

AVR STUDIO

1 Inhalt

2	Installation.....	2
2.1	Quelle	2
2.2	Starten von Atmel Studio	3
3	Programmierung	4
3.1	ASSEMBLER.....	4
3.1.1	LED EIN UND AUSSCHALTEN	4
3.1.2	BCD-7-SEGMENT Codierung Version1	10
3.1.3	BCD-7-SEGMENT Codierung Version2	15
3.1.4	Multiplexen von zwei 7 Segment Anzeigen Version 3	21
3.1.5	Funktion.....	21
3.2	Interrupt Programmierung	23
3.2.2	BCD-7-SEGMENT Codierung Version4	25
3.3	Programmierung in C / C++	34
3.4	Atmega644 Datasheet	35
3.4.1	Components of the ATMEGA644	35

2 Installation

Siehe auch

http://www.atmel.com/products/microcontrollers/avr/start_now.aspx

2.1 Quelle

<http://www.microchip.com/avr-support/atmel-studio-7>

Windows (x86/x64)

Atmel Studio 7.0 (build 1645) web installer (recommended) -

This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.35.0 and Atmel Toolchains. It is recommended to use this installer if you have internet access while installing.

October
2017

2.4 MB



Atmel Studio 7.0 (build 1645) offline installer -

This installer contains Atmel Studio 7.0 with Atmel Software Framework 3.35.0 and Atmel Toolchains. Use this installer if you do not have internet access while installing.

October
2017

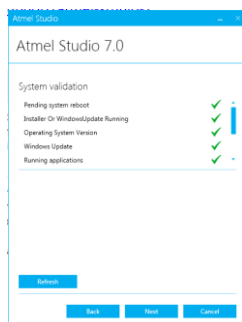
887 MB



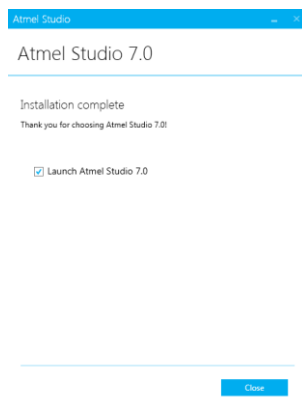
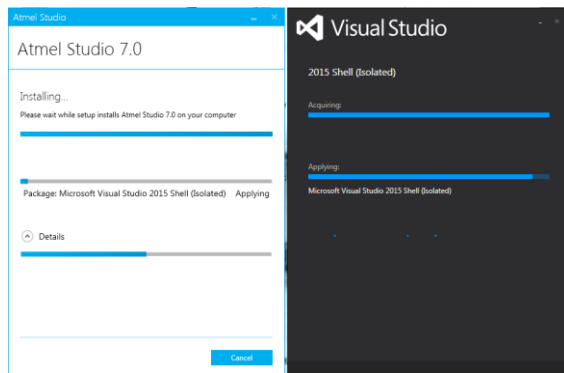
SHA1: b7da45b0a5c594d7298640f4237eb1d84a3a5834

Version number: 7.0.1645

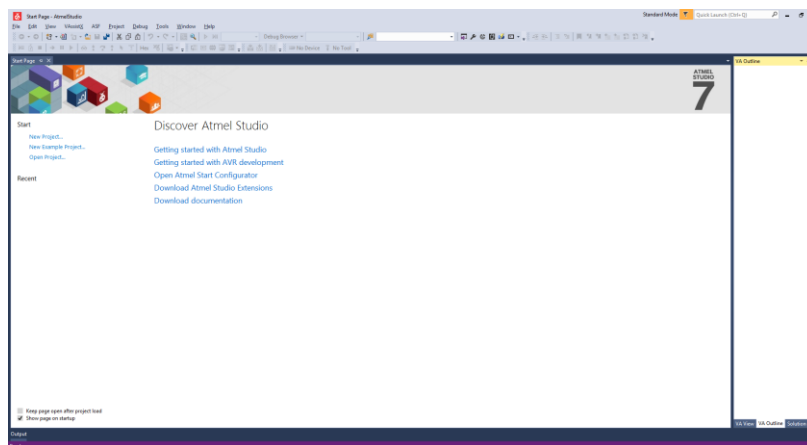
Nach dem starten von <as-installer-7.0.1645-full.exe> und selektieren der AVR Familien erscheint



Nach klicken von next wird Visual Studio C++ 2013 installiert.



2.2 Starten von Atmel Studio



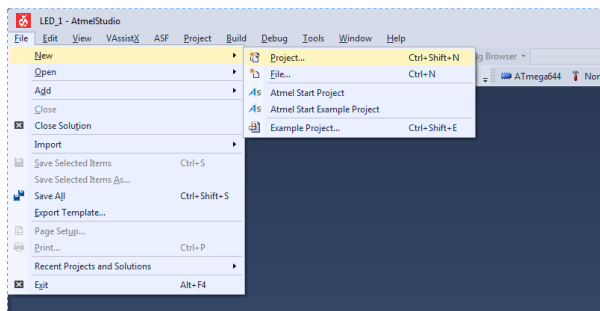
3 Programmierung

3.1 ASSEMBLER

3.1.1 LED EIN UND AUSSCHALTEN

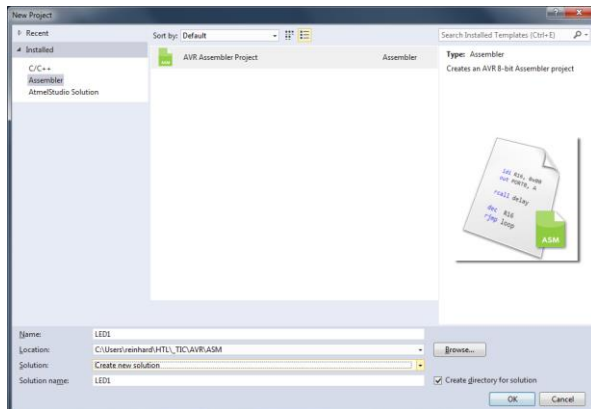
Ein einfaches Programm zum Ein und Ausschalten einer LED. Das Programm kann für eine 7-Segmentanzeige erweitert werden.

1. Neues Projekt erstellen



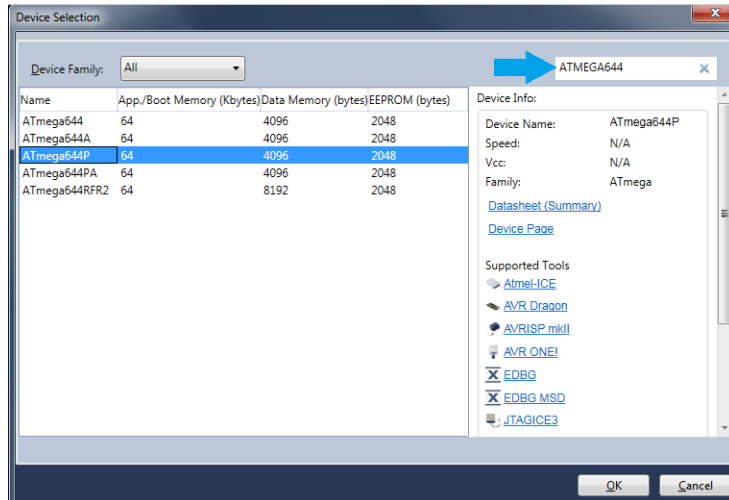
2. Assembler als Sprache auswählen

Location und Name des Projekts festlegen und mit OK bestätigen.

