

Das grosse Buch für ESP32forth

version 1.6 - 3. Nov. 2023



Autor

- Marc PETREMANN petremann@arduino-forth.com

Mitarbeiter

- Vaclav POSSELT
- Thomas SCHREIN

Inhalt

Autor.....	1
Mitarbeiter.....	1
Einführung.....	7
Übersetzungshilfe.....	7
Entdeckung der ESP32-Karte.....	8
Präsentation.....	8
Die Stärken Punkten.....	8
GPIO-Ein-/Ausgänge auf ESP32.....	9
ESP32-Board-Peripheriegeräte.....	11
Installieren Sie ESP32Forth.....	12
Laden Sie ESP32forth herunter.....	12
Kompilieren und Installieren von ESP32forth.....	12
Einstellungen für ESP32 WROOM.....	14
Starten Sie die Kompilierung.....	15
Fehler beim Hochladen der Verbindung beheben.....	17
Warum auf ESP32 in FORTH-Sprache programmieren?.....	19
Präambel.....	19
Grenzen zwischen Sprache und Anwendung.....	20
Was ist ein FORTH-Wort?.....	20
Ein Wort ist eine Funktion?.....	20
FORTH-Sprache im Vergleich zur C-Sprache.....	21
Was FORTH Ihnen im Vergleich zur C-Sprache ermöglicht.....	22
Aber warum ein Stapel statt Variablen?.....	23
Sind Sie überzeugt?.....	23
Gibt es professionelle Bewerbungen, die in FORTH verfasst sind?.....	24
Ein echtes 32-Bit FORTH mit ESP32Forth.....	26
Werte auf dem Datenstapel.....	26
Werte im Gedächtnis.....	26
Textverarbeitung je nach Datengröße oder -typ.....	27
Abschluss.....	28
Wörterbuch / Stapel / Variablen / Konstanten.....	30
Wörterbuch erweitern.....	30
Wörterbuchverwaltung.....	30
Stapel und umgekehrte polnische Notation.....	31
Umgang mit dem Parameterstapel.....	32
Der Return Stack und seine Verwendung.....	33
Speichernutzung.....	33
Variablen.....	33
Konstanten.....	34
Pseudokonstante Werte.....	34
Grundlegende Tools für die Speicherzuweisung.....	35

Textfarben und Anzeigeposition auf dem Terminal.....	36
ANSI-Kodierung von Terminals.....	36
Textfärbung.....	36
Anzeigeposition.....	38
Lokale Variablen mit ESP32Forth.....	40
Einführung.....	40
Der Fake-Stack-Kommentar.....	40
Aktion auf lokale Variablen.....	41
Datenstrukturen für ESP32forth.....	44
Präambel.....	44
Tabellen in FORTH.....	44
Eindimensionales 32-Bit-Datenarray.....	44
Tabellendefinitionswörter.....	45
Lesen und schreiben Sie in eine Tabelle.....	45
Praktisches Beispiel für die Verwaltung eines virtuellen Bildschirms.....	46
Management komplexer Strukturen.....	49
Definition von Sprites.....	51
Reale Zahlen mit ESP32forth.....	53
Die echten mit ESP32forth.....	53
Echte Zahlengenauigkeit mit ESP32forth.....	53
Reale Konstanten und Variablen.....	54
Arithmetische Operatoren für reelle Zahlen.....	54
Mathematische Operatoren für reelle Zahlen.....	54
Logische Operatoren für reelle Zahlen.....	55
Ganzzahlige ↔ reelle Transformationen.....	55
Zahlen und Zeichenfolgen anzeigen.....	57
Änderung der Zahlenbasis.....	57
Definition neuer Anzeigeformate.....	58
Anzeigen von Zeichen und Zeichenfolgen.....	60
String-Variablen.....	62
Wortcode zur Verwaltung von Textvariablen.....	62
Hinzufügen von Zeichen zu einer alphanumerischen Variablen.....	64
Vokabeln mit ESP32forth.....	66
Liste der Vokabeln.....	66
Grundlegende Vokabeln.....	67
Liste der Vokabelinhalte.....	67
Verwendung von Vokabeln.....	67
Verkettung von Vokabeln.....	68
Passen Sie Steckbretter an das ESP32-Board an.....	70
Testplatten für ESP32.....	70
Bauen Sie ein Steckbrett, das für das ESP32-Board geeignet ist.....	70
Stromversorgung der ESP32-Karte.....	72
Wahl der Stromquelle.....	72
Stromversorgung über Mini-USB-Anschluss.....	72
Stromversorgung über 5V-Pin.....	72

Automatischer Start eines Programms.....	74
Installieren und verwenden Sie das Tera Term-Terminal unter Windows.....	76
Installieren Sie Tera Term.....	76
Tera Term einrichten.....	76
Verwendung von Tera Term.....	79
Kompilieren Sie den Quellcode in der Forth-Sprache.....	80
Verwaltung von Quelldateien nach Blöcken.....	82
Die Blöcke.....	82
Öffnen Sie eine Blockdatei.....	82
Bearbeiten Sie den Inhalt eines Blocks.....	83
Blockinhalte zusammenstellen.....	84
Praktisches Schritt-für-Schritt-Beispiel.....	84
Abschluss.....	85
Bearbeiten von Quelldateien mit VISUAL Editor.....	86
Bearbeiten Sie eine FORTH-Quelldatei.....	86
Bearbeiten des FORTH-Codes.....	86
Kompilieren von Dateiinhalten.....	87
Das SPIFFS-Dateisystem.....	88
Zugriff auf das SPIFFS-Dateisystem.....	88
Umgang mit Dateien.....	89
Organisieren und kompilieren Sie Ihre Dateien auf der ESP32-Karte.....	90
Quelldateien bearbeiten und übertragen.....	90
Organisieren Sie Ihre Dateien.....	90
Übertragen Sie eine große Datei an ESP32forth.....	91
Abschluss.....	92
Eine Ampel mit ESP32 managen.....	93
GPIO-Ports auf der ESP32-Karte.....	93
Montage der LEDs.....	94
Verwaltung von Ampeln.....	95
Abschluss.....	95
Hardware-Interrupts mit ESP32forth.....	97
Unterbrechungen.....	97
Montage eines Druckknopfes.....	97
Softwarekonsolidierung des Interrupts.....	98
Weitere Informationen.....	99
Verwendung des Drehgebers KY-040.....	101
Encoder-Übersicht.....	101
Montage des Encoders auf dem Steckbrett.....	102
Analyse von Encodersignalen.....	103
Encoder-Programmierung.....	104
Testen der Kodierung.....	105
Erhöhen und dekrementieren Sie eine Variable mit dem Encoder.....	105
Blinken einer LED pro Timer.....	107
Erste Schritte mit der FORTH-Programmierung.....	107
Blinken nach TIMER.....	108

Hardware- und Software-Interrupts.....	109
Verwenden Sie die Wörter intervall und rerun.....	109
Haushälterin-Timer.....	112
Präambel.....	112
Eine Lösung.....	112
Ein FORTH-Timer für ESP32Forth.....	113
Verwaltung der Licht-Ein-Taste.....	114
Abschluss.....	116
Software-Echtzeituhr.....	117
Das Wort MS-TICKS.....	117
Verwalten einer Softwareuhr.....	117
Messen der Ausführungszeit eines FORTH-Wortes.....	118
Messung der Leistung von FORTH-Definitionen.....	118
Ein paar Schleifen testen.....	119
Installieren der OLED-Bibliothek für SSD1306.....	122
Installieren des HTTP-Clients.....	124
Bearbeiten der Datei ESP32forth.ino.....	124
HTTP-Client-Tests.....	125
Rufen Sie die Uhrzeit von einem WEB-Server ab.....	127
Senden und Empfangen der Uhrzeit von einem Webserver.....	127
Verständnis der Übertragung per GET an einen WEB-Server.....	129
Übertragung von Daten an einen Server per GET.....	129
Parameter in einer URL.....	129
Übergabe mehrerer Parameter.....	129
Verwalten der Parameterübergabe mit ESP32forth.....	130
Datenübertragung an einen WEB-Server.....	132
Datenaufzeichnung auf der Webserverseite.....	132
Zugangsschutz.....	132
Aufgezeichnete Daten anzeigen.....	133
Fügen Sie die zu übertragenden Daten hinzu.....	134
Abschluss.....	136
Der Zufallszahlengenerator.....	137
Charakteristisch.....	137
Programmievorgang.....	138
RND-Funktion im XTENSA-Assembler.....	138
Detaillierter Inhalt der ESP32forth-Vokabulare.....	140
Version v 7.0.7.15.....	140
FORTH.....	140
asm.....	141
bluetooth.....	142
editor.....	142
ESP.....	142
httpd.....	142
insides.....	142

internals.....	142
interrupts.....	143
ledc.....	143
oled.....	143
registers.....	143
riscv.....	143
rtos.....	144
SD.....	144
SD_MMC.....	144
Serial.....	144
sockets.....	144
spi.....	144
SPIFFS.....	144
streams.....	144
structures.....	144
tasks.....	145
telnetd.....	145
visual.....	145
web-interface.....	145
WiFi.....	145
xtensa.....	145
Ressourcen.....	147
Auf Englisch.....	147
Auf Französisch.....	147
GitHub.....	147

Einführung

Seit 2019 verwalte ich mehrere Websites, die sich der FORTH-Sprachentwicklung für ARDUINO- und ESP32-Karten sowie der eForth-Webversion widmen. :

- ARDUINO : <https://arduino-forth.com/>
- ESP32 : <https://esp32.arduino-forth.com/>
- eForth web : <https://eforth.arduino-forth.com/>

Diese Websites sind in zwei Sprachen verfügbar: Französisch und Englisch. Jedes Jahr bezahle ich für das Hosting der Hauptseite arduino-forth.com.

Es wird früher oder später – und zwar so spät wie möglich – passieren, dass ich die Nachhaltigkeit dieser Seiten nicht mehr gewährleisten kann. Die Folge wird sein, dass die von diesen Websites verbreiteten Informationen verschwinden.

Dieses Buch ist die Zusammenstellung von Inhalten meiner Websites. Es wird kostenlos über ein Github-Repository verteilt. Diese Verbreitungsmethode ermöglicht eine größere Nachhaltigkeit als Websites.

Wenn übrigens einige Leser dieser Seiten einen Beitrag leisten möchten, sind sie herzlich willkommen:

- Kapitel vorschlagen ;
- um Fehler zu melden oder Änderungen vorzuschlagen ;
- um bei der Übersetzung zu helfen...

Übersetzungshilfe

Mit Google Translate können Sie Texte einfach, aber mit Fehlern übersetzen. Deshalb bitte ich um Hilfe bei der Korrektur der Übersetzungen.

In der Praxis stelle ich die bereits übersetzten Kapitel im LibreOffice-Format zur Verfügung. Wenn Sie bei diesen Übersetzungen helfen möchten, besteht Ihre Aufgabe lediglich darin, diese Übersetzungen zu korrigieren und zurückzugeben.

Das Korrigieren eines Kapitels nimmt wenig Zeit in Anspruch, von einer bis zu mehreren Stunden.

Um mich zu erreichen : petremann@arduino-forth.com

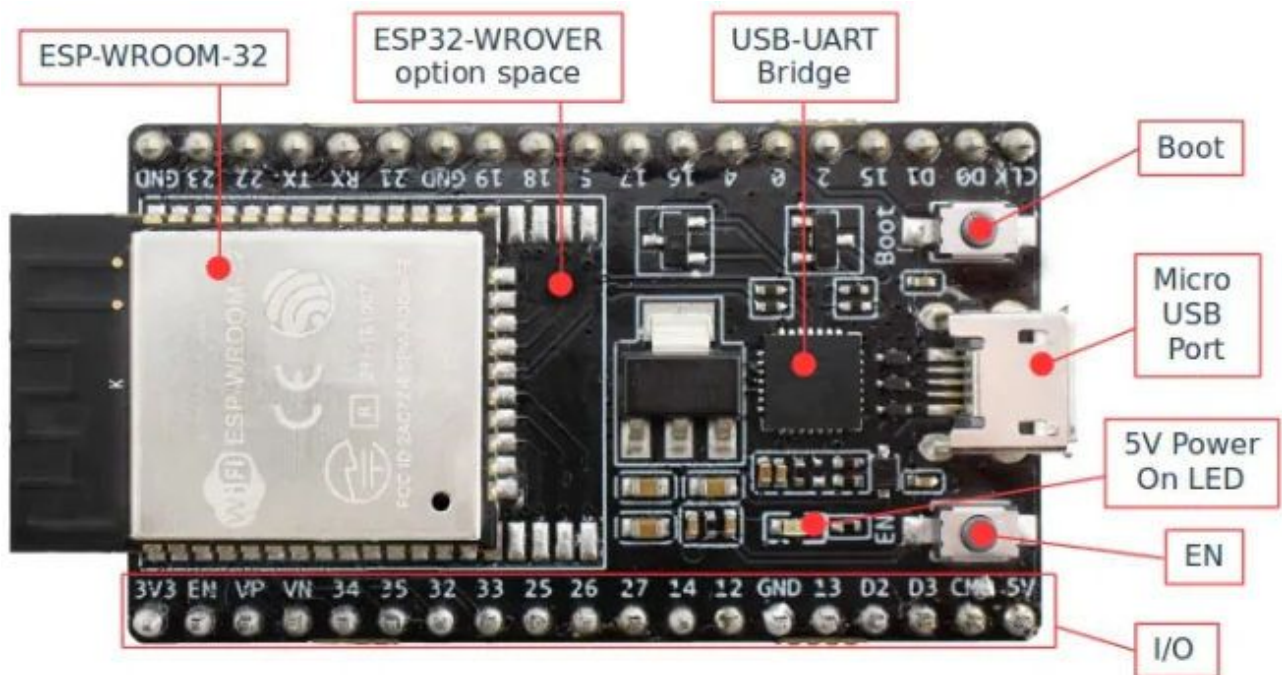
Entdeckung der ESP32-Karte

Präsentation

Das ESP32-Board ist kein ARDUINO-Board. Entwicklungstools nutzen jedoch bestimmte Elemente des ARDUINO-Ökosystems, wie beispielsweise die ARDUINO-IDE.

Die Stärken Punkten

Hinsichtlich der Anzahl der verfügbaren Ports liegt die ESP32-Karte zwischen einem



ARDUINO NANO und ARDUINO UNO. Das Basismodell verfügt über 38 Anschlüsse :

Zu den ESP32-Geräten gehören :

- 18 Analog-Digital-Wandlerkanäle (ADC).
- 3 SPI-Schnittstellen
- 3 UART-Schnittstellen
- 2 I2C-Schnittstellen
- 16 PWM-Ausgangskanäle
- 2 Digital-Analog-Wandler (DAC)
- 2 I2S-Schnittstellen

- 10 kapazitive GPIOs

Die ADC- (Analog-Digital-Wandler) und DAC-Funktionalität (Digital-Analog-Wandler) sind bestimmten statischen Pins zugewiesen. Sie können jedoch entscheiden, welche Pins UART, I2C, SPI, PWM usw. sind. Sie müssen sie nur im Code zuweisen. Dies ist dank der Multiplexing-Funktion des ESP32-Chips möglich.

Die meisten Steckverbinder haben mehrere Verwendungszwecke.

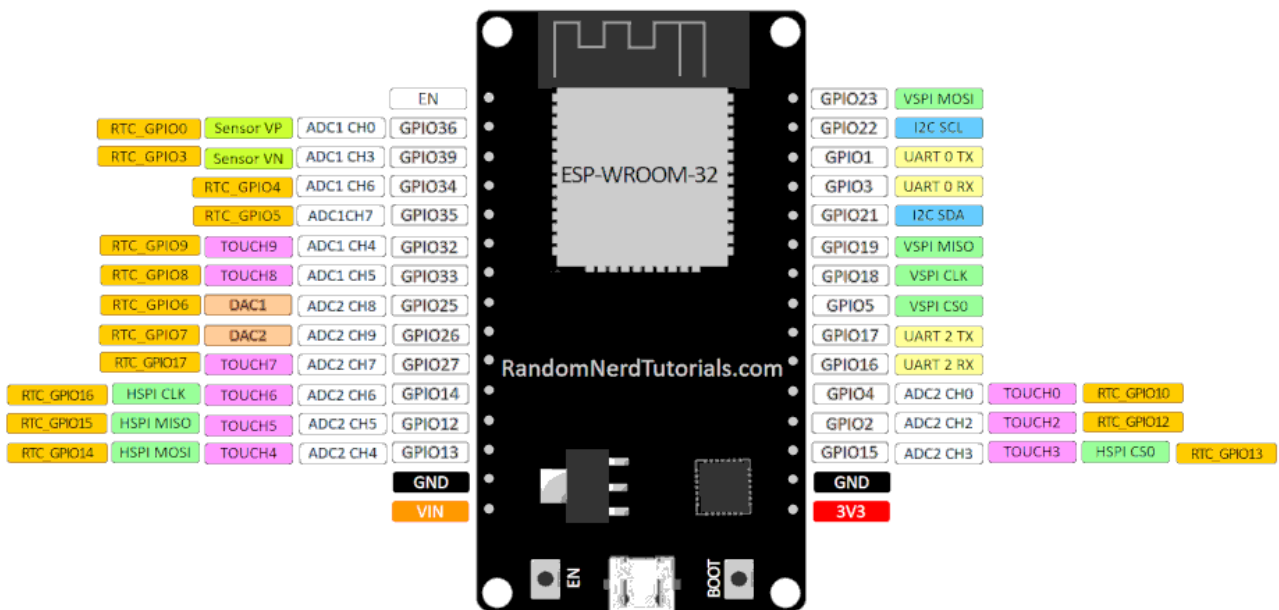
Was das ESP32-Board jedoch auszeichnet, ist, dass es standardmäßig mit WLAN- und Bluetooth-Unterstützung ausgestattet ist, was ARDUINO-Boards nur in Form von Erweiterungen bieten.

GPIO-Ein-/Ausgänge auf ESP32

Hier im Foto die ESP32-Karte, anhand derer wir die Rolle der verschiedenen GPIO-Ein-/Ausgänge erklären :



Die Position und Anzahl der GPIO-I/Os kann sich je nach Kartenmarke ändern. In diesem Fall sind nur die Angaben auf der physischen Karte authentisch. Im Bild, untere Reihe, von links nach rechts: CLK, SD0, SD1, G15, G2, G0, G4, G16.....G22, G23, GND.



In diesem Diagramm sehen wir, dass die untere Reihe mit 3V3 beginnt, während sich dieser I/O auf dem Foto am Ende der oberen Reihe befindet. Daher ist es sehr wichtig, sich nicht auf das Diagramm zu verlassen, sondern den korrekten Anschluss der Peripheriegeräte und Komponenten auf der physischen ESP32-Karte noch einmal zu überprüfen.

Entwicklungsboards auf Basis eines ESP32 verfügen neben denen für die Stromversorgung in der Regel über 33 Pins. Einige GPIO-Pins haben besondere Funktionen :

GPIO	Mögliche Namen
6	SCK/CLK
7	SCK/CLK
8	SDO/SD0
9	SDI/SD1
10	SHD/SD2
11	CSC/CMD

Wenn Ihre ESP32-Karte über I/O GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11 verfügt, sollten Sie diese auf keinen Fall verwenden, da sie mit dem Flash-Speicher des ESP32 verbunden sind. Wenn Sie sie verwenden, funktioniert der ESP32 nicht.

GPIO1(TX0) und GPIO3(RX0) I/O werden für die Kommunikation mit dem Computer in UART über den USB-Port verwendet. Wenn Sie diese verwenden, können Sie nicht mehr mit der Karte kommunizieren.

GPIO36(VP), GPIO39(VN), GPIO34, GPIO35 I/O können nur als Eingang verwendet werden. Sie verfügen auch nicht über eingebaute interne Pullup- und Pulldown-Widerstände.

Mit dem EN-Anschluss können Sie den Zündstatus des ESP32 über ein externes Kabel steuern. Es wird mit der EN-Taste auf der Karte verbunden. Wenn der ESP32 eingeschaltet ist, liegt er bei 3,3 V. Wenn wir diesen Pin mit Masse verbinden, wird der ESP32 ausgeschaltet. Sie können es verwenden, wenn sich der ESP32 in einer Box befindet und Sie ihn mit einem Schalter ein-/ausschalten möchten.

ESP32-Board-Peripheriegeräte

Um mit Modulen, Sensoren oder elektronischen Schaltkreisen zu interagieren, verfügt der ESP32 wie jeder Mikrocontroller über eine Vielzahl an Peripheriegeräten. Davon gibt es mehr als auf einem klassischen Arduino-Board.

ESP32 verfügt über die folgenden Peripheriegeräte :

- 3 UART-Schnittstellen
- 2 I2C-Schnittstellen
- 3 SPI-Schnittstellen
- 16 PWM-Ausgänge
- 10 kapazitive Sensoren
- 18 analoge Eingänge (ADC)
- 2 DAC-Ausgänge

Einige Peripheriegeräte werden bereits im Grundbetrieb von ESP32 genutzt. Somit gibt es pro Gerät weniger mögliche Schnittstellen.

Installieren Sie ESP32Forth

Laden Sie ESP32forth herunter

Der erste Schritt besteht darin, den Quellcode von ESP32forth in C-Sprache wiederherzustellen. Verwenden Sie vorzugsweise die aktuellste Version:

<https://esp32forth.appspot.com/ESP32forth.html>

Inhalt der heruntergeladenen Datei:

```
ESP32forth-7.0.x,x
  ESP32forth
    readme.txt
    esp32forth.ino
    optional
      SPI-flash.h
      serial-blueooth.h
      ... usw...
```

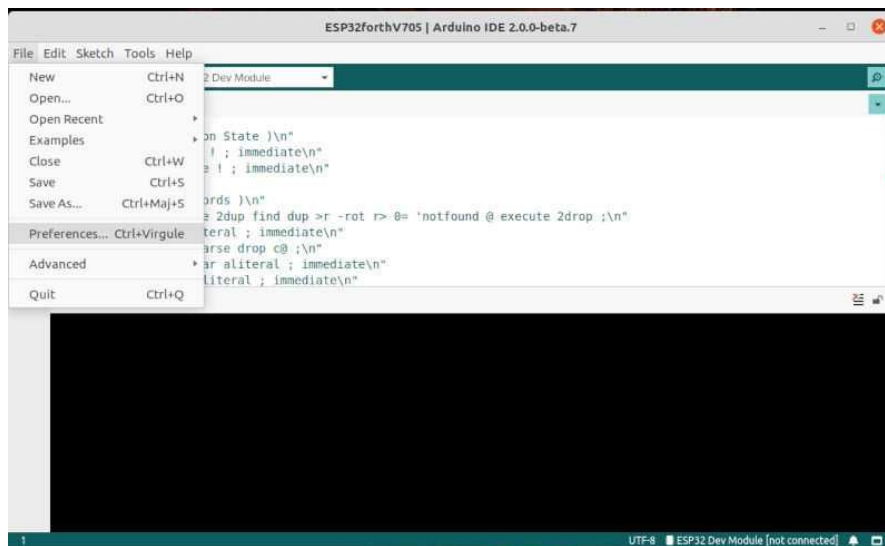
Kompilieren und Installieren von ESP32forth

Datei **esp32forth.ino** in ein Arbeitsverzeichnis. Das optionale Verzeichnis enthält Dateien, die die Erweiterung von ESP32forth ermöglichen. Für unseren ersten Build und Upload von ESP32forth werden diese Dateien nicht benötigt.

