Das erste Projekt

Nachdem nun einige Grundeinstellungen[1] vorgenommen wurden, soll ein erstes sehr einfaches Projekt erstellt werden: Es soll über digitale Ein- und Ausgänge des Mikrocontrollers der Zustand eines Tasters eingelesen werden und anhand dessen eine LED ein- bzw. ausgeschaltet werden.

Hierzu ist es erforderlich, sich der Hardware des Mikrocontrollers bzw. des Arduino Boards etwas zu nähern. Keine Angst...er beißt nicht :-)

Hierzu ist es SEHR sinnvoll, sich die entsprechenden Unterlagen zu beschaffen und diese zu studieren. Für den ATmega 2560 sind dies erst mal die folgenden drei Informationsquellen:

1. Das Datenblatt zum Mikrocontroller
   Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

2. Der Befehlssatz des Mikrocontrollers
   Atmel-0856-AVR-Instruction-Set-Manual.pdf

3. Das Pinmapping für das verwendete Arduino Board mit dem ATmega 2560
   https://www.arduino.cc/en/Hacking/PinMapping2560

Die ersten beiden Dokumente sind derart umfangreich, dass man den „Wald vor lauter Bäumen nicht sieht"...daher nähern wir uns schrittweise!

Zentrale Bedeutung haben die sogenannten Register. Register sind prozessorinterne Speicherstellen, die die Werte aufnehmen, die die ALU verarbeiten soll. Der ATmega 2560 enthält 32 Register mit den Bezeichnung R0 bis R31 (siehe Kapitel „7.5 General Purpose Register File").

Für das erste oben beschriebene Projekt müssen digitale Signale eingelesen und ausgegeben werden. Mikrocontroller verfügen hierfür über sogenannte Ports. Schaut man nun im Datenblatt des ATmega 2560 in den Bereich „Overview", werden dort u.a. 10 sogenannte Ports mit jeweils acht Anschlüssen aufgeführt. Die Ports heißen Port A bis Port L (wobei es keinen Port I gibt).

Die jeweils acht Anschlüsse heißen dann beim Port A: PORTA 0 bis PORTA7 oder z.B. beim Port K eben PORTK0 bis PORTK7. Somit stehen allein mittels dieser zehn Ports insgesamt 80 universelle I/O-Anschlüsse zur Verfügung.
Eine kurze Beschreibung findet man im Kapitel „2.3 Pin Descriptions".

Eine genauere Beschreibung findet sich im Kapitel „13 I/O Ports". Hier wird der detaillierte Aufbau eines einzelnen I/O-Pins beschrieben. Nicht erschrecken :)

Betrachten wir das am Beispiel vom Port E, da wir diesen nachher zuerst nutzen werden.

Wie bereits beschrieben, besteht der Port E aus acht digitalen I/O-Pins.

Jeder dieser acht Pins kann einzeln entweder als Eingang oder als Ausgang konfiguriert werden.

Für jeden Port existieren drei Register mit jeweils acht Bit:

1. Das Data Direction Register; z.B. **DDRE** mit den Bits **DDE0** bis **DDE7**
2. Das Data Register; z.B. **PORTE** mit den Bits **PORTE0** bis **PORTE7**
3. Das Port Input Register; z.B. **PINE** mit den Bits **PINE0** bis **PINE7**

Hört sich erst mal kompliziert an, ist es aber nicht!

1 Sinnvollerweise sollte man in einer Netzwerkumgebung noch den Standard-Projektspeicherort ändern.

Data Direction Register

Mit dem „Data Direction Register“ (künftig: DDR) legt man fest, ob ein bestimmter Pin als Eingang oder als Ausgang dienen soll. Standardmäßig enthält das DDR den binären Wert 0 und somit sind alle acht Pins als Eingang konfiguriert. Will man nun einen Pin als Ausgang konfigurieren, muss an diese Bitposition im DDR der binäre Wert 1 geschrieben werden.

Beispiel

Das Data Direction Register E enthält den Wert 0x32. Dann haben die Bits **DDE1**, **DDE4** und **DDE5** den Wert 1 und die zugeordneten Pins können als Ausgang genutzt werden.
Die verbleibenden Pins sind als Eingang konfiguriert.

Data Direction Register	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
DDRE =	0	0	1	1	0	0	1	0

Data Register

Das Data Register enthält den Wert, der am Ausgang erscheinen soll. Relevant sind erst mal nur diejenigen Bits, die auch als Ausgangsbits konfiguriert sind!

