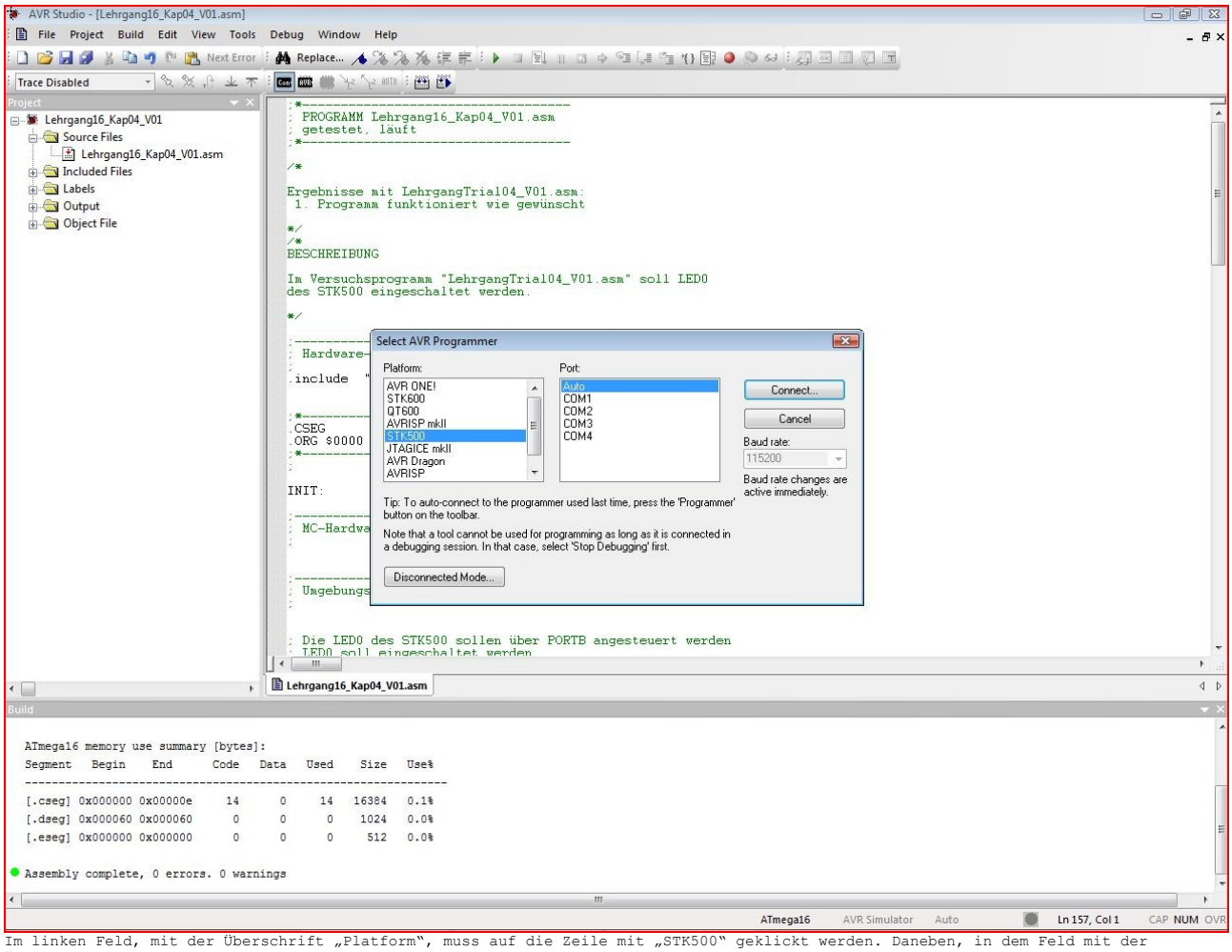
Kapitel 04: Das erste Projekt
Abschnitt 03: Das Programm flashen

Steckt der ATmega16 schon in seiner Fassung auf dem STK500? Wenn nicht, dann ist es jetzt Zeit, ihn dort, so wie in Kap02 beschrieben einzustecken, alle Brücken in die richtige Position zu setzen und das RS232-Kabel vom PC an der (von der Steckerseite aus gesehen) rechten RS232-Buchse einzustöpseln. Das macht man natürlich alles bei ausgeschalteter Spannungsversorgung am STK500.

Mit dem Programm wollen wir die LED des STK500 über den PORTB ansteuern. Deshalb müssen die beiden 10-Pin-Pfostendoppelreihen „LEDS" mit der „PORTB" mit einem der mitgelieferten Flachbandkabel verbunden werden. Das Flachband darf dabei nicht verdrillt sein, d.h. die rote Ader muss bei beiden Pfostenreihen auf der Seite mit der Aufschrift „VTG" enden.

