Mikrocontroller-Programmierung mit dem

neuen Board „EST-ATM1“

der Elektronikschule Tettnang unter

und

Band 1b: Einstieg, Einrichtung und Vorgehen bei „C-“ und Assembler-Programmen unter Verwendung der RS232-Schnittstelle

Version 1.1

Von
Gunthard Kraus, Oberstudienrat und Steffen Brink, Studienassessor
Elektronikschule Tettnang

15. Juni 2008

Änderungshinweise:

==========

Software:

Die Firma ATMEL hat die verwendete Programmiersoftware „FLIP“ *am 1. Juni 2008 wesentlich geändert*. Deshalb gilt diese neue Einsteiger-Anleitung (Version 1.1) ab

„Flip 3_3_1 mit integrierter JAVA-Umgebung (*)“

Bitte gleich diesen Software-Stand aus dem Internet (oder von der ATMEL-ATM1-CD) herunterladen und auf dem eigenen Rechner installieren!

(*) = Dadurch wird die Software auf unterschiedlichen Plattformen (z. B. auch unter LINUX) lauffähig, aber nach dem Programmstart dauert es etliche Zeit, bis die JAVA-Umgebung vollständig installiert ist und sich etwas tut.….Außerdem stieg die Größe des Programmpakets von ca. 2 MB auf über 20MB an.

==========

Hardware:

1) Die beiden Widerstände **R5 und R4** (bisher: 6,8k und 10k) wurden am 14. Juni 2008 auf **680Ω und 1kΩ vermindert**, da der integrierte AD-Wandler des Controllers beim Wandelvorgang die mit diesen Widerständen erzeugte Referenzspannung zu stark belastete. Durch die Schaltungsänderung bleibt jetzt die Abweichung unter 2%.

---

2) Die an einen Analogport angelegte Spannung wird nur während eines Messvorganges durch die Wandlerelektronik des Controllers ebenfalls stark belastet. Deshalb sollte der Innenwiderstand der zugehörigen Spannungsquelle (z. B. Sensor) deutlich **kleiner als 5 kΩ** sein.

---

3) Der Vorwiderstand für die grüne POWER-LED in der Stromversorgung wurde auf **1kΩ** erhöht.

Aber jetzt geht es wirklich los….

Inhaltsverzeichnis Seite

1. Vorstellung des neuen EST-ATM1-Boards

Die Zeit war reif -- so könnte man die Geburt unseres neuen Microcontroller-Boards umschreiben. Dabei sollte jedoch nicht ein kompletter Bruch mit der Vergangenheit stattfinden, sondern so viel Erfahrung und Arbeit wie möglich vom Vorgänger („Compuboard 80C535“) übernommen werden. Deshalb lauteten die Vorgaben:

a) Weiterhin ein Controller der 8051-Familie mit möglichst naher Verwandtschaft zum 80C535. Dadurch werden die Weiterverwendung der eingeführten und bewährten KEIL-Software zur Programmerstellung sowie die Portierung der bisher erstellten Programme möglich.

b) Programmspeicherung in einem internen Flash-EPROM, ebenso ein XRAM als EEPROM auf dem Chip

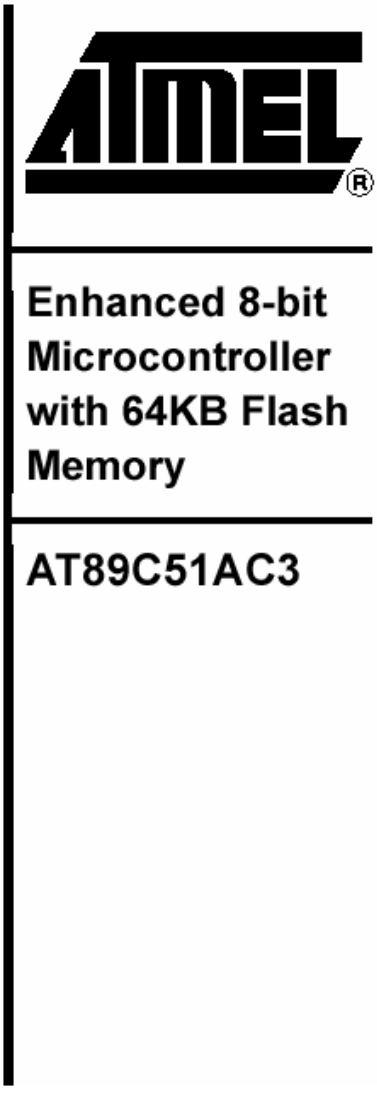
c) Kleinere Board-Abmessungen

d) Eine weitere Ausführung mit „On Board“-USB-Schnittstelle soll irgendwann folgen. Bis dahin kann bei Bedarf mit einem käuflichen **USB-RS232-Konverter** gearbeitet werden. **Siehe dazu den getrennten Band 1a dieses Tutorials!**

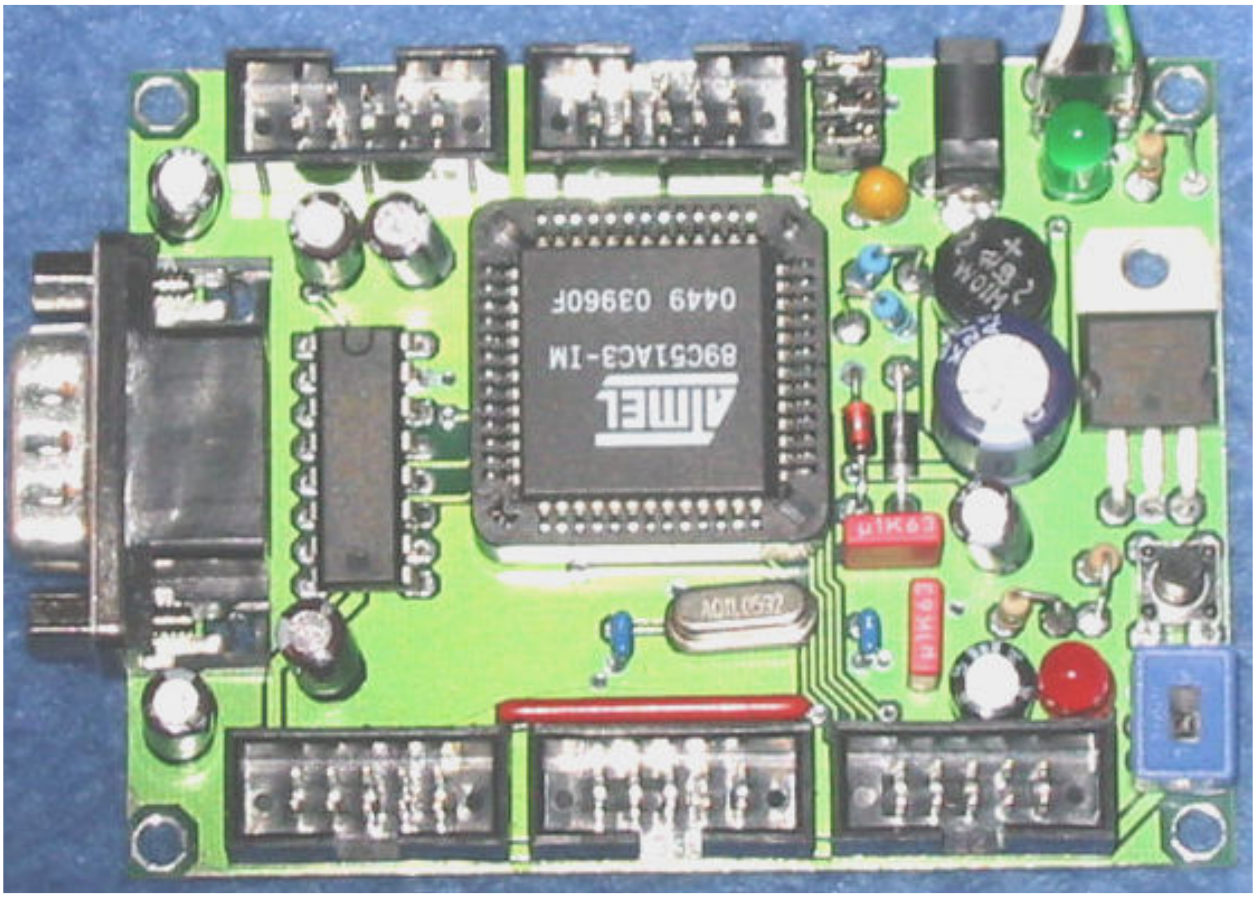
Deshalb fiel die Wahl nach sorgfältiger Analyse auf den **ATMEL AT89C51AC3 im PLCC52-Gehäuse**, der zusätzlich viel Neues bietet, aber leider auch einige Einschränkungen und Klimmzüge bedingen wird….

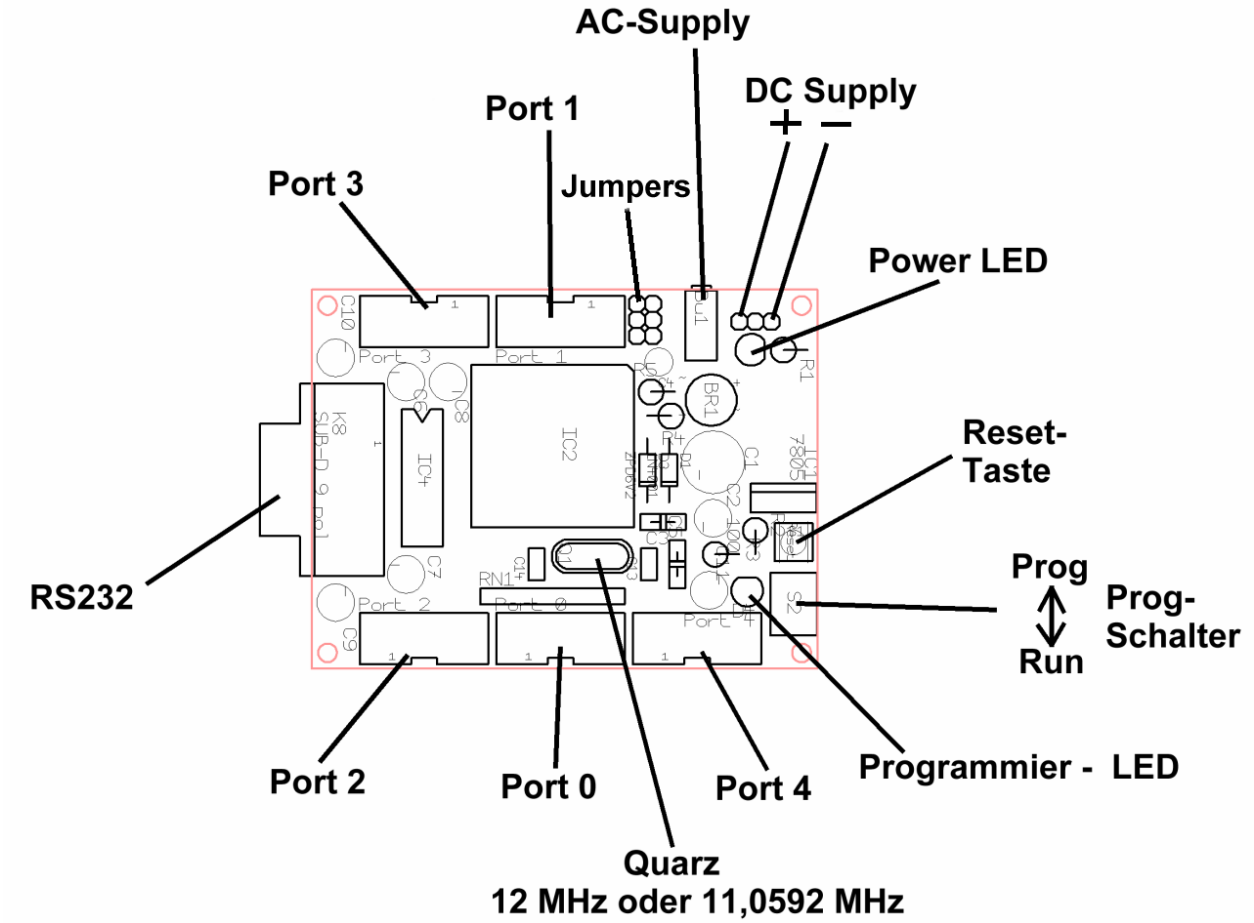
Features

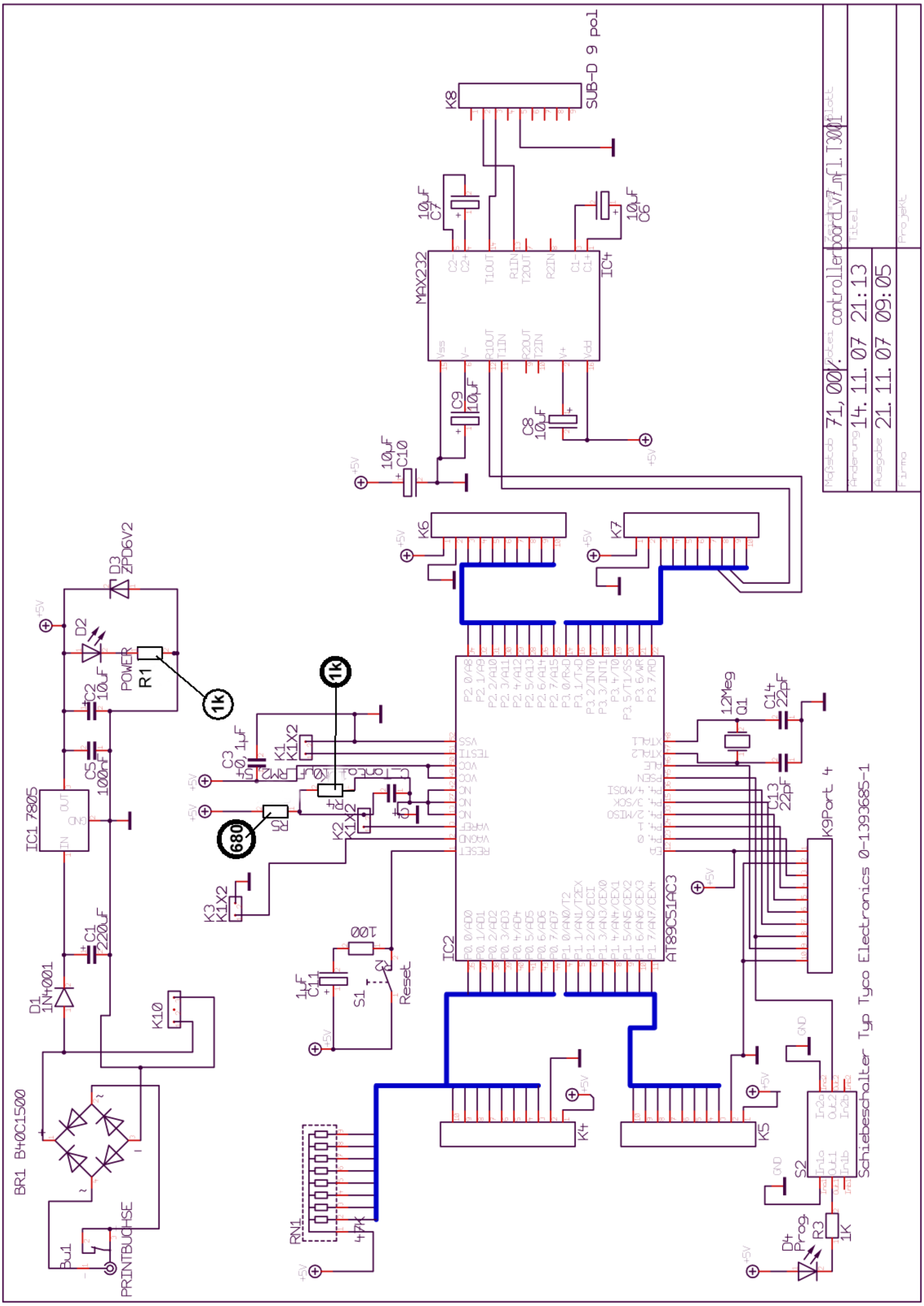
- 80C51 Core Architecture
- 256 Bytes of On-chip RAM
- 2048 Bytes of On-chip ERAM
- 64K Bytes of On-chip Flash Memory
  - Data Retention: 10 Years at 85°C
  - Read/Write Cycle: 100K
- Boot Code Section with Independent Lock Bits
- 2K Bytes of On-chip Flash for Bootloader
- In-System Programming by On-Chip UART Boot Program and IAP Capability
- 2K Bytes of On-chip EEPROM
  Read/Write Cycle: 100K
- Integrated Power Monitor (POR: PFD) To Supervise Internal Power Supply
- 14-sources 4-level Interrupts
- Three 16-bit Timers/Counters
- Full Duplex UART Compatible 80C51
- High-speed Architecture
  - In Standard Mode:
    40 MHz (Vcc 3V to 5.5V, both Internal and external code execution)
    60 MHz (Vcc 4.5V to 5.5V and Internal Code execution only)
  - In X2 mode (6 Clocks/machine cycle)
    20 MHz (Vcc 3V to 5.5V, both Internal and external code execution)
    30 MHz (Vcc 4.5V to 5.5V and Internal Code execution only)
- Five Ports: 32 + 4 Digital I/O Lines
- Five-channel 16-bit PCA with
  - PWM (8-bit)
  - High-speed Output
  - Timer and Edge Capture
- Double Data Pointer
- 21-bit WatchDog Timer (7 Programmable Bits)
- A 10-bit Resolution Analog to Digital Converter (ADC) with 8 Multiplexed Inputs
- SPI Interface (PLCC52 and VPFP64 packages only)
- On-chip Emulation Logic (Enhanced Hook System)
- Power Saving Modes
  - Idle Mode
  - Power-down Mode
- Power Supply: 3 volts to 5.5 volts
- Temperature Range: Industrial (-40° to +85°C)
- Packages: VQFP44, PLCC44, VQFP64, PLCC52



Auf der nächsten Seite folgt ein Foto des Boards samt Lageplan mit den wichtigsten Bauteile und Baugruppen.