Beispiel

Soll am Port E das Ausgangsbit **PORTE1** den Wert 1, das Ausgangsbit **PORTE4** den Wert 0 und das Ausgangsbit **PORTE5** wieder den Wert 1 haben, schreibt man eben diese Werte an die betreffenden Bitpositionen (an Stelle von z.B. **PORTE0** kann man auch **PE0** schreiben).

Data Register	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
PORTE =	-	-	1	0	-	-	1	-

Die fünf anderen Bits haben für die Ausgabe von binären Signalen keine Bedeutung!

Port Input Register

Dieses Register enthält die binären Werte, die als Eingangssignal an den betreffenden Pins anliegen. Da es sich letztendlich um Spannungspegel handelt und posotive Logik verwendet wird, entspricht ein Spannungspegel von ~0 V dem binären Wert 0 und ein Spannungspegel von ~5 V dem binären Wert 1.

Beispiel

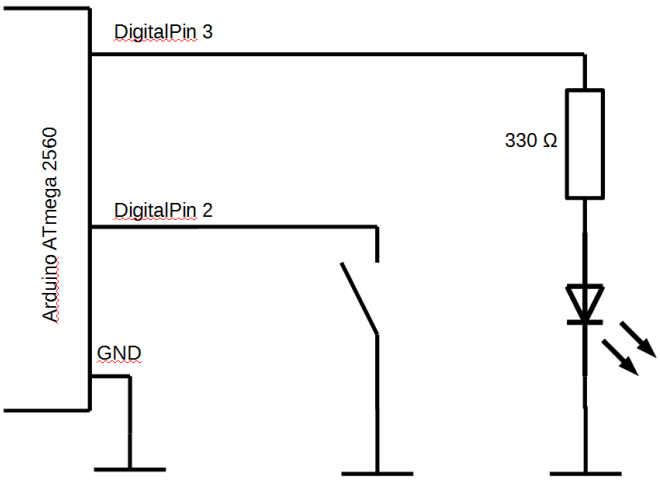
Die als Eingang konfigurierten Bits **PINE0**, **PINE2**, **PINE3**, **PINE6** und **PINE7** enthalten nun die binären Werte, die an den betreffenden Anschlüssen des ATmega 2560 als Spannungspegel vorhanden sind. Die drei als Ausgang konfigurierten Pins sind irrelevant.

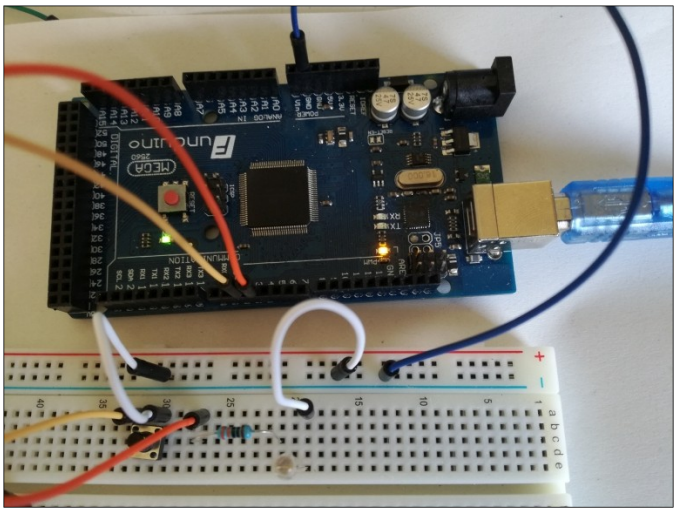
Port Input Register	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
PINE =	0	1	-	-	1	0	-	0

Hinsichtlich der Konfiguration eines Eingangs-Pins muss u.U. noch etwas optimiert werden. Dazu später mehr im Zusammenhang mit der Außenbeschaltung.

Vorgehensweise: Bevor ein bestimmter Pin als Ausgang genutzt werden kann, muss im jeweiligen DDR das betreffende Bit auf den binären Wert 1 gesetzt werden.

Die Schaltung für das erste Projekt





Aufgabenstellung

Mit Hilfe eines Taster, der am DigitalPin 2 angeschlossen ist, soll eine LED, die am DigitalPin 3 angeschlossen ist, eingeschaltet werden.

Die Pinnummern sind auf der Arduino-Platine aufgedruckt.