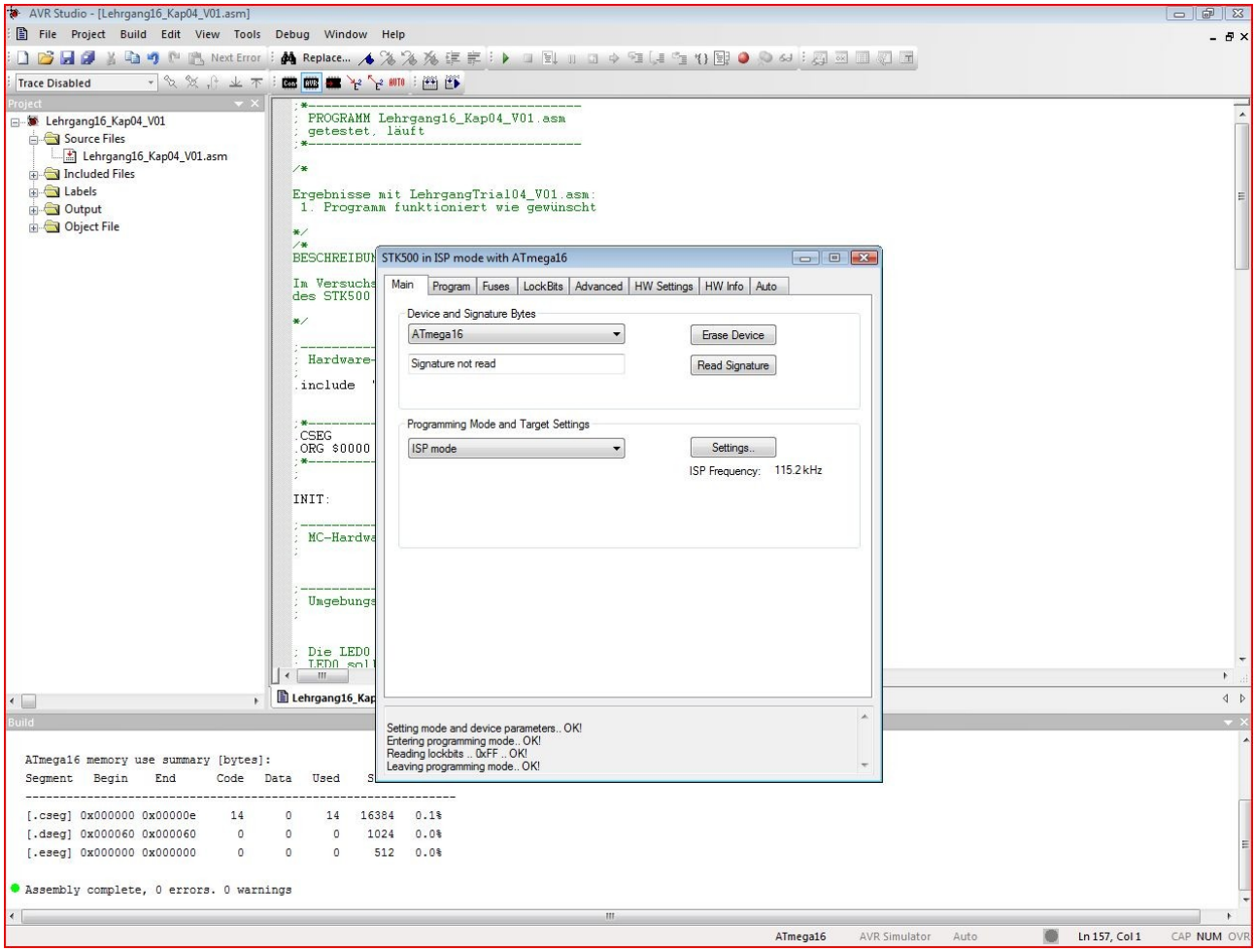
Also dann: Das STK500 mit dem Schiebeschalter einschalten. Jetzt muss gleich neben dem Einschalter die rote LED leuchten, sonst fehlt der Saft. Etwas weiter im Innern des STK500 ist eine grossflächigere LED. Sie muss nach dem Einschalten kurzzeitig rot, dann gelb, dann grün leuchten und einmal blinken. Noch weiter hinten, an demselben Rand wie die rote LED und direkt neben der Reihe mit den Steckbrücken muss eine grüne LED leuchten. Wenn das Leuchtmuster der LEDs irgendwie anders ist, dann die Spannung gleich wieder ausschalten - dann stimmt etwas nicht. In diesem Fall gilt es noch einmal alle Verbindungen und Steckbrücken anhand des STK500-Handbuches zu kontrollieren. Auch darauf achten, dass der ATmega richtig herum eingesteckt ist: Die Kerbe am Chipgehäuse muss zu den LED hin und von den RS232-Steckbuchsen weg zeigen! Wenn er andersherum steckt (kommt mir auch immer noch, immer wieder vor) - nicht verzweifeln: Normalerweise überlebt der Chip diese Tortur. Es schadet aber nichts, gleich nach dem Einschalten für ein Weilchen einen Finger auf den ATmega-Chip zu halten, um die Versorgungsspannung sofort abschalten zu können, falls er warm wird. In aller Regel, wenn er keine grossen Verbraucher treiben muss, bleibt der ATmega nämlich völlig kalt.

