

1 Hello Blinking World

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

1. Hello Blinking World

AVR Programmierung für Dummies

Achtung!

Ziele

Video

Es werde Licht! Oder auch nicht...

Ziele

weiterführende Links

Video

Übung

2

2

2

2

2

2

3

2

3

1. Hello Blinking World

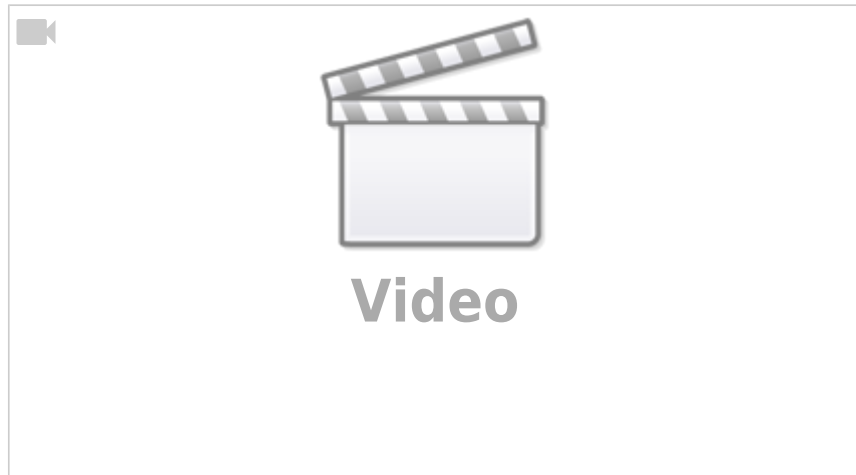
AVR Programmierung für Dummies

Im Video werden im ersten Teil (bis 15min30sec) die Grundkonzepte für die AVR Programmierung erklärt.

Achtung!

Ab der 16. Minute geht es in die Assembler Programmierung. Dies ist nicht Teil des Kurses Mikroprozessortechnik.

Video



Ziele

Nach dieser Lektion sollten Sie:

1. die verschiedenen Pins von AVR Chips kennen.
2. die verschiedenen Arten von Speichern in Chips kennen.
3. wissen was Register sind.
4. wissen für was das Spezialregister DDRx ($x=\{A, B, C, D\}$) nützlich ist.

Es werde Licht! Oder auch nicht...

Ziele

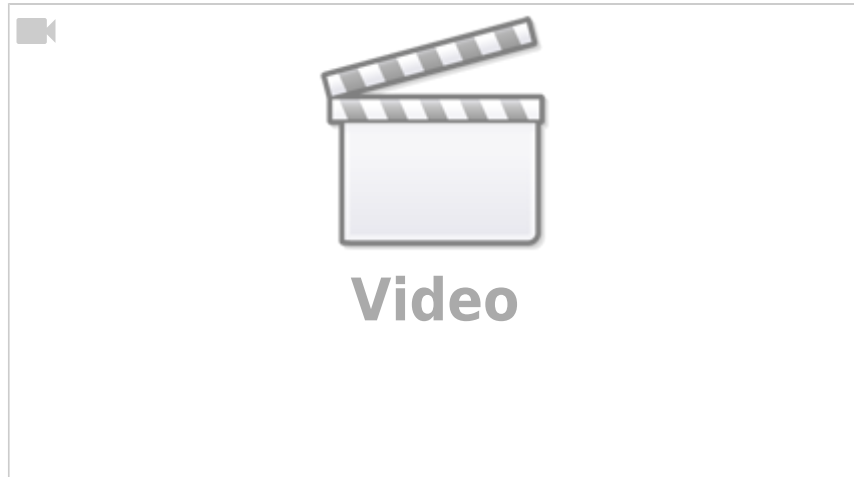
Video

Nach dieser Lektion sollten Sie:

1. wissen, wie man im Atmel Studio ein Projekt anlegt.
2. wissen, wie der Programmierungsumgebung

die Taktfrequenz des Microcontrollers festgelegt wird.