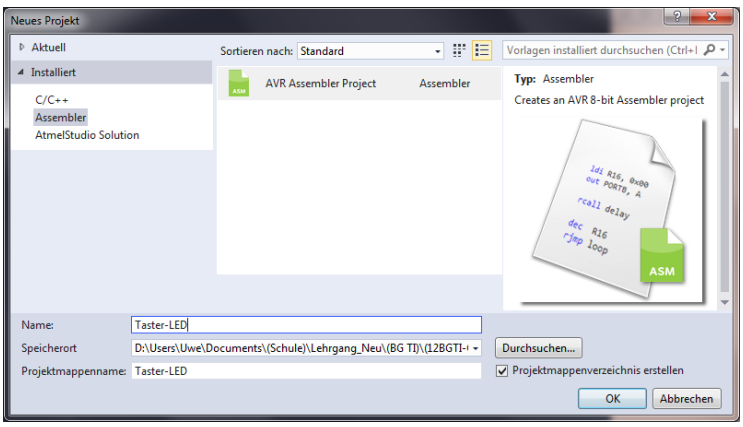
Der Taster verbindet den als Eingang konfigurierten Digitalpin 2 mit Ground (GND), also dem Spannungspegel 0 V. Der DigitalPin 3 ist als Ausgang konfiguriert und liefert bei dem binären Wert 1 einen Spannungspegel von ~5 V. Da eine LED (je nach Farbe) eine Flussspannung U_F von ~2 V hat und zum Leuchten ein Strom I_F von ~10 mA fließen soll, muss die LED mit einem Vorwiderstand betrieben werden. Genauere Werte für U_F und I_F muss man ggf. dem Datenblatt der LED entnehmen. Hier soll nun ein Vorwiderstand von 330 Ω verwendet werden. Der Spannungsabfall an der LED beträgt ca. 1,7 V bei einem Strom von ~10 mA.

Ein Ausgangspin kann maximal einen Strom von 40 mA liefern. Somit ist diese Dimensionierung unkritisch.

Genaueres zu den Spannungs- und Stromwerten lässt sich dem Datenblatt in den Kapiteln „31 Electrical Characteristics“ und „31.1 DC Characteristics“ entnehmen.

Der prinzipielle Aufbau der Schaltung lässt sich dem Foto entnehmen.

Nun soll programmiert werden :)



Erstellung des Projekts in Atmel Studio 7

Starten Sie die Entwicklungsumgebung Atmel Studio 7.

Wählen Sie im Menü „**Datei**“ den Menüpunkt „**Neu-Projekt...**“.

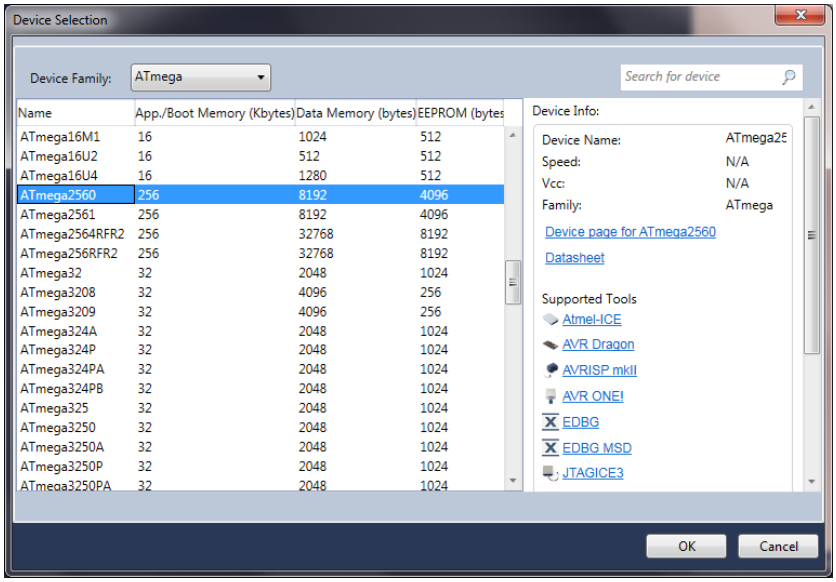
Es öffnet sich ein Assistent zur Auswahl eines Projekttyps.

