Multitasking / virtuelle Maschinen mittels Atmel AVR-Mikrocontrollern (Simple & Stupid)

Stand: 26. 1. 2010

Zweck:
Elementare Demonstration der Mehrprogrammausführung auf Grundlage eines Atmel-AVR-Mikrocontrollers.

Der Ansatz soll ganz einfach und schmucklos sein. Prinzip: Multitasking mit zyklischer Zwangsumschaltung (Round Robin) über Counter/Timer. Jede Task ist eine Art virtueller Maschine. "Virtuelle Maschine" deshalb, weil jeder Task der gesamte Registersatz ohne Einschränkungen zur Verfügung stehen soll. Die Aufteilung des SRAM muß allerdings zu Fuß erfolgen, da der AVR keine Adreßumsetzungsvorkehrungen hat.

Bereiche im SRAM:

a) Taskzustandsbereich. Besteht aus einem Zeiger auf die aktuelle Task und einem Speicherbereich je Task, der Zustandsbits sowie den aktuellen Stackpointer enthält. Der Stackpointer ist dann gültig, wenn die Task keine Laufzeit hat. Er zeigt auf das erste freie Byte im Stack (über dem geretteten Taskzustand).
b) Stackbereiche. Je Task ist ein Bereich vorgesehen. Größe: beispielsweise 90 Bytes. Der gerettete Taskzustand umfaßt 36 Bytes:
   - den Befehlszähler,
   - die 32 Register,
   - das Statusregister,
   - ein Zustandsbyte (reserviert).

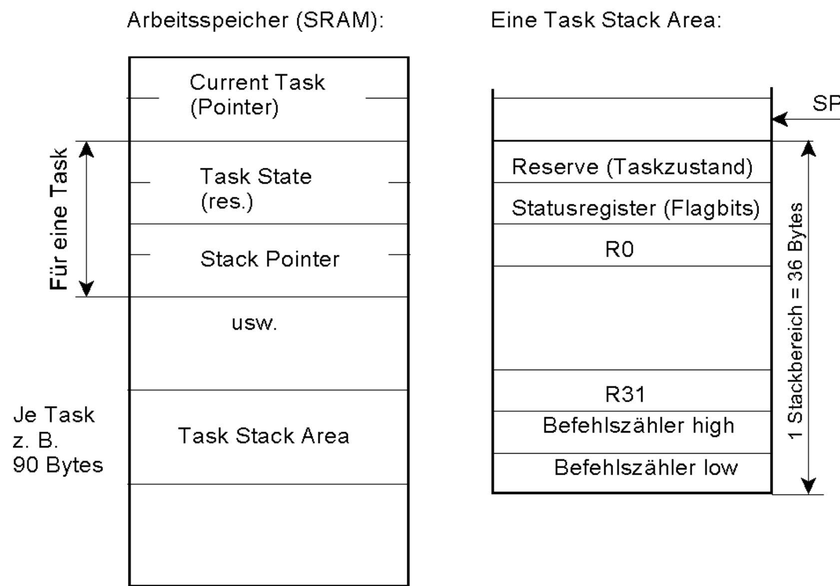
Taskumschaltung:

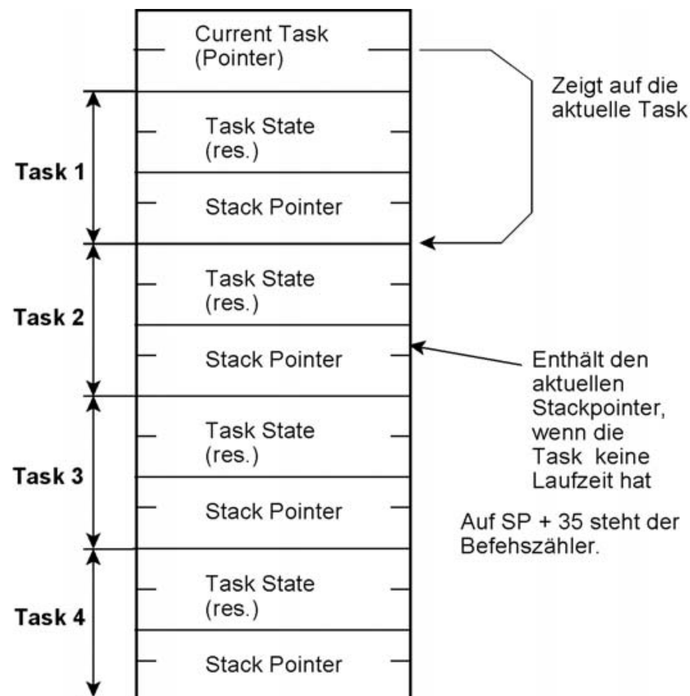
1. Eintritt mit Interrupt oder Call rettet den Befehlszähler auf den Stack.
2. Dann kommen alle Register einschließlich Flags auf den Stack.
3. Der Stackpointer wird in den Taskzustandsbereich gerettet.
4. Der nächste Stackpointer wird geholt und eingerichtet.
5. Alle Register (einschließlich Flags) werden aus dem Stack geholt.
6. Umschaltung erfolgt durch Laden des Befehlszählers mittels RET oder RETI.