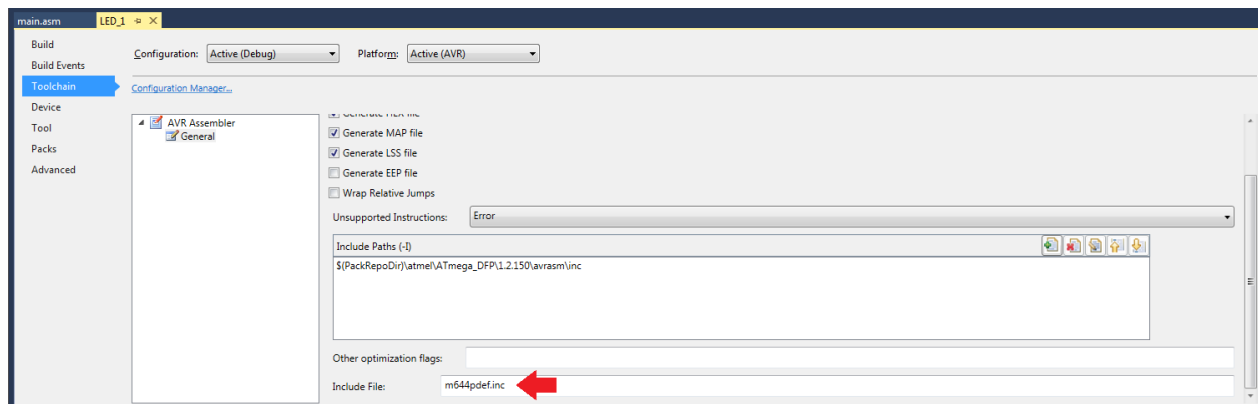


3. Baustein aus der AVR Familie auswählen

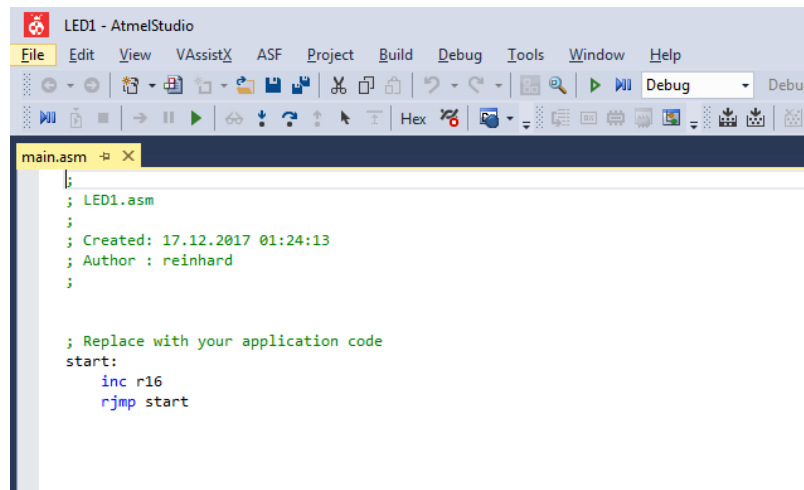
Nach dem Drücken der OK taste wird man in einem Popup aufgefordert den Baustein zu selektieren. Wir verwenden den Atmega644P vom Crump-Modul.



Das entsprechende include file für den Baustein Atmega644P ist in den **toolchain** eingetragen.



Nach der Bestätigung mit OK wird ein ASM Dummy Programm mit einer Endlosschleife erstellt, welches auch automatisch angezeigt wird.



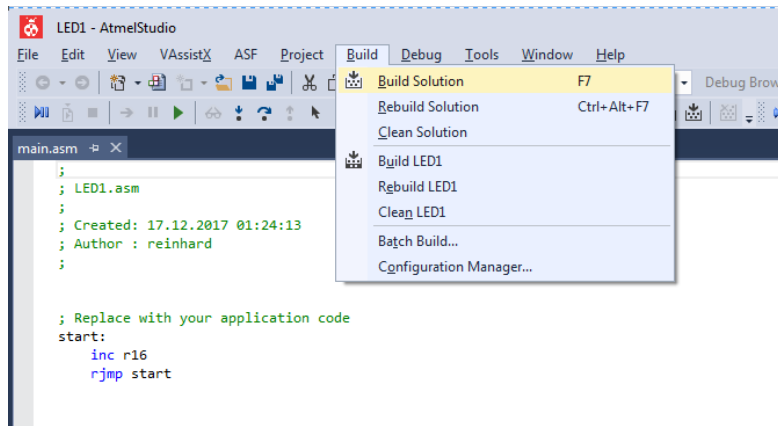
The screenshot shows the AVR Studio IDE interface. The title bar reads "LED1 - AtmelStudio". The menu bar includes File, Edit, View, VAssistX, ASF, Project, Build, Debug, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging. The main editor window displays the file "main.asm" with the following assembly code:

```
;
; LED1.asm
;
; Created: 17.12.2017 01:24:13
; Author : reinhard
;

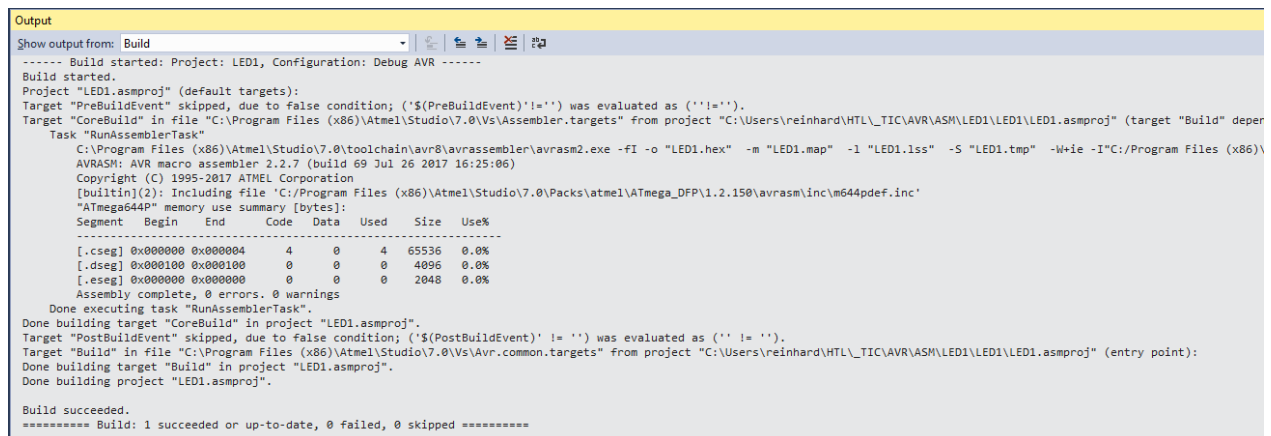
; Replace with your application code
start:
    inc r16
    rjmp start
```

4. Kompilieren

Mit Hilfe von Build / Build Solution kann das Projekt kompiliert werden.

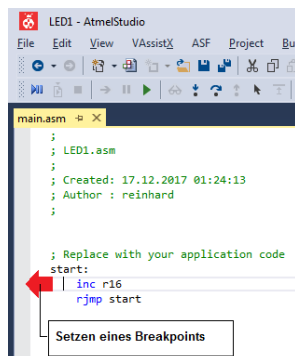


Compile- Report

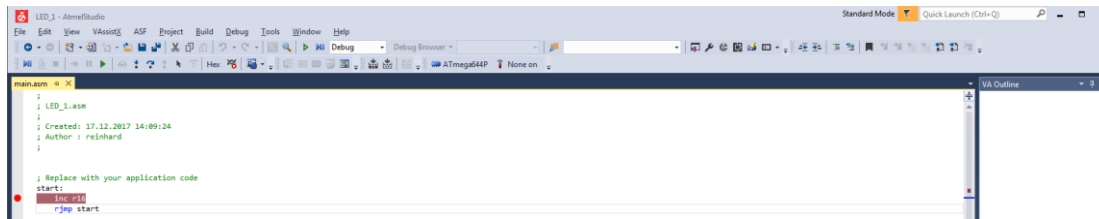


5. Debugging

Im linken Rand kann ein Breakpoint (durch Drücken der linken Maustaste) gesetzt werden.



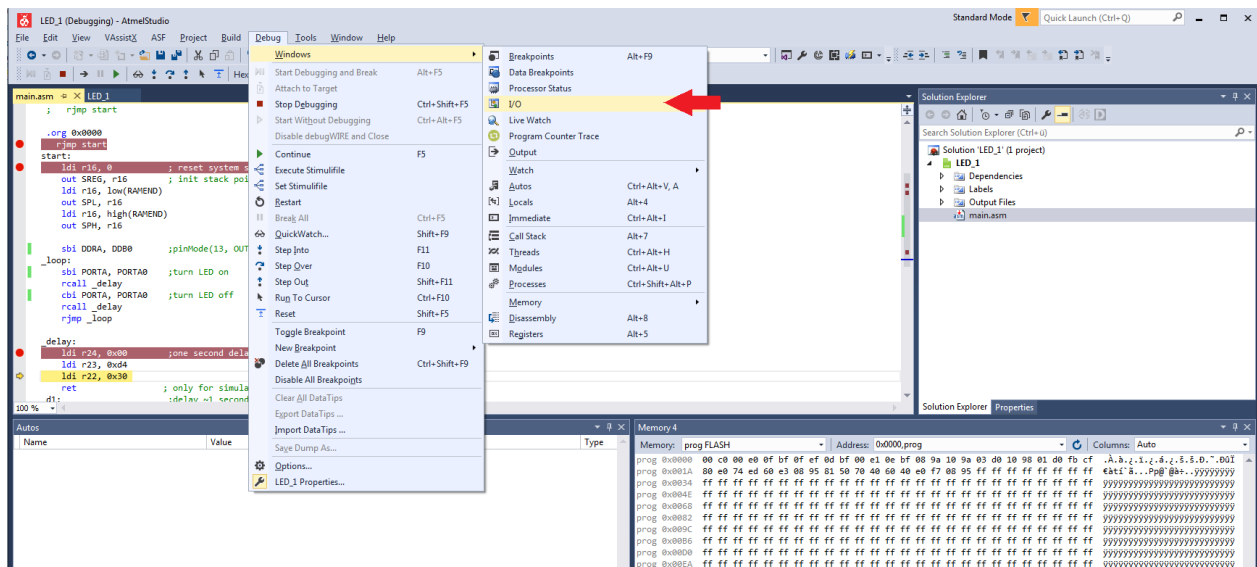
Nach dem setzen des Breakpoints erscheint ein roter Punkt im linken Rand des Eingabefensters.



