversion results in one of 2^N possible numerical levels
  - Raw input signal can be dynamic[1] or static[2]
    - Dynamic signal (green) sampled at specific time intervals
    - Samples transformed into series of discrete values (blue)



Example

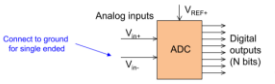
3-bit ADC
- 8 possible levels (000 – 111)
- Each conversion corresponds to one out of 8 levels

1) changing over time 2) time-invariant

- **Input signals**
  - Differential inputs
  - V_{in+} signal to convert (non-inverting input)
  - V_{in-} signal to convert (inverting input)

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

- **Single ended mode**
  - Only V_{in+} used
  - V_{in-} is grounded

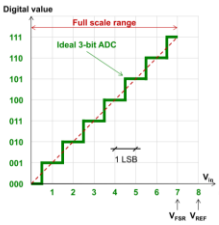


- **Reference voltage V_{REF+}**
  - Internal or external stable voltage
  - Needed to weight input voltage

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

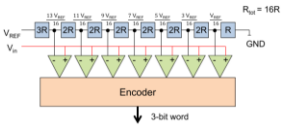
- **Resolution**
  - Number of bits N
  - Size of digital word
- **LSB[1]**
  - 1 LSB ≙ $V_{REF} / (2^N)$
- **Full Scale Range (FSR)**
  - Range between analog levels of minimum and maximum digital codes
  - V_{FSR} is one LSB less than V_{REF}

Example
V_{REF} = 8 V, N = 3 bits → 1 LSB = 8 V / 8 = 1 V
→ FSR from 0 V to 7 V



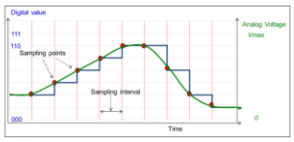
- **Example Flash ADC**
  - Network of 2^N resistors to divide V_{REF} into 2^N levels
  - $2^N - 1$ analog comparators
    - Compare input signal to divided reference voltages
  - Encoder transforms digital comparator results into N-bit word

3-bit ADC
Example: V_{REF} = 8V, V_{in} = 2.3V
$V_{REF} / 16$ = 0.5 V
$3\ V_{REF} / 16$ = 1.5 V
$5\ V_{REF} / 16$ = 2.5 V
....
Comparator stream = 0000011
3-bit output word = 010



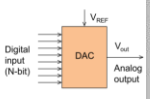
Digital to Analog Converter

- **DAC – Digital to Analog Converter**
  - Converts N-bit digital input to analog voltage level
  - E.g. music from your MP3 player is read and converted back to sound
    - A series of different values in the digital domain leads to a series of steps in the analog domain. The result is a dynamic output signal
    - "Play-back" time depends on time between conversions (sampling interval)

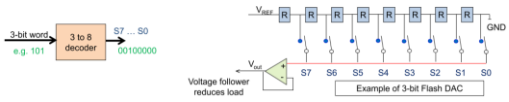


- **Reference voltage V_{REF}**
  - Accurate reference voltage (from internal or external source)
  - Needed to relate digital value to a voltage
- **Output signal V_{out}**
  - Analog output
    - Unipolar (only positive)
    - Bipolar (positive or negative)
  - Conversion yields approximation of digital signal

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



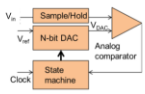
- **Example Flash DAC**
  - Network of resistors (of same value) creates 2^N voltage levels
  - N-bit digital input decoded into 2^N values (S0 … Sx)
    - Select single voltage level as DAC output



SAR ADC

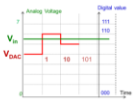
- **Successive Approximation Register (SAR) ADC**
  - Approach V_{in} with successive division by 2
    - Binary search
  - Start with half the digital value
    - MSB = 1, all other bits at 0
  - DAC generates analog value V_{DAC} that is compared to V_{in}
    - If $V_{DAC} < V_{in}$ → keep MSB at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

SAR ADC as an alternative to the previously introduced Flash ADC



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

3-bit result = 101



Flash ADC und SAR ADC

- **Flash ADC**
  - Fast conversion
  - Requires many elements
    - e.g. 255 comparators for 8-bit resolution
    - Power hungry
    - Consumes large chip area
- **SAR ADC**
  - Used on most microcontrollers
  - Good trade-off between speed, power and cost
    - Up to 5 Msps
    - Resolution from 8 to 16 bits

ADC Charakteristiken

Sample rate (Abtastrate)

-Eingangssignal, das zu diskreten Zeitpunkten abgetastet wird
-Sollte mindestens doppelt so hoch sein wie die höchste Frequenzkomponente des Eingangssignals sein - Nyquist-Shannon-Abtasttheorem

Conversion time (Umwandlungszeit)

-Zeit zwischen Beginn der Abtastung und digitaler Ausgabe verfügbar -Das Programmieren einer höheren Auflösung kann die Konvertierungszeit verlängern

Monotonicity (Monotonie)