3. die wichtigsten Bitmanipulationen (Bitmaske zum setzen und löschen eines einzelnen Bits, togglen) kennen und anwenden können.



weiterführende Links

- kurzes Youtube Video: [ein einzelnes Bit setzen oder löschen mit bitweisen Operationen](#)

Übung

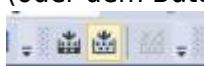
I. Vorarbeiten

[main.c](#)

- installieren Sie [SimulIDE und Atmel Studio](#)
- falls es Probleme bei der Programmierung gibt: nutzen Sie die [Tipps für die Fehlersuche](#)

II. Eingabe in Atmel Studio

- öffnen Sie Atmel Studio
- Anlegen eines neuen Projekts
 - File » New » Project...
 - Wählen Sie "GCC C Executable Project", da ein ausführbares C Code Beispiel erstellt werden soll
 - Geben Sie bei Name Einfuehrung_v01 und drücken Sie auf Ok
 - Der Cursor sollte nun im Eingabefeld des Suchfenster für die Microcontroller stehen. Geben Sie dort 328 ein
 - Wählen Sie den ATmega328 aus den möglichen Chips aus und drücken Sie auf Ok
- Eingabe und Kompilieren des Code
 - Ersetzen Sie den vorhandenen Code, durch den rechts stehenden Code
 - Kompilieren Sie den Code durch Build » Build solution (oder dem Button "Build Solution"



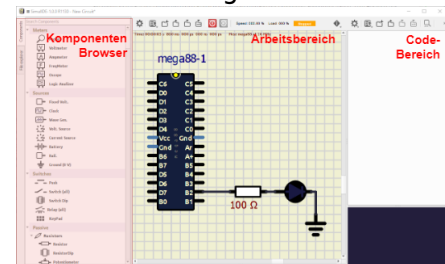
oder <F7>)

```

1. #define F_CPU
   8000000UL
2.
3. #include
   <avr/io.h>
4. #include
   <util/delay.h>
5.
6. int main(void)
7. {
8.
   DDRD=0b01000000;
9.     while (1)
10.    {
11.        PORTD |=
   (1<<6);
12.
   _delay_ms(1000);
13.        PORTD &=
   ~(1<<6);
14.
   _delay_ms(1000);
15.    }
16. }
```

3. Im unteren Teil des Fensters sollte nun die Ausgabe des Compilers sichtbar werden. Diese sollte ===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ===== lauten
4. Auswählen der hex-Datei
 1. im Atmel Studio finden Sie rechts im Fenster den "Solution Explorer"
 2. gehen Sie dort im Solution Explorer zu Solution » Einfuehrung_v01 » Output Files
 3. klicken Sie mit rechter Maustaste auf Einfuehrung_v01.hex und wählen Sie Pfad und Name aus

Fig. 1: einfache Diodenschaltung in SimulIDE



III. Ausführung in Simulide

1. Öffnen Sie SimulIDE (unter ...\\bin\\simulide.exe)
 1. links in SimulIDE sollten Sie den Komponenten Browser finden. Wählen Sie dort Micro»AVR»atmega»atmega328
 2. Ziehen Sie den Eintrag atmega328 per Drag and Drop in den Arbeitsbereich (rechter, beiger Teil des Fensters)
 3. Es sollte nun ein Chip names atmega328-1 dargestellt sein
2. Erstellen der Ausgangsschaltung
 1. Im Programm wurde im auf PortD das 6bit angesprochen. Entsprechend soll auch hier am Port D der Ausgang 6 genutzt werden. Am Chip ist dieser mit D6 gekennzeichnet
 2. Fügen Sie eine LED (im Komponenten Browser über Output LED) und ein Massepotential ein (Sources Ground)
 3. Die Komponenten können mit dem Kontextmenu (Rechtsklick) gedreht und gespiegelt werden. Außerdem ist mit der Auswahl von Properties im Kontextmenu die Änderung von
 4. Verbinden Sie die LED mit Masse und mit Port D6. Achten Sie auf die richtige Richtung der LED. Die Verbindungen lassen sich dadurch erstellen, dass auf ein Komponenten-Pin geklickt wird und die Linie zu einem nächsten Komponenten-Pin gezogen wird.
3. Flashen der Software
 1. Klicken Sie rechts auf den Microcontroller und wählen Sie Load firmware

2. Fügen Sie hier den Pfad und Name des oben erstellten Einfuehrung_v01.hex ein und öffnen Sie dieses
4. Starten der Simulation
 1. klicken Sie im Menu den Power-on Button
 2. Die Simulation startet
5. Bugfixing
 1. vermutlich ist bei Ihnen zu sehen, dass die Diode nicht gleichmäßig an und aus dargestellt wird. Dies ist kein Fehler des Simulationsprogramms. Es wurde noch eine wichtige Komponente vergessen, welche immer bei der Verwendung von diskreten LEDs verwendet werden muss. Fügen Sie diese ein und Testen Sie die Schaltung nochmal

Sie sollten sich nach der Übung die ersten Kenntnisse mit dem Umgang der Umgebung angeeignet haben. Zum Festigen des der Fähigkeiten bieten sich folgende Aufgaben an:

Aufgaben

1. Welche [Vorgaben für die Softwareentwicklung](#) wurden verletzt, trotzdem das Programm lauffähig ist? (Interrupts werden erst in späteren Übungen erklärt)
2. Wie könnte ein Ampel-Licht-Abfolge oder Lauflicht aus 4 Dioden erstellt und programmiert werden? Welche Optimierungen könnten im Code vorgenommen werden? Welche Komponente in SimulIDE kann genutzt werden? Wie kann die Farbe der LEDs geändert werden?
3. Lesen Sie auf Mikrocontroller.net im Kapitel [Warteschleifen](#) die "erste Seite", also bis:

Abhängig von der Version der Bibliothek verhalten sich die Bibliotheksfunktionen etwas unterschiedlich.

