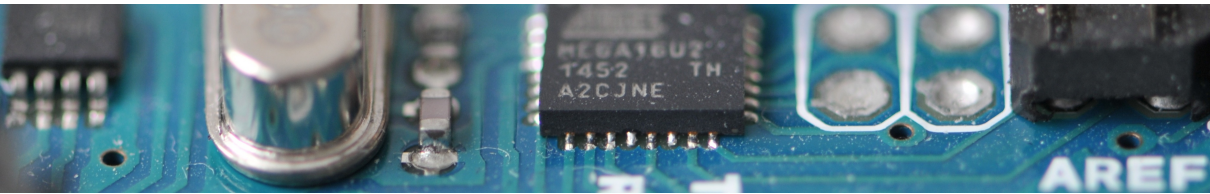
universität
innsbruck



Rechnerarchitektur

Ein-/Ausgabe

Univ.-Prof. Dr.-Ing. Rainer Böhme

Wintersemester 2021/22 · 15. Dezember 2021

Gliederung heute

1. **Touch-Eingabetechnologien**
2. Ansteuerung von E/A-Bausteinen
3. Unterbrechungsanforderungen
4. Praxisbeispiel zur Ausgabe

Technologien zur Berührungsmessung

Berührung wird immer indirekt gemessen.

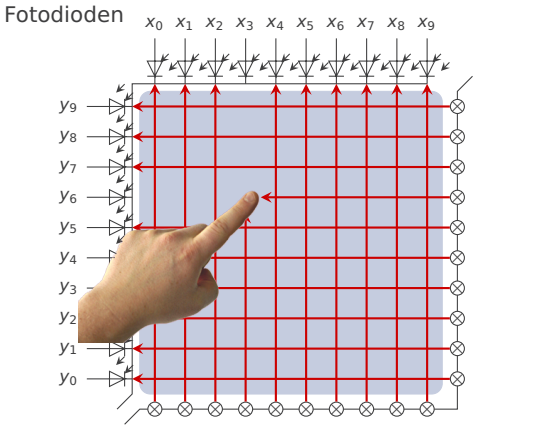
- **Lichtintensität**
  Infrarotgitter, Kamera-basiert
- **Spannung**
  Resistive Verfahren
- **Strom**
  Oberflächen-kapazitive Verfahren
- **Kapazitätsänderung**
  Projiziert-kapazitive Verfahren
- **Zeitverzögerung**
  Akustische Oberflächenwelle
- **Kraft**

Vergleichskriterien

- Haltbarkeit
- Durchsichtigkeit
- Flexibilität der Eingabe (Stift, Finger)
- Multitouch-Fähigkeit
- Kalibrierungsstabilität
- Energiebedarf
- Bauform
- Kosten