0449 03960F
89C51AC3-IM
ATMEL


AC-Supply
DC Supply
+
−
Port 1
Port 3
Jumpers
Power LED
Reset-
Taste
Prog
Prog-
Schalter
Run
RS232
Port 2
Port 0
Port 4
Programmier - LED
Quarz
12 MHz oder 11,0592 MHz


MAX232
IC4
K8
SUB-D 9 pol
C7 10uF
C6 10uF
C9 10uF
C8 10uF
C10 10uF
K6
K7
IC2
AT89C51AC3
Q1 12Meg
C14 22pF
C13 22pF
K9 Port 4
S2
Schiebeschalter Typ Tyco Electronics 0-1393685-1
R3 1K
D4 Prog
K4
K5
RN1 47K
S1 Reset
C11 1uF
100
K3 K1X2
K2 K1X2
K1 K1X2
C3 0,1uF
680
1k
1k
R1 POWER
D2
D3 ZPD6V2
C2 10uF
C5 100nF
IC1 7805
C1 220uF
D1 1N4001
BR1 B40C1500
K10
Bu1 PRINTBUCHSE
71,00%
14.11.07 21:13
21.11.07 09:05

2. Hinweise zur Programmierung

Zu diesem Tutorial (Band 1a bzw. Band 1b) gehört noch ein zweites Manuskript als Band 2. Es soll zeigen, wie unter „C“ die verschiedensten praktischen Anwendungen mit unserem neuen ATMEL-Board „EST-ATM1“ verwirklicht werden können. Aus dem bisherigen Unterricht mit dem 80C535-Board heraus lagen bereits viele Beispiele vor, die auf das neue Board portiert und dann dort ausgetestet wurden. Sie sind alle garantiert lauffähig und sollen sowohl bei der eigenen Arbeit (als Nachschlagewerk) helfen wie auch zu neuen Projekten und Entwürfen anregen.

Folgende Dinge sind allerdings für eine befriedigende Arbeit erforderlich:

a) Ein PC oder Laptop mit der neuesten Version von **KEIL µvision3** und **ATMEL FLIP**

b) Für viele Laptops: ein **USB-RS232-Konverter** (Preis: ca. 10 Euro)

c) Das **EST-ATM1-Board** mit einer passenden **Stromversorgung** (U_{DC} min. +8V, max. +10…+12V erforderlich. Bitte bei Verwendung des zusätzlich vorgesehenen AC-Eingangs am Board nachmessen).

d) Ein **RS232-Null-Modem-Kab**el („Crossed Link“) zur Verbindung mit dem PC.

e) **Ein zusätzlicher Quarz mit 11,059 MHz** für Schnittstellenprogramme. (**Normalbestückung: 12 MHz**).

f) Die **Zusatzplatinen samt LCD-Disp**lay auf der zugehörigen Platine.

g) Ein griffbereiter Ausdruck dieses **ersten ATM1-Tutorials**, denn am Anfang muss man doch noch manchmal nachschlagen, wie das so alles geht.

h) Ein griffbereiter Ausdruck des **zweiten ATM1-Tutorials**, denn es behandelt auf über 140 Seiten in Form von getesteten Beispielen alles, was zur erfolgreichen C-Programmierung nötig ist..

i) Dann sollte man (ggf. auf der ATMEL-Board-CD) notfalls bei Fragen oder Problemen mal im **Datenblatt des Controllers AT89C51AC3** nachsehen können. Dieses Datenblatt findet sich darauf als pdf-File.

==========

Beim Board stehen dem Anwender folgende Ports zur Verfügung:

a) **Port 1** für beliebige Ein- und Ausgabezwecke (einschließlich LCD-Display-Steuerung). Breite 8 Bit = 1 Byte

b) **Port 2** dient normalerweise als Adressbus und kann hier -- da das interne Flash-Eprom verwendet wird -- ebenfalls als 8 Bit – Eingabe- und Ausgabeport (einschließlich LCD-Display-Steuerung) verwendet werden.

c) **Vorsicht**: **Port 0** ist der gemultiplexte Daten- / Adressbus und macht deshalb als Ausgabeport arge Schwierigkeiten. Ist jedoch das zusätzlich nötige externe R-Array (= 8 x 47 Kilo-Ohm) in die Fassung gesteckt, dann kann er ohne Probleme als **Eingabeport** verwendet werden (Ohne die Pullup-Widerstände würden die Pins beim Umschalten auf „Lesen“ sonst frei floaten).

d) Bitte nie vergessen, vor dem Einlesen eines Ports **alle seine Pins durch Beschreiben mit 0xFF auf „HIGH“ zu setzen** und so auf „Lesen“ umzuschalten!

e) Bei **Port 3** sind die Pins P3.0 und P3.1 durch die **Serielle Schnittstelle** belegt. Ebenso liegen auf den Pins P3.6 und P3.7 das „**Write“ und „Read**“-Signal des Steuerbusses. Zur Verwendung der restlichen Pins: Siehe Datenbuch.

f) Bei **Port P4** dürfen **nur die beiden Leitungen P4.0 und P4.1** für Ein- und Ausgabezwecke verwendet werden.

Grundsätzlich gilt:
Ein Portpin kann nur dann zum Einlesen verwendet werden, wenn er zuvor auf „HIGH“ gesetzt wurde. Das gilt natürlich auch für einen kompletten Port, der folglich vorher mit „0xFF“ beschrieben werden muss!

Bitte alle übrigen Informationen zu den Ports und ihre Pin-Belegungen aus dem Datenblatt des Controllers entnehmen.

==========

Achtung:

Der von der Firma KEIL mitgelieferte Header „t89c51ac2.h“ war für das Vorgängermodell des verwendeten Controllers vorgesehen und ist deshalb etwas unvollständig für den hier eingesetzten AT89C51AC3.
Im Anhang dieses Manuskripts und auf der neuen ATMEL-Board – CD findet sich deshalb ein

selbst geschriebener neuer Header „AT89C51AC3.h“, der nun ALLE vom Aufbau her möglichen Deklarationen enthält und bei Bedarf verwendet werden kann.

Soweit nichts anderes angegeben, sind aber die Beispiele mit dem „alten“ (und von KEIL mitgelieferten) Header realisiert.

==========

Das Controllerboard wird normalerweise mit einem **Quarz** bestückt, der eine

Taktfrequenz von 12 MHz

ergibt**. Damit arbeiten alle Programme mit einem „minimalen Befehlstakt“ von 1 MHz und die Timer zählen im Rhythmus von einer Mikrosekunde.**
Dafür sind auch alle Timerprogramme geschrieben.

Sollen jedoch Programme für die integrierte Serielle Schnittstelle erstellt werden, dann wird der Quarz gegen ein Exemplar mit

11,059 MHz

getauscht und deshalb wurde eine Steckfassung zum leichten Wechsel vorgesehen. Nur damit werden die Toleranzgrenzen für die üblichen Baudraten (2400 / 4800 / 9600 Baud) eingehalten und das Board arbeitet korrekt mit anderen RS232-Baugruppen zusammen! (Allerdings werden dadurch die von den Timern produzierten nun um ca. 9% länger….)

Außerdem ergibt sich ein weiterer Vorteil durch diese Umstellung:

Wegen des korrekten Timings können nun die erstellten Hex-Files ohne Probleme und Übertragunsfehler mit maximal 115 200 Baud mittels FLIP in das Board geflasht werden! (bei 12 MHz Quarzfrequenz sind es maximal 9600 Baud)

==========

3. C-Programmierung mit KEIL µvision3 und ATMEL „FLIP“

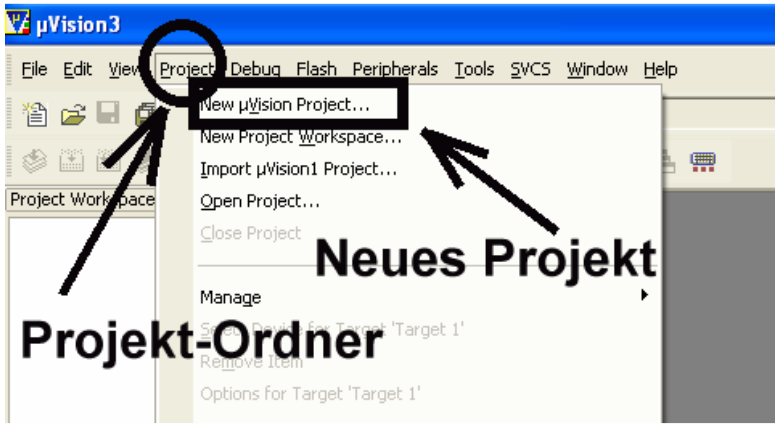
3.1. Einrichtung eines neuen C-Projektes

Zuerst ein Wort zur erforderlichen KEIL-C51-Demo-Software, die wir auf unserem Rechner installieren:

Diese Version „c51v8.09a“muss es MINDESTENS sein, denn erst ab diesem Zeitpunkt wurde unser AT89C51AC3 offiziell in die KEIL-Library aufgenommen!

Sie kann aus der KEIL-Homepage (www.keil.com) kostenlos heruntergeladen werden.

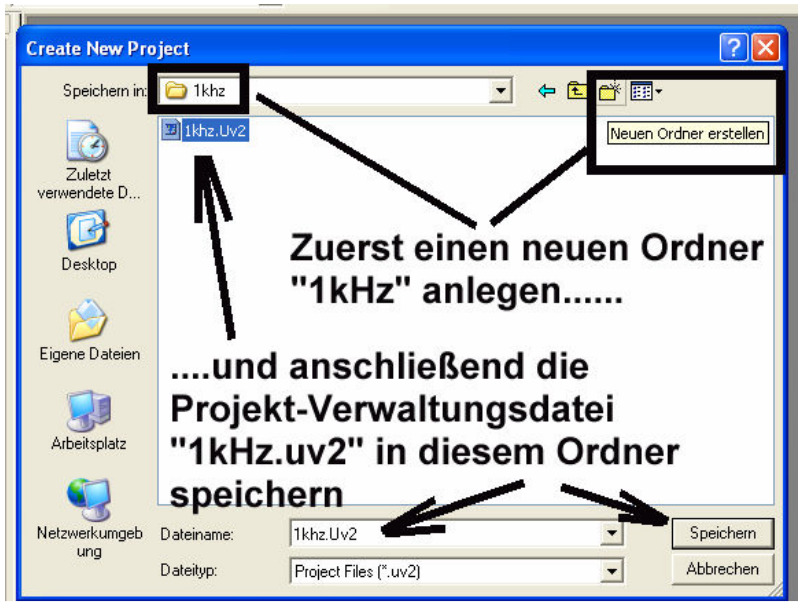
Die Software wird nun gestartet und wir gehen gleichauf den Projekt-Ordner los.



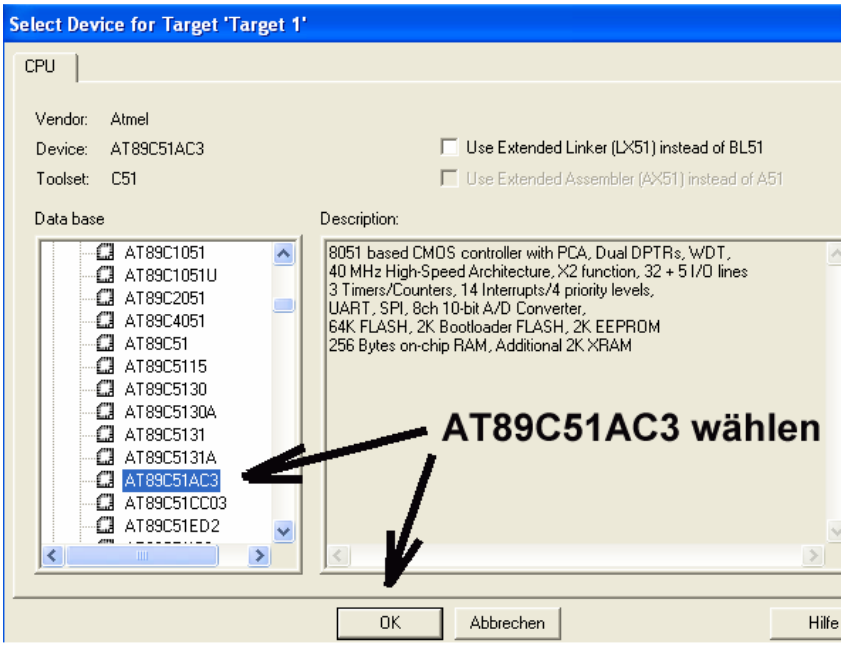
Ein eventuell noch geöffnetes anderes Projekt wird erst mal geschlossen und dann ein neues µvision-Projekt angelegt.

Wir wollen uns nun eine einfache Aufgabe stellen und dieses Projekt auf unserem Board zum Laufen bringen:

„Erzeugen Sie an Portpin P1^0 ein Rechtecksignal mit der Frequenz f = 1kHz“



1. Schritt:
Bitte zuerst an einem Ort Ihrer Wahl einen Ordner mit dem Namen „**1kHz**“ anlegen und darin die Projekt-Verwaltungsdatei „**1kHz.uv2**“ speichern..



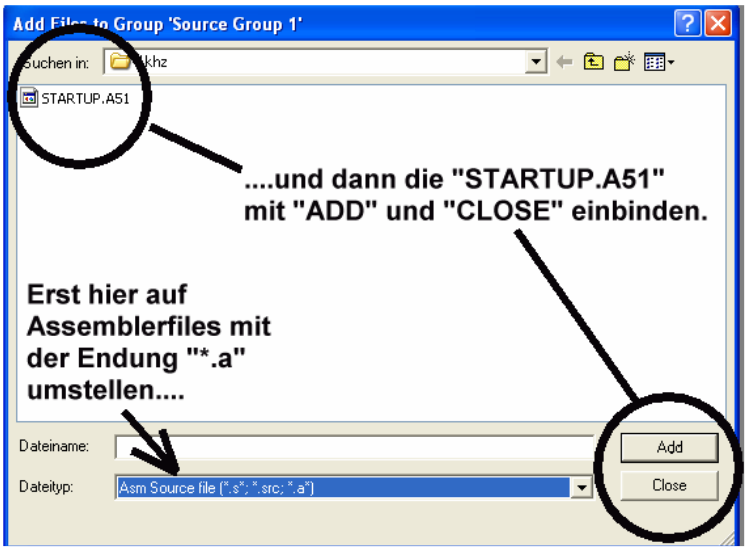
2. Schritt:
Aus der angebotenen Herstellerliste wählen wir „**Atmel**“, suchen nach unserem Controllertyp „**AT89C51AC3**“ und bestätigen ihn mit OK.

3. Schritt:
Dieser Frage wird mit

„**Ja**“

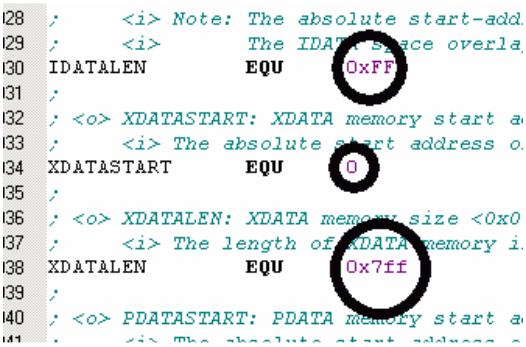
zugestimmt.

4. Schritt:
Nun beginnen wir, diese **„Startup.A51**“-Datei in unser Projekt einzubinden. Das ist der erste Teil….



5. Schritt:
…und das der zweite Teil.

6. Schritt.
So muss jetzt die Projektverwaltung aussehen!



7. Schritt:
Die Startup.A51-Datei wird geöffnet und darin werden die nebenstehenden drei Einträge kontrolliert bzw. vorgenommen:

IDATALENGTH muss auf 0xFF

XDATASTART muss auf 0 (oder: 0x00)

XDATALENGTH muss auf 0x7FF

stehen!

