



Übungsblatt 3 (praktisch)

Einführung Assemblerprogrammierung ATmega16

Vorlesung: Technische Grundlagen der Informatik 1, Sommersemester 2020

Dozent: Dr.-Ing. Kristian Ehlers

Ziele und Umfeld des Versuchs

Das Ziel dieses Versuchs ist die Implementierung eines Steuerverhaltens auf einem ATmega16 Mikrocontroller. Dafür werden zuerst eine Einleitung und Erklärung in die Thematik gegeben, um Ihnen eine Hilfestellung für die Bearbeitung bereitzustellen. Die genaue Versuchsspezifikation folgt später.

Die Realisierung von Schaltwerken für verschiedene Arten von Steuerungssystemen kann für aufwendige Systeme schnell komplex und unübersichtlich werden. Weiterhin haben festverdrahtete Steuerungen den Nachteil, dass Erweiterungen und Modifikationen oft nur mit sehr hohem Aufwand möglich sind und eine komplette Überarbeitung des Entwurfs nötig machen.

Sofern es vom Zeitverhalten her möglich ist, werden komplexe Steuerungen aus diesen Gründen auf programmierbarer Hardware durch Programme realisiert, die einfacher zu testen, zu erweitern und zu modifizieren sind. Hier bieten sich Mikrocontrollerlösungen an, die durch geringen Bauteil Aufwand gekennzeichnet sind und eine schnelle Anbindung weiterer Peripherie wie digitale Ein-/Ausgabeports oder Analog-/Digitalwandler möglich machen.

Mikrocontroller (MCU: Micro Controller Unit) sind sehr leistungsstarke, frei programmierbare Bauelemente. Sie enthalten einen Mikroprozessor und integrierte Peripheriebaugruppen, wie z.B. Ein-/Ausgabeports, serielle und parallele Schnittstellen, verschiedene Timer und Zähler und A/D-Wandler, über die die MCU mit der Prozessumwelt kommunizieren kann. Zudem ist in den meisten MCUs direkt RAM (Random Access Memory) und ROM (Read Only Memory) / EEPROM (Electrically Erasable Programmable Read Only Memory)-Speicher integriert, dessen Speicherumfang je nach Variante der MCU von einigen hundert Bytes bis zu einigen Kilobytes reicht. Der interne Speicher ist für viele Anwendungsfälle ausreichend, so dass neben der MCU nur noch wenige Bauteile wie Kondensatoren, Widerstände und Quarze für eine vollständige Applikation erforderlich sind. Typische Einsatzgebiete für Single-Chip-Lösungen sind Massenprodukte wie zum Beispiel Waschmaschinen und Kameras.

Die Hardware des ATmega16

Der ATmega16 (siehe Abbildung 1) gehört zur Familie der AVR Mikrocontroller des Herstellers Atmel. Seine Stromaufnahme ist sehr gering, was seinen Einsatz z.B. auch in batteriebetriebenen Geräten ermöglicht. Das Programm des Mikrocontrollers wird im sogenannten Flash-ROM (16 KB) gespeichert. Ein weiterer Speicher nennt sich EEPROM (512 bytes). Dabei handelt es sich um einen beim Ausschalten der Versorgungsspannung nicht-flüchtigen Speicherbereich, der durch eine interne Logik beschrieben werden kann. Hier lassen sich z.B. Daten halten, die bei jedem Einschalten vorliegen sollten, wie etwa der Zuordnung zwischen Programmtasten und Senderkanälen bei einem Fernsehgerät. Zur weiteren Aufnahme von Variablen während der Programmabarbeitung und als Stapelspeicher (Stack) dient der SRAM Speicher, der beim ATmega16 1KB umfasst.

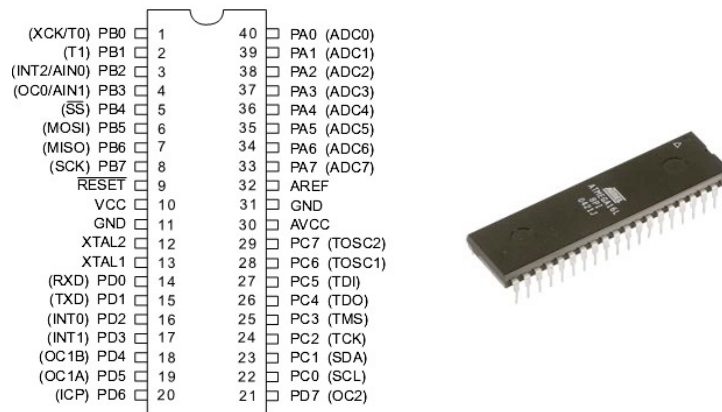


Abbildung 1: Die Pinbelegung des verwendeten ATmega16 Mikrocontrollers.

Die Register

Der ATmega16 verfügt wie alle AVR Controller über 32 Register R0 bis R31 von jeweils 8 Bit Breite. Die Register R26 bis R31 stehen weiterhin paarweise zusammengefasst als Register X, Y und Z für bestimmte Adressierungsarten zur Verfügung.