Initialisierung:

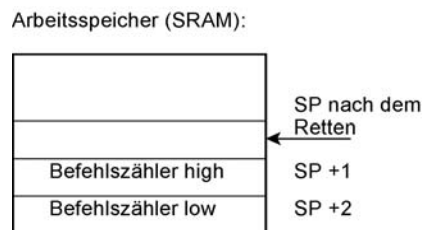
- Für jede Task wird der erste Stackbereich eingerichtet.
- Alle Befehlszähler in den Stacks zeigen auf ein lauffähiges Programm (und wenn es ein Sprung auf sich selbst ist).
- Dann wird der Timer-Interrupt scharfgemacht und darauf gewartet, daß etwas losgeht.

Alle Tasks werden angeworfen. Jede Task muß eine endlose äußerste Schleife enthalten (also ständig umlaufen – und ggf. abfragen, was zu tun ist).





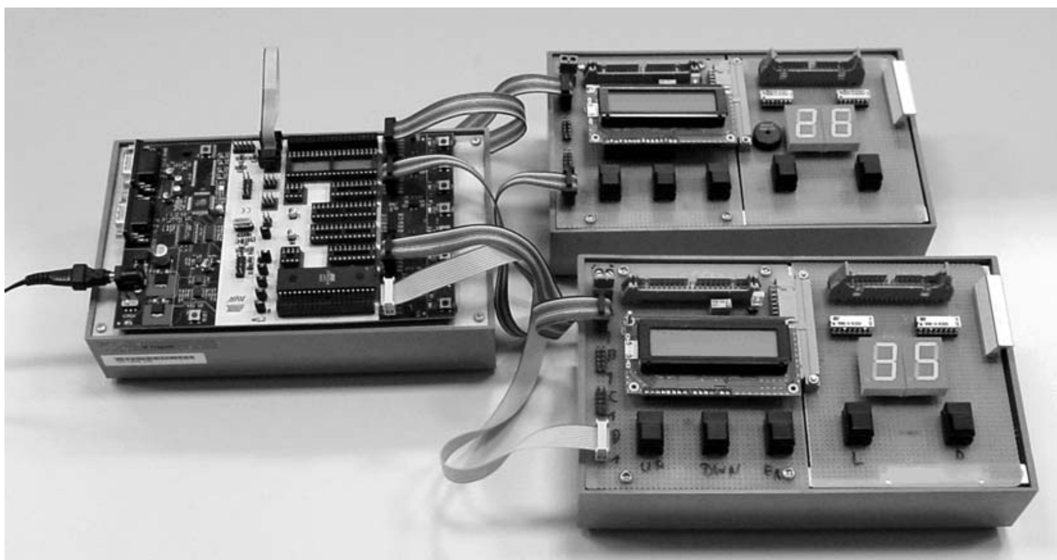
Der gerettete Befehlszähler im Stack:



Achtung – oben das High-Byte, unten das Low-Byte, obwohl es sich eigentlich um eine Little-Endian-Maschine handeln sollte. Gurkensache ...

Offset v. SP	Inhalt	Offset v. SP	Inhalt	Offset v. SP	Inhalt	Offset v. SP	Inhalt
1	Zustand	10	R7	19	R16	28	R25
2	SReg	11	R8	20	R17	29	R26
3	R0	12	R9	21	R18	30	R27
4	R1	13	R10	22	R19	31	R28
5	R2	14	R11	23	R20	32	R29
6	R3	15	R12	24	R21	33	R30
7	R4	16	R13	25	R22	34	R31
8	R5	17	R14	26	R23	35	PC high
9	R6	18	R15	27	R24	36	PC low

Versuchsanordnung:
Atmel Starterkit STK 500 mit ATmega16. Taktfrequenz: 4 MHz. Versuchsperipherie: zwei Übungsplattformen UeIDE 04 mit angesteckten Siebensegmentanzeigen UeSSTa 04a. Die Anordnung belegt vier Ports. Jede Siebensegmentanzeige wird – zusammen mit der jeweils zugeordneten Taste – als elektronischer Würfel betrieben (umlaufende Zählung von Eins bis Sechs, solange die Taste betätigt wird).