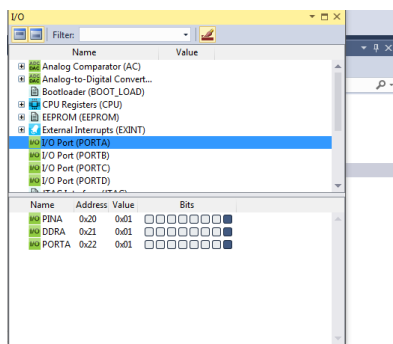
6. Anzeige Port Status

Um den Port Status während der Simulation anzuzeigen selektieren Sie

Debug/Windows/I/O

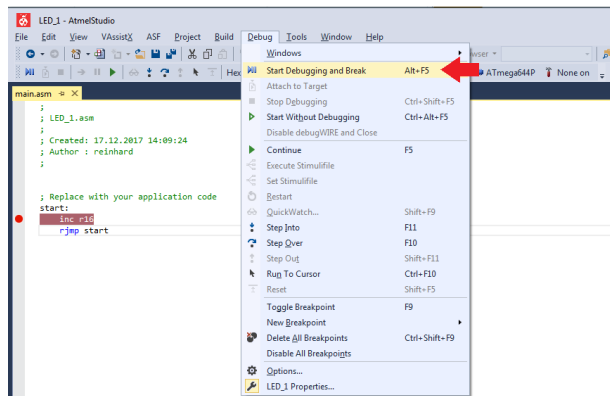


Sie erhalten daraufhin folgendes Fenster



Hier können Ports / Pins angezeigt bzw. Lese-Pins über die Maus gesetzt werden.

7. Start Debugging



8. ASM-Code für Blink-LED

```

; start:
;   inc r16
;   rjmp start

.org 0x0000
rjmp start

start:
    ldi r16, 0           ; reset system status
    out SREG, r16        ; init stack pointer
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16

    sbi DDRA, DDB0       ;pinMode(13, OUTPUT);

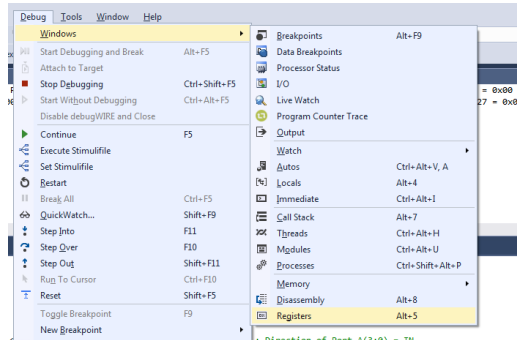
_loop:
    sbi PORTA, PORTA0    ;turn LED on
    rcall _delay
    cbi PORTA, PORTA0    ;turn LED off
    rcall _delay
    rjmp _loop

_delay:
    ldi r24, 0x00        ;one second delay iteration
    ldi r23, 0xd4
    ldi r22, 0x30
    ; ret                ; only for simulation
_d1:
    subi r24, 1
    sbci r23, 0
    sbci r22, 0
    brcc _d1
    ret

```

3.1.2 BCD-7-SEGMENT Codierung Version1

Für Debug Zwecke blenden wir ein Fenster der Registerinhalte ein.



3.1.2.1 Lesen der unteren 4-Bit von Port A

Setzen des DIR Registers auf 0xF0. [W W W W **R R R R**]. Die unteren 4 Bit des A-Ports sind für Lese PINS konfiguriert.

```
; initialize PORT A(3:0) read
ldi acc, (1<<DDA7)|(1<<DDA6)|(1<<DDA5)|(1<<DDA4)
out DDRA, acc;
```

Diese 4 Pins werden mit den Zähler Ausgängen verbunden. Im Debugger können wir diese 4 Eingangspins per Mausclick setzen oder löschen. Wir setzen die PINS (A3, und A2 auf 1).

Name	Address	Value	Bits
I/O PINA	0x20	0x0C	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
I/O DDRA	0x21	0xF0	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
I/O PORTA	0x22	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Mit Hilfe von:

Loop:

```
in  ACC, PINA;
cbr ACC, (1<<7);
cbr ACC, (1<<6);
cbr ACC, (1<<5);
cbr ACC, (1<<4);
out PORTB, ACC;
```

```
nop ;
```

```
rjmp loop ; go back to loop
```

werden die entsprechenden Bits auf Port B ausgegeben.

Name	Address	Value	Bits
I/O PINB	0x23	0x03	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
I/O DDRB	0x24	0xFF	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
I/O PORTB	0x25	0x0C	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

3.1.2.2 Umcodierung auf die 7-Segment Ausgabe

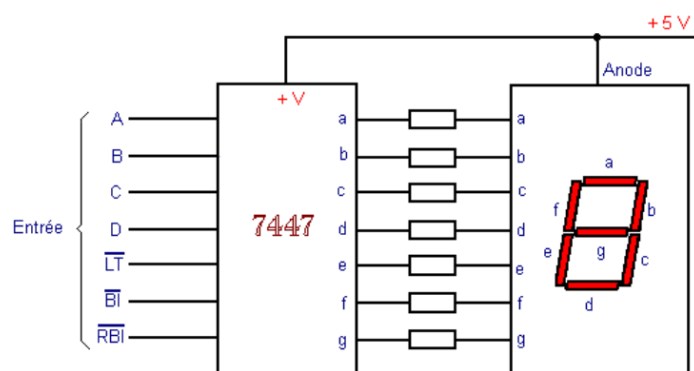
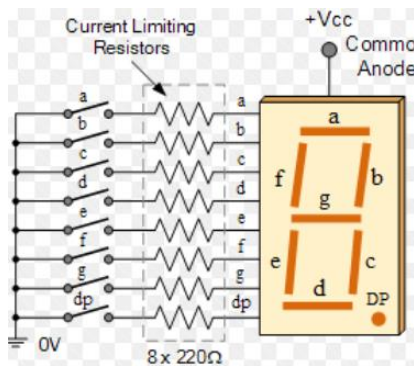


Fig. 37. - Afficheur 7 segments commandé par un décodeur 7447.

Eingangswert	7-Segment Output [0, g, f, e, d, c, b, a]	HEX OUTPUT
0	$\wedge(0, 0, 1, 1, 1, 1, 1, 1)$	0xC0 / 0x3F
1	$\wedge(0, 0, 0, 0, 0, 1, 1, 0)$	0xF9 / 0x06
2	$\wedge(0, 1, 0, 1, 1, 0, 1, 1)$	0xA4 / 0x5B
3	$\wedge(1, 0, 1, 1, 1, 1, 1, 1)$	0x40 / BF
	...	
F		

Registers: R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00 R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0x02 R17 = 0x02 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

```

main.asm
; P R O G R A M M L O O P
;
;
Loop:
    in  HELP, PINA;
    cbr HELP, (1<7);
    cbr HELP, (1<6);
    cbr HELP, (1<5);
    cbr HELP, (1<4);

    ;ldi HELP, 0x01
    ;sub ACC, HELP;

    ldi ACC, 0x01
    cp  HELP, ACC
    breq SEG01

    ldi ACC, 0x02
    cp  HELP, ACC
    breq SEG02

```

Hardware View: I/O Port (PORTA) is selected. Address: 0x20, Value: 0x02.

Registers: R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00 R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0x04 R17 = 0x02 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

```

main.asm
rjmp loop;

SEG01:
    ldi ACC, 0xF9
    out PORTB, ACC
    rjmp loop;

SEG02:
    ldi ACC, 0xA4
    out PORTB, ACC
    rjmp loop;

SEG03:
    ldi ACC, 0xB0
    out PORTB, ACC
    rjmp loop;

; out PORTB, ACC;

```

Hardware View: I/O Port (PORTB) is selected. Address: 0x21, Value: 0xF9.

Programm Listing

```

;
; BCD.asm
;
; Created: 07.01.2018 15:03:52
; Author : reinhard
;
; .INCLUDE "m644def.inc"

.DSEG

.EQU tab_size = 16
table: .BYTE tab_size ;

; REGISTER DEFINITION
.DEF ACC = R16 ; Multipurpose register
.DEF HELP = R17;

.CSEG
.ORG 0x00;
; .db 1,2,3,4,5,65,6,7

; -----
; -- INTERRUPT SERVICEROUTINE --
; -----

;reti ; Int vector 1
;reti ; Int vector 2
;reti ; Int vector 3
;reti ; Int vector 4
;reti ; Int vector 5
;reti ; Int vector 6
;reti ; Int vector 7
;reti ; Int vector 8

rjmp start

start:

;initialize stack
ldi r16, low(RAMEND)
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16

; initialize PORT A(3:0) read
ldi acc, (1<<DDA7)|(1<<DDA6)|(1<<DDA5)|(1<<DDA4); Direction of Port A(3:0) = IN
out DDRA, acc;
ldi acc, (1<<DDB7)|(1<<DDB6)|(1<<DDB5)|(1<<DDB4)|(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0); PORT B = OUT
out ddrb, acc;

; =====
; P R O G R A M L O O P
; =====
;
; Loop:
;     in  HELP, PINA;
;     cbr HELP, (1<<7);
;     cbr HELP, (1<<6);
;     cbr HELP, (1<<5);
;     cbr HELP, (1<<4);

;
;     ldi ACC, 0x00
;     cp  HELP, ACC
;     breq SEG00
;
;     ldi ACC, 0x01
;     cp  HELP, ACC
;     breq SEG01
;
;     ldi ACC, 0x02
;     cp  HELP, ACC
;     breq SEG02
;
;     ldi ACC, 0x03
;     cp  HELP, ACC
;     breq SEG03
;
;     ldi ACC, 0x04
;     cp  HELP, ACC
;     breq SEG04
;
;     ldi ACC, 0x05
;     cp  HELP, ACC
;     breq SEG05
;
; ...

