Low Level AVR

Arduino? Nein danke!

Felix Queißner

shackspace

2020

Wer bin ich?

- Felix „xq“ Queißner
- Baujahr 1993
- Mit 12 angefangen, zu programmieren
- Mit 13 angefangen, Elektronik zu basteln
- Mache gerne Dinge mit Code und alten Computern

Worum geht's?

- Grundlagenwissen AVR-Programmierung vermitteln
- „Wie komme ich klar, wenn es mit der Arduino-IDE nicht mehr weiter geht?“
- „Awareness schaffen“ für Code Bloat
- **Nicht:** AVR/Arduino löten
- **Nicht:** C programmieren lernen

Outline

Wie kommen wir von hier ...

```
int button = getPressedButton();
if (button == 1) {
  digitalWrite(RELAIS_1_PIN, HIGH);
  delay(350);
  digitalWrite(RELAIS_2_PIN, LOW);
  digitalWrite(RELAIS_3_PIN, LOW);
  digitalWrite(RELAIS_4_PIN, LOW);
}
if (button == 2) {
  digitalWrite(RELAIS_1_PIN, LOW);
  digitalWrite(RELAIS_2_PIN, HIGH);
  delay(350);
  digitalWrite(RELAIS_3_PIN, LOW);
  digitalWrite(RELAIS_4_PIN, LOW);
}
if (button == 3) {
  digitalWrite(RELAIS_1_PIN, LOW);
  digitalWrite(RELAIS_2_PIN, LOW);
  digitalWrite(RELAIS_3_PIN, LOW);
```

… nach da?

```
1 int button = getPressedButton();
2 PORTD = 0x00;
3 if (button != 0) {
4   delay(350);
5   PORTD = (1 << (button + 2));
6 }
```

The Arduino way

- Schnell von „Idee“ zu „Es blinkt schon mal was“ kommen
- Single-Click-Solution
- Mit was man genau arbeitet ist eigentlich egal, es erfüllt alles den gleichen Zweck

Blink

- Lässt eine LED mit $\frac{1}{2}$ Hertz blinken
- Benutzt `setup()` und `loop()`

```
1  void setup() {
2    pinMode(LED_BUILTIN, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(LED_BUILTIN, HIGH);
7    delay(1000);
8    digitalWrite(LED_BUILTIN, LOW);
9    delay(1000);
10 }
```

Serial

- Gibt `"Hello, World!"` auf der Konsole aus
- Spiegelt anschließend die Texteingabe zurück
- Benutzt `setup()` und `loop()`
- Benutzt `Serial.*`

```
1 void setup() {
2   Serial.begin(19200);
3   Serial.print("Hello, World!\r\n");
4 }
5
6 void loop() {
7   Serial.write(Serial.read());
8 }
```

Pro/Contra Arduino

Pro:

- Man kommt schnell zum Ziel
- Zugrunde liegened Hardware ist schnell ausgetauscht
- Es gibt viele Beispiele für die Arduino-Welt
- „Batterien inklusive“

Contra:

- Es passiert sehr viel *Magie*
- Arduino-Programme sind „fett und träge“
- Wir haben keine wirkliche Kontrolle über das Programm

In vier Schritten zum Ziel

1. Dokumentation lesen
2. Code schreiben
3. Compilen & Linken
4. Flashen

Warum tun wir uns das an?