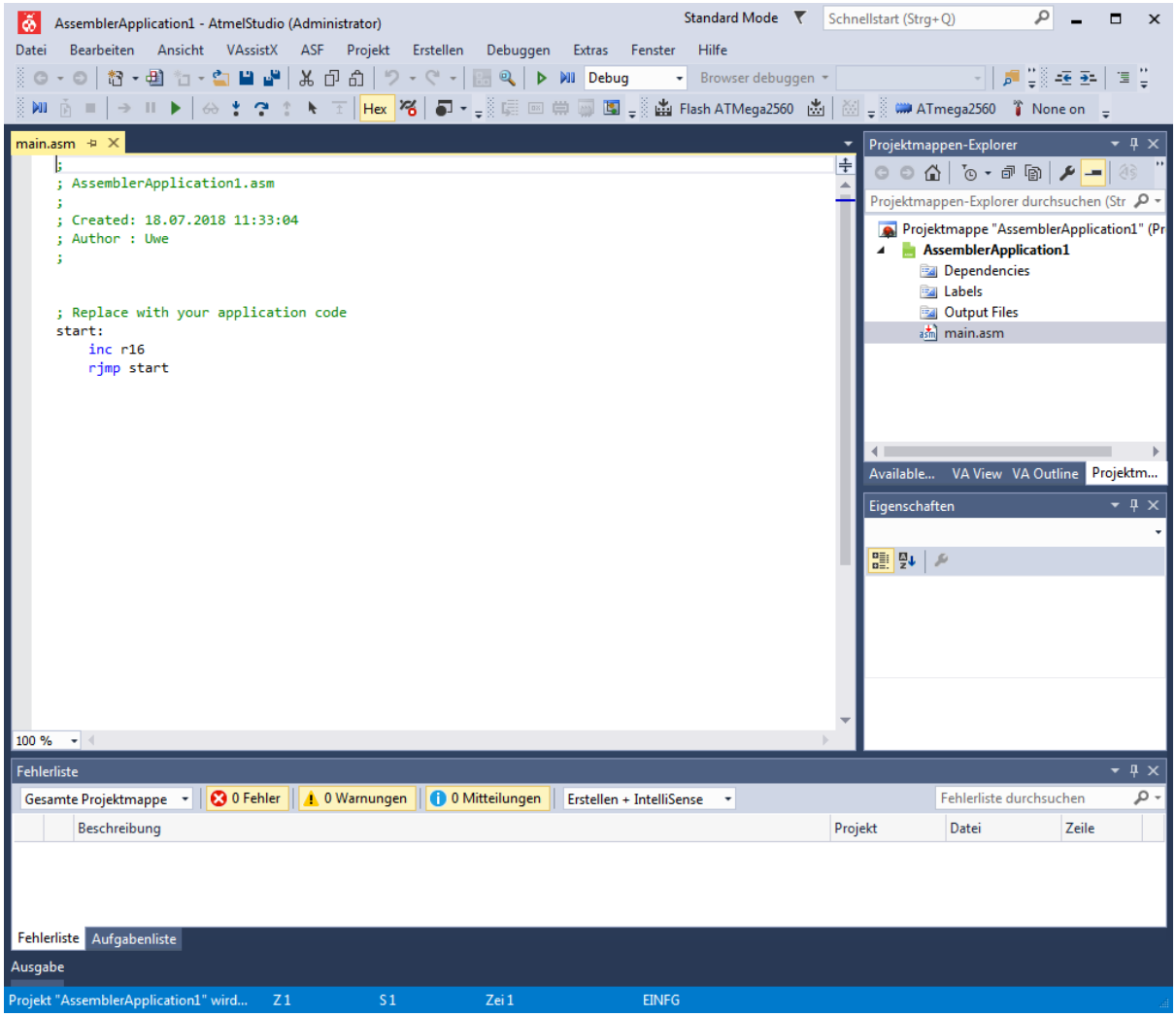
Wählen Sie links „Assembler“ und geben Sie einen Projektnamen (hier: „Taster-LED“) ein und bestätigen Sie mit „**OK**“.

Im nächsten Schritt muss der richtige Mikrocontroller ausgewählt werden. Dies ist der ATmega 2560.

Nach der Bestätigung der Auswahl wird nun ein Programm-Grundgerüst erstellt und es wird eine Datei „**`main.asm`**“ angelegt, in der das Assembler-Programm einzugeben ist.

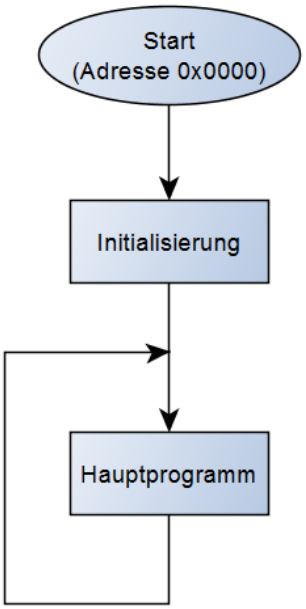
Das Ergebnis zeigt der folgende Screenshot.