Um ESP32forth zu kompilieren, muss ARDUINO IDE bereits auf Ihrem Computer installiert sein:

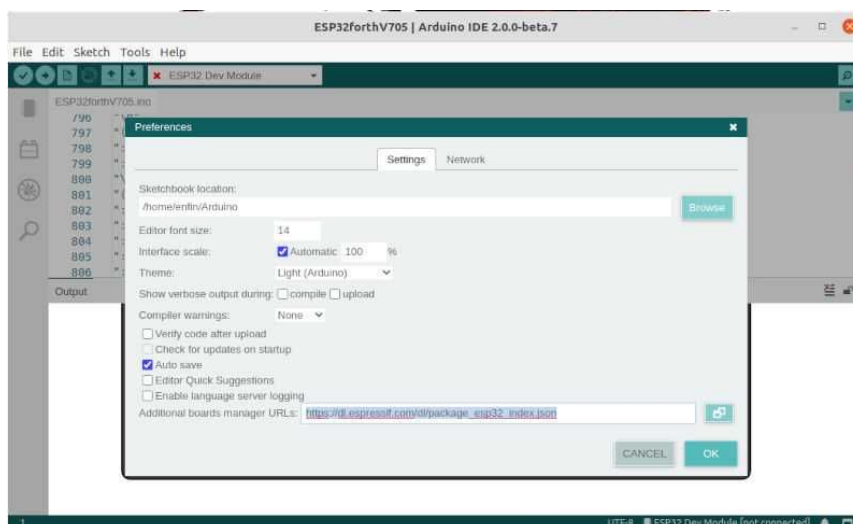
<https://docs.arduino.cc/software/ide-v2>

Sobald ARDUINO IDE installiert ist, starten Sie es. ARDUINO IDE ist geöffnet, hier Version 2.0. Klicken Sie auf *file* und wählen Sie *Preferences* :

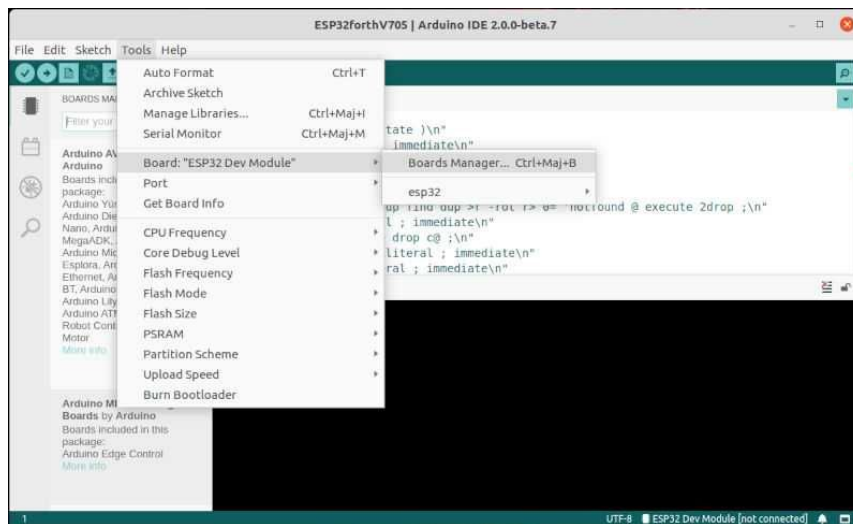


Gehen Sie im angezeigten Fenster zum Eingabefeld mit der Bezeichnung *Additional boards manager URLs* : und geben Sie diese Zeile ein:

https://dl.espressif.com/dl/package_esp32_index.json



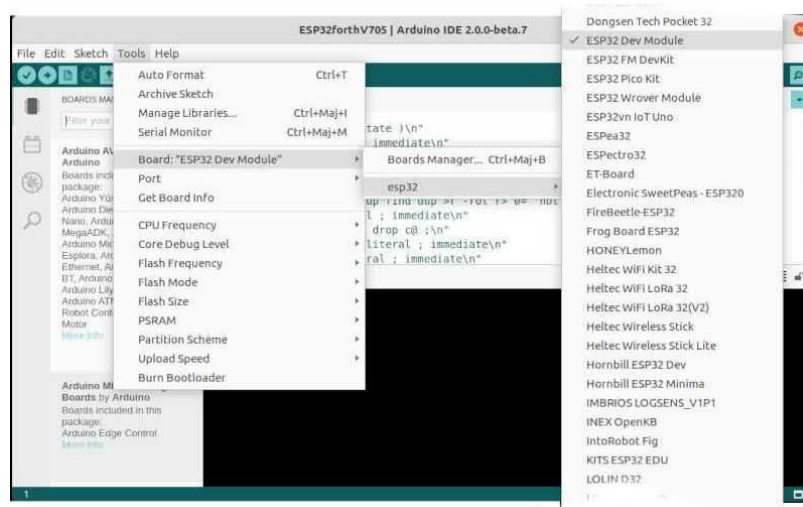
Klicken Sie anschließend auf *Tools* und wählen Sie *Board* :



Diese Auswahl sollte Ihnen die Installation von Paketen für ESP32 anbieten. Akzeptieren Sie diese Installation.

Anschließend sollten Sie auf die Auswahl der ESP32-Karten zugreifen können:

Auswahl der **ESP32 Dev Module** platine :



Einstellungen für ESP32 WROOM

Hier sind die anderen Einstellungen, die vor dem Kompilieren von ESP32forth erforderlich sind. Greifen Sie auf die Einstellungen zu, indem Sie erneut auf *Extras klicken* :

```
-- TOOLS-----+-- BOARD      -----+-- ESP32 -----+-- ESP32 Dev Module
+-- Port: -----+-- COMx
|
```

```

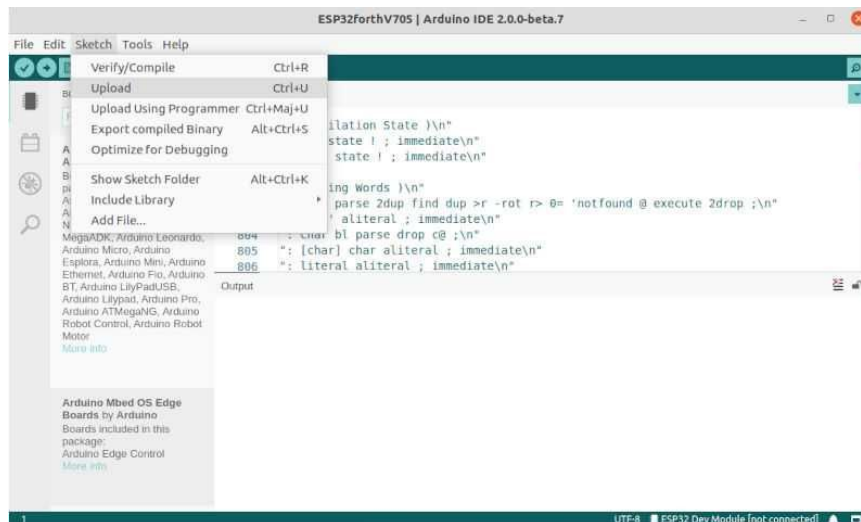
+-- CPU Frequency -----+-- 240 Mhz
+-- Core Debug Level -----+-- None
+-- Erase All Flash...-----+-- Disabled
+-- Events Run On -----+-- Core 1
+-- Flash Frequency -----+-- 80 Mhz
+-- Flash Mode -----+-- QIO
+-- Flash Size -----+-- 4MB
+-- JTAG Adapter -----+-- FTDI Adapter
+-- Arduino Runs on -----+-- Core 1
+-- PSRAM -----+-- Disabled
+-- Partition Scheme -----+-- Default 4MB with SPIFFS
+-- Upload Speed -----+-- 921600

```

Starten Sie die Kompilierung

Jetzt muss nur noch ESP32forth kompiliert werden. Laden Sie den Quellcode mit *File* und *Open*.

Es wird davon ausgegangen, dass Ihr ESP32-Board an einen USB-Anschluss angeschlossen ist. Starten Sie die Zusammenstellung, indem Sie auf *Sketch* und *Upload* auswählen :



Wenn alles korrekt läuft, sollten Sie den Binärcode automatisch in die ESP32-Karte übertragen. Wenn die Kompilierung fehlerfrei verläuft, aber ein Übertragungsfehler vorliegt, kompilieren Sie die Datei **esp32forth.ino neu** . Drücken Sie zum Zeitpunkt der Übertragung die mit **BOOT gekennzeichnete Taste** auf der ESP32-Karte. Dadurch sollte die Karte für die Übertragung des ESP32forth-Binärcodes verfügbar sein.

Installation und Konfiguration der ARDUINO IDE im Video:

- Windows: <https://www.youtube.com/watch?v=2AZQfieHv9g>
- Linux: https://www.youtube.com/watch?v=JeD3nz0__nc

Fehler beim Hochladen der Verbindung beheben

Erfahren Sie, wie Sie den schwerwiegenden Fehler beheben können, der beim Versuch, ein für alle Mal einen neuen Code auf Ihre ESP32-Karte hochzuladen, aufgetreten ist: "Failed to connect to ESP32: Timed out waiting for packet header".

Einige ESP32-Entwicklungsboards (siehe „Beste ESP32-Boards“) wechseln beim Herunterladen von neuem Code nicht automatisch in den Flash-/Upload-Modus.

Das bedeutet, dass beim Versuch, eine neue Skizze auf Ihr ESP32-Board hochzuladen, ARDUINO IDE keine Verbindung zu Ihrem Board herstellen kann und Sie die folgende Fehlermeldung erhalten:

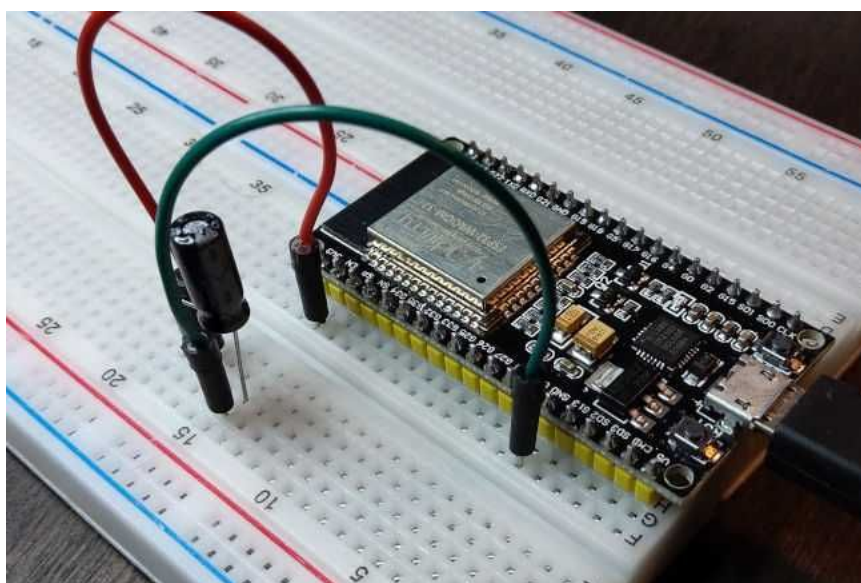


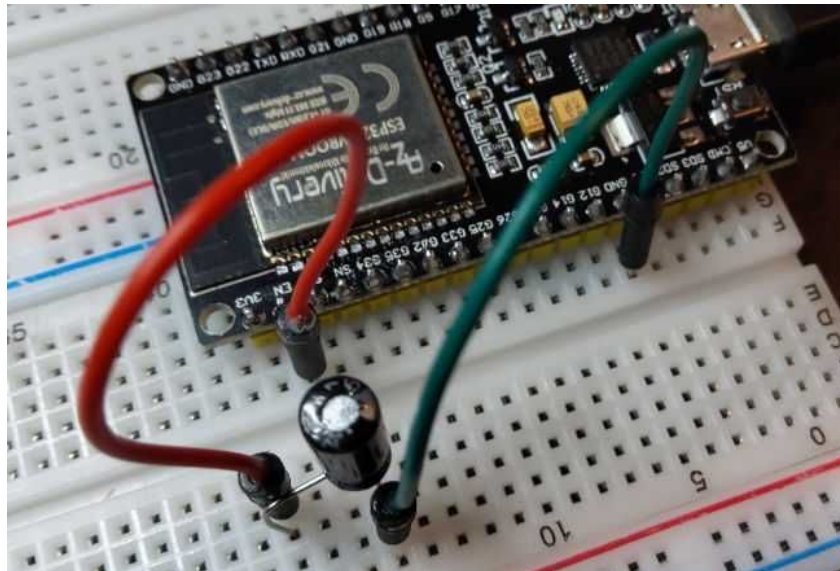
```
Blink
7 | it is attached to digital pin 13, on MKR1000 on pin 6. LED BUILTIN is set to

A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header
python /home/enfin/.arduino15/packages/esp32/tools/esptool_py/3.0.0/esptool.py --chip esp32 -
esptool.py v3.0-dev
python /home/enfin/.arduino15/packages/esp32/hardware/esp32/1.0.6/tools/gen_esp32part.py -q /
/home/enfin/.arduino15/packages/esp32/tools/xtensa-esp32-elf-gcc/1.22.0-97-gc752ad5-5.2.0/bin
Le croquis utilise 198842 octets (15%) de l'espace de stockage de programmes. Le maximum est
Les variables globales utilisent 13248 octets (4%) de mémoire dynamique, ce qui laisse 314432
python /home/enfin/.arduino15/packages/esp32/tools/esptool_py/3.0.0/esptool.py --chip esp32 -
esptool.py v3.0-dev
Serial port /dev/ttyUSB0
Connecting.....
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header

abled, Default 4MB with spiiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), Q10, 80MHz, 4MB (32Mb), 921600, None sur /dev/ttyUSB0
```

Damit das ESP32-Board automatisch in den Flash-/Download-Modus wechselt, können wir einen 10uF-Elektrolytkondensator zwischen dem EN- und dem GND-Pin anschließen:





Diese Manipulation ist nur erforderlich, wenn Sie sich in der Hochladephase von ESP32forth von der ARDUINO IDE befinden. Sobald ESP32forth auf der ESP32-Platine installiert ist, ist die Verwendung dieses Kondensators nicht mehr notwendig.

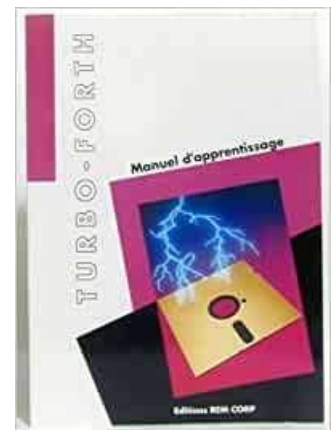
Warum auf ESP32 in FORTH-Sprache programmieren?

Präambel

Ich programmiere seit 1983 in FORTH. Ich habe 1996 mit dem Programmieren in FORTH aufgehört. Aber ich habe nie aufgehört, die Entwicklung dieser Sprache zu verfolgen. Ich habe 2019 wieder mit dem Programmieren auf ARDUINO mit FlashForth und dann mit ESP32forth begonnen.

Ich bin Co-Autor mehrerer Bücher über die FORTH-Sprache :

- Introduction au ZX-FORTH (ed Eyrolles - 1984 - ASIN:B0014IGOXO)
- Tours de FORTH (ed Eyrolles - 1985 - ISBN-13: 978-2212082258)
- FORTH pour CP/M et MSDOS (ed Loisetech - 1986)
- TURBO-Forth, manuel d'apprentissage (ed Rem CORP - 1990)
- TURBO-Forth, guide de référence (ed Rem CORP - 1991)



Das Programmieren in der FORTH-Sprache war schon immer ein Hobby, bis mich 1992 der Manager eines Unternehmens kontaktierte, das als Zulieferer für die Automobilindustrie tätig war. Sie hatten ein Interesse an der Softwareentwicklung in der Sprache C. Sie mussten einen Industrieautomaten bestellen.

Die beiden Softwareentwickler dieser Firma programmierten in der Sprache C: TURBO-C von Borland, um genau zu sein. Und ihr Code konnte nicht kompakt und schnell genug sein, um in den 64 Kilobyte großen RAM-Speicher zu passen. Es war 1992 und Flash-Speichererweiterungen gab es noch nicht. In diesen 64 KB RAM mussten wir MS-DOS 3.0 und die Anwendung unterbringen!

Einen Monat lang hatten C-Entwickler das Problem in alle Richtungen umgedreht, sogar Reverse Engineering mit SOURCER (einem Disassembler), um nicht wesentliche Teile des ausführbaren Codes zu entfernen.

Ich habe das mir vorgelegte Problem analysiert. Von Grund auf habe ich innerhalb einer Woche alleine einen perfekt funktionsfähigen Prototyp erstellt, der den Spezifikationen entsprach. Drei Jahre lang, von 1992 bis 1995, habe ich zahlreiche Versionen dieser Anwendung erstellt, die auf den Montagebändern mehrerer Automobilhersteller eingesetzt wurden.

Grenzen zwischen Sprache und Anwendung

Alle Programmiersprachen werden auf diese Weise geteilt :

- ein Interpreter und ausführbarer Quellcode: BASIC, PHP, MySQL, JavaScript usw. Die Anwendung ist in einer oder mehreren Dateien enthalten, die bei Bedarf interpretiert werden. Das System muss den Interpreter, der den Quellcode ausführt, dauerhaft hosten ;
- ein Compiler und/oder Assembler: C, Java usw. Einige Compiler generieren nativen Code, also speziell auf einem System ausführbar. Andere, wie Java, kompilieren ausführbaren Code auf einer virtuellen Java-Maschine.

Eine Ausnahme bildet die FORTH-Sprache. Es integriert :

- ein Dolmetscher, der jedes Wort in der FORTH-Sprache ausführen kann
- ein Compiler, der das Wörterbuch der FORTH-Wörter erweitern kann

Was ist ein FORTH-Wort?

Ein FORTH-Wort bezeichnet einen beliebigen Wörterbuchausdruck, der aus ASCII-Zeichen besteht und bei der Interpretation und/oder Kompilierung verwendet werden kann: Mit Wörter können Sie alle Wörter im FORTH-Wörterbuch auflisten.

Bestimmte FORTH-Wörter können nur bei der Kompilierung verwendet werden: **if else then** zum Beispiel.

Bei der FORTH-Sprache besteht das wesentliche Prinzip darin, dass wir keine Anwendung erstellen. In FORTH erweitern wir das Wörterbuch! Jedes neue Wort, das Sie definieren, ist ebenso Teil des FORTH-Wörterbuchs wie alle beim Start von FORTH vordefinierten Wörter. Beispiel :

```
: typeToLoRa ( -- )
  0 echo !      \ Deaktivieren Sie das Echo der Terminalanzeige
  ['] serial2-type is type
;
: typeToTerm ( -- )
  ['] default-type is type
  -1 echo !     \ Aktiviert das Display-Echo des Terminals
;
```

Wir erstellen zwei neue Wörter: **typeToLoRa** und **typeToTerm**, die das Wörterbuch vordefinierter Wörter vervollständigen.

Ein Wort ist eine Funktion?

Ja und nein. Tatsächlich kann ein Wort eine Konstante, eine Variable, eine Funktion sein... Hier in unserem Beispiel die folgende Sequenz :

```
: typeToLoRa ...code... ;
```

hätte sein Äquivalent in der C-Sprache :

```
void typeToLoRa() { ...code... }
```

In der FORTH-Sprache gibt es keine Grenze zwischen Sprache und Anwendung.

In FORTH können Sie wie in der Sprache C jedes bereits definierte Wort in der Definition eines neuen Worts verwenden.

Ja, aber warum dann FORTH statt C?

Ich habe diese Frage erwartet.

In der C-Sprache kann auf eine Funktion nur über die Hauptfunktion main() zugegriffen werden. Wenn diese Funktion mehrere Zusatzfunktionen integriert, wird es bei einer Fehlfunktion des Programms schwierig, einen Parameterfehler zu finden.

Im Gegenteil, mit FORTH ist es möglich, über den Interpreter jedes vordefinierte oder von Ihnen definierte Wort auszuführen, ohne das Hauptwort des Programms durchlaufen zu müssen.

Der FORTH-Interpreter ist über ein Terminalprogramm und eine USB-Verbindung zwischen der ESP32-Karte und dem PC sofort auf der ESP32-Karte zugänglich.

Die Kompilierung von in FORTH-Sprache geschriebenen Programmen erfolgt in der ESP32-Karte und nicht auf dem PC. Es erfolgt kein Upload. Beispiel:

```
: >gray ( n -- n' )  
  dup 2/ xor      \ n' = n xor ( 1 logische Verschiebung nach rechts )  
;
```

Diese Definition wird per Kopieren/Einfügen in das Terminal übertragen. Der FORTH-Interpreter/Compiler analysiert den Stream und kompiliert das neue Wort **>gray**.

In der Definition von **>gray** sehen wir die Sequenz **dup 2/ xor**. Um diese Sequenz zu testen, geben Sie sie einfach in das Terminal ein. Um **>gray** auszuführen, geben Sie einfach dieses Wort in das Terminal ein, gefolgt von der Zahl, die umgewandelt werden soll.

FORTH-Sprache im Vergleich zur C-Sprache

Das ist der Teil, den ich am wenigsten mag. Ich vergleiche die FORTH-Sprache nicht gern mit der C-Sprache. Aber da fast alle Entwickler die C-Sprache verwenden, werde ich die Übung ausprobieren.

Hier ist ein Test mit **if()** in C-Sprache:

```
if(j > 13){                // Wenn alle Bits empfangen wurden  
    rc5_ok = 1;            // Der Dekodierungsprozess ist OK  
    detachInterrupt(0);    // Externen Interrupt deaktivieren (INT0)
```

```

    return;
}

```

Testen Sie mit if in FORTH-Sprache (Code-Snippet) :

```

var-j @ 13 >          \ Wenn alle Bits empfangen wurden
    if
        1 rc5_ok !    \ Der Dekodierungsprozess ist OK
        di            \ Externen Interrupt deaktivieren (INT0)
        exit
    then

```

Hier ist die Initialisierung von Registern in C-Sprache :

```

void setup() {
    // Konfigurieren des Timer1-Moduls
    TCCR1A = 0;
    TCCR1B = 0;          // Deaktiviert das Timer1-Modul
    TCNT1  = 0;          // Setzt den Vorladewert von Timer1 auf 0 (reset)
    TIMSK1 = 1;          // Überlauf-Interrupt aktivieren Timer1
}

```

Die gleiche Definition in der FORTH-Sprache:

```

: setup ( -- )
    \ Konfigurieren des Timer1-Moduls
    0 TCCR1A !
    0 TCCR1B !    \ Deaktiviert das Timer1-Modul
    0 TCNT1  !    \ Setzt den Vorladewert von Timer1 auf 0 (reset)
    1 TIMSK1 !    \ Überlauf-Interrupt aktivieren Timer1
;

```

Was FORTH Ihnen im Vergleich zur C-Sprache ermöglicht

Wir verstehen, dass FORTH sofort Zugriff auf alle Wörter im Wörterbuch bietet, aber nicht nur darauf. Über den Interpreter greifen wir auch auf den gesamten Speicher der ESP32-Karte zu. Stellen Sie eine Verbindung zum ARDUINO-Board her, auf dem FlashForth installiert ist, und geben Sie dann einfach Folgendes ein:

```
hex here 100 dump
```

Sie sollten dies auf dem Terminalbildschirm finden :

```

3FFEE964          DF DF 29 27 6F 59 2B 42 FA CF 9B 84
3FFEE970          39 4E 35 F7 78 FB D2 2C A0 AD 5A AF 7C 14 E3 52
3FFEE980          77 0C 67 CE 53 DE E9 9F 9A 49 AB F7 BC 64 AE E6
3FFEE990          3A DF 1C BB FE B7 C2 73 18 A6 A5 3F A4 68 B5 69
3FFEE9A0          F9 54 68 D9 4D 7C 96 4D 66 9A 02 BF 33 46 46 45
3FFEE9B0          45 39 33 33 2F 0D 08 18 BF 95 AF 87 AC D0 C7 5D
3FFEE9C0          F2 99 B6 43 DF 19 C9 74 10 BD 8C AE 5A 7F 13 F1
3FFEE9D0          9E 00 3D 6F 7F 74 2A 2B 52 2D F4 01 2D 7D B5 1C
3FFEE9E0          4A 88 88 B5 2D BE B1 38 57 79 B2 66 11 2D A1 76

```

3FFEE9F0	F6 68 1F 71 37 9E C1 82 43 A6 A4 9A 57 5D AC 9A
3FFEEA00	4C AD 03 F1 F8 AF 2E 1A B4 67 9C 71 25 98 E1 A0
3FFEEA10	E6 29 EE 2D EF 6F C7 06 10 E0 33 4A E1 57 58 60
3FFEEA20	08 74 C6 70 BD 70 FE 01 5D 9D 00 9E F7 B7 E0 CA
3FFEEA30	72 6E 49 16 0E 7C 3F 23 11 8D 66 55 EC F6 18 01
3FFEEA40	20 E7 48 63 D1 FB 56 77 3E 9A 53 7D B6 A7 A5 AB
3FFEEA50	EA 65 F8 21 3D BA 54 10 06 16 E6 9E 23 CA 87 25
3FFEEA60	E7 D7 C4 45

Dies entspricht dem Inhalt des Flash-Speichers.