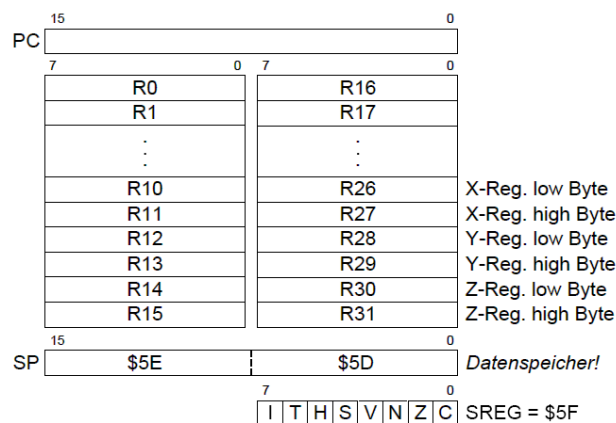


Abbildung 2: Die verschiedenen zur Verfügung stehenden Register des ATmega16.

Universalregister R0 bis R31

Die meisten datenverarbeitenden Anweisungen können auf alle 32 Register direkt zugreifen. In wenigen Ausnahmefällen kann nur ein Teil der Register als Operand verwendet werden (z.B. `ldi` = Load Immediate - kann nur auf die Register R16 bis R31 zugreifen).

Indexregister (X, Y und Z)

Bestimmte Adressierungsarten nutzen die 16-Bit Register X, Y und Z als Indexregister. So kann indirekt bzw. indiziert auf den Datenspeicher zugegriffen werden.

Program Counter (PC)

Dieses Register enthält die Adresse des nächsten abzuarbeitenden Befehls.

Stackpointer (SPH, SPL)

Der ATmega16 unterstützt einen Stapelspeicher, wobei der Stackpointer auf die nächste freie Adresse weist. Die Register SPL und SPH enthalten die Adresse des Stacks im Datenspeicher (High- und Low-Byte). Zu Beginn eines jeden Programms muss der Programmierer diese Register und damit die Lage des Stacks im Arbeitsspeicher initialisieren, danach organisiert die CPU den Stack selbständig. Der Stack kann unterschiedlichen Zwecken dienen:

- Sicherung des aktuellen Wertes des Programmcounters
- Parameterübergabe zwischen Unterprogrammen
- Inhalte von Registern, die kurzfristig anderweitig benötigt werden zwischenspeichern

Status Register (SREG)

Bit 7 – I: Global Interrupt Enable

Muss gesetzt sein, um Interrupts zu erlauben bzw. sperrt Interrupts, wenn gelöscht. Einzelne Interruptquellen können über separate Kontrollregister ein- und ausgeschaltet werden.

Bit 6 – T: Bit Copy Storage

Die Befehle BLD und BST nutzen dieses Statusregisterbit, um ein einzelnes Bit aus einem beliebigen Register zwischenzuspeichern.

Bit 5 – H: Half Carry Flag

Übertrag von Bit 3 nach einer Addition (wird für BCD-Arithmetik benötigt).

Bit 4 – S: Sign Bit, $S = N \oplus V$

Vorzeichenbit. Das S-bit entspricht der Exklusiv-Oder-Verknüpfung von N-Flag und V-Flag

Bit 3 – V: Two's Complement Overflow Flag

Überlauf des gültigen Zahlenbereichs (Zweierkomplement).

Bit 2 – N: Negative Flag

Kopie von Bit 7 des Ergebnisses der letzten arithmetischen/logischen Operation (Vorzeichenbit), nicht aber bei Ladebefehlen etc.

Bit 1 – Z: Zero Flag

Wird gesetzt, wenn Ergebnis der letzten arithmetischen/logischen Operation Null ist, nicht aber bei Ladebefehlen etc.

Bit 0 – C: Carry Flag

Übertrag bei Addition oder 'Borgen' bei Subtraktion, Kopie von Bit 15 (MSB) bei Multiplikation.

Die verschiedenen Schnittstellen

Zur Kommunikation mit der Außenwelt verfügt der ATmega16 über 4 parallele Schnittstellen (Port A bis D). Wie alle integrierten Peripheriebausteine werden auch die Ports über Register konfiguriert. Diese Register sind wie die Universalregister R0 bis R31 alternativ über bestimmte Adressen im Hauptspeicher ansprechbar. Die I/O-Register sind zusätzlich für die Befehle „in“ und „out“ unter einer weiteren, anderen Adresse in einem speziellen I/O-Bereich adressierbar. Diese I/O-Adressen werden in der hier verwendeten Definitionsdatei verwendet. Nutzen Sie daher bitte auch die Befehle „in“ und „out“!

Für jeden Port existieren die Register PORTx, PINx und DDRx. Letzteres dient der Konfiguration des Ports als Ein- bzw. Ausgang. Ist ein I/O-Port als Ausgang konfiguriert, so wird der Inhalt des entsprechenden Registers PORTx am korrespondierenden I/O-Pin des Bausteins ausgegeben. Aus den Registern PINx kann umgekehrt der Zustand der I/O-Pins eingelesen werden.

Zusätzlich sind diverse Pins für spezielle Funktionen reserviert. So können einzelne Pins als serielle Schnittstelle eingerichtet werden, oder auch als Anschlussmöglichkeit für elektronischen Bussystemen (bspw. I²C). Die Pins des Port A können als Analogeingänge genutzt werden, um eine angelegte analoge Spannung mit dem Mikrocontroller auszuwerten.

