

The ATTINY-Projekt

Why FORTH?

Designed and written by

Georg Heinrichs

*Translated into English, edited and published
by Juergen Pintaske, ExMark*

Inhalt/Contents

<http://www.g-heinrichs.de/wordpress/index.php/attiny/>

- 1 [Wie FORTH entstand](#)
- 2 [Einstieg in MikroForth](#)
- 3 [Definieren von Wörtern](#)
- 4 [Arbeiten mit dem Stack](#)
- 5 [Portbefehle bei MikroForth](#)
- 6 [Schleifen und Verzweigungen](#)
- 7 [COM, I2C und EEPROM bei MikroForth](#)
- 8 [MikroForth-Variablen](#)
- 9 [Der Compiler von MikroForth](#)
- 10 [Funktionsweise der do-loop-Schleife](#)
- 11 [DO-IT-YOURSELF: A-Wörter selbst herstellen](#)
- 12 [Rekursion mit MikroForth](#)
- 13 [MikroForth-Interrupts](#)
- 14 [MikroForth einstellen](#)

Software:

[MikroForth-Compiler](#) [MikroForth-Vokabular](#)

+++++

- 1 How FORTH started
- 2 Getting started with MikroForth
- 3 Defining words
- 4 Working with the stack
- 5 Port commands for MikroForth
- 6 Loops and Branches
- 7 COM, I2C and EEPROM in MikroForth
- 8 MikroForth Variables
- 9 The MikroForth Compiler
- 10 How the Do-Loop works
- 11 DO-IT-YOURSELF: Create A-Words yourself
- 12 Recursion in MikroForth
- 13 MikroForth interrupts
- 14 Setting the MikroForth

Software:

MikroForth Compiler Mikro Forth Vocabulary

0_Collected Documents MikroForth_v13_A5 English German

0 Wie FORTH entstand

Forth wurde von Charles H. Moore 1969 entwickelt. FORTH weist eine Reihe von Eigentümlichkeiten auf, die es stark von herkömmlichen Programmiersprachen unterscheidet. FORTH stellt i. A. nicht nur eine Entwicklungsumgebung, sondern auch ein Betriebssystem dar.

Diese Eigentümlichkeiten lassen sich gut aus der Entstehungsgeschichte erklären. Moore hatte zur Steuerung des Teleskops einer Sternwarte einen Rechner ohne Software gekauft. Er hatte sich vorgenommen, alle Komponenten selbst zu programmieren, die für eine komfortable Programmierung und den Betrieb des Rechners notwendig sind. Hierzu gehören ein Betriebssystem, eine Hochsprache und eine Entwicklungsumgebung. All diese Komponenten wurden innerhalb eines einzelnen Programms verwirklicht – dem Forth-System.

Moore erzählte später selbst:



Ich entwickelte FORTH im Laufe mehrerer Jahre als eine Schnittstelle zwischen mir und den Computern, die ich programmierte. Die traditionellen Sprachen lieferten nicht die Leistungsfähigkeit, Einfachheit oder Flexibilität, die ich wollte. Ich missachtete viele geltende Weisheiten, um exakt die Fähigkeiten einzubauen, die ein produktiver Programmierer benötigt.

*Die allerwichtigste davon ist die Möglichkeit, neue Eigenschaften hinzuzufügen, die später einmal notwendig werden. Als ich zum ersten Mal die Ideen, die ich entwickelt hatte, zu einer Einheit zusammenfasste, arbeitete ich auf **Abbildung 1** einer IBM 1130, einem Computer der "dritten Generation". Das Ergebnis schien mir so leistungsfähig, dass ich es für eine "Sprache der vierten Computergeneration" hielt. Ich würde sie FOURTH genannt haben, aber die 1130 erlaubte nur eine Kennung mit 5 Buchstaben. So wurde aus FOURTH FORTH, immerhin ein nettes Wortspiel.*

(forth = vorwärts)



Beispiel Quelle:

<http://computermuseum.informatik.uni-stuttgart.de/dev/ibm1130/ibm1130.html>

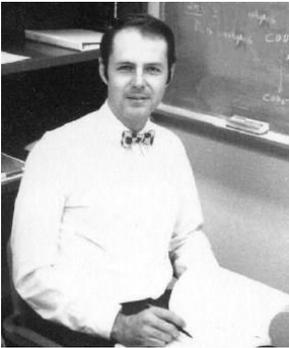
(Zitiert nach L. Brodie: FORTH)

0 How FORTH started

Forth was developed by Charles H. Moore 1969. FORTH has a number of peculiarities, that make it very different from conventional programming languages. FORTH represents i.e. not only a development environment, but also an operating system.

These peculiarities can be easily explained from the history of the origin. Moore had bought a computer without a software to control the telescope of an observatory.

He had planned to program all the components himself, which are necessary for a comfortable programming and operation of the computer. These include an operating system, a language and a development environment. All these components were realized within a single program - the Forth system.