Weiterführende Fragen und Infos

Was hat es mit "Release" und "Debug" in der Standard-Toolbar auf sich?

Mittels dieser Auswahl lassen sich verschiedene Konfigurationen des Compilers setzen. Der

Compiler übersetzt den C-Code in maschinenlesbaren Code und wird bei Bedarf Hinweise/Warnungen/Fehler ausgeben, diverse Optimierungen vornehmen und die ausführbaren Dateien ("Executables") ablegen. Über eine Konfiguration kann damit folgende Dinge geändert werden:

- Kriterien für Hinweise/Warnungen/Fehler
- Optimierungen bei der Kompilierung
- Ablageort der Dateien

Was ist DDRD, PORTD?

Die Anschlüsse (Pins) des Chips sind in 8er Gruppen sortiert, den sogenannten Ports. Für jeden Port sind jeweils drei [Register](#) Bytes vorhanden: DDRx, PORTx und PINx. Diese Speicherstellen ermöglichen die Konfiguration des Ports. Die Bits in den Registern stehen für die einzelnen Anschlüsse: So ist zum Beispiel das 6. Bit in DDRD, PORTD und PIND für den Anschluss D6 zuständig.

Das Bit im **DDRx** (Data Direction Register) wählt die Richtung des Pins aus. Wenn dort logisch Eins geschrieben wird, wird der entsprechende Pin als Ausgangspin konfiguriert. Wenn dort logisch Null geschrieben wird, wird der entsprechende Pin als Eingangspin konfiguriert.

Das Bit im **PORTx** Register hat mehrere Eigenschaften: Wenn das gewünschte Bit in PORTx logisch eins geschrieben wird und der Pin als Ausgangspin konfiguriert ist, wird der Portpin auf high (eins) gesetzt. Wenn das gewünschte Bit in PORTx logisch Null geschrieben wird und der Pin als Ausgangspin konfiguriert ist, wird der Portpin auf Low (Null) getrieben.

Auch wenn ein Pin als Eingangspin konfiguriert wurde, hat PORTx eine Funktion. Wenn in diesem Fall das gewünschte Bit in PORTx logisch eins geschrieben wird, wird der Pull-up-Widerstand aktiviert. Ein Pull-up-Widerstand ist ein höherohmiger Widerstand (im Bereich $20\text{ k}\Omega$... $100\text{ k}\Omega$), der bei nicht weiter verbundenem Pin den ausgegebenen Wert auf logisch Eins zieht. Um den Pull-up-Widerstand auszuschalten, muss das gewünschte Bit in PORTx logisch Null geschrieben werden oder der Pin muss als

Ausgangspin konfiguriert werden.

Das Einlesen der Signale wird in einem späteren Kapitel erklärt.

Wie findet man die Namen der Anschlüsse?

Die Namen sind im Datenblatt des verwendeten Microcontrollers zu finden. Das lässt sich in diesem Fall in einer Suchmaschine über `atmega328 "datasheet" site:microchip.com filetype:pdf` finden, da es sich beim Datasheet um ein PDF des Herstellers Microchip handelt. Zum Lesen der Datenblätter empfiehlt sich ein Download und die Betrachtung über einen PDF-Viewer, welcher ein Inhaltsverzeichnis als Seitenleiste ermöglicht (z.B. Acrobat Reader). Ansonsten ist das Inhaltsverzeichnis häufig auch auf den hinteren Seiten des Datenblatts zu finden.

Die gesuchte Pinbelegung ist für den ATmega328P konkret auf Seite 3 unter "1. Pin Configurations":

Falls Sie mit dem englischen Datenblatt Probleme haben, kann ich die [deutsche Übersetzung des ATmega88](#) empfehlen, welcher sich im Wesentlichen ähnlich verhält. Beachten Sie aber, dass viele weitere Datenblätter nur in englisch vorhanden sind. Ein Umgang mit der englischen Dokumentation sollte also erlernt werden.

In Simulide fehlen die Anschlüsse für PORTA und weitere PINs. Außerdem kann man GND und VCC nicht verbinden.