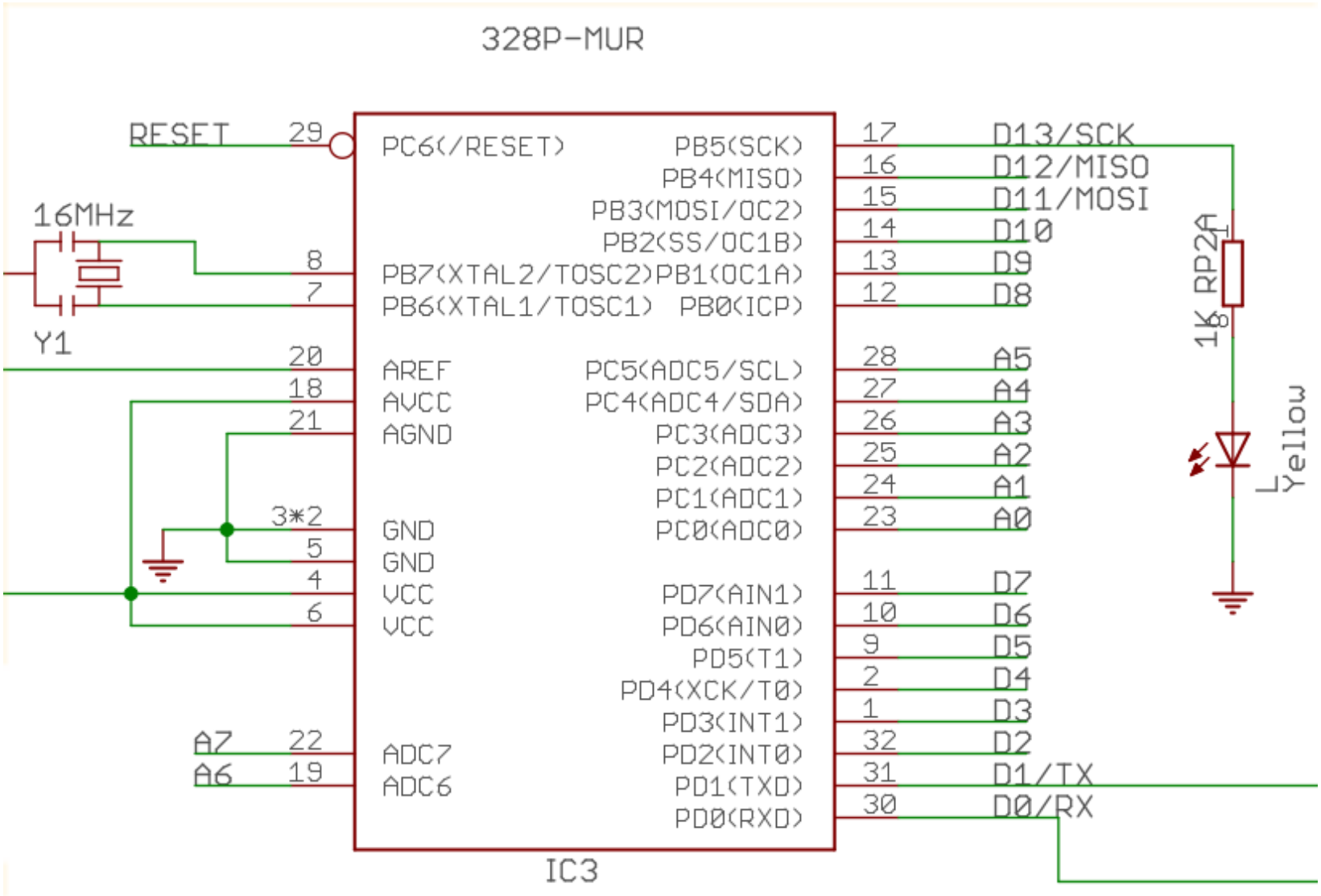
- Programme werden kleiner
- Programme werden schneller
- Wir können die Hardware voll ausnutzen

Schritt 1: Dokumentation lesen

Welche Dokumente brauchen wir?

- Schaltplan
- Datenblatt

Schaltplan


328P-MUR
RESET
29
PC6(/RESET)
PB5(SCK)
PB4(MISO)
PB3(MOSI/OC2)
PB2(SS/OC1B)
PB7(XTAL2/TOSC2)
PB1(OC1A)
PB6(XTAL1/TOSC1)
PB0(ICP)
16MHz
8
7
Y1
20
AREF
18
AVCC
21
AGND
3*2
GND
5
GND
4
VCC
6
VCC
A7
22
ADC7
A6
19
ADC6
PC5(ADC5/SCL)
PC4(ADC4/SDA)
PC3(ADC3)
PC2(ADC2)
PC1(ADC1)
PC0(ADC0)
PD7(AIN1)
PD6(AIN0)
PD5(T1)
PD4(XCK/T0)
PD3(INT1)
PD2(INT0)
PD1(TXD)
PD0(RXD)
17
D13/SCK
16
D12/MISO
15
D11/MOSI
14
D10
13
D9
12
D8
28
A5
27
A4
26
A3
25
A2
24
A1
23
A0
11
D7
10
D6
9
D5
2
D4
1
D3
32
D2
31
D1/TX
30
D0/RX
IC3
1K
RP2A
1
8
Yellow

Schaltplan

- *Ground Truth* für Pinbelegungen
- Zeigt uns, wie die Hardware funktioniert
- Offenbart manchmal undokumentierte Möglichkeiten

Datenblatt

11.3.3 Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 11-9.

