

# Mikroprozessortechnik

**Prof. Dr. Michael Lipp**

# Super Loop (Aktuelle Anwendungs- architektur)

# Super Loop Architektur

- **Bislang: Reine Super-Loop Architektur**

- Initialisierung
- Endlosschleife
  - Abfragen, ob Bedingung 1 eingetreten ist
  - Wenn ja, bearbeiten
  - Abfragen, ob Bedingung 2 eingetreten ist
  - Wenn ja, bearbeiten
  - Abfragen, ob Bedingung 3 eingetreten ist
  - Wenn ja, bearbeiten
  - ...

```
// Labor 1:
```

```
void task7() {  
    CPolledTimer twoHz(500);  
    CPolledTimer threeHz(333);  
    CPolledTimer fourHz(250);  
    while (true) {  
        if (twoHz.timeReached()) {  
            leds = leds ^ (1 << 0);  
        }  
        if (threeHz.timeReached()) {  
            leds = leds ^ (1 << 1);  
        }  
        if (fourHz.timeReached()) {  
            leds = leds ^ (1 << 2);  
        }  
    }  
}
```

# Super Loop Architektur

- **Nachteile**

- Offensichtlich

- Bearbeitung einer eingetretenen Bedingung blockiert den Prozessor
      - → Behandlung von wichtigen Ereignissen kann „beliebig“ verzögert werden
    - Komponenten können nicht vollständig gekapselt werden
      - Beispiel 7-Segment-Anzeige: `CPolledTimer` für die Anzeige in `main` definiert

- Weniger offensichtlich

- ...

- **Verbesserung für zeitnahe Bearbeitung von (wichtigen) Ereignissen → Interrupts**

# Interrupts

# Interrupts

- **Ein Interrupt ist eine Unterbrechung des „regulären“ Programmablaufs**
- **Ein Interrupt wird durch einen Zustand oder Zustandswechsel einer Peripherieeinheit ausgelöst („Ereignis“)**
  - Beispiel: Grenzwertüberschreitung des ADC
- **Der Interrupt führt zur sofortigen (genauer: möglichst zeitnahen) Ausführung von Code**
  - Dieser Code (Interrupt Handler) behandelt den Zustand oder Zustandswechsel und setzt ihn zurück (oder deaktiviert die erneute Auslösung des Interrupts)

# Interrupts

- **Aufruf des Interrupt Handler ist einem „erzwungenen“ Unterprogrammaufruf vergleichbar**

... Unterprogrammaufruf ... ???

# Unterprogramm- aufrufe (Wdh./Vertiefung)



# Wdh: Unterprogrammaufruf

- **Unterprogrammaufruf**

- Beispiel:

```
void main(void){
```

```
    int i,k;
```

```
    i = sub(13);
```

```
    k = sub(14);
```

```
    // ...
```

```
}
```

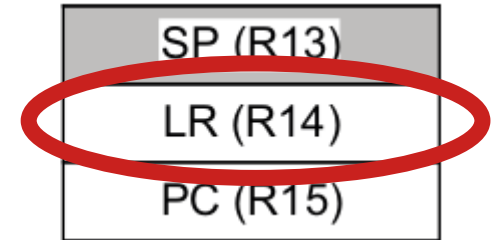
```
int sub(int i){
```

```
    return i*2;
```

```
}
```

- Spring aus main nach sub (PC  $\leftarrow$  sub)
- Welchen Wert lädt return in den PC?