Viele verschiedene Technologien mit spezifischen Vor- und Nachteilen

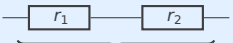
Optische Berührungsmessung



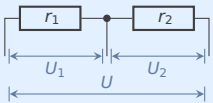
Resistive Berührungsmessung

1D-Modell

Schaltung von Widerständen



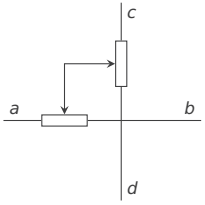
$$r = r_1 + r_2$$

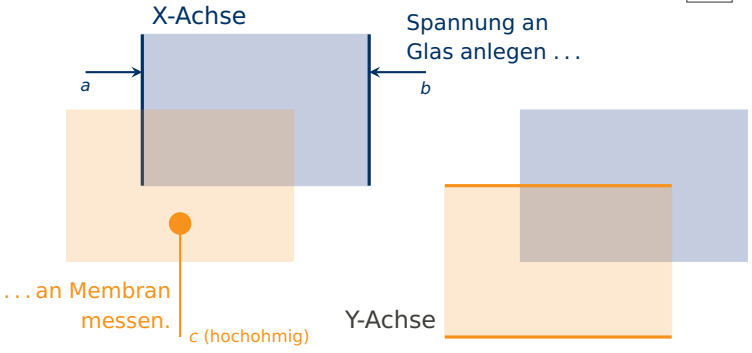


$$U_i = \frac{U \cdot r_i}{r_1 + r_2}$$

Resistive Berührungsmessung

2D-Realisierung

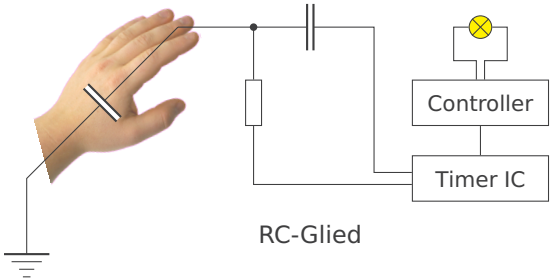




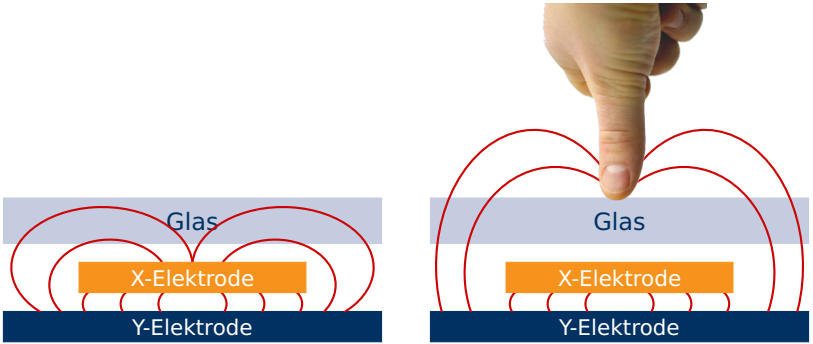
Alternierende Messung von X- und Y-Koordinate

Kapazitive Berührungsmessung

Kapazität beeinflusst Frequenz einer Wechselstromschaltung



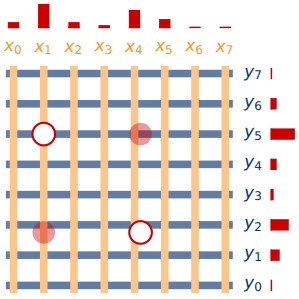
Projiziert-kapazitive Berührungsmessung



- Finger „stehlen“ Ladung von der X-Elektrode. Dadurch ändert sich die Kapazität zwischen den Elektroden.
- Bei Berührung werden die Feldlinien über die berührungsempfindliche Oberfläche hinaus **projiziert**.

Ausleseverfahren bei „Pro-Cap“-Touchscreens

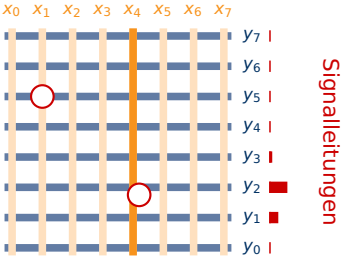
Variante 1: **Perimeter Scan**



- Messwerte $(x_0, \ldots, x_7, y_0, \ldots, y_7)$ sequenziell an A/D-Wandler
- „Geisterpunkte“ bei zwei Fingern

Variante 2: **Imaging**

Steuerleitungen: Spaltenauswahl



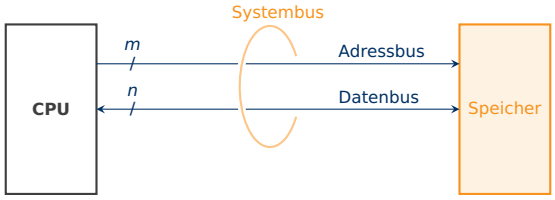
- Smartphones z. B. 9×16, 20–200 Hz
- Interpolierte Auflösung: 1024×1024, ca. 16 Punkte

Gliederung heute

1. Touch-Eingabetechnologien
2. **Ansteuerung von E/A-Bausteinen**
3. Unterbrechungsanforderungen
4. Praxisbeispiel zur Ausgabe

Systembus

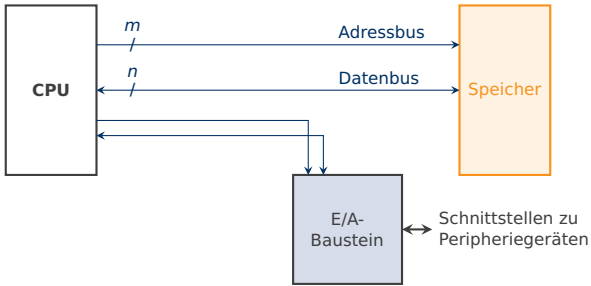
Kommunikationskanal der CPU mit anderen Systemkomponenten