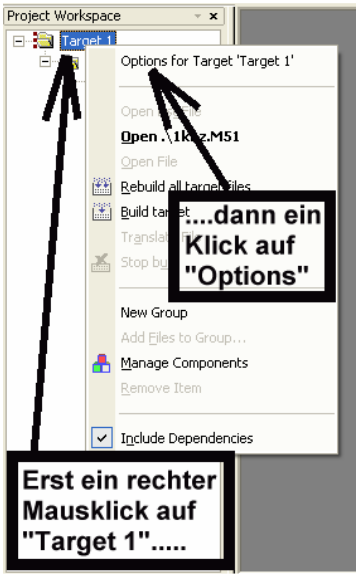
```
113                 NAME    ?C_STARTUP
114
115
116 ?C_C51STARTUP   SEGMENT   CODE
117 ?STACK          SEGMENT   IDATA
118
119                 RSEG    ?STACK
120                 DS      1
121
122                 EXTRN CODE (?C_START)
123                 PUBLIC  ?C_STARTUP
124
125                 CSEG    AT      0
126 ?C_STARTUP:     LJMP    STARTUP1
127
128                 RSEG    ?C_C51STARTUP
```

8. Schritt:
Weiter unten in der Datei wird bei **„?C_STARTUP“** geprüft, ob der Eintrag

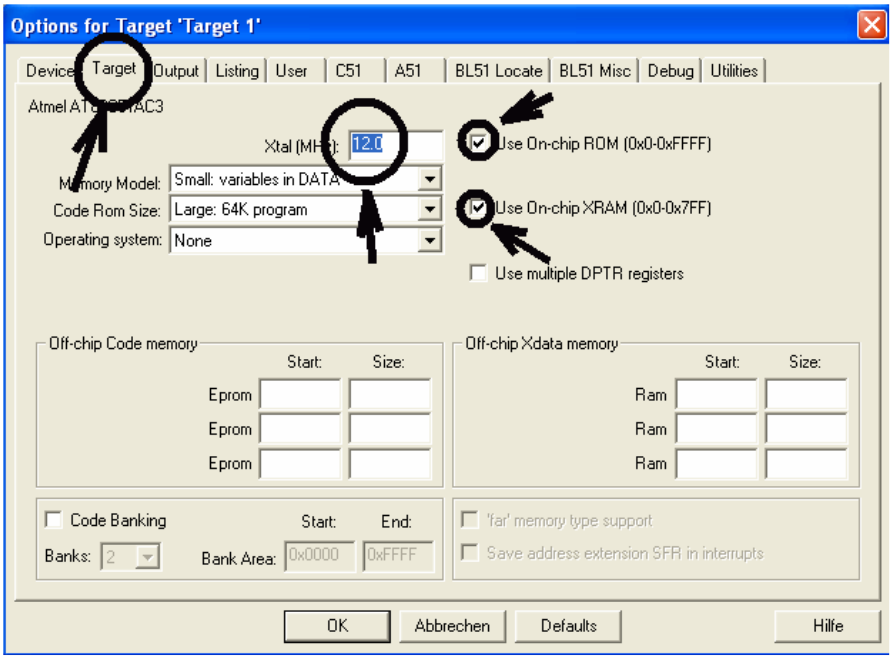
CSEG AT 0

stimmt.

Wenn ja, dann haben wir den ersten Teil geschafft!



9. Schritt:
In der nebenstehenden Reihenfolge (= rechter Klick auf „Target 1“ im Project Workspace, dann „Options“) holen wir uns nun das Menü für die **Grundeinstellungen** des Projektes auf den Schirm:



10. Schritt:

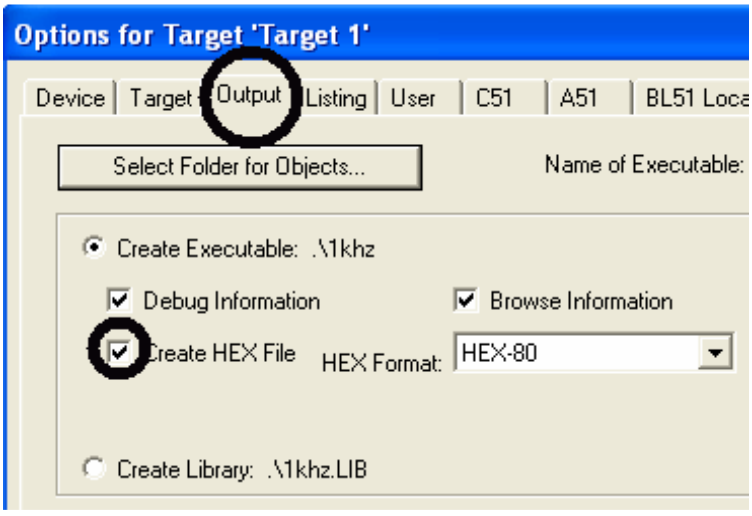
Auf der ersten Karteikarte „**Target**“ tragen wir ein:

Quarztakt (XTAL) = 12MHz

Use On-Chip-ROM

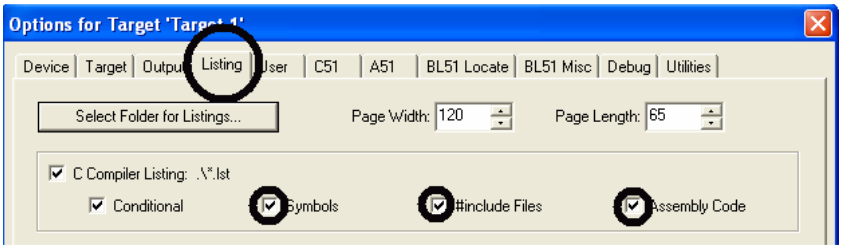
Use On-Chip XRAM

Der Rest der Einstellungen bleibt unverändert (Siehe nebenstehendes Bild).



11. Schritt:
Auf der Karteikarte „**Output**“ setzen wir ein Häkchen bei „**Create HEX File**“

12. Schritt:
Auf der Karteikarte „Listing“ werden

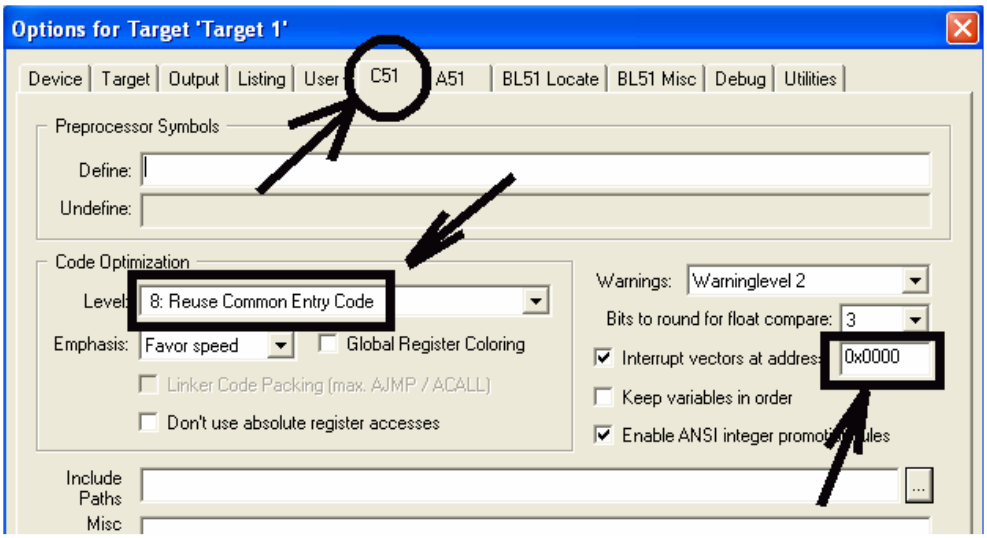


Symbols

#include Files

Assembly Code

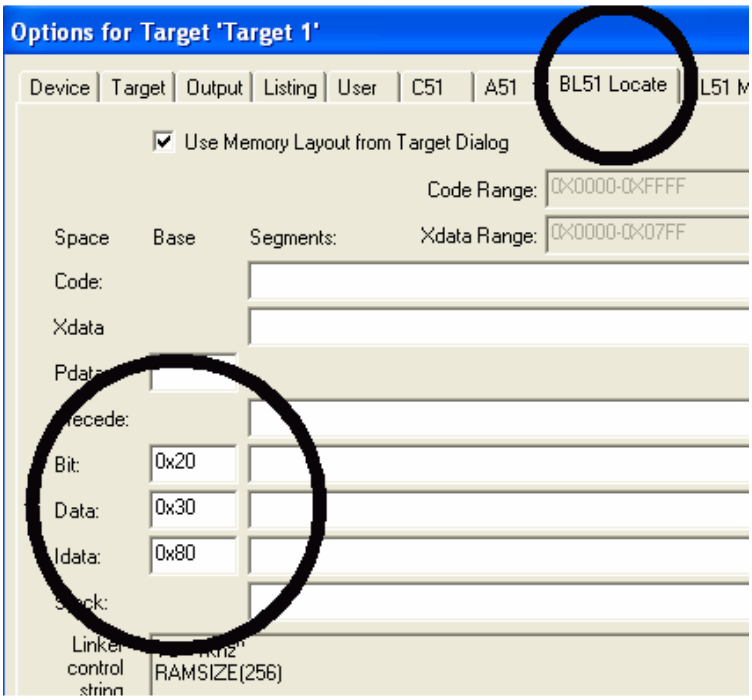
durch Häkchen aktiviert



13. Schritt:

Die Karteikarte „User“ überspringen wir!

Auf der nächsten Karteikarte „**C51**“ werden nur sorgfältig alle Einstellungen mit dem nebenstehenden Bild verglichen. Besonders wichtig ist die korrekte **Interrupt Vector-Einstellung mit 0x0000** und **Level 8** bei der **Optimierung**.



14. Schritt:

Die Karteikarte für „A51“ überspringen wir wieder.

Nun bleiben nur noch die Einstellungen für den **Linker („BL51 Locate“).**

Hier gilt:

Bit-Deklarationen ab Adresse 0x20

Data-Bereich ab Adresse 0x30

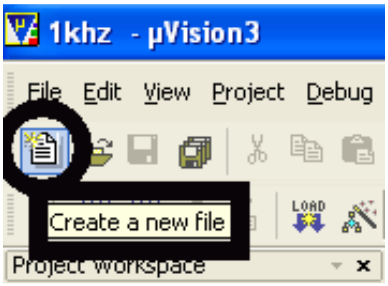
IDATA-Bereich ab Adresse 0x80

Wenn wir den Simulator zum Programmtest verwenden wollen, dann gilt bei der Karteikarte „Debug“ die untenstehende Einstellung.
(Bitte alle Häkchen kontrollieren und dann mit OK alles bestätigen).

Die letzte Karteikarte „Utilities“ benützen wir nicht und die vorhergehende Karteikarte „BL51 Misc.“ wird ebenfalls übergangen.

Damit sind wir mit den Grundeinstellungen fertig. Jetzt müssen wir unser C-Programm schreiben und es in unser Projekt einbinden.

3.2. C-Programm-Erstellung und Hex-File-Erzeugung



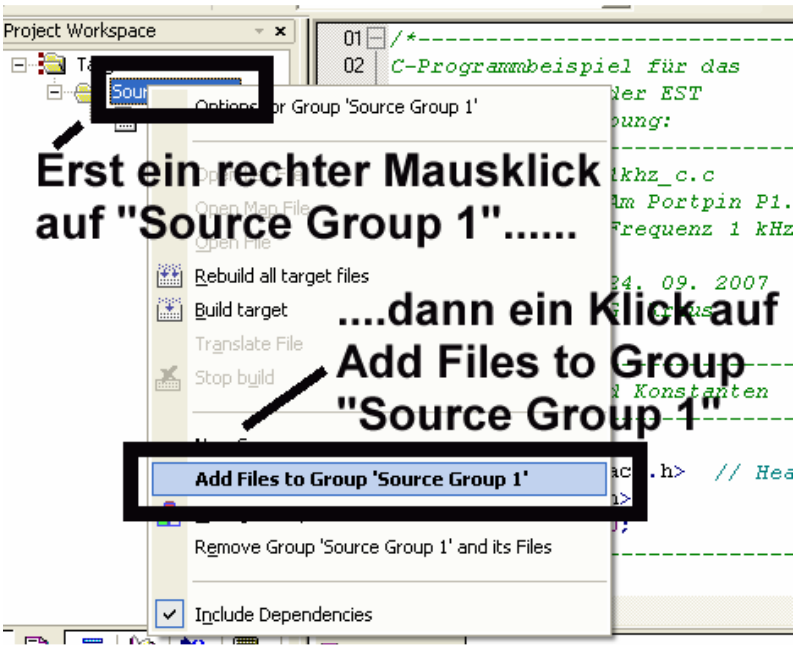
Wir schließen unsere Optionseinstellungen wieder und öffnen ein neues leeres File. Darin schreiben wir unser C-Programm für den Piepston.

Achtung:
Der von der Firma KEIL mitgelieferte Header „t89c51ac2.h“ war für das Vorgängermodell des verwendeten Controllers vorgesehen und deshalb „etwas unvollständig“ bei der Verwendung des AT89C51AC3.
Im Anhang dieses Manuskripts und auf der neuen ATMEL-Board – CD findet sich deshalb ein selbst geschriebener Header „AT89C51AC3.h“, der nun ALLE erforderlichen Deklarationen enthält.

Für die einfacheren Beispiele kann aber weiterhin der „normale“ Header „t89c51ac2.h“ aus der Keil-INC-Library verwendet werden.

So würde das Musterprogramm aussehen:

```
#include <t89c51ac2.h>    // Header für Controller "AT89C51AC3"
#include <stdio.h>

sbit ausgang=P1^0;        // Portpin P1.0 als Signalausgang

void zeit(void);          // Prototypen-Anmeldung

void main(void)
{
    while(1)                    // Endlos-Schleife
    {   zeit();                 // Delay von 500 Mikrosekunden
        ausgang=~ausgang;       // Bit komplementieren
    }
}

void zeit(void)                 // Wartezeit-Erzeugung
{   unsigned int x;
    for(x=0;x<=62;x++);     //Int-Wert 62 ergibt 500 Mikrosekunden
}
```

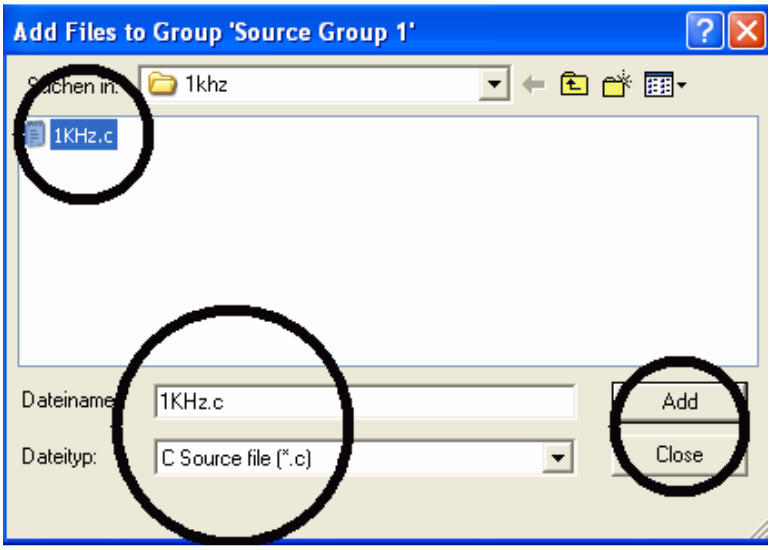


Nach der Fertigstellung speichern wir dieses File unter dem Namen

1kHz.c

in unserem **Projektordner „1kHz“.**

Anschließend muss dieses File in unser Projekt eingebunden werden. Der Vorgang sollte uns von der „Startup.A51“ bekannt vorkommen…



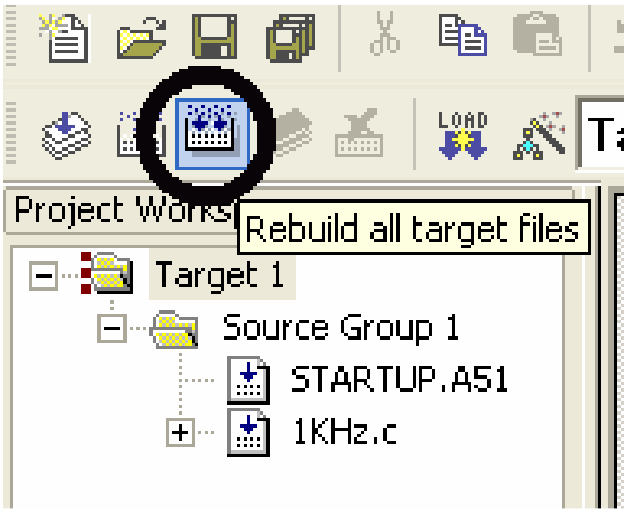
…Erst auf den Datentyp

„C Source file (*.C)“

umstellen, dann unser File **„1kHz.c“ auswählen** und dieses schließlich mit

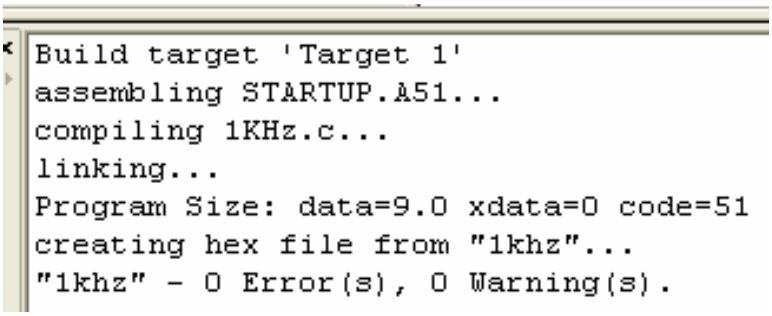
Add und Close

in das Projekt einbinden.



Links im Bild ist gut zu sehen, dass nun beide beteiligten Files („Startup.A51“ und „1khz.c“) korrekt im Projekt stehen.

Deshalb drücken wir nun den gekennzeichneten Knopf „**Rebuild all target files**“, um den Übersetzungsvorgang (= Assemblieren, Compilieren, Linken, Hex-File erzeugen..) zu starten.



Wenn diese Meldung

„0 Error(s), 0 Warning(s)“

erscheint, sind wir am Ziel.

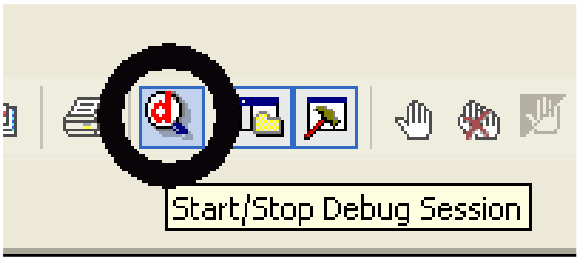
Nun haben wir die Wahl:
entweder wir simulieren zuerst unser Projekt noch mit KEIL

oder

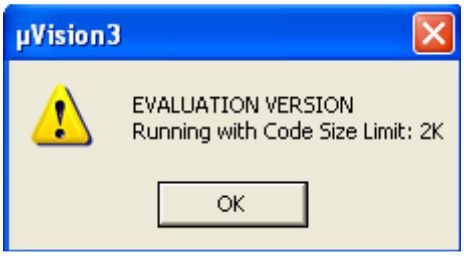
wir wechseln gleich zu „FLIP“, um das Programm in das EPROM unseres Controllerboards zu flashen und es direkt mit der Hardware auszutesten.