So, wenn alle LED in der richtigen Farbe und ohne zu blinken leuchten, dann sind wir jetzt bereit, die Unterhaltung zwischen AVRStudio, STK500 und dem ATmega16 einzuleiten. Dazu muss man auf den Knopf in der zweiten Toolbar-Zeile klicken, auf dem auf einem stilisierten, schwarzen IC mit weisser Schrift „Con" steht (der Knopf befindet sich direkt unter dem Knopf mit dem Fernglas). Zum Vorschein kommt (STK500_Connect_V01.JPG):



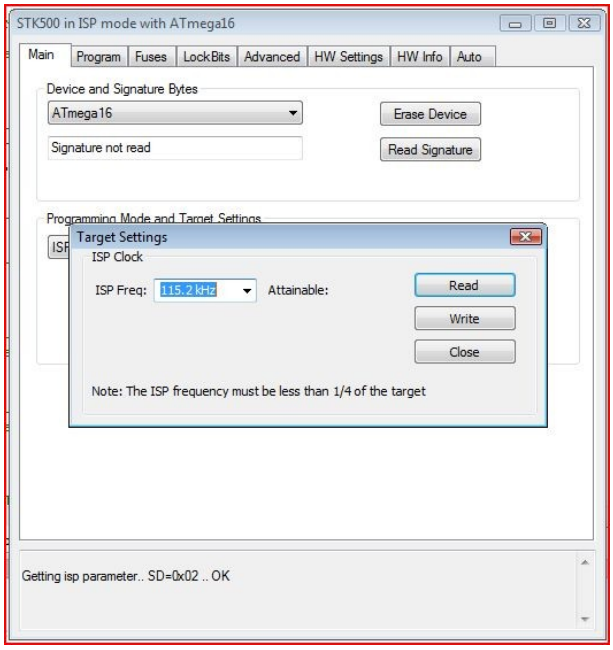
Im linken Feld, mit der Überschrift „Platform", muss auf die Zeile mit „STK500" geklickt werden. Daneben, in dem Feld mit der Überschrift „Port" klickt man am Besten auf die Zeile „Auto", dann sucht sich das AVRStudio selber den Anschluss aus, über den das STK500 mit dem PC verbunden ist. Die Baudrate (in der Drop-down-box unter der Überschrift „Baud rate" habe ich nie anzufassen brauchen, die Einstellung lief immer automatisch. Dann können wir jetzt auf den Knopf „Connect" klicken.

Und das Ergebnis ist (STK500_Config001_V01.JPG):



In diesem Dialogfenster müssen wir uns erst noch durch ein paar Reiter durch klicken, bevor wir den ATmega programmeren („flashen") können.