Ganz prinzipiell besteht ein Assembler-Programm aus zwei Programmteilen:

1. einem Initialisierungsteil, in dem alle benötigte Hardware mit den erforderlichen Startwerten versehen wird
2. einer Endlos-Schleife, in der das Hauptprogramm ausgeführt wird



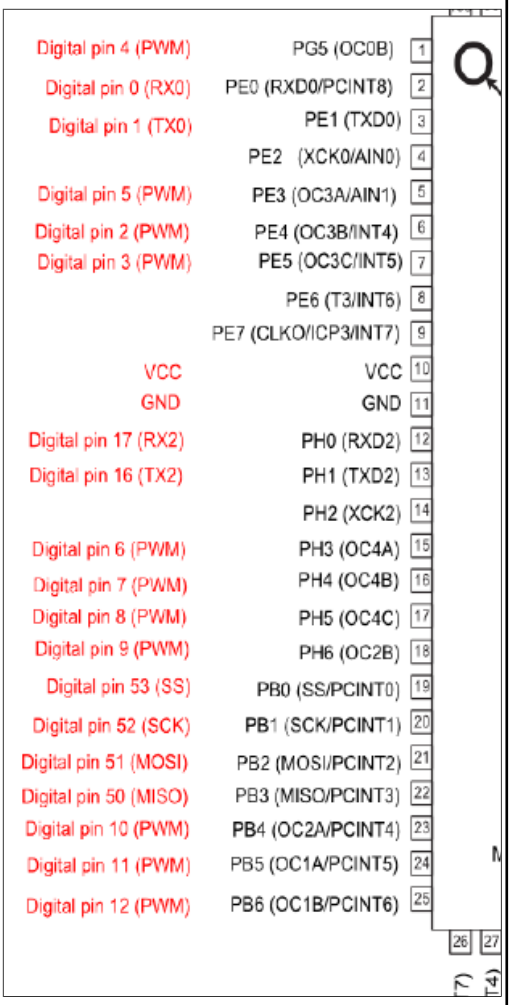
Der Programmstart erfolgt in der Regel an der Programm-Speicher-Adresse[2] **`0x0000`**. Diese Startadresse ist spezifisch für jeden Mikrocontroller und durch das Hardware-Design festgelegt.

Initialisierung für das Programm

Die Initialisierung ist für das erste Programm sehr übersichtlich. Prinzipiell sind ja alle Portbits als Eingang konfiguriert. Daher muss lediglich der DigitalPin, an dem die LED angeschlossen ist, als Ausgang konfiguriert werden.

Hierzu muss man jedoch wissen, mit welchem DigitalPin des Arduino-Boards das gewünschte Portbit auf der Platine per Leiterbahn verschaltet ist. Hierzu benötigt man nun die Beschreibung des eingangs erwähnten **Pinmapping**, das sich an der angegebenen Quelle befindet.

Deshalb ist rechts der relevante Ausschnitt aus der Quelle dargestellt. Die in schwarzer Schrift angegebenen Bezeichnungen sind die des ATmega 2560, wie man sie im Datenblatt findet. In roter Schrift sind die Pinbezeichnungen des Arduino-Boards.



Digital pin 4 (PWM)	PG5 (OC0B)	1
Digital pin 0 (RX0)	PE0 (RXD0/PCINT8)	2
Digital pin 1 (TX0)	PE1 (TXD0)	3
	PE2 (XCK0/AIN0)	4
Digital pin 5 (PWM)	PE3 (OC3A/AIN1)	5
Digital pin 2 (PWM)	PE4 (OC3B/INT4)	6
Digital pin 3 (PWM)	PE5 (OC3C/INT5)	7
	PE6 (T3/INT6)	8
	PE7 (CLKO/ICP3/INT7)	9
VCC	VCC	10
GND	GND	11
Digital pin 17 (RX2)	PH0 (RXD2)	12
Digital pin 16 (TX2)	PH1 (TXD2)	13
	PH2 (XCK2)	14
Digital pin 6 (PWM)	PH3 (OC4A)	15
Digital pin 7 (PWM)	PH4 (OC4B)	16
Digital pin 8 (PWM)	PH5 (OC4C)	17
Digital pin 9 (PWM)	PH6 (OC2B)	18
Digital pin 53 (SS)	PB0 (SS/PCINT0)	19
Digital pin 52 (SCK)	PB1 (SCK/PCINT1)	20
Digital pin 51 (MOSI)	PB2 (MOSI/PCINT2)	21
Digital pin 50 (MISO)	PB3 (MISO/PCINT3)	22
Digital pin 10 (PWM)	PB4 (OC2A/PCINT4)	23
Digital pin 11 (PWM)	PB5 (OC1A/PCINT5)	24
Digital pin 12 (PWM)	PB6 (OC1B/PCINT6)	25
		26 27

Der DigitalPin 2 des Arduino-Boards ist mit dem Bit **`PE4`** bzw. **`PORTE4`** verbunden, der DigitalPin 3 hingegen mit dem Bit **`PE5`** bzw. **`PORTE5`**.