3.3. Programmsimulation mit dem KEIL-Debugger

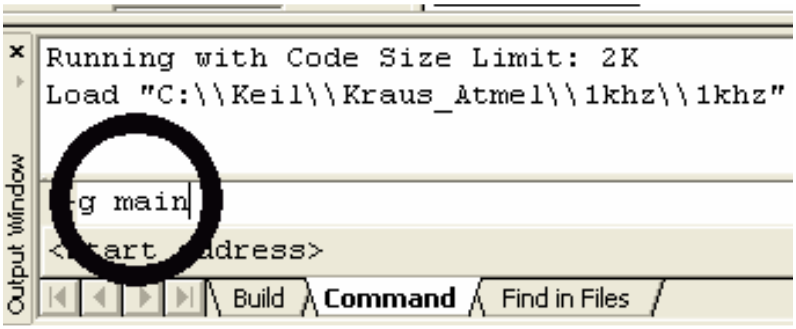
Wenn die **Target-Optionen auf „Simulation“ umgestellt** sind (Siehe Schritt 14 im Kapitel 3.1), ist das eine nette Sache und nicht schwierig:



Zuerst drücken wir den „**Debug-Button**“ rechts oben in der Menüleiste…



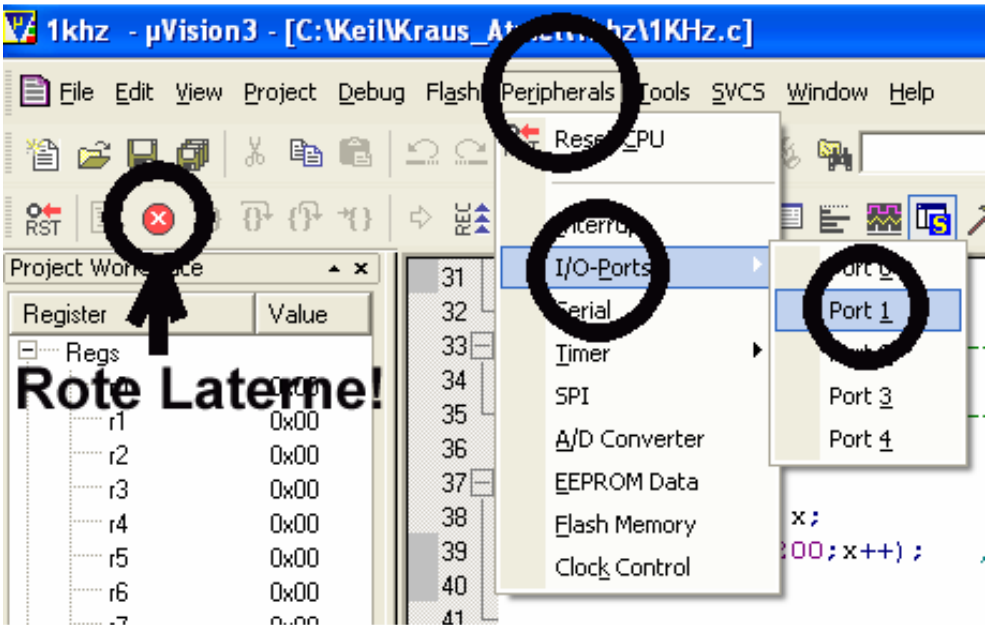
…und dann klicken wir bei der auftauchenden Meldung auf „**OK**“.



Unten links auf dem Bildschirm finden wir die „Kommandozeile“. Sobald man dort hineinklickt, blinkt der Cursor und wir tippen

g main

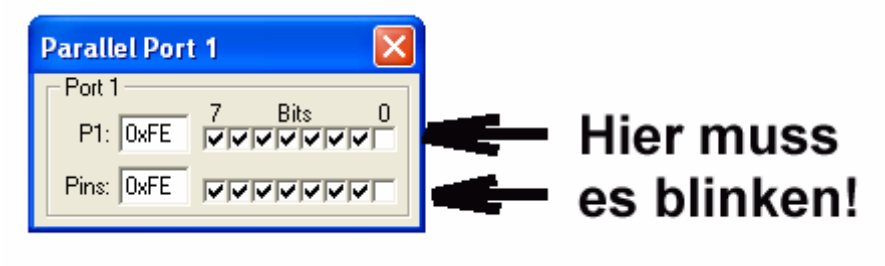
ein (..das bedeutet: „go to main“).. Nach Betätigen der „ENTER-Taste“ läuft bereits die Simulation, aber wir sehen noch nichts davon.



Das einzige Lebenszeichen der Simulation ist das Aufleuchten der „Roten Laterne“.

Also hangeln wir uns über „**Peripherals**“ und „**I/O-Ports**“ bis zu „**Port 1**“ und klicken darauf.

Das sollte der Erfolg sein:



Übrigens,
wer sich über die beiden Reihen von Kästchen für Port P1 wundert, möge bitte genau hinschauen:

Die obere Reihe der Kästchen stellt das Register von Port P1 im SFR-RAM-Bereich des Controllers dar.

Die untere Reihe der Kästchen entspricht dagegen den Anschlusspins von Port P1 am Controllergehäuse!

Jedes einzelne Kästchen lässt sich durch Klicken mit der Maus beeinflussen und das entspricht dem Setzen oder Löschen des entsprechenden Bits.

Und noch eine Info:
Durch einen Klick auf die Rote Laterne wird die **Simulation gestoppt**, ein weiterer Klick auf den **Debug-Button** führt zum **Verlassen des Debuggers**. Damit kehren wir wieder zu unserem Ausgangspunkt, also dem Editor, zurück.

3.4. Flashen mit FLIP

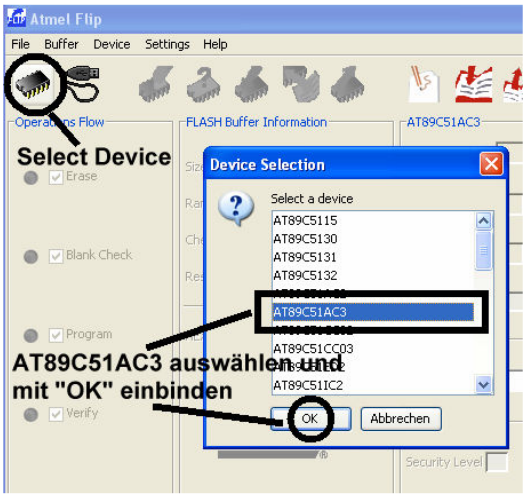
Vorbereitungen:

a) Das Board wird mit einer Gleichspannung von **+8….+10V** versorgt (Bitte nicht wesentlich überschreiten, sonst wird es dem Festspannungsregler auf der Platine zu warm). Wahlweise kann direkt eine Gleichspannung eingespeist **ODER** ein Steckernetzteil (z. B. AC-Adapter) verwendet werden. Passende Stecker sind auf dem Board vorhanden (Siehe Foto des Boards in der Einführung)!

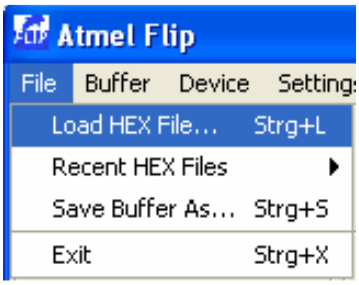
b) Die COM-Schnittstellen des PCs und des Boards werden über ein „**Null-Modem-Kabel**“ (= Crossed Link Cable) verbunden.

c) Der Programmierschalter auf dem Board wird auf „**Flash**“ eingestellt -- die zugehörige rote LED auf dem Board muss leuchten.

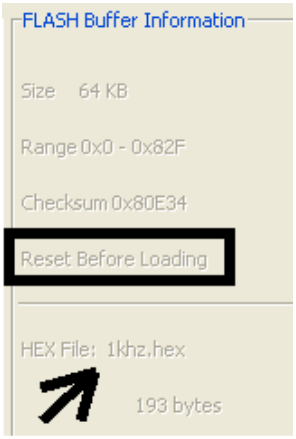
d) Nun wird FLIP gestartet.



e) Der AT89C51AC3 wird zuerst über „**Select Device“** in nebenstehender Reihenfolge eingebunden.



f) Über „**File**“ kommen wir an „**Load HEX File**“ heran, klicken darauf und suchen nach unserem File „**1kHz.hex**“ im zugehörigen **Projektordner „1kHz**“, den wir ja selbst angelegt haben. Das wird geöffnet…



g) …und der Erfolg lässt sich sofort in der Mitte des FLIP-Bildschirms kontrollieren. Allerdings ist das Ganze recht unscheinbar und kaum zu erkennen, da diese Informationen erst später vor dick markiert werden…

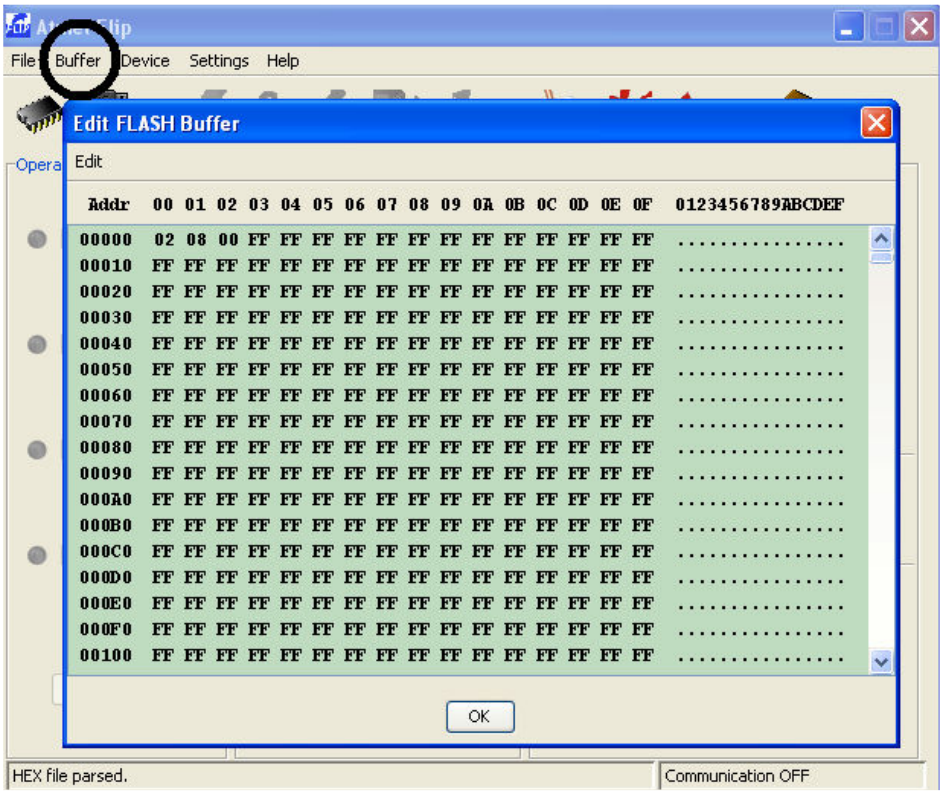
Achtung: genau so unauffällig steht hier eine wichtige Information, ohne die wir anschließend Probleme bei der Flasherei hätten:

Reset before Loading

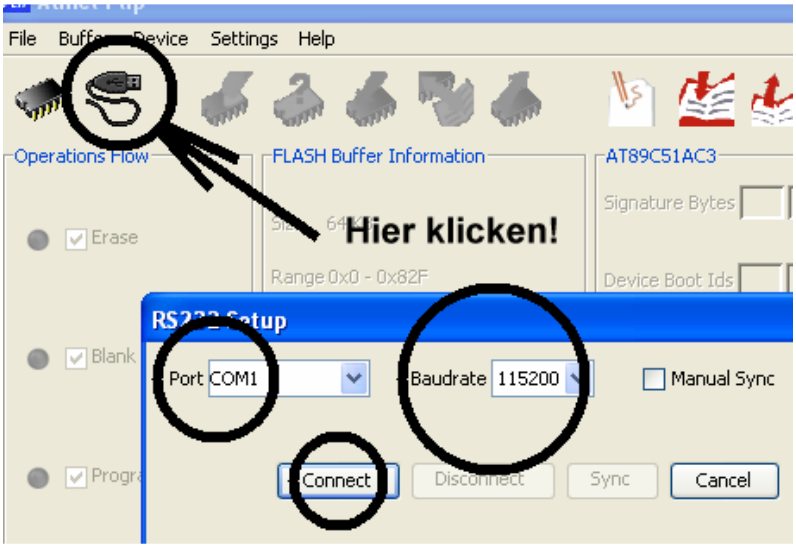
Das müssen wir beim nächsten Schritt unbedingt beherzigen.

Übrigens:

Wer den Knopf „**Buffer**“ und dann „**Edit**“ drückt, bekommt etwas besonders Hübsches zu sehen, nämlich die übertragenen Maschinencodes im Hex-File einschließlich der zugehörigen Speicherplatzadressen. Und zusätzlich werden noch erkannte ASCII-Zeichen ausgegeben…
Bitte mal testen!



So wird das aussehen und damit kann man genau die Speicherbelegung untersuchen!



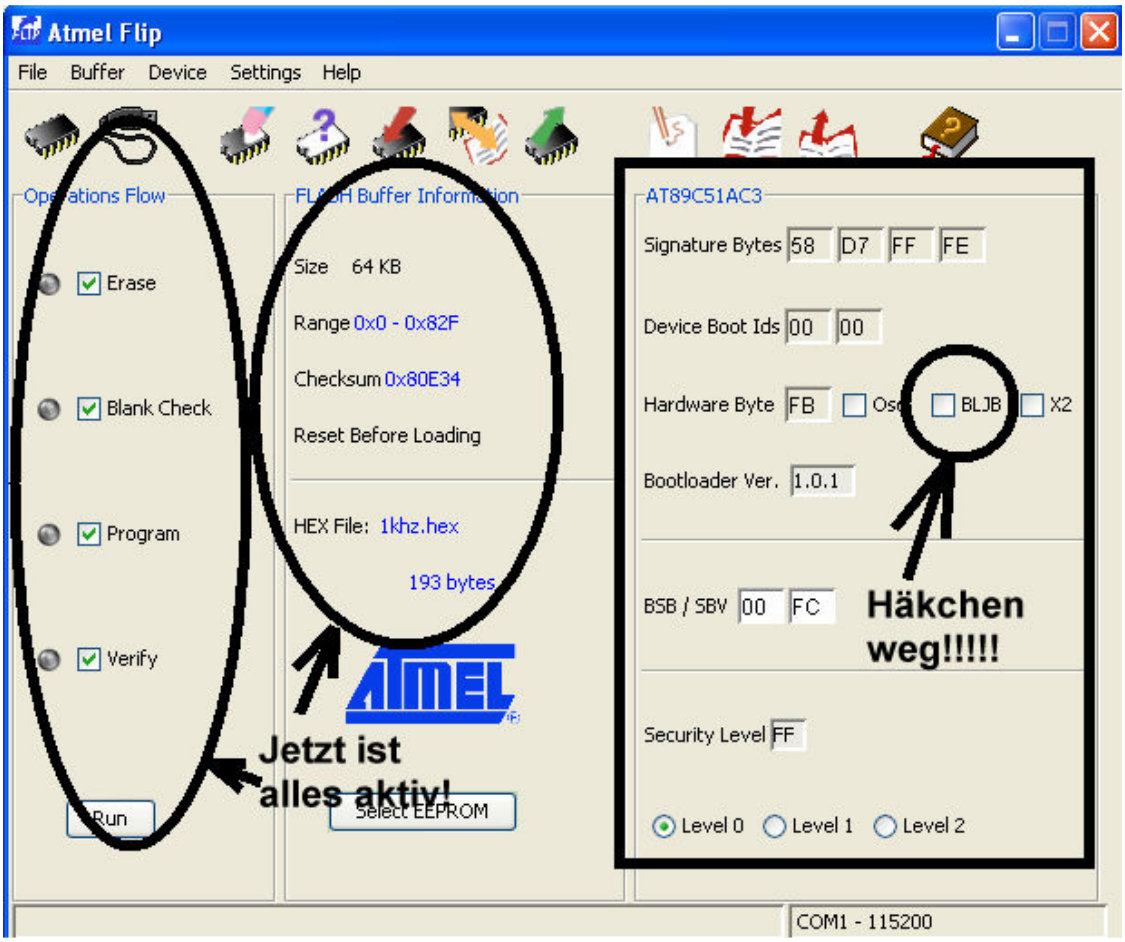
h) Nun klicken wir erst auf den **Kommunikations-Button** („……mit dem Schnittstellenkabel drauf….) und wählen „**RS232**“. Jetzt sind wir im nebenstehenden Setup-Menu, kontrollieren den **COM-Port** und wählen eine **passende Baudrate.**

Bitte daran denken:
Mit einem 12 MHz-Quarz dürfen wir höchstens mit 9600 Baud arbeiten.
Setzt man dagegen den 11,059 MHz-Quarz in die Fassung, dann sind sogar die maximalen 115200 Baud zulässig!