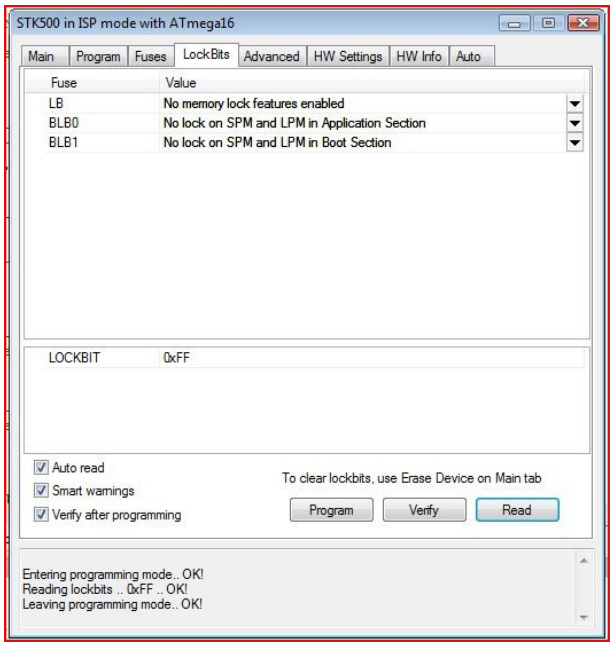
Los geht's mit dem Reiter „Main". In der Drop-down-box „Device and Signature Bytes" gilt es, den ATmega16 auszuwählen. In der Drop-down-box „Programming Mode and Target Settings" muss „ISP mode" ausgewählt werden. Den „PP/HVSP mode", den High-Voltage Programming mode braucht man eigentlich nur, wenn man versehentlich die Fuses oder LockBits falsch eingestellt hat. Dann bietet diese Programmierart die letzte Möglichkeit, den Chip vor dem Mülleimer zu retten. Als nächstes gilt es auf den Knopf „Settings" zu klicken (STK500_ConfigSettings001_V01.JPG):



In dem Fenster, das sich öffnet, lässt sich die Frequenz einstellen, mit der das Programm in den Flashspeicher des ATmega übertragen wird. Dazu muss man die Drop-down-box neben „ISP-Freq:" betätigen. Ich wähle gern „115.2kHz", weil der Vorgang so sehr zügig vonstatten geht und meist nicht das Grenzkriterium (siehe die „Note" darunter) verletzt, d.h. 1/4 des MC-Taktes nicht überschreitet (bei 8MHz wäre das 2MHz). Will man ganz sichergehen, dann kann man auch anfangs die „4.00 kHz" oder gar „1.21 kHz" wählen. Aber, spätestens, nachdem der MC-Takt mit den Fuses eingestellt ist, kann man getrost eine höhere Übertragungsrate einstellen.

Die Einstellung muss jetzt dem STK500 mitgeteilt werden, also auf den Knopf „Write" klicken. Wenn daraufhin nichts passiert, dann hat's geklappt. Um die Einstellung zu kontrollieren, anschliessend nochmal schnell auf „Read" klicken und sehen, ob die Anzeige in der Drop-down-box „ISP Freq:" unverändert bleibt. Manchmal verstellt sich diese Einstellung nämlich von selber. Das ist die ärgerlichste Fehlfunktion, die mir bisher im AVRStudio 4.18 bisher vorgekommen ist. Aber jetzt ist es Zeit auf den „Close"-Knopf zu klicken; mit diesem Fenster sind wir fertig.

Jetzt knöpfen wir uns das Heikelste vor: Das Fenster mit dem Reiter „LockBits“. Locks sind eine Art Riegel, mit denen man Zugriffe auf den Programmspeicher des ATmega sperren kann. Man kann so verhindern, dass das Programm im MC überschrieben oder gelesen werden kann (STK500_ConfigLockBits001_V01.JPG).



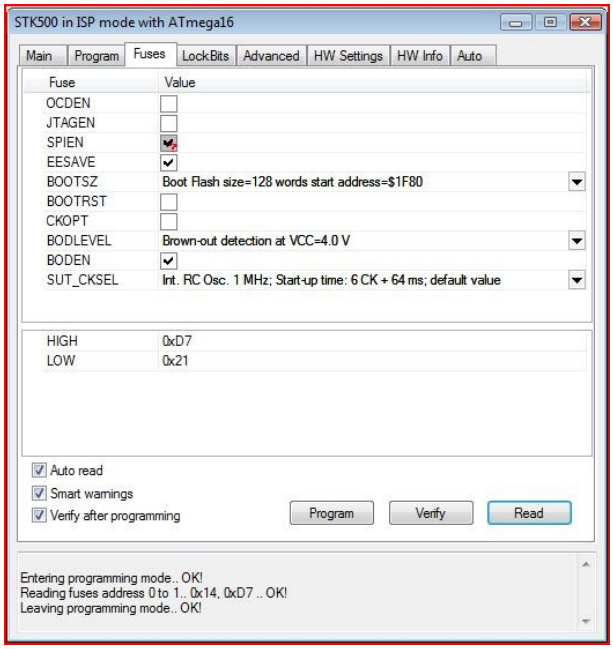
Bei Auslieferung sind die LockBits werkseitig in der „ungefährlichen“ Stellung:
„LB No memory lock features enabled“
„BLB0 No lock on SPM and LPM in Application Sektion“
„BLB1 No lock on SPM and LPM in Boot Section“

Ist man sich unsicher, dann ist es meist eine gute Entscheidung, auf den Knopf „Read“ zu klicken. Dann lädt das AVRStudio die aktuellen Einstellungen aus dem MC herauf.