Da die LED am DigitalPin 3 angeschlossen werden soll, muss nun das betreffende Bit **`PORTE5`** gesetzt werden. Das Setzen eines Bits erfolgt mit dem Befehl „**`sbi`**“. Das steht für „Set Bit in I/O Register“. Der Befehl muss wissen, welches Bit in welchem I/O-Register gemeint ist. Für dieses Beispiel ist es das I/O-Register **`DDRE`** und darin das Bit **`DDE5`**.

```
; Startadresse für das Programm
.org 0x0000

; Setze das Bit DDE5 als Ausgang
     sbi  DDRE, DDE5
```

Die **`ORG`**-Anweisung ist **KEIN** Assembler-Befehl aus dem Befehlsvorrat des Mikrocontrollers, sondern eine Anweisung[3] an

2 Die Werte von Adressen und Daten werden typischerweise in hexadezimaler Form angegeben.

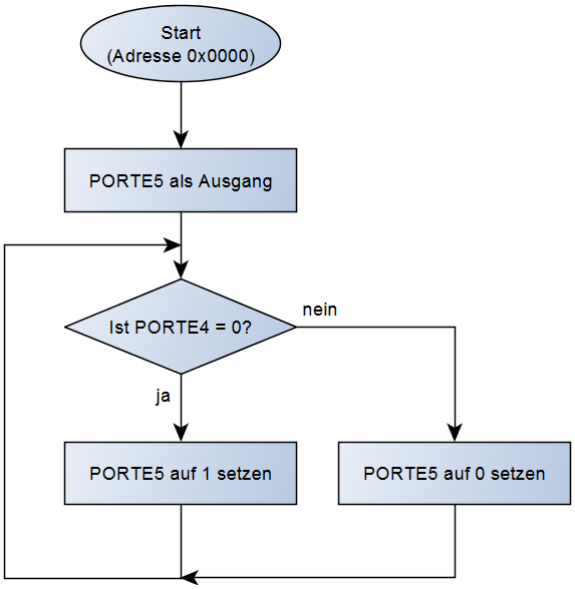
3 Man sagt auch Assembler-Direktive

den Assembler (das Programm, das die Assembler-Befehle in die Maschinensprache überführt). Hierdurch soll das Programm ab der Startadresse **`0x0000`** liegen. Dies wird dann wichtig, wenn im Programm Verzweigungen enthalten sind, um dann die Adresse für eine Verzweigung (mittels Sprung-Befehl) zu berechnen. Dazu später mehr.

Da nach einem Reset des Mikrocontrollers alle Register und I/O-Ports den Bitwert 0 enthalten, ist auch der Wert von **`PORTE5`** (der bestimmt, ob die LED leuchtet [1] oder nicht [0]) auf 0. Die LED leuchtet also nicht. Hierfür ist also nichts zu tun.

Okay...das war doch einfach, oder? Jetzt kommen wir zum Hauptprogramm!

Im Hauptprogramm muss nun im Rahmen einer Endlosschleife überprüft werden, ob der Taster gedrückt (**`PORTE4`** = 0) oder nicht gedrückt (**`PORTE4`** = 1) ist.



Hierzu muss also der Wert vom Bit **`PORTE4`** ermittelt werden. Je nach dem, muss eine Verzweigung in unserem Programm erfolgen!

Das Testen eines Bits kann mit dem Befehl **`sbis`** bzw. **`sbic`** erfolgen.

Diese Mnemonics stehen für „Skip if Bit in I/O Register is Set“ bzw. „Skip if Bit in I/O Register is Cleared“.

Da ja auf den Bitwert 0 in **`PORTE4`** getestet werden soll, ist der Befehl **`sbic`** etwas besser geeignet.

Wenn diese Überprüfung erfolgreich ist (d.h. der Taster wurde gedrückt), wird der nachfolgende Befehl hinter **`sbic`** ausgelassen.

Um an eine andere Stelle in einem Programm zu verzweigen, gibt es sogenannte Sprungbefehle. Der einfachste Sprungbefehl ist ein sogenannter „unbedingter Sprung“.

Das zugehörige Mnemonic lautet „**`jmp`**“ und dahinter wird das Ziel des Sprunges, d.h. die Stelle an der das Programm fortgesetzt werden soll, angegeben.