Beispiel zur Programmierung der parallelen I/O-Ports

Um die Vorgehensweise bei der Programmierung der Ports zu verdeutlichen, folgt hier ein kleines Beispiel. Der Controller soll dabei einen Byte-Wert von PortB einlesen, ihn invertieren und auf PortA wieder ausgeben.

```
main:
; configure PORTA
ldi    R16, 0xFF          ; alternativ auch -ser R16-
out    DDRA, R16          ; entspricht Ausgabeport
; configure PORTB
ldi    R16, 0x00          ; alternativ auch -clr R16-
out    DDRB, R16          ; entspricht Eingabeport
loop:
; read PORTB
in     R16, PINB
ldi    R17, 0xFF
eor    R16, R17
; write PORTA
out    PORTA, R16
jmp    loop
```

Vorbereitungsaufgaben

Die folgenden Vorbereitungsaufgaben dienen dazu, die Aufgabe genau zu spezifizieren und Ihnen eine Schritt-für-Schritt Anleitung zur Lösung des Problems zu geben.

Simulieren und testen Sie den Versuch zu Hause mit Hilfe [des ITmega16-Simulators](#) vollständig und halten Sie zum Besprechungstermin mit Ihrem Tutor die Lösung im Simulator zur Besprechung und Vorführung bereit.

Eine Einführung in die Simulationsumgebung wurde in der Vorlesung gegeben und kann bei Bedarf im Mitschnitt nochmal nachverfolgt werden.

Die Konfigurationsdatei für die Peripherie `laulicht_layout.js` ist im Moodle verfügbar. Implementieren Sie das Verhalten in der `laulicht_main.asm`, die ebenfalls im Moodle bereitgestellt wird.

Aufgabenstellung

Das Ziel dieses Versuchs ist die Implementierung eines einfachen Steuerprogramms mit einem ATmega16 Mikrocontroller. Dabei wird das Verhalten eines sogenannten Lauflichts implementiert.

Für den Versuch werden an den Mikrocontroller 12 LEDs angeschlossen, die als Lauflicht betrieben werden. Der Ablauf des Verhaltens kann umgangssprachlich folgendermaßen beschrieben werden:

Die LEDs befinden sich in der bekannten Versuchsbox in einer Reihe. Zum Zeitpunkt t_1 wird die erste LED eingeschaltet – alle anderen LEDs sind aus. Bei jedem nachfolgenden Zeitpunkt t_i wird die aktuell leuchtende LED ausgeschaltet und die lokal benachbarte LED angeschaltet. Ist das Ende des sogenannten LED-Arrays erreicht, beginnt das Verhalten von vorne. Dieses Gesamtverhalten wird periodisch wiederholt und man nennt die wandernde LED Lauflicht. Damit entsteht für den Betrachter der Eindruck, dass sich die Lichtquelle selbst bewegt. Dieses Verhalten ist in Abbildung 3 illustriert.

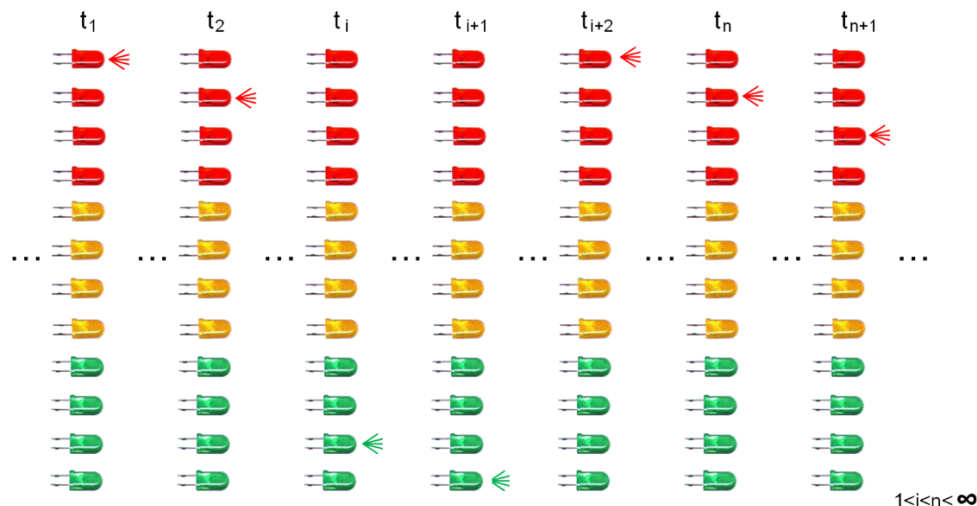


Abbildung 3: Grafische Darstellung eines Lauflichts. Zu jedem Zeitpunkt t_i wird die leuchtende LED um eine Position weiter geschaltet. Am Ende angekommen wird wieder von vorne begonnen.

Die gegebene Aufgabe der Implementierung eines Lauflichts wird um zwei weitere Teilaufgaben erweitert.

1. Die Geschwindigkeit des Lauflichts soll von außerhalb durch den Benutzer gesteuert werden. Dafür werden zwei Taster verwendet, mit deren Hilfe die Zeitabschnitte zwischen dem Wechseln der LEDs geändert werden können. Ein Taster dient zum Vergrößern der Wartezeit zwischen dem Umschalten zweier LEDs und ein Taster dient zu deren Verkleinerung.
2. Die aktuelle Wartezeit zwischen dem Umschalten zweier LEDs soll auf einer zweistelligen 7-Segment-Anzeige als hexadezimale Zahl ausgegeben werden.