Moore later told himself: I developed FORTH over the course of several years as an interface between me and the computers that I programmed. The traditional languages did not provide the performance, simplicity or flexibility that I wanted. I hated many valid wisdoms to incorporate exactly the skills a productive programmer needs. The most important of these is the possibility to add new properties, which are necessary later.

When I first combined the ideas that I had developed into one unit, I worked on an IBM 1130 see **Picture 1**, a computer of the "third generation". The result seemed to me so powerful that I thought of it as a "language of the fourth generation of computers". I would have called it FOURTH, but the 1130 allowed only an identifier with 5 letters. So, from FOURTH to FORTH, a nice play on words.

(Forth = forward)



Picture 1 (1) : IBM 1130

(There is a number and one in brackets:
Original file related and in one sequence here)

Example source: <http://computermuseum.informatik.uni-stuttgart.de/dev/ibm1130/ibm1130.html>

(Cited from L. Brodie: Starting FORTH)

1 Wie MikroForth entstand

Eines Tages kam mein Sohn zu mir - er war gerade 14 Jahre alt - und fragte mich, wie man einen Compiler herstelle. Er würde gerne einen selbst programmieren. Nun hatte ich einmal gelesen, wie man Compiler baut; aber das war schon viele Jahre her und das meiste hatte ich wieder vergessen. Allerdings hatte ich sehr wohl noch in Erinnerung, dass der Compilerbau schon etwas komplexer ist und kaum etwas für einen 14-Jährigen. Und das sagte ich ihm dann auch.

Aber er ließ nicht locker. Einige Monate später - ich arbeitete gerade an einem Konzept für eine Mikrocontroller-Fortbildung - kamen wir auf die Idee, einen Compiler für den Mikrocontroller Attiny 2313 zu programmieren. Als Sprache wählten wir FORTH, nicht zuletzt wegen der einfachen Grundstruktur.

Unser FORTH-Compiler sollte allerdings nicht auf dem Mikrocontroller selbst laufen, sondern auf einem PC: Dieser sollte den FORTH-Code in Maschinencode umsetzen, welcher dann auf dem Attiny hochgeladen werden sollte. Der Compiler selbst ist relativ einfach, er greift auf eine Datenbank zurück, in der sich eine Sammlung von A-Wörtern (Befehlsfolgen in Maschinencode) und F-Wörtern (Befehlsfolgen in FORTH-Code) befindet. Da es uns hauptsächlich um das Prinzip ging, haben wir auch nicht alle gängigen FORTH-Wörter (Befehle) implementiert, sondern nur einen kleinen Bruchteil.

Es war äußerst lehrreich, derartige Programmschnipsel zu schreiben, und ich kann nur jedem raten, dies einmal selbst zu versuchen. MikroFORTH macht dies möglich; denn es ist ein offenes System: Die Datenbank kann nach eigenen Vorstellungen beliebig verändert und erweitert werden. Dies entspricht ja gerade auch der Moore'schen Leitvorstellung.

Faszinierend für uns war insbesondere folgender Umstand: Als wir mit dem Projekt begannen, kannten wir nur einige wesentliche Eigenschaften von FORTH. Später - als das Projekt fast fertig war - haben wir einmal recherchiert, wie FORTH im Original beim

Compilieren vorgeht. Und siehe da - wir fanden einige unserer Ideen wieder.

Was die Betrachtung von FORTH leistet

- Verständnis für einfachen Compiler
- Verständnis für Assembler und Maschinencode
- Verständnis und Übung im Umgang mit Stapeln
- Verständnis mit der Übergabe von Parametern
- Last - but not least: effizienten Maschinencode

1 How MicroForth started

One day, my son came to me - he was just 14 years old - and asked me how to make a compiler. He wanted to program one himself. Now, I had once read about how to build compilers; But that had been many years ago and I had forgotten most of it.

However, I could still very well still remember, that compiler construction is somewhat more complex and hardly anything for a 14-year-old. And I told him that.

But he did not let go and kept asking again. A few months later - I was working on the concept of a Microcontroller Training - we came up with the idea of programming a compiler for the Atmel/Microchip ATtiny 2313 microcontroller. As a language to write this compiler for, we chose FORTH, not least because of the simple structure of this language.

Our FORTH compiler should not run on the microcontroller itself, but on a PC: This compiler should convert the FORTH code into machine code, which should then be uploaded to the Attiny. The compiler itself is rather straightforward: it accesses a database which contains a collection of A-words (command sequences in machine code – and the A stands for Assembler) and F-words (command sequences in FORTH code).

Since we were mainly concerned about the principle, we did not implemented all common FORTH words (commands), but only a small subset.

It was extremely instructive to write such a compiler program, and I can only advise anyone to try it yourself. Micro-FORTH makes this possible; As it is an open system: the database can be changed and expanded as desired. This is exactly the same as Moore's leading concept wanted it.

We were particularly fascinated by the following: when we started the project, we only knew a few of FORTH's essential characteristics.