Und die C-Sprache konnte das nicht?

Ja, aber nicht so einfach und interaktiv wie in der FORTH-Sprache.

Aber warum ein Stapel statt Variablen?

Der Stack ist ein Mechanismus, der auf fast allen Mikrocontrollern und Mikroprozessoren implementiert ist. Sogar die C-Sprache nutzt einen Stack, aber Sie haben keinen Zugriff darauf.

Nur die FORTH-Sprache bietet vollständigen Zugriff auf den Datenstapel. Um beispielsweise eine Addition durchzuführen, stapeln wir zwei Werte, führen die Addition aus und zeigen das Ergebnis an: **2 5 + .** zeigt **7** an.

Es ist ein wenig destabilisierend, aber wenn Sie den Mechanismus des Datenstapels verstehen, werden Sie seine beeindruckende Effizienz sehr zu schätzen wissen.

Mit dem Datenstapel können Daten viel schneller zwischen FORTH-Worten übertragen werden als durch die Verarbeitung von Variablen wie in der C-Sprache oder einer anderen Sprache, die Variablen verwendet.

Sind Sie überzeugt?

Persönlich bezweifle ich, dass dieses einzelne Kapitel Sie endgültig zum Programmieren in der FORTH-Sprache bekehren wird. Wenn Sie ESP32-Boards beherrschen möchten, haben Sie zwei Möglichkeiten :

- Programmieren Sie das Programm in C-Sprache und nutzen Sie die zahlreichen verfügbaren Bibliotheken. Sie bleiben jedoch an die Möglichkeiten dieser Bibliotheken gebunden. Die Anpassung von Codes an die C-Sprache erfordert echte Programmierkenntnisse in der C-Sprache und die Beherrschung der Architektur von ESP32-Karten. Die Entwicklung komplexer Programme wird immer ein Problem sein.
- Probieren Sie das FORTH-Abenteuer aus und erkunden Sie eine neue und aufregende Welt. Natürlich wird es nicht einfach sein. Sie müssen die Architektur von ESP32-Karten, die Register und die Registerflags im Detail verstehen. Im

Gegenzug erhalten Sie Zugang zu einer Programmierung, die perfekt zu Ihren Projekten passt.

Gibt es professionelle Bewerbungen, die in FORTH verfasst sind?

Oh ja! Beginnend mit dem HUBBLE-Weltraumteleskop, dessen bestimmte Komponenten in der FORTH-Sprache geschrieben wurden.

Der deutsche TGV ICE (Intercity Express) nutzt RTX2000-Prozessoren zur Steuerung von Motoren über Leistungshalbleiter. Die Maschinensprache des RTX2000-Prozessors ist die FORTH-Sprache.

Derselbe RTX2000-Prozessor wurde für die Philae-Sonde verwendet, die versuchte, auf einem Kometen zu landen.

Die Wahl der FORTH-Sprache für professionelle Anwendungen erweist sich als interessant, wenn wir jedes Wort als Blackbox betrachten. Jedes Wort muss einfach sein, daher eine relativ kurze Definition haben und von wenigen Parametern abhängen.

Während der Debugging-Phase ist es einfach, alle möglichen Werte zu testen, die von diesem Wort verarbeitet werden. Sobald dieses Wort vollkommen zuverlässig ist, wird es zu einer Blackbox, also zu einer Funktion, deren ordnungsgemäßes Funktionieren wir absolut vertrauen können. Von Wort zu Wort ist es in FORTH einfacher, ein komplexes Programm zuverlässig zu machen als in jeder anderen Programmiersprache.

Aber wenn es uns an Genauigkeit mangelt, wenn wir Gasanlagen bauen, ist es auch sehr leicht, dass eine Anwendung schlecht funktioniert oder sogar völlig abstürzt!

Schließlich ist es in der FORTH-Sprache möglich, die von Ihnen definierten Wörter in jeder menschlichen Sprache zu schreiben. Allerdings sind die verwendbaren Zeichen auf den ASCII-Zeichensatz zwischen 33 und 127 beschränkt. So könnten wir die Wörter **high** und **low** symbolisch umschreiben :

```
\ Aktiver Port-Pin, andere nicht ändern.  
: __/ ( pinmask portadr -- )  
  mset  
;  
\ Das Deaktivieren eines Port-Pins hat keine Auswirkungen auf die anderen.  
: \__ ( pinmask portadr -- )  
  mclr  
;
```

Ab diesem Moment können Sie zum Einschalten der LED Folgendes eingeben :

```
_o_ __/ \ allume LED
```

Ja! Die Sequenz **_o_ __/** ist in FORTH-Sprache!

Mit ESP32forth stehen Ihnen hier alle Zeichen zur Verfügung, die ein FORTH-Wort bilden können:

```
~}|{|zyxwvutsrqponmlkjihgfedcba`_  
^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?  
>=<;:9876543210/.-, +*) ('&%$#"!
```

Gute Programmierung.

Ein echtes 32-Bit FORTH mit ESP32Forth

ESP32Forth ist ein echtes 32-Bit FORTH. Was bedeutet das ?

Die FORTH-Sprache bevorzugt die Manipulation ganzzahliger Werte. Diese Werte können Literalwerte, Speicheradressen, Registerinhalte usw. sein.

Werte auf dem Datenstapel

Wenn ESP32Forth startet, ist der FORTH-Interpreter verfügbar. Wenn Sie eine beliebige Zahl eingeben, wird diese als 32-Bit-Ganzzahl auf dem Stapel abgelegt:

```
35
```

Wenn wir einen anderen Wert stapeln, wird dieser ebenfalls gestapelt. Der vorherige Wert wird um eine Position nach unten verschoben:

```
45
```

Um diese beiden Werte zu addieren, verwenden wir ein Wort, hier **+** :

```
+
```

Unsere beiden 32-Bit-Ganzzahlwerte werden addiert und das Ergebnis auf dem Stapel abgelegt. Um dieses Ergebnis anzuzeigen, verwenden wir das Wort **.** :

```
. \ zeigt 80 an
```

In der FORTH-Sprache können wir alle diese Operationen in einer einzigen Zeile konzentrieren:

```
35 45 +. \ 80 anzeigen
```

Im Gegensatz zur C-Sprache definieren wir keinen **int8-**, **int16-** oder **int32- Typ** .

Mit ESP32Forth wird ein ASCII-Zeichen durch eine 32-Bit-Ganzzahl bezeichnet, deren Wert jedoch auf [32..256[begrenzt ist. Beispiel :

```
67 emit \ zeigt C
```

Werte im Gedächtnis

Mit ESP32Forth können Sie Konstanten und Variablen definieren. Ihr Inhalt liegt immer im 32-Bit-Format vor. Aber es gibt Situationen, in denen uns das nicht unbedingt passt. Nehmen wir ein einfaches Beispiel und definieren ein Morsecode-Alphabet. Wir brauchen nur ein paar Bytes:

- eines, um die Anzahl der Morsezeichen zu definieren

- ein oder mehrere Bytes für jeden Buchstaben des Morsecodes

```
create mA ( -- addr )
  2 c,
  char . c,   char - c,

create mB ( -- addr )
  4 c,
  char - c,   char . c,   char . c,   char . c,

create mC ( -- addr )
  4 c,
  char - c,   char . c,   char - c,   char . c,
```

Hier definieren wir nur 3 Wörter, **mA** , **mB** und **mC** . In jedem Wort werden mehrere Bytes gespeichert. Die Frage ist : Wie werden wir die Informationen in diesen Worten abrufen ?

Die Ausführung eines dieser Wörter hinterlegt einen 32-Bit-Wert, einen Wert, der der Speicheradresse entspricht, an der wir unsere Morsecode-Informationen gespeichert haben. Es ist das Wort **c@** , das wir verwenden werden, um den Morsecode aus jedem Buchstaben zu extrahieren :

```
mA c@ . \ zeigt 2 an
mB c@ . \ zeigt 4 an
```

Das erste so extrahierte Byte wird zur Verwaltung einer Schleife zur Anzeige des Morsecodes eines Buchstabens verwendet :

```
: .morse ( addr -- )
  dup 1+ swap c@ 0 do
    dup i + c@ emit
  loop
  drop
;
mA .morse \ zeigt .-
mB .morse \ zeigt ...
mC .morse \ zeigt --.
```

Es gibt sicherlich viele elegantere Beispiele. Hier soll eine Möglichkeit gezeigt werden, 8-Bit-Werte, unsere Bytes, zu manipulieren, während wir diese Bytes auf einem 32-Bit-Stack verwenden.

Textverarbeitung je nach Datengröße oder -typ

In allen anderen Sprachen gibt es ein generisches Wort wie **echo** (in PHP), das jede Art von Daten anzeigt. Ob Integer, Real, String, wir verwenden immer das gleiche Wort. Beispiel in PHP-Sprache:

```
$bread = "Gebackenes Brot";
$preis = 2,30;
echo $bread . " : " . $preis;
```

```
// zeigt Gebackenes Brot: 2,30
```

Für alle Programmierer ist diese Vorgehensweise DER STANDARD! Wie würde FORTH dieses Beispiel in PHP umsetzen?

```
: bread s" Baked bread" ;  
: price s" 2.30" ;  
bread type    s" : " type    price type  
\ display    Baked bread: 2.30
```

Hier sagt uns der **type** , dass wir gerade eine Zeichenfolge verarbeitet haben.

Wo PHP (oder eine andere Sprache) über eine generische Funktion und einen Parser verfügt, kompensiert FORTH dies mit einem einzigen Datentyp, aber angepassten Verarbeitungsmethoden, die uns über die Art der verarbeiteten Daten informieren.

Hier ist ein absolut trivialer Fall für FORTH, bei dem eine Anzahl von Sekunden im Format HH:MM:SS angezeigt wird:

```
: :##  
  # 6 base !  
  # decimal  
  [char] : hold  
;  
: .hms ( n -- )  
  <# :## :## # # #> type  
;  
4225 .hms \ zeigt: 01:10:25
```

Ich liebe dieses Beispiel, weil bisher **KEINE ANDERE PROGRAMMIERSPRACHE** in der Lage ist, diese HH:MM:SS-Konvertierung so elegant und prägnant durchzuführen.

Sie haben verstanden, das Geheimnis von FORTH liegt in seinem Wortschatz.

Abschluss

FORTH hat keine Datentypisierung. Alle Daten durchlaufen einen Datenstapel. Jede Position im Stapel ist IMMER eine 32-Bit-Ganzzahl!

Das ist alles, was man wissen muss.

Puristen hyperstrukturierter und ausführlicher Sprachen wie C oder Java werden sicherlich Häresie ausrufen. Und hier erlaube ich mir, sie zu beantworten : Warum müssen Sie Ihre Daten eingeben ?

Denn in dieser Einfachheit liegt die Stärke von FORTH: ein einzelner Datenstapel mit einem untypisierten Format und sehr einfachen Operationen.

Und ich zeige Ihnen, was viele andere Programmiersprachen nicht können, nämlich neue Definitionswörter zu definieren :

```

: morse: ( comp: c -- | exec -- )
  create
    c,
  does>
    dup 1+ swap c@ 0 do
      dup i + c@ emit
    loop
  drop space
;
2 morse: mA      char . c,   char - c,
4 morse: mB      char - c,   char . c,   char . c,   char . c,
4 morse: mC      char - c,   char . c,   char - c,   char . c,
mA mB mC      \ zeigt  .- -... -.-.

```

Hier ist das Wort **morse:** zu einem Definitionswort geworden, ebenso wie **constant** oder **variable** ...

Denn FORTH ist mehr als eine Programmiersprache. Es handelt sich um eine Metasprache, also um eine Sprache zum Aufbau einer eigenen Programmiersprache....

Wörterbuch / Stapel / Variablen / Konstanten

Wörterbuch erweitern

Forth gehört zur Klasse der gewebten Interpretationssprachen. Das bedeutet, dass es auf der Konsole eingegebene Befehle interpretieren sowie neue Unterroutinen und Programme kompilieren kann.

Der Forth-Compiler ist Teil der Sprache und spezielle Wörter werden verwendet, um neue Wörterbucheinträge (d. h. Wörter) zu erstellen. Die wichtigsten sind **:** (eine neue Definition beginnen) und **;** (beendet die Definition). Versuchen wir es, indem wir Folgendes eingeben:

```
: *+ * + ;
```

Was ist passiert? Die Aktion von: besteht darin, einen neuen Wörterbucheintrag mit dem Namen ***+** zu erstellen und vom Interpretationsmodus in den Kompilierungsmodus zu wechseln. Im Kompilierungsmodus sucht der Interpreter nach Wörtern und anstatt sie auszuführen, installiert er Zeiger auf ihren Code. Wenn es sich bei dem Text um eine Zahl handelt, legt ESP32forth die Zahl nicht auf den Stapel, sondern erstellt sie im für das neue Wort zugewiesenen Wörterbuchraum. Dabei folgt er einem speziellen Code, der die gespeicherte Zahl bei jeder Ausführung des Worts auf den Stapel legt. Die Ausführungsaktion von ***+** besteht daher darin, die zuvor definierten Wörter ***** und **+** nacheinander auszuführen.

Das Wort **;** ist besonders. Es ist ein unmittelbares Wort und wird immer ausgeführt, auch wenn sich das System im Kompilierungsmodus befindet. Was bedeutet **;** ist zweifach. Erstens wird Code installiert, der die Kontrolle an die nächste externe Ebene des Interpreters zurückgibt, und zweitens kehrt es vom Kompilierungsmodus in den Interpretationsmodus zurück.

Probieren Sie jetzt Ihr neues Wort aus:

```
dezimal 5 6 7 *+ . \ zeigt 47 ok<#,ram> an
```

Dieses Beispiel veranschaulicht zwei Hauptarbeitsaktivitäten in Forth: das Hinzufügen eines neuen Worts zum Wörterbuch und das Ausprobieren, sobald es definiert wurde.

Wörterbuchverwaltung

Das Wort **forget** gefolgt vom zu löschenden Wort entfernt alle Wörterbucheinträge, die Sie seit diesem Wort gemacht haben:

```
: test1 ;
```

```
: test2 ;  
: test3 ;  
forget test2 \ test2 und test3 aus dem Wörterbuch löschen
```

Stapel und umgekehrte polnische Notation

Forth verfügt über einen explizit sichtbaren Stapel, der zum Übergeben von Zahlen zwischen Wörtern (Befehlen) verwendet wird. Die effektive Verwendung von Forth zwingt Sie dazu, im Stapel zu denken. Das kann am Anfang schwierig sein, aber wie bei allem wird es mit der Übung viel einfacher.

In FORTH ist der Stapel analog zu einem Kartenstapel mit darauf geschriebenen Zahlen. Zahlen werden immer oben auf dem Stapel hinzugefügt und oben vom Stapel entfernt. ESP32forth integriert zwei Stacks: den Parameter-Stack und den Feedback-Stack, die jeweils aus einer Reihe von Zellen bestehen, die 16-Bit-Zahlen aufnehmen können.

Die FORTH-Eingabezeile:

```
dezimal 2 5 73 -16
```

lässt den Parameterstapel unverändert

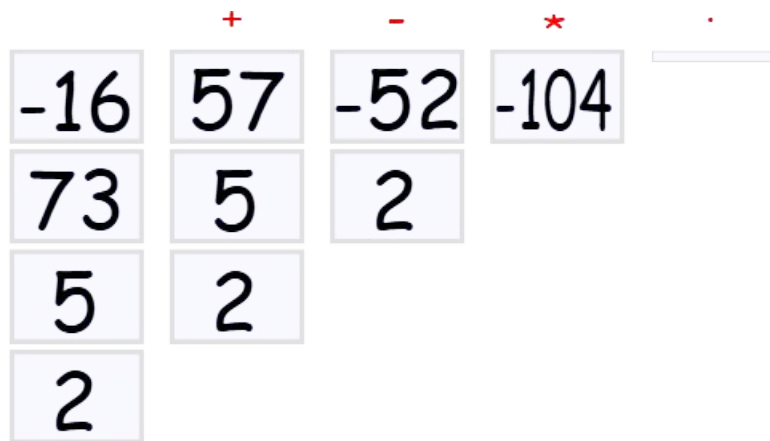
Zelle	Inhalt	Kommentar
0	-16	(TOS) Oben rechts
1	73	(NOS) Als nächstes im Stapel
2	5	
3	2	

In Forth-Datenstrukturen wie Stapeln, Arrays und Tabellen verwenden wir normalerweise eine auf Null basierende relative Nummerierung. Beachten Sie, dass bei der Eingabe einer Zahlenfolge auf diese Weise die Zahl ganz rechts *zum TOS wird* und die Zahl ganz links unten im Stapel liegt.

Angenommen, wir folgen der ursprünglichen Eingabezeile mit der Zeile

```
+ - * .
```

Die Operationen würden aufeinanderfolgende Stapeloperationen erzeugen:



Nach den beiden Zeilen zeigt die Konsole Folgendes an:

```
decimal 2 5 73 -16    \ zeigt an: 2 5 73 -16 ok
+ - * .               \ zeigt an: -104 ok
```

Beachten Sie, dass ESP32forth die Stapелеlemente bei der Interpretation jeder Zeile bequem anzeigt und dass der Wert -16 als 32-Bit-Ganzzahl ohne Vorzeichen angezeigt wird. Darüber hinaus ist das Wort `.` verbraucht den Datenwert -104 und lässt den Stapel leer. Wenn wir ausführen. Auf dem nun leeren Stack bricht der externe Interpreter mit einem Stack-Pointer-Fehler STACK UNDERFLOW ERROR ab.

Die Programmiernotation, bei der die Operanden zuerst erscheinen, gefolgt von den Operatoren, wird Reverse Polish Notation (RPN) genannt.

Umgang mit dem Parameterstapel

Da es sich um ein stapelbasiertes System handelt, muss ESP32forth Möglichkeiten bieten, Zahlen auf den Stapel zu legen, sie zu entfernen und ihre Reihenfolge neu zu ordnen. Wir haben bereits gesehen, dass wir Zahlen einfach durch Eintippen auf den Stapel legen können. Wir können auch Zahlen in die Definition eines FORTH-Wortes integrieren.

Durch das Wort **drop** wird eine Zahl von der obersten Stelle des Stapels entfernt, sodass die nächste Zahl oben liegt. **swap** Worttausch werden die ersten beiden Zahlen ausgetauscht. **dup** kopiert die Zahl oben und verschiebt alle anderen Zahlen nach unten. **rot** dreht die ersten 3 Zahlen. Diese Aktionen werden im Folgenden vorgestellt.



Der Return Stack und seine Verwendung

Beim Kompilieren eines neuen Wortes stellt ESP32forth Verknüpfungen zwischen dem aufrufenden Wort und zuvor definierten Wörtern her, die bei der Ausführung des neuen Wortes aufgerufen werden sollen. Dieser Verknüpfungsmechanismus verwendet zur Laufzeit den Rstack. Die Adresse des nächsten aufzurufenden Wortes wird auf dem hinteren Stapel abgelegt, sodass das System nach Abschluss der Ausführung des aktuellen Worts weiß, wohin es zum nächsten Wort wechseln muss. Da Wörter verschachtelt werden können, muss ein Stapel dieser Rücksprungadressen vorhanden sein.

Der Benutzer dient nicht nur als Reservoir für Rücksprungadressen, sondern kann auch den Rückgabestapel speichern und von dort abrufen. Dabei muss jedoch sorgfältig vorgegangen werden, da der Rückgabestapel für die Programmausführung unerlässlich ist. Wenn Sie den Rückgabeakku zur vorübergehenden Speicherung verwenden, müssen Sie ihn in seinen ursprünglichen Zustand zurückversetzen, da es sonst wahrscheinlich zu einem Absturz des ESP32forth-Systems kommt. Trotz der Gefahr gibt es Zeiten, in denen die Verwendung von Backstack als temporärer Speicher Ihren Code weniger komplex machen kann.

Zum Speichern auf dem Stapel verwenden Sie **>r** , um den oberen Rand des Parameterstapels an den oberen Rand des Rückgabestapels zu verschieben. Um einen Wert abzurufen, verschiebt **r>** den obersten Wert vom Stapel zurück an die Spitze des Parameterstapels. Um einfach einen Wert oben vom Stapel zu entfernen, gibt es das Wort **rdrop** . Das Wort **r@** kopiert den oberen Teil des Stapels zurück in den Parameterstapel.

Speichernutzung

@ (fetch) aus dem Speicher auf den Stapel geholt und mit dem Wort **!** von oben im Speicher abgelegt. (blind). **@** erwartet eine Adresse auf dem Stapel und ersetzt die Adresse durch ihren Inhalt. **!** erwartet eine Nummer und eine Adresse, um es zu speichern. Die Nummer wird an dem Speicherort abgelegt, auf den sich die Adresse bezieht, wobei dabei beide Parameter verbraucht werden.

Vorzeichenlose Zahlen, die 8-Bit-Werte (Byte) darstellen, können in zeichengroße Zeichen eingefügt werden. Speicherzellen mit **c@** und **c!** .

```
create testVar
  cell allot
  $f7 testVar c!
testVar c@ . \ zeigt 247 an
```

Variablen

Eine Variable ist ein benannter Speicherort im Speicher, der eine Zahl, beispielsweise das Zwischenergebnis einer Berechnung, außerhalb des Stapels speichern kann. Zum Beispiel :

```
Variable x
```

erstellt einen Speicherort mit dem Namen **x** , der ausgeführt wird und die Adresse seines Speicherorts oben im Stapel belässt:

```
x . \ zeigt die Adresse an
```

Wir können die Daten dann an dieser Adresse abholen oder speichern:

```
Variable x
3 x !
x @ . \ zeigt an: 3
```

Konstanten

Eine Konstante ist eine Zahl, die Sie während der Ausführung eines Programms nicht ändern möchten. Das Ergebnis der Ausführung des mit einer Konstanten verbundenen Wortes ist der Wert der auf dem Stapel verbleibenden Daten.

```
\ definiert VSPI-Pins
19 constant VSPI_MISO
23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS

\ legt die SPI-Portfrequenz fest
4000000 Konstante SPI_FREQ

\ SPI-Vokabular auswählen
only FORTH SPI also

\ initialisiert den SPI-Port
: init.VSPI ( -- )
  VSPI_CS OUTPUT pinMode
  VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
  SPI_FREQ SPI.setFrequency
;
```

Pseudokonstante Werte

Ein mit value definierter Wert ist ein Hybridtyp aus Variable und Konstante. Wir legen einen Wert fest, initialisieren ihn und er wird wie eine Konstante aufgerufen. Wir können einen Wert auch ändern, so wie wir eine Variable ändern können.

```
decimal
13 value thirteen
thirteen . \ Anzeige: 13
47 to thirteen
thirteen . \ Anzeige: 47
```

Das Wort **to** funktioniert auch in Wortdefinitionen und ersetzt den darauf folgenden Wert durch den Wert, der sich gerade ganz oben im Stapel befindet. Sie müssen darauf achten, dass auf **to** ein durch value definierter Wert folgt und nicht etwas anderes.