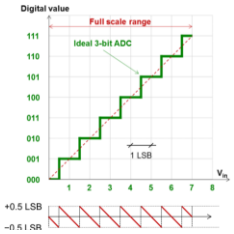
-Eine Erhöhung von Vin führt zu einer Erhöhung oder keiner Änderung der digitalen Ausgabe und umgekehrt

Quantization Fehler

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

Quantization error can be reduced by reducing LSB, e.g. either by increasing number of bits (resolution) or by reducing V_{REF}

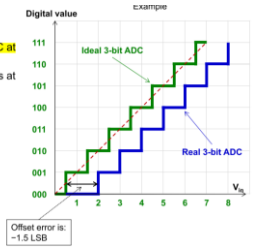
Reducing V_{REF} also reduces full scale range



Offset Fehler

Offset error
- Also called 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

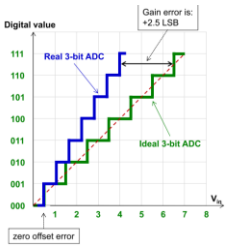
Measuring the offset error:
Zero-scale voltage is applied to analog input and is increased until first transition occurs



Gain Fehler

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



DAC output voltage

- **DAC output voltage**
  - Digital values are converted to output voltages on a linear conversion between 0 and V_{REF+}
  - Analog output voltages on each DAC channel pin are determined by the following equation:
    $DAC_{output} = V_{REF+} * DOR / 4095$
    - DOR is the digital value that should be converted
    - Use of an external V_{REF+} can help to improve results.
      - By choosing a smaller V_{REF+}, the minimal "analog step" is reduced
      - Using a dedicated pin allows the use of less noisy references

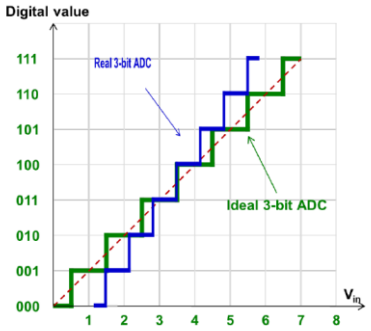
Programming the ADC

```
/* Setup GPIO and ADC3 */
hal_rcc_set_peripheral(PER_GPIOF, ENABLE);
hal_rcc_set_peripheral(PER_ADC3, ENABLE);

GPIOF->MODER |= (0x3 << 12);   // analog pin conf on PF.6
ADCCOM->CCR = (0x3 << 16);     // ADC prescaler 8

ADC3->CR1 = 0x0;               // 12 bit resolution, no scan
ADC3->CR2 = 0x1;               // single conv., enable ADC, right align

ADC3->SMPR1 = 0x0;
ADC3->SMPR2 = (0x6 << 12);     // ch4: 144 cycles sampling time

ADC3->SQR1 = 0x0;              // sequence length: 1
ADC3->SQR2 = 0x0;
ADC3->SQR3 = 0x4;              // ch4 is first in sequence

while (1) {
    ADC3->CR2 |= (0x1 << 30);      // start conversion
    while(!(ADC3->SR & 0x2)) {     // wait while conversion not finished
    }
    CT_SEG7->BIN.HWORD = ADC3->DR;  // show on 7-segment display
}
```

What is the max. achievable sampling rate with these settings and APB2 clock = 42 MHz?

Quiz_V10

Aufgabe 1: Bestimmen Sie den Offset- und Gain Error für die folgende Fehlerkurve eines 3-Bit AD Converters. Antwort auf Menti



Offset Error: -1.0

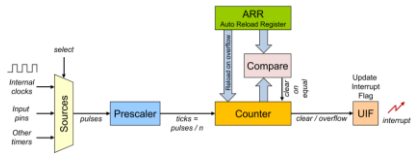
Gain Error: 2.0 (Blaue linie, muss auf den Startpunkt geschoben werden um besser z. erken.)

Full Scale Error: offset + Gain = -1.0 + 2.0 = 1.0

Kapitel 8 Timer / Counter

Counter overflow

- **Counter overflow**
  - Up Clear if counter reaches ARR
  - Down Reload with ARR if counter reaches zero
  - Set interrupt flag → trigger interrupt

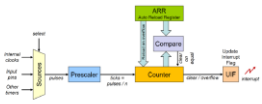


Down-counting example

Down-counting example
- Prescaler → divide by 4
- Count down to zero
  - Set interrupt request (UIF)
  - Restart from value in ARR

Up-counting example

Up-counting example
- Prescaler → divide by 4
- Count up to the value in reload register
  - Set interrupt request
  - Restart from 0



Timers TIM2-TIM5

Timers TIM2 - TIM5
- 16-bit → TIM3 and TIM4 32-bit → TIM2 and TIM5
  - Up, down, up/down
  - Auto-reload
- 16-bit programmable prescaler
  - Dividing counter clock frequency by factor between 1 and 65536
- Up to 4 independent channels for:
  - 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