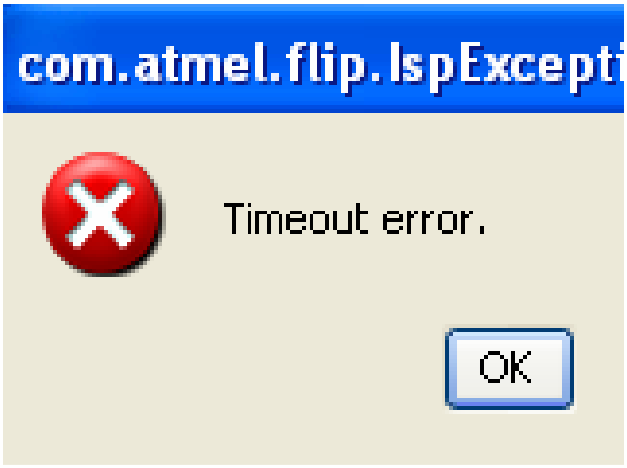
Ist das geklärt, dann drücken wir kurz auf den Reset-Button des Controllerboards und beim einem abschließenden Klick auf „Connect“ sollte -- wenn alles klappt -- die Verbindung zwischen FLIP und dem Board hergestellt sein.

Da gibt es nun natürlich zwei Möglichkeiten:

Wenn die Verbindung steht, sieht der Bildschirm so aus:



Gleich prüfen: Ist wirklich das Häkchen bei BLJB (= Boot Loader Jump Bit) entfernt? Das ist lebenswichtig, denn nur dann startet die in den Controller geflashte Application automatisch beim Reset oder Einschalten…



Wenn es dagegen nicht geklappt hat, erkennen wir das an dieser Meldung.
Da wiederholt man nochmals den Prozess und prüft:

a) Steht der **Board-Schalter** auf „**Flash**“ und leuchtet die rote Programmier-LED?

b) Sind alle **Schnittstellenkabe**l am Board, PC und USB-Converter eingesteckt?

c) Stimmt der **COM-Port?**

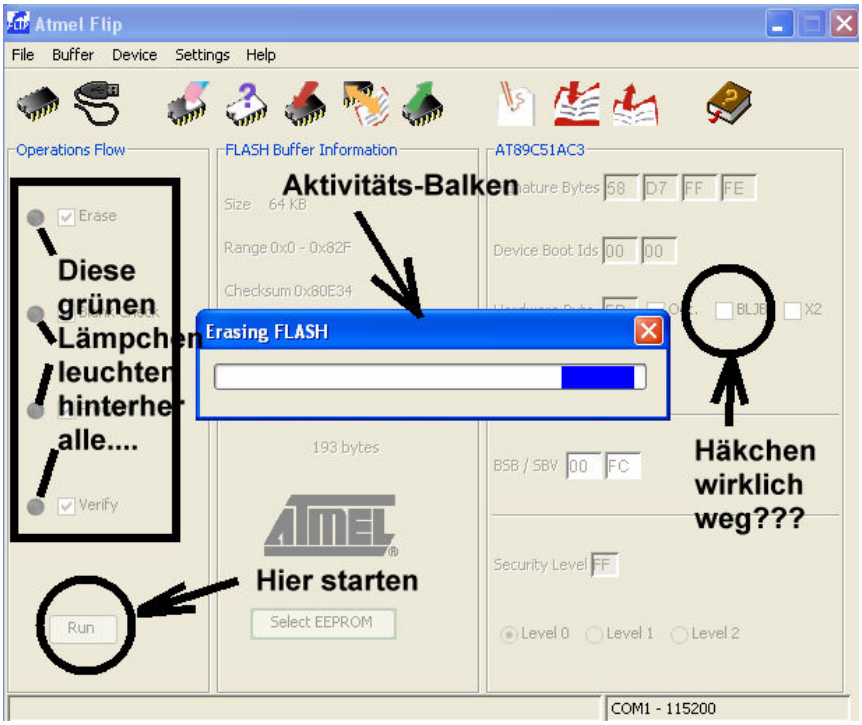
d) Ist die **Baudrate viel zu hoch** eingestellt (z. B. **max. 9600**, weil ein Quarz mit 12MHz statt 11,0592MHz benützt wird)?

e) Hilft eine **Reduktion der Baudrate auf 2400** ?

Und bitte nie vorher den Reset beim Board vergessen!

Bei hartnäckigen Fällen empfiehlt die ONLINE-Hilfe ein andere Reihenfolge der Schritte für Prozedur h) (…offensichtlich gab es da Laptops, die Ärger machten….):
Erst den Kommunikationsbutton drücken, den COM-Port prüfen und die Baudrate auf max. 9600 stellen. Dann das Häkchen bei „**Manual Sync**“ setzen und auf „**Connect**“ klicken. Nun ein- oder zweimal auf die Reset-Taste des Boards drücken und anschließend den Button „**Sync**“ anklicken.

Zum Trost: Irgendwann klappt es immer und dann ist das Ziel wirklich nahe:



Zuerst kontrollieren wir nochmals sehr sorgfältig, ob **bei „BLJB“ das Häkchen entfernt ist**. Es handelt sich hier um das sehr wichtige

„Boot Loader Jump Bit“

und das **muss gelöscht** sein.

Nur wenn das der Fall ist, kann das Controllerboard nach einem Reset (= Betätigen der Reset-Taste ODER nach dem Einschalten der Betriebsspannung) das in das EPROM geflashte Programm abarbeiten!!

Ist das erledigt, dann klicken wir auf „**RUN**“ (links unten im Bild) und sehen dem PC bei der Arbeit zu (..dazu dient der Aktivitätsbalken). Sobald er fertig ist, leuchten alle vier Lämpchen (von „Erase bis „Verify“) grün auf.

Jetzt kann man beim Controllerboard mit dem Schiebeschalter von „Flash Mode“ auf „RUN“ umschalten -- dann geht die rote LED aus. Ein Druck auf die Reset-Taste des Boards (oder ein Aus- und Einschalten der Versorgungsspannung) startet endlich das Controllerprogramm.

Bei Programm-Änderungen oder bei der Fehlersuche geht man einfach zu KEIL µvision3 zurück, korrigiert die Fehler und erzeugt schließlich ein **neues Hex-File**. Das lädt das erneut in der eben besprochenen Weise ins Flash-EPROM und startet nochmals die Anwendung.

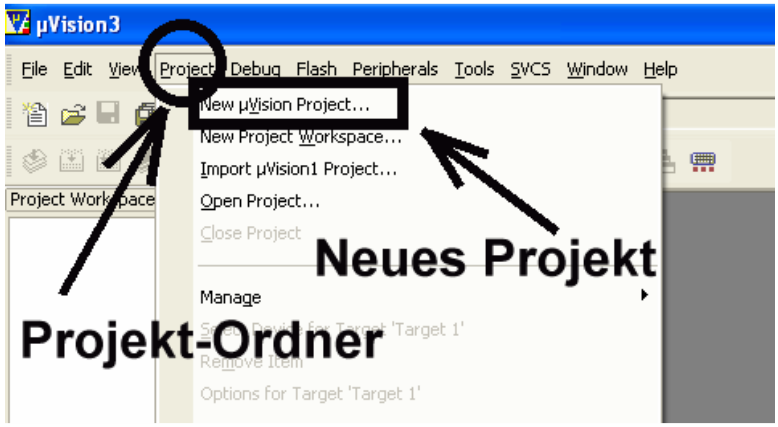
4 Assembler-Programmierung mit KEIL µvision3 und ATMEL „FLIP“
4.1. Einrichtung eines neuen Assembler-Projektes

Zuerst ein Wort zur erforderlichen KEIL-C51-Demo-Software, die wir auf unserem Rechner installieren:

Diese Version muss es MINDESTENS sein, denn erst ab diesem Zeitpunkt wurde unser AT89C51AC3 offiziell in die KEIL-Library aufgenommen!

Sie kann aus der KEIL-Homepage (www.keil.com) kostenlos heruntergeladen werden.

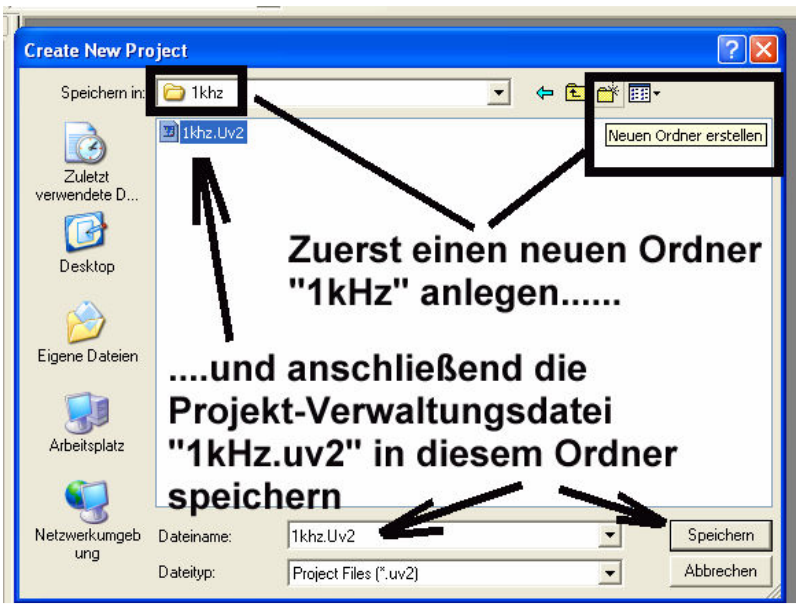
Die Software wird nun gestartet und wir gehen gleichauf den Projekt-Ordner los.



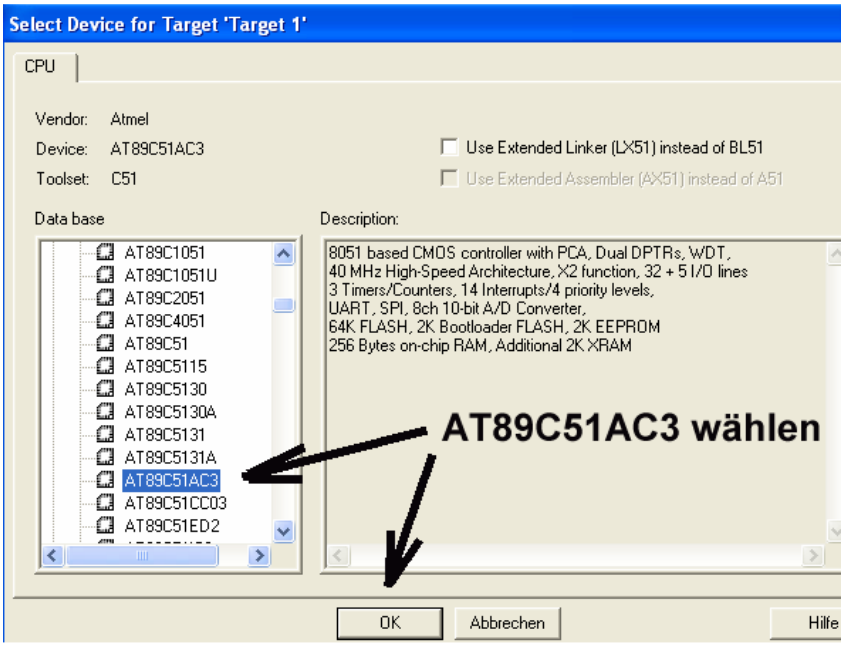
Ein eventuell noch geöffnetes anderes Projekt wird erst mal geschlossen und dann ein **neues µvision-Projekt** angelegt.

Wir wollen uns nun eine einfache Aufgabe stellen und dieses Projekt auf unserem Board zum Laufen bringen:

„Erzeugen Sie an Portpin P1^0 ein Rechtecksignal mit der Frequenz f = 1kHz“



1. Schritt:
Bitte an einem Ort Ihrer Wahl einen **Ordner mit dem Namen „1kHz“** anlegen und darin die Projekt-Verwaltungsdatei „**1kHz.uv2**“ speichern..

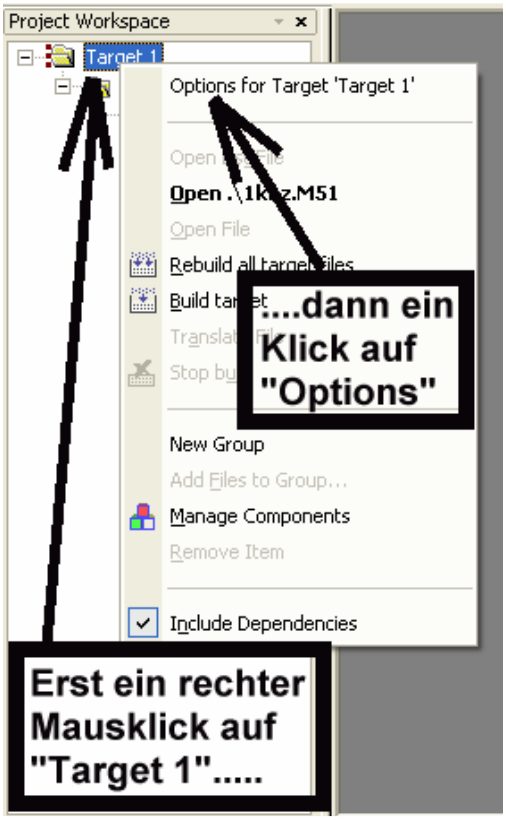


2. Schritt:
Aus der angebotenen Herstellerliste wählen wir „Atmel“, suchen nach unserem Controllertyp „**AT89C51AC3**“ und bestätigen ihn mit OK.

3. Schritt:
Diese folgende Frage wird bei Assemblerprogrammen immer mit

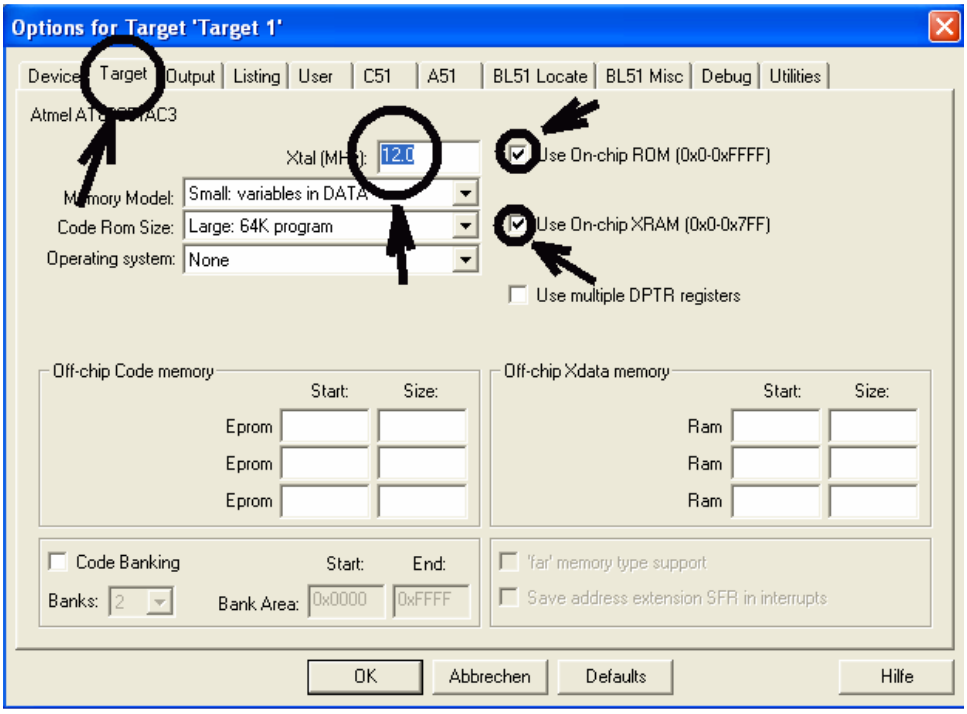
„Nein“

beantwortet.



4. Schritt:
Nun können wir ein neues Projekt einrichten.

In der nebenstehenden Reihenfolge (= Klick auf „Target 1“ im Project Workspace, dann „Options“) holen wir uns dazu das Menü für die Grundeinstellungen des Projektes auf den Schirm:



5. Schritt:

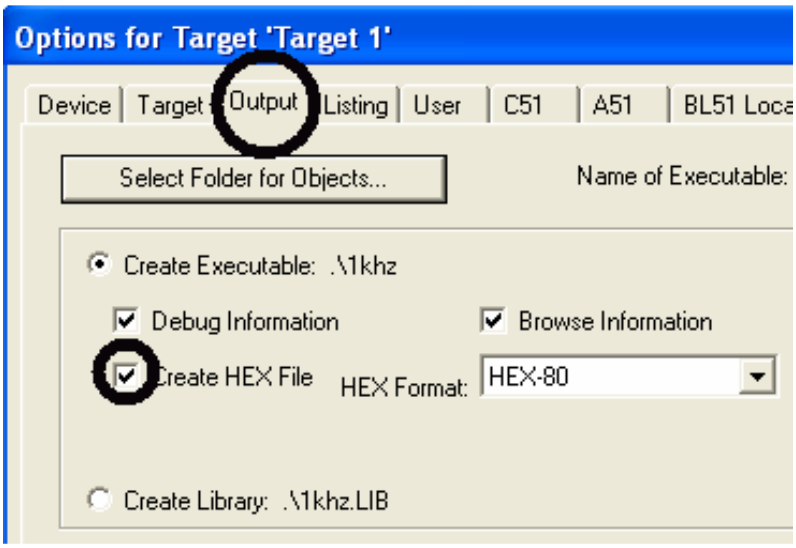
Auf der ersten Karteikarte „Target“ tragen wir je nach benütztem Quarz ein:

Quarztakt (XTAL) = 12MHz oder 11,0592MHz

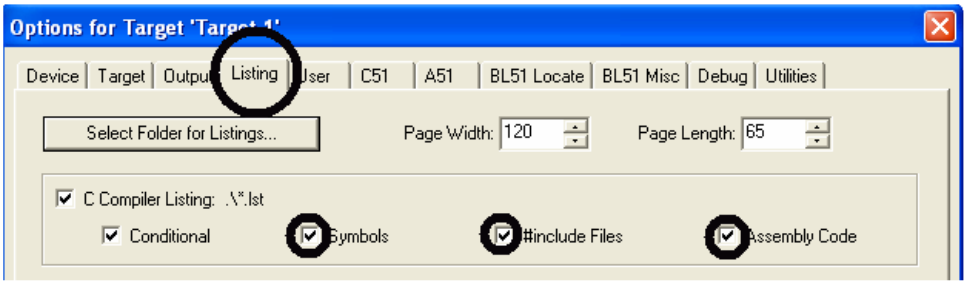
Use On-Chip-ROM

Use On-Chip XRAM

Der Rest der Einstellungen bleibt unverändert (Siehe nebenstehendes Bild).