Die vier Tasks (= virtuellen Maschinen) werden zu Fuß eingerichtet und hart deklariert.

```
.DSEG
.ORG 0x0060

current_task: .byte 2
first_task:   .byte 4
sec_task:     .byte 4
third_task:   .byte 4
last_task:    .byte 4


.ORG ramend-(4*90)
task_1:       .byte 90
task_2:       .byte 90
task_3:       .byte 90
task_4:       .byte 90
```

Der Zeit-Interrupt wird vom Counter/Timer 1 ausgelöst.

Die eigentliche Taskumschaltung

Wir nehmen den Stack zur Adressierung, um kein Universalregister belegen zu müssen.

```
;  Interrupt / System Call
brk:
cli

timeslice:
; sbi porta,0  ; Messimpuls ein - dient zur Zeitmessung mittels Oszilloskop
push r31
push r30
push r29
push r28
push r27
push r26
push r25
push r24
push r23
push r22
push r21
push r20
push r19
push r18
push r17
push r16
push r15
push r14
push r13
push r12
push r11
push r10
push r9
push r8
push r7
push r6
push r5
push r4
push r3
push r2
push r1
```

```
push r0
in temp,sreg
push temp
ldi temp,0 ; Reserve. Taskzustandsbyte
push temp  ; alles im Stack

; **************** Alte Task gerettet ************

lds yl,current_task
lds yh,current_task+1

in r24,spl                      ; Stackpointer holen
in r25,sph

std y+2,r24                     ; Stackpointer abspeichern
std y+3,r25

cpi yl,low(last_task)
brne nexttask
cpi yh, high(last_task)
brne nexttask

firsttask:
ldi yl,low(first_task-4)
ldi yh,high(first_task-4)

nexttask:
adiw y,4

sts current_task,yl
sts current_task+1,yh ; ----------- neue Task eingestellt

bkout:                          ; Die ausgewählte Task erhält Laufzeit
ldd r24,y+2                     ; Stackpointer einrichten
ldd r25,y+3

out spl,r24
out sph,r25

; --------------------------------------------------
pop temp                        ; Reserve. Taskzustandsbyte
pop temp
andi temp,0x7f                  ; Interrupt Enable Flag aus
out sreg,temp                   ; Flagbits einstellen
pop r0
pop r1
pop r2
pop r3
pop r4
pop r5
pop r6
pop r7
pop r8
pop r9
pop r10
pop r11
pop r12
pop r13
pop r14
pop r15
pop r16
pop r17
pop r18
```

```
pop r19
pop r20
pop r21
pop r22
pop r23
pop r24
pop r25
pop r26
pop r27
pop r28
pop r29
pop r30
pop r31
; cbi porta,0                     ; Messimpuls aus
reti                              ; nächste Task erhält Laufzeit
```

Das ist alles ...

Initialisierung der Tasks

Damitdas eigentliche Anwendungsprogramm (elektronischer Würfel) reentrant sein kann, brauchen wir eine virtuelle E-A-Adressierung. Hierzu nutzen wir die Tatsache, daß die E-A-Ports auch über den Speicheradreßraum zugänglich sind. Jeder Würfel braucht einen Port. Dementsprechend wird für jede Task eine virtuelle Portadresse im Taskzustandsbereich hinterlegt.