Register address = Base address + Offset

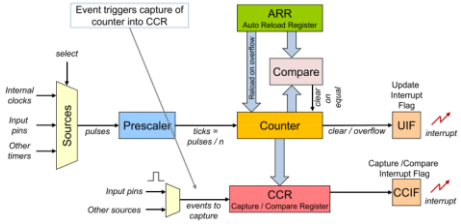
Register address = Base address + Offset
- Offset address is given for each register in Reference Manual
- Base address defined in memory map
  → Reference Manual

Boundary address	Peripheral
0x4000 0C00 - 0x4000 0FFF	TIM5
0x4000 0800 - 0x4000 0BFF	TIM4
0x4000 0400 - 0x4000 07FF	TIM3
0x4000 0000 - 0x4000 03FF	TIM2
0x4002 3800 - 0x4002 3BFF	RCC

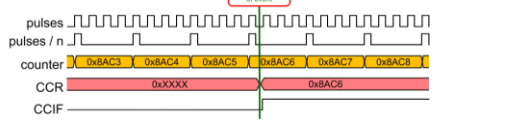
RCC: Reset and Clock Configuration

Input Capture

Measuring intervals → pulse lengths and periods
- Count ticks between timer start and an event

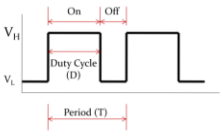


Capture example
- Stop watch
- At which moment in time does the user push the button?
  - Event = rising edge on input pin
  - Time of event is captured
  - Count continues



Duty Cycle – Definition

Duty Cycle – Definition



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

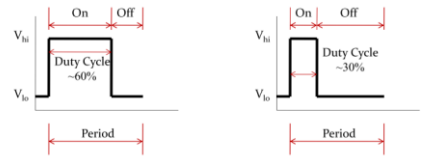
Average signal

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

Usually, V_L is taken as zero volts for simplicity.

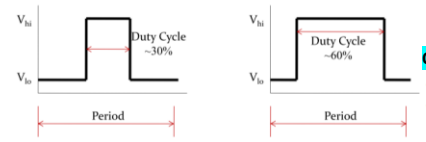
Types of PWM – Left Aligned

Types of PWM – Left Aligned
- Left edge is fixed, the trailing edge is modulated.

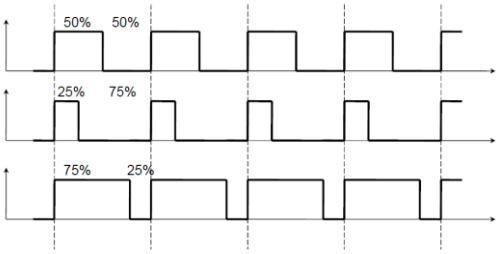


Types of PWM – Center Aligned

Types of PWM – Center Aligned
- Center of signal is fixed, both edges are modulated

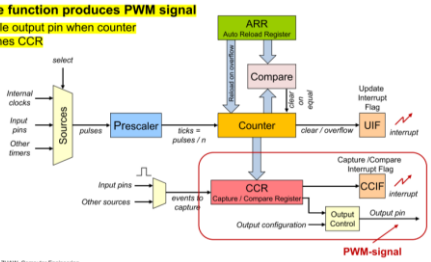


PWM-Signale sind digitale Signale (0/1) mit einer definierten Frequenz und variablen Impulsbreite



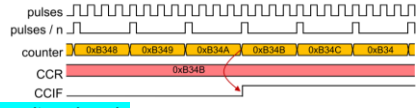
Output Compare – Generating PWM Signals

Compare function produces PWM signal
- Toggle output pin when counter reaches CCR



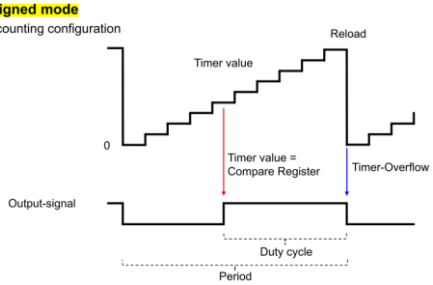
23.10.2020 ZHAW, Computer Engineering

Compare example
- Raise an alarm when specified count is reached or exceeded
- Continuously compare counter value to a reference value

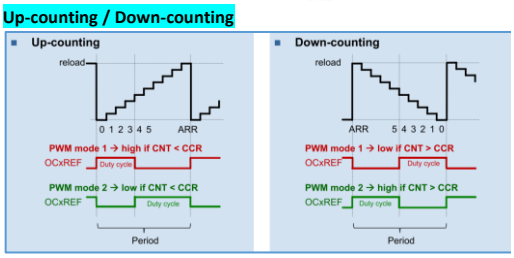


Edge-aligned mode

Edge-aligned mode
- Up-counting configuration



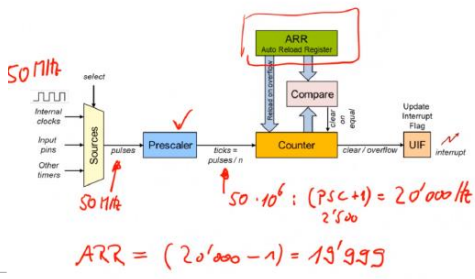
Up-counting / Down-counting



Quiz_V10a

Aufgabe 1: Timer

a) Welchen Wert müssen Sie in das ARR (Auto-Reload Register) des Zählers schreiben (up-counting), um aus einem Clock-Signal mit der Frequenz 50 MHz am Ausgang des Timers genau einen Interrupt pro Sekunde zu erzeugen?
Im Prescaler stehe der Wert 2499.



b) Was ändert sich, wenn Sie statt aufwärts abwärts zählen?