Later, when the project was almost finished, we once researched how a FORTH compilation works in other implementations. And to our surprise - we found some of our ideas in there to be exactly ythe same.

What can you get out of looking at FORTH:

- Understanding simple Compilers
- Understanding of Assembler and Machine Code
- Understanding and practice of handling Stacks
- Understanding how to pass Parameters
- Last - but not least: how to write efficient Machine Code

2 Einstieg in MikroForth

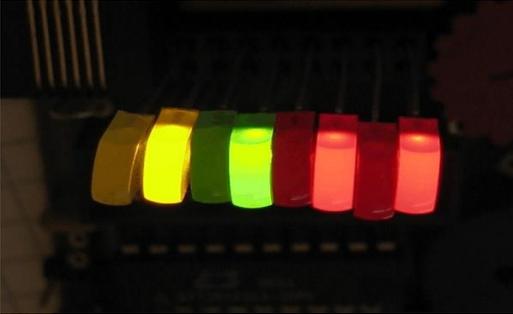


Abbildung 1 (2) 8 LEDs

MikroForth ist ein FORTH-Compiler für den Attiny 2313. Damit ist gemeint: Das Programm wird auf einem PC eingegeben und kompiliert. Der dabei erzeugte HEX-Code wird anschließend mit einem Brennprogramm auf den Attiny übertragen.

Im Stile einer Schritt-für-Schritt-Anweisung wollen wir in diesem Kapitel zeigen, wie man mit MikroForth umgeht. Dazu werden wir mit MikroForth den Attiny so programmieren, dass er die Leuchtdioden an Port B in einem solchen Muster wie in Abb. 1 aufleuchten lässt. Die wenigen benötigten FORTH-Sprachelemente werden wir einfach angeben; wie sie funktionieren - und vor allem: wie man damit eigenständig Programme entwickelt, das werden wir dann in den folgenden Kapiteln darlegen.

Beginnen wir ganz vorne: mit der Installation von MikroForth. Kopieren Sie dazu einfach das Forth-Verzeichnis in das Programme-Verzeichnis (oder auch ein anderes Verzeichnis Ihrer Wahl). Öffnen Sie nun dieses Verzeichnis und starten Sie das Programm Forth2.

Während das Programm startet, lädt es die Datei “forthvoc.vok” mit dem Vorrat an Forth-Befehlen.

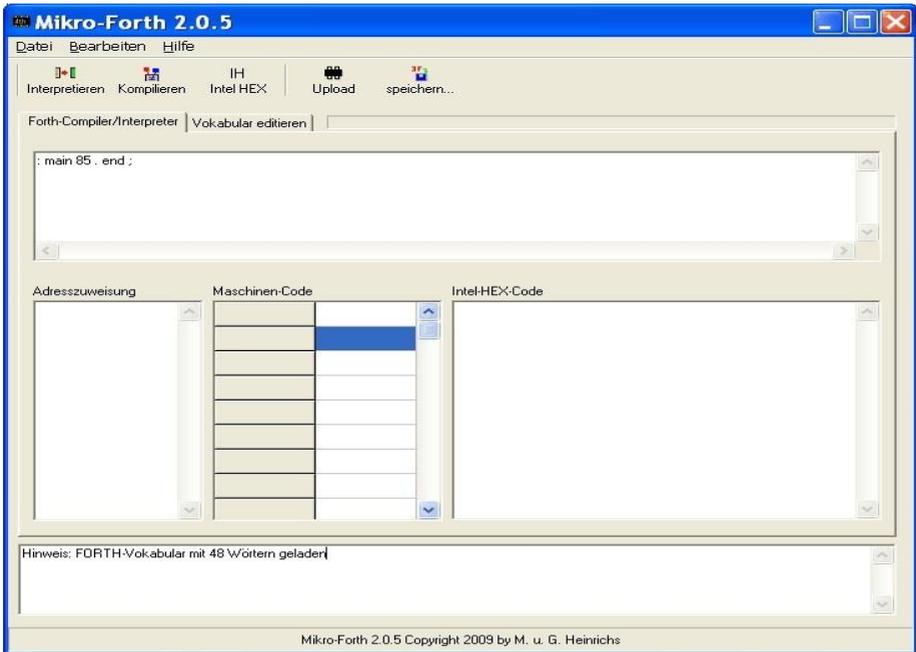


Abbildung 2 (3)

Unser erstes Programm besteht nur aus einer einzigen kurzen Zeile:

```
: main 85 . end ;
```

Geben Sie diesen Forth-Quelltext ganz oben im Formular ein (Abb. 2). Achten Sie auf die Leerzeichen zwischen den einzelnen Befehlen, insbesondere hinter dem Doppelpunkt und vor dem Semikolon; die Eingabe braucht man nicht mit der RETURN-Taste  abschließen.

Unser Programm gibt zunächst die Zahl 85 am Port B aus; da die Zahl 85 im Zweiersystem als 01010101 geschrieben wird, sollte dies das gewünschte Muster bei den LEDs erzeugen (vgl. Abb. 1). Anschließend führt das Programm eine Endlosschleife aus.