Nicht dargestellt: Takt- und Steuerleitungen ($w/\overline{r}$),
Speicherverwaltungslogik (MMU)

Ansteuerung von E/A-Bausteinen

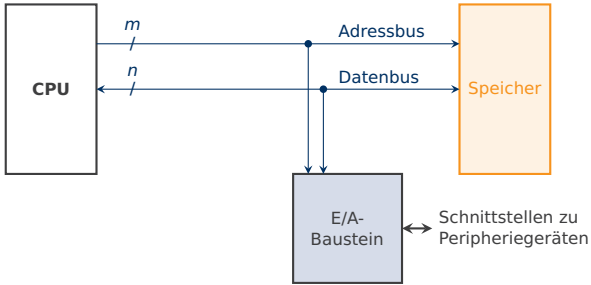
E/A-Bausteine (engl. *I/O controller*) steuern physikalische Schnittstellen.



Variante 1 **Ports** an einem separatem E/A-Bus werden mitspeziellen Instruktionen (z. B. `in`, `out` bei x86) angesprochen.

Ansteuerung von E/A-Bausteinen

E/A-Bausteine (engl. *I/O controller*) steuern physikalische Schnittstellen.



Variante 2 Einblenden der **Register** von E/A-Bausteinen in den (Daten-)Adressraum der CPU. Zugriff wie auf Speicher.

Typen von E/A-Registern

1. **Kontrollregister** zur Initialisierung und Funktionswahl
   - Auswahl ob Leitung als Eingang oder Ausgang arbeitet
   - Aktivierung von Unterbrechungsanforderungen
2. **Datenregister** zum Puffern von ein- oder ausgehenden Daten
   - E/A-Geräte verarbeiten Daten langsamer und asynchron zur CPU.
   - Oft als FIFO-Puffer (*first in, first out*) realisert
3. **Statusregister** zum Austausch von Zustandsinformationen
   - Verfügbarkeit neuer Eingabewerte an Datenregister
   - Abschluss des Sendevorgangs aus Datenregister
   - Timer

   Statusregister werden bei **programmierter Ein-/Ausgabe** regelmäßig vom Hauptprogramm abgefragt (*polling*).

Viele E/A-Register sind nur lesbar oder nur schreibbar.

Beispiel: Zugriff auf System-Timer

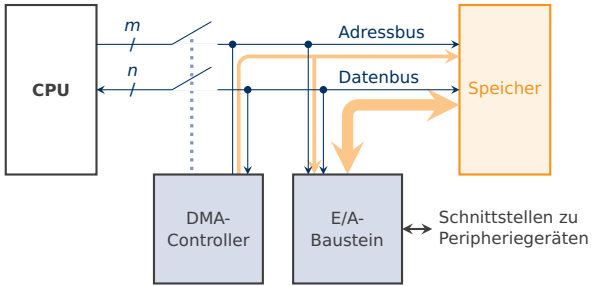
Dieser Timer ist wie ein E/A-Baustein im BCM 2835 des Raspberry Pi Zero integriert.

```
wait:           ; warte r0 Mikrosekunden (r0 · 10^-6 Sekunden)
        STMFD   sp!, {r0-r12,lr} ; Register auf Stapel sichern (Push)

        LDR     r3, =0x20003000  ; Basisadresse des System-Timers
        LDR     r2, [r3, #4]     ; 32-Bit-Mikrosekundenzähler holen
sleep:
        LDR     r1, [r3, #4]     ; Aktuellen Zählerstand
        SUB     r1, r1, r2       ; um Startwert verringern
        CMP     r1, r0           ; und mit Wartezeit vergleichen.
        BLS     sleep            ; Wiederhole, wenn nicht abgelaufen

        LDMFD   sp!, {r0-r12,pc} ; Register wiederherstellen (Pop)
                                 ; und Rücksprung
```

Vorsicht, dieses Beispiel funktioniert nur fast immer. **Warum?**

Speicherdirektzugriff

(engl. *direct memory access*, DMA)