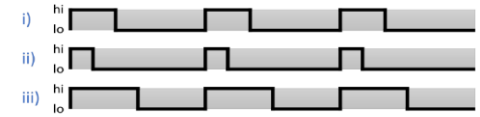
Up: CNT 0-1-2 ... - ARR-0 ...
Down: CNT ARR-(ARR-1)-...-1-0-ARR ...

von aussen kein Unterschied!

Quiz_V11

Aufgabe 1: PWM

Wie gross schätzen Sie den Duty Cycles in den Folgenden Darstellungen? (Antwort auf Menti)



i)	1/3	0.33	= 33%
ii)	1/6	0.166	= 16.7%
iii)	1/2	0.5	= 50%

Aufgabe 2: Impulsfolge generieren

Ein Timer mit INT_CLK = 100 kHz ist mit folgenden Werte konfiguriert: ARR = 39'999, PRE = 0. Welche Frequenz hat der erzeugte Interrupt? (Antwort auf menti.com)

Tick = INT_CLK / (PRE+1) = 100 kHz

$F_{UI} = F_{Tick} / (ARR + 1) = 2.5$ Hz

$T_{INT} = 400$ms

Uebung_kapitel_8

Aufgabe 1

a) Erklären Sie in Stichworten die einzelnen Funktionseinheiten eines Timers anhand der Tabelle.

Register	Inhalt	Funktion(en)
Prescaler	Divisor für Eingangssignal	Es wird nur jeder n-te Wert gezählt.
Counter	Aktueller Timerwert	Der Wert wird mit jedem n-ten Tick um eins erhöht oder erniedrigt
Reload	Wert für Timer-überlauf	Upcounter: Timer zählt bis zu diesem Wert, dann Überlauf Downcounter: Startwert für Timer
Capture/Compare	Vergleichswert	Capture: Bei einem Event wird der Wert des Counters hier gespeichert Compare: Sobald der Wert vom Counter erreicht wird, wird ein Event ausgelöst.

b) Erläutern Sie die Funktion Capture.

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

c) Erläutern Sie die Funktion Compare.

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

Aufgabe 2

Es soll Timer 3 des STM32F429 konfiguriert werden. Verwenden Sie das Reference Manual (zu finden in OLAT) zur Lösung der Aufgabe. Geben Sie die entsprechende Codezeilen in C an.

a) Als Source soll die interne Clock CK_INT mit 84 MHz verwendet werden. Setzen Sie die Bits im entsprechenden Register auf die notwendigen Werte.
Hinweis: andere Bits des Registers sollen nicht verändert werden.

```
TIM3_SMCR &= 0xFFF8;   // TIM3_SMCR [2:0]   0b000
```

b) Der Timer soll als Upcounter konfiguriert werden. Hinweis: andere Bits des Registers sollen nicht verändert werden.

```
TIM3_CR1 &= 0xFF8F
```

c) Die Zeit für den Timerüberlauf soll 200 ms betragen. Welche Werte müssen Sie in die Register PSC und ARR schreiben (Angabe hexadezimaler Werte)? Hinweis: Es sind verschiedene richtige Lösungen möglich.

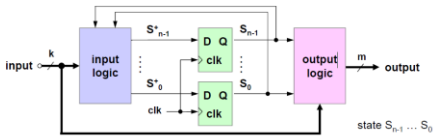
```
TIM3_PSC = 0x20CF   // (8400 - 1) entspricht 10 kHz
TIM3_ARR = 0x07CF   // (2000  - 1) entspricht 200 ms

TIM3_PSC = 0x0347   // (840 - 1) entspricht 100 kHz
TIM3_ARR = 0x4E1F   // (20000  - 1) entspricht 200 ms

TIM3_PSC = 0x01A3   // (420 - 1) entspricht 200 kHz
TIM3_ARR = 0x9C3F   // (40000  - 1) entspricht 200 ms
```

Kapitel 9 Software State Machine

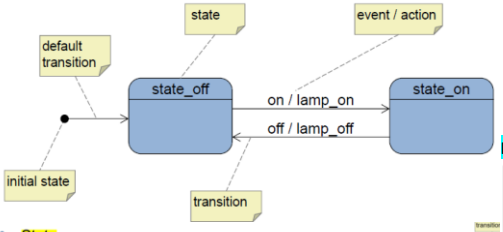
Finite State Machine (FSM) in hardware

Finite State Machine (FSM) in hardware

- Flip-flops store internal state
- Clock-driven
  - Inputs are evaluated[1] at each clock edge
  - State can only change on a clock edge