Wenn man nicht gerade professionelle Anwendungen programmiert, bei denen Geheimhaltung und der Schutz der Urheberreche eine Rolle spielt, dann genügt es völlig die LockBits in der „ungefährlichen“ Stellung zu belassen (oder sie in diese Stellung zu bringen).

Einzelheiten über die LockBits kann man im Datenblatt des ATmega16 in Abschnitt 26.1 (Seite 264) nachlesen. Die LockBits können auch vom Assemblerprogramm verändert werden.

Dann gehen wir mal weiter zu dem Reiter „Fuses“ (STK500_ConfigFuses001_V01.JPG).



Diese Einstellungen sind zwar nicht so heikel wie die der LockBits, aber mit den Einstellungen von SPIEN und SUT_CKSEL sollte man sehr sorgfältig umgehen. Bei Auslieferung sind die Einstellungen
„OCDEN kein Häkchen (ausgeschaltet)“
„JTAG Häkchen (eingeschaltet)“
„SPIEN Häkchen (eingeschaltet)“
„EESAVE kein Häkchen (ausgeschaltet)“
„BOOTSZ Boot Flash size = 128 words start adress=$1F80“
„BOOTRST kein Häkchen (ausgeschaltet)“
„BODLEVEL“ Brown-out detection at VCC=4.0 V“
„CKOPT kein Häkchen (ausgeschaltet)“
„BODEN Häkchen (eingeschaltet)“
„SUT_CKSEL Int. RC Osc. 1 MHz; Start-up time: 6 CK+64 ms“

Die Bedeutung der einzelnen Fuses ist im Datenblatt in Abschnitt 26.2 (Seite 265) ausführlich beschrieben. Hier wird nur auf die grössten Fettnäpfchen eingegangen, in die man hier treten kann.

Das erste ist die Fuse „JTAGEN“. Mit steuert man den Zugang zum MC für JTAG-Geräte. Sie ermöglichen die Fehlersuche im Programm (Debugging) direkt auf dem Chip (Details siehe Abschnitt 23 des Datenblattes, Seite 226ff). Dazu müssen sie über bestimmte Ein- und Ausgangskanäle mit dem ATmega „reden“ können. Beim ATmega16 sind dies die Kanäle
PC2 (TCK) an Bein 24
PC3 (TKS) an Bein 25
PC4 (TDO) an Bein 26
PC5 (TDI) an Bein 27
Diese Kanäle sind dadurch aber für alle anderen Funktionen blockiert, auch für das selbstgeschriebene Programm. Wenn man also die JTAG-Option nicht benutzt, sollte man die Fuse „JTAGEN“ abschalten (kein Häkchen).

Und da wir schon einmal dabei sind: Auch, wenn man den ATmega mit dem STK500 oder einem anderen ISP-Programmierer flasht, müssen dazu bestimmte Ein- und Ausgangskanäle zur Verfügung stehen. Beim ATmega16 sind das die drei Kanäle
PB5 (MOSI) an Bein 6
PB6 (MISO) an Bein 7
PB7 (SCK) an Bein 8
Diese Kanäle dürfen in Anwendungen auch für andere Dinge benutzt werden (ein Beispiel dafür ist unser erstes Programm, in dem über die Kanäle des PORTB die LED auf dem STK500 angesteuert werden können), d.h. mit anderen Bauteilen verdrahtet werden. Man muss aber immer ausprobieren, ob diese anderen Bauteile den Flash-Vorgang behindern oder nicht. Wenn ja, dann muss man Jumper einbauen, die während des Flashens die störenen Bauteile von den Kanälen trennen.