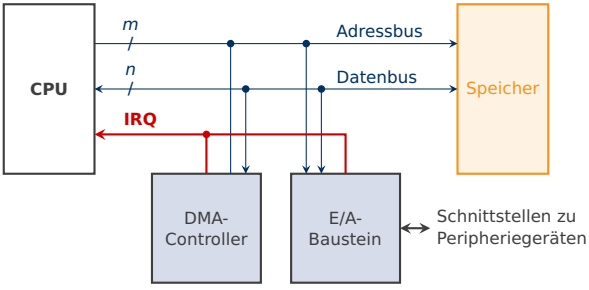
- Effizientere Übertragung großer Datenmengen
- Gleichzeitige Nutzung der CPU für „anspruchsvollere“ Aufgaben

Gliederung heute

1. Touch-Eingabetechnologien
2. Ansteuerung von E/A-Bausteinen
3. **Unterbrechungsanforderungen**
4. Praxisbeispiel zur Ausgabe

Unterbrechungsanforderungen

(engl. *interrupt request*, IRQ)



Alternative zum Polling

- E/A-Bausteine melden Statusänderung über spezielle Leitung.
- CPU verändert Kontrollfluss i. d. R. beim nächsten Fetch-Zyklus.

ARM Exception Vector Table (EVT)

Adresse	Ausnahme (*exception*)	Kommentar
0xffff0000	Reset	(und Systemstart)
0xffff0004	Undefined instruction	nützlich für Softwareemulation
0xffff0008	Software interrupt	SWI-Instruktion
0xffff000c	Prefetch abort	pc enthält unzulässige Adresse
0xffff0010	Data abort	unzulässiger Speicherzugriff
0xffff0018	Interrupt request (IRQ)	
0xffff001c	Fast IRQ (FIQ)	höhere Priorität

- EVT-Einträge enthalten in der Regel eine Sprunganweisung (B) an die Speicheradresse des **Handlers**. Zum Beispiel steht an Adresse 0xffff0008 ein Sprung zum Linux-Syscall-Handler.
- Die höchstwertigen 16 Bit der Speicheradresse des EVT werden durch ein Statusbit festgelegt (z. B. 1 bei Linux, 0 beim CPUlator).

Bei x86: Interrupt Descriptor Table (IDT), abhängig vom Betriebsmodus

Aufbau eines Interrupt-Handlers

Minimalbeispiel für eine Quelle

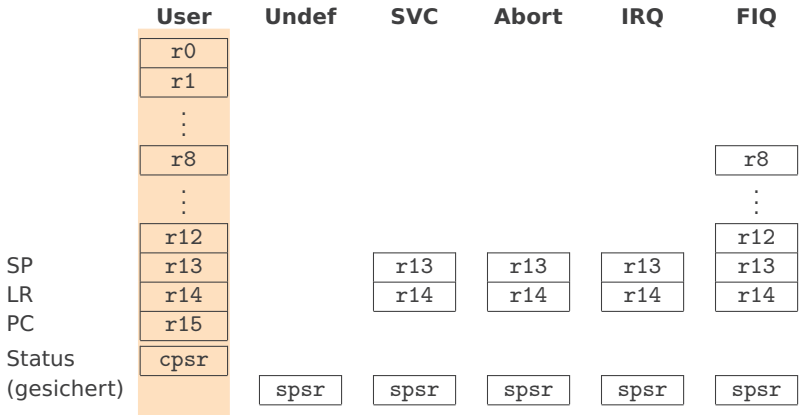
```
irq:
        STMFD   sp!, {r0-r12,lr} ; Register auf Stapel sichern (Push)
        BL      do_something     ; Interrupt-Logik

        LDR     r0, =timerbase   ; Interrupt-Quelle zurücksetzen
        STR     r0, [r0+0x0c]    ; hier z. B. ARM SP804 Timer

        LDMFD   sp!, {r0-r12,lr} ; Register wiederherstellen (Pop)
        SUBS    pc, lr, #4       ; Sprung an korr. Rücksprungadresse
                                 ; und Statusregister wiederherstellen
```

Interrupt-Multiplexing

Wenn mehrere Quellen IRQs auslösen, findet der Interrupt-Handler die Quelle heraus (durch Auslesen aller infrage kommenden *interrupt pending*-Bits) und verzweigt in das dazugehörige Unterprogramm.