Als nächstes betätigen wir die “Interpretieren”-Schaltfläche. Wir erhalten die folgende Warnung:

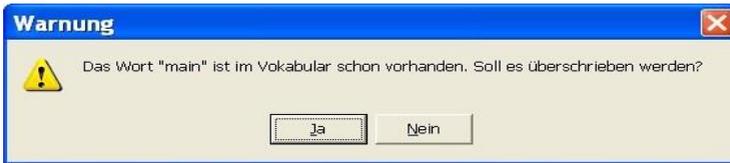


Abbildung 3 (4) Warnung bezüglich “main”

Was bedeutet das?

“.” oder “end” sind sogenannte Wörter; diese stellen Befehle oder Befehlsfolgen dar, die der Mikrocontroller ausführen soll. Die Gesamtheit aller Wörter bezeichnen wir als Forth-Vokabular. Beim Interpretieren der eingegebenen Zeile wird hier dem Vokabular ein neues Wort “main” hinzugefügt, welches die Zahl 85 und die Wörter “.” und “end” zusammenfasst. Offensichtlich existierte schon ein Wort “main” im Vokabular. Wie wir noch sehen werden, ist dieses Erstellen neuer Befehle ein wesentliches Konzept der Programmiersprache FORTH.

Da das alte “main”-Wort durch unser neues ersetzt werden soll, klicken wir bei dem Warnhinweis in Abb. 3 auf “Ja”.

Aus diesem neuen Wort “main” muss nun Maschinencode für den Mikrocontroller erzeugt werden. Dazu betätigen wir die “Kompilieren”-Schaltfläche. Unser FORTH-Compiler besorgt sich aus der Datei “forthvoc.vok” die Programmschnipsel für die einzelnen Bestandteile von “main”, also für die Wörter “.” und “end” und fügt sie zu einem Gesamtprogramm zusammen.

Im Adresszuweisungsbereich (Abb. 4) erkennt man die Zuweisung dieser Unterprogramme zu bestimmten Programmspeicheradressen; die Zeile “. □ \$001A “ bedeutet z.B.: Das Unterprogramm für “.”, welches für die Ausgabe auf Port B verantwortlich ist, beginnt bei der Adresse 26 = \$1A. Das gesamte Programm wird dann im Maschinencodebereich angezeigt. Unter jeder Adresse finden wir ein Maschinencode-Wort, bestehend aus 2 Byte.

Im Logbuch entdecken wir weitere Einträge; auf deren Bedeutung werden wir zu einem späteren Zeitpunkt eingehen.

So wie es im Maschinencodebereich angezeigt wird, so wird das Programm auch im Mikrocontroller abgelegt (wenn man davon absieht, dass im Speicher des Mikrocontrollers das höherwertige Byte eines Maschinencode-Wortes nicht wie hier vor, sondern hinter dem niederwertigen steht). Die meisten Brennprogramme benutzen allerdings ein anderes Format, welches noch zusätzliche Kontrollbytes besitzt: das Intel-HEX-Format. Mit der Schaltfläche “Intel-HEX” wandeln wir deswegen nun unseren Maschinencode in dieses Format um. Der Intel-HEX-Code erscheint sogleich im Intel-HEX-Bereich (Abb. 4).

Adresszuweisung	Maschinen-Code	Intel-HEX-Code
stackinit> \$0014	\$0000 \$23C0	:1000000023C01895189518951895189518951895189552
85> \$0017	\$0001 \$1895	:1000100018951895189518951895189518951895189578
.> \$001A	\$0002 \$1895	:1000200018951895189518951895A0E6B0E0089505E57F
end> \$001F	\$0003 \$1895	:100030000D9308951FEF0E9117BB08BB0895FFC6D6
main> \$0020	\$0004 \$1895	:10004000F6DFF8DFFC0F0895EFDFFADFF8DF089571
init> \$0024	\$0005 \$1895	:00000001FF
	\$0006 \$1895	
	\$0007 \$1895	
	\$0008 \$1895	

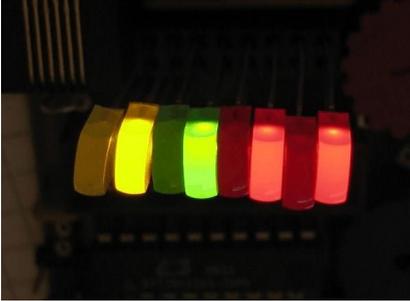
Abbildung 4 (5)

Zu guter letzt müssen wir den Code noch in den Attiny laden. Bei unserer Attiny-Platine kommt wieder das Uploader-Programm zum Einsatz. Wir rufen es auf, indem wir die “Upload”-Schaltfläche betätigen; der Intel-HEX-Code wird dann automatisch vom Uploader-Programm übernommen und kann auf dem üblichen Weg in den Attiny übertragen werden.