Table 11-9. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	AIN1 (Analog Comparator Negative Input) PCINT23 (Pin Change Interrupt 23)
PD6	AIN0 (Analog Comparator Positive Input) OC0A (Timer/Counter0 Output Compare Match A Output) PCINT22 (Pin Change Interrupt 22)
PD5	T1 (Timer/Counter 1 External Counter Input) OC0B (Timer/Counter0 Output Compare Match B Output) PCINT21 (Pin Change Interrupt 21)
PD4	XCK (USART External Clock Input/Output) T0 (Timer/Counter 0 External Counter Input) PCINT20 (Pin Change Interrupt 20)

Datenblatt

- *Ground Truth* für Hardware-Funktionalität
- Zeigt uns, wie der Microcontroller funktioniert
- Hilfreich: Doppelfunktionen für Pin-Belegungen

Schritt 2: Code schreiben

Allgemein:

- C ist die primäre Embedded-Programmiersprache
- C++, Basic, Pascal, Zig, Rust, ... sind auf AVR ebenfalls verfügbar
- Dynamische Speicherverwaltung wird selten benötigt

Hands on:

1. Code
2. Datenblatt verwenden
3. Komplexer Hardware-Init
4. Interrupt

Hands on: Code (Blink 1)

```
1  #include <stdbool.h>
2  #include <avr/io.h>
3  #include <util/delay.h>
4
5  int main()
6  {
7    DDRB = (1 << PIN5);
8    while (true)
9    {
10     PORTB ^= (1 << PIN5);
11     for (int i = 0; i < 5; i++)
12       _delay_ms(200);
13   }
14 }
```

Hands on: Datenblatt (Blink 2)

```
1  int main() {
2    TCCR1A = 0;
3    TCCR1B = (1 << CS12) | (1 << CS10) | (1 << WGM12);
4    TCCR1C = 0;
5
6    OCR1AH = (TIMER1_LIMIT >> 8) & 0xFF;
7    OCR1AL = (TIMER1_LIMIT & 0xFF);
8
9    TIMSK1 = (1 << OCIE1A);
10
11   sei();
12   while (true);
13 }
14 ISR(TIMER1_COMPA_vect) {
15   PORTB ^= (1 << PIN5);
16 }
```

Hands on: Hardware-Init (UART 1)

```
1  void uart_tx(char c) {
2    while((UCSR0A & (1<<UDRE0)) == 0);
3    UDR0 = c;
4  }
5  char uart_rx() {
6    while((UCSR0A & (1<<RXC0)) == 0);
7    return UDR0;
8  }
9
10 int main() {
11   UCSR0A = 0;
12   UCSR0B = (1<<RXEN0) | (1<<TXEN0);
13   UCSR0C =  (1<<UCSZ01) | (1<<UCSZ00);
14   UBRR0 = (UBRR_BAUD & 0xFFF);
15
16   while(true) {
17     char c = uart_rx();
18     uart_tx(c);
19   }
20 }
```

Hands on: Interrupts (UART 2)

```
1  int main() {
2    …
3
4    UCSR0B |= (1<<RXCIE0);
5
6    sei();
7
8    while(true);
9  }
10
11 ISR(USART_RX_vect) {
12   UDR0 = UDR0;
13 }
```

Schritt 3: Compilen & Linken

- Was ist ein
  - Compiler?
  - Linker?
  - Bibliothek?
  - Makefile?
- Dateiformate
- Tools

Was ist ein Compiler?

- Übersetzt Quellcode in Maschinensprache
- Gibt „Object Files“ aus
- Kann verschiedene Optimierungen durchführen
-

```
avr-gcc \
  -mmcu=atmega328p \
  -Os \
  -DF_CPU=16000000ULL \
  -o example.o \
  -c \
  example.c
```

Was ist ein Linker?

- Verknüpft „Object Files“
- Weißt allen Dingen Adressen zu
- Sortiert Objekte im Speicher
- Bindet Bibliotheken ein
- Ist konfigurierbar (Linkerscript)

```
avr-ld \
  -mmcu=atmega328p \
  -o example.elf \
  -lm \
  example.o additional.o
```

Was ist eine Bibliothek?