Portbelegung

Vor der Erstellung des Programms sind die erforderlichen Schnittstellen zur Kommunikation mit der Prozessumwelt zu spezifizieren. Bei diesem Versuch sind folgende Ports des ATmega16 belegt. Das Anschließen der „Hardware“ in der Simulation wird Ihnen durch die `lauflicht_layout.js` Datei abgenommen. Sie müssen lediglich die Konfiguration der Ein- und Ausgänge des ATmega vornehmen.

ATmega I/O Port	I/O Richtung	Anschluss
PB7-PB0	Ausgang	Anzeigen des Warte-Faktors
PA0	Ausgang	LED Rot (1)
PA1	Ausgang	LED Rot (2)
PA2	Ausgang	LED Rot (3)
PA3	Ausgang	LED Rot (4)
PA4	Ausgang	LED Gelb (1)
PA5	Ausgang	LED Gelb (2)
PA6	Ausgang	LED Gelb (3)
PA7	Ausgang	LED Gelb (4)
PD0	Ausgang	LED Grün (1)
PD1	Ausgang	LED Grün (2)
PD2	Ausgang	LED Grün (3)
PD3	Ausgang	LED Grün (4)
PD4	Eingang	Taster (Wartezeit vergrößern)
PD5	Eingang	Taster (Wartezeit verkleinern)

Implementierung

Beginnen Sie zur Implementierung des beschriebenen Verhaltens mit der Definition von Variablennamen: Das Register `R16` soll wie üblich unter dem Namen `Temp` verwendet werden können. In den Registern `R20` und `R21` sollen die Ausgabewerte für die LEDs unter den Namen `LEDlow` und `LEDhigh` gespeichert werden. Das Register `R22` dient zur Speicherung der aktuellen Wartezeit unter dem Namen `Speed`. Definieren sie den symbolischen Namen für die Register an der entsprechen Stelle. Eigene von Ihnen gewählte Definitionen erfolgen bitte im selben Abschnitt.

Als nächstes erfolgt die Initialisierung der verschiedenen Ein- und Ausgabeports. Führen Sie die Initialisierungen entsprechend der obigen Tabelle durch. Jeder Ausgabe-Port soll als initialen Status den Wert `0x00` bekommen.

Weiterhin soll das Register `LEDlow` den initialen Wert 1 haben, damit die erste LED leuchtet, das Register `LEDhigh` demensprechend den Initialwert 0 und das Wartezeit `Speed` soll initial den Wert `0x50` enthalten.

Die Implementierung des Hauptptogramms (`main`) erfolgt in mehreren Teilschritten, die Sie nacheinander abarbeiten sollten:

1. Im ersten Schritt soll das Weiterschalten der leuchtenden LED realisiert werden. Dieses kann am einfachsten durch eine Shift-Operation erreicht werden. Bedenken Sie, dass immer nur eine LED zur Zeit leuchten soll, und das Sie nach der achten LED vom `PortA` auf den `PortD` wechseln müssen. Weiterhin müssen Sie nach dem Leuchten der letzten LED direkt wieder bei der ersten LED beginnen.
2. Der nächste Schritt ist die Programmierung einer blockierenden Warteschleife. Dies geschieht in Assembler mit ineinander geschachtelten Schleifen. Mit der Kenntnis der CPU Taktfrequenz (hier: 1MHz) und der Ausführzeit der einzelnen Befehle in Takten, kann eine ungefähre Wartezeit in Millisekunden abgeschätzt werden. Orientieren Sie sich bei der Implementierung an dem Beispiel der Vorlesung. Adaptieren Sie die äußere Schleife in der Art, dass diese von `Speed` bis 0 zählt und folglich einem Multiplikationsfaktor der dort festgelegten Wartezeit der inneren Schleife entspricht.

3. Im letzten Schritt erfolgt die Implementieren der Abfrage der beiden Tasterbausteine, um erkennen zu können, ob die Wartezeit zwischen dem Umschalten der LEDs vergrößert oder verkleinert werden muss. Überprüfen sie dafür die Zustände der beiden Eingabepins `PD4` (verkleinern) und `PD5` (vergrößern). Sollte einer der Taster gedrückt sein, verändern Sie die Wartezeit im Register `Speed` dementsprechend um den Wert ± 5 . Anschließend geben Sie die Wartezeit über den `PortB` aus, sodass die aktuelle Zeit auf der 7-Segment-Anzeige dargestellt werden kann.

Die nachfolgenden Ausführungen dienen lediglich der Information und müssen im Rahmen dieser Praktischen Übung bei der Bearbeitung nicht berücksichtigt werden. Es erfolgt eine Einführung in die Hard- und Softwareumgebung des Labors sowie die Durchführungsbeschreibung im Labor. Diese Information sollen das Verständnis für das spätere Tutorial zu dieser Übung fördern und den Hardwarebezug in der Laborumgebung aufrechterhalten.