6. Schritt:
Auf der Karteikarte „Output“ setzen wir ein Häkchen bei „**Create HEX File“**



7. Schritt:

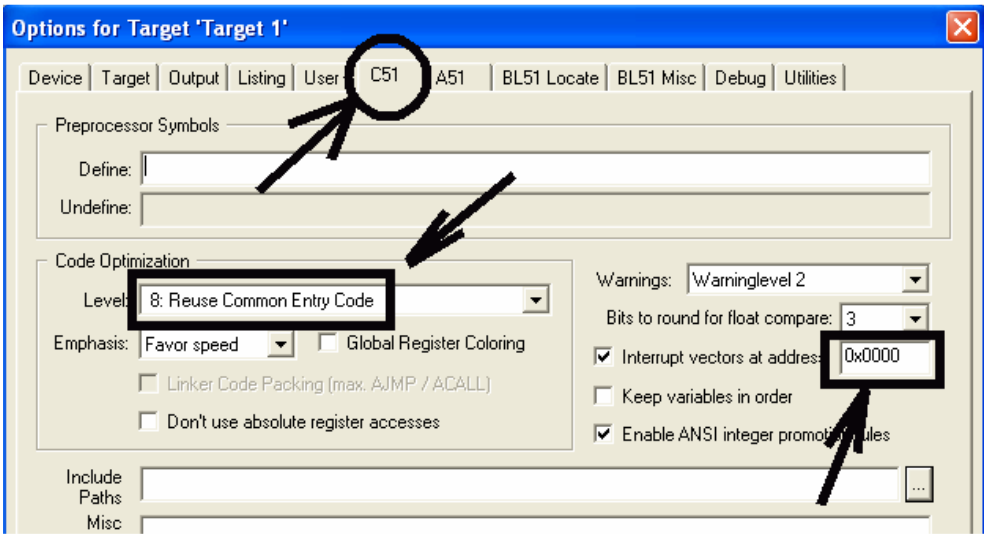
Auf der Karteikarte „Listing“ werden

Symbols

#include Files

Assembly Code

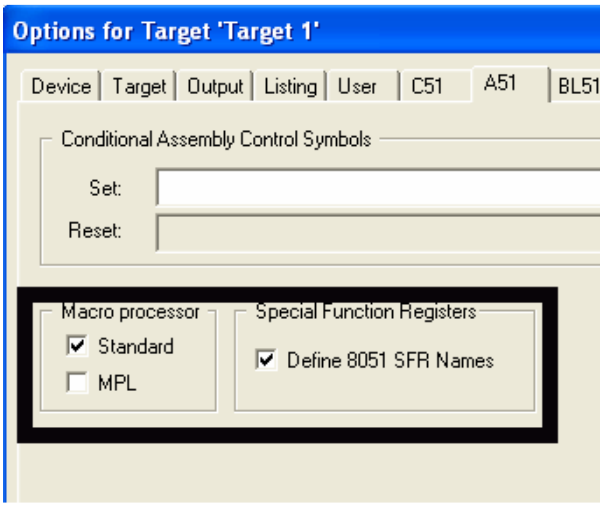
durch Häkchen aktiviert



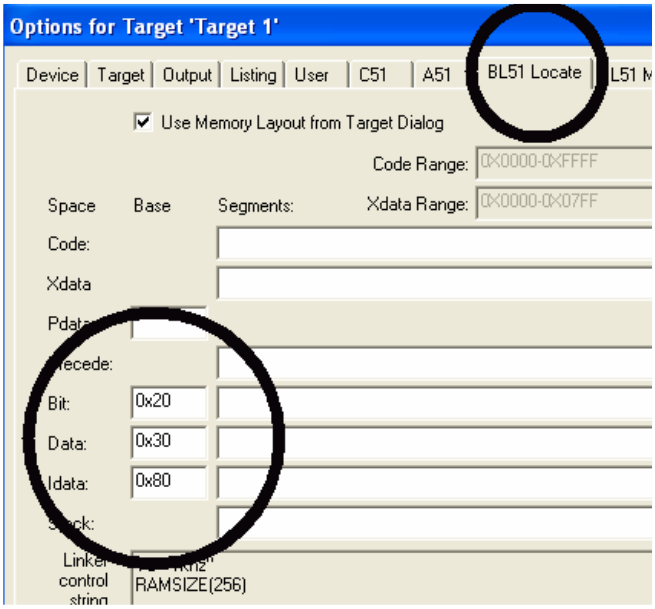
8. Schritt:

Die Karteikarte „User“ überspringen wir!

Auf der Karteikarte „**C51**“ werden nur sorgfältig alle Einstellungen mit dem nebenstehenden Bild verglichen. Besonders wichtig ist die korrekte **Interrupt Vector-Einstellung mit 0x0000** und **Level 8** bei der **Optimierung**.



9. Schritt:
Die Karteikarte „A51“ vergleichen wir mit der nebenstehenden Vorgabe und prüfen, ob die Häkchen richtig gesetzt sind.



10. Schritt:

Nun bleiben nur noch die Einstellungen für den **Linker („BL51 Locate“).**

Hier gilt:

Bit-Bereich ab Adresse 0x20

Data-Bereich ab Adresse 0x30

IDATA-Bereich ab Adresse 0x80

11. Schritt

Wenn wir den Simulator zum Programmtest verwenden wollen, dann gilt bei der Karteikarte „Debug“ die nebenstehende Einstellung.
(Bitte alle Häkchen kontrollieren und dann mit OK alles bestätigen).

Die letzte Karteikarte „Utilities“ benützen wir nicht und die vorhergehende Karteikarte „BL51 Misc.“ wird ebenfalls übergangen.

Damit sind wir mit den Grundeinstellungen fertig. Jetzt müssen wir unser Assembler-Programm schreiben und es noch in unser Projekt einbinden.

4.2. Assemblerprogramm-Erstellung und Hex-File-Erzeugung

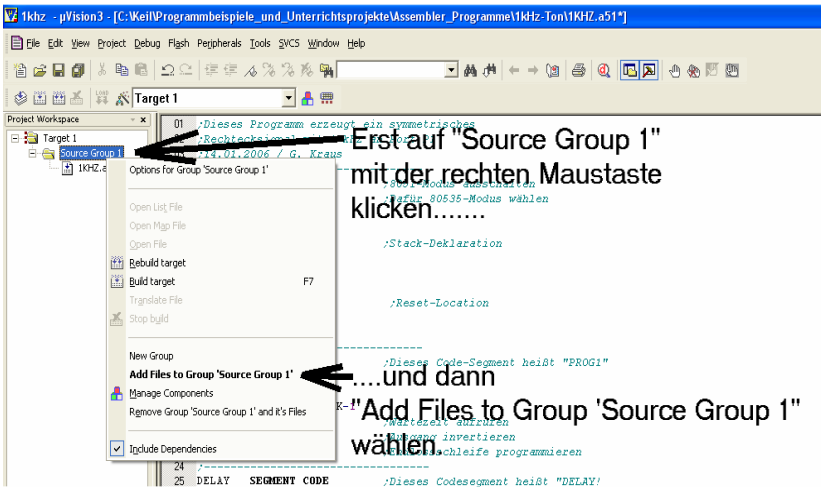
Wir klicken auf den bekannten WINDOWS-Button „**New File**“ und schreiben darin die folgende Programmdatei. (Die Kommentare hinter den Strichpunkten können auch weggelassen werden. Aber im eigenen Interesse und aus Erfahrung heraus kann nur dringend dazu geraten werden, bei neuen Projekten JEDE Programmzeile zu kommentieren. 3 Monate später ist man beim Nachgucken froh darüber....)

Hinweis: das folgende Assembler-Programm-File sollte man beim nächsten Projekt in den neuen Projektordner hineinkopieren, den Namen ändern und den Inhalt anpassen…..geht am schnellsten!

Wer sich über den „Organisationsaufwand“ beim Schreiben eines solchen Assemblerfiles im Gegensatz zu einem C-Programm wundert, dem sei gesagt: dies ist die offizielle Methode bei der Assemblerprogrammierung in der Industrie und sie wird z. B. in ALLEN mitgelieferten Beispielen der KEIL-Software verwendet!

```
;Dieses Programm erzeugt ein symmetrisches
;Rechtecksignal mit 1 kHz an Portpin P1.0
;14.10.2007 / G. Kraus
;-------------------------------------------------------------------------------------------------
$nomod51                        ; 8051-Modus ausschalten
$include(t89c51ac2.inc)         ; Dafür AT89C51AC3 verwenden

?STACK  SEGMENT IDATA           ; Stack-Deklaration
        RSEG ?STACK             ; Aufruf dieses IDATA-Segments
        DS 5                    ; define a Store with 5 bytes

CSEG at 0                       ; Reset-Location
        jmp START               ; jump to label “START”
;-------------------------------------------------------------------------------------------------
PROG1   SEGMENT CODE            ; Dieses CODE-Segment heißt "PROG1"
        RSEG PROG1              ; Aufruf dieses CODE-Segments, um es nun zu schreiben

START:  mov SP,#?STACK-1        ; Anwerfen des Stacks
loop1:  call zeit               ; Wartezeit aufrufen
        cpl p1.0                ; Ausgangspin P1.0 invertieren
        jmp loop1               ; Endlos-Schleife programmieren
;-------------------------------------------------------------------------------------------------
DELAY   SEGMENT CODE            ; Dieses CODE-Segment heißt "DELAY“
        rseg DELAY              ; Aufruf dieses CODE-Segments, um es zu schreiben

zeit:   mov r7,#250             ; Lade Register R7 mit der Zahl 250
        djnz r7,$               ; Dekrementiere R7 bis auf Null
        ret                     ; Zurück zum Hauptprogramm

end
```

Ist alles fertig, dann wird dieses Blatt mit „SAVE AS“ im Ordner „1kHz“ unter dem Namen „1kHz.A51“ gespeichert und anschließend in das Projekt eingebunden. Dann geht es los:



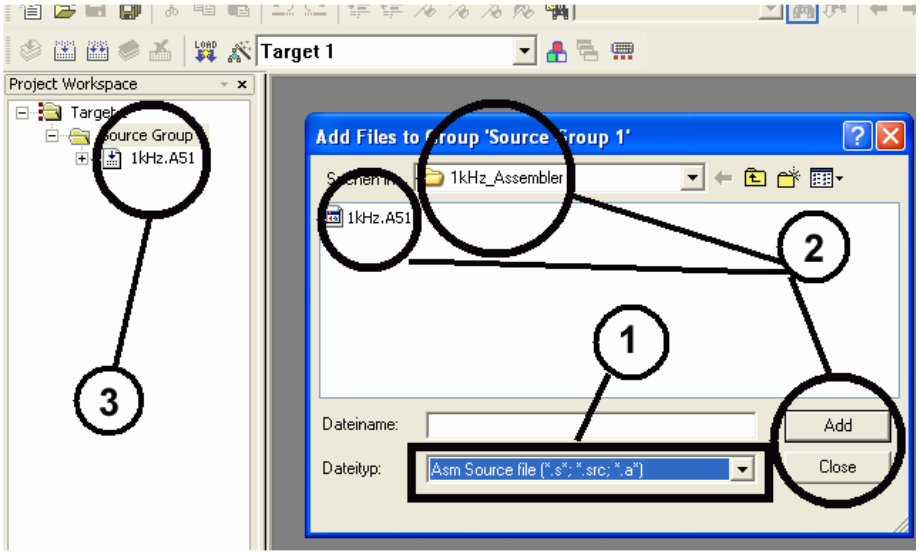
1. Schritt:
Nach dem Abspeichern dieser selbst geschriebenen Assemblerdatei „**1kHz.a51**“ geht es mit einem rechten Mausklick auf **Source Group 1** weiter**.**

Im Pulldown-Menü muss nun

Add Files to Group ‘Source Group1’

angeklickt werden.

2. Schritt:



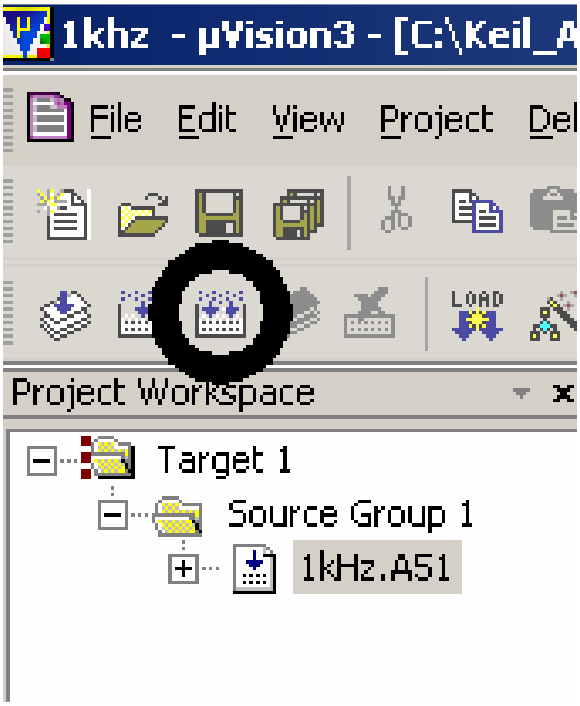
Wir ändern zuerst die Dateityp-Anzeige auf „**ASM Source File**“

Die eben geschriebene Datei „**1kHz.A51**“ wird dann durch Anklicken markiert, mit „**ADD**“ eingebunden und schließlich mit „**CLOSE**“ das Menü wieder verlassen.

Im linken Datei-Verwaltungsfeld kann man jetzt prüfen, ob diese Übernahme korrekt erfolgt ist und die Datei in der Source File List auftaucht.

WARNUNG:
Bei Assemblerprogrammen bitte NIEMALS die „Startup.A51“-Datei einbinden, das gibt sonst Probleme.

Grund: alle nötigen Speicherplatz-Zuweisungen müssen nun im Assemblerfile vorgenommen werden -- und das beißt sich mit der Startup.A51!

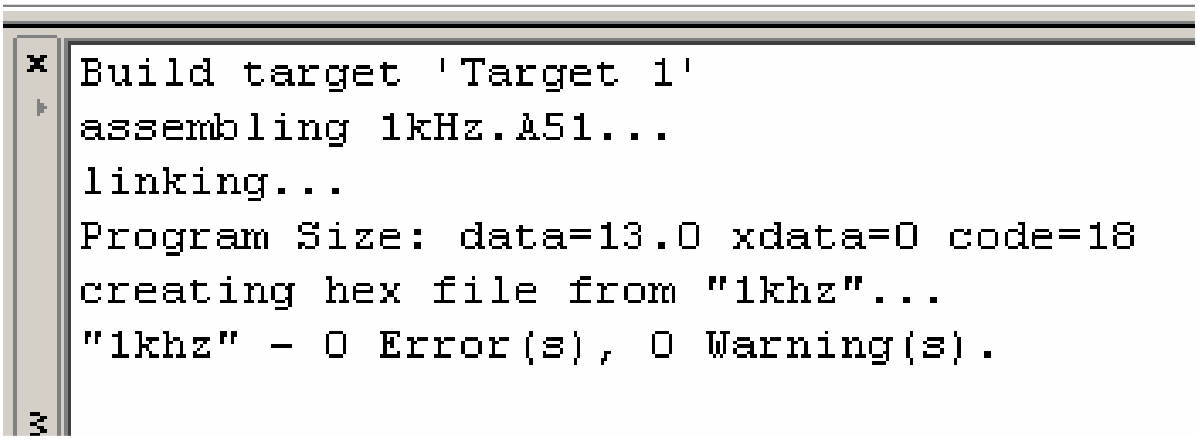


3. Schritt:
Ein Klick auf den gekennzeichneten

„Build All“-Button

startet den **Assembliervorgang, den Link-Vorgang und die Hex-File-Erzeugung.**

4. Schritt:
Nun wird ein scharfer Blick in das „Message-Fenster“ am unteren Bildrand geworfen. Wenn diese Meldung auftaucht, haben wir es geschafft!.



Das Hex-File ist damit produziert und wir können zum Test schreiten!

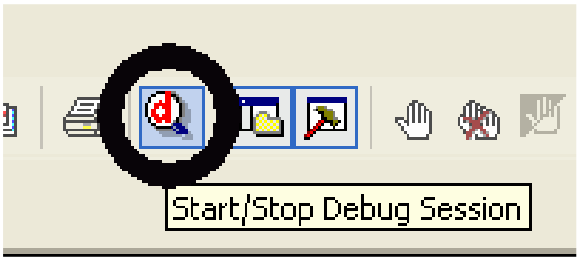
Nun haben wir die Wahl:
entweder wir simulieren zuerst unser Projekt noch mit KEIL

oder

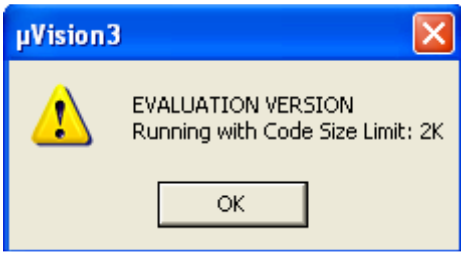
wir wechseln gleich zu „FLIP“, um das Hex-File in das EPROM unseres Controllerboards zu flashen und es direkt mit der Hardware auszutesten.