Grundlegende Tools für die Speicherzuweisung

Die Wörter **create** und **allot** sind die grundlegenden Werkzeuge zum Reservieren von Speicherplatz und zum Anbringen einer Bezeichnung. Die folgende Transkription zeigt beispielsweise einen neuen Eintrag **graphic-array** im Wörterbuch :

```
create graphic-array ( --- addr )
  %00000000 c,
  %00000010 c,
  %00000100 c,
  %00001000 c,
  %00010000 c,
  %00100000 c,
  %01000000 c,
  %10000000 c,
```

Bei der Ausführung überträgt das Wort **graphic-array** die Adresse des ersten Eintrags.

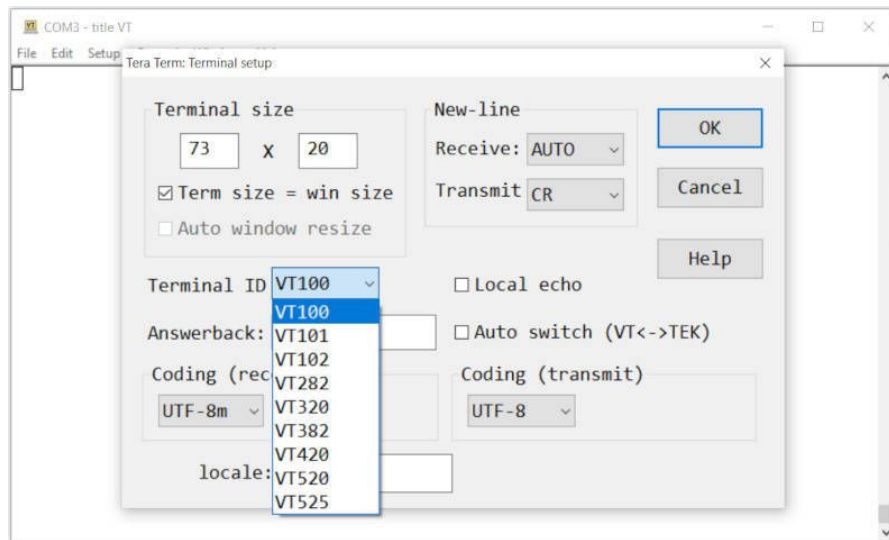
Wir können jetzt mit den zuvor erläuterten Abruf- und Speicherwörtern auf den dem **graphic-array** zugewiesenen Speicher zugreifen. Um die Adresse des dritten Bytes zu berechnen, das dem **graphic-array**, können wir **graphic-array 2 +** schreiben , wobei wir bedenken, dass die Indizes bei 0 beginnen.

```
30 graphic-array 2 + c!
graphic-array 2 + c@ .      \ affiche 30
```

Textfarben und Anzeigeposition auf dem Terminal

ANSI-Kodierung von Terminals

Wenn Sie Terminalsoftware zur Kommunikation mit ESP32forth verwenden, besteht eine gute Chance, dass dieses Terminal ein VT-Terminal oder ein gleichwertiges Terminal emuliert. Hier ist TeraTerm so konfiguriert, dass es ein VT100-Terminal emuliert:



Diese Terminals verfügen über zwei interessante Funktionen :

- Färben Sie den Seitenhintergrund und den anzuzeigenden Text
- Positionieren Sie den Anzeigecursor

Beide Funktionen werden durch ESC-Sequenzen (Escape-Sequenzen) gesteuert. So sind die Wörter **bg** und **fg** in ESP32forth definiert:

```
forth definitions ansi
: fg ( n -- ) esc ." [38;5;" n. ." m" ;
: bg ( n -- ) esc ." [48;5;" n. ." m" ;
: normal   esc ." [0m" ;
: at-xy ( x y -- ) esc ." [" 1+ n. ." ;" 1+ n. ." H" ;
: page    esc ." [2J" esc ." [H" ;
```

normal Wort überschreibt die durch **bg** und **fg** definierten Farbsequenzen.

Das Wort **page** löscht den Bildschirm des Terminals und positioniert den Cursor in der oberen linken Ecke des Bildschirms.

Textfärbung

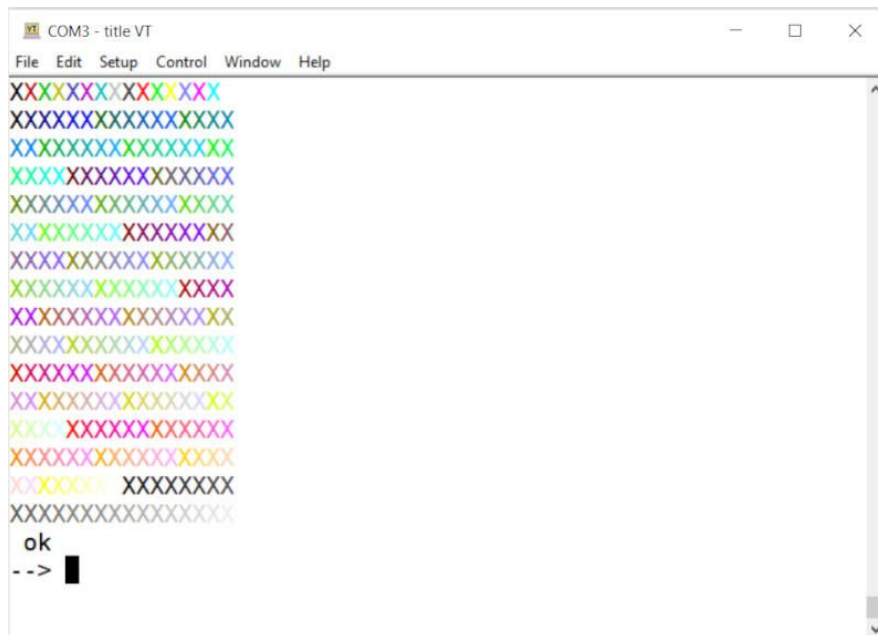
Sehen wir uns zunächst an, wie man den Text einfärbt :

```

: testFG ( -- )
  page
  16 0 do
    16 0 do
      j 16 * i + fg
      ." X"
    loop
  cr
  loop
normal
;

```

Ausführen von **testFG** wird Folgendes angezeigt:



Um die Hintergrundfarben zu testen, gehen wir wie folgt vor:

```

: testBG ( -- )
  page
  16 0 do
    16 0 do
      j 16 * i + bg
      space space
    loop
  cr
  loop
normal
;

```

Ausführen von **testBG** wird Folgendes angezeigt :



Anzeigeposition

Das Terminal ist die einfachste Lösung zur Kommunikation mit ESP32forth. Mit ANSI-Escape-Sequenzen lässt sich die Darstellung von Daten leicht verbessern.

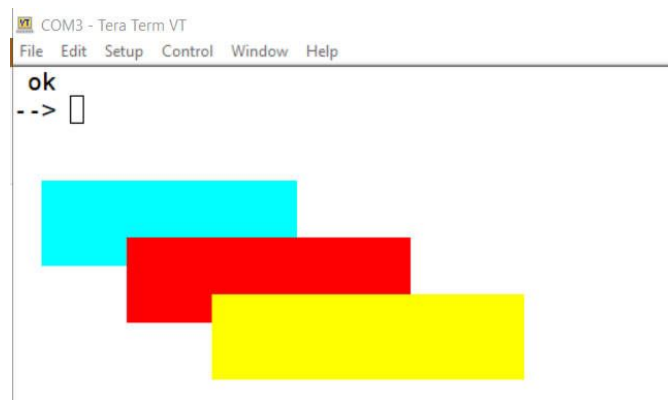
```

09 constant red
11 constant yellow
14 constant cyan
15 constant whyte
: box { x0 y0 xn yn color -- }
  color bg
  yn y0 - 1+ \ determine height
  0 do
    x0 y0 i + at-xy
    xn x0 - spaces
  loop
  normal
;

: 3boxes ( -- )
  page
  2 4 20 6 cyan box
  8 6 28 8 red box
  14 8 36 10 yellow box
  0 0 at-xy
;

```

Das Ausführen von **3boxes** zeigt Folgendes :



Sie sind nun in der Lage, einfache und effektive Schnittstellen zu erstellen, die die Interaktion mit den von ESP32forth kompilierten FORTH-Definitionen ermöglichen.

Lokale Variablen mit ESP32Forth

Einführung

Die FORTH-Sprache verarbeitet Daten hauptsächlich über den Datenstapel. Dieser sehr einfache Mechanismus bietet eine unübertroffene Leistung. Umgekehrt kann es schnell komplex werden, den Datenfluss zu verfolgen. Lokale Variablen bieten eine interessante Alternative.

Der Fake-Stack-Kommentar

(und) umrahmten Stapelkommentare aufgefallen sein . Beispiel:

```
\ addiert zwei vorzeichenlose Werte, hinterlässt Summe
\ und Übertrag auf dem Stapel
: um+ (u1 u2 -- Summenübertrag)
  \ hier die Definition
;
```

Hier hat der Kommentar (u1 u2 -- sum Carry) absolut keine Auswirkung auf den Rest des FORTH-Codes. Das ist reiner Kommentar.

Bei der Vorbereitung einer komplexen Definition besteht die Lösung darin, lokale Variablen zu verwenden, die durch { und } eingerahmt sind. Beispiel :

```
: 2OVER { a b c d }
  a b c d a b
;
```

Wir definieren vier lokale Variablen **abc** und **d** .

Die Wörter { und } ähneln den Wörtern (und) , haben aber überhaupt nicht die gleiche Wirkung. Codes zwischen { und } sind lokale Variablen. Die einzige Einschränkung: Verwenden Sie keine Variablennamen, die FORTH-Wörter aus dem FORTH-Wörterbuch sein könnten. Wir hätten unser Beispiel auch so schreiben können:

```
: 2OVER { varA varB varC varD }
  varA varB varC varD varA varB
;
```

Jede Variable nimmt den Wert des Datenstapels in der Reihenfolge ihrer Hinterlegung auf dem Datenstapel an. hier geht 1 in **varA** , 2 in **varB** usw.:

```
--> 1 2 3 4
ok
1 2 3 4 --> 2over
ok
1 2 3 4 1 2 -->
```


Unser Fake-Stack-Kommentar kann wie folgt vervollständigt werden:

```
: 2OVER { varA varB varC varD -- varA varB varC varD varA varB }
.....
```

Die folgenden Zeichen `--` haben keine Wirkung. Der einzige Sinn besteht darin, unseren Fake-Kommentar wie einen echten Stack-Kommentar aussehen zu lassen.

Aktion auf lokale Variablen

Lokale Variablen verhalten sich genau wie durch Werte definierte Pseudovariablen.

Beispiel:

```
: 3x+1 { var -- sum }
  var 3 * 1 +
;
```

Hat den gleichen Effekt wie dieser:

```
0 value var
: 3x+1 ( var -- sum )
  to var
  var 3 * 1 +
;
```

In diesem Beispiel wird `var` explizit durch den Wert definiert.

Wir weisen einer lokalen Variablen mit dem Wort `to` oder `+to` einen Wert zu, um den Inhalt einer lokalen Variablen zu erhöhen. In diesem Beispiel fügen wir im Code unseres Wortes eine auf Null initialisierte lokale Variable `result` hinzu:

```
: a+bEXP2 { varA varB -- (a+b)EXP2 }
  0 { result }
  varA varA *      to result
  varB varB *      +to result
  varA varB * 2 * +to result
  result
;
```

Ist es nicht besser lesbar?

```
: a+bEXP2 ( varA varB -- result )
  2dup
  * 2 * >r
  dup *
  swap dup * +
  r> +
;
```

Hier ist ein letztes Beispiel, die Definition des Wortes `um+`, das zwei vorzeichenlose Ganzzahlen addiert und die Summe und den Überlaufwert dieser Summe auf dem Datenstapel belässt:

```

\ Addiert zwei ganze Zahlen ohne Vorzeichen,
\ hinterlässt Summe und Übertrag auf dem Stapel
: um+ { u1 u2 -- sum carry }
  0 { sum }
  cell for
    aft
      u1 $100 /mod to u1
      u2 $100 /mod to u2
      +
      cell 1- i - 8 * lshift +to sum
    then
  next
  sum
  u1 u2 + abs
;

```

Hier ist ein komplexeres Beispiel für das Umschreiben von **DUMP** mithilfe lokaler Variablen:

```

\ lokale Variablen in DUMP:
\ START_ADDR      \ erste Adresse für Dump
\ END_ADDR        \ letzte Adresse für Dump
\ OSTART_ADDR     \ erste Adresse für Schleife im Dump
\ LINES           \ Anzahl der Zeilen für die Dump-Schleife
\ myBASE          \ aktuelle numerische Basis
internals
: dump ( start len -- )
  cr cr ." --addr--- "
  ." 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----"
  2dup + { END_ADDR }          \ letzte Adresse für den Dump speichern
  swap { START_ADDR }          \ START-Adresse zum Dump speichern
  START_ADDR 16 / 16 * { OSTART_ADDR } \ calc. Adresse für Schleifenstart
  16 / 1+ { LINES }
  base @ { myBASE }            \ aktuelle Basis speichern
  hex
    \ äußere Schleife
  LINES 0 do
    OSTART_ADDR i 16 * +        \ calc start address for current line
    cr <# # # # # [char] - hold # # # # #> type
    space space                \ and display address
    \ first inner loop, display bytes
    16 0 do
      \ calculate real address
      OSTART_ADDR j 16 * i + +
      ca@ <# # # #> type space \ display byte in format: NN
    loop
    space
    \ second inner loop, display chars
    16 0 do
      \ calculate real address
      OSTART_ADDR j 16 * i + +
      \ display char if code in interval 32-127
      ca@      dup 32 < over 127 > or
      if      drop [char] . emit
      else    emit
      then
    loop
  loop
loop

```

```

myBASE base !           \ restore current base
cr cr
;
forth

```

Die Verwendung lokaler Variablen vereinfacht die Datenmanipulation auf Stacks erheblich. Der Code ist besser lesbar. Beachten Sie, dass es nicht notwendig ist, diese lokalen Variablen vorab zu deklarieren. Es reicht aus, sie bei der Verwendung anzugeben, zum Beispiel: **base @ { myBASE } .**

WARNUNG: Wenn Sie lokale Variablen in einer Definition verwenden, verwenden Sie nicht mehr die Wörter **>r** und **r>**, da sonst die Gefahr besteht, dass die Verwaltung lokaler Variablen unterbrochen wird. Schauen Sie sich einfach die Dekompilierung dieser Version von **DUMP** an, um den Grund für diese Warnung zu verstehen:

dump cr cr s" --addr-- " type

```

s" 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----" type
2dup + >R SWAP >R -4 local@ 16 / 16 * >R 16 / 1+ >R base @ >R
hex -8 local@ 0 (do) -20 local@ R@ 16 * + cr
<# # # # 45 hold # # # # > type space space
16 0 (do) -28 local@ j 16 * R@ + + CA@ <# # # # > type space 1 (+loop)
0BRANCH rdrop rdrop space 16 0 (do) -28 local@ j 16 * R@ + + CA@ DUP 32 < OVER 127 > OR
0BRANCH DROP 46 emit BRANCH emit 1 (+loop) 0BRANCH rdrop rdrop 1 (+loop)
0BRANCH rdrop rdrop -4 local@ base ! cr cr rdrop rdrop rdrop rdrop rdrop ;

```

Datenstrukturen für ESP32forth

Präambel

ESP32forth ist eine 32-Bit-Version der FORTH-Sprache. Diejenigen, die FORTH seit seinen Anfängen praktizieren, haben mit 16-Bit-Versionen programmiert. Diese Datengröße wird durch die Größe der auf dem Datenstapel abgelegten Elemente bestimmt. Um die Größe der Elemente in Bytes herauszufinden, müssen Sie das Wort `cell` ausführen. Führen Sie dieses Wort für ESP32forth aus:

```
cell . \ zeigt 4 an
```

Der Wert 4 bedeutet, dass die Größe der auf dem Datenstapel platzierten Elemente 4 Bytes oder $4 \times 8 \text{ Bits} = 32 \text{ Bits}$ beträgt.

Bei einer 16-Bit-Forth-Version stapelt Zelle den Wert 2. Wenn Sie eine 64-Bit-Version verwenden, stapelt Zelle ebenfalls den Wert 8.

Tabellen in FORTH

Beginnen wir mit relativ einfachen Strukturen: Tabellen. Wir werden nur ein- oder zweidimensionale Arrays diskutieren.

Eindimensionales 32-Bit-Datenarray

Dies ist die einfachste Art von Tabelle. Um eine Tabelle dieses Typs zu erstellen, verwenden wir das Wort **create**, gefolgt vom Namen der zu erstellenden Tabelle:

```
create temperatures
  34 , 37 , 42 , 36 , 25 , 12 ,
```

In dieser Tabelle speichern wir 6 Werte: 34, 37...12. Um einen Wert abzurufen, verwenden Sie einfach das Wort `@`, indem Sie die nach `temperatures` gestapelte Adresse mit dem gewünschten Offset erhöhen:

```
temperatures \ empile addr
  0 cell *   \ calcule décalage 0
  +          \ ajout décalage à addr
  @ .       \ affiche 34

temperatures \ empile addr
  1 cell *   \ calcule décalage 1
  +          \ ajout décalage à addr
  @ .       \ affiche 37
```

Wir können den Zugangscode auf den gewünschten Wert faktorisieren, indem wir ein Wort definieren, das diese Adresse berechnet:

```
: temp@ ( index -- value )
  cell * temperatures + @
;
0 temp@ . \ affiche 34
2 temp@ . \ affiche 42
```

Sie werden feststellen, dass für n in dieser Tabelle gespeicherte Werte, hier 6 Werte, der Zugriffsindex immer im Intervall [0..n-1] liegen muss.

Tabellendefinitionswörter

So erstellen Sie eine Wortdefinition für eindimensionale Ganzzahl-Arrays:

```
: array ( comp: -- | exec: index -- addr )
  create
  does>
    swap cell * +
;
array myTemps
  21 , 32 , 45 , 44 , 28 , 12 ,
0 myTemps @ . \ zeigt 21
5 myTemps @ . \ zeigt 12
```

array -Variante zu erstellen, um unsere Daten kompakter zu verwalten :

```
: arrayC (comp: -- | exec: index -- addr)
erstellen
tut>
+
;
arrayC myCTime
21c, 32c, 45c, 44c, 28c, 12c,
0 myCTemps c@. \anzeige 21
5 myCTemps c@. \anzeige 12
```

Bei dieser Variante werden die gleichen Werte auf viermal weniger Speicherplatz gespeichert.

Lesen und schreiben Sie in eine Tabelle

Es ist durchaus möglich, ein leeres Array mit n Elementen zu erstellen und Werte in dieses Array zu schreiben und zu lesen:

```
: arrayC ( comp: -- | exec: index -- addr )
  create
  does>
    +
;
arrayC myCTemps
  21 c, 32 c, 45 c, 44 c, 28 c, 12 c,
0 myCTemps c@ . \ display 21
5 myCTemps c@ . \ display 12
```

In unserem Beispiel enthält das Array 6 Elemente. Mit ESP32forth steht genügend Speicherplatz zur Verfügung, um deutlich größere Arrays, beispielsweise mit 1.000 oder 10.000 Elementen, zu verarbeiten. Es ist einfach, mehrdimensionale Tabellen zu erstellen. Beispiel für ein zweidimensionales Array:

```
63 constant SCR_WIDTH
16 constant SCR_HEIGHT
create mySCREEN
  SCR_WIDTH SCR_HEIGHT * allot          \ réserve 63 * 16 octets
  mySCREEN SCR_WIDTH SCR_HEIGHT * bl fill \ remplis avec 'space'
```

Hier definieren wir eine zweidimensionale Tabelle namens **mySCREEN**, die einen virtuellen Bildschirm mit 16 Zeilen und 63 Spalten darstellt.

Reservieren Sie einfach einen Speicherplatz, der das Produkt der Abmessungen X und Y der zu verwendenden Tabelle ist. Sehen wir uns nun an, wie man dieses zweidimensionale Array verwaltet:

```
: xySCRaddr { x y -- addr }
  SCR_WIDTH y *
  x + mySCREEN +
;
: SCR@ ( x y -- c )
  xySCRaddr c@
;
: SCR! ( c x y -- )
  xySCRaddr c!
;
char X 15 5 SCR!    \ speichert char X am Hals 15 Zeile 5
15 5 SCR@ emit      \ wird angezeigt
```

Praktisches Beispiel für die Verwaltung eines virtuellen Bildschirms

Bevor wir mit unserem Beispiel zur Verwaltung eines virtuellen Bildschirms fortfahren, sehen wir uns an, wie man den Zeichensatz des TERA TERM-Terminals ändert und anzeigt.

TERA TERM starten:

- Klicken Sie in der Menüleiste auf **Setup**
- Wählen Sie **Font** und **Font...**
- Konfigurieren Sie die Schriftart unten:


```
$db dup 5 2 SCR!      6 2 SCR!
$b2 dup 7 3 SCR!      8 3 SCR!
$b1 dup 9 4 SCR!     10 4 SCR!
```

Sehen wir uns nun an, wie wir den Inhalt unseres virtuellen Bildschirms anzeigen. Wenn wir jede Zeile des virtuellen Bildschirms als alphanumerische Zeichenfolge betrachten, müssen wir nur dieses Wort definieren, um eine der Zeilen unseres virtuellen Bildschirms anzuzeigen:

```
: dispLine { numLine -- }
SCR_WIDTH numLine *
mySCREEN + SCR_WIDTH-Typ
;
```

Unterwegs erstellen wir eine Definition, die es ermöglicht, dasselbe Zeichen n-mal anzuzeigen:

```
: nEmit ( c n -- )
  for
    aft dup emit then
  next
  drop
;
```

Und jetzt definieren wir das Wort, mit dem wir den Inhalt unseres virtuellen Bildschirms anzeigen können. Um den Inhalt dieses virtuellen Bildschirms klar zu erkennen, rahmen wir ihn mit Sonderzeichen ein:

```
: dispScreen
  0 0 at-xy
  \ affiche bord superieur
  $da emit   $c4 SCR_WIDTH nEmit   $bf emit   cr
  \ affiche contenu ecran virtuel
  SCR_HEIGHT 0 do
    $b3 emit   i dispLine   $b3 emit   cr
  loop
  \ affiche bord inferieur
  $c0 emit   $c4 SCR_WIDTH nEmit   $d9 emit   cr
;
```

Wenn Sie unser **dispScreen-** Wort ausführen, wird Folgendes angezeigt :



In unserem virtuellen Bildschirmbeispiel zeigen wir, dass die Verwaltung eines zweidimensionalen Arrays eine konkrete Anwendung hat. Unser virtueller Bildschirm ist zum Schreiben und Lesen zugänglich. Hier zeigen wir unseren virtuellen Bildschirm im Terminalfenster an. Diese Anzeige ist alles andere als effizient. Auf einem echten OLED-Bildschirm kann es aber deutlich schneller gehen.

Management komplexer Strukturen

ESP32forth verfügt über das Strukturvokabular. Der Inhalt dieses Vokabulars ermöglicht die Definition komplexer Datenstrukturen.