Das verwendete Experimentiersystem

Für die Durchführung des Versuchs sind 5 Bausteine auf dem bekannten Steckbrett zu platzieren. Zu den Bausteinen gehören:

- Ein ATmega16
- Zwei Tasterbausteine
- Ein LED-Baustein
- Und eine zweistellige 7-Segmentanzeige

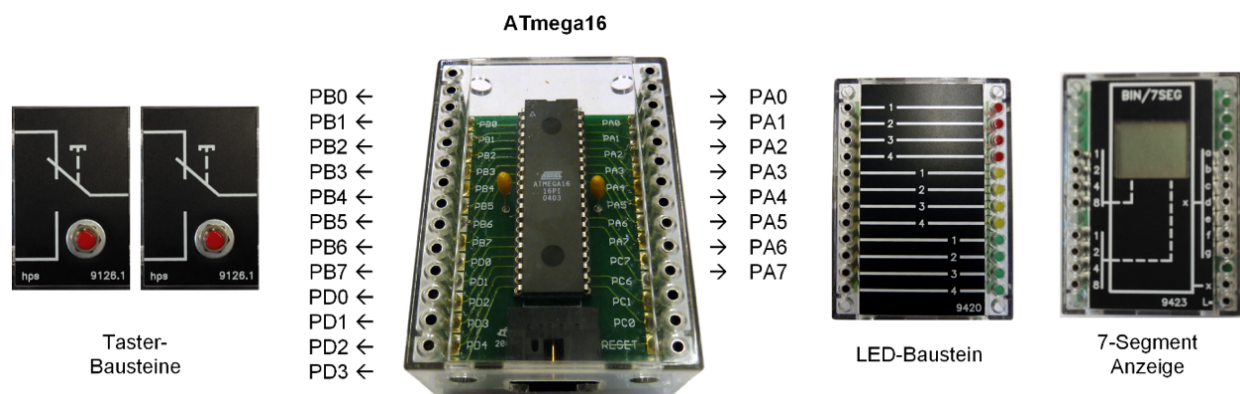


Abbildung 4: Das verwendete Experimentiersystem

Der in diesem Versuch verwendete ATmega16 ist in einem kleinen Kunststoffgehäuse eingebaut (siehe Abbildung 4). Die Belegung dieser Box ist dem Platinaufdruck zu entnehmen. Der ATmega16 wird darüber hinaus über ein Programmiergerät mit dem PC verbunden, damit Sie Ihr assembliertes Programm als .hex Datei in den Flash-Speicher des ATmegas kopieren können. Dieses Programm wird sofort nach dem Anlegen einer Versorgungsspannung am Mikrocontroller gestartet. Als Programmiergerät verwenden Sie in diesem Praktikum die sogenannten AVR Dragon Programmierer (siehe Abbildung 5). Dieser wird unterhalb des ATmegas an der Box angeschlossen und programmiert diesen über die sogenannte JTAG Schnittstelle.



Abbildung 5: AVR Dragon Programmiergerät

Die Entwicklungsumgebung Atmel Studio 7

Die in diesem Praktikum verwendete Entwicklungsumgebung für den ATmega16 stellt im Wesentlichen vier Funktionen zur Verfügung, die Ihnen das Schreiben und Austesten von Programmen für den Mikrocontroller ermöglichen sollen:

- Einen Editor, der zum Eingeben und Bearbeiten des zu erstellenden Programms dient.
- Den Assembler, der Ihr Assemblerprogramm in Maschinensprache übersetzt.
- Einen Simulator, mit dem Sie erstellte Programme vor dem Übertragen und Starten auf dem ATmega16 austesten können.
- Eine Schnittstelle zum Programmiergerät, mit dem Sie Ihr Programm auf den Mikrocontroller übertragen können.

Alle Funktionen sind unter einer gemeinsamen Bedieneroberfläche vereinigt und können über Pull-Down-Menüs oder Tastenkombinationen aufgerufen werden (siehe Abbildung 6).

Die Entwicklungsumgebung Atmel Studio kann inkl. Simulator kostenfrei heruntergeladen werden. Enthalten sind auch Code-Beispiele und eine vollständige Liste der Befehle des ATmega16.