4.3. Programmsimulation mit dem KEIL-Debugger

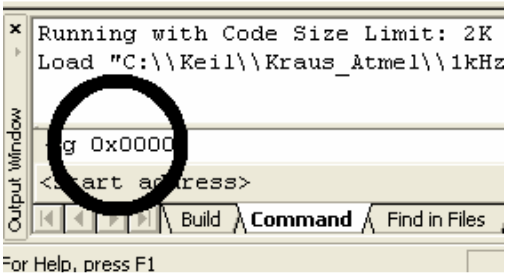
Wenn die Target-Optionen auf „Simulation“ umgestellt sind (**Siehe Schritt 11 im Kapitel 4.1**), ist das eine nette Sache und nicht schwierig:



Zuerst drücken wir den „Debug-Button“ rechts oben in der Menüleiste…



…und dann klicken wir bei der auftauchenden Meldung auf „OK“.

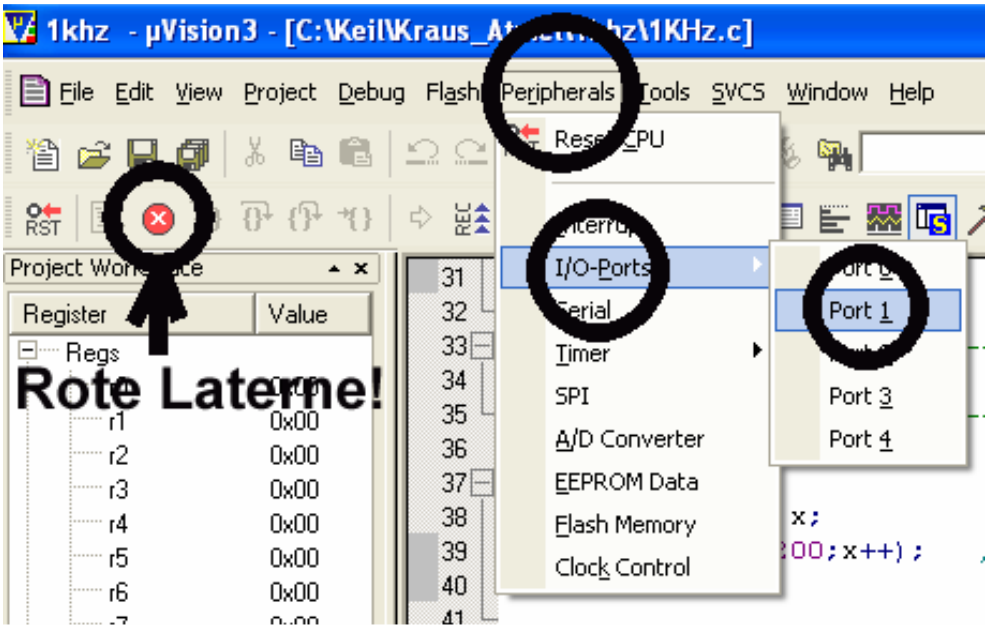


Unten links auf dem Bildschirm finden wir die „Kommandozeile“. Sobald man dort hineinklickt, blinkt der Cursor und wir tippen

g 0x0000

ein (..das bedeutet: „**go to ROM-Adress 0x0000**“)..

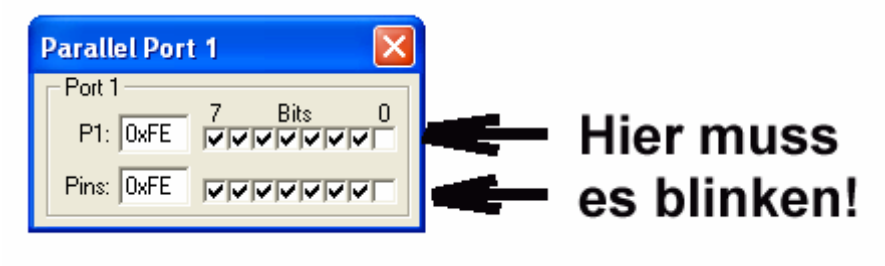
Nach Betätigen der „ENTER-Taste“ läuft bereits die Simulation, aber wir sehen noch nichts davon.



Das einzige Lebenszeichen der Simulation ist das Aufleuchten der „**Roten Laterne**“.

Also hangeln wir uns über „**Peripherals**“ und „**I/O-Ports**“ bis zu „**Port 1**“ und klicken darauf.

Das sollte der Erfolg sein:



Übrigens,
wer sich über die beiden Reihen von Kästchen für Port P1 wundert, möge bitte genau hinschauen:

Die obere Reihe der Kästchen stellt das Register von Port P1 im SFR-RAM-Bereich des Controllers dar.

Die untere Reihe der Kästchen entspricht dagegen den Anschlusspins von Port P1 am Controllergehäuse!

Jedes einzelne Kästchen lässt sich durch Klicken mit der Maus beeinflussen und das entspricht dem Setzen oder Löschen des entsprechenden Bits.

Und noch eine Info:
Durch einen Klick auf die Rote Laterne wird die **Simulation gestoppt**, ein weiterer Klick auf den **Debug-Button** führt zum **Verlassen des Debuggers**. Damit kehren wir wieder zu unserem Ausgangspunkt, also dem Editor, zurück.

4.4. Flashen mit FLIP

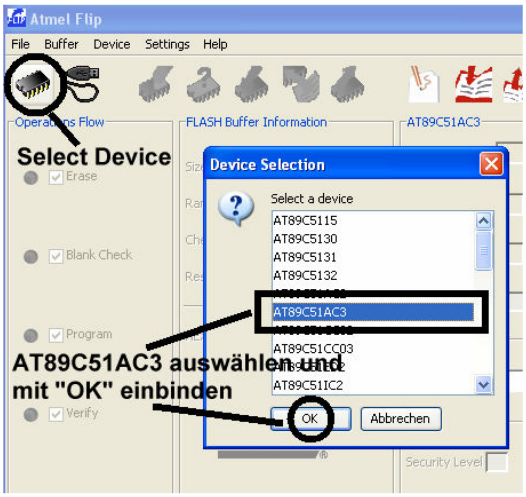
Vorbereitungen:

e) Das Board wird mit einer Gleichspannung von **+8….+10V** versorgt (Bitte nicht wesentlich überschreiten, sonst wird es dem Festspannungsregler auf der Platine zu warm). Wahlweise kann direkt eine Gleichspannung eingespeist **ODER** ein Steckernetzteil (z. B. AC-Adapter) verwendet werden. Passende Stecker sind auf dem Board vorhanden (Siehe Foto des Boards in der Einführung)!

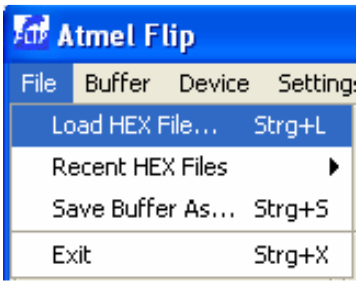
f) Die COM-Schnittstellen des PCs und des Boards werden über ein „**Null-Modem-Kabel**“ (= Crossed Link Cable) verbunden.

g) Der Programmierschalter auf dem Board wird auf „**Flash**“ eingestellt -- die zugehörige rote LED auf dem Board muss leuchten.

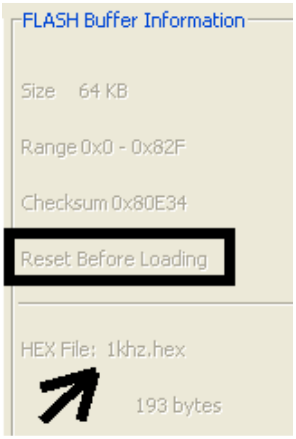
h) Nun wird FLIP gestartet.



e) Der AT89C51AC3 wird zuerst über „**Select Device“** in nebenstehender Reihenfolge eingebunden.



f) Über „**File**“ kommen wir an „**Load HEX File**“ heran, klicken darauf und suchen nach unserem File „**1kHz.hex**“ im zugehörigen **Projektordner „1kHz**“, den wir ja selbst angelegt haben. Das wird geöffnet…



g) …und der Erfolg lässt sich sofort in der Mitte des FLIP-Bildschirms kontrollieren. Allerdings ist das Ganze recht unscheinbar und kaum zu erkennen, da diese Informationen erst später vor dem Flashen dick markiert werden…

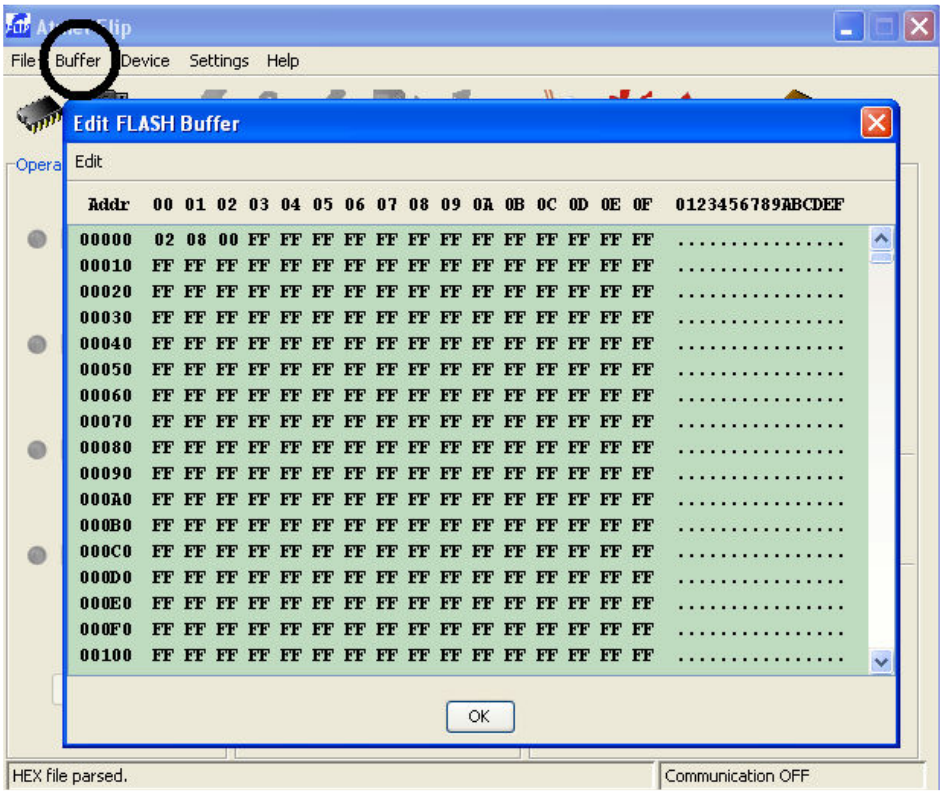
Achtung: genau so unauffällig steht hier eine wichtige Information, ohne die wir anschließend Probleme bei der Flasherei hätten:

Reset before Loading

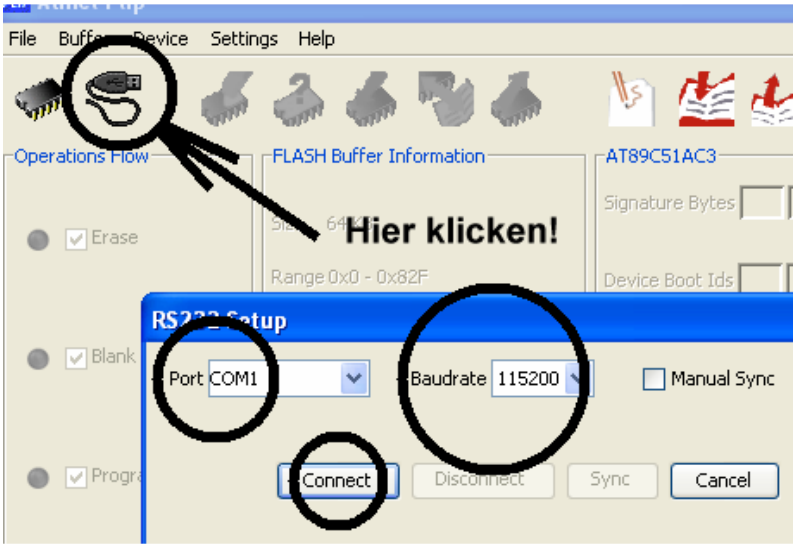
Das müssen wir beim nächsten Schritt unbedingt beherzigen.

Übrigens:

Wer den Knopf „**Buffer**“ und dann „**Edit**“ drückt, bekommt etwas besonders Hübsches zu sehen, nämlich die übertragenen Maschinencodes im Hex-File einschließlich der zugehörigen Speicherplatzadressen. Und zusätzlich werden noch erkannte ASCII-Zeichen ausgegeben…
Bitte mal testen!



So wird das aussehen und damit kann man genau die Speicherbelegung untersuchen!



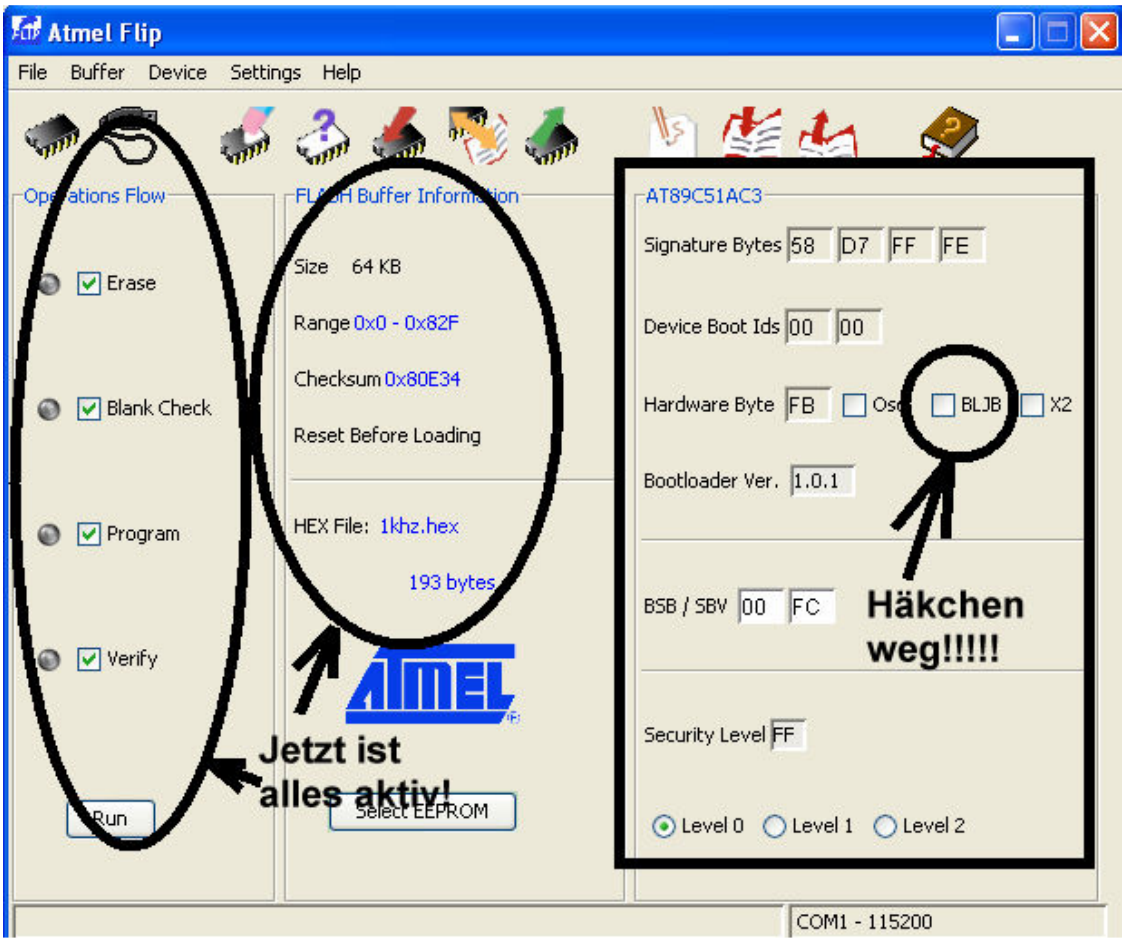
h) Nun klicken wir erst auf den **Kommunikations-Button** („……mit dem Schnittstellenkabel drauf….) und wählen „**RS232**“. Jetzt sind wir im nebenstehenden Setup-Menu, kontrollieren den **COM-Port** und wählen eine **passende Baudrate.**

Bitte daran denken:
Mit einem 12 MHz-Quarz dürfen wir höchstens mit 9600 Baud arbeiten.
Setzt man dagegen den 11,059 MHz-Quarz in die Fassung, dann sind sogar die maximalen 115200 Baud zulässig!