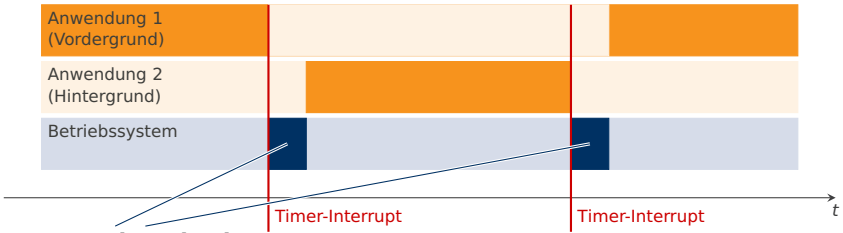
Registersatz mit Bänken

Abhängig vom **Modus** trennt ARM einige Registerinhalte nach Bänken:



Einsatz zum Mehrprozessbetrieb auf einem Kern

(engl.: *multitasking*, *time sharing*)



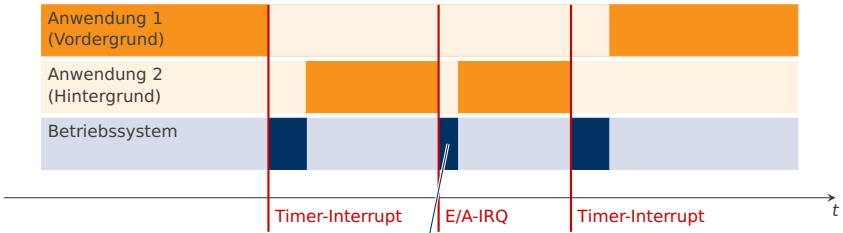
Taskwechsel

- Register sichern
- Stack-Rahmen anpassen (pro Prozess ein Stapel)
- Zugriffsrechte (Kontext) anpassen
- Register des nächsten Prozesses wiederherstellen
- „Rücksprung“

Vertiefung in **Betriebssysteme**, Pflichtmodul, 2. Semester

Einsatz zum Mehrprozessbetrieb auf einem Kern

(engl.: *multitasking*, *time sharing*)



E/A-Baustein, z. B.

- Mausbewegung
- Datenpaket vom Netz

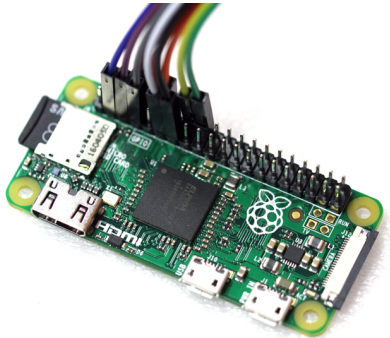
Vertiefung in **Betriebssysteme**, Pflichtmodul, 2. Semester

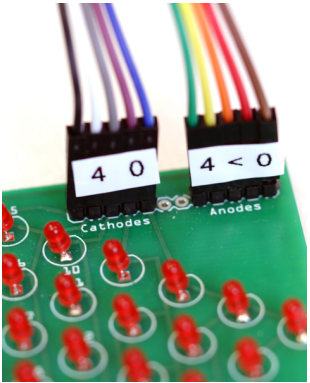
Gliederung heute

1. Touch-Eingabetechnologien
2. Ansteuerung von E/A-Bausteinen
3. Unterbrechungsanforderungen
4. **Praxisbeispiel zur Ausgabe**

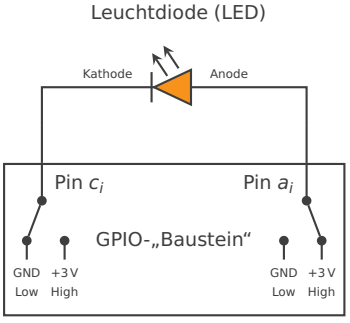
Hardware

Der Raspberry Pi Zero V1.3 basiert auf einem System-on-a-Chip (SoC) mit 1 GHz ARM-CPU und *general purpose I/O* (GPIO)-Pins.