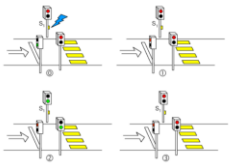
UML state diagram example

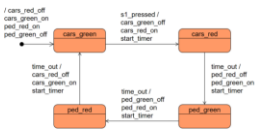


- State
  - Internal state of the system in which it is awaiting the next event
- Event
  - Asynchronous input that may cause a transition
- Transition
  - Reaction to an event: May change the state and/or trigger an action
- Action
  - Output associated with a transition
    - ► Either a directly carried out operation or a
    - ► Message to another FSM (seen as an event by the receiving FSM)
- Finite State Machine (FSM)
  - Machine with a finite number of states and transitions

Beispiel mit Ampel

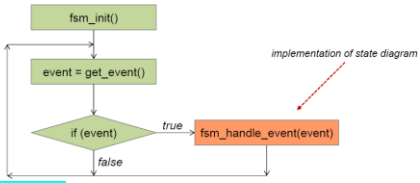
- **Example: Traffic light**





Simple FSM System

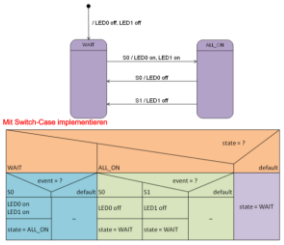
Simple FSM System



Beispiel LED Control

- **Example: LED control**

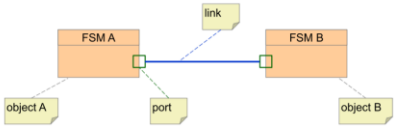
```
int main(void)
{
    event_t event;

    fsm_init();
    while (1) {
        event = get_event();
        if (event != NO_SWITCH){
            fsm_handle_event(event);
        }
    }
}
```



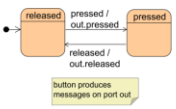
→ see code example

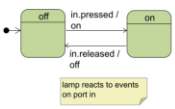
Interaktionen von FSM

- **Port**
  - Defines the messages that can be sent and received by an FSM
    - Output message → action of the FSM
    - Input message → event of the FSM
- **Link**
  - Defines a connection for sending messages



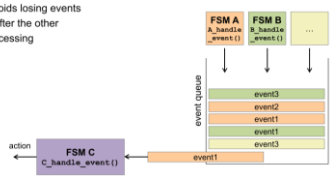
- **Example**
  - Reactive system partitioned into two FSMs
  - Interaction of FSMs happens through event-messages



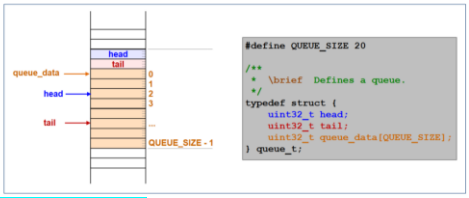


Event queue

- **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



- **Event queues**
  - Covered in Microcomputer Systems 1 (MC1)



Beispiel mit swtich Case

```
switch (state) {
    case STATE_A:
        switch (event) {
            case S0:
                action();
                state = NEW_STATE;
                break;
            default:
                state = WAIT;
        }
        break;

    case STATE_B:
        switch (event)
```

Kapitel 10 Interrupt Performance

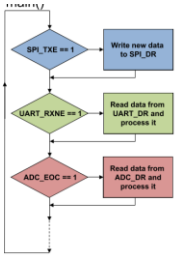
Polling

«

Periodic Query of Status Information

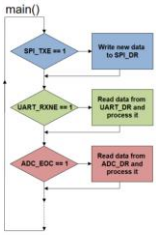
- Synchronous with main program
- Advantages
  - Simple and straightforward Einfach und unkompliziert
  - Implicit synchronization
  - Deterministic
  - No additional interrupt logic required Keine zusätzliche Interrupt-Logik erforderlich
- Disadvantages
  - Busy wait → wastes CPU time
  - Reduced throughput Reduzierter Durchsatz
  - Long reaction times
    in case of many I/O devices or if the CPU is working on other tasks

[1] to poll → abfragen

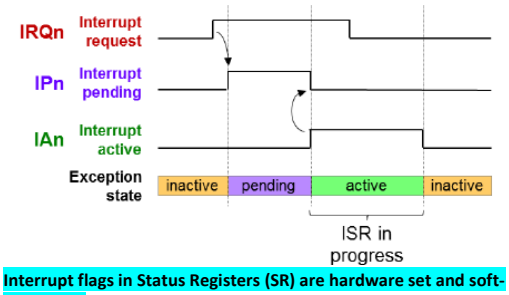


Implementation

```
while(1){
    if (spi_is_txe_set()) {
        ...
        spi_write_data(...);
    }

    if (uart_is_rxne_set()) {
        uart_data = uart_read_data();
        ...
    }

    if (adc_is_eoc()) {
        adc_data = adc_read_data();
        ...
    }

    ...
}
```