```

```
SEG00:      ldi ACC, 0x3F
            out PORTB, ACC
            rjmp LOOP;

SEG01:      ldi ACC, 0x06
            out PORTB, ACC
            rjmp LOOP;

SEG02:      ldi ACC, 0x5B
            out PORTB, ACC
            rjmp LOOP;

SEG03:      ldi ACC, 0x4F
            out PORTB, ACC
            rjmp LOOP;

SEG04:      ldi ACC, 0x66
            out PORTB, ACC
            rjmp LOOP;

...

            rjmp LOOP;

SEG08:      ldi ACC, 0x7F
            out PORTB, ACC
            rjmp LOOP;

SEG09:      ldi ACC, 0x6E
            out PORTB, ACC
            rjmp LOOP;

            nop ;
rjmp loop ; go back to loop
;
; End of source code
;
```

3.1.3 BCD-7-SEGMENT Codierung Version2

3.1.3.1 Allgemeines

3.1.3.1.1 General Purpose Register

	7	0	Addr.	
General Purpose Working Registers	R0		0x00	
	R1		0x01	
	R2		0x02	
	...			
	R13		0x0D	
	R14		0x0E	
	R15		0x0F	
	R16		0x10	
	R17		0x11	
	...			
	R26		0x1A	X-register Low Byte
	R27		0x1B	X-register High Byte
	R28		0x1C	Y-register Low Byte
	R29		0x1D	Y-register High Byte
	R30		0x1E	Z-register Low Byte
	R31		0x1F	Z-register High Byte

Abbildung 1 General purpose register

3.1.3.1.2 Erstellen einer Code Tabelle

```
.CSEG
rjmp start;
.ORG 0x40; interrupt einspruenge uebergehen

BCD_7SEG_TABLE:
.DB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x27, 0x7F, 0x6E;
```

Abbildung 2 Code Tabelle an der Adresse 0x40

Beachte, die Tabelle ist auf 16 Bit integer organisiert, daher muss für byte Adressierung der Wert mit 2 multipliziert werden!

```
prog 0x0000 44 c0 ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0010 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0020 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0030 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0040 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0050 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0060 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0070 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0080 3f 06 5b 4f 66 6d 7d 27 7f 6e 0f ef 0d bf 00 e1
prog 0x0090 0e bf 00 ee 01 b9 0f ef 04 b9 0e 94 5f 00 80 2f
prog 0x00A0 90 e0 0e 94 54 00 f9 cf ef 93 ff 93 e0 e8 f0 e0
prog 0x00B0 e8 0f f9 1f 14 91 15 b9 ef 91 ff 91 08 95 00 b1
prog 0x00C0 0f 77 0f 7b 0f 7d 0f 7d 08 95 ff ff ff ff ff
prog 0x00D0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00E0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00F0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0100 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

3.1.3.1.3 Analyse des Hex Dumps

Der der Tabelle unmittelbar folgende Befehl ist der in **Gelb** markierte Befehl ldi (load immediate).

```
BCD_7SEG_TABLE:
.DB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x27, 0x7F, 0x6E;

start:

; initialize stack
ldi ACC1, LOW(RAMEND);
```

In der Dokumentation findet man folgende Übersetzung

ldi: 1110-KKKK-dddd-KKKK

Die RAMEND Adresse, für das indirekt adressierbare RAM liegt beim ATMEGA644 auf **0x10FF**. Das LOW Byte von RAMEND ist daher FF. Da die Register R16 bis R31 zugelassen sind, ist für R16 dddd=0000. Die Übersetzung des Befehls liefert daher:

1110 1111 0000 1111 -> ef 0f im FLASH Hex Dump sehen wir daher **0f ef**.

```
out SPL, ACC1;
```

Übersetzt ist der Befehl

1011-1AAD-DDDD-AAAA

SPL ist ein SPECIAL FUNCTION REGISTER (Stackpointer LOW).

Adresse des Registers R16 = 10H

```
116 .equ    SPL = 0x3d
117 .equ    SPH = 0x3e
```

OU | **T111-0000 - 1101**

1011-1AAD-DDDD-AAAA → **DDDD-AAAA-1011-1AAD** → **0000-1101-1011-1111** → **0d bf**

Funktion

Eingabe der Zahl 7:

Name	Address	Value	Bits
IO PINA	0x20	0x07	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
IO DDRA	0x21	0xE0	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Wir überprüfen die Register und sehen dass



IN ACC1, PINA

Den Wert 7 in das Register 16 (Adresse 0x10) überträgt

Memory:	data REGISTERS
data 0x0000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0010	07 4f 00 00 00 00 00 00 03 00 00 00 00 00 00 00
data 0x0020	07 e0 00 4f ff 4f 00 00 00 00 00 00 00 00 00 00
data 0x0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Registers

R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 =
 R15 = 0x00 R16 = 0x07 R17 = 0x27 R18 = 0x00 R19
 R30 = 0x87 R31 = 0x00

I/O DDRB 0x24 0xFF 
 I/O PORTB 0x25 0x27 

3.1.3.1.3.1 Stackpointer

```
; initialize stack
```

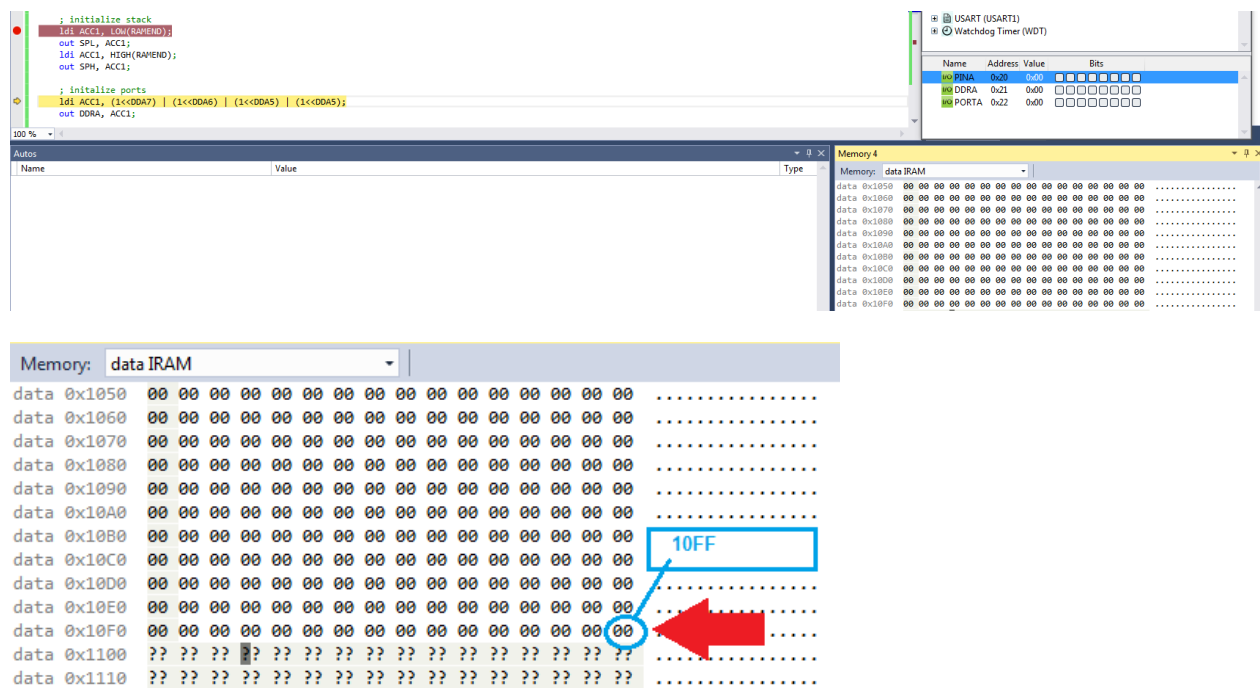
```
ldi ACC1, LOW(RAMEND);
```

```
out SPL, ACC1; RAMEND 0x000010ff
```

```
ldi ACC1, HIGH(RAMEND);
```

```
out SPH, ACC1;
```

Der Stackpointer verweist auf das Ende des indirekt adressierbaren RAM's (0x10FF).



The screenshot displays the AVR Studio interface. The code window shows the assembly code for initializing the stack pointer. The I/O window shows the status of DDRB and PORTB. The Memory window shows the RAM memory map, with a red arrow pointing to the 0x10FF address.

Code window:

```
; initialize stack
ldi ACC1, LOW(RAMEND);
out SPL, ACC1;
ldi ACC1, HIGH(RAMEND);
out SPH, ACC1;