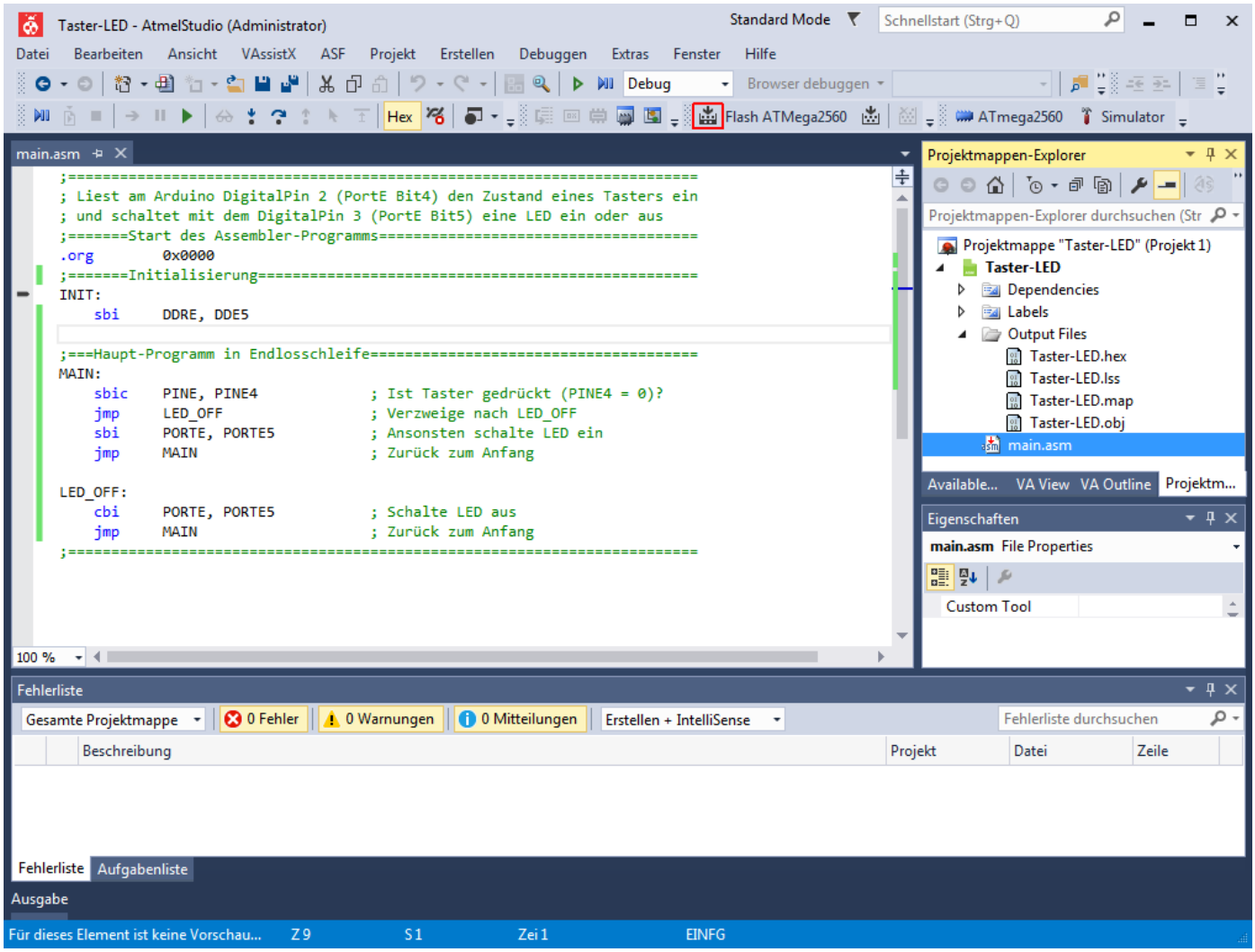
Ein sogenanntes „Spungziel“ ist einfach eine Bezeichnung, die von einem Doppelpunkt gefolgt ist.

Um ein einzelnes Bit zurück zu setzen, gibt es einen Befehl mit dem Mnemonic „**`cbi`**“ für „Clear Bit in I/O Register“. Auch dieser Befehl benötigt als Operand das gewünschte I/O-Register un darin das gewünschte Bit.