Der gesamte Rettungsbereich umfaßt 36 Bytes. Der Stackpointer zeigt auf den nächstniedrigeren Adreßwert (TOS – 1). Ganz am End jedes Stackbereichs wird ein Byte frei gelassen (Schutzstelle). Endadresse des Stackbereichs = Anfangsadresse + 89. Deshalb muß der Stackpointer auf Anfangsadresse + 88 – 36 zeigen. Das Low-Byte des Befehlszählers kommt auf Anfangsadresse + 88, das High-Byte auf Anfangsadresse + 87.

```
ldi r24,low(task_1+88-36)         ; Stackpointer *********** TASK 1
ldi r25,high(task_1+88-36)
sts first_task+2,r24
sts first_task+3,r25

ldi r24,0x39                      ; Virtual Port Adrs
sts first_task,r24

ldi r24,low(init_1)               ; Befehlszähler
ldi r25,high(init_1)
sts task_1+87,r25
sts task_1+88,r24


ldi r24,low(task_2+88-36)         ; Stackpointer ************ TASK 2
ldi r25,high(task_2+88-36)
sts sec_task+2,r24
sts sec_task+3,r25

ldi r24,0x36                      ; Virtual Port Adrs
sts sec_task,r24

ldi r24,low(init_2)               ; Befehlszähler
ldi r25,high(init_2)
sts task_2+87,r25
sts task_2+88,r24
```

```
ldi r24,low(task_3+88-36)           ; Stackpointer ************* TASK 3
ldi r25,high(task_3+88-36)
sts third_task+2,r24
sts third_task+3,r25

ldi r24,0x33                        ; Virtual Port Adrs
sts third_task,r24

ldi r24,low(init_3)                 ; Befehlszähler
ldi r25,high(init_3)
sts task_3+87,r25
sts task_3+88,r24


ldi r24,low(task_4+88-36)           ; Stackpointer ************** TASK 4
ldi r25,high(task_4+88-36)
sts last_task+2,r24
sts last_task+3,r25

ldi r24,0x30                        ; Virtual Port Adrs
sts last_task,r24

ldi r24,low(init_4)                 ; Befehlszähler
ldi r25,high(init_4)
sts task_4+87,r25
sts task_4+88,r24

; ************ Alle Taskzustandsbereiche sind initialisiert;
; ************ in allen Stackbereichen steht ein Befehlszähler.

; ************ Es soll mit Task 1 losgehen.
; ************ Hierzu nutzen wir die Rückkehrfunktion aus.

ldi r24,low(first_task)             ; Auf erste Task einstellen
ldi r25, high(first_task)
sts current_task,r24
sts current_task+1,r25

lds yl,current_task                 ; Vorbereitung für den Eintritt
lds yh,current_task+1               ; in den Rückkehrablauf

; ************ Jetzt wird nur noch der Systemzeitgeber CTC 1 gestartet

ldi temp, high (times)
out ocr1ah,temp
ldi temp, low(times)
out ocr1al, temp
ldi temp,0x09                       ; CTC 1 auf Normal Mode; 1:1 Takt
out tccr1b,temp                     ; Clear on Compare Match A
ldi temp,0x10
out timsk,temp                      ; Compare Match A Interrupt

rjmp bkout                          ; Mit der ausgewählten Task starten

; ************* Interrupt wird scharf *************

iwait:                              ; Ewige Warteschleife
rjmp iwait                          ; als Abschluss der Initalisierung

; **************************************************
```

Der Start der Anwendungen in den Tasks

Jede Task hat eine Einsprungstelle. Von dort aus kann zu einer beliebigen Anwendung verzweigt werden. Hier ist es immer dieselbe.

```
init_1:
jmp wue

init_2:
jmp wue

init_3:
jmp wue

init_4:
jmp wue
```

Das Anwendungsprogramm der Versuchsanordnung

```
wue:

lds yl,current_task
lds yh,current_task+1
ldd zl,y+0                          ; Virtuelle Portadresse
ldi zh,0
ldi einer,1

wdisplay21:
mov temp,einer
rcall convseg                       ; Siebensegmentanzeige
std z+2,r0

wwarten2:                           ; Tastenabfrage
ldd r16,z+0
andi r16,0x80
brne wwarten2

inc einer
cpi einer,7
brne wdisplay21
rjmp wue


; *******************  Unterprogramm binär zu Siebensegment

convseg:
push zl
push zh                             ;Binärzahl in temp
ldi zl,low(segtab*2)
ldi zh, high(segtab*2)
add zl,temp
ldi temp,0
adc zh,temp
lpm
pop zh
pop zl
ret

segtab:
.db 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90
```

Zeitangaben

Eine Taskumschaltung dauert – bei 4 MHz Takt – ca. 50 µs.

Die Dauer einer Zeitscheibe ergibt sich durch das Setzen des Vergleichsregisters im Zähler/Zeitgeber 1. Laufzeit der Task = Dauer der Zeitscheibe – Dauer der Taskumschaltung.

```
.equ times = 3000                    ; Einstellungsbeispiel
```

Weitere Beispiele (Näherungswerte):

Takte je Zeitscheibe	Zeitscheibe insgesamt	Programmlaufzeit in der Zeitscheibe	Verhältnis Umschaltzeit zu Programmlaufzeit	Overhead in Bezug auf gesamte Zeitscheibe
500	125 µs	75 µs	1 : 1,5	40 %
2 000	500 µs	450 µs	1 : 9	10 %
3 000	750 µs	700 µs	1 : 14	6,7 %
4 000	1 ms	950 µs	1:19	5 %
20 000	5 ms	4,95 ms	1:99	1 %

Von 1 ms an aufwärts wird die Dauer der Taskumschaltung vernachlässigbar. Zeitscheiben zwischen 1 und 2 ms ermöglichen es, vier bis acht Tasks während einer Netzhalbwelle (8 ms (60- Hz-Netz)) laufen zu lassen.