Stack Pointer  
Link Register  
Program Counter



# Wdh.: Unterprogrammaufruf

- **Lösung**

- Sprung zu sub mit „branch with link“ („bl“)
  - Speichert die Adresse des nächsten Befehls (nach „bl“) im Link-Register
  - Lädt PC mit Adresse des Sprungziels
- Rücksprung
  - Lade den Wert von lr in pc („bx lr“)

# Wdh.: Unterprogrammaufruf

MIT-ARM - AssemblerLinkRegister/myCode/main.s - STM32CubeIDE

File Edit Navigate Search Project Run Window Help

Debug Project Explorer

AssemblerLinkRegister [STM32 Cortex-M C/C++ Application]

AssemblerLinkRegister.elf [cores: 0]

Thread #1 [main] 1 [core: 0] (Suspended : Step)

Function\_Test() at main.s:27 0x800019e

Reset\_Handler() at main.s:20 0x8000198

arm-none-eabi-gdb (8.1.0.20180315)

ST-LINK (ST-LINK GDB server)

main.s

```

17 /* Unterprogramm(oder Funktions-)aufruf. C-Äquivalent:
18 * Function_Test();
19 */
20 bl Function_Test // Erster Aufruf
21 bl Function_Test // Zweiter Aufruf
22
23 Main_Loop:
24 b Main_Loop // Springe zum Befehl nach dem Label "Main_Loop"
25
26 Function_Test:
27 bx lr // Branch and Exchange, PC wird mit Inhalt von LR geladen
28
29 .global Default_Handler
30 Default_Handler:
31 Infinite_Loop:
32 b Infinite_Loop
33
34 // A minimal vector table for a Cortex M3, from CubeMX generated code.
35 .section .isr_vector,"a",%progbits
36
37 .word _estack
38 .word Reset_Handler
39 .word NMI_Handler
40 .word HardFault_Handler
41 .word MemManage_Handler
42 .word BusFault_Handler
43 .word UsageFault_Handler
44 .word 0
45 .word 0
46 .word 0
47 .word 0
48 .word 0
49 .word 0
50 .word 0
51 .word 0
52 .word 0
53 .word 0
54 .word 0
55 .word 0
56 .word 0
57 .word 0
58 .word 0
59 .word 0
60 .word 0
61 .word 0
62 .word 0
63 .word 0
64 .word 0
65 .word 0
66 .word 0
67 .word 0
68 .word 0
69 .word 0
70 .word 0
71 .word 0
72 .word 0
73 .word 0
74 .word 0
75 .word 0
76 .word 0
77 .word 0
78 .word 0
79 .word 0
80 .word 0
81 .word 0
82 .word 0
83 .word 0
84 .word 0
85 .word 0
86 .word 0
87 .word 0
88 .word 0
89 .word 0
90 .word 0
91 .word 0
92 .word 0
93 .word 0
94 .word 0
95 .word 0
96 .word 0
97 .word 0
98 .word 0
99 .word 0
100 .word 0
101 .word 0
102 .word 0
103 .word 0
104 .word 0
105 .word 0
106 .word 0
107 .word 0
108 .word 0
109 .word 0
110 .word 0
111 .word 0
112 .word 0
113 .word 0
114 .word 0
115 .word 0
116 .word 0
117 .word 0
118 .word 0
119 .word 0
120 .word 0
121 .word 0
122 .word 0
123 .word 0
124 .word 0
125 .word 0
126 .word 0
127 .word 0
128 .word 0
129 .word 0
130 .word 0
131 .word 0
132 .word 0
133 .word 0
134 .word 0
135 .word 0
136 .word 0
137 .word 0
138 .word 0
139 .word 0
140 .word 0
141 .word 0
142 .word 0
143 .word 0
144 .word 0
145 .word 0
146 .word 0
147 .word 0
148 .word 0
149 .word 0
150 .word 0
151 .word 0
152 .word 0
153 .word 0
154 .word 0
155 .word 0
156 .word 0
157 .word 0
158 .word 0
159 .word 0
160 .word 0
161 .word 0
162 .word 0
163 .word 0
164 .word 0
165 .word 0
166 .word 0
167 .word 0
168 .word 0
169 .word 0
170 .word 0
171 .word 0
172 .word 0
173 .word 0
174 .word 0
175 .word 0
176 .word 0
177 .word 0
178 .word 0
179 .word 0
180 .word 0
181 .word 0
182 .word 0
183 .word 0
184 .word 0
185 .word 0
186 .word 0
187 .word 0
188 .word 0
189 .word 0
190 .word 0
191 .word 0
192 .word 0
193 .word 0
194 .word 0
195 .word 0
196 .word 0
197 .word 0
198 .word 0
199 .word 0
200 .word 0