Ansteuerung einer Leuchtdiode



Pin		
c_i	a_i	LED
L	L	aus
L	H	an
H	L	aus
H	H	aus

Software

Drei Schritte zur Initialisierung (LEDs aus)

1. Pins im Kontrollregister als Ausgänge schalten

```
LDR   r9, =0x20200000 ; Basisadresse der GPIO-Komponente
MOV   r0, #1          ; Bit-Kodierung 001 für „Ausgang“ nach r0
STR   r0, [r9, #0]    ; setze Funktion in GPFSEL-Register
...                   ; wiederholen für weitere Pins
```

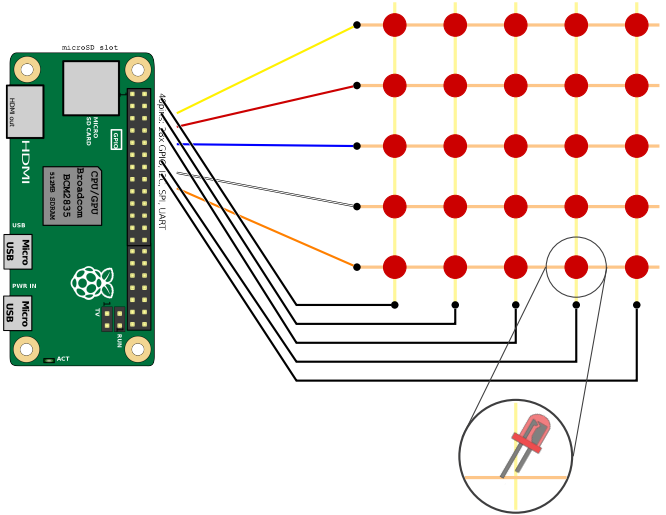
2. Kathoden auf H schalten (über Set-Datenregister)

```
                      ; r1 enthält gesetztes Bits pro Kathoden-Pin
STR   r1, [r9, #28]   ; Bits in GPSET0-Register (Basis+28) setzen
```

3. Anoden auf L schalten (über Clear-Datenregister)

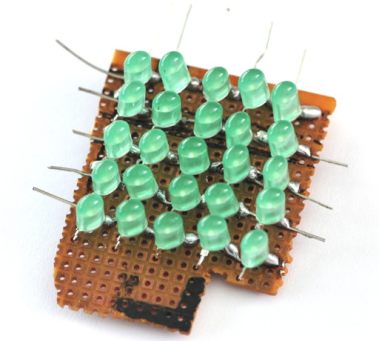
```
                      ; r2 enthält gesetztes Bit pro Anoden-Pin
STR   r2, [r9, #40]   ; Bits in GPCLR0-Register setzen
```

Schaltung für 25 LEDs



Erste Umsetzung

für eine Ansteuerung mit 10 Pins



Bits im GPIO-Datenregister	
Kathoden	2, 3, 4, 17, 27
Anoden	14, 15, 18, 23, 24

→ Ansteuerung unter Nutzung menschlicher Wahrnehmungsschwächen

Registerbelegung

Register	Funktion
`r0`	Hilfsregister, temporäre Belegung
`r1`	Kathoden-Maske: GPIO_0 *clear* für LED an; in Spalten
`r2`	Anoden-Maske: GPIO_0 *set* für LED an; in Zeilen
`r3`	Laufvariable für Kathoden-Pins (stets nur ein Bit gesetzt)
`r4`	Laufvariable für Anoden-Pins (stets nur ein Bit gesetzt)
`r5`	Laufvariable für LED $0, \ldots, 24$
`r6`	LED-Bild in Bits 0–24 (Bit gesetzt $\Rightarrow$ LED an)
`r7`	Spaltenbild: Einsen für alle Zeilen-Pins, wo LEDs leuchten
`r8`	Schleifenzähler der Bildwiederholung
`r9`	Basisadresse der GPIO-Register

Steuerung der Laufvariablen für Pins

Idee: Ein Bit „läuft“ durch Rotation alle Pins ab.
Masken-Register zeigen gültige Bit-Positionen an.

```
            LDR     r1, =0x0802001c ; Kathoden-Maske
            LDR     r2, =0x0184c000 ; Anoden-Maske
            LDR     r3, =0x00000004 ; Laufvariable für Kathoden-Pins
            LDR     r4, =0x00004000 ; Laufvariable für Anoden-Pins
            ...
shiftAnod:
            TST     r2, r4          ; testen, ob die Eins richtig steht
            MOVEQ   r4, r4, ROR #31 ; falls nicht, eins nach links rotieren
            BEQ     shiftAnod
shiftCath:
            TST     r1,r3           ; analog für Kathoden
            ...
```