Und wenn Sie keinen Eingabefehler gemacht haben, die LEDs korrekt in die Buchsen gesteckt wurden, die Batterie noch voll ist und die Übertragung reibungslos funktioniert hat, ja - dann sollte auch das Bitmuster aus Abb. 1 tatsächlich angezeigt werden und wir gratulieren Ihnen zur erfolgreichen Implementation Ihres ersten FORTH-Programms!



2 Getting into MikroForth



Picture 1 (6) 8 LED Display

MicroForth is a FORTH compiler for the ATtiny 2313. This means: The program is entered on a PC and then compiled. The generated HEX code is then transferred into the Attiny with a Flash Programming utility.

In a step-by-step procedure, we will show you how to deal with MikroForth in this chapter. For this purpose, we will first program the ATtiny with MikroForth program to generate an LED pattern on Port B as shown in Picture 1.

We will simply specify the few FORTH language elements required first; how they work, - and above all - how to develop own programs, we will explain in the following chapters.

Let's start with the installation of MikroForth:

Just copy the Forth directory into the program directory (or another directory of your choice).

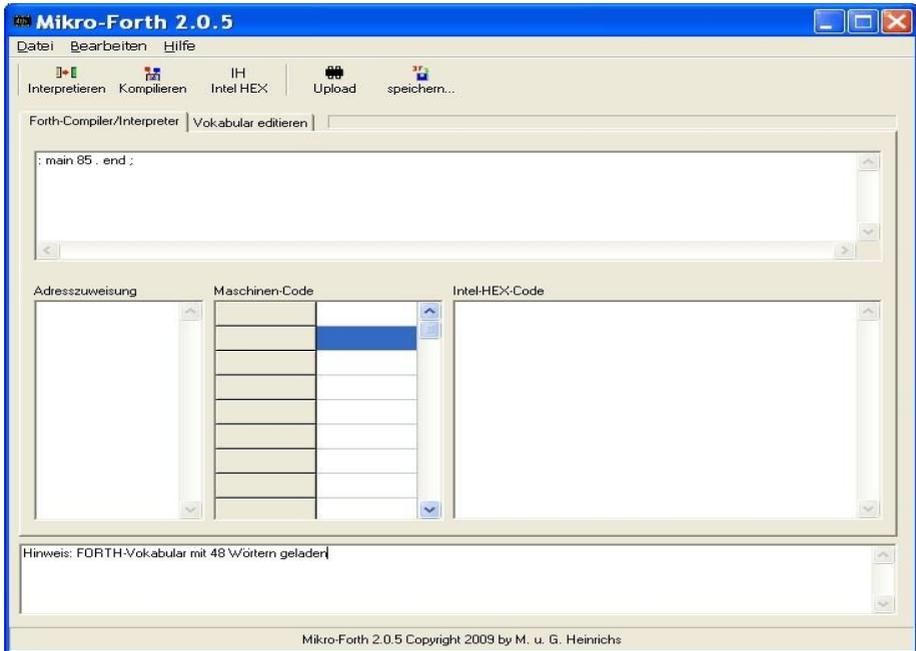
Or let us assume you start a new Folder c:/microforth. Then

Download the software from <http://www.g-heinrichs.de/attiny/forth/forth2.zip>

and unzip it. There is now a folder Forth2 including a few files.

Open this directory and start the **Forth2.exe** program.

As this program starts, it loads the file "forthvoc.vok" with the already defined vocabulary of Forth commands.



File Work on Help
 Interpret Compile Intel HEX Upload Save
 Forth Compiler/Interpreter Edit Vocabulary
 Address Allocation Machine Code Intel HEX Code

Figure 2 (3) The main MikroForth Window

Our first program consists of just one short line of code:

```
: main 85 . end ;
```

Enter this Forth source at the top of the form (Figure 2) under Forth Compiler/Interpreter. Do not forget the separating spaces between each command, especially after the colon and before the semicolon.

This input does not need to be completed with the RETURN key.

Our program first displays the number 85 at port B; Since the number 85 is written as 01010101 in the binary system, this should produce the desired pattern for the LEDs (see Fig. 1).

The program then gets into an endless loop.

Next, we press the "Interpret" button.

We receive the following warning:



The word “main” already exists in the vocabulary. Should it be overwritten?

Yes No

Figure 3

What does that mean?

". ." or "**end**" are so-called words; They represent commands or command sequences to be executed by the microcontroller. The totality of all available words is called the Forth vocabulary.

When interpreting the line typed in, a new word "main" is added to the vocabulary, which contains the number **85** and the words ". ." and "**end**". Obviously, there was already a word "main" in the vocabulary.

As we shall see, creating new commands is an essential concept of the FORTH programming language.

Since the old word "main" – whatever its meaning was - is to be replaced with our new one, we click on "Yes" in the warning in Picture 3.

This new word "**main**", now needs the correct machine code for this microcontroller. To do this, we press the "Compile" button.

