

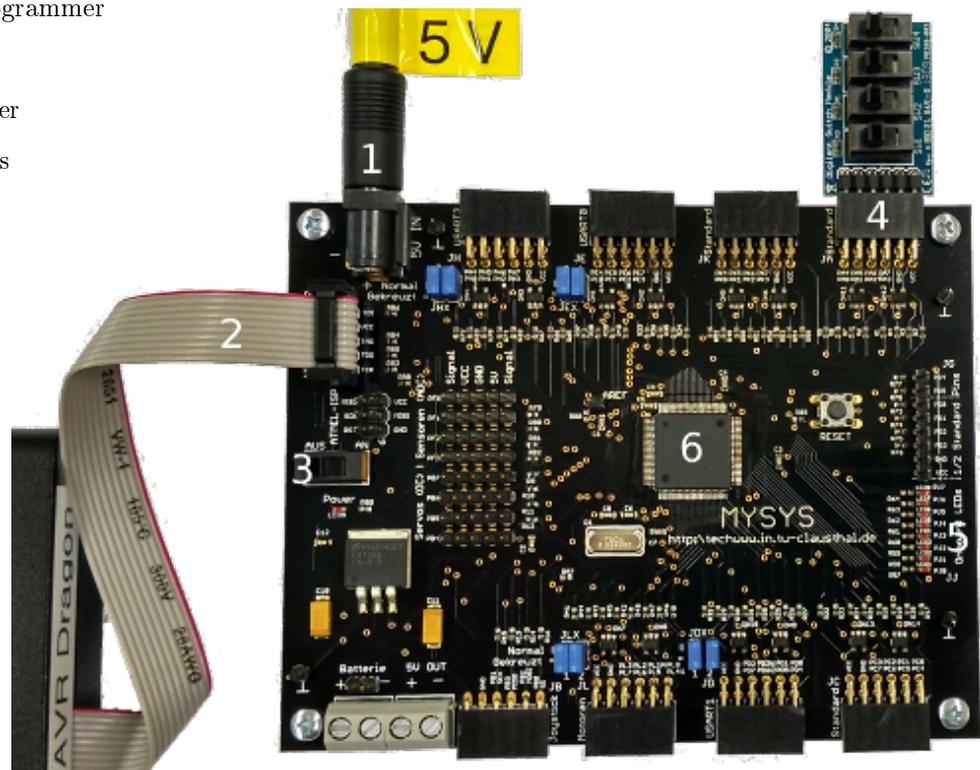
Rechnerarchitektur, Einführung in die Laborübungen

G. Kemnitz

24. November 2016

Inbetriebnahme der Mikrorechnerbaugruppe

1. Anschluss 5V-Netzteil
2. Anschluss Programmer
3. Einschalter
4. Eingabeschalter
5. Ausgabe LEDs
6. Prozessor



Verbindung auf dem PC herstellen

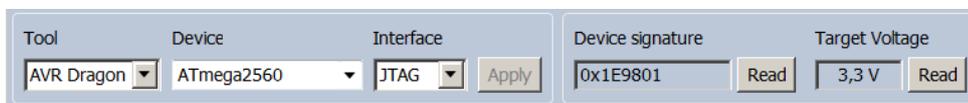
- Rechner unter Windows starten
- Web-Browser öffnen. Foliensatz zum Mitlesen öffnen:

techwww.in.tu-clausthal.de/site/Lehre/Rechnerarchitektur_2016/

- Atmel Studio starten 

Zur Kontrolle, ob der Prozessor richtig angeschlossen und vom System erkannt wird, in Atmel Studio:

- Tools > Device Programming



- Tool, Device, Interface einstellen; Apply, Read, Read: Device Signature und Target Voltage sollten richtig angezeigt werden.