201 .word 0
202 .word 0
203 .word 0
204 .word 0
205 .word 0
206 .word 0
207 .word 0
208 .word 0
209 .word 0
210 .word 0
211 .word 0
212 .word 0
213 .word 0
214 .word 0
215 .word 0
216 .word 0
217 .word 0
218 .word 0
219 .word 0
220 .word 0
221 .word 0
222 .word 0
223 .word 0
224 .word 0
225 .word 0
226 .word 0
227 .word 0
228 .word 0
229 .word 0
230 .word 0
231 .word 0
232 .word 0
233 .word 0
234 .word 0
235 .word 0
236 .word 0
237 .word 0
238 .word 0
239 .word 0
240 .word 0
241 .word 0
242 .word 0
243 .word 0
244 .word 0
245 .word 0
246 .word 0
247 .word 0
248 .word 0
249 .word 0
250 .word 0
251 .word 0
252 .word 0
253 .word 0
254 .word 0
255 .word 0
256 .word 0
257 .word 0
258 .word 0
259 .word 0
260 .word 0
261 .word 0
262 .word 0
263 .word 0
264 .word 0
265 .word 0
266 .word 0
267 .word 0
268 .word 0
269 .word 0
270 .word 0
271 .word 0
272 .word 0
273 .word 0
274 .word 0
275 .word 0
276 .word 0
277 .word 0
278 .word 0
279 .word 0
280 .word 0
281 .word 0
282 .word 0
283 .word 0
284 .word 0
285 .word 0
286 .word 0
287 .word 0
288 .word 0
289 .word 0
290 .word 0
291 .word 0
292 .word 0
293 .word 0
294 .word 0
295 .word 0
296 .word 0
297 .word 0
298 .word 0
299 .word 0
300 .word 0
301 .word 0
302 .word 0
303 .word 0
304 .word 0
305 .word 0
306 .word 0
307 .word 0
308 .word 0
309 .word 0
310 .word 0
311 .word 0
312 .word 0
313 .word 0
314 .word 0
315 .word 0
316 .word 0
317 .word 0
318 .word 0
319 .word 0
320 .word 0
321 .word 0
322 .word 0
323 .word 0
324 .word 0
325 .word 0
326 .word 0
327 .word 0
328 .word 0
329 .word 0
330 .word 0
331 .word 0
332 .word 0
333 .word 0
334 .word 0
335 .word 0
336 .word 0
337 .word 0
338 .word 0
339 .word 0
340 .word 0
341 .word 0
342 .word 0
343 .word 0
344 .word 0
345 .word 0
346 .word 0
347 .word 0
348 .word 0
349 .word 0
350 .word 0
351 .word 0
352 .word 0
353 .word 0
354 .word 0
355 .word 0
356 .word 0
357 .word 0
358 .word 0
359 .word 0
360 .word 0
361 .word 0
362 .word 0
363 .word 0
364 .word 0
365 .word 0
366 .word 0
367 .word 0
368 .word 0
369 .word 0
370 .word 0
371 .word 0
372 .word 0
373 .word 0
374 .word 0
375 .word 0
376 .word 0
377 .word 0
378 .word 0
379 .word 0
380 .word 0
381 .word 0
382 .word 0
383 .word 0
384 .word 0
385 .word 0
386 .word 0
387 .word 0
388 .word 0
389 .word 0
390 .word 0
391 .word 0
392 .word 0
393 .word 0
394 .word 0
395 .word 0
396 .word 0
397 .word 0
398 .word 0
399 .word 0
400 .word 0
401 .word 0
402 .word 0
403 .word 0
404 .word 0
405 .word 0
406 .word 0
407 .word 0
408 .word 0
409 .word 0
410 .word 0
411 .word 0
412 .word 0
413 .word 0
414 .word 0
415 .word 0
416 .word 0
417 .word 0
418 .word 0
419 .word 0
420 .word 0
421 .word 0
422 .word 0
423 .word 0
424 .word 0
425 .word 0
426 .word 0
427 .word 0
428 .word 0
429 .word 0
430 .word 0
431 .word 0
432 .word 0
433 .word 0
434 .word 0
435 .word 0
436 .word 0
437 .word 0
438 .word 0
439 .word 0
440 .word 0
441 .word 0
442 .word 0
443 .word 0
444 .word 0
445 .word 0
446 .word 0
447 .word 0
448 .word 0
449 .word 0
450 .word 0
451 .word 0
452 .word 0
453 .word 0
454 .word 0
455 .word 0
456 .word 0
457 .word 0
458 .word 0
459 .word 0
460 .word 0
461 .word 0
462 .word 0
463 .word 0
464 .word 0
465 .word 0
466 .word 0
467 .word 0
468 .word 0
469 .word 0
470 .word 0
471 .word 0
472 .word 0
473 .word 0
474 .word 0
475 .word 
```