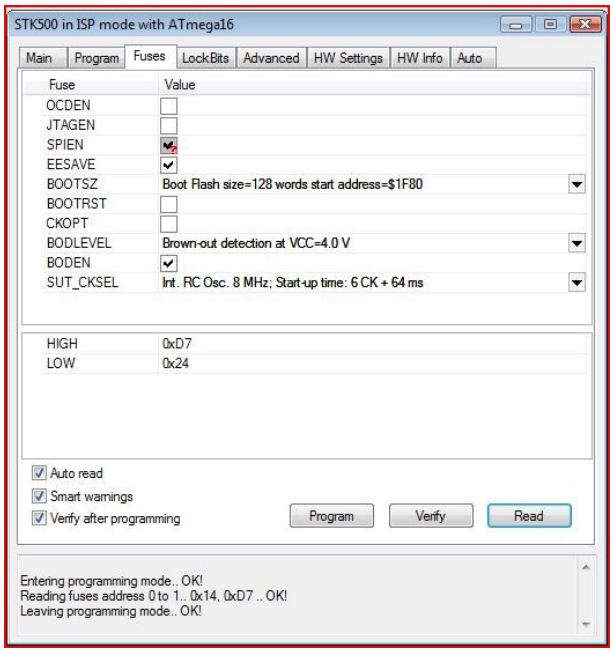
Kritisch ist auch die Fuse „SPIEN“. Damit kann man den SPI-Bus des ATmega desaktivieren. Über diesen Bus wird aber auch das Programm in den Chip übertragen. Wenn man also die Fuse „SPIEN“ abschaltet (kein Häkchen), dann kann man den Chip nicht mehr programmieren. Mit dieser Fuse gilt es also sehr sorgfältig umzugehen.

Weniger kritisch, aber doch heimtückisch, kann die Fuse SUT_CKSEL werden. Sie dient dazu, die Signalquelle für den ATmega einzustellen. Die möglichen Einstellungen findet man im Datenblatt in Abschnitt 8.4 (Seite 26ff). Wenn der ATmega schon auf einer Platine eingelötet und auf den internen Oszillator eingestellt ist, dann kann man mit dieser Fuse versehentlich auf einen externen Taktgeber umschalten. Und weil es den auf der Platine nicht gibt, fehlt dem MC danach der Takt, ohne den man ihn weder programmieren, noch das Programm abarbeiten kann. Der MC ist also so gut wie tot - die einizige mögliche Wiederbelegungsmassnahem ist, ihn auszulöten, in eine andere Platine mit einem externen Taktgeber passender Frequenz einzusetzen und die Fuse „SUT_CKSEL“ dort neu zu programmieren. Meist endet dieses Manöver aber mit einem kaputten MC und schlimmstenfalls, einer kaputten Platine. Vorsichtige Anwender spendieren ihren ATmegas deshalb auf der Platine lieber einen Quarz oder externen Oszillator.

Die Fuse „EESAVE“ kann keinen vergleichbaren Schaden anrichten, aber viel Zeit kosten. Ist sie nicht eingeschaltet (Häkchen), dann wird der Inhalt des EEPROM bei jedem Programmiervorgang gelöscht. Im EEPROM speichert man als Anwender gern Betriebsdaten, die für das Gerät charakteristisch sind, in dem der MC eingebaut ist. Das können z.B. Regelparameter, charakteristische Abmessungen oder Kalibrierdaten sein. Oft hat es viel Zeit und Mühe gekostet, diese Daten zu ermitteln. Wenn man dann vergisst, die Fuse „EESAVE“ einzuschalten (sie werkseitig bei Auslieferung abgeschaltet), dann sind diese Daten nach jedem Update des Programms futsch. Man merkt das fast immer sofort, weil sich das Gerät komisch verhält. Aber meist glaubt man dann zuerst an einen Programmierfehler und sucht ihn eine frustrierende Ewigkeit lang, bis man schliesslich merkt, dass die fehlenden Betriebsdaten die Ursache sind. Wenn man also das EEPROM in seinem Programm verwendet, dann unbedingt die Fuse „EESAVE“ einschalten!

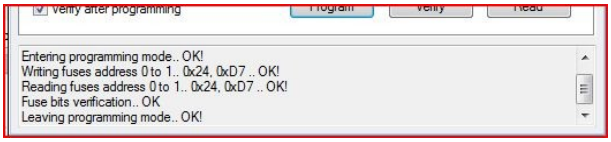
Die drei Häkchen bei „Auto read“, „Smart warnings“ und „Verify after programming“ sollten immer gesetzt sein.

Für unser erstes Testprogramm genügt es aber völlig, mit der Fuse „SUT_CKSEL“ die Taktquelle „Int. RC Osc. 8 MHz; Start-up time: CK +64 ms“ einzustellen. Das Fenster sollte jetzt so aussehen (STK500_ConfigFuses002_V01.JPG):