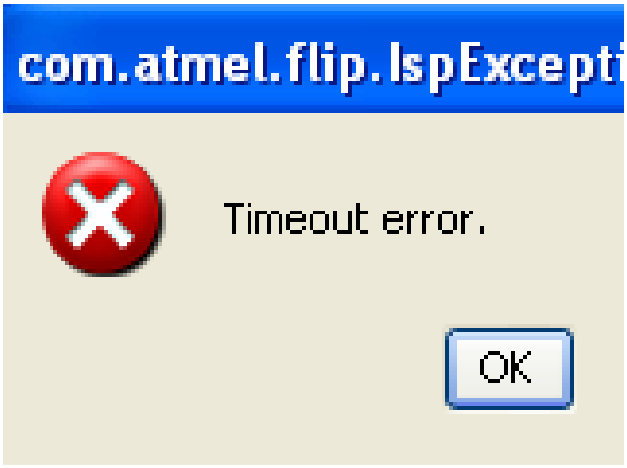
Ist das geklärt, dann drücken wir kurz auf den Reset-Button des Controllerboards und beim einem abschließenden Klick auf „Connect“ sollte -- wenn alles klappt -- die Verbindung zwischen FLIP und dem Board hergestellt sein.

Da gibt es nun natürlich zwei Möglichkeiten:

Wenn die Verbindung steht, sieht der Bildschirm so aus:



Gleich prüfen: Ist wirklich das Häkchen bei BLJB (= Boot Loader Jump Bit) entfernt? Das ist lebenswichtig, denn nur dann startet die in den Controller geflashte Application automatisch beim Reset oder Einschalten…



Wenn es dagegen nicht geklappt hat, erkennen wir das an dieser Meldung.
Da wiederholt man nochmals den Prozess und prüft:

f) Steht der **Board-Schalter** auf „**Flash**“ und leuchtet die rote Programmier-LED?

g) Sind alle **Schnittstellenkabe**l am Board, PC und USB-Converter eingesteckt?

h) Stimmt der **COM-Port (…da war doch was mit dem Converter…..)**?

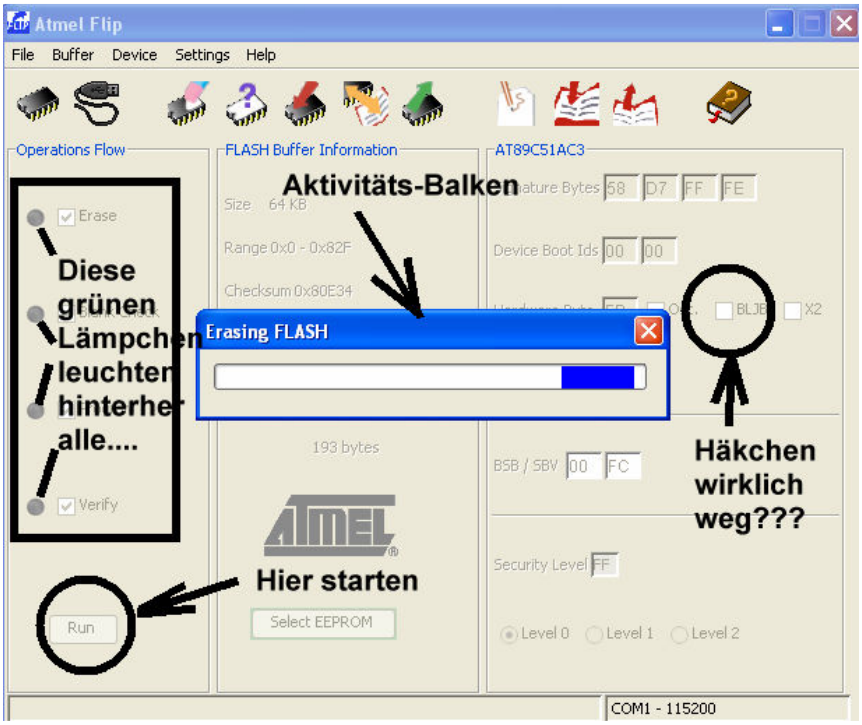
i) Ist die **Baudrate viel zu hoch** eingestellt (z. B. **max. 9600**, weil ein Quarz mit 12MHz statt 11,0592MHz benützt wird)?

j) Hilft eine **Reduktion der Baudrate auf 2400** ?

Und bitte nie vorher den Reset beim Board vergessen!

Bei hartnäckigen Fällen empfiehlt die ONLINE-Hilfe ein andere Reihenfolge der Schritte für Prozedur h) (…offensichtlich gab es da Laptops, die Ärger machten….):
Erst den Kommunikationsbutton drücken, den COM-Port prüfen und die Baudrate auf max. 9600 stellen. Dann das Häkchen bei „**Manual Sync**“ setzen und auf „**Connect**“ klicken. Nun ein- oder zweimal auf die Reset-Taste des Boards drücken und anschließend den Button „**Sync**“ anklicken.

Zum Trost: Irgendwann klappt es immer und dann ist das Ziel wirklich nahe:



Zuerst kontrollieren wir nochmals sehr sorgfältig, ob **bei „BLJB“ das Häkchen entfernt ist**. Es handelt sich hier um das sehr wichtige

„Boot Loader Jump Bit“

und das **muss gelöscht** sein.

Nur wenn das der Fall ist, kann das Controllerboard nach einem Reset (= Betätigen der Reset-Taste ODER nach dem Einschalten der Betriebsspannung) das in das EPROM geflashte Programm abarbeiten!!

Ist das erledigt, dann klicken wir auf „**RUN**“ (links unten im Bild) und sehen dem PC bei der Arbeit zu (..dazu dient der Aktivitätsbalken). Sobald er fertig ist, leuchten alle vier Lämpchen (von „Erase bis „Verify“) grün auf.

Jetzt kann man beim Controllerboard mit dem Schiebeschalter von „Flash Mode“ auf „RUN“ umschalten -- dann geht die rote LED aus. Ein Druck auf die Reset-Taste des Boards (oder ein Aus- und Einschalten der Versorgungsspannung) startet endlich das Controllerprogramm.

Bei Programm-Änderungen oder bei der Fehlersuche geht man einfach zu KEIL µvision3 zurück, korrigiert die Fehler und erzeugt schließlich ein **neues Hex-File**. Das lädt das erneut in der eben besprochenen Weise ins Flash-EPROM und startet nochmals die Anwendung.

Anhang 1: Das INTEL-HEX-File -- ein unbekanntes Wesen

Als Controller-Programmierer sollte man sich ruhig auch mal mit diesem treuen Diener beschäftigen, der für die richtige Programmspeicherung im EPROM sorgt. Sehen wir uns doch dieses erzeugte HEX-File „**1kHz.hex**“ an, wenn das Assemblerfile „1kHz.A51“ eingebunden und dann assembliert + gelinkt wurde:

```
:03000000020800F3
:0A08000075810712080AB29080F912
:05080A007FFADFFE2271
:00000001FF
```

Das sieht zunächst etwas wild aus, ist aber recht einfach zu verstehen. Dahinter steckt folgendes System:

Anzahl der Daten-Bytes	**Code-adresse ab**	**Datentyp**	**Programmcode**	**Prüfbyte**
:03	0000	00	020800 (hier z. B.: springe nach 0x0800)	F3
:0A	0800	00	75810712080AB29080F9	12
:05	080A	00	7FFADFFE22	71
Und diese letzte Zeile markiert schon das Ende:				
:00	0000	01	FF	

==========

Beim ***Datentyp*** gilt: „00“ bedeutet abgelegte Hex-Zahlen (= Daten- oder Code-Bytes),
„01“ dagegen das File-Ende

==========

==========

Das ***Prüfbyte*** jeder Zeile wird auf folgende Weise ermittelt und dieser Weg am Beispiel der ersten Zeile demonstriert:

1. Schritt:
Alle Bytes der Zeile (mit Ausnahme des Prüfbytes) werden zusammenaddiert und das Ergebnis als Hex-Zahl ausgedrückt:

```
03h  +  00h  +  00h  +  00h  +  02h  +  08h  +  00h  =
3    +  0    +  0    +  0    +  2    +  8    +  0    =    13    =  0Dh
```

2. Schritt:
Das Hex-Ergebnis wird als Dualzahl geschrieben und anschließend **komplementiert** (= Nullen in Einsen bzw. Einsen in Nullen umgewandelt):

```
                0Dh   =  0000 1101
Komplement            =  1111 0010
```

3. Schritt:
Durch Addieren einer „1“ erhält man das „**Zweier-Komplement**“ -- und das stellt dann exakt unser Prüfbyte dar!

1111 0010 + 1 = 1111 0011 = **F3h**

Siehe oben....

==========

==========

Das Entschlüsseln der Programmcodes bei einem unbekannten Hex-File funktioniert entweder in „Handarbeit“ mit Hilfe der Befehlsliste des verwendeten Controllers oder mit einem Profiprogramm namens „Disassembler“.

Anhang 2:
Selbstgeschriebener vollständiger Header für den ATMEL-Controller „AT89C51AC3“

```
/*****************************************************************************
* NAME: AT89C51AC3.h   =  Header for ATMEL AT89C51AC3
*----------------------------------------------------------------------------
* PURPOSE: include file for KEIL
*
* Modification of <t89c51ac2.h> by Gunthard Kraus, Elektronikschule
Tettnang
*****************************************************************************/
#ifndef _AT89C51AC3_H_

#define _AT89C51AC3_H_

#define Sfr(x, y)     sfr x = y
#define Sbit(x, y, z)   sbit x = y^z
#define Sfr16(x,y) sfr16 x = y

/*--------------------------------------*/
/* Include file for 8051 SFR Definitions  */
/*--------------------------------------*/

/*  BYTE Register  */
Sfr (P0 , 0x80);
Sfr (P1 , 0x90);

Sbit (P1_7, 0x90, 7);
Sbit (P1_6, 0x90, 6);
Sbit (P1_5, 0x90, 5);
Sbit (P1_4, 0x90, 4);
Sbit (P1_3, 0x90, 3);
Sbit (P1_2, 0x90, 2);
Sbit (P1_1, 0x90, 1);
Sbit (P1_0, 0x90, 0);

Sfr (P2 , 0xA0);

Sbit (P2_7 , 0xA0, 7);
Sbit (P2_6 , 0xA0, 6);
Sbit (P2_5 , 0xA0, 5);
Sbit (P2_4 , 0xA0, 4);
Sbit (P2_3 , 0xA0, 3);
Sbit (P2_2 , 0xA0, 2);
Sbit (P2_1 , 0xA0, 1);
Sbit (P2_0 , 0xA0, 0);


Sfr (P3 , 0xB0);

Sbit (P3_7 , 0xB0, 7);
Sbit (P3_6 , 0xB0, 6);
Sbit (P3_5 , 0xB0, 5);
Sbit (P3_4 , 0xB0, 4);
Sbit (P3_3 , 0xB0, 3);
Sbit (P3_2 , 0xB0, 2);
Sbit (P3_1 , 0xB0, 1);
Sbit (P3_0 , 0xB0, 0);
```

```
Sbit (RD , 0xB0, 7);
Sbit (WR , 0xB0, 6);
Sbit (T1 , 0xB0, 5);
Sbit (T0 , 0xB0, 4);
Sbit (INT1, 0xB0, 3);
Sbit (INT0, 0xB0, 2);
Sbit (TXD , 0xB0, 1);
Sbit (RXD , 0xB0, 0);

Sfr (P4 , 0xC0);

Sfr (PSW , 0xD0);

Sbit (CY  , 0xD0, 7);
Sbit (AC  , 0xD0, 6);
Sbit (F0  , 0xD0, 5);
Sbit (RS1 , 0xD0, 4);
Sbit (RS0 , 0xD0, 3);
Sbit (OV  , 0xD0, 2);
Sbit (UD  , 0xD0, 1);
Sbit (P   , 0xD0, 0);

Sfr (ACC , 0xE0);
Sfr (B , 0xF0);
Sfr (SP , 0x81);
Sfr (DPL , 0x82);
Sfr (DPH , 0x83);

Sfr (PCON , 0x87);
Sfr (CKCON0 , 0x8F);
Sfr (CKCON1 , 0x9F);

/*------------------- TIMER registers ----------------------*/
Sfr (TCON , 0x88);
Sbit (TF1 , 0x88, 7);
Sbit (TR1 , 0x88, 6);
Sbit (TF0 , 0x88, 5);
Sbit (TR0 , 0x88, 4);
Sbit (IE1 , 0x88, 3);
Sbit (IT1 , 0x88, 2);
Sbit (IE0 , 0x88, 1);
Sbit (IT0 , 0x88, 0);

Sfr (TMOD , 0x89);

Sfr  (T2CON , 0xC8);
Sbit (TF2   , 0xC8, 7);
Sbit (EXF2  , 0xC8, 6);
Sbit (RCLK  , 0xC8, 5);
Sbit (TCLK  , 0xC8, 4);
Sbit (EXEN2 , 0xC8, 3);
Sbit (TR2   , 0xC8, 2);
Sbit (C_T2  , 0xC8, 1);
Sbit (CP_RL2, 0xC8, 0);

Sfr (T2MOD , 0xC9);
Sfr (TL0 , 0x8A);
Sfr (TL1 , 0x8B);
Sfr (TL2 , 0xCC);
Sfr (TH0 , 0x8C);
Sfr (TH1 , 0x8D);
Sfr (TH2 , 0xCD);
```

```
Sfr (RCAP2L , 0xCA);
Sfr (RCAP2H , 0xCB);
Sfr (WDTRST , 0xA6);
Sfr (WDTPRG , 0xA7);


/*------------------- UART registers ------------------------*/
Sfr (SCON , 0x98);
Sbit (SM0  , 0x98, 7);
Sbit (FE   , 0x98, 7);
Sbit (SM1  , 0x98, 6);
Sbit (SM2  , 0x98, 5);
Sbit (REN  , 0x98, 4);
Sbit (TB8  , 0x98, 3);
Sbit (RB8  , 0x98, 2);
Sbit (TI   , 0x98, 1);
Sbit (RI   , 0x98, 0);

Sfr (SBUF , 0x99);
Sfr (SADEN , 0xB9);
Sfr (SADDR , 0xA9);


/*---------------------Serial Port Interface SPI------------*/
Sfr (SPCON , 0xD4);
Sfr (SPSCR , 0xD5);
Sfr (SPDAT , 0xD6);


/*-------------------- ADC registers ----------------------*/
Sfr (ADCLK , 0xF2);
Sfr (ADCON , 0xF3);
#define MSK_ADCON_PSIDLE 0x40
#define MSK_ADCON_ADEN   0x20
#define MSK_ADCON_ADEOC  0x10
#define MSK_ADCON_ADSST  0x08
#define MSK_ADCON_SCH    0x07
Sfr (ADDL , 0xF4);
#define MSK_ADDL_UTILS   0x03
Sfr (ADDH , 0xF5);
Sfr (ADCF , 0xF6);

/*-------------------- FLASH EEPROM registers ------------*/
Sfr (FCON  , 0xD1);
#define MSK_FCON_FBUSY 0x01
#define MSK_FCON_FMOD  0x06
#define MSK_FCON_FPS   0x08
#define MSK_FCON_FPL   0xF0
Sfr (EECON  , 0xD2);
#define MSK_EECON_EEBUSY 0x01
#define MSK_EECON_EEE    0x02
#define MSK_EECON_EEPL   0xF0

Sfr (AUXR  , 0x8E);
Sfr (AUXR1 , 0xA2);
#define MSK_AUXR_M0      0x20

#define MSK_AUXR1_ENBOOT 0x20

Sfr (FSTA , 0xD3);
```

```
/*--------------------- Interrupt registers -----------------------*/
Sfr (IPL1 , 0xF8);
Sfr (IPH1 , 0xF7);
Sfr (IEN0 , 0xA8);
Sfr (IPL0 , 0xB8);
Sfr (IPH0 , 0xB7);
Sfr (IEN1 , 0xE8);


/*  IEN0  */
Sbit (EA   , 0xA8, 7);
Sbit (EC   , 0xA8, 6);
Sbit (ET2  , 0xA8, 5);
Sbit (ES   , 0xA8, 4);
Sbit (ET1  , 0xA8, 3);
Sbit (EX1  , 0xA8, 2);
Sbit (ET0  , 0xA8, 1);
Sbit (EX0  , 0xA8, 0);


/*  IEN1  */
Sbit (ESPI , 0xE8, 3);
Sbit (EADC , 0xE8, 1);


/*  IPL0  */
Sbit (PPC  , 0xB8, 6);
Sbit (PT2  , 0xB8, 5);
Sbit (PS   , 0xB8, 4);
Sbit (PT1  , 0xB8, 3);
Sbit (PX1  , 0xB8, 2);
Sbit (PT0  , 0xB8, 1);
Sbit (PX0  , 0xB8, 0);


/*  IPL1  */
Sbit (SPIL   , 0xF8, 3);
Sbit (PADCL  , 0xF8, 1);



/*---------------------- PCA registers --------------------*/
Sfr (CCON , 0xD8);
Sbit(CF  , 0xD8, 7);
Sbit(CR  , 0xD8, 6);
Sbit(CCF4, 0xD8, 4);
Sbit(CCF3, 0xD8, 3);
Sbit(CCF2, 0xD8, 2);
Sbit(CCF1, 0xD8, 1);
Sbit(CCF0, 0xD8, 0);

Sfr (CMOD , 0xD9);
Sfr (CH , 0xF9);
Sfr (CL , 0xE9);
Sfr (CCAP0H  , 0xFA);
Sfr (CCAP0L  , 0xEA);
Sfr (CCAPM0  , 0xDA);
Sfr (CCAP1H  , 0xFB);
Sfr (CCAP1L  , 0xEB);
Sfr (CCAPM1  , 0xDB);
Sfr (CCAP2H  , 0xFC);
Sfr (CCAP2L  , 0xEC);
Sfr (CCAPM2  , 0xDC);
```

```
Sfr (CCAP3H  , 0xFD);
Sfr (CCAP3L  , 0xED);
Sfr (CCAPM3  , 0xDD);
Sfr (CCAP4H  , 0xFE);
Sfr (CCAP4L  , 0xEE);
Sfr (CCAPM4  , 0xDE);

#endif
```