; initialize ports
ldi ACC1, (1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4);
out DDRA, ACC1;
```

I/O window:

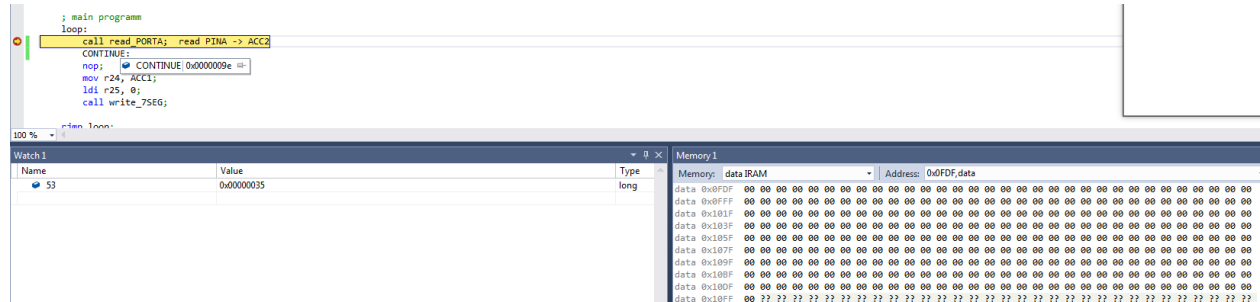
Name	Address	Value	Bits
DDRA	0x20	0x00	00000000
DDRB	0x21	0xFF	11111111
PORTA	0x22	0x00	00000000
PORTB	0x23	0x27	00100111

Memory window:

Address	Value
0x1050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1100	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0x1110	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

- Unterprogramm Aufruf <read_PORTA>

Vor der dem Funktionsaufruf <read_PORTA> ist das IRAM mit 0-en gefüllt. Wir sehen, dass die **Byte-Adresse** des dem Label <CONTINUE> folgenden Befehls (also der Befehl nop)



0x0000009E (entspricht der Word-Adresse **0x00004F**) auf den Stack geschrieben werden muss.

Abbildung 3 STACK (IRAM) vor dem Aufruf <powerDownMode>

Im Unterprogramm <read_PORTA> sehen wir daher die Rücksprungadresse auf dem Stack (IRAM) (0x0000004F).

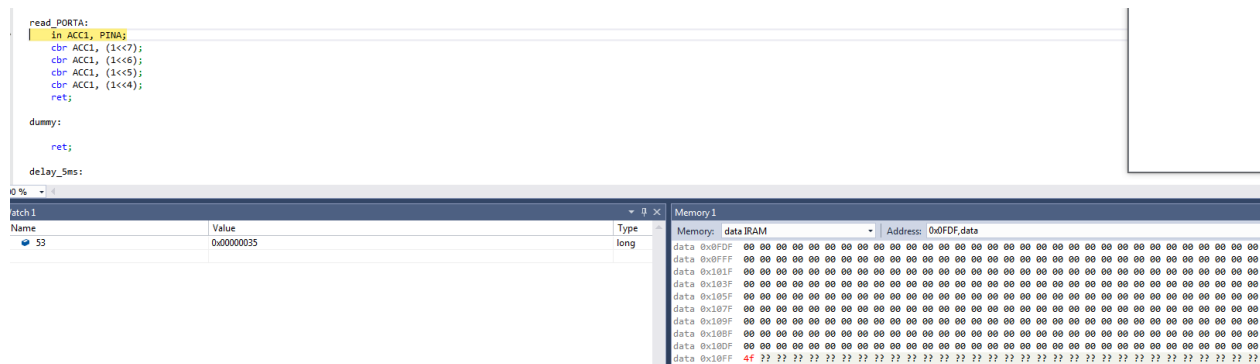


Abbildung 4 STACK (IRAM) nach dem Aufruf <powerDownMode>

Der Programmcode:

```

;
; BCD_7SEG.asm
.DEF ACC1 = R16;
.DEF ACC2 = R17;
.EQU takt = 16000000;

.CSEG
rjmp start;
.ORG 0x40; interrupt einspruenge übergehen

BCD_7SEG_TABLE:
.DB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x27, 0x7F, 0x6E;
.DB 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x02, 0x82, 0xD8, 0x80, 0x91;

start:
; initialize stack
ldi ACC1, LOW(RAMEND);
out SPL, ACC1;
ldi ACC1, HIGH(RAMEND);
out SPH, ACC1;

; initalize ports
ldi ACC1, (1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4);
out DDRA, ACC1;
ldi ACC1, 0xFF;
out DDRB, ACC1;

; main programm
loop:
    call read_PORTA; read PINA -> ACC2
    mov r24, ACC1;
    ldi r25, 0;
    call write_7SEG;
    rjmp loop;

write_7SEG:
    push ZL;
    push ZH;
    ldi ZL, LOW(BCD_7SEG_TABLE*2);
    ldi ZH, HIGH(BCD_7SEG_TABLE*2);
    add ZL, r24;
    adc ZH, r25;
    lpm ACC2, Z;
    out PORTB, ACC2;
    pop ZH;
    pop ZL;
    ret;

read_PORTA:
    in ACC1, PINA;
    cbr ACC1, (1<<7);
    cbr ACC1, (1<<6);
    cbr ACC1, (1<<5);
    cbr ACC1, (1<<4);
    ret;

```

3.1.4 Multiplexen von zwei 7 Segment Anzeigen Version 3

3.1.5 Funktion

Funktion

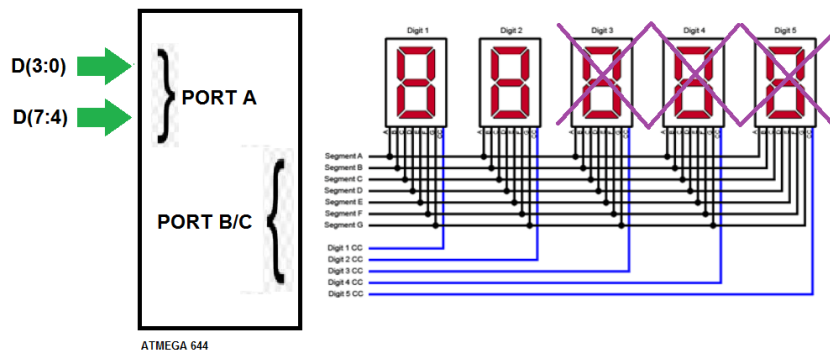
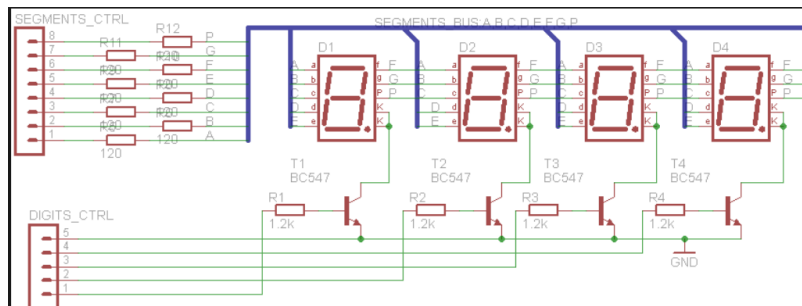


Abbildung 5 Prinzipschaltung für die Programmierung

Schaltung



Die Schaltfrequenz (Multiplexfrequenz) soll einstellbar sein.

Wir schalten den Anzeige-Token DIGIT-CTRL nach einem Delay von ca. 100 – 200 ms zum nächsten weiter. Schaltungstyp Brake before Make, wir schalten die Digits aus reichen den Token weiter und schalten anschließend die Digits der nächsten Stelle wieder ein.

Delay im AVR Assembler:

```
Delay_15mS:                ; For CLK(CPU) = 8 MHz
    LDI    dly1,    120    ; One clock cycle;
Delay1:
    LDI    dly2,    250    ; One clock cycle
Delay2:
    DEC    dly2            ; One clock cycle
    NOP                                ; One clock cycle
    BRNE   Delay2          ; Two clock cycles for true 1 clock for false