Our FORTH compiler obtains the program sections for the individual components of "main" from the file "forthvoc.vok" which are here the words "." and "**end**", and combines them into a new program.

In the address allocation area (Fig. 4), see further down, the assignment of these subroutines to the specific program memory addresses can be seen; the line ". o \$ 001A "means, for example:

The subroutine for ".", which is responsible for the output on port B, and it starts at address 26 = \$ 1A. The entire program is then displayed in the machine code area. Under each address, we find a machine code word consisting of 2 bytes.

In this logbook area we find further entries; the meaning of those will be discussed later on.

As shown in the machine code area, the program is stored in the microcontroller (if one does not take into account that the higher-order byte of a machine code word is not stored before in the memory of the microcontroller, but rather behind in the low-order one).

However, most programmers use a different format, which includes additional control bytes: the Intel HEX format. With the "Intel-HEX" button, we are now converting our machine code into this format. The Intel HEX code appears immediately in the Intel HEX area (Picture 4).

Adresszuweisung	Maschinen-Code	Intel-HEX-Code
stackinit> \$0014	\$0000	:1000000023C01895189518951895189518951895189552
85> \$0017		:1000100018951895189518951895189518951895189578
.)> \$001A	\$0001	:1000200018951895189518951895A0E6B0E0089505E57F
end> \$001F	\$0002	:100030000D9308951FEF0E9117BB08BB0895FFCFD6
main> \$0020	\$0003	:10004000F6DF8DFCFDF0895EFDFFADFF8DF089571
init> \$0024	\$0004	:00000001FF
	\$0005	
	\$0006	
	\$0007	
	\$0008	

Address Allocation Machine Code Intel Hex Code

Picture 4: Address allocation the machine code and the resulting Intel HEX code for programming

Finally, we need to load this code into the Attiny. The Uploader program is used again with our ATtiny board. We call it by pressing the "Upload" button; The Intel HEX code is then automatically handed over to the Uploader program and can be transferred to the ATtiny in the usual way.

If you have not made any input error when typing, the LEDs have been inserted correctly into the sockets, the battery is still full and the transmission has worked smoothly, yes - then the bit pattern from Picture 1 should be displayed and we congratulate you on the successful implementation.

Your first FORTH program sucessfully finished and running!



<http://www.g-heinrichs.de/wordpress/index.php/attiny/>

Exeter 2017_08_07 v13_A5

1	A	B	C	D	E		F	G	H	I	K	1
2	○	○	○	○	○		○	○	○	○	○	2
3	○	○	○	○	■	RS_01 20_Vc	■	○	○	○	○	3
4	○	○	○	○	■	RX_02 19_B7	■	○	○	○	○	4
5	○	○	○	○	■	TX_03 18_B6	■	○	○	○	○	5
6	○	○	○	○	■	A1_04 17_B5	■	○	○	○	○	6
7	○	○	○	○	■	A0_05 16_B4	■	○	○	○	○	7
8	○	○	○	○	■	D2_06 15_B3	■	○	○	○	○	8
9	○	○	○	○	■	D3_07 14_B2	■	○	○	○	○	9
10	○	○	○	○	■	D4_08 13_B1	■	○	○	○	○	10
11	○	○	○	○	■	D5_09 12_B0	■	○	○	○	○	11
12	○	○	○	○	■	G0_10 11_D6	■	○	○	○	○	12
13	○	○	○	○	○		○	○	○	○	○	13
14	○	○	○	○	○		○	○	○	○	○	14
15	○	○	○	○	○		○	○	○	○	○	15
16	○	○	○	○	○		○	○	○	○	○	16
17	○	○	○	○	○		○	○	○	○	○	17
	A	B	C	D	E		F	G	H	I	K	

Hallo Jürgen,

vielen Dank für Deine Email. Den Bootloader und Informationen dazu findest Du unter

<http://www.g-heinrichs.de/wordpress/index.php/attiny/downloads/>

In den Infos wird das Protokoll beschrieben; daraus müsste sich leicht ein einfacher Programmierer entwickeln lassen.

Wegen der Platine werde ich heute mit Herrn Eube sprechen. Auf

<http://www.forum.g-heinrichs.de/viewtopic.php?f=12&t=104> findet

man allerdings auch eine Anleitung für einen Breadboard-Ersatz unserer Attiny-Platine. Vielleicht reicht das ja schon?

Liebe Grüße

Georg Heinrichs

From: Juergen Pintaske

Sent: Friday, July 28, 2017 1:29 PM

To: 'G. Heinrichs'

Subject: <https://wiki.forth-ev.de/doku.php/attiny>

<https://wiki.forth-ev.de/doku.php/attiny>

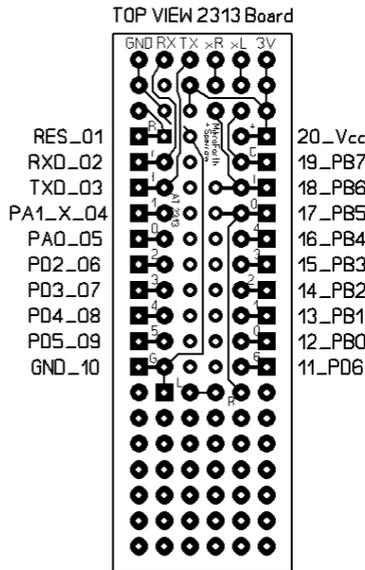
Hallo Georg,

Ich bin nicht sicher, dass ich dieses Link schon geschickt habe.