Live Coding

# Registerbehandlung

- **ARM Calling Convention**

- Offiziell: AAPCS (ARM Architecture Procedure Call Standard)
- R0-R3: Argumente / Rückgabewert(e), können von der Subroutine überschrieben werden
- R4-R11: Lokale Variablen, dürfen von der Subroutine **nicht** überschrieben werden, d.h. wenn benötigt, Werte auf dem Stack sichern und vor Rücksprung wiederherstellen
- R12: Hilfsregister für Aufruf der Subroutine, Wert muss nicht erhalten werden

# Beispielfunktion

- **C-Code:**

```
int32_t add(int32_t a, int32_t b) {  
    return a + b;  
}
```

- **Assembler-Code:**

```
add    r0, r1  
bx     lr
```

Für diese einfache Funktion  
reichen die verfügbaren  
Register aus.

# Beispielfunktion

- **C-Code:**

```
int32_t calculate(int32_t a, int32_t b, int32_t c, int32_t d) {  
    return add(a, b) * add(c, d);  
}
```

- **Assembler-Code:**

```
// Es werden 3 zusätzliche Register benötigt ...  
push {r4, r5, r6, lr} // ... und die return-Adresse muss gespeichert werden.  
mov r5, r2 // r2 in r5 speichern ...  
mov r6, r3 // ... und r6 in r3 ...  
bl add // ... da der Unterprogrammaufruf r2 und r3 ändern darf.  
mov r4, r0 // Zwischenergebnis in r4 speichern.  
mov r1, r6 // In r5 und r6 gespeicherte Argumente in r1 ...  
mov r0, r5 // ... und in r0 übertragen  
bl add  
muls r0, r4 // Zwischenergebnisse multiplizieren, Endergebnis in r0  
pop {r4, r5, r6, pc} // „Geschicktes“ return, lr → pc
```

# (Zurück zum Interrupt)

# Interrupt

- **Aufruf Interrupt Handler (Reaktion auf eingetretene Bedingung)**
  - Zustandsregister, PC, LR, R12 und R3-R0 auf dem Stack speichern (dürfen von „normalem“ Unterprogramm verändert werden)
  - LR wird mit „Spezialwert“ geladen, so dass beim „Return“ ( $PC \leftarrow LR$ ) auf dem Stack gespeicherten Informationen wiederhergestellt werden
  - Register, die von einem „normalen“ Unterprogramm nicht verändert werden dürfen, dürfen auch von Funktionen zur Interrupt-Behandlung nicht verändert werden (müssen auf dem Stack „gerettet“ werden)
    - D. h. der Code für Funktionen für die Interrupt-Behandlung unterscheidet sich nicht von dem Code „normaler“ Funktionen
  - Funktionen für die Interrupt-Behandlung haben keine Parameter
    - (Woher sollten auch die Argumente kommen?)