- Kontrolle der Sicherungsbits (Fuses, Grundeinstellungen):

Interface settings	Fuse Name	Value
Tool information	✓ BODLEVEL	DISABLED ▾
Device information	✓ OCDEN	<input type="checkbox"/>
Memories	✓ JTAGEN	<input checked="" type="checkbox"/>
Fuses	✓ SPIEN	<input checked="" type="checkbox"/>
Lock bits	✓ WDTON	<input type="checkbox"/>
Production file	✓ EESAVE	<input checked="" type="checkbox"/>
	✓ BOOTSZ	4096W_1F000 ▾
	✓ BOOTRST	<input type="checkbox"/>
	✓ CKDIV8	<input type="checkbox"/>
	✓ CKOUT	<input type="checkbox"/>
	✓ SUT_CKSEL	EXTXOSC_3MHZ_8MHZ_1KCK_0MS ▾

Für Praktika sollten JTAGEN (JTAG Enabled), SPIEN (SPI Enabled) und OCDEN (On-Chip Debugging Enabled) gesetzt und WDTEN (WatchDog Timer Enabled) nicht gesetzt sein.

Das erste Programm

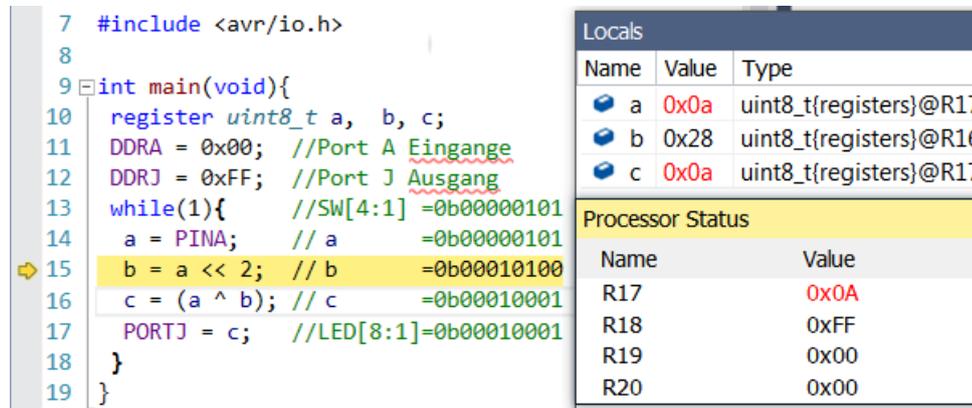
```
#include <avr/io.h>

int main(void){
    register uint8_t a, b, c;
    DDRA = 0x00; //Port A Eingänge
    DDRJ = 0xFF; //Port J Ausgang
    while(1){ //SW[4:1] = 0b00000101
        a = PINA; //a = 0b00000101
        b = a << 2; //b = 0b00010100
        c = (a ^ b); //c = 0b00010001
        PORTJ = c; //LED[8:1]=0b00010001
    }
}
```

Projekt anlegen:

- File > New > Project
- GCC Executable Project, ...
- (Fortsetzung)
- Name: logtest, ...
- Device: rechts oben ATmega2560 eingeben und dann links auswählen, OK
- Programm eingeben.
- Übersetzen: Build > Build Solution (F7).
- Wenn Fehler angezeigt werden, diese beseitigen.
- Programmer auswählen: Project > logtest Properties (Alt+F7) > Tools > Select debugger ...: AVR Dragon ..., Interface: JTAG;
- Compileroptimierung ausschalten: ... > Toolchain > AVR/GNU C Compiler > Optimization > Optimization Level »None (-O0)«; Speichern (Strg+S).
- Programm im Debugger-Modus starten: Debug > Start Debugging and Break (Alt+F5).
- Fenster zum Anschauen der Variablen öffnen: Debug > Windows > Locals (Alt+F4).
- ...

- Fenster zum Anschauen der Prozessorregister öffnen¹: Debug > Windows Processor Status.
- Schalterwerte SW[4:1]=1010 einstellen, ...
- Mit  (Step Into, F11) Programm zeilenweise abarbeiten.
- Werte der Variablen und der Register, in denen sie stehen und nach Zeile 17 LED-Ausgabe kontrollieren.



```

7 #include <avr/io.h>
8
9 int main(void){
10     register uint8_t a, b, c;
11     DDRA = 0x00; //Port A Eingänge
12     DDRJ = 0xFF; //Port J Ausgang
13     while(1){ //SW[4:1] =0b00000101
14         a = PINA; // a =0b00000101
15         b = a << 2; // b =0b00010100
16         c = (a ^ b); // c =0b00010001
17         PORTJ = c; //LED[8:1]=0b00010001
18     }
19 }