    DEC    dly1            ; One clock Cycle
    BRNE   Delay1          ; Two clock cycles for true 1 clock for false
RET
```

3.2 Interrupt Programmierung

3.2.1.1.1 Interrupt Vektoren

ATmega328P Interrupt Vector Table

Vector No	Program Address	Source	Interrupt Definition	Arduino/C++ ISR() Macro Vector Name
1	0x0000	RESET	Reset	
2	0x0002	INT0	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	0x0004	INT1	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	0x0006	PCINT0	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
5	0x0008	PCINT1	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
6	0x000A	PCINT2	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)
7	0x000C	WDT	Watchdog Time-out Interrupt	(WDT_vect)
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A	(TIMER0_COMPA_vect)
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B	(TIMER0_COMPB_vect)
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow	(TIMER0_OVF_vect)
18	0x0022	SPI, STC	SPI Serial Transfer Complete	(SPI_STC_vect)
19	0x0024	USART, RX	USART Rx Complete	(USART_RX_vect)
20	0x0026	USART, UDRE	USART, Data Register Empty	(USART_UDRE_vect)
21	0x0028	USART, TX	USART, Tx Complete	(USART_TX_vect)
22	0x002A	ADC	ADC Conversion Complete	(ADC_vect)
23	0x002C	EE READY	EEPROM Ready	(EE_READY_vect)
24	0x002E	ANALOG COMP	Analog Comparator	(ANALOG_COMP_vect)
25	0x0030	TWI	2-wire Serial Interface (I2C)	(TWI_vect)
26	0x0032	SPM READY	Store Program Memory Ready	(SPM_READY_vect)

Table 10-1. Reset and Interrupt Vectors

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	PCINT0	Pin Change Interrupt Request 0
6	\$000A	PCINT1	Pin Change Interrupt Request 1
7	\$000C	PCINT2	Pin Change Interrupt Request 2
8	\$000E	PCINT3	Pin Change Interrupt Request 3
9	\$0010	WDT	Watchdog Time-out Interrupt
10	\$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	\$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	\$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	\$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	\$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	\$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	\$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	\$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	\$0022	TIMER0_COMPB	Timer/Counter0 Compare match B
19	\$0024	TIMER0_OVF	Timer/Counter0 Overflow
20	\$0026	SPI_STC	SPI Serial Transfer Complete
21	\$0028	USART0_RX	USART0 Rx Complete
22	\$002A	USART0_UDRE	USART0 Data Register Empty
23	\$002C	USART0_TX	USART0 Tx Complete
24	\$002E	ANALOG_COMP	Analog Comparator
25	\$0030	ADC	ADC Conversion Complete
26	\$0032	EE_READY	EEPROM Ready
27	\$0034	TWI	2-wire Serial Interface
28	\$0036	SPM_READY	Store Program Memory Ready

Abbildung 6 Interrupt-Tabelle des ATMEGA644P

3.2.2 BCD-7-SEGMENT Codierung Version4

3.2.2.1 Allgemeines

In dieser Version wollen wir nur dann die Konvertierung durchführen, wenn sich das externe Signal (INT0) von L nach H ändert.

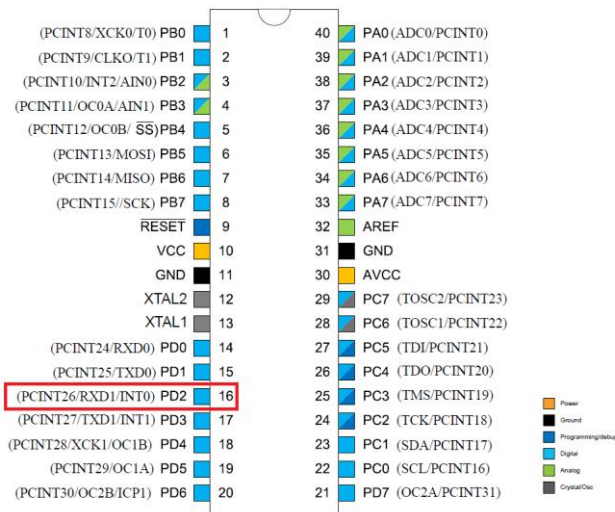


Abbildung 7 Pinbelegung des Prozessors

Dies könnte z.B. die Taktleitung des Zählers sein. Die restliche Zeit ist unser Programm im Power Down Modus um Strom zu sparen.

Wir verbinden daher die Taktleitung des Zählers mit der externen Interrupt Leitung (INT0 = PD2) siehe Pinbelegung.

Im Definitionsfile des Prozessors <m644pdef.pas> sind die Einsprung-Adressen eingetragen. Nach dem Auslösen des Interrupt 0 wird das Programm an der Adresse 2 fortgesetzt, oder in anderen Worten, der Programmcounter des Prozessors wird auf die Adresse 2 gesetzt.

Beachte:

Bevor der Programmcounter verändert wird, wird die Adresse des nächsten Befehls, welcher ohne Unterbrechung ausgeführt werden würde, auf dem Stapel gesichert. Die Adresse des Stapels findet der Prozessor im Stack Pointer.

3.2.2.2 Initialisierung der Interrupt Einsprung-Tabelle

m644pdef.pas

```

1108
1109 ; ***** INTERRUPT VECTORS *****
1110 .equ INT0addr = 0x0002 ; External Interrupt Request 0
1111 .equ INT1addr = 0x0004 ; External Interrupt Request 1
1112 .equ INT2addr = 0x0006 ; External Interrupt Request 2
1113 .equ PCI0addr = 0x0008 ; Pin Change Interrupt Request 0
1114 .equ PCI1addr = 0x000a ; Pin Change Interrupt Request 1
1115 .equ PCI2addr = 0x000c ; Pin Change Interrupt Request 2
1116 .equ PCI3addr = 0x000e ; Pin Change Interrupt Request 3

```

Abbildung 8 Ausschnitt der Interrupt Einsprungtabelle

Die Interrupt-Tabelle wird mit Hilfe der .ORG Direktive beschrieben.

```

; Einsprung externer Interrupt Routine
.ORG Int0addr
rjmp taste;

```

Im FLASH EPROM findet man den entsprechenden Eintrag auf der Word-Adresse 2 (Rote Markierung).

Memory:	prog FLASH	Address
prog 0x0000	0e c0 ff ff 2a c0 3f 06 06 5b 4f 66	
prog 0x0020	04 b9 0e 94 27 00 80 2f 90 e0 0e	

Einsprungsadresse
0x02

Entsprechend gilt für INT1:

```

; Einsprung externer Interrupt Routine
.ORG Int1addr
rjmp taste;

```

Memory:	prog FLASH	Address
prog 0x0000	0e c0 ff ff ff ff 2a c0 3f	
prog 0x0020	01 b9 0f ef 04 b9 0e 94 29 00 00	

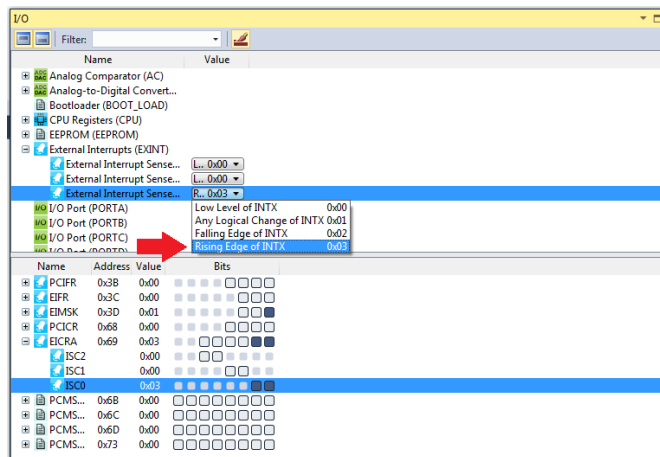
Einsprungsadresse
0x04

3.2.2.3 Initialisierung des INT0

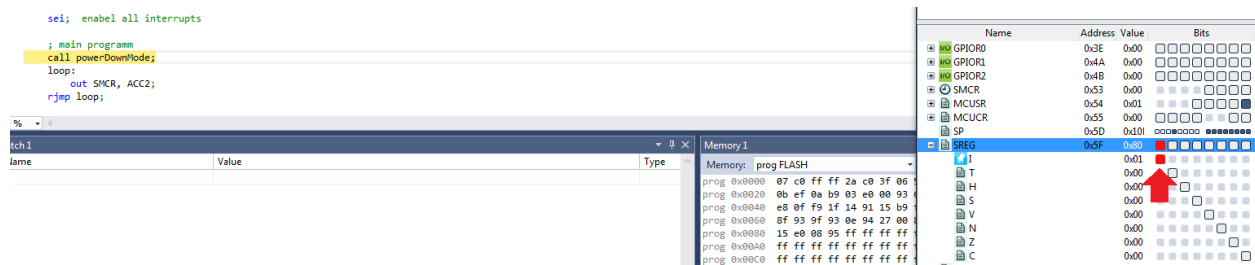
➔ Die steigende Flanke von INT0 (PD2) soll einen Interrupt auslösen

Beschreibung	Befehl
Interrupt PIN (PD2) = LESE PIN	<pre>; set int0 PD2 as read pin, all others are write pins ldi ACC1, (1<<DDD7) (1<<DDD6) (1<<DDD5) (1<<DDD4) (1<<DDD3) (1<<DDD1) (1<<DDD0); out DDRD, ACC1;</pre>
Der Interrupt 0 soll auf die steigende Flanke reagieren	<pre>ldi ACC1, (1<<ISC00) (1<<ISC01) ; rising edge of int0 generates an interrupt sts EICRA, ACC1;</pre>
Int0 aktivieren	<pre>ldi ACC1, (1<<INT0); INT0 aktivieren out EIMSK, ACC1;</pre>
Interrupts erlauben	<pre>sei; enable all interrupts</pre>

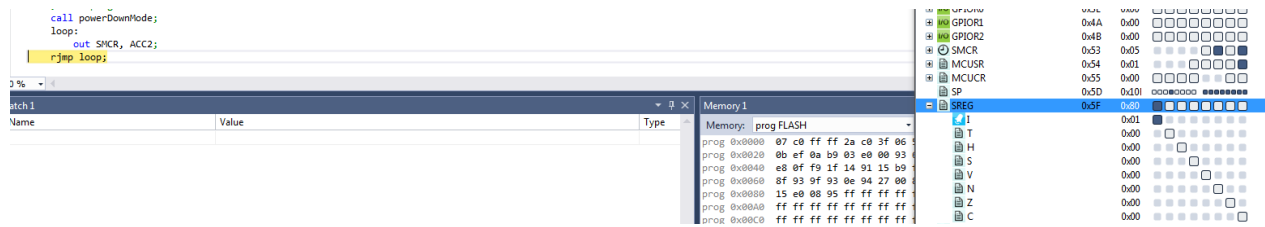
Wir sehen, dass RISING EDGE of INTX aktiviert ist.



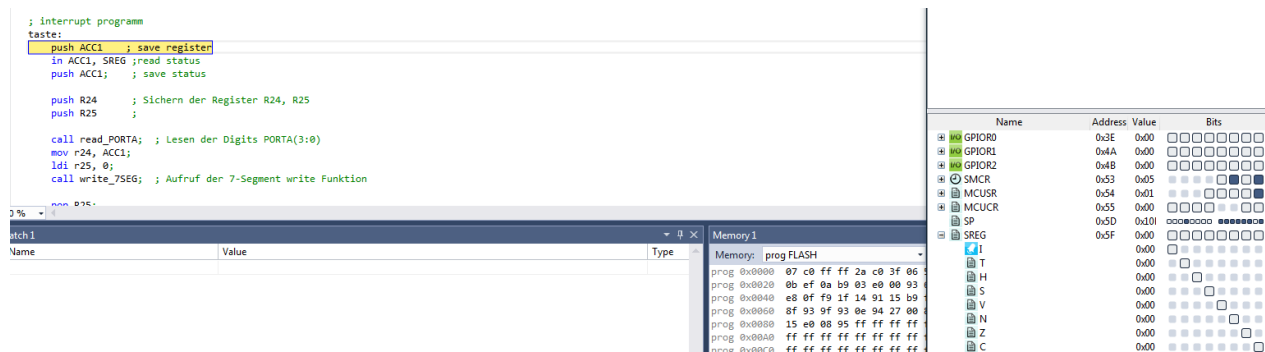
Nachdem der sei Befehl ausgeführt wird, ist das Interrupt FLAG auf 1 gesetzt.



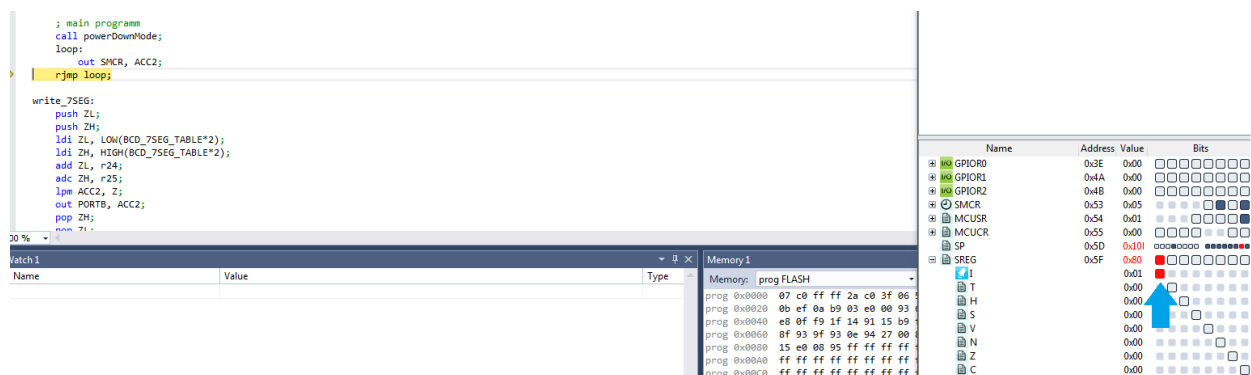
Nach Auslösen des Interrupts sehen wir dass das Interrupt Flag von der HW gelöscht wurde, also kein weiterer Interrupt ausgelöst werden kann. Natürlich kann man mit <sei> dies erlauben.



In der Interrupt-Routine ist das Interrupt Flag gelöscht, daher kann die Interrupt Routine nicht von einem Interrupt unterbrochen werden. Dies kann über sei (in der Interrupt Routine) ermöglicht werden.



Der Befehl **reti** (return from interrupt) setzt das Interrupt Flag wieder.



➔ Das Programm-Listing der Initialisierung