Ablaufskizze

r5	r3	r4
0	0x00000004	0x00004000
1	0x00000004	0x00008000
2	0x00000004	0x00040000
3	0x00000004	0x00800000
4	0x00000004	0x01000000
5	0x00000008	0x00004000
⋮	⋮	⋮
9	0x00000008	0x01000000
⋮	⋮	⋮
24	0x08000000	0x01000000
Masken	r1	r2
	0x0802001c	0x0184c000

Bitweise OR-Verknüpfung aller Belegungen von `r4`, bei denen LED an $\Leftrightarrow$ $\texttt{r6}_{(\texttt{r5})} = 1$.

← **LED-Spalte ein, warten, aus**

nächste Spalte

← usw.

Spaltensteuerung

LED-Bild befindet sich in Bits 0–24 von `r6`.

```
            MOV     r5, #0          ; LED-Zähler initialisieren
            ...
rowloop:
            MOV     r7, #0          ; Spaltenbild leeren
            ...
shiftAnod:  ...
            MOV     r0, r6, LSR r5  ; Bitmaske nach rechts schieben
            TST     r0, #1          ; testen, ob diese LED leuchten soll
            ORRNE   r7, r7, r4      ; falls ja, OR-verknüpfen

            ADD     r5, r5, #1      ; 25 LED Zähler inkrementieren
            MOV     r4, r4, ROR #31 ; Anoden-Maske verschieben
            CMP     r4, r2          ; Spalte fertig ?
            BLO     shiftAnod       ; wiederhole, falls nicht
```

Ansteuerung des GPIO-Bausteins

Programmierte Ein-/Ausgabe über Datenregister

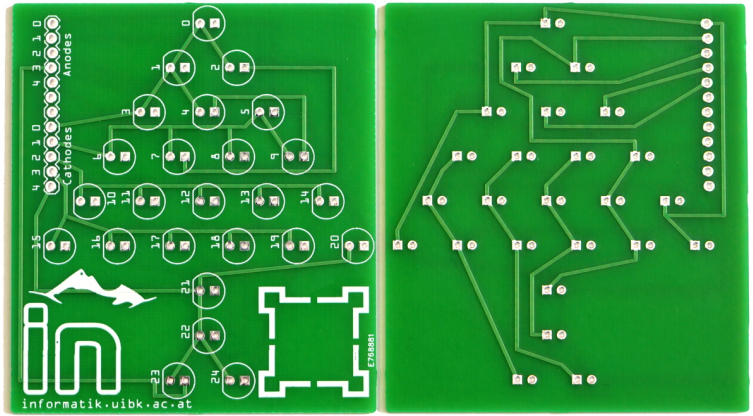
(LED-Spalte ein, warten, aus)

```
STR   r7, [r9, #28] ; gesammelte Anoden auf H
STR   r3, [r9, #40] ; entsprechende Kathoden auf L

LDR   r0, =50       ; Wartezeit 50 µs
BL    wait          ; aktives Warten über Timer (Folie 16)

STR   r3, [r9, #28] ; Kathoden wieder H
STR   r7, [r9, #40] ; Anoden wieder L

MOV   r7, #0        ; Spaltenbild zurücksetzen
```

Wir ersparen uns (und Ihnen) den Rest der Schleifenlogik.

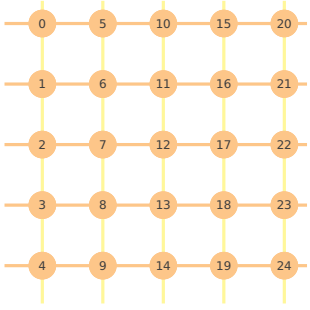
Leiterplatte

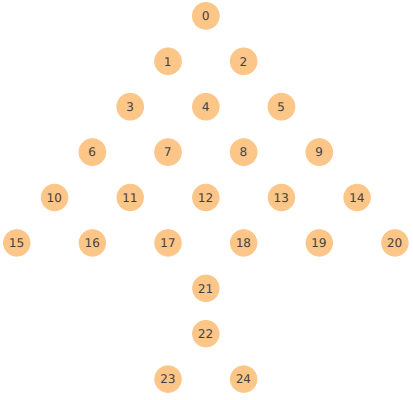
(engl. *printed circuit board*, PCB)



Vorderseite Rückseite

Von der Matrix zum Baum



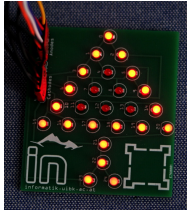


Beispielanimation: Rotation eines Bits in $\texttt{r6}_{(24)}, \ldots, \texttt{r6}_{(0)}$.