Die ATmega Microcontroller basieren alle auf einer ähnlichen Struktur. Bei verschiedenen Varianten (insbesondere bei denen mit geringem "Pin Count") werden Anschlüsse zusammengefasst oder weggelassen. PORTA ist bei älteren Chips der Anschlussbereich für die Eingänge zum Analog-Digital-Wandler gewesen. Diese sind nun auf PORTC zu finden. Aus Kompatibilitätsgründen ist PORTA weggelassen worden. Ähnlich verhält es sich mit Pin PC7.

Die Anschlüsse GND (Masse) und VCC (Versorgung) sind in der Simulation automatisch verbunden. Der Microcontroller ist also stets betriebsbereit.

Äh.. wir haben 1000ms als Wartezeit eingegeben, aber die LED blinkt doppelt so schnell..

Richtig bemerkt. Das liegt daran, dass der verwendete Chip in Simulide auf 16 MHz läuft. Dies ist in der Simulation oben unter dem Start-Button zu sehen.

In Realität wird die Taktfrequenz durch die Randbedingungen wie Performance, Stromverbrauch, Spannungsversorgung oder Platinengröße vorgegeben. Die reale Hardware des ATmega 328 beinhaltet einen internen Taktgeber mit 8 MHz, welche durch einen Teiler auf 1 MHz reduziert sind. Über einen externen Quarz an den Pins PB6 und PB7 kann bei der realen Hardware (mit weiteren Einstellung der Fuses) ein externer Takt eingespeist werden. Häufig werden hierbei - neben ganzzahligen MHz - auch Vielfache von 256 genutzt, wie z.B. 12,288 MHz. Dies vereinfacht das exakte Abzählen von (Milli)Sekunden, da intern hierzu 8-Bit-Zähler genutzt werden können. Der Takt kann bei der realen Hardware in der Regel nicht zur Laufzeit beliebig geändert werden, sondern liegt fest vor.

Der Controller kann bei der Ausführung des Codes die Taktfrequenz nicht. Sollen genaue Bruchteile einer Sekunde erzeugt werden, muss dem Compiler die Frequenz über einen define mitgeteilt werden. Im Code geschieht dies über `#define F_CPU 8000000UL`

In der Simulation kann über rechte Maustaste auf den uC » Eigenschaften » MHz die Frequenz zwischen 1kHz und 100MHz beliebig geändert werden. Dies ist in Realität nicht möglich. Die höchste Frequenz ist laut Datenblatt 16MHz.

In diesem Fall wäre also die Lösung entweder den Wert von F_CPU oder die Frequenz in der Simulation anzupassen.

Wie ist die Ansteuerung von den Pins nun genau zu verstehen?

Was macht nun der

Unterschied zwischen `static const int var = 5;` `#define var 5` und `enum { var = 5 };` define für SET / CLR / TGL

Wie sähe der Code aus, wenn man sich an die Vorgaben für sauberen Code hielte?

Hierbei ist zu beachten, dass auch die Verwendung von delays vermieden werden soll. Diese sind ein "aktives Nichtstun" des Prozessors. Das heißt er kann in der Zeit nicht andere Aufgaben erledigen. Ein Multitasking ist damit nicht möglich. Da dies einer nachträglichen Erweiterung des Codes im Weg steht, sollten **generell keine delays** verwendet werden. Eine Ausnahme davon bildet Treiber-Code in welchem Microsekunden genaue Asgaben an einem Pin gemacht werden müssen.

[main.c](#)

```
1. #define F_CPU
   8000000UL
2. #define SET_BIT(PORT, BIT)
   ((PORT) |= (1 << (BIT))) //
   Port-Bit Setzen
3. #define TGL_BIT(PORT, BIT)
   ((PORT) ^= (1 << (BIT))) //
   Port-Bit Toggeln
4.
5. #define LED_WAIT_TIME
   1000 // Dauer bis zum
   Taktwechsel am Pin
6. #define LED_PIN
   6 // Pin an dem die LED
   anschlossen ist
7.
8. #include <avr/io.h>
9. #include <util/delay.h>
10.
11. int main(void)
12. {
13.     SET_BIT(DDRD, LED_PIN);
14.     while (1)
15.     {
16.         TGL_BIT(PORTD,
LED_PIN);
```

```
17.     _delay_ms(LED_WAIT_TIME);  
18.     }  
19. }
```

From:

<https://wiki.mexle.org/> - **MEXLE Wiki**

Permanent link:

https://wiki.mexle.org/microcontrollertechnik/1_hello_blinking_world?rev=1601573669

Last update: **2021/05/09 10:07**