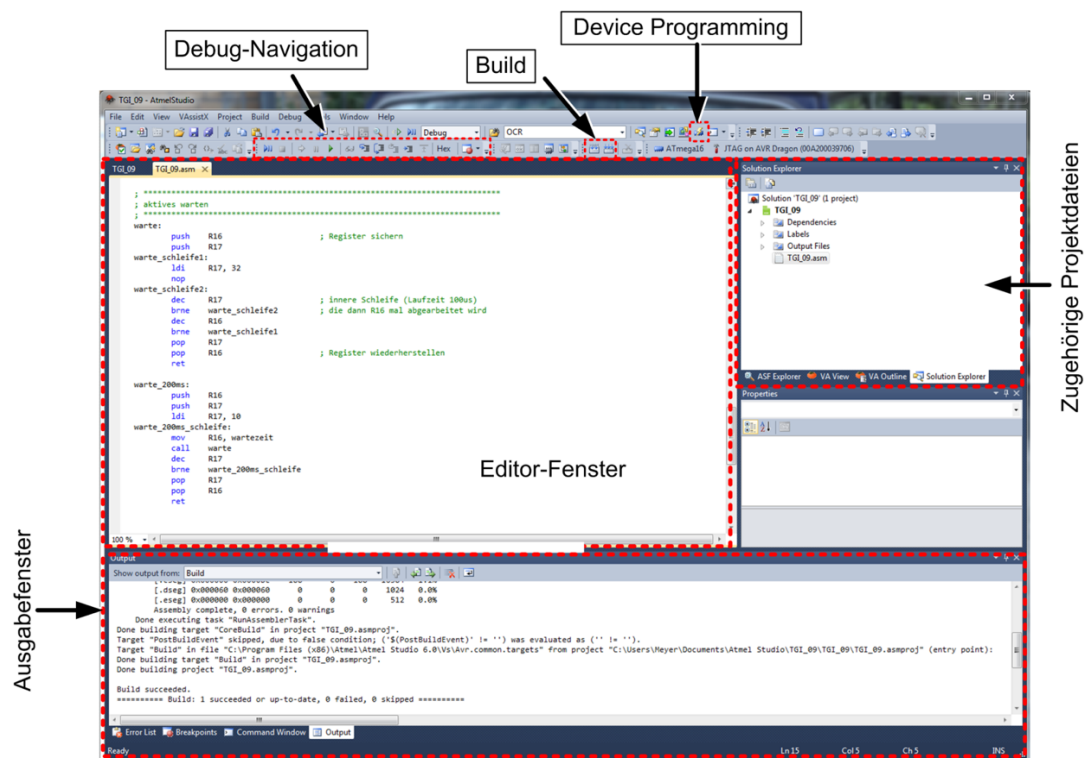


Abbildung 6: Das Atmel Studio

Der Editor

Um ein neues Programm für den ATmega16 einzugeben, öffnen Sie zunächst über das Menü „File/New/Project“ ein neues Projekt. Im folgenden Fenster kann dort die Programmiersprache, in diesem Fall Assembler ausgewählt werden. Weiterhin wird der Projektpfad und Name gesetzt. Als nächstes erscheint ein Fenster, in dem der Mikrocontroller ATmega16 ausgewählt werden sollte. So werden die ATmega16 spezifischen Eigenschaften geladen und zur Verfügung gestellt.

Anschließend können Sie sofort mit der Eingabe beginnen. Achten Sie dabei auf eine strukturierte Eingabe, d.h. sorgen Sie dafür, dass Label, Assembleranweisungen, Operand(en) und Kommentar(e) jeweils in der gleichen Textspalte beginnen. Arbeiten Sie dazu mit Tabulatoren und lassen Sie ausreichend Platz für längere, aussagekräftige Label-Namen. Kommentieren Sie Ihr Programm an allen wichtigen Stellen.

Der Assembler

Haben Sie die Eingabe des Programms bzw. eines Teils beendet, können Sie es über das Menü „Build/Build Solution“, oder auch über den Knopf über dem Editor-Fenster übersetzen lassen. Etwaige (Tipp-)Fehler werden im Ausgabefenster des Assemblers angezeigt. Ist der Assembler mit Ihrem Programm zufrieden, so können Sie Ihr Programm jetzt mit dem Simulator testen, oder in den Controller übertragen.

Der Simulator

Bevor der Debugger benutzt werden kann, muss die Zielplattform ausgewählt werden. Hierfür muss der verwendete Mikrokontrollertyp eingestellt (ATmega16) und festgelegt werden, ob das Programm im Simulator auf dem PC, oder direkt in einem über ein spezielles Programmiergerät (AVR Dragon) angebundenen Mikrocontroller ausgeführt werden soll. Die entsprechende Auswahlmöglichkeit findet sich im Menü „Project“ unter dem Punkt „Properties“ (vgl. Abbildung 6).

Nach dem Start des Debuggers wird im Editorfenster die erste Anweisung mit einem gelben Pfeil markiert. Sie haben nun die Möglichkeit Ihr Programm schrittweise oder „frei“ laufen zu lassen. Rechts vom Editor-Fenster werden die Register der MCU im „Prozessor View / IO View“ übersichtlich dargestellt. Im Simulator-Betrieb können Sie hier auch Eingaben vornehmen. Dieses Fenster wird jedoch nur aktualisiert, wenn Sie Ihr Programm schrittweise ausführen, oder aber kurz unterbrechen!

In weiteren Fenstern können Ausschnitte aus dem Adressraum des Controllers (z.B. Registerbelegungen und Speicher) angezeigt werden.

Zum Debuggen Ihres Programmes wird es nützlich sein, an bestimmten Stellen im Programm Breakpoints einzufügen (Taste F9). Das Programm wird dann gestoppt, sobald die entsprechende Zeile erreicht wird. Im Simulator können Sie beliebig viele Breakpoints einfügen, falls Sie das Programm auf dem „echten“ ATmega16 debuggen, sollten Sie möglichst wenige Breakpoints zeitgleich aktivieren.