```

start:
    ; initialize stack
    ldi ACC1, LOW(RAMEND);
    out SPL, ACC1;
    ldi ACC1, HIGH(RAMEND);
    out SPH, ACC1;

    ; initialize ports
    ldi ACC1, (1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4);
    out DDRA, ACC1;
    ldi ACC1, 0xFF;
    out DDRB, ACC1;

    ; set int0 PD2 as read pin, all others are write pins
    ldi ACC1, (1<<DDD7)|(1<<DDD6)|(1<<DDD5)|(1<<DDD4)|(1<<DDD3)|(1<<DDD1)|(1<<DDD0);
    out DDRD, ACC1;

    ; -----
    ; - initialize interrupts          -
    ; - rising edge of INT1          -
    ; -----

    ldi ACC1, (1<<ISC00) | (1<<ISC01) ; rising edge of int0 generates an interrupt
    sts EICRA, ACC1;

    ldi ACC1, (1<<INT0); INT0 aktivieren
    out EIMSK, ACC1;

    sei; enable all interrupts

    ; main program
    call powerDownMode;
loop:
    out SMCR, ACC2;
    rjmp loop;

```

Abbildung 9 Initialisierung

Das Hauptprogramm ist eine loop Schleife die den power Down Modus aktiviert.

3.2.2.4 Die Interrupt Routine

```

; interrupt programm
taste:
    push ACC1      ; save register
    in ACC1, SREG ;read status
    push ACC1;     ; save status

    push R24      ; Sichern der Register R24, R25
    push R25      ;

    call read_PORTA; ; Lesen der Digits PORTA(3:0)
    mov r24, ACC1;
    ldi r25, 0;
    call write_7SEG; ; Aufruf der 7-Segment write Funktion

    pop R25;
    pop R24;

    pop ACC1;
    out SREG, ACC1;
    POP ACC1;

    call powerDownMode; ; Vorbereitenh des Power Down modes
    reti;

```

3.2.2.5 Das gesamte Listing

```

;
; BCD_7SEG.asm
.DEF ACC1 = R16;
.DEF ACC2 = R17;
.EQU takt = 16000000;

.CSEG
rjmp start;
.ORG 0x40; interrupt einspruenge übergehen

; Einsprung externer Interrupt Routine
.ORG Int0addr
rjmp taste;

BCD_7SEG_TABLE:
.DB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x27, 0x7F, 0x6E;
; .DB 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x02, 0x82, 0xD8, 0x80, 0x91;

start:
    ; initialize stack
    ldi ACC1, LOW(RAMEND);
    out SPL, ACC1;

```

```

ldi ACC1, HIGH(RAMEND);
out SPH, ACC1;

; initialize ports
ldi ACC1, (1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4);
out DDRA, ACC1;
ldi ACC1, 0xFF;
out DDRB, ACC1;

; set int0 PD2 as read pin, all others are write pins
ldi ACC1, (1<<DDD7)|(1<<DDD6)|(1<<DDD5)|(1<<DDD4)|(1<<DDD3)|(1<<DDD1)|(1<<DDD0);
out DDRD, ACC1;

; -----
; - initialize interrupts      -
; - rising edge of INT1      -
; -----

ldi ACC1, (1<<ISC00) | (1<<ISC01) ; rising edge of int0 generates an interrupt
sts EICRA, ACC1;

ldi ACC1, (1<<INT0); INT0 aktivieren
out EIMSK, ACC1;

sei; enable all interrupts

; main programm
call powerDownMode;
loop:
    out SMCR, ACC2;
    rjmp loop;

write_7SEG:
    push ZL;
    push ZH;
    ldi ZL, LOW(BCD_7SEG_TABLE*2);
    ldi ZH, HIGH(BCD_7SEG_TABLE*2);
    add ZL, r24;
    adc ZH, r25;
    lpm ACC2, Z;
    out PORTB, ACC2;
    pop ZH;
    pop ZL;
    ret;

read_PORTA:
    in ACC1, PINA;
    cbr ACC1, (1<<7);
    cbr ACC1, (1<<6);
    cbr ACC1, (1<<5);
    cbr ACC1, (1<<4);
    ret;

; interrupt programm
taste:
    push ACC1 ; save register
    in ACC1, SREG ;read status
    push ACC1; ; save status

```

```

    push R24      ; Sichern der Register R24, R25
    push R25      ;

    call read_PORTA; ; Lesen der Digits PORTA(3:0)
    mov r24, ACC1;
    ldi r25, 0;
    call write_7SEG; ; Aufruf der 7-Segment write Funktion

    pop R25;
    pop R24;

    pop ACC1;
    out SREG, ACC1;
    POP ACC1;

    call powerDownMode; ; Vorbereitenh des Power Down modes
    reti;