# Interrupt

- **Interrupt Handler**

- Unterschiedliche Interrupt Handler für unterschiedliche Hardware-Komponenten
  - Sonst müsste man im Handler erst alle möglichen Auslöser prüfen
- Zuordnung „Ereignis“ zu Interrupt Handler über den Vector Table
  - „Fortsetzung“ der beim Booten verwendeten Werte an den Adressen 0x0 und 0x4
- Vector Table enthält die Start-Adressen der Interrupt Handler

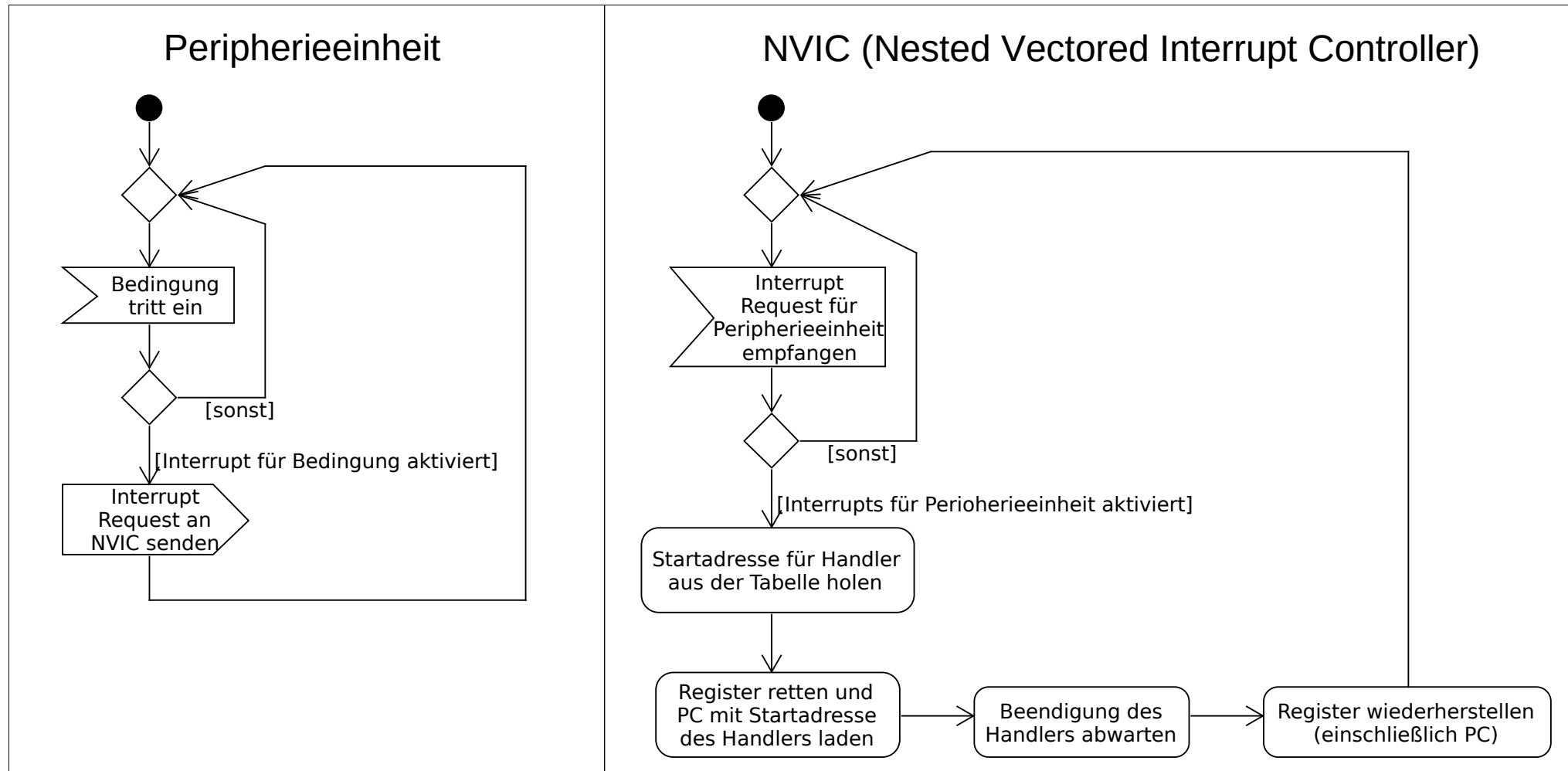
# Vector Table

Table 38. Vector table for STM32F401xB/CSTM32F401xD/E

Position	Priority	Type of priority	Acronym	Description	Address
...					
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	EXTI16 / PVD	EXTI Line 16 interrupt / PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	EXTI21 / TAMP_STAMP	EXTI Line 21 interrupt / Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	EXTI22 / RTC_WKUP	EXTI Line 22 interrupt / RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058

Quelle:  
[F401-RM]

# Gesamtablauf (Verhalten der Hardware)



# Beispiel ADC-Interrupt

- **Mbed OS unterstützt keine ADC Interrupts**
  - ADC muss manuell konfiguriert werden
  - Minimalkonfiguration
    - Beschreibungen im Reference Manual bzw. Datenblatt lesen
    - GPIO-Pin als Analogeingang konfigurieren (vergl. Vorlesung 2)
    - ADC mit Takt versorgen
    - ADC einschalten
    - Zu konvertierenden Kanal festlegen
- **(Takt für Peripherieeinheiten)**
  - Versorgung der Peripherieeinheiten mit Takt wird später im Detail behandelt, im Moment Beispielvorgehen verwenden

# ADC-Konfigurationstest mit Polling

```
int main() {
    keys.mode(PullDown);

    // Eingang PB0 als "Analog" konfigurieren (vergl. Vorlesung 2).
    MODIFY_REG(GPIOB->MODER, 0, GPIO_MODER_MODER0);
    MODIFY_REG(GPIOB->OTYPER, GPIO_OTYPER_OT0, 0);