Grundlegend:

- Sammlung von Funktionalität
- Erlaubt Wiederverwendung von Code

Statische Bibliothek:

- Eine Sammlung von vorcompilierten Object Files
- Keine Laufzeitkosten
- Kann mit der richtigen Technik sogar nachträglich optimiert werden

Dynamische Bibliothek:

- Wird zu Programmstart in unsere ausführbare Datei geladen und gelinkt
- Im Embedded-Bereich quasi nie eingesetzt
- Konzept auf AVR unmöglich umzusetzen

Was ist ein Makefile?

Wofür ein Buildsystem?

- Viele wiederkehrende Tasks beim Entwickeln
- Bedingtes Ausführen von Tasks
- Bequeme Bedienung

GNU Make:

- Einfach zu bedienen
- `Makefile` enthält Regeln zum Ausführen von Befehlen
- Tracking von Abhängigkeiten über Schreibdatum
- Kann beliebige Sequenz von Shellbefehlen ausführen

Was ist ein Makefile?

```
1  all: blink1.elf
2    avr-size --mcu=$(MCU) $^
3
4  hex: blink1.hex
5
6  flash: blink.hex
7    avrdude -P /dev/ttyUSB0 -c arduino -b 57600 -p
       atmega328p -e -U flash:w:$<
8
9  %.elf: %.o
10   avr-ld -mmcu=atmega328p -o $@ -lm $^
11
12 %.hex: %.elf
13   avr-objcopy -O ihex $< $@
14
15 %.o: %.c
16   avr-gcc -mmcu=atmega328p -Os -o $@ -c $<
17
18 .PHONY: flash
19 .SUFFIXES:
```

Dateiformate

- ELF object
  - Ist das Ergebnis des Compilers
  - Enthält compilierten Code ohne absolute Adressen
  - Muss noch gelinkt werden, wiederverwendbar
  - Enthält Debug-Informationen
- ELF executable
  - Ist das Ergebnis des Linkers
  - Enthält compilierten Code mit absolute Adressen
  - Enthält Debug-Informationen
- Intel HEX
  - Einfaches Plain-Text-Format
  - Enthält ausschließlich Daten
  - Speichert eine Ansammlung aus Adresse-Daten-Paaren

Tools

- Theoretisch ist das vorgestellte ausreichend zum Entwickeln
- Arbeit ist mühsahm, Trial und Error
- Gibt in den *binutils* eine große Menge an Tools, die die Arbeit erleichtern

Beispiele:

- objdump
- size
- addr2line
- ...

objdump

- Dumpt Object Files und Executables
- Disassembler
- Hex-Dump
- Symboltabellen
- ...

objdump

```
blink1.elf:     file format elf32-avr

Disassembly of section .text:

00000080 <main>:
  80: 80 e2         ldi r24, 0x20 ; 32
  82: 84 b9         out 0x04, r24 ; 4
  84: 90 e2         ldi r25, 0x20 ; 32
  86: 85 b1         in  r24, 0x05 ; 5
  88: 89 27         eor r24, r25
  8a: 85 b9         out 0x05, r24 ; 5
  8c: 2f ef         ldi r18, 0xFF ; 255
  8e: 33 ec         ldi r19, 0xC3 ; 195
  90: 89 e0         ldi r24, 0x09 ; 9
  92: 21 50         subi  r18, 0x01 ; 1
  94: 30 40         sbci  r19, 0x00 ; 0
  96: 80 40         sbci  r24, 0x00 ; 0
  98: e1 f7         brne  .-8       ; 0x92 <main+0x12>
  9a: 00 c0         rjmp  .+0       ; 0x9c <main+0x1c>
  9c: 00 00         nop
  9e: 2f ef         ldi r18, 0xFF ; 255
```