Sollten Sie auf dem echten ATmega debuggen wollen, achten Sie auf die richtige Konfiguration (siehe Abbildung 7)

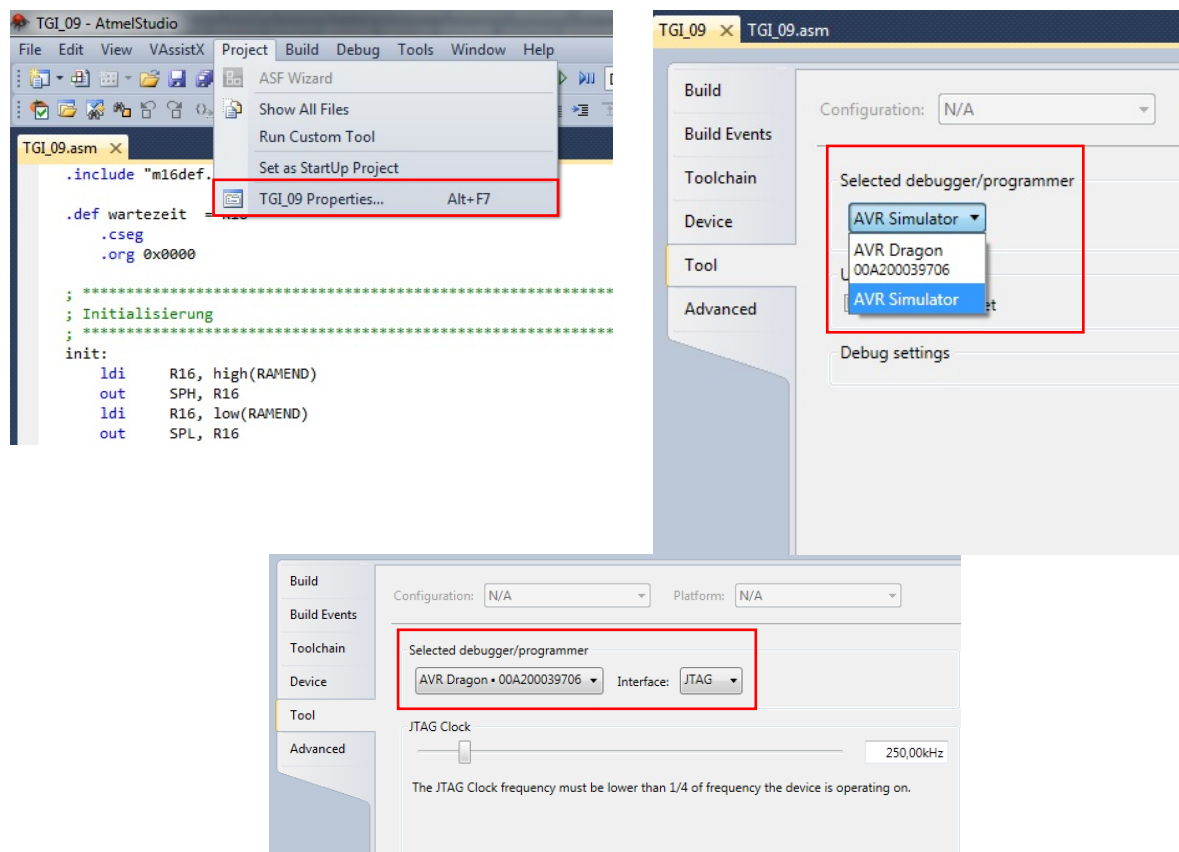


Abbildung 7: Auswahl der Zielplattform

Versuchsdurchführung

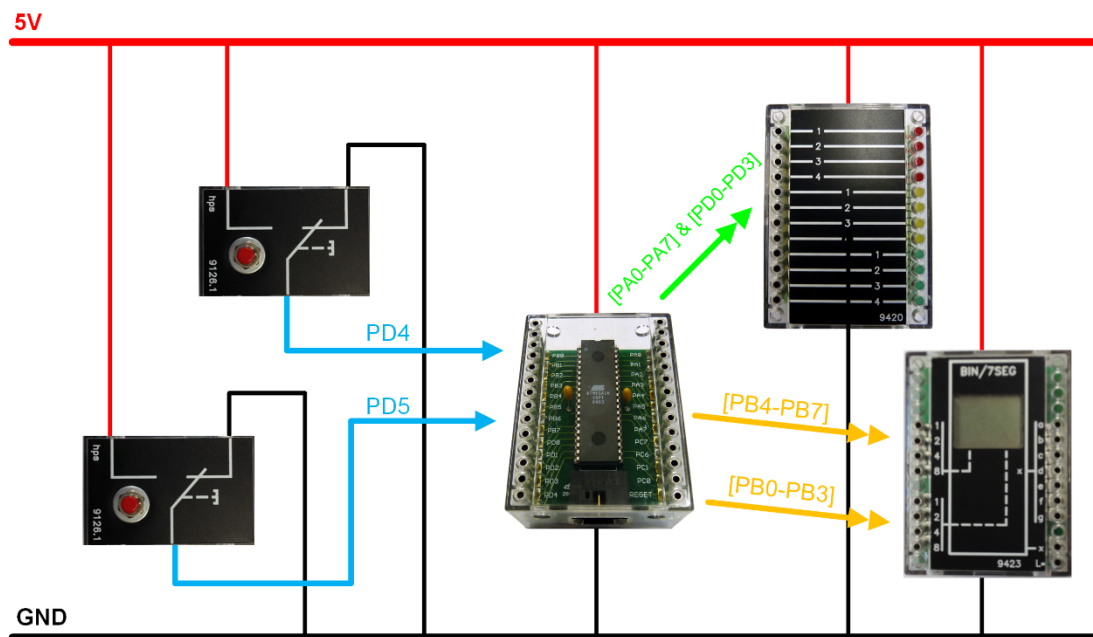
Bevor Sie ihren Versuch auf dem Steckbrett aufbauen, sollten Sie das von Ihnen in der Versuchsvorbereitung geschriebene Programm eingeben und mit Hilfe des Simulators überprüfen. Starten Sie dazu die Entwicklungsumgebung Atmel Studio 7.