Hier ist eine triviale Beispielstruktur :

```
structures
struct YMDHMS
  ptr field >year
  ptr field >month
  ptr field >day
  ptr field >hour
  ptr field >min
  ptr field >sec
```

Hier definieren wir die YMDHMS-Struktur. Diese Struktur verwaltet die Zeiger **>year** **>month** **>day** **>hour** **>min** und **>sec** .

des **YMDHMS** -Wortes besteht darin, die Zeiger in der komplexen Struktur zu initialisieren und zu gruppieren. So werden diese Zeiger verwendet :

```
create DateTime
  YMDHMS allot

2022 DateTime >year  !
  03 DateTime >month !
  21 DateTime >day   !
  22 DateTime >hour  !
  36 DateTime >min   !
```

```

15 DateTime >sec    !

: .date ( date -- )
  >r
  ." YEAR: " r@ >year    @ . cr
  ." MONTH: " r@ >month  @ . cr
  ." DAY: " r@ >day      @ . cr
  ." HH: " r@ >hour      @ . cr
  ." MM: " r@ >min       @ . cr
  ." SS: " r@ >sec       @ . cr
  r> drop
;

DateTime .date

```

Wir haben das Wort **DateTime** definiert, das eine einfache Tabelle aus 6 aufeinanderfolgenden 32-Bit-Zellen ist. Der Zugriff auf jede Zelle erfolgt über den entsprechenden Zeiger. Wir können den zugewiesenen Speicherplatz unserer **YMDHMS**-Struktur neu definieren, indem wir das Wort **i8** verwenden, um auf Bytes zu verweisen:

```

structures
struct cYMDHMS
  ptr field >year
  i8  field >month
  i8  field >day
  i8  field >hour
  i8  field >min
  i8  field >sec

create cDateTime
  cYMDHMS allot

2022 cDateTime >year    !
03 cDateTime >month    c!
21 cDateTime >day      c!
22 cDateTime >hour     c!
36 cDateTime >min      c!
15 cDateTime >sec      c!

: .cDate ( date -- )
  >r
  ." YEAR: " r@ >year    @ . cr
  ." MONTH: " r@ >month  c@ . cr
  ." DAY: " r@ >day      c@ . cr
  ." HH: " r@ >hour      c@ . cr
  ." MM: " r@ >min       c@ . cr
  ." SS: " r@ >sec       c@ . cr
  r> drop
;

cDateTime .cDate    \ zeigt:
\ YEAR: 2022
\ MONTH: 3
\ DAY: 21
\ HH: 22
\ MM: 36
\ SS: 15

```

In dieser **cYMDHMS**-Struktur haben wir das Jahr im 32-Bit-Format beibehalten und alle anderen Werte auf 8-Bit-Ganzzahlen reduziert. Wir sehen im **.cDate**-Code, dass die

Verwendung von Zeigern einen einfachen Zugriff auf jedes Element unserer komplexen Struktur ermöglicht....

Definition von Sprites

Wir haben zuvor einen virtuellen Bildschirm als zweidimensionales Array definiert. Die Dimensionen dieses Arrays werden durch zwei Konstanten definiert. Erinnerung an die Definition dieses virtuellen Bildschirms:

```
63 constant SCR_WIDTH
16 constant SCR_HEIGHT
create mySCREEN
    SCR_WIDTH SCR_HEIGHT * allot
mySCREEN SCR_WIDTH SCR_HEIGHT * b1 fill
```

Der Nachteil dieser Programmiermethode besteht darin, dass die Dimensionen in Konstanten, also außerhalb der Tabelle, definiert werden. Interessanter wäre es, die Abmessungen der Tabelle in die Tabelle einzubetten. Dazu definieren wir eine an diesen Fall angepasste Struktur :

```
structures
struct cARRAY
    i8 field >width
    i8 field >height
    i8 field >content

create myVscreen \ definit un ecran 8x32 octets
    32 c, \ compile width
    08 c, \ compile height
myVscreen >width c@
myVscreen >height c@ * allot
```

Um ein Software-Sprite zu definieren, geben wir ganz einfach diese Definition weiter:

```
: sprite: ( width height -- )
    create
        swap c, c, \ compile width et height
    does>
;
2 1 sprite: blackChars
    $db c, $db c,
2 1 sprite: greyChars
    $b2 c, $b2 c,
blackChars >content 2 type \ affiche contenu du sprite blackChars
```

Voici comment définir un sprite 5 x 7 octets:

```
5 7 sprite: char3
    $20 c, $db c, $db c, $db c, $20 c,
    $db c, $20 c, $20 c, $20 c, $db c,
    $20 c, $20 c, $20 c, $20 c, $db c,
    $20 c, $db c, $db c, $db c, $20 c,
    $20 c, $20 c, $20 c, $20 c, $db c,
    $db c, $20 c, $20 c, $20 c, $db c,
```

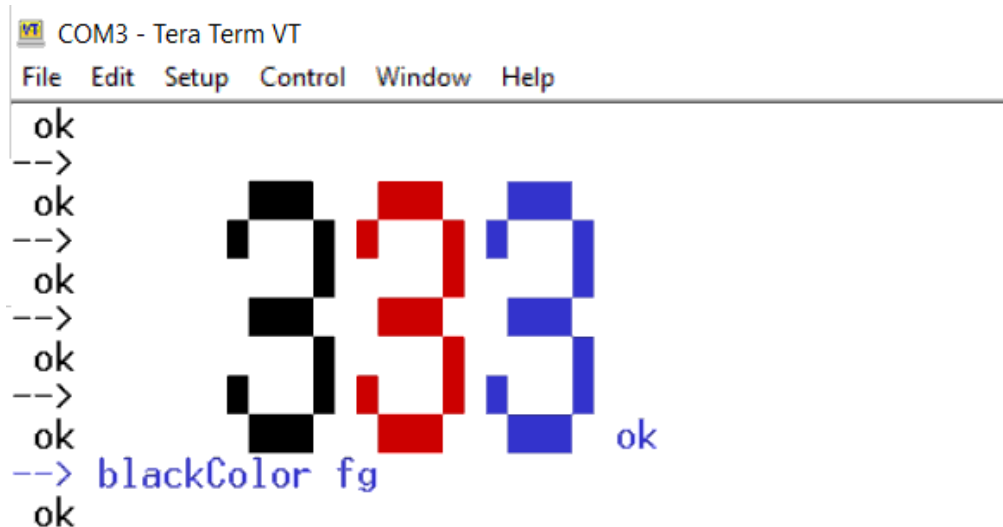
```
$20 c, $db c, $db c, $db c, $20 c,
```

Um das Sprite von einer xy-Position im Terminalfenster aus anzuzeigen, reicht eine einfache Schleife:

```
: .sprite { xpos ypos sprAddr -- }
  sprAddr >height c@ 0 do
    xpos ypos at-xy
    sprAddr >width c@ i * \ calculate offset in sprite datas
    sprAddr >content + \ calculate real address for line n in
sprite datas
    sprAddr >width c@ type \ display line
    1 +to ypos \ increment y position
  loop
;

0 constant blackColor
1 constant redColor
4 constant blueColor
10 02 char3 .sprite
redColor fg
16 02 char3 .sprite
blueColor fg
22 02 char3 .sprite
blackColor fg
cr cr
```

Ergebnis der Anzeige unseres Sprites:



Ich hoffe, der Inhalt dieses Kapitels hat Ihnen einige interessante Ideen gegeben, die Sie gerne teilen möchten ...

Reale Zahlen mit ESP32forth

Wenn wir die Operation **1 3 /** in der FORTH-Sprache testen, ist das Ergebnis 0.

Es ist nicht überraschend. Grundsätzlich verwendet ESP32forth über den Datenstack nur 32-Bit-Ganzzahlen. Ganzzahlen bieten bestimmte Vorteile:

- Geschwindigkeit der Verarbeitung;
- Ergebnis von Berechnungen ohne Driftgefahr bei Iterationen;
- für fast alle Situationen geeignet.

Auch bei trigonometrischen Berechnungen können wir eine Tabelle mit ganzen Zahlen verwenden. Erstellen Sie einfach eine Tabelle mit 90 Werten, wobei jeder Wert dem Sinus eines Winkels multipliziert mit 1000 entspricht.

Aber auch ganze Zahlen haben Grenzen:

- unmögliche Ergebnisse für einfache Divisionsberechnungen, wie unser 1/3-Beispiel;
- erfordert komplexe Manipulationen, um physikalische Formeln anzuwenden.

Seit Version 7.0.6.5 enthält ESP32forth Operatoren, die sich mit reellen Zahlen befassen.

Reelle Zahlen werden auch Gleitkommazahlen genannt.

Die echten mit ESP32forth

Um reelle Zahlen zu unterscheiden, müssen sie mit dem Buchstaben „e“ enden:

```
3          \ push 3 on the normal stack
3e         \ push 3 on the real stack
5.21e f.   \ display 5.210000
```

Es ist das Wort **F.** Dadurch können Sie eine reelle Zahl anzeigen, die sich oben auf dem reellen Stapel befindet.

Echte Zahlengenauigkeit mit ESP32forth

Mit dem Wort **set-precision** können Sie die Anzahl der Dezimalstellen angeben, die nach dem Dezimalpunkt angezeigt werden sollen. Sehen wir uns das mit der Konstante **pi** an :

```
pi f.      \ display 3.141592
4 set-precision
```

```
pi f.      \ display 3.1415
```

Die Grenzgenauigkeit für die Verarbeitung reeller Zahlen mit ESP32forth beträgt sechs Dezimalstellen :

```
12 set-precision  
1.987654321e f.      \ display 1.987654668777
```

Wenn wir die Anzeigegenauigkeit reeller Zahlen unter 6 reduzieren, werden die Berechnungen immer noch mit einer Genauigkeit von 6 Nachkommastellen durchgeführt.

Reale Konstanten und Variablen

Eine echte Konstante wird mit dem Wort **fconstant** definiert :

```
0.693147e fconstant ln2    \ natural logarithm of 2
```

fvariable definiert :

```
fvariable intensity  
170e 12e F/ intensity SF!    \ I=P/U    ---    P=170w    U=12V  
intensity SF@ f.              \ display 14.166669
```

ACHTUNG: Alle reellen Zahlen durchlaufen den **Stapel reeller Zahlen** . Bei einer realen Variablen durchläuft nur die Adresse den Datenstapel, die auf den realen Wert zeigt.

Das Wort **SF!** speichert einen reellen Wert an der Adresse oder Variablen, auf die seine Speicheradresse zeigt. Durch die Ausführung einer echten Variablen wird die Speicheradresse auf dem klassischen Datenstapel platziert.

Das Wort **SF@** stapelt den realen Wert, auf den seine Speicheradresse zeigt.

Arithmetische Operatoren für reelle Zahlen

ESP32Forth verfügt über vier arithmetische Operatoren **F+ F- F* F/** :

```
1.23e 4.56e F+ f.    \ display 5.790000    1.23+4.56  
1.23e 4.56e F- f.    \ display -3.330000    1.23-4.56  
1.23e 4.56e F* f.    \ display 5.608800    1.23*4.56  
1.23e 4.56e F/ f.    \ display 0.269736    1.23/4.56
```

ESP32forth hat auch diese Worte:

- **1/F** berechnet den Kehrwert einer reellen Zahl;
- **fsqrt** berechnet die Quadratwurzel einer reellen Zahl.

```
5. 1/F f. \display 0,200000 1/5  
5. fsqrt f. \ display 2.236068 sqrt(5)
```

Mathematische Operatoren für reelle Zahlen

ESP32forth verfügt über mehrere mathematische Operatoren:

- **F**** erhöht einen echten r_val auf die Potenz r_exp
- **FATAN2** berechnet den Winkel im Bogenmaß aus der Tangente.
- **FCOS** ($r1 - r2$) Berechnet den Kosinus eines Winkels, ausgedrückt im Bogenmaß.
- **FEXP** ($\ln-r - r$) berechnet den reellen Wert, der $e^{EXP\ r}$ entspricht
- **FLN** ($r - \ln-r$) berechnet den natürlichen Logarithmus einer reellen Zahl.
- **FSIN** ($r1 - r2$) berechnet den Sinus eines Winkels, ausgedrückt im Bogenmaß.
- **FSINCOS** ($r1 - rcos\ rsin$) berechnet den Kosinus und Sinus eines Winkels, ausgedrückt im Bogenmaß.

Einige Beispiele :

```
2e 3e f** f.    \ display 8.000000
2e 4e f** f.    \ display 16.000000
10e 1.5e f** f.  \ display 31.622776

4.605170e FEXP F.    \ display 100.000018

pi 4e f/
FSINCOS f. f.    \ display 0.707106 0.707106
pi 2e f/
FSINCOS f. f.    \ display 0.000000 1.000000
```

Logische Operatoren für reelle Zahlen

Mit ESP32forth können Sie auch Logiktests an realen Daten durchführen:

- **F0<** ($r -- fl$) testet, ob eine reelle Zahl kleiner als Null ist.
- **F0=** ($r - fl$) zeigt wahr an, wenn der reelle Wert Null ist.
- **f<** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 < r2$.
- **f<=** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 \leq r2$.
- **f<>** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 \neq r2$.
- **f=** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 = r2$.
- **f>** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 > r2$.
- **f>=** ($r1\ r2 -- fl$) fl ist wahr, wenn $r1 \geq r2$.

Ganzzahlige ↔ reelle Transformationen

ESP32forth verfügt über zwei Wörter, um ganze Zahlen in reelle Zahlen umzuwandeln und umgekehrt:

- **F>S** (r -- n) wandelt eine reelle Zahl in eine ganze Zahl um. Belassen Sie den ganzzahligen Teil im Datenstapel, wenn die reelle Zahl Dezimalteile hat.
- **S>F** (n -- r: r) wandelt eine ganze Zahl in eine reelle Zahl um und überträgt diese reelle Zahl auf den reellen Stapel.

Beispiel :

```
35 S>F
F.    \ display 35.000000

3.5e F>S .    \ display 3
```


Zahlen und Zeichenfolgen anzeigen

Änderung der Zahlenbasis

FORTH verarbeitet nicht irgendwelche Zahlen. Bei den Zahlen, die Sie beim Ausprobieren der vorherigen Beispiele verwendet haben, handelt es sich um vorzeichenbehaftete Ganzzahlen mit einfacher Genauigkeit. Der Definitionsbereich für 32-Bit-Ganzzahlen ist -2147483648 bis 2147483647. Beispiel:

```
2147483647 .      \ zeigt 2147483647 an
2147483647 1+ .   \ zeigt -2147483648 an
-1 u.            \ zeigt 4294967295 an
```

Diese Zahlen können in jeder Zahlenbasis verarbeitet werden, wobei alle Zahlenbasen zwischen 2 und 36 gültig sind:

```
255 HEX . DECIMAL \ zeigt FF an
```

Sie können eine noch größere numerische Basis wählen, aber die verfügbaren Symbole fallen dann außerhalb des alphanumerischen Satzes [0..9,A..Z] und es besteht die Gefahr, dass sie inkonsistent werden.

Die aktuelle numerische Basis wird durch eine Variable namens **BASE** gesteuert, deren Inhalt geändert werden kann. Um also auf Binär umzustellen, speichern Sie einfach den Wert **2** in **BASE**. Beispiel:

```
2 BASE !
```

und geben Sie **DECIMAL** ein, um zur dezimalen numerischen Basis zurückzukehren.

ESP32forth verfügt über zwei vordefinierte Wörter, mit denen Sie verschiedene numerische Basen auswählen können:

- **DECIMAL**, um die dezimale numerische Basis auszuwählen. Dies ist die numerische Basis, die standardmäßig beim Starten von ESP32forth verwendet wird.
- **HEX** zur Auswahl der hexadezimalen numerischen Basis.

Bei Auswahl einer dieser Zahlenbasen werden die Literalzahlen in dieser Basis interpretiert, angezeigt oder verarbeitet. Jede zuvor in einem anderen Zahlensystem als dem aktuellen Zahlensystem eingegebene Zahl wird automatisch in das aktuelle Zahlensystem umgewandelt. Beispiel :

```
DECIMAL      \ Basis in Dezimalzahl
255          \ Stapel 255
HEX          \ wählt die Hexadezimalbasis aus
1+           \ erhöht 255 zu 256
```

```
. \ zeigt 100 an
```

BASE speichert . Beispiel :

```
: BINARY ( ---) \ wählt die binäre Zahlenbasis aus
2 BASIS! ;
DECIMAL 255 BINARY . \ zeigt 11111111 an
```

Der Inhalt von **BASE** kann wie der Inhalt jeder anderen Variablen gestapelt werden:

```
VARIABLE RANGE_BASE \ RANGE-BASE-Variablendefinition
BASE @ RANGE_BASE ! \ Speicherung des BASE-Inhalts in RANGE-BASE
HEX FF 10 + . \ zeigt 10F an
RANGE_BASE @ BASE ! \ stellt BASE mit Inhalten von RANGE-BASE wieder
her
```

In einer Definition : kann der Inhalt von **BASE** den Rückgabestapel passieren:

```
: OPERATION ( ---)
  BASE @ >R \ speichert BASE auf dem hinteren Stapel
HEX FF 10 + . \ Operation des vorherigen Beispiels
R> BASE ! ; \ stellt den anfänglichen BASE-Wert wieder her
```

ACHTUNG : Die Wörter **>R** und **R>** können nicht im interpretierten Modus verwendet werden. Sie können diese Wörter nur in einer Definition verwenden, die zusammengestellt wird.

Definition neuer Anzeigeformate

Forth verfügt über Grundelemente, mit denen Sie die Anzeige einer Zahl an jedes beliebige Format anpassen können. Mit ESP32forth verarbeiten diese Grundelemente Ganzzahlen :

- **<#** beginnt eine Formatdefinitionssequenz;
- **#** fügt eine Ziffer in eine Formatdefinitionssequenz ein;
- **#S** entspricht einer Folge von **#** ;
- **HOLD** fügt ein Zeichen in eine Formatdefinition ein;
- **#>** vervollständigt eine Formatdefinition und hinterlässt auf dem Stapel die Adresse und Länge der Zeichenfolge, die die anzuzeigende Zahl enthält.

Diese Wörter können nur innerhalb einer Definition verwendet werden. Beispiel, um entweder eine Zahl anzuzeigen, die einen auf Euro lautenden Betrag mit dem Komma als Dezimaltrennzeichen ausdrückt:

```
: .EUROS ( n ---)
  <# # # [char] , hold #S #>
  type space ." EUR" ;
1245 .euros
```

Ausführungsbeispiele :

```
35 .EUROS          \ zeigt 0,35 EUR an
3575 .EUROS         \ zeigt 35,75 EUR an
1015 3575 + .EUROS  \ zeigt 45,90 EUR an
```

In der **.EUROS**-Definition beginnt die Anzeigeformat-Definitionssequenz mit dem Wort **<# .** Die beiden Wörter **#** platzieren die Einer- und Zehnerstellen in der Zeichenfolge. Das Wort **HOLD** fügt das Zeichen **,** (Komma) nach den beiden Ziffern auf der rechten Seite ein, das Wort **#S** vervollständigt das Anzeigeformat mit den Ziffern ungleich Null nach **,**. Das Wort **#>** schließt die Formatdefinition ab und legt die Adresse und die Länge der Zeichenfolge mit den Ziffern der anzuzeigenden Zahl auf dem Stapel ab. Das Wort **TYPE** zeigt diese Zeichenfolge an.

Zur Laufzeit befasst sich eine Anzeigeformatsequenz ausschließlich mit vorzeichenbehafteten oder vorzeichenlosen 32-Bit-Ganzzahlen. Die Verkettung der verschiedenen Elemente der Zeichenfolge erfolgt von rechts nach links, d. h. beginnend mit den niedrigstwertigen Ziffern.

Die Verarbeitung einer Zahl durch eine Anzeigeformatfolge erfolgt auf Basis der aktuellen Zahlenbasis. Die Zahlenbasis kann zwischen zwei Ziffern geändert werden.

Hier ist ein komplexeres Beispiel, das die Kompaktheit von FORTH demonstriert. Dazu muss ein Programm geschrieben werden, das eine beliebige Anzahl von Sekunden in das Format HH:MM:SS umwandelt:

```
: :00 ( ---)
  DECIMAL #          \ Zifferneinheit in Dezimalzahl einfügen
  6 BASE !           \ Basis 6 Auswahl
  #                  \ Ziffer zehn einfügen
  [char] : HOLD      \ Einfügezeichen:
  DECIMAL ;          \ Dezimalbasis zurückgeben
: HMS (n ---)        \ zeigt das Zahlen-Sekunden-Format HH:MM:SS an
<# :00 :00 #S #> TYPE SPACE ;
```

Ausführungsbeispiele :

```
59 HMS      \ zeigt 0:00:59 an
60 HMS      \ zeigt 0:01:00 an
4500 HMS     \ zeigt 1:15:00 an
```

Erläuterung : Das System zur Anzeige von Sekunden und Minuten wird Sexagesimalsystem genannt. Einheiten werden im Dezimalformat ausgedrückt, **Zehner im** Sechzersystem. Das Wort **:00** verwaltet die Umrechnung von Einheiten und Zehnern in diese beiden Basen zur Formatierung der Zahlen, die Sekunden und Minuten entsprechen. Bei Zeiten sind alle Zahlen dezimal.

Ein weiteres Beispiel, um ein Programm zu definieren, das eine Dezimalzahl mit einfacher Genauigkeit in eine Binärzahl umwandelt und sie im Format bbbb bbbb bbbb bbbb anzeigt:

```
: FOUR-DIGITS ( ---)
  # # # # 32 HOLD ;
: AFB ( d ---)          \ format 4 digits and a space
  BASE @ >R              \ Current database backup
  2 BASE !               \ Binary digital base selection
  <#
  4 0 DO                 \ Format Loop
    FOUR-DIGITS
  LOOP
  #> TYPE SPACE          \ Binary display
  R> BASE ! ;           \ Initial digital base restoration
```

Ausführungsbeispiel:

```
DECIMAL 12 AFB      \ zeigt      0000 0000 0000 0110 an
HEX 3FC5 AFB        \ zeigt      0011 1111 1100 0101 an
```

Ein weiteres Beispiel ist die Erstellung eines Telefonbuch, bei dem eine oder mehrere Telefonnummern einem Nachnamen zugeordnet sind. Wir definieren ein Wort anhand des Nachnamens :

```
: .## ( ---)
  # # [char] . HOLD ;
: .TEL ( d ---)
  CR <# .## .## .## .## # # #> TYPE CR ;
: SCHNOKELOCH ( ---)
  0618051254 .TEL ;
SCHNOKELOCH \ zeigt: 06.18.05.12.54 an
```

Dieser Telefonbuch, der aus einer Quelldatei zusammengestellt werden kann, lässt sich leicht bearbeiten, und obwohl die Namen nicht klassifiziert sind, ist die Suche äußerst schnell.

Anzeigen von Zeichen und Zeichenfolgen

Ein Zeichen wird mit dem Wort **EMIT** angezeigt :

```
65 EMIT \ zeigt A an
```

Die darstellbaren Zeichen liegen im Bereich 32..255. Es werden auch Codes zwischen 0 und 31 angezeigt, sofern bestimmte Zeichen als SteuerCodes ausgeführt werden. Hier ist eine Definition, die den gesamten Zeichensatz der ASCII-Tabelle zeigt:

```
variable #out
: #out+! ( n -- )
  #out +!          \ incrémente #out
;
: (.) ( n -- a l )
```

```

DUP ABS <# #S ROT SIGN #>
;
: .R ( n l -- )
  >R (.) R> OVER - SPACES TYPE
;
: JEU-ASCII ( ---)
  cr 0 #out !
  128 32
  DO
    I 3 .R SPACE          \ affiche code du caractère
    4 #out+!
    I EMIT 2 SPACES       \ affiche caractère
    3 #out+!
    #out @ 77 =
    IF
      CR 0 #out !
    THEN
  LOOP ;

```

Beim Ausführen von **JEU-ASCII** werden die ASCII-Codes und Zeichen angezeigt, deren Code zwischen 32 und 127 liegt. Um die entsprechende Tabelle mit den ASCII-Codes im Hexadezimalformat anzuzeigen, geben Sie **HEX JEU-ASCII** ein :

```

hex jeu-ascii
20      21 !    22 "    23 #    24 $    25 %    26 &    27 '    28 (    29 )    2A *
2B +    2C ,    2D -    2E .    2F /    30 0    31 1    32 2    33 3    34 4    35 5
36 6    37 7    38 8    39 9    3A :    3B ;    3C <    3D =    3E >    3F ?    40 @
41 A    42 B    43 C    44 D    45 E    46 F    47 G    48 H    49 I    4A J    4B K
4C L    4D M    4E N    4F O    50 P    51 Q    52 R    53 S    54 T    55 U    56 V
57 W    58 X    59 Y    5A Z    5B [    5C \    5D ]    5E ^    5F _    60 `    61 a
62 b    63 c    64 d    65 e    66 f    67 g    68 h    69 i    6A j    6B k    6C l
6D m    6E n    6F o    70 p    71 q    72 r    73 s    74 t    75 u    76 v    77 w
78 x    79 y    7A z    7B {    7C |    7D }    7E ~    7F      ok

```

Zeichenfolgen werden auf verschiedene Arten angezeigt. Die erste, die nur bei der Kompilierung verwendet werden kann, zeigt eine durch das Zeichen " (Anführungszeichen) getrennte Zeichenfolge an:

```

: TITEL ." ALLGEMEINES MENÜ" ;
TITEL      \ zeigt das ALLGEMEINE MENÜ an

```

Die Zeichenfolge ist vom Wort **."** durch mindestens ein Leerzeichen getrennt.