    MODIFY_REG(GPIOB->OSPEEDR, GPIO_OSPEEDER_OSPEEDR0, 0);
    MODIFY_REG(GPIOB->PUPDR, GPIO_PUPDR_PUPD0, 0);

    // Sicherstellen, dass ADC mit Takt versorgt wird und ADC einschalten
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
    ADC1->CR2 |= ADC_CR2_ADON;

    // PB0 ist mit ADC1_IN8 verbunden ([F401-DS] Table 8). Länge der Sequenz
    // der zu konvertierenden Eingänge auf 1 setzen und ADC1_IN8 als ersten
    // (und einzigen) abzufragenden Kanal eintragen (s. F401-RM 11.3.3
    // und 11.12.9).
    MODIFY_REG(ADC1->SQR1, ADC_SQR1_L, (1 << ADC_SQR1_L_Pos));
    MODIFY_REG(ADC1->SQR3, ADC_SQR3_SQ1, (8 << ADC_SQR3_SQ1_Pos));

    // Prescaler (bestimmt Takt, s. Vorlesung)
    MODIFY_REG(ADC1_COMMON->CCR, 0, (1 << ADC_CCR_ADCPRE_Pos));

    task1();
}
```

```
/**
 * Beispiel für die Nutzung des direkt programmierten ADC im
 * "polling" (Abfrage-)Modus.
 */
void task1() {
    while (true) {
        // Polling: Konvertierung starten und auf Ende warten
        ADC1->CR2 |= ADC_CR2_SWSTART;
        while (!(ADC1->SR & ADC_SR_EOC)) {
        }

        // Ergebnis anzeigen
        leds = ADC1->DR >> 4;
    }
}
```

Live Coding

# Beispiel ADC Interrupt

## • Aktivierbare ADC-Interrupts

### 11.12.2 ADC control register 1 (ADC\_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	Bit 5					EOCIE: Interrupt enable					
Reserved					OVRIE	RES		AWDEN	JAWDEN									This bit is set and cleared by software. 0: EOC interrupt disabled 1: EOC interrupt enabled				
					rw		rw	rw	rw													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]											
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw						

Bit 26 **OVRIE**: Overrun interrupt enable

This bit is set and cleared by software to enable/disable the Overrun interrupt.