Geben Sie nun schrittweise Ihren Programmcode ein. Dabei sollten Sie *bottom-up* vorgehen, d.h. zuerst einzelne, in sich abgeschlossene Programmteile eingeben und testen. Vergessen Sie nicht, Ihren Quelltext gelegentlich abzuspeichern.

Benutzen Sie nach der Eingabe verschiedener Programmteile den Assembler und korrigieren Sie nacheinander die eventuell angezeigten syntaktischen Fehler in Ihrem Code. Danach können Sie die korrekte Funktion der Programmteile im Simulator überprüfen. Außerdem können Sie während der Simulation auf der rechten Seite die Schnittstellen zum ATmega beeinflussen und so bspw. Eingaben am Controller simulieren, indem Sie einen Port auswählen und die Pins manuell mit der Maus setzen. Beobachten Sie dabei die Zustände der angezeigten Register und testen Sie, ob sich Ihr Unterprogramm erwartungsgemäß verhält. Stoßen Sie beim Simulieren Ihres Programms auf Fehler, ändern Sie es im Editor entsprechend ab und assemblieren Sie danach das Programm erneut.

Bauen Sie anschließend den Versuch entsprechend des unten gegebenen Versuchsaufbaus auf dem Steckbrett auf. Lassen Sie sich den Aufbau von einem Betreuer unbedingt abnehmen!

Abbildung 8: Versuchsaufbau



Übertragen Sie jetzt Ihr Programm auf den Mikrocontroller und starten Sie es dort. Sie müssen hierfür den Button  (Device Programming) in der Symbolleiste von Atmel Studio betätigen oder das Programm mit Hilfe des Debuggers  auf den Mikrocontroller übertragen und starten. Für den ersten Fall stellen sie jeweils als Programmiergerät AVR Dragon, als Interface JTAG (nicht ISP) und als Mikrocontroller ATmega16 ein und betätigen Sie „Apply“.

Um nun ihr zuletzt assembliertes Programm auf den ATmega zu laden, wählen Sie im Auswahlménú links „Memories“ aus. Dort können Sie im Bereich „Flash“ ihr erstelltes Hex-File auswählen und mittels „Program“ auf den ATmega überspielen. Direkt nach erfolgreicher Übertragung startet so das Programm.

Ändern Sie auf keinen Fall etwas in den Bereichen der anderen Reiter (Fuses etc.), da sonst der ATmega zerstört werden kann.

Demonstrieren Sie Ihr hoffentlich funktionstüchtiges Programm abschließend einem Betreuer.

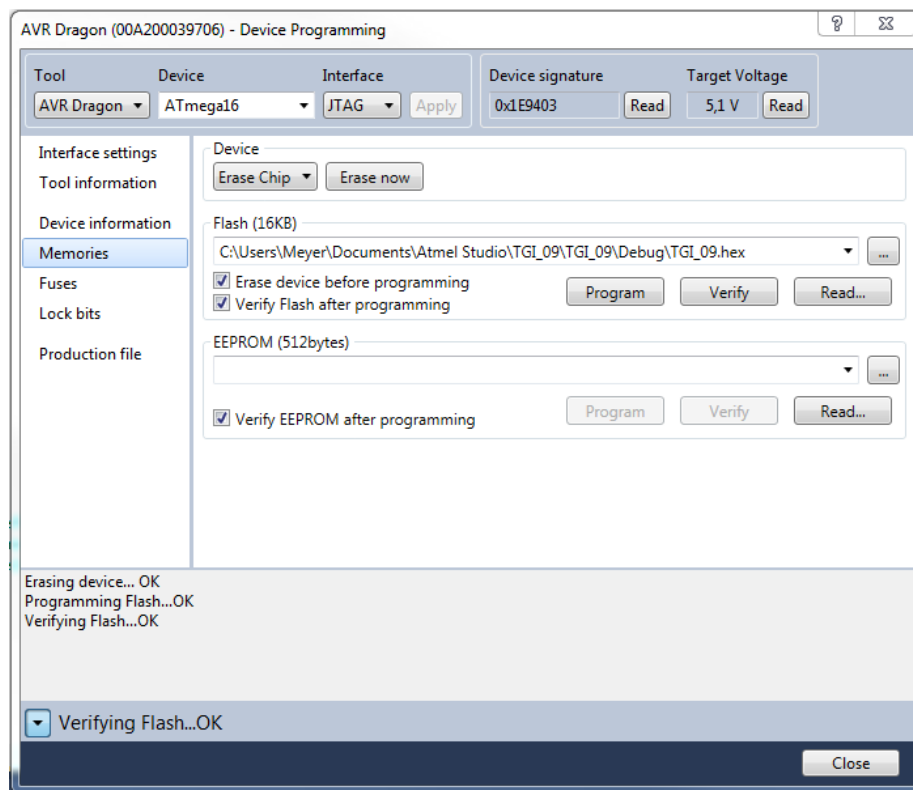


Abbildung 9: Programmierdialog des Atmel Studio