Eine Zeichenfolge kann auch durch das Wort **s" zusammengesetzt** und durch das Zeichen " (Anführungszeichen) begrenzt werden:

```

: LIGNE1 ( --- adr len)
  S" D..Datenprotokollierung" ;

```

Durch die Ausführung von **LINE1** werden die Adresse und Länge der in der Definition kompilierten Zeichenfolge auf dem Datenstapel platziert. Die Anzeige erfolgt durch das Wort **TYPE**:

```
LINE1 TYPE \ zeigt D..Datenprotokollierung an
```

Am Ende der Anzeige einer Zeichenfolge muss auf Wunsch der Zeilenumbruch ausgelöst werden:

```
CR TITEL TYPE CR CR LINE1 TYPE CR
\ zeigt
\ ALLGEMEINES MENÜ
\
\ D..Datenprotokollierung
```

Am Anfang oder Ende der Anzeige einer alphanumerischen Zeichenfolge können ein oder mehrere Leerzeichen hinzugefügt werden:

```
SPACE \ zeigt ein Leerzeichen an
10 SPACES \ zeigt 10 Leerzeichen an
```

String-Variablen

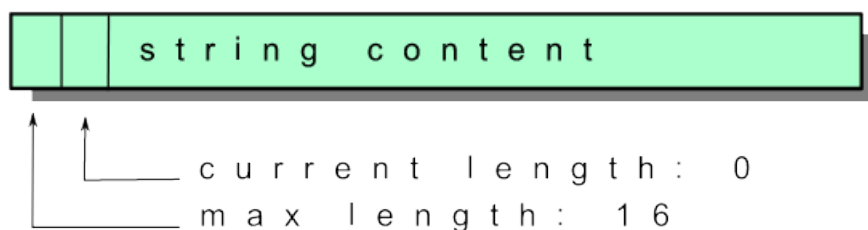
Alphanumerische Textvariablen sind in ESP32forth nativ nicht vorhanden. Hier ist der erste Versuch, die Wortfolge **STRING** zu definieren :

```
\ define a strvar
: string ( comp: n --- names_strvar | exec: --- addr len )
  create
    dup
      c, \ n is maxlength
      0 c, \ 0 is real length
    allot
  does>
    2 +
    dup 1 - c@
;
```

Eine Zeichenfolgenvariable wird wie folgt definiert:

```
16 String strState
```

So ist der für diese Textvariable reservierte Speicherplatz organisiert:



Wortcode zur Verwaltung von Textvariablen

Hier ist der vollständige Quellcode zum Verwalten von Textvariablen:

```
DEFINED? --str [if] forget --str [then]
create --str
```

```

\ compare two strings
: $= ( addr1 len1 addr2 len2 --- fl)
    str=
    ;

\ define a strvar
: string ( n --- names_strvar )
    create
        dup
        ,
        0 ,
        allot
    does>
        cell+ cell+
        dup cell - @
    ;

\ get maxlength of a string
: maxlen$ ( strvar --- strvar maxlen )
    over cell - cell - @
    ;

\ storestr into strvar
: $! ( str strvar --- )
    maxlen$
    nip rot min
    2dup swap cell - !
    cmove
    \ get maxlength of strvar
    \ keep min length
    \ store real length
    \ copy string
    ;

\ Example:
\ : s1
\     s" this is constant string" ;
\ 200 string test
\ s1 test $!

\ set length of a string to zero
: 0$! ( addr len -- )
    drop 0 swap cell - !
    ;

\ extract n chars right from string
: right$ ( str1 n --- str2 )
    0 max over min >r + r@ - r>
    ;

\ extract n chars left from string
: left$ ( str1 n --- str2 )
    0 max min
    ;

\ extract n chars from pos in string
: mid$ ( str1 pos len --- str2 )
    >r over swap - right$ r> left$
    ;

\ append char c to string
: c+$! ( c str1 -- )

```

```

over >r
+ c!
r> cell - dup @ 1+ swap !
;

\ work only with strings. Don't use with other arrays
: input$ ( addr len -- )
  over swap maxlen$ nip accept
  swap cell - !
;

```

Das Erstellen einer alphanumerischen Zeichenfolge ist sehr einfach:

```
64 String myNewString
```

Hier erstellen wir eine alphanumerische Variable **myNewString**, die bis zu 64 Zeichen enthalten kann.

Um den Inhalt einer alphanumerischen Variablen anzuzeigen, verwenden Sie einfach **den Typ**. Beispiel :

```

s" This is my first example.." myNewString $!
myNewString type \ display: This is my first example..

```

Wenn wir versuchen, eine Zeichenfolge zu speichern, die länger ist als die maximale Größe unserer alphanumerischen Variablen, wird die Zeichenfolge abgeschnitten:

```

s" This is a very long string, with more than 64 characters. It can't store
complete"
myNewString $!
myNewString type
\ disp: This is a very long string, with more than 64 characters. It can

```

Hinzufügen von Zeichen zu einer alphanumerischen Variablen

Einige Geräte, zum Beispiel der LoRa-Sender, erfordern die Verarbeitung von Befehlszeilen, die nicht alphanumerischen Zeichen enthalten. Das Wort **c+\$!** ermöglicht das Einfügen dieses Codes:

```

32 string AT_BAND
s" AT+BAND=868500000" AT_BAND $! \ set frequency at 865.5 Mhz
$0a AT_BAND c+$!
$0d AT_BAND c+$! \ add CR LF code at end of command

```

Der Speicherauszug des Inhalts unserer alphanumerischen Variablen **AT_BAND** bestätigt das Vorhandensein der beiden Steuerzeichen am Ende der Zeichenfolge:

```

--> AT_BAND dump
--addr--- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----
3FFF-8620 8C 84 FF 3F 20 00 00 00 13 00 00 00 41 54 2B 42  ...? .....AT+B
3FFF-8630 41 4E 44 3D 38 36 38 35 30 30 30 30 30 0A 0D BD  AND=868500000...
ok

```


Hier ist eine clevere Möglichkeit, eine alphanumerische Variable zu erstellen, mit der Sie einen Wagenrücklauf (**CR+LF**) **übertragen können**, der mit dem Ende von Befehlen für den LoRa-Sender kompatibel ist:

```
2 string $crlf
$0d $crlf c+$!
$0a $crlf c+$!

: crlf ( -- )      \ same action as cr, but adapted for LoRa
    $crlf type
;
```

Vokabeln mit ESP32forth

In FORTH existiert der Begriff von Prozedur und Funktion nicht. FORTH-Anweisungen werden WORDS genannt. Wie eine traditionelle Sprache organisiert FORTH die Wörter, aus denen es besteht, in VOKABULAREN, einer Reihe von Wörtern mit einem gemeinsamen Merkmal.

Beim Programmieren in FORTH geht es darum, ein vorhandenes Vokabular zu bereichern oder ein neues Vokabular zu definieren, das sich auf die zu entwickelnde Anwendung bezieht.

Liste der Vokabeln

Ein Vokabular ist eine geordnete Liste von Wörtern, die vom zuletzt erstellten Wort bis zum zuletzt erstellten Wort durchsucht wird. Die Suchreihenfolge ist ein Stapel von Vokabeln. Durch das Ausführen eines Vokabularnamens wird der oberste Rand des Suchreihenfolgestapels durch dieses Vokabular ersetzt.

Um die Liste der verschiedenen in ESP32forth verfügbaren Vokabulare anzuzeigen, verwenden wir das Wort **voclist** :

```
-> internals voclist \ wird angezeigt
registers
ansi
editor
streams
tasks
rtos
sockets
Serial
ledc
SPIFFS
SD_MMC
SD
WiFi
Wire
ESP
structures
internalized
internals
FORTH
```

Diese Liste ist nicht begrenzt. Wenn wir bestimmte Erweiterungen kompilieren, können zusätzliche Vokabulare erscheinen.

Das Hauptvokabular heißt **FORTH** . Alle anderen Vokabeln sind dem **FORTH**- Vokabular zugeordnet .

Grundlegende Vokabeln

Hier ist die Liste der wichtigsten Vokabeln, die in ESP32forth verfügbar sind :

- **ansi**- Anzeigeverwaltung in einem ANSI-Terminal;
- **editor** bietet Zugriff auf Befehle zum Bearbeiten von Blockdateien;
- **oled** Management von 128 x 32 oder 128 x 64 Pixel großen OLED-Displays. Der Inhalt dieses Vokabulars ist erst nach dem Kompilieren der Erweiterung **oled.h** verfügbar ;
- **structures** komplexer Strukturen;

Liste der Vokabelinhalte

Um den Inhalt eines Vokabulars anzuzeigen, verwenden wir das Wort **vlist** , nachdem wir zuvor das entsprechende Vokabular ausgewählt haben:

```
sockets vlist
```

Sockets- Vokabular aus und zeigt seinen Inhalt an:

```
--> sockets vlist \ zeigt an:  
ip. ip# ->h_addr ->addr! ->addr@ ->port! ->port@ sockaddr l, s, bs, SO_REUSEADDR  
SOL SOCKET sizeof(sockaddr_in) AF_INET SOCK_RAW SOCK_DGRAM SOCK_STREAM  
socket setsockopt bind listen connect sockaccept select poll send sendto  
sendmsg recv recvfrom recvmsg gethostbyname errno sockets-builtins
```

Durch die Auswahl eines Vokabulars erhalten Sie Zugriff auf die in diesem Vokabular definierten Wörter.

Beispielsweise ist das Wort **voclist** nicht zugänglich, ohne zuvor die Vokabularinterna aufzurufen .

Das gleiche Wort kann in zwei verschiedenen Vokabularien definiert werden und zwei unterschiedliche Aktionen haben: Das Wort **l** ist sowohl im **asm** als auch im **editor** Vokabular definiert .

Noch deutlicher wird dies beim Wort **server**, das in den Vokabularien **httpd** , **telnetd** und **web-interface** definiert ist .

Verwendung von Vokabeln

Um ein Wort zusammenzustellen, das in einem anderen Vokabular als FORTH definiert ist, gibt es zwei Lösungen. Die erste Lösung besteht darin, einfach dieses Vokabular aufzurufen, bevor das Wort definiert wird, das Wörter aus diesem Vokabular verwendet.

Hier definieren wir ein Wort **vom serial2-type**, das das im seriellen Vokabular definierte Wort **serial2** verwendet:

```
serial \ Auswahlvokabular serial
: serial2-type ( a n -- )
  Serial2.write drop
;
```

Mit der zweiten Lösung können Sie ein einzelnes Wort aus einem bestimmten Vokabular integrieren:

```
: serial2-type ( a n -- )
  \ Wort aus Vokabular serial kompilieren
  [ serial ] Serial2.write [ FORTH ]
  drop
;
```

Die Auswahl eines Vokabulars kann implizit aus einem anderen Wort im FORTH-Vokabular erfolgen.

Verkettung von Vokabeln

Die Reihenfolge, in der ein Wort in einem Vokabular gesucht wird, kann sehr wichtig sein. Bei Wörtern mit demselben Namen beseitigen wir Unklarheiten, indem wir die Suchreihenfolge in den verschiedenen Vokabeln steuern, die uns interessieren.

Bevor wir eine Vokabularkette erstellen, beschränken wir die Suchreihenfolge mit **only**:

```
asm xtensa
order \ zeigt:      xtensa >> asm >> FORTH
only
order \ zeigt:      FORTH
```

Anschließend duplizieren wir die Verkettung der Vokabeln mit dem Wort **also**:

```
only
order \ zeigt:      FORTH
asm also
order \ zeigt:      asm >> FORTH
xtensa
order \ zeigt:      xtensa >> asm >> FORTH
```

Hier ist eine kompakte Verkettungssequenz:

```
only asm also xtensa
```

Das letzte so verkettete Vokabular wird als erstes erforscht, wenn wir ein neues Wort ausführen oder kompilieren.

```
only
order \ zeigt:      FORTH
also ledc also serial also SPIFFS
```

order	\ zeigt:	SPIFFS >> FORTH
	\	Serial >> FORTH
	\	ledc >> FORTH
	\	FORTH

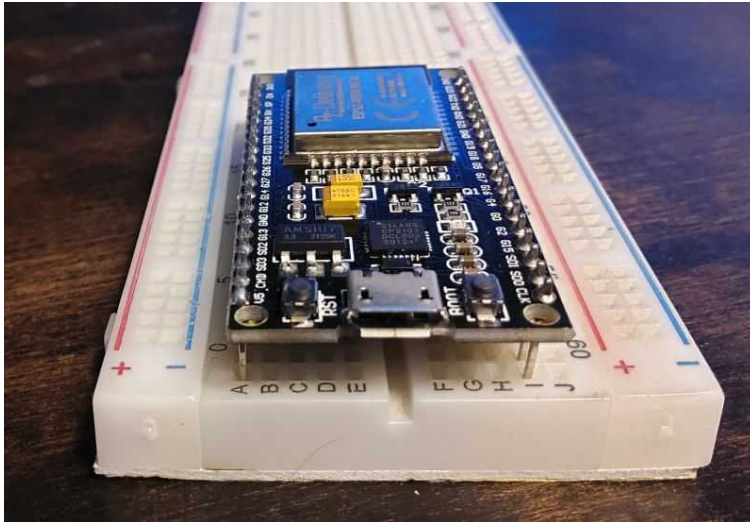
Die Suchreihenfolge beginnt hier mit dem **SPIFFS**- Vokabular , dann mit **serial** , dann mit **ledc** und schließlich mit dem **FORTH**- Vokabular :

- Wenn das gesuchte Wort nicht gefunden wird, liegt ein Kompilierungsfehler vor;
- Wenn das Wort in einem Vokabular gefunden wird, wird dieses Wort zusammengestellt, auch wenn es im folgenden Vokabular definiert ist.

Passen Sie Steckbretter an das ESP32-Board an

Testplatten für ESP32

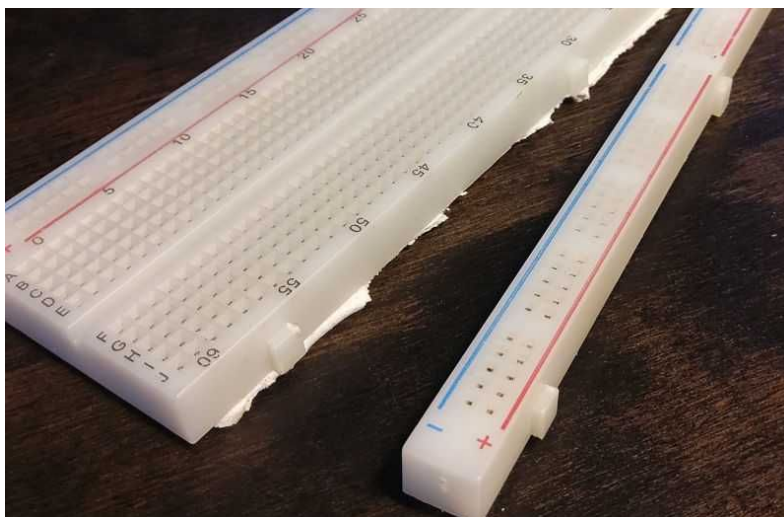
Sie haben gerade Ihre ESP32-Karten erhalten. Und die erste böse Überraschung: Diese Karte passt sehr schlecht auf das Testboard:



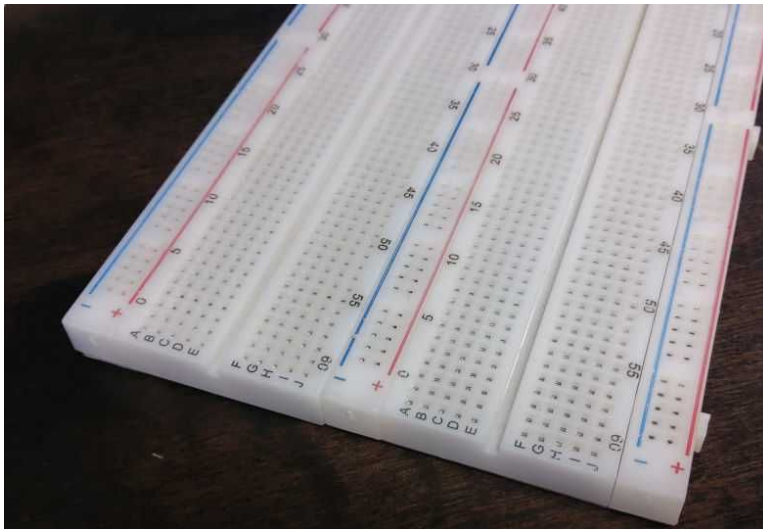
Es gibt kein Steckbrett, das speziell für ESP32-Boards geeignet ist.

Bauen Sie ein Steckbrett, das für das ESP32-Board geeignet ist

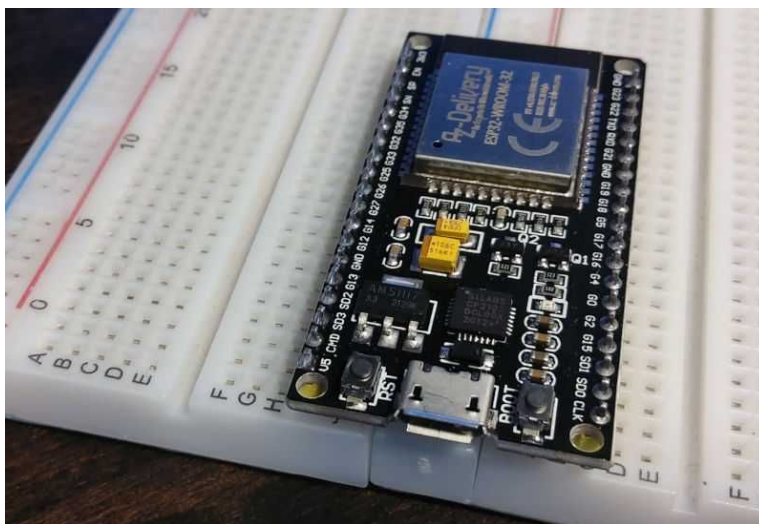
Wir werden unsere eigene Testplatte bauen. Dazu benötigen Sie zwei identische Testplatten. Auf einer der Platten werden wir eine Stromleitung entfernen. Verwenden Sie dazu einen Cutter und schneiden Sie von unten. Sie sollten diese Stromleitung folgendermaßen trennen können:



Mit dieser Karte können wir dann die gesamte Karte wieder zusammensetzen. An den Seiten der Testplatten befinden sich Sparren, um diese miteinander zu verbinden:



Und los geht's! Jetzt können wir unsere ESP32-Karte platzieren:



I/O-Ports können problemlos erweitert werden.

Stromversorgung der ESP32-Karte

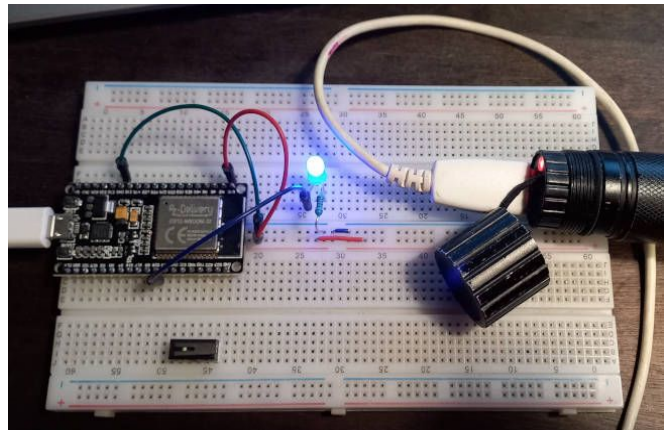
Wahl der Stromquelle

Hier erfahren Sie, wie Sie eine ESP32-Karte mit Strom versorgen. Ziel ist es, Lösungen für die Ausführung von FORTH-Programmen bereitzustellen, die von ESP32forth kompiliert wurden.

Stromversorgung über Mini-USB-Anschluss

Dies ist die einfachste Lösung. Wir ersetzen die vom PC kommende Stromversorgung durch eine andere Quelle:

- eine Netzstromversorgung, wie sie zum Aufladen eines Mobiltelefons verwendet wird;
- ein Backup-Akku für ein Mobiltelefon (Powerbank).



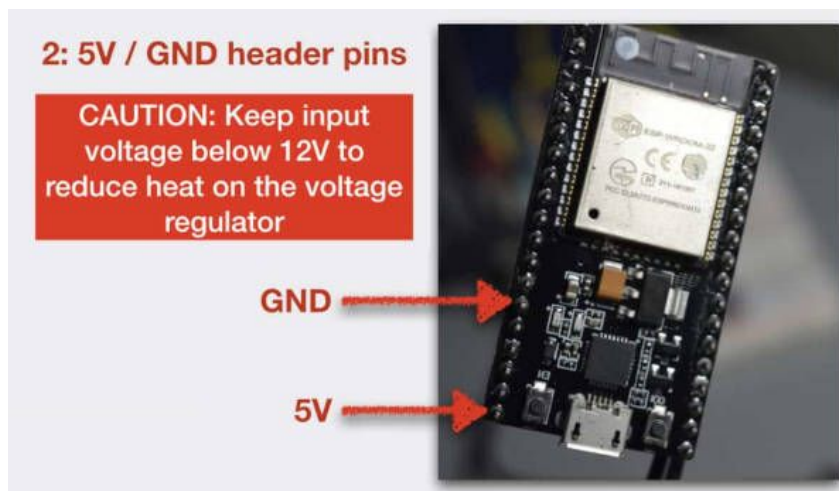
Hier versorgen wir unser ESP32-Board mit einem Backup-Akku für mobile Geräte.

Stromversorgung über 5V-Pin

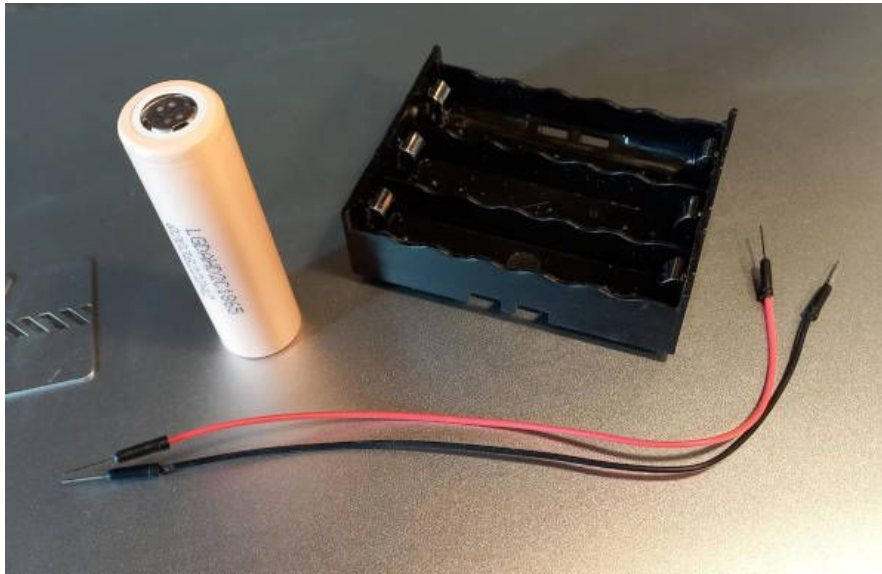
Die zweite Möglichkeit besteht darin, eine externe unregulierte Stromversorgung an den 5V-Pin und Masse anzuschließen. Alles zwischen 5 und 12 Volt sollte funktionieren.