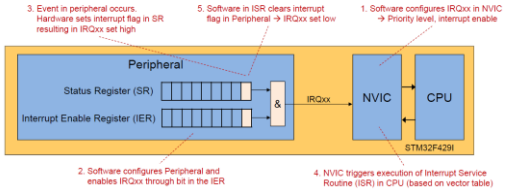


Interrupt Driven I/O



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

- Software has to enable IRQxx line in peripheral

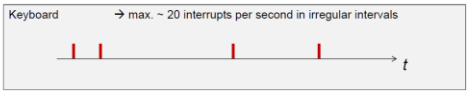


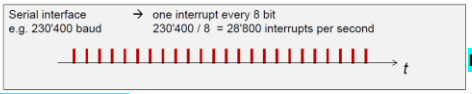
Interrupt frequency

Interrupt frequency

f_{INT}

- How often does an interrupt occur
- Varies from source to source, e.g.



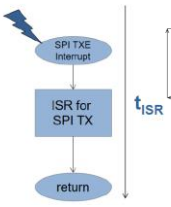


Interrupt service time

Interrupt service time

t_{ISR}

- Required time to process an interrupt
  - I.e. execution time of ISR
- Depends on
  - Number of instructions in ISR
  - Required number of clock cycles per instruction
    → depends on CPU architecture
  - CPU clock frequency
  - Time for switching to and returning from ISR



Impact on system performance

Impact on system performance

- Percentage of CPU time used to service interrupts

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

- Example keyboard

$f_{INT} = 20\ Hz = 20\ \frac{1}{s}$ $t_{ISR} = 6\ us$ [1]

Impact = 20 Hz * 6 us * 100 % = 0.012 %

- Example serial interface with 230'400 Baud

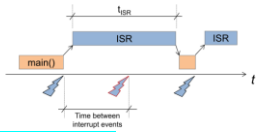
$f_{INT} = 230'400 / 8 = 28'800\ Hz$ $t_{ISR} = 6\ us$ [1]

Impact = 28'800 Hz * 6 us * 100 % = 17.3 %

Time between two interrupt events

t_{ISR} > "Time between two interrupt events"

- Some interrupt events will not be serviced (lost) Einige Interrupt-Ereignisse werden nicht bedient (verloren)
  - Data will be lost
- f_{INT} as well as t_{ISR} may vary over time
  - Average may be ok, but individual interrupt events may still be lost

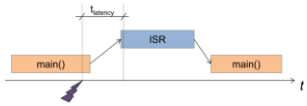


Caution in case of several interrupt sources
- Interrupts can occur simultaneously
- Computing power required for both ISRs

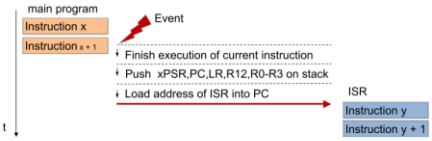
Interrupt latency

Interrupt latency definition

- Time between interrupt event and start of servicing by ISR Zeit zwischen Interrupt-Ereignis und Beginn der Wartung durch ISR
  - How long does it take until first 'useful' instruction in ISR is executed
- Range
  - From about 50 nanoseconds (ns) up to several milliseconds (ms)
- Relevant in cases where guaranteed service times are required
  - E.g. audio/video streaming



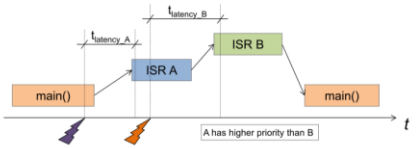
Latency influenced by hardware (CPU)



- Different instructions may have different execution times
- Multi-cycle instructions on Cortex-M3/M4
  - Some (e.g. SDIV/UDIV) are abandoned and restarted after ISR
  - Some (e.g. LDM/STM) are interrupted and resumed after ISR

Latency influenced by software (code)

- Saving additional registers on stack
- Process ongoing or higher prioritized ISRs
- Masked (disabled) interrupts → CPSID i / CPSIE i
- In case several sources are using the same interrupt line
  - SW has to use polling to know which peripheral requires servicing



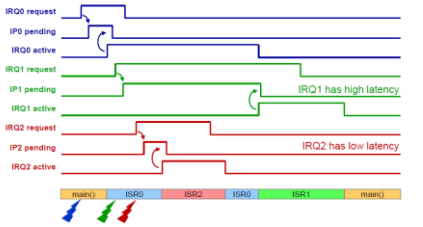
Beispiel Interrupt Prioritäten

Example interrupt priorities

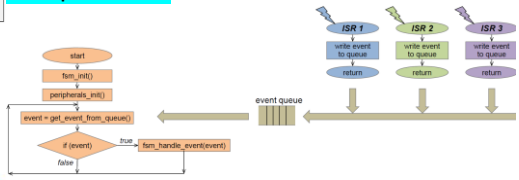
- ISR1 does not pre-empt ISR0
- ISR2 pre-empts ISR0

assuming

IRQ0	PL0 = 0x2	medium priority
IRQ1	PL1 = 0x3	lowest priority
IRQ2	PL2 = 0x1	highest priority