Somit haben wir nun alles notwendige für das Hauptprogramm zusam-

```
; Startadresse für das Programm
.org 0x0000

; Setze das Bit DDE5 als Ausgang
     sbi  DDRE, DDE5

MAIN:
; Ist der Taster gedrückt?
     sbic PINE, PINE4
     jmp  LED_OFF

; Schalte LED ein
     sbi  PORTE, PORTE5
     jmp  MAIN

LED_OFF:
; Schalte LED aus
     cbi  PORTE, PORTE5
     jmp  MAIN
```

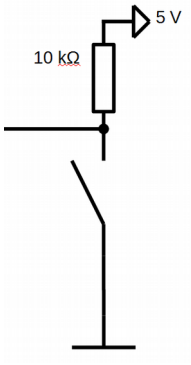
men. Nachfolgend finden Sie das Programm mit Kommentaren dargestellt.



Und nun die Realität...

Wenn nun das Programm vollständig eingegeben wurde, muss es in Maschinensprache überführt werden. Dies erfolgt mit dem Menüpunkt „Taster-LED erstellen“ im Menü „Erstellen“ oder eben durch einen Klick auf das rot umrandete Symbol in der Symbolleiste.

Anschließend muss das Maschinenprogramm auf das Arduino-Board übertragen werden. Dies erfolgt durch einen Klick auf den selbst definierten Befehl „Flash ATmega2560“ in der gleichen Symbolleiste.



Und nun? Komisch...warum leuchtet die LED bereits? Drückt man nun auf den Taster, leuchtet die LED etwas heller als zuvor. Was stimmt da nicht?

Die Erklärung liegt in der Hardware begründet! Da der Eingang ohne gedrückten Taster kein definiertes Potenzial hat und quasi „in der Luft hängt“, wird in einigen Fällen beim Überprüfen des Bits **`PINE5`** eine 0 und in den anderen Fällen eine 1 gelesen. Aufgrund der Ablaufgeschwindigkeit des Programms wird die LED nun mal ein- und mal ausgeschaltet. Dies wirkt wie ein etwas „dunkleres“ Leuchten.

Abhilfe schafft hier, in dem man den Eingang für den Zustand „nicht gedrückt“ auf ein definiertes HIGH-Potenzial legt, in dem man vom Eingang einen Widerstand

mit ~10...100 kΩ gegen die Versorgungsspannung 5 V schaltet.

Man nennt solch einen Widerstand auch „Pull-Up-Widerstand“, da er das Potenzial am Eingangspin gegen das Maximum der Versorgungsspannung zieht. Durch das Drücken des Tasters wird der Eingang mit GND verbunden. So ist gewährleistet, dass für beide Taster-Zustände immer ein definiertes Potenzial am Eingang herrscht.

Ergänzen Sie daher diesen Widerstand auf dem Breadboard und verbinden Sie die „+ Schiene“ mit dem 5V Pin auf dem Arduino. Jetzt sollte die Schaltung funktionieren!

Und jetzt die „gute Nachricht“ :)

Mikrocontroller-intern ist ein solcher Pull-Up-Widerstand bereits eingebaut. Allerdings muss dieser Widerstand auch aktiviert werden.

Hierzu wird im Data Register des gewünschten Ports das betreffende Bit auf den binären Wert 1 gesetzt. Das Data Register enthält also nicht nur für Ausgänge den Wert, der am Ausgang erscheinen soll, sondern aktiviert für Eingänge den internen Pull-Up-Widerstand.

Beispiel

In unserem TASTER-LED-Programm wird das Bit 4 als Eingang benutzt. Demzufolge hat dieses Bit im Data Register eigentlich keine Bedeutung. Setzt man jedoch das Bit **PORTE4** im Register **PORTE** auf den binären Wert 1, wird der interne Pull-Up-Widerstand für das Eingangsbit **PINE4** aktiviert und ein externer Pull-Up-Widerstand ist nicht mehr erforderlich.

Data Register	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
PORTE =	-	-	-	1	-	-	-	-

Erreichen lässt sich dies im Initialisierungsteil durch den Befehl

```
sbi   PORTE, PORTE4
```

Übung

1. Modifizieren Sie das Assembler-Programm so, dass der Taster den Zustand der LED durch jeden Druck wechselt.

   LED leuchtet nicht --> Taster drücken --> LED leuchtet --> Taster drücken --> LED leuchtet nicht....

2. Erstellen Sie hierzu einen Programmablaufplan.

3. Erstellen Sie ein neues Projekt TASTER-LED_V2. Übertragen Sie das Programm auf das Arduino-Board und testen Sie es.

   Das Programm wird wahrscheinlich nicht einwandfrei funktionieren.
   Informieren Sie sich, woran dies liegen kann![4]

4 **https://de.wikipedia.org/wiki/Prellen**