0: Overrun interrupt disabled

1: Overrun interrupt enabled. An interrupt is generated when the OVR bit is set.

Bit 7 **JEOCIE**: Interrupt enable for injected channels

This bit is set and cleared by software to enable/disable the end of conversion interrupt for injected channels.

0: JEOC interrupt disabled

1: JEOC interrupt enabled. An interrupt is generated when the JEOC bit is set.

Bit 6 **AWDIE**: Analog watchdog interrupt enable

This bit is set and cleared by software to enable/disable the analog watchdog interrupt.

0: Analog watchdog interrupt disabled

1: Analog watchdog interrupt enabled

Bit 5 **EOCIE**: Interrupt enable for EOC

This bit is set and cleared by software to enable/disable the end of conversion interrupt.

0: EOC interrupt disabled

1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

# Beispiel ADC Interrupt

Table 38. Vector table for STM32F401xB/CSTM32F401xD/E

Position	Priority	Type of priority	Acronym	Description	Address
18	25	settable	ADC	ADC1 global interrupts	0x0000 0088

- **„Anmelden“ eines Interrupt Handlers**
  - Vector Table enthält Adressen von „Default Handlern“
  - Werden durch Definition einer C-Funktion mit gleichem Namen ersetzt
  - Standard-Eintrag bei 0x000000088: ADC\_IRQHandler

# Einschub: C- und C++ Funktionen

- **C- und C++-Funktionen sehen gleich aus**
  - C-Funktion: `void f();`
  - C++-Funktion: `void f();`
- **Tatsächlich befinden sich die Funktionen aber in unterschiedlichen „Namensräumen“**
  - Eine in C++ Quelltext definierte Funktion `void f()` kann aus C nicht aufgerufen werden, außer sie ist definiert (oder deklariert) als  
**extern "C"** `void f() {...}`  
d. h. „extern“ wird sie wie eine C-Funktion behandelt



# Beispiel ADC-Interrupt

- **Umstellen auf Interrupt-Betrieb (allgemein)**
  - Interrupt-Handler definieren
    - Im Interrupt-Handler sicherstellen, dass die Ursache des Interrupts beseitigt wird (sonst wird der Interrupt-Handler immer wieder aufgerufen)
  - Am zentralen Interrupt-Controller (NVIC) die Verarbeitung von Interrupts der Peripherieeinheit aktivieren
    - Flag in einem Register, kann über Makro `__NVIC_EnableIRQ(InterruptNummer)`; gesetzt werden
  - In der Konfiguration der Peripherieeinheit das Auslösen eines Interrupts unter den gewünschten Bedingungen aktivieren

# Beispiel ADC-Interrupt

Live Coding

```
/**
 * Beispiel für die Nutzung des direkt programmierten ADC im
 * Interrupt-Modus.
 */
void task2() {
    // Interrupt-getrieben: Interrupts einschalten ...
    __NVIC_EnableIRQ(ADC_IRQn); // ... ADC-Interrupts allgemein und ...
    ADC1->CR1 |= ADC_CR1_EOCIE; // ... speziell den EOC-Interrupt

    // (Erste) Konvertierung starten
    ADC1->CR2 |= ADC_CR2_SWSTART;

    while (true) {
    }
}
```

```
/**
 * Interrupt-Handler für den ADC. Gibt den Konvertierten Wert auf den
 * LEDs aus und startet eine neue Konvertierung.
 */
extern "C" void ADC_IRQHandler() {
    // Prüfen, ob der Interrupt wegen eines abgeschlossenen
    // Konvertierungsvorgangs ausgelöst wurde.
    if (ADC1->SR & ADC_SR_EOC) {
        // Lesen des DR Registers setzt das Interrupt-Flag zurück.
        leds = ADC1->DR >> 4;
        ADC1->CR2 |= ADC_CR2_SWSTART;
    }
}
```