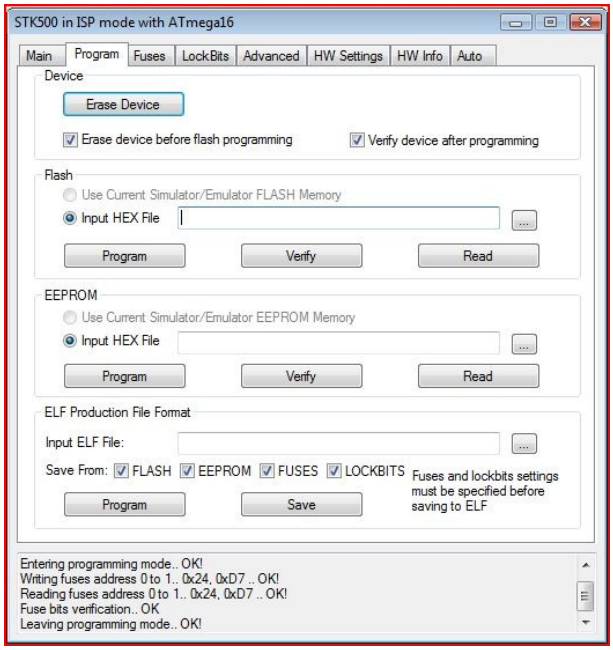


Und jetzt programmieren wir zum ersten Mal etwas in den ATmega: Die Fuse-Einstellungen. Dazu braucht man einfach nur auf den Knopf „Program“ klicken.

Das Resultat wird im untersten Bereich des Fensters angezeigt. Wenn alles geklappt hat, steht da jetzt (STK500_ConfigFuses003_V01.JPG):



So. damit können wir daran gehen, unser erstes Programm in den MC zu wuchten. Dazu muss man auf den Reiter „Program“ klicken. Es erscheint folgendes Fenster (STK500_Program001_V01.JPG):



Von diesem Fenster aus können wir unser Programm, dass dank des Assemblers jetzt als „.hex“-Datei auf der Festplatte steht, in den Programmspeicher des ATmega übertragen. Seine Speicherzellen sind funktionieren nach dem „Flash“ („floating gate „)-Prinzip. Wie das im Einzelnen aussieht braucht uns nicht weiter interessieren. Aber es hat sich eingebürgert, beim ATmega den Programmspeichern auch als Flash Speicher zu bezeichnen und den Programmiervorgang als „flashen“.

Unter der Überschrift „Device“ findet sich hier zunächst ein Knopf „Erase Device“. Den braucht man eigentlich nie, denn das Häkchen darunter links neben „Erase device before flash programming“ sollte immer gesetzt sein. Dadurch löscht AVRStudio den Flash spleicher jedesmal, bevor es eine neues Programm einschreibt.

In den einschlägigen Foren gibt es öfters besorgte Fragen, ob man nicht vorsichtig sein müssen, weil so ein Flashspeicher ja nur eine begrenzte Anzahl von Programmierzyklen vertrüge; im Datenblatt Abschnitt 7.2 (Seite 16) heisst es, der Flash Speicher vertrüge mindestens 10.0000 Lösch- und Programmierzyklen. Mir ist es noch nie passiert, dass ein ATmega aus diesem Grund seinen Dienst versagte; die wenigen, die mir wirklich kaputtgegangen sind, habe ich anders gemeuchelt. - Also: keine Sorge, der Flash Speicher verträgt allerhand.

Das Häkchen neben „Verify device after programming“ sollte auch gesetzt sein. Es bewirkt, das AVRStudio nach dem Programmieren immer den Inhalt des Flash Speichers zurückliest und mit dem Inhalt der „.hex“-Datei vergleicht. Das ist sehr praktisch, weil man auf diese Weise sofort mitkriegt, sollte doch einmal eine Speicherzelle schlapp machen.

Darunter ist ein Kringel neben „Use Current Simulator/Emulator FLASH Memory“, der nicht gesetzt ist. Sollte er auch nicht sein, denn er würde den Programmiervorgang nicht mit dem ATmega ausführen, sondern das Programm nur dem im AVRStudio eingebauten Simulator übergeben. Wir wollen es aber tatsächlich auf dem Chip haben. Deshalb muss dieser Kringel leer bleiben und der darunter, neben „Input HEX File“ muss gefüllt sein. In dem Feld direkt rechts davon muss nun der volle Name der „.hex“-Datei angegeben werden, das der Assemblierer erzeugt hat; und zwar mit dem kompletten Pfad. Gottseidank, es gibt den „...“-Knopf daneben, der es einem ermöglicht, den Dateipfad und -namen über den Datei-Explorer anzuklicken.