```

Locals		
Name	Value	Type
a	0x0a	uint8_t(registers)@R17
b	0x28	uint8_t(registers)@R16
c	0x0a	uint8_t(registers)@R17

Processor Status	
Name	Value
R17	0x0A
R18	0xFF
R19	0x00
R20	0x00

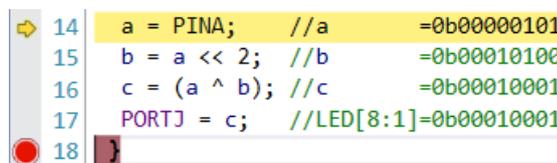
Test mit Schaltern und LEDs

- Programm mit  (Continue, F5) starten und für eine Stichprobe von Schalterwerten die LED-Ausgabe kontrollieren:

Schalter und a	0000 1010	0000 1011	...
Variable b	0010 1000	0010 1100	...
c, LEDs	0010 0010	0010 0111	...

Test mit Unterbrechungspunkt

- Programm mit  (Break all, Shift+F5) anhalten.
- Unterbrechungspunkt nach LED-Ausgabe (Rechtsklick auf grauen Rand).



```

14 a = PINA; //a =0b00000101
15 b = a << 2; //b =0b00010100
16 c = (a ^ b); //c =0b00010001
17 PORTJ = c; //LED[8:1]=0b00010001
18 }

```

- Nach Start mit  (Continue, F5) hält das Programm am Unterbrechungspunkt und erlaubt eine Kontrolle und Veränderung der Variablenwerte.

-
- Disassembliertes Programm anzeigen: Debug > Windows > Disassembly (Alt+8).

¹Wegen »register uint8_t a, ...« stehen die Variablen in Registern.

```

    a = PINA;    //x    =0b00000101
    0000008C LDI R24,0x20    Load immediate
    0000008D LDI R25,0x00    Load immediate
    0000008E MOVW R30,R24    Copy register pair
    0000008F LDD R17,Z+0    Load indirect with displacement
    b = a << 2; //a    =0b00010100
    00000090 MOV R16,R17    Copy register
    00000091 LSL R16    Logical Shift Left
    00000092 LSL R16    Logical Shift Left
    c = (a ^ b); //y    =0b00010001
    00000093 EOR R17,R16    Exclusive OR
    PORTJ = c; //LED[8:1]=0b00010001
    00000094 LDI R24,0x05    Load immediate
    00000095 LDI R25,0x01    Load immediate
    00000096 MOVW R30,R24    Copy register pair
    00000097 STD Z+0,R17    Store indirect with displacement
    }
    00000098 RJMP PC-0x000C    Relative jump

```

Compiler-Optimierung

Mit Optimierung (-O1 und höher) entstehen kürzere Programme:

- Stop Debugging ■ (Alt+Shift+F5).
- Project > logi Properties (Alt+F7) > Toolchain > AVR/GNU C Compiler > Optimization > Optimization Level »-O1«
- Speichern (Strg+S) , Übersetzen, Neustarten, ...

```

    a = PINA;    //x    =0b00000101
    00000083 IN R24,0x00    In from I/O location
    b = a << 2; //a    =0b00010100
    00000084 MOV R25,R24    Copy register
    00000085 LSL R25    Logical Shift Left
    00000086 LSL R25    Logical Shift Left
    c = (a ^ b); //y    =0b00010001
    00000087 EOR R24,R25    Exclusive OR
    PORTJ = c; //LED[8:1]=0b00010001
    00000088 STD Z+0,R24    Store indirect with displacement
    }
    00000089 RJMP PC-0x0006    Relative jump

```

Programmierung in Assembler

```

.global main
main:
    OUT 0x01,R1    ; DDRA = 0x00; (Port A Eingänge)
                  ; R1 muss bei Aufruf immer 0 sein
    SER R24        ; R24 = 0xFF
    STS 0x0104,R24 ; DDRJ = 0xFF; (Port J Ausgänge)
loop:
    IN R24,0x00    ; a = PINA;
    MOV R25,R24    ; b = a << 2;
    LSL R25
    LSL R25
    EOR R24,R25    ; c = (a ^ b);
    STS 0x105,R24  ; PORTJ = c;
    RJMP loop

```

Die Sprungmarke »main« muss als global vereinbart sein und das Unterprogramm »int main()« ist auszukommentieren.