Ein Ehrenplatz fuer MikroForth und Leiterplatte, wie Du sehen kannst.

Und ich werde bald ein Board mit 2313 aufbauen

Programmieren muesste ich dann wohl ueber Sparrow System, denn der Bootloader fehlt mir ja.



Noch ein schoenes Wochenende.

Best regards

Juergen Pintaske

ExMark

L: 00 44 1392 422 480

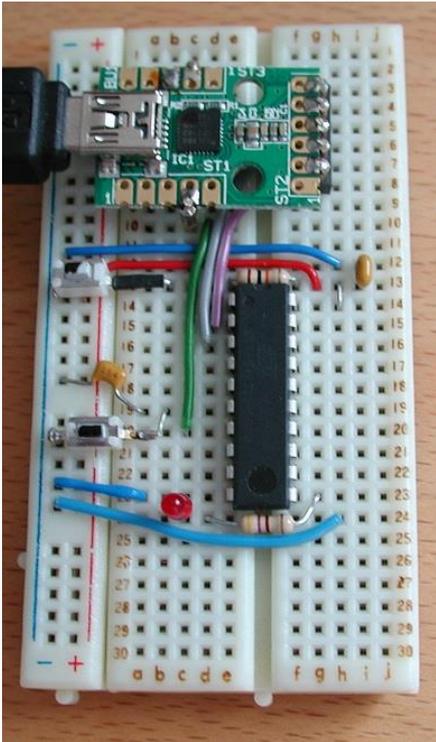
I: www.exemark.com



[Attiny-Platine ohne Platine?!](#)

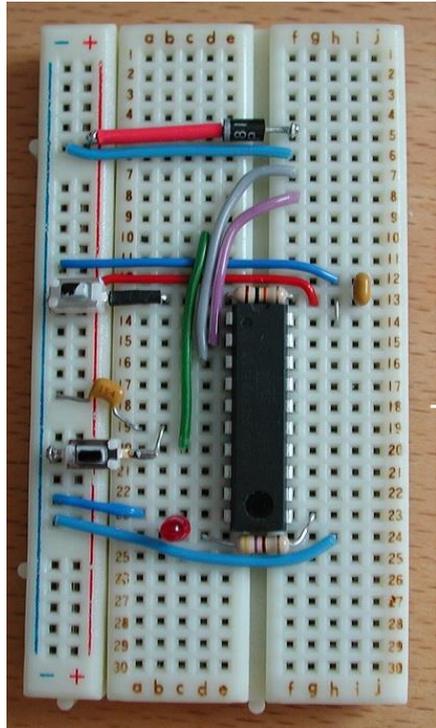
[Beitrag](#) von [Heinrichs](#) » Mi 7. Jun 2017, 09:08

Eine Attiny-Platine ohne Platine? Wie soll das gehen? Ganz einfach: Dank der für wenige Euro erhältlichen USB-UART-Wandler sind nur wenige Bauteile erforderlich, um mit dem Attiny 2313 zu experimentieren; und die kann man rasch auf ein Breadboard stecken. Wie man hierzu vorgehen kann, ist in den Abb. 1 und 2 zu sehen; dabei wurde in Abb. 2 der USB-UART-Wandler entfernt, um dessen Verdrahtung besser sichtbar zu machen.



*Attiny-Breadboard
UART-Wandler*

Attiny_BB_1.jpg (112.5 KiB)
337 mal betrachtet



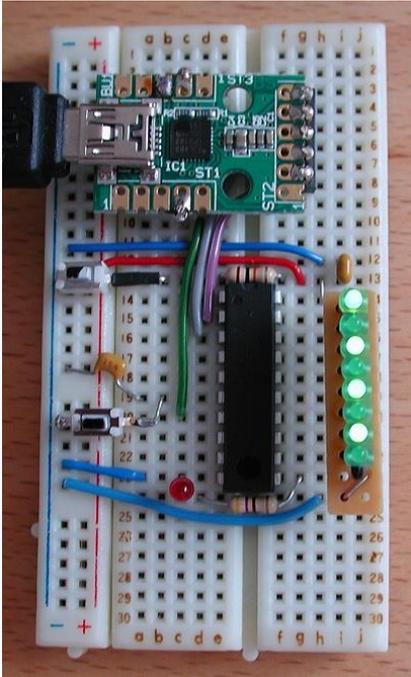
*Attiny-Breadboard ohne USB-
UART-Wandler*