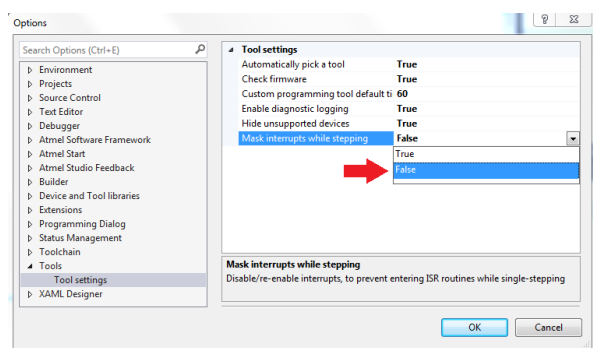
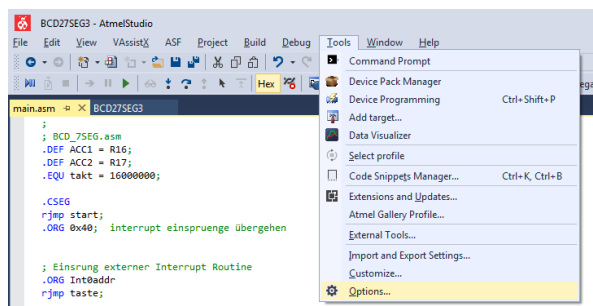
powerDownMode:
    ldi ACC2, (1<<SM1)|(1<<SE);
ret;

```

3.2.2.6 Debuggen der Interrupt Routine

Interrupts kann man debuggen wenn der Debug Modus im Simulator freigeschaltet wird.

Mit:



Für debuggen von Interrupt Funktionen setze **<Mask interrupt while stepping = false>**

Die Interrupt Routine <taste> wird ausgeführt, wenn auf PD2 ein L→H Übergang erkannt wird.

Stimuli Files für Interrupts. Toggeln des Pins INT0 (PD2).

<https://www.codeproject.com/Tips/1107908/Introduction-to-Simulate-External-Interrupts-Using>

Stimuli File TOGGLE PIND (PD2)

```
// Set PIN2 (INT0) high and low repeatedly
$repeat 2000000
  PIND |= 0x04
  #20
  PIND = 0x0
  #20
$endrep
```

3.3 Programmierung in C / C++

Getting started with AVR

https://www.youtube.com/playlist?list=PLtQdQmNK_0DRhBWYZ32BEILOykXLpJ8tP

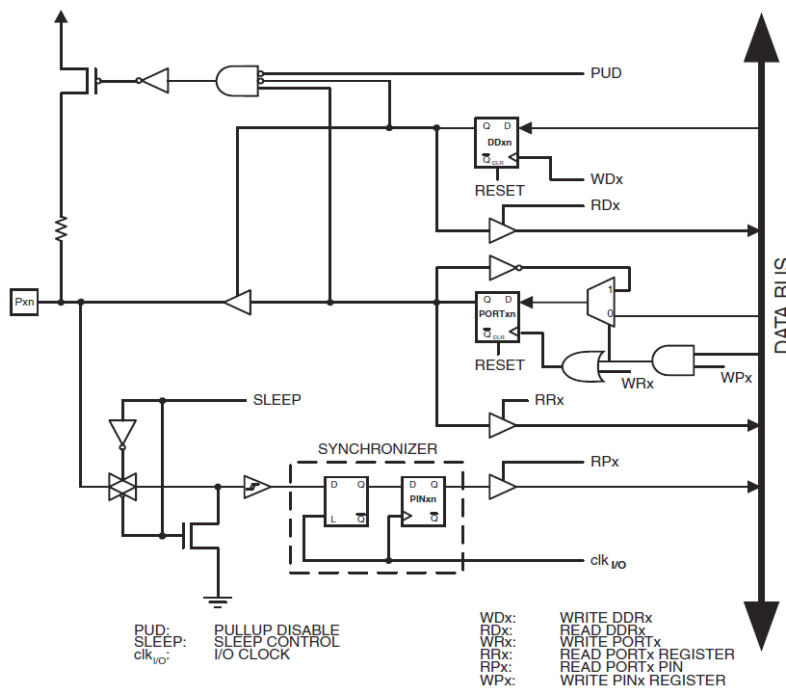
https://www.youtube.com/watch?v=mXPyRF-Y1E&list=PLtQdQmNK_0DRhBWYZ32BEILOykXLpJ8tP&index=4

3.4 Atmega644 Datasheet

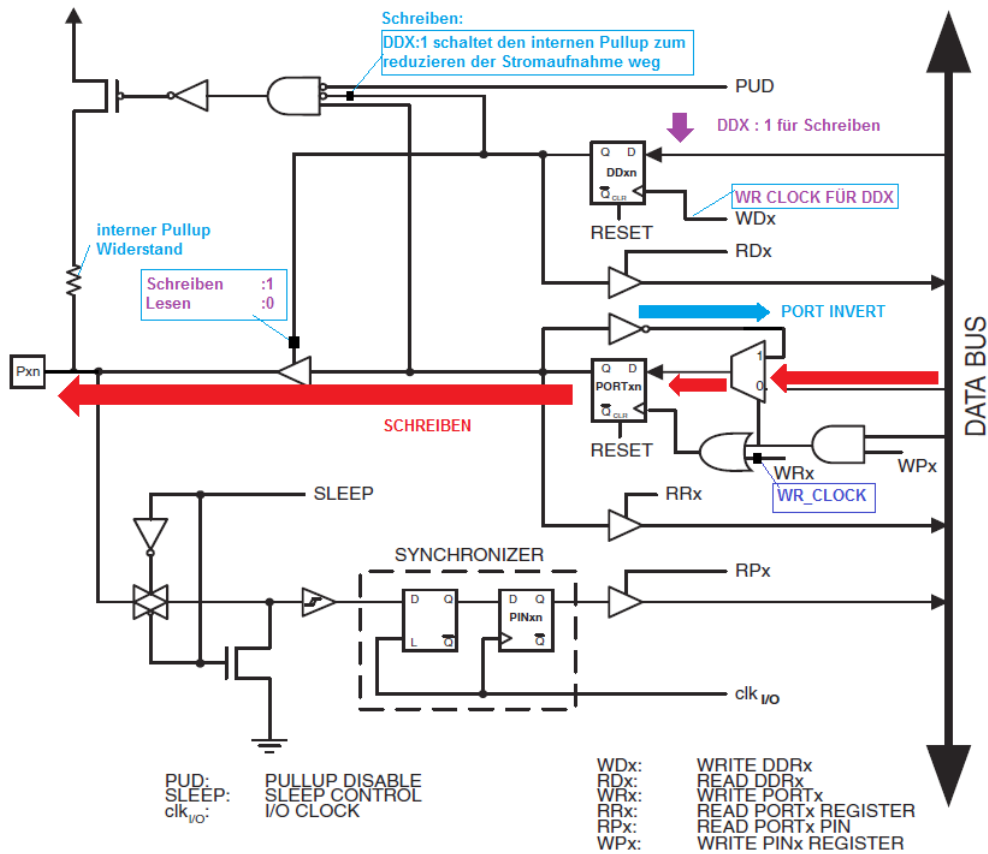
<http://www.atmel.com/images/doc2593.pdf>

3.4.1 Components of the ATMEGA644

3.4.1.1 I/O PORTS



- Schreiben auf einen PIN des I/O Ports



4 Anhang

4.1 Code Optimierung ausschalten

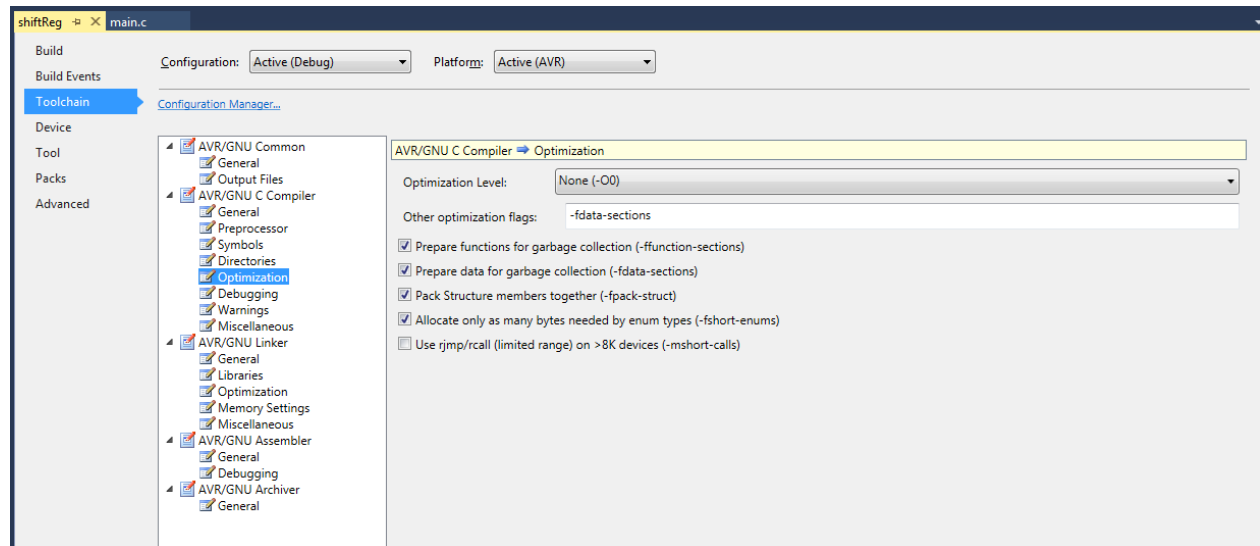


Abbildung 10 Code Optimierung ändern