- Debugger beenden ■ (Alt+Shift+F5).
- Assemblerdatei anlegen: Project > Add neu Item (Ctrl+Shift+A) > Assembler File > logi2.s
- Programm von der Folie zuvor eingeben.
- Im C-Programm alle Zeilen ab »main()« mit /* ... */ auskommentieren.
- Übersetzen: Built > Built Solution (F7).
- Programm im Debugger starten: Debug > Start Debugging and Break.
- Ausprobierend im Schrittbetrieb und freilaufend.

Von der logischen Funktion zum Programm

Der Programmrahmen (Initialisierung, Endlosschleife, Einlesen der Schalterwerte und Ergebnisausgabe an die LEDs) bleibt. Nur die logische Verarbeitung dazwischen ist zu ändern.

```
#include <avr/io.h>

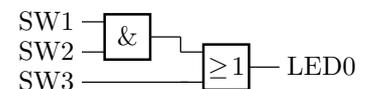
int main(void){
    register uint8_t a, b, c, ...;
    DDRA = 0x00; //Port A Eingänge
    DDRJ = 0xFF; //Port J Ausgang
    while(1){ //Endlosschleife
        a = PINA; //Einlesen: SW[4:1]

        <Programmierung der logischen Funktion>

        PORTJ = ...; //Ausgabe an LED[8:1]
    }
}
```

Beispiel und seine Programmierung in C

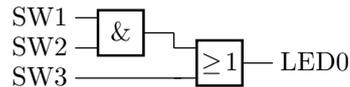
```
register uint8_t a, b, c, d, e;
...
a = PINA;
b = a >> 1;
c = b >> 1;
d = (a & b) | c;
e = d & 1;
PORTJ = e;
```



- Auskommentieren Assemblerprogr.: Einrahmen mit /* ... */.
- Einkommentieren des C-Programms (Löschen von /* ... */).
- Ändern der Logikberechnung.
- Übersetzen: Built > Built Solution (F7).
- Debugger starten: Debug > Start Debugging and Break.
- Ausprobierend im Schrittbetrieb und freilaufend.

Programmierung als Assemblerprogramm

- Disassembliertes Programm anzeigen: Debug > Windows > Disassembly (Alt+8).
- Mit Copy und Paste in eine Datei kopieren.
- Debugger beenden ■ (Alt+Shift+F5).
- »main()« im C-Programm auskommentieren.
- Assemblerprogramm einkommentieren (Löschen von /*...*/).
- Befehlsfolge zur Logikberechnung ändern²:



- Übersetzen: Built > Built Solution (F7).
- Debugger starten: Debug > Start Debugging and Break.
- Ausprobierend im Schrittbetrieb und durchlaufend.

Assemblerprogramm für die Beispielfunktion

```

.global main
main:
    OUT 0x01,R1    ; DDRA = 0x00; (Port A Eingänge)
    SER R24       ; R24 = 0xFF
    STS 0x0104,R24; DDRJ = 0xFF; (Port J Ausgänge)
loop:
    IN R24,0x00   ; a = PINA;
    LSR R24       ; b = b >> 1
    AND R24,R25   ; c = a & b
    LSR R24       ; c = c >> 1
    OR  R24,R25   ; d = c | b
    ANDI R24,1    ; e = d & 1
    STS 0x105,R24 ; PORTJ = e;
    RJMP loop
  
```

Prägen Sie sich die einzelnen Arbeitsschritte für die eigenständige Lösung der nachfolgenden Aufgaben ein.

²Die erforderlichen Befehle kann man dem gespeicherten disassemblierten C-Programm entnehmen. Die nächste Folie zeigt eine Beispielbefehlsfolge.