objdump

```
SYMBOL TABLE:
000000ec g       .text   00000000 _etext
0000007c  w      .text   00000000 __vector_24
0000007c  w      .text   00000000 __vector_12
0000007c g       .text   00000000 __bad_interrupt
000000ec g       *ABS*   00000000 __data_load_end
0000007c  w      .text   00000000 __vector_6
00000068 g       .text   00000000 __trampolines_end
0000007c  w      .text   00000000 __vector_3
0000007c  w      .text   00000000 __vector_23
000000ec g       *ABS*   00000000 __data_load_start
00000068 g       .text   00000000 __dtors_end
00000400 g       *ABS*   00000000 __LOCK_REGION_LENGTH__
0000007c  w      .text   00000000 __vector_25
0000007c  w      .text   00000000 __vector_11
00000068  w      .text   00000000 __init
0000007c  w      .text   00000000 __vector_13
0000007c  w      .text   00000000 __vector_17
0000007c  w      .text   00000000 __vector_19
0000007c  w      .text   00000000 __vector_7
00810000 g       .text   00000000 __eeprom_end
```

size

- Gibt eine Zusammenfassung der Sektionsgrößen aus

```
AVR Memory Usage
----------------
Device: atmega328p

Program:     236 bytes (0.7% Full)
(.text + .data + .bootloader)

Data:          0 bytes (0.0% Full)
(.data + .bss + .noinit)
```

addr2line

- Gibt für eine Adresse die dazu passende Codezeile
- Funktioniert nur, wenn mit Debugsymbolen compiliert wird (`-gdwarf-2`)

```
1 [user@pc src]$ avr-addr2line -e blink1.elf 86
2 /home/felix/projects/avr/src/blink1.c:10
3 [user@pc src]$ avr-addr2line -e blink1.elf e6
4 /home/felix/projects/avr/src/blink1.c:11
```

Schritt 4: Flashen

- Möglichkeiten
  - High Voltage Serial Programming
  - In-System Programming
  - Bootloader
- Tools
  - avrdude
  - PonyProg
  - AVRStudio
  - ...

In-System Programming

- Benötigt einen 6-Pin-Header auf der Platine
- Alternativ: 10-Pin-Header
- **Pro:** Flash ist zu 100% nutzbar
- **Pro:** Ermöglicht Änderung der Fuse Bits
- **Contra:** Benötigt spezielle Programmierhardware
- **Contra:** Mäßig schnell

Bootloader

- Benötigt eine *beliebige* Schnittstelle zum System
- Liegt am Ende des Flashs, wird zu Beginn ausgeführt
- Ist frei programmierbare Software
- **Pro:** Benötigt keine spezielle Hardware
- **Pro:** Support beliebiger Programmiermöglichkeiten
- **Pro:** Kann max. CPU-Frequenz ausnutzen
- **Contra:** Ermöglicht keine Änderung der Fuse Bits
- **Contra:** Flash ist nicht vollständig nutzbar
- **Contra:** Keine standardisierte Programmierschnittstelle

avrdude

- Open-Source Tool
- Kann quasi jede Programmierhardware ($>$ 90 unterstützte Geräte)
- Kommandozeilentool, gibt aber GUIs

```
avrdude \
  -P /dev/ttyUSB0 \
  -c arduino \
  -p atmega328p \
  -e \
  -U flash:w:example.hex
```

Was haben wir gewonnen?

- Wir haben die Kontrolle über den Code
- Wir können die Hardware voll ausnutzen
- Unsere Programme sind kleiner
- Unsere Programme benötigen weniger CPU-Zeit

Vergleich: Blink

```
1   text    data     bss     dec     hex filename
2    924       0       9     933     3a5 arduino-blink.
    elf
3    236       0       0     236      ec blink1.elf
4    198       0       0     198      c6 blink2.elf
```

- Größenreduktion auf 25% bzw. 21% im Vgl. zum Arduino-Code
- Interrupt-Lösung benötigt quasi keine Rechenzeit
- Blink liese sich sogar ohne Rechenzeit implementieren

Vergleich: Serial

```
   text    data     bss     dec     hex filename
   1454      34     166    1654     676 arduino-uart.elf
    306      16       0     322     142 uart1.elf
    322      16       0     338     152 uart2.elf
```

- Größenreduktion auf 19% bzw. 20% im Vgl. zum Arduino-Code
- Massiv weniger RAM-Verbrauch (10% beim Arduino vs. 0,7% bei Selbstprogrammiert)
- Interrupt-Lösung benötigt quasi keine Rechenzeit

Wie geht's weiter?

Programmierung:

- C++
- Bibliotheken

Links:

- mikrocontroller.net
- Roboternetz.de
- Quellcode und Slides
  https://github.com/MasterQ32/avr-tutorial

Fragen?