Interrupt Driven FSM



Kapitel 11 Remaining Data types

Binary representation

Binary representation

2^i	2^{i-1}	•••	4	2	1	1/2	1/4	1/8	•••	2^{-j}
b_i	b_{i-1}	•••	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	•••	b_{-j}

- **Value** $\sum_{k=-j}^{i} b_k \times 2^k$
- **Examples**
  - 5 3/4 101.11_2
  - 2 7/8 10.111_2

Real numbers: general notation

Value = Sign · Fraction · Base $^{\text{Exponent}}$

Normalized scientific notation

- Single non-zero digit to the left of the decimal (binary) point

Decimal vs. binary

- Decimal: 0.00002796 $= 2.796 \cdot 10^{-5}$
- Binary: 0.000010110111 $= 1.0110111 \cdot 2^{-5}$

Defines how floating point numbers are represented

- Easy exchange of data between machines
- Simplifies hardware algorithms
- Exists since 1980

Binary notation

Sign | Exponent | Fraction

S	E	F

Fraction

Fraction

S	E	F

- Binary notation
- Representing normalized numbers → number of form ***1***.xxxx..
- In IEEE 754 standard, the ***1*** is implicit

Fraction value = (1 + F)

- Example

1.265625 d = 1.010001 b → F = 010001

Remark: 1.010001 b = (1 + 0 x 2^{-1} + 1 x 2^{-2} + ... + 1 x 2^{-6}) d

Exponent

Exponent

S	E	F

- Binary excess notation

Exponent value = (E - Bias)

- Bias: 127 (float) / 1023 (double)
- Example (float):

0111'1010b = 122 → 122 – 127 = -5 → Value = 2^{-5}

Value = 2^{-5} → -5 + 127 = 122 = 0111'1010b

Exponent Single precision (float)

S	E	F

E		Meaning	
0000 0000		Reserved	→ Unnormalized → See later…
0000 0001		-126_{10}	
0000 0010		-125_{10}	
0111 1111		0_{10}	
1111 1110		127_{10}	
1111 1111		Reserved	→ Not a Number → See later…

Sign

Sign

S	E	F

- 1 bit
- Sign value = $(-1)^S$

Sign and Magnitude Representation

S	E	F

Value = $(-1)^S \cdot (1 + F) \cdot 2^{(E - Bias)}$

- More exponent bits → wider range of numbers
- More fraction bits → higher precision

Beispiel

Decimal 7.5 d = ?

7.5d = 111.1 b = 1.111b * 2^2

E = 2 + 127 = 129

01000000111100000000000000000000 = 0x40F00000

Decimal → binary	7.5d = 111.1b
Normalise (1.xx)	$111.1 = 1.111 \cdot 2^2$
Exponent	E = 2 + 127 = 129d = 1000'0001b
Sign	7.5 > 0 → V = 0
«Package»	01000000'11110000'00000000'00000000

Vorzeichen | Exponent | Fraction

Binary 1'0111111'0'1010000'00000000'00000000 = ?

S = 1 (Vorzeichen)
E = 128 -> E - Basis = 126 - 127 = -1
F = 101

-1.101 * 2^-1 --> -0.1101 = -(0.5+0.25+0+0.0625) = -0.8125

Special Values

Special Values

E	F	S	Result
255	≠ 0		Not a Number (NaN)
255	0	1	- ∞
255	0	0	∞
0 < E < 255			$(-1)^S \cdot (1 + F) \cdot 2^{(E-Bias)}$ Normalized
0	≠ 0		$(-1)^S \cdot (0 + F) \cdot 2^{(1-Bias)}$ Unnormalized
0	0	1	- 0
0	0	0	+ 0

Ranges (Normalized / Unnormalized)

Normalized (positive): $(1 + F) \cdot 2^{(E-Bias)}$

- Smallest `0 00000001 00000000000000000000000` $+2^{-126}(1+0) = 2^{-126}$
- Largest `0 11111110 11111111111111111111111` $+2^{127}(1+(1-2^{-23}))$

Unnormalized (positive): $(0 + F) \cdot 2^{(1-Bias)}$

- Smallest `0 00000000 00000000000000000000001` $+2^{-126}(2^{-23}) = 2^{-149}$
- Largest `0 00000000 11111111111111111111111` $+2^{-126}(1-2^{-23})$

Single precision and double precision

Single precision (float)

1 bit | 8 bits | 23 bits

S	E	F

- 32 bits
- Exponent bias: 127
- Dynamics: $-3.4 \cdot 10^{38}$ ... $-1.17 \cdot 10^{-38}$, 0, $1.17 \cdot 10^{-38}$ ... $3.4 \cdot 10^{38}$
- Resolution: $2^{-24} = 0.6 \cdot 10^{-7}$ → 7 digits

Double precision (double)