Attiny_BB_2.jpg (123.41 KiB)
337 mal betrachtet

Ein paar Erläuterungen zum Aufbau: Im Norden sieht man den USB-UART-Wandler von ELV (UM 2102, nicht UM 2102N); dieser kann besonders leicht mit Steckerstiften versehen auf das Breadboard gesteckt werden. Südwestlich davon sind der Resettaster und der Taster Ta0 (letzterer mittels Kondensator entprellt) zu sehen; Ta0 ist mit dem Eingang D.2 des Mikrocontrollers versehen und kann dort den Interrupt INTO auslösen. Die rote LED südöstlich von dem Taster Ta0 leuchtet auf, wenn der Mikrocontroller in den Upload-Modus versetzt worden ist.

Der Mikrocontroller befindet sich im Südosten des Breadboards; damit er korrekt mit unserem Uploader-Programm arbeiten kann, muss er

entsprechend eingerichtet sein (vgl. <http://www.g-heinrichs.de/attiny/Erstinstallation.zip>). Die Anschlüsse 22.g-j, 21.g-j, ..., 15.g-j führen zu den Ports B.0 bis B.7. Hier können z. B. LEDs oder auch unser LCD-Modul angeschlossen werden (s. Abb. 3 und 4).



*Attiny-Breadboard mit LED-Array
Modul*

Attiny_BB_3.jpg (119.04 KiB)
336 mal betrachtet



*Attiny-Breadboard mit LCD-
Modul*

Attiny_BB_4.jpg (115.92 KiB)
336 mal betrachtet

In der Abb. 5 sind die Schaltskizzen für die Abb. 1 sowie für das LCD (Pollin-LCD) und das LED-Array (vgl. Abb. 3) angegeben.

<https://www.pollin.de/bauelemente-bauteile/aktive-bauelemente/displays/>

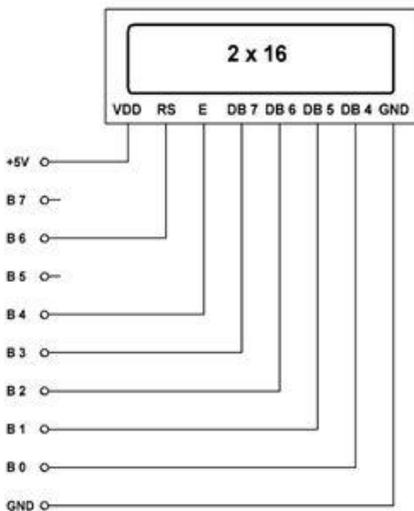
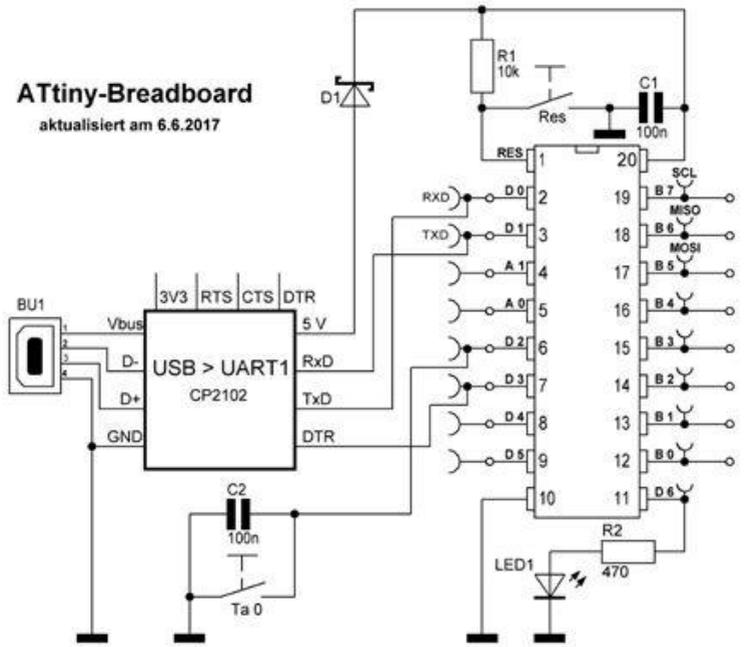
Sämtliche Fotos und Zeichnungen dieses Beitrags stammen von Ulf Ihlefeldt. Vielen Dank dafür!

[Nach oben](#)

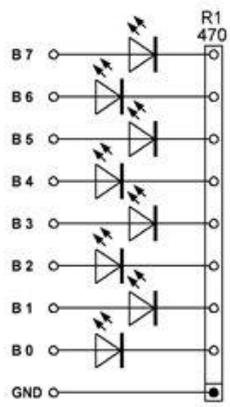
1 Beitrag • Seite 1 von 1

[Zurück zu „Projekte“](#)

ATtiny-Breadboard
aktualisiert am 6.6.2017



|



Schaltskizzen Attiny_BB_Schaltskizzen.jpg (44.31 KiB)

####