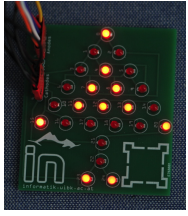
Hoh, Hoh, Hörsaalfrage

24 82 94 16

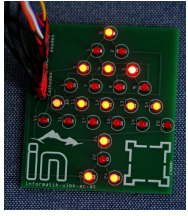
Welche Ausgabe erhalten Sie, wenn Sie das gezeigte Programm mit dem Wert `0x00262a6f` in Register `r6` aufrufen?



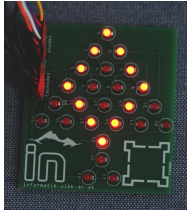
Antwort A



Antwort B



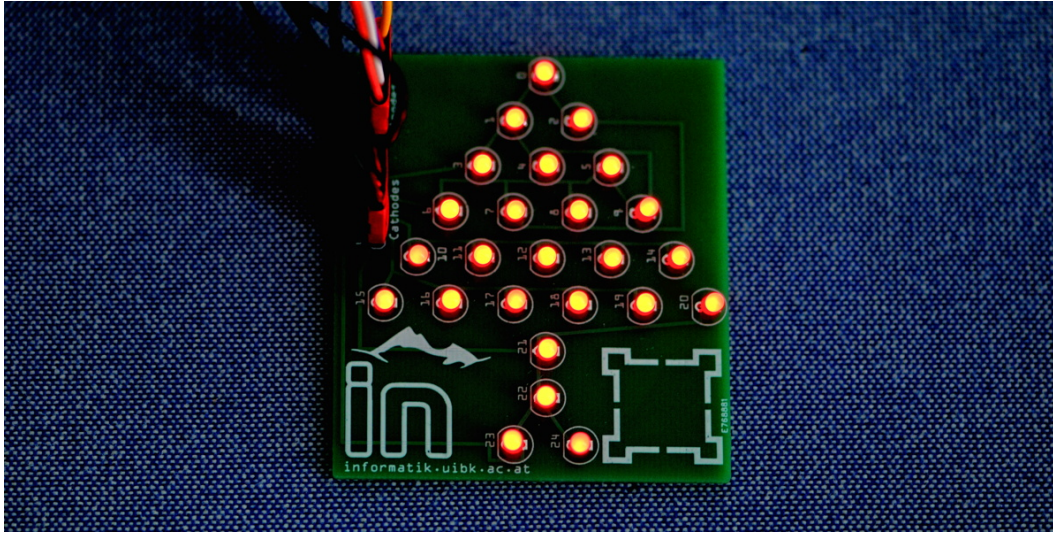
Antwort C



Antwort D

Zugang: `https://arsnova.uibk.ac.at` mit Zugangsschlüssel **24 82 94 16**. Oder scannen Sie den QR-Kode.

Frohes Fest



Ihr **Rechnerarchitektur** Team

Syllabus – Wintersemester 2021/22

06.10.21	1.	Einführung
13.10.21	2.	Kombinatorische Logik I
20.10.21	3.	Kombinatorische Logik II
27.10.21	4.	Sequenzielle Logik I
03.11.21	5.	Sequenzielle Logik II
10.11.21	6.	Arithmetik I
17.11.21	7.	Arithmetik II
24.11.21	8.	Befehlssatzarchitektur (ARM) I
01.12.21	9.	Befehlssatzarchitektur (ARM) II
15.12.21	10.	Ein-/Ausgabe
12.01.22	11.	Prozessorarchitekturen
19.01.22	12.	Speicher
26.01.22	13.	Leistung
02.02.22		**Klausur (1. Termin)**