1 bit | 11 bits | 52 bits

S	E	F

- 64 bits
- Exponent bias: 1023
- Dynamics: $-1.8 \cdot 10^{308}$ ... $-2.2 \cdot 10^{-308}$, 0, $2.2 \cdot 10^{-308}$ ... $1.8 \cdot 10^{308}$
- Resolution: $2^{-53} = 0.11 \cdot 10^{-15}$ → 15 digits

Percision errors

Precision:

```
int main(void) {
        float a = 1.8;
        float b = 18;
        if(a*10 == b) printf("\nNumbers are the same\n\n");
        else          printf("\nNumbers differ!!\n\n");
}
```

- Output:

```
Numbers are the same
```

```
int main(void) {
        float a = 1.81;
        float b = 18.1;
        if(a*10 == b) printf("\nNumbers are the same\n\n");
        else          printf("\nNumbers differ!!\n\n");
}
```

- Output

```
Numbers differ!!
```

Precision:

```
int main(void) {
        int x   = 33554431;
        float y = 33554431;
        printf("int: %d\n", x );
        printf("float %f\n", y );
}
```

- Output:

```
int: 33554431
float: 33554432.000000
```

- Single Precision float of `33554431`:

Binary32: 4C000000			
Status	**Sign [1]**	**Exponent [8]**	**Significand [23]**
Normal	0 (+)	10011000 (+25)	1.00000000000000000000000 (1.0)

2^{25} = 33554432

Arrays

Pointer to first element of array

```
int main(void) {
    int j, sum = 0;
    int array[5];
    int *ptr;                   // pointer to integer
    for (j = 0; j < 5; j++) {
       array[j] = j;
    }
    ptr = array;                // copy pointer
    for (j = 0; j < 5; j++) {
       sum = sum + *ptr;        // add value ptr points
       ptr++;                   // point to next element
    }
}
```

```
for (j = 0; j < 5; j++) {
    sum = sum + array[j];
}
```

Multidimensional arrays are "Arrays of Arrays"

```
void main(void) {
   int array [3][3];
   int *ptr;
   int i, j, k;
   j = 0;                       // fill array with ascending
   for (i = 0; i < 3; i++) {    // numbers
      for (k = 0; k < 3; k++) {
         array [i][k] = j++;
   } }
   ptr = array [0];             // get pointer to first element
   for (i = 0; i < 9; i++) {
      printf("index: %d, value: %d\n", i, *ptr);
      ptr++;
   } }
```

- Array a[3][3]: [a00 a01 a02 / a10 a11 a12 / a20 a21 a22]
- Is stored as: [a00 a01 a02 a10 a11 a12 a20 a21 a22]

Strings

Strings in C are not objects

- Declaration with char arrays
- Example

```
char Name[16] = "Meier";
```

- Question: How is the end of a string recognised?
- All Strings in C are 0 – terminated
  → after the last character follows a '0'

'M'	'e'	'i'	'e'	'r'	'\0'	'i'	'e'	'a'	'Q'	'u'	'a'	'r'	'\0'	'l'	'l'

valid String | Terminator | undefined content

- Length of strings not stored → count characters to '\0'

Structs

Structs contain different data that belong together

- Declaration

```
struct person {
  char       name[8];
  char       prename[8];
  int        pers_nr;
  unsigned   char day_of_birth;
  unsigned   char month_of_birth;
};
```

- Definition

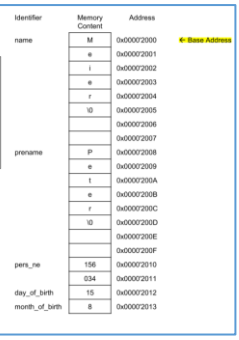
```
struct person aperson;
```

- Usage

```
strcpy(aperson.Name, "Meier");
strcpy(aperson.prename, "Peter");
```

Access to struct

```
struct person {
  char       name[8];
  char       prename[8];
  int        pers_nr;
  unsigned   char day_of_birth;
  unsigned   char month_of_birth;
};
```



- Using base address

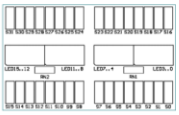
- name:	base address + 0
- prename:	base address + 8
- pers_nr:	base address + 16
- day_of_birth:	base address + 18
- month_of_birth:	base address + 19

`sizeof(struct person) = 20`

Union Memory Layout

Union Memory Layout

School of Engineering
HES Institute of Embedded Systems



```
typedef union {
    struct {
        volatile uint8_t S7_0;
        volatile uint8_t S15_8;
        volatile uint8_t S23_16;
        volatile uint8_t S31_24;
    } BYTE;
    struct {
        volatile uint16_t S15_0;
        volatile uint16_t S31_16;
    } HWORD;
    volatile uint32_t WORD;
} reg_ct_dipsw_t;

#define CT_DIPSW ((reg_ct_dipsw_t *) 0x60000200)
...
sizeof (reg_ct_dipsw_t) = 4
```

Memory Map

S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	0x6000'0200
S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	0x6000'0201
S 23	S 22	S 21	S 20	S 19	S 18	S 17	S 16	0x6000'0202
S 31	S 30	S 29	S 28	S 27	S 26	S 25	S 24	0x6000'0203