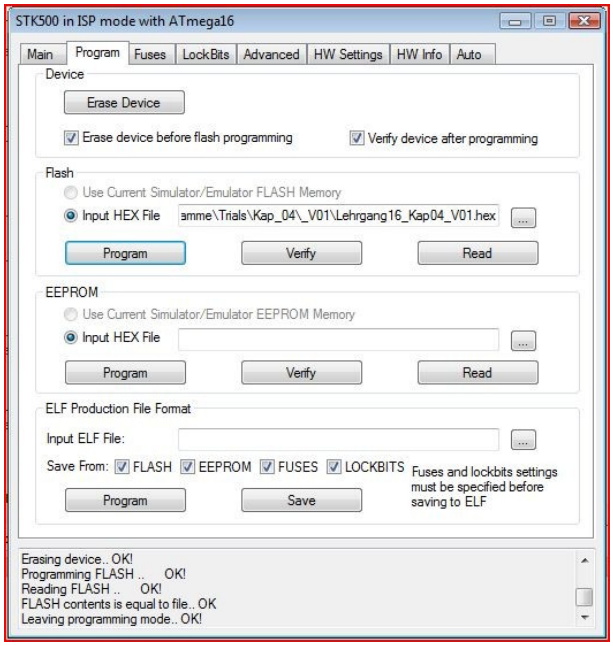
Unter dem Feld für den Dateinamen gibt es noch einmal drei Knöpfe: Auf dem ganz rechten steht „Read“. Damit kann man den Inhalt des Flash Speichers aus einem ATmega in eine „.hex“-Datei auf der Festplatte hochladen. Auf diese Weise kann man auch fremde Programme einlesen und analysieren. Habe ich noch nie gemacht, ich mache meine Programme lieber selbst. Den Knopf in der Mitte, mit „Verify“ drauf, braucht man, um den Inhalt des Flash Speichers eines ATmegas mit dem einer „.hex“-Datei zu vergleichen. Das ist sehr hilfreich, wenn man nicht mehr sicher ist, welche Programmversion man in den Chip geschrieben hat. Das kommt oft vor, wenn der Chip auf einer Platine sitzt, die man vor längerer Zeit einmal gelötet und programmiert hat. Der linke Knopf „Program“ ist der auf den wir

jetzt klicken. Was sonst noch in diesem Fenster zu sehen ist, brauchen wir jetzt nicht.

Wenn das Programm länger ist, dann kann man den Fortschritt beim Speichern und Rückvergleichen des Programms an einem grünen Balken verfolgen, der in der untersten Zeile des AVRStudio-Fensters zu sehen ist, direkt unter dem horizontalen Schieber und links von „ATmega16". Aber bei unserem kleinen Programm geht das Ganze so schnell, dass man's gar nicht sehen kann.

Mehr Aktivitäten sieht man auf dem STK500: Solange das AVRStudio über den ISP-Anschluss auf den ATmega zugreift, leuchtet die grosse LED (die etwas links von der Mitte des Boards) rot statt grün, und die drei LED (LED5, LED6 und LED7) am rechten Ende des Boards leuchten gelb auf.

Das Fenster sieht danach so aus (STK500_Program002_V01.JPG):



Das Entscheidende steht wieder in dem grauen Feld ganz unten:
„Erasing device..OK!" Der Löschvorgang war erfolgreich
„Programming FLASH .. OK!" Der Schreibvorgang war erfolgreich
„Reading FLASH.. OK!" Das Programm wurde erfolgreich aus dem Flash Speicher wieder in den PC hochgeladen
„FLASH contents is equal to file ... OK" Der Inhalt des hochgeladenen Speicherinhalts ist mit dem Inhalt der „.hex"-Datei identisch. - Wunderbar.
„Leaving programming mode..OK!" Fertig mit Programmieren.

So - jetzt schnell einen Blick auf das STK500 werfen: Leuchtet die LED0? Wenn ja, dann läuft unser erstes Programm auf dem ATmega16.

Ob das auch wirklich unser Programm ist, das diese LED leuchten lässt? Das lässt sich ganz leicht prüfen: Nur mal schnell den Taster „RESET" auf dem STK500 drücken; er befindet sich etwas einwärts von der rot leuchtenden LED am rechten Rand des Boards. Solange man den Taster gedrückt hält, wird der RESET-Eingang des ATmega auf Null gezogen. Solange ist der ATmega untätigt. Und tatsächlich, alle LED sind solange dunkel.

Lässt man den Taster wieder los, dann fängt der ATmega an, sein Programm auszuführen. Die LED0 leuchtet wieder auf. Weil unser Programm es so will. Prima!

Nach diesem Erfolg können wir das Fenster „STK500 in ISP mode with ATmega16" aus-X-en und uns endlich einmal das Programm gründlicher ansehen.