Es ist jedoch am besten, die Eingangsspannung bei etwa 6 oder 7 Volt zu halten, um zu vermeiden, dass zu viel Strom durch Wärme am Spannungsregler verloren geht.

Hier sind die Anschlüsse, die eine externe 5-12-V-Stromversorgung ermöglichen:



Um die 5V-Stromversorgung nutzen zu können, benötigen Sie folgende Ausrüstung:



- zwei 3,7-V-Lithiumbatterien
- ein Batteriehalter
- zwei Dupont-Söhne

Wir löten ein Ende jedes Dupont-Kabels an die Anschlüsse der Batteriehalterung. Hier nimmt unser Halter drei Batterien auf. Wir werden nur eine Batterieeinheit betreiben. Die Batterien sind in Reihe geschaltet.

Sobald die Dupont-Drähte verlötet sind, installieren wir die Batterie und prüfen, ob die Ausgangspolarität eingehalten wird:



Jetzt können wir unsere ESP32-Karte über den 5V-Pin mit Strom versorgen.

ACHTUNG : Die Batteriespannung muss zwischen 5 und 12 Volt liegen.

Automatischer Start eines Programms

Wie können wir sicher sein, dass die ESP32-Karte gut funktioniert, sobald sie mit unseren Batterien betrieben wird?

Die einfachste Lösung besteht darin, ein Programm zu installieren und dieses Programm so einzustellen, dass es automatisch startet, wenn die ESP32-Karte eingeschaltet wird. Kompilieren Sie dieses Programm:

```
18 constant myLED

0 value LED_STATE

: led.on ( -- )
    HIGH dup myLED pin
    to LED_STATE
;

: led.off ( -- )
    LOW dup myLED pin
    to LED_STATE
;

timers also \ select timers vocabulary

: led.toggle ( -- )
    LED_STATE if
        led.off
    else
        led.on
    then
    0 rerun
;

: led.blink ( -- )
    myLED output pinMode
    [''] led.toggle 500000 0 interval
    led.toggle
;

startup: led.blink
bye
```

Installieren Sie eine LED am G18-Pin.

Schalten Sie den Strom aus und schließen Sie die ESP32-Karte wieder an. Wenn alles gut gelaufen ist, sollte die LED nach einigen Sekunden blinken. Dies ist ein Zeichen dafür, dass das Programm ausgeführt wird, wenn das ESP32-Board startet.

Ziehen Sie den USB-Anschluss ab und schließen Sie den Akku an. Das ESP32-Board sollte hochfahren und die LED blinken.

Das ganze Geheimnis liegt in der **startup: led.blink** . Diese Sequenz friert den von ESP32forth kompilierten FORTH-Code ein und legt das Wort **led.blink** als das Wort fest, das beim Starten von ESP32forth ausgeführt werden soll, sobald die ESP32-Karte eingeschaltet ist.

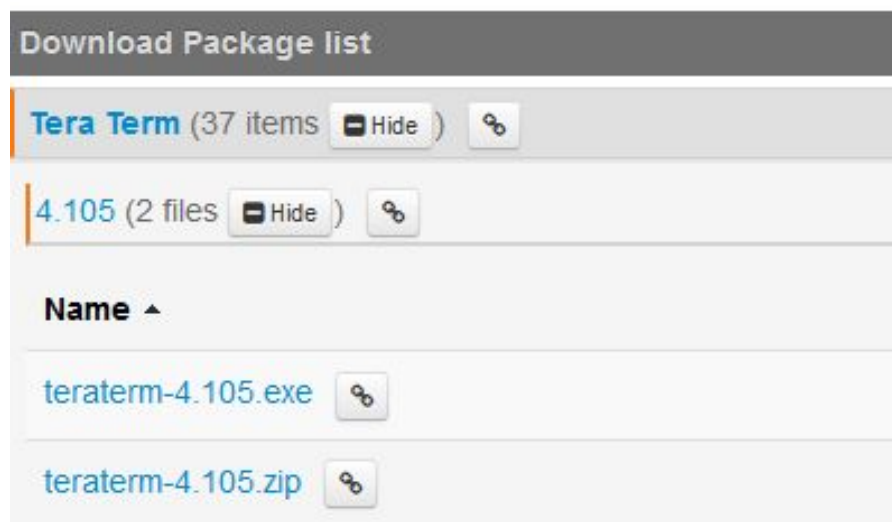
Installieren und verwenden Sie das Tera Term- Terminal unter Windows

Installieren Sie Tera Term

Die englische Seite für Tera Term finden Sie hier:

<https://ttssh2.osdn.jp/index.html.en>

Gehen Sie zur Download-Seite und holen Sie sich die EXE- oder ZIP-Datei:

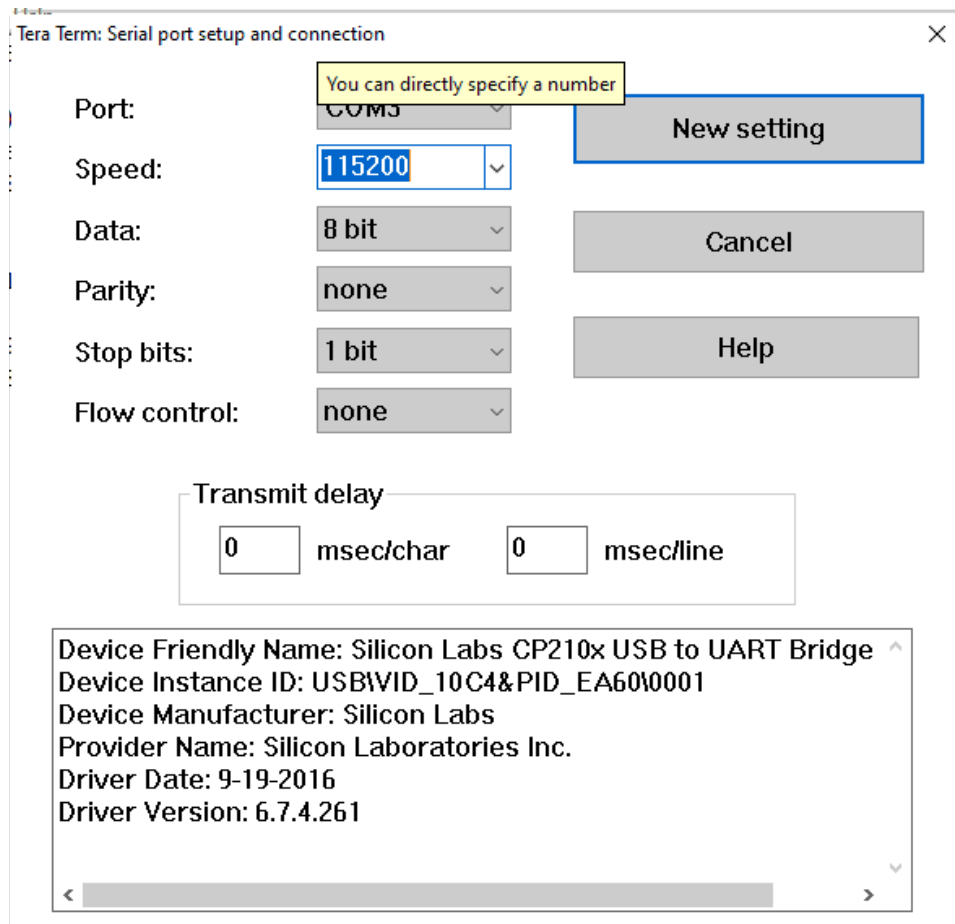


Installieren Sie Tera Term. Die Installation ist schnell und einfach.

Tera Term einrichten

Um mit der ESP32-Karte zu kommunizieren, müssen Sie bestimmte Parameter anpassen:

- Klicken Sie auf Konfiguration -> Serieller Port



Für komfortables Betrachten:

- Klicken Sie auf Konfiguration -> Fenster

Tera Term: Window setup

Title:

Cursor shape

☒ Block

☐ Vertical line

☐ Horizontal line

☐ Hide title bar

☐ Hide menu bar

☒ 16 Colors (PC style)

☒ 16 Colors (aixterm style)

☒ 256 Colors (xterm style)

☒ Enable bold font

☒ Scroll buffer: lines

Color

☒ Text Attribute

☐ Background

R: 0 < > ABC

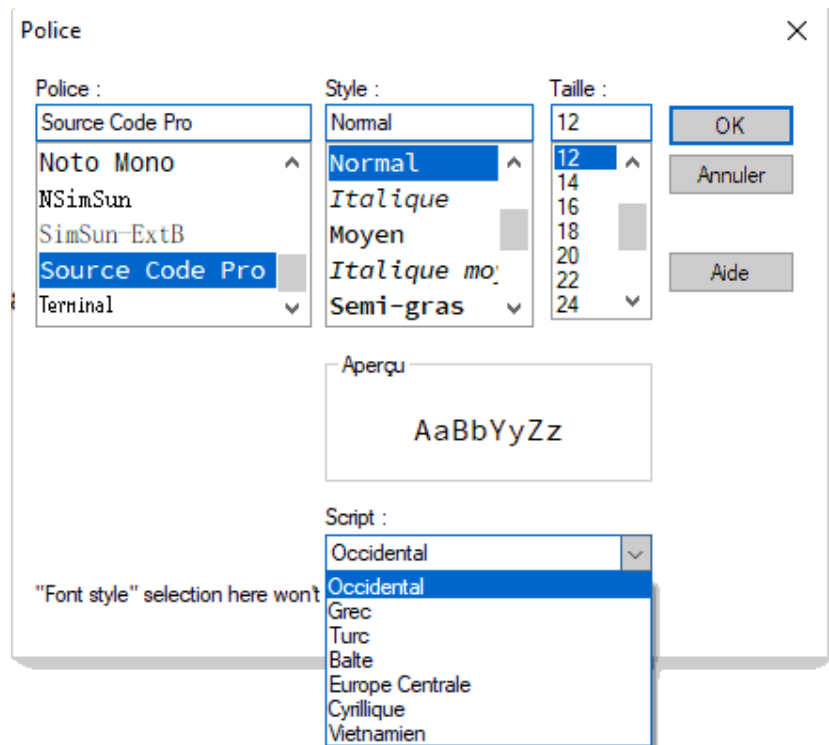
G: 0 < >

B: 0 < >

☒ Always use Normal text's BG

Für lesbare Zeichen:

- Klicken Sie auf Konfiguration -> Schriftart



Um alle diese Einstellungen beim nächsten Start des Tera Term-Terminals wiederzufinden, speichern Sie die Konfiguration:

- Klicken Sie auf *Setup -> Setup speichern*
- Akzeptieren Sie den Namen **TERATERM.INI** .

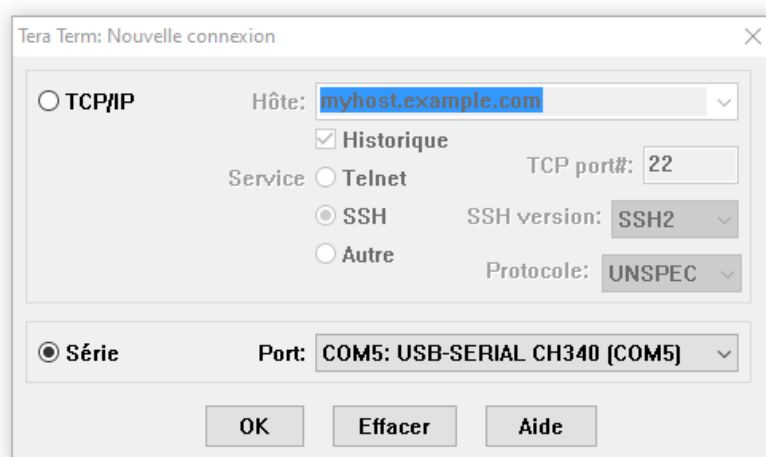
Verwendung von Tera Term

Schließen Sie nach der Konfiguration Tera Term.

Verbinden Sie Ihr ESP32-Board mit einem freien USB-Anschluss Ihres PCs.

Starten Sie Tera Term neu und klicken Sie dann auf *Datei -> Neue Verbindung*

Wählen Sie den seriellen Port aus :



Wenn alles gut gelaufen ist, sollten Sie Folgendes sehen:

A screenshot of a Tera Term VT window titled "COM3 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area displays "ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d" followed by "ok" and a "-->" prompt. A vertical scrollbar is visible on the right side of the text area.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help
ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d
ok
-->
```

Kompilieren Sie den Quellcode in der Forth-Sprache

Denken wir zunächst daran, dass sich die FORTH-Sprache auf dem ESP32-Board befindet! FORTH ist nicht auf Ihrem PC. Daher können Sie den Quellcode eines Programms in der FORTH-Sprache nicht auf dem PC kompilieren.

Um ein Programm in der FORTH-Sprache zu kompilieren, müssen Sie zunächst eine Quelldatei auf dem PC mit dem Editor Ihrer Wahl öffnen.

Anschließend kopieren wir den Quellcode zum Kompilieren. Hier Open-Source-Code mit Wordpad:

```
\ *** decode content of LoRaRX
*****

2 string $CrLf
$0d $CrLf c+$!
$0a $CrLf c+$!

\ delete crlf at end of string
: normalize$ ( addr len -- )
  2dup + 2 - 2
  $CrLf $= if \ if end string = crlf
  2 - swap
  cell - ! \ subtract 2 at length of string
else
  2drop
then
;

\ test if string begin with "+RCV="
: RCV? ( addr len -- f1 )
  dup 0 > if
  drop 5
  s" +RCV=" $=
```

Der Quellcode in FORTH-Sprache kann mit jedem Texteditor erstellt und bearbeitet werden: Notepad, PSpad, Wordpad.

Persönlich verwende ich die Netbeans-IDE. Mit dieser IDE können Sie Quellcodes in vielen Programmiersprachen bearbeiten und verwalten.

Wählen Sie den Quellcode oder Teil des Codes aus, der Sie interessiert. Klicken Sie dann auf Kopieren. Der ausgewählte Code befindet sich im PC-Bearbeitungspuffer.

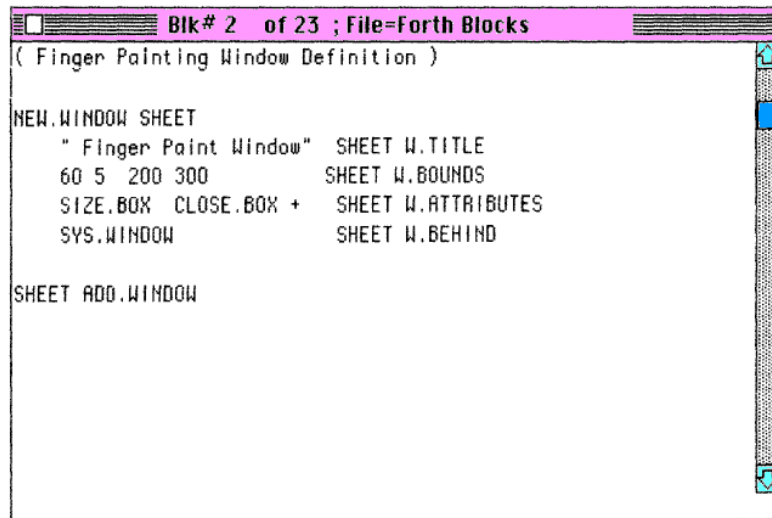
Klicken Sie auf das Tera Term-Terminalfenster. Paste herstellen.

Bestätigen Sie einfach durch Klicken auf „OK“ und der Code wird interpretiert und/oder kompiliert. Um kompilierten Code auszuführen, geben Sie einfach das Wort FORTH zum Starten über das Tera Term-Terminal ein.

Verwaltung von Quelldateien nach Blöcken

Die Blöcke

Hier ein Block auf einem alten Computer :



```
Blk# 2 of 23 ; File=Forth Blocks
( Finger Painting Window Definition )

NEW WINDOW SHEET
  " Finger Paint Window" SHEET W.TITLE
  60 5 200 300 SHEET W.BOUNDS
  SIZE.BOX CLOSE.BOX + SHEET W.ATTRIBUTES
  SYS.WINDOW SHEET W.BEHIND

SHEET ADD.WINDOW
```

Ein Block ist ein Speicherplatz, dessen Einheit 16 Zeilen mit 64 Zeichen umfasst. Die Größe eines Blocks beträgt also $16 \times 64 = 1024$ Bytes. Es ist genau so groß wie ein Kilobyte!

Öffnen Sie eine Blockdatei

Eine Datei ist standardmäßig bereits geöffnet, wenn ESP32forth startet.

Datei **blocks.fb** .

Im Zweifelsfall führen Sie **default-use** aus .

Um herauszufinden, was in dieser Datei enthalten ist, verwenden Sie die Editor-Befehle, indem Sie zuerst **editor** eingeben .

Hier sind unsere ersten Befehle, die Sie kennen sollten, um den Inhalt von Blöcken zu verwalten:

- **l** listet den Inhalt des aktuellen Blocks auf
- **n** wählt den nächsten Block aus
- **p** wählt den vorherigen Block aus

ACHTUNG: Ein Block hat immer eine Nummer zwischen 0 und n. Wenn Sie am Ende eine negative Blocknummer erhalten, wird ein Fehler generiert.

Bearbeiten Sie den Inhalt eines Blocks

Nachdem wir nun wissen, wie man einen bestimmten Block auswählt, sehen wir uns an, wie man Quellcode in die FORTH-Sprache einfügt ...

Eine Strategie besteht darin, mit einem Texteditor eine Quelldatei auf Ihrem Computer zu erstellen. Sie müssen dann nur noch Ihren Quellcode zeilenweise in die Blockdateien kopieren/einfügen.

Hier sind die wesentlichen Befehle zum Verwalten des Inhalts eines Blocks:

- **Wipe** wird der Inhalt des aktuellen Blocks geleert
- **d** löscht Zeile n. Die Zeilennummer muss im Bereich 0..14 liegen. Die folgenden Zeilen bewegen sich nach oben. Beispiel: 3 D löscht den Inhalt von Zeile 3 und ruft den Inhalt der Zeilen 4 bis 15 auf.
- **e** löscht den Inhalt der Zeile n. Die Zeilennummer muss im Bereich 0..15 liegen. Die anderen Zeilen gehen nicht nach oben.
- **a** fügt eine Zeile n ein. Die Zeilennummer muss im Bereich 0..14 liegen. Die nach der eingefügten Zeile liegenden Zeilen werden wieder nach unten verschoben. Beispiel: 3 Ein Test fügt test in Zeile 3 ein und verschiebt den Inhalt der Zeilen 4 nach unten nach 15.
- **r** ersetzt den Inhalt der Zeile n. Beispiel: 3 R test ersetzt den Inhalt von Zeile 3 durch test

Hier ist unser Block 0, der derzeit bearbeitet wird:

```
Block 0
| 0
create sintab \ 0...90 Grad, Index in Grad | 1
0000 , 0175 , 0349 , 0523 , 0698 , | 2
0872 , 1045 , 1219 , 1392 , 1564 , | 3
1736 , 1908 , 2079 , 2250 , 2419 , | 4
2588 , 2756 , 2924 , 3090 , 3256 , | 5
3420 , 3584 , 3746 , 3907 , 4067 , | 6
4226 , 4384 , 4540 , 4695 , 4848 , | 7
5000 , 5150 , 5299 , 5446 , 5592 , | 8
5736 , 5878 , 6018 , 6157 , 6293 , | 9
| 10
| 11
| 12
| 13
| 14
| 15
ok
--> 10 R 6428 , 6561 , 6691 , 6820 , 6947 ,
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Déconr
```

Am unteren Bildschirmrand wird Zeile **10 R 6428, 6561,** in unseren Block bei Zeile 10 integriert.

Sie bemerken, dass Zeile 0 keinen Inhalt hat. Dies erzeugt einen Fehler beim Kompilieren des FORTH-Codes. Um dies zu beheben, geben Sie einfach **0 R** gefolgt von zwei Leerzeichen ein.

Mit etwas Übung haben Sie in wenigen Minuten Ihren FORTH-Code in diesen Block eingefügt.

Machen Sie bei Bedarf dasselbe für die folgenden Blöcke. Wenn Sie zum nächsten Block wechseln, erzwingen Sie das Speichern des Inhalts der Blöcke, indem Sie **„flush“ eingeben** .

Blockinhalte zusammenstellen

Bevor wir den Inhalt einer Blockdatei kompilieren, prüfen wir, ob der Inhalt gut gespeichert ist. Dafür:

- Geben Sie **flush** ein und trennen Sie dann die ESP32-Karte.
- Warten Sie einige Sekunden und schließen Sie die ESP32-Karte wieder an.
- Geben Sie **editor** und **1** . Sie müssen Ihren Block 0 mit dem von Ihnen bearbeiteten Inhalt finden.

Um den Inhalt Ihrer Blöcke zusammenzustellen, haben Sie zwei Wörter:

- **load** mit vorangestellter Nummer des Blocks, dessen Inhalt wir ausführen und/oder kompilieren möchten. Um den Inhalt unseres Blocks 0 zu kompilieren, führen wir **0load aus** ;
- **thru** , dem zwei Blocknummern vorangestellt sind, führt den Inhalt der Blöcke aus und/oder kompiliert ihn, als würden wir eine Folge von **Ladewörtern ausführen** .
Beispiel: **0 2 bis** führt den Inhalt der Blöcke 0 bis 2 aus und/oder kompiliert ihn.

Die Geschwindigkeit der Ausführung und/oder Kompilierung von Blockinhalten erfolgt nahezu augenblicklich.

Praktisches Schritt-für-Schritt-Beispiel

Wir werden anhand eines praktischen Beispiels sehen, wie man Quellcode in Block 1 einfügt. Wir nehmen einen Code, der bereit ist, in unseren Block integriert zu werden:

```
1 list
editor
0 r \ tools for REGISTERS definitions and manipulations
1 r : mclr { mask addr -- }      addr @  mask invert and addr !  ;
2 r : mset { mask addr -- }      addr @  mask or addr !  ;
```

```

3 r : mtst { mask addr -- x }  addr @  mask and ;
4 r : defREG: \ define a register, similar as constant
5 r      create ( addr1 -- <name> )  ,
6 r      does> ( -- regAddr )        @  ;
7 r : .reg ( reg -- ) \ display reg content
8 r      base @ >r binary @ <#
9 r      4 for aft 8 for aft # then next
10 r      bl hold then next #>
11 r      cr space ." 33222222 22221111 11111100 00000000"
12 r      cr space ." 10987654 32109876 54321098 76543210"
13 r      cr type r> base ! ;
14 r : defMASK: create ( mask0 position -- )  lshift ,
15 r      does> ( -- mask1 )                @  ;
save-buffers

```

Kopieren Sie einfach Teile des obigen Codes, fügen Sie ihn ein und führen Sie diesen Code über ESP32 Forth aus:

- **1 list** , um auszuwählen und zu sehen, was Block 1 enthält
- **editor** , um den **Vokabeleditor** auszuwählen
- Kopieren Sie die Zeilen **n r...** in Dreierpaketen und führen Sie sie aus
- **save-buffers** speichert Code fest in einer Blockdatei

Schalten Sie die ESP32-Karte aus. Starten Sie es neu. Wenn Sie **1 list** eingeben, sollte der Code bearbeitet und gespeichert werden.

Um diesen Code zu kompilieren, geben Sie einfach **1 load** ein.

Abschluss

Der verfügbare Dateispeicher für ESP32forth beträgt etwa 1,8 MB. Sie können daher problemlos Hunderte von Blöcken für Quelldateien in der FORTH-Sprache verwalten. Es wird empfohlen, Quellcodes aus stabilen Codeteilen zu installieren. Somit wird es während der Programmentwicklungsphase viel einfacher, in Ihren Code in der Entwicklungsphase zu integrieren:

```

2 5 thru \ integrierte pwm-Befehle für Motoren

```

anstatt diesen Code systematisch über die serielle Schnittstelle oder WLAN neu zu laden.

Der andere Vorteil von Blöcken besteht darin, dass sie die Einbettung von Parametern, Datentabellen usw. vor Ort ermöglichen, die dann von Ihren Programmen verwendet werden können.

Bearbeiten von Quelldateien mit VISUAL Editor

Bearbeiten Sie eine FORTH-Quelldatei

Um eine FORTH-Quelldatei mit ESP32forth zu bearbeiten, verwenden wir den visuellen Editor.

Um eine **dump.fs**- Datei zu bearbeiten , gehen Sie von dem Terminal aus, das an eine ESP32-Karte mit ESP32forth angeschlossen ist, wie folgt vor:

```
visual edit /spiffs/dump.fs
```

Der vollständige **DUMP**- Code ist hier verfügbar:

<https://github.com/MPETREMANNN11/ESP32forth/blob/main/tools/dumpTool.txt>

das Wort **edit** folgt das Verzeichnis, in dem die Quelldateien gespeichert sind:

- Wenn die Datei nicht existiert, wird sie erstellt.
- Wenn die Datei vorhanden ist, wird sie im Editor abgerufen.

Notieren Sie sich den Namen der von Ihnen erstellten Datei.

Sie **fs** als Dateierweiterung für **Forth Source** .

Bearbeiten des FORTH-Codes

Bewegen Sie im Editor den Cursor mit den auf der Tastatur verfügbaren Links-Rechts-Aufwärts-Abwärtspfeilen.



Das Terminal aktualisiert die Anzeige jedes Mal, wenn der Cursor bewegt oder der Quellcode geändert wird.

Um den Editor zu verlassen:

- CTRL-S: speichert den Inhalt der aktuell bearbeiteten Datei
- CTRL-X: Bearbeitung beenden:
 - N: ohne Dateiänderungen zu speichern
 - Y: mit Speicherung der Änderungen

Kompilieren von Dateiinhalten

Das Kompilieren des Inhalts unserer **dump.fs**- Datei erfolgt folgendermaßen:

```
include /spiffs/dump.fs
```

Das Kompilieren geht deutlich schneller als über das Terminal.

Die mit ESP32forth in die ESP32-Karte eingebetteten Quelldateien sind persistent. Nach dem Ausschalten und erneuten Anschließen der ESP32-Karte bleibt die gespeicherte Datei sofort verfügbar.

Sie können so viele Dateien wie nötig definieren.

Daher ist es einfach, in die ESP32-Karte eine Sammlung von Tools und Routinen zu integrieren, aus denen Sie je nach Bedarf schöpfen können.

Das SPIFFS-Dateisystem

ESP32Forth enthält ein rudimentäres Dateisystem im internen Flash-Speicher. Der Zugriff auf die Dateien erfolgt über eine serielle Schnittstelle namens SPIFFS für Serial Peripheral Interface Flash File System.

Auch wenn das SPIFFS-Dateisystem einfach ist, erhöht es die Flexibilität Ihrer Entwicklungen mit ESP32Forth erheblich:

- Konfigurationsdateien verwalten
- Integrieren Sie auf Anfrage verfügbare Softwareerweiterungen
- Modularisieren Sie Entwicklungen in wiederverwendbare Funktionsmodule

Und viele andere Verwendungsmöglichkeiten, die wir Ihnen zeigen werden...

Zugriff auf das SPIFFS-Dateisystem

Geben Sie Folgendes ein, um den Inhalt einer durch visuelle Bearbeitung bearbeiteten Quelldatei zu kompilieren:

```
INCLUDE /spiffs/dumpTool.fs
```

Das Wort **include** muss immer vom Terminal aus verwendet werden.

Um die Liste der SPIFFS-Dateien anzuzeigen, verwenden Sie das Wort **ls** :

```
ls /spiffs/  
\ Zeigt an:  
\ dumpTool.fs
```

Hier wurde die Datei **dumpTool.fs** gespeichert. Für SPIFFS sind Dateierweiterungen irrelevant. Dateinamen dürfen keine Leerzeichen oder das /-Zeichen enthalten.

Lassen Sie uns eine neue **myApp.fs**- Datei mit dem **visual editor** bearbeiten und speichern . Lassen Sie uns noch einmal **ls** ausführen :

```
ls /spiffs/  
\ Anzeige:  
\ dumpTool.fs  
\ myApp.fs
```

Das SPIFFS-Dateisystem verwaltet keine Unterordner wie auf einem Linux-Computer. Um ein Pseudoverzeichnis zu erstellen, geben Sie es einfach beim Erstellen einer neuen Datei an. Lassen Sie uns beispielsweise die Datei **other/myTest.fs** bearbeiten . Sobald wir es bearbeitet und gespeichert haben, führen wir **ls aus** :

```
ls /spiffs/  
\ Anzeige:  
\ dumpTool.fs  
\ myApp.fs  
\ other/myTest.fs
```

Anderen Pseudoverzeichnis anzeigen möchten , müssen Sie **/spiffs/** mit dem Namen dieses Pseudoverzeichnisses folgen :

```
ls /spiffs/other  
\ Anzeige:  
\ myTest.fs
```

Es gibt keine Möglichkeit, Dateinamen oder Pseudoverzeichnisse zu filtern.

Umgang mit Dateien

Um eine Datei vollständig zu löschen, verwenden Sie das Wort **rm** gefolgt vom Namen der zu löschenden Datei:

```
rm /spiffs/other/myTest.fs  
ls /spiffs/  
\ anzeige:  
\ dumpTool.fs  
\ myApp.fs
```

Um eine Datei umzubenennen, verwenden Sie das Wort **mv** :

```
mv /spiffs/myApp.fs /spiffs/main.fs  
ls /spiffs/  
\ anzeige:  
\ dumpTool.fs  
\ main.fs
```

Um eine Datei zu kopieren, verwenden Sie das Wort **cp** :

```
cp /spiffs/main.fs /spiffs/mainTest.fs  
ls /spiffs/  
\ anzeige:  
\ dumpTool.fs  
\ main.fs  
\mainTest.fs
```

Um den Inhalt einer Datei anzuzeigen, verwenden Sie das Wort **cat** :

```
cat /spiffs/dumpTool.fs  
\ zeigt den Inhalt von dumpTool.fs an
```

Um den Inhalt einer Zeichenfolge in einer Datei zu speichern, gehen Sie in zwei Phasen vor:

- Erstellen Sie eine neue Datei mit **touch**
- Dump-Datei speichern mit **dump-file**

```
touch /spiffs/mTest.fs \ erstellt eine neue mTest,fs-Datei
ls /spiffs/           \ zeigt Folgendes an:
ls /spiffs/

\ Zeichenfolge „Meinen Text in mTest einfügen“ in mTest speichern
r| ." Meinen Text in mTest einfügen" | s" /spiffs/mTest" dump-file

include /spiffs/mTest \ zeigt an: Meinen Text in mTest einfügen
```

Organisieren und kompilieren Sie Ihre Dateien auf der ESP32-Karte

Wir werden sehen, wie man Dateien für eine Anwendung verwaltet, die auf einem ESP32-Board entwickelt wird, auf dem ESP32forth installiert ist.

Es wird vereinbart, dass alle verwendeten Dateien im ASCII-Textformat vorliegen.

Die folgenden Erläuterungen dienen lediglich der Orientierung. Sie verfügen über eine gewisse Erfahrung und zielen darauf ab, die Entwicklung großer Anwendungen mit ESP32forth zu erleichtern.

Quelldateien bearbeiten und übertragen

Alle Quelldateien für Ihr Projekt befinden sich auf Ihrem Computer. Es empfiehlt sich, diesem Projekt einen Unterordner zuzuweisen. Sie arbeiten beispielsweise an einem SSD1306 OLED-Display. Sie erstellen also ein Verzeichnis mit dem Namen SSD1306.

Bezüglich Dateinamenerweiterungen empfehlen wir die Verwendung der **fs- Erweiterung**.

Das Bearbeiten von Dateien auf einem Computer erfolgt mit einem beliebigen Textdateieditor.

Verwenden Sie in diesen Quelldateien keine Zeichen, die nicht im ASCII-Code enthalten sind. Einige erweiterte Codes können die Programmkompilierung stören.

Diese Quelldateien werden dann über die serielle Verbindung und ein terminalartiges Programm auf die ESP32-Karte kopiert oder übertragen:

- durch Kopieren/Einfügen mit Visual auf ESP32forth, reserviert für kleine Dateien;
- mit einem spezifischen Verfahren, das später für wichtige Dateien detailliert beschrieben wird.

Organisieren Sie Ihre Dateien

Im Folgenden werden alle unsere Dateien die Erweiterung **fs haben**.

Beginnen wir mit unserem SSD1306-Verzeichnis auf unserem Computer.

, ist die Datei **main.fs**. Diese Datei enthält die Aufrufe zum Laden aller anderen Dateien unserer in der Entwicklung befindlichen Anwendung.

Beispiel für den Inhalt unserer **main.fs**- Datei :

```
\ OLED SSD1306 128x32 dev et tests affichage
s" /SPIFFS/config.fs" included
```

In der Entwicklungsphase wird der Inhalt dieser **main.fs**- Datei manuell geladen, indem **include** wie folgt ausgeführt wird:

```
include /spiffs/main.fs
```

Dadurch wird der Inhalt unserer **main.fs**- Datei ausgeführt . Das Laden anderer Dateien erfolgt aus dieser **main.fs**- Datei . Hier laden wir die Datei **config.fs** , von der hier ein Auszug ist:

```
\ *****
\ Konfiguration für SSD1306 128x32 OLED-Display
\ *****

\ für SSD1306_128_32
  128 constant SSD1306_LCDWIDTH
  32 constant SSD1306_LCDHEIGHT
```

config.fs- Datei werden wir alle konstanten Werte und verschiedenen Parameter einfügen, die von den anderen Dateien verwendet werden.

Unsere nächste Datei wird **SSD10306commands.fs** sein . So laden Sie den Inhalt aus main.fs:

```
\ OLED SSD1306 128x32 Entwicklungs- und Anzeigetests
s" /spiffs/config.fs" included
s" /spiffs/SSD10306commands.fs" included
```

Der Inhalt der Datei **SSD10306commands.fs** umfasst fast 230 Codezeilen. Es ist nicht möglich, den Inhalt dieser Datei Zeile für Zeile in den visuellen Editor von ESP32forth zu kopieren. Hier ist eine Methode, um den Inhalt dieser großen Datei auf einmal auf ESP32forth zu kopieren und zu speichern.

Übertragen Sie eine große Datei an ESP32forth

Um diese große Dateiübertragung zu ermöglichen, kompilieren Sie diesen Code in ESP32forth:

```
: noType
  2drop ;
```

```
visual
: xEdit
  ['] noType is type
  edit
  ['] default-type is type
;
```

Mit diesem sehr kurzen Forth-Code können Sie ein sehr langes FORTH-Programm vom Editor auf dem PC in eine Datei auf der ESP32-Karte übertragen:

- Kompilieren Sie mit ESP32forth den FORTH-Code, hier die Wörter noType und xEdit
- Öffnen Sie das Programm auf Ihrem PC, um es in eine Datei auf der ESP32-Karte zu übertragen
- Fügen Sie die Zeile oben im Programm hinzu, die diese schnelle Übertragung ermöglicht: **xEdit /spiffs/SSD10306commands.fs**
- Kopieren Sie den gesamten Code Ihres bearbeiteten Programms auf Ihren PC.
- in das mit ESP32 verbundene Terminal weiterleiten
- Kopieren Sie Ihren Code

Wenn alles gut geht, sollte auf dem Terminalbildschirm nichts erscheinen. Warten Sie ein paar Sekunden.

Geben Sie als Nächstes Folgendes ein: STRG-X und drücken Sie dann „Y“.

Sie sollten die Kontrolle wiedererlangen.

Überprüfen Sie das Vorhandensein Ihrer neuen Datei: **ls /spiffs/**

Sie können nun den Inhalt Ihrer neuen Datei kompilieren.

Abschluss

Im ESP32forth SPIFFS-Dateisystem gespeicherte Dateien sind dauerhaft verfügbar.

Wenn Sie das ESP32-Board außer Betrieb nehmen und anschließend wieder einstecken, sind die Dateien sofort verfügbar.

Der Inhalt der Dateien kann vor Ort durch **visual edit** geändert werden.

Dieser Komfort wird die Entwicklung viel schneller und einfacher machen.

Eine Ampel mit ESP32 managen

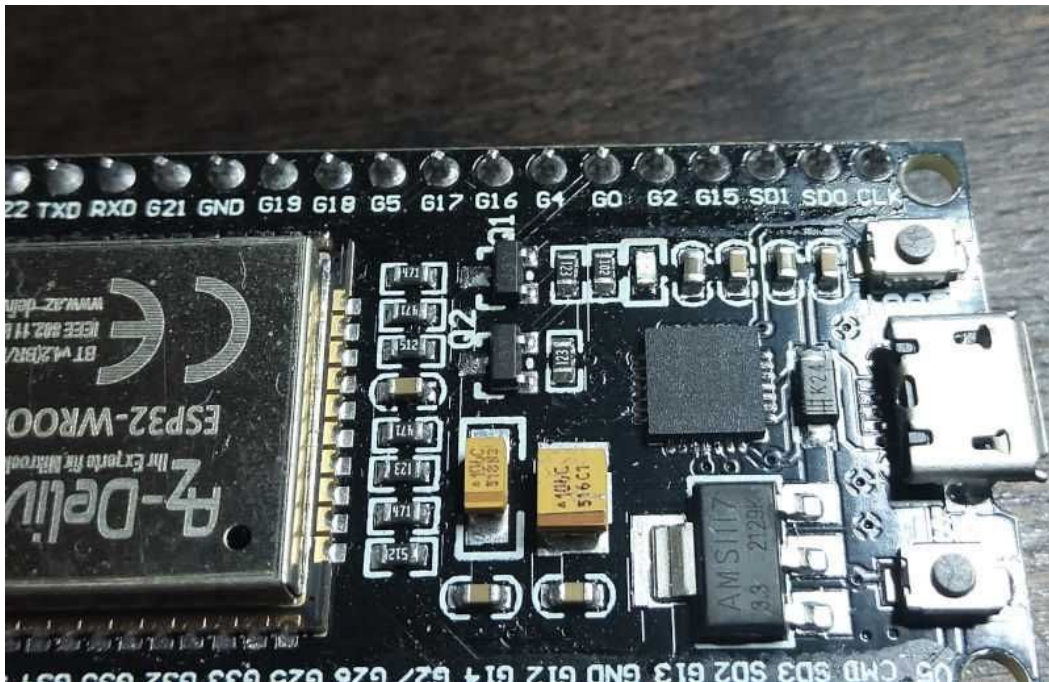
GPIO-Ports auf der ESP32-Karte

GPIO-Ports (General Purpose Input/Output) sind Ein-/Ausgabe-Ports, die in der Welt der Mikrocontroller weit verbreitet sind.

Der ESP32-Chip verfügt über 48 Pins mit mehreren Funktionen. Auf ESP32-Entwicklungsboards werden nicht alle Pins verwendet und einige Pins können nicht verwendet werden.

Es gibt viele Fragen zur Verwendung von ESP32-GPIOs. Welche Anschlüsse sollten Sie verwenden? Welche Konnektoren sollten Sie in Ihren Projekten vermeiden?

Wenn wir eine ESP32-Karte unter die Lupe nehmen, sehen wir Folgendes:



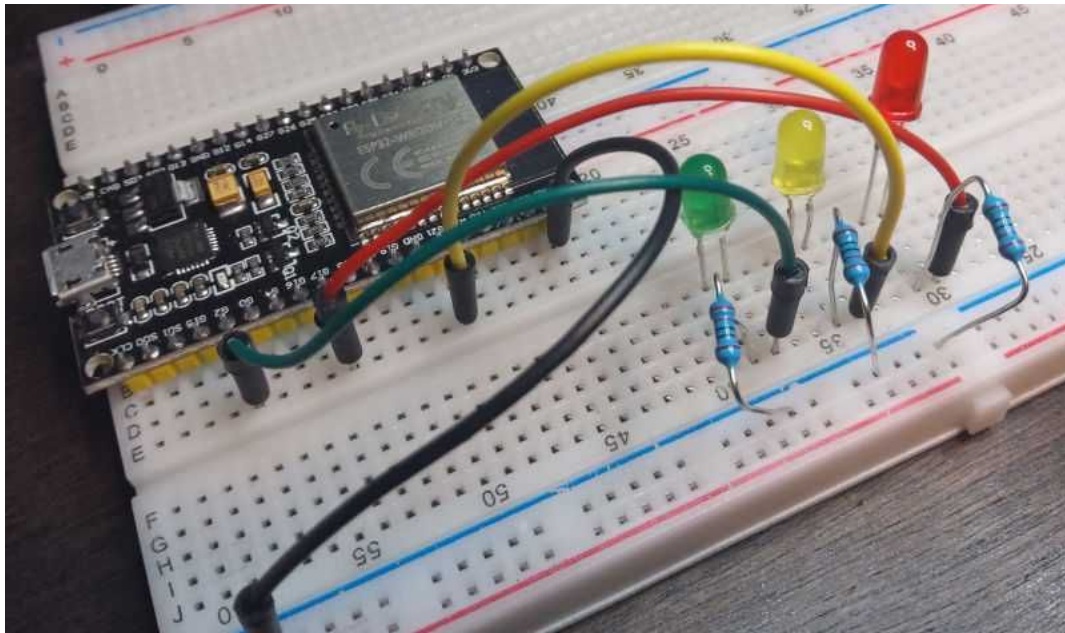
Jeder Anschluss ist durch eine Reihe von Buchstaben und Zahlen gekennzeichnet, hier von links nach rechts auf unserem Foto: G22 TXD RXD G21 GND G19 G18 usw.

Den Konnektoren, die uns für diese Handhabung interessieren, wird der Buchstabe G vorangestellt, gefolgt von einer oder zwei Zahlen. G2 entspricht beispielsweise GPIO 2.

Das Definieren und Betreiben eines GPIO-Anschlusses im Ausgabemodus ist recht einfach.

Montage der LEDs

Der Zusammenbau ist recht einfach und nur ein Foto reicht:



- Grüne LED an G2 angeschlossen – grünes Kabel
- Gelbe LED an G21 angeschlossen – gelbes Kabel
- Rote LED an G17 angeschlossen – rotes Kabel
- Schwarzes Kabel mit GND verbunden

Wir definieren unsere LEDs mit **defPin:**

```
\ Use:
\ numGPIO defPIN: PD7  ( define portD pin #7)
: defPIN: ( GPIOx --- <word> | <word> --- GPIOx )
    value
;

2 defPIN: ledGREEN
21 defPIN: ledYELLOW
17 defPIN: ledRED

: LEDinit
    ledGREEN    output pinMode
    ledYELLOW   output pinMode
    ledRED      output pinMode
;
```

Viele Programmierer haben die schlechte Angewohnheit, Anschlüsse nach ihrer Nummer zu benennen. Beispiel :


```
17 defPin: pin17
```

Oder

```
17 defPin: GPIO17.
```

Um effektiv zu sein, müssen Sie die Anschlüsse nach ihrer Funktion benennen. Hier definieren wir die **ledRED-** oder **ledGREEN- Anschlüsse** .

Wofür? Da Sie an dem Tag, an dem Sie Zubehör hinzufügen und beispielsweise den G21-Stecker lösen müssen, einfach **21 defPIN: ledYELLOW** mit der neuen Steckernummer neu definieren. Der Rest des Codes bleibt unverändert und verwendbar.

Verwaltung von Ampeln

Hier ist der Teil des Codes, der unsere LEDs in unserer Ampelsimulation steuert:

```
\ trafficLights führt einen Lichtzyklus aus
: trafficLights ( ---)
  high ledGREEN   pin    3000 ms    low ledGREEN   pin
  high ledYELLOW  pin     800 ms    low ledYELLOW  pin
  high ledRED     pin    3000 ms    low ledRED     pin
  ;

\ klassische Ampelschleife
: lightsLoop ( ---)
  LEDinit
  begin
    trafficLights
  key? until
  ;

\ Deutscher Ampelstil
: Dtraffic ( ---)
  high ledGREEN   pin    3000 ms    low ledGREEN   pin
  high ledYELLOW  pin     800 ms    low ledYELLOW  pin
  high ledRED     pin    3000 ms
  ledYELLOW high    800 ms
  \ simultaneous red and yellow ON
  high ledRED     pin \ simultaneous red and yellow OFF
  high ledYELLOW  pin
  ;

\ deutsche Ampelschleife
: DlightsLoop ( ---)
  LEDinit
  begin
    Dtraffic
  key? until
  ;
```

Abschluss

Dieses Ampelmanagementprogramm hätte durchaus in der Sprache C geschrieben sein können. Der Vorteil der FORTH-Sprache besteht jedoch darin, dass sie über das Terminal

die Kontrolle über die Analyse, Fehlerbeseitigung und Änderung von Funktionen sehr schnell ermöglicht (in FORTH sagen wir Wörter).

Das Verwalten von Ampeln ist in der C-Sprache eine einfache Übung. Wenn die Programme jedoch etwas komplexer werden, wird der Kompilierungs- und Upload-Prozess schnell mühsam.

Handeln Sie einfach über das Terminal und kopieren/fügen Sie einfach ein beliebiges Fragment des FORTH-Sprachcodes ein, damit es kompiliert und/oder ausgeführt wird.

Wenn Sie ein Terminalprogramm zur Kommunikation mit der ESP32-Karte verwenden, geben Sie einfach **DlightsLoop** oder **LightsLoop** ein , um zu testen, wie das Programm funktioniert. Diese Wörter verwenden eine bedingte Schleife. Drücken Sie einfach eine Taste auf der Tastatur und das Wort wird am Ende der Schleife nicht mehr abgespielt.

Hardware-Interrupts mit ESP32forth

Unterbrechungen

Wenn wir externe Ereignisse, beispielsweise einen Druckknopf, verwalten möchten, haben wir zwei Lösungen:

- Testen Sie den Zustand der Schaltfläche so regelmäßig wie möglich über eine Schleife. Wir werden entsprechend dem Status dieser Schaltfläche handeln.
- Verwenden Sie einen Interrupt. Wir weisen den Ausführungscode einem an einen Pin angeschlossenen Interrupt zu. Der Button ist mit diesem Pin verbunden und der Zustandswechsel führt dieses Wort aus.

Die Interrupt-Lösung ist die eleganteste. Dadurch können Sie das Hauptprogramm entlasten, indem Sie die Überwachung der Schaltfläche in einer Schleife vermeiden.

In seiner ESP32forth-Dokumentation gibt Brad NELSON ein einfaches Beispiel für die Interrupt-Behandlung:

```
17 input pinMode
: test ." pinvalue: " 17 digitalRead . cr ;
' test 17 pinchange
```

Allerdings besteht bei diesem Beispiel, so wie es geschrieben wurde, eine gute Chance, dass es nicht funktioniert. Wir werden sehen, warum und die Elemente bereitstellen, damit es funktioniert.

Montage eines Druckknopfes

Der Taster wird als Eingang an die 3,3V-Stromversorgung des ESP32-Boards angeschlossen.

Der Druckknopfausgang ist mit dem GPIO17-Pin verbunden. Das ist alles.

Damit das Beispiel von Brad NELSON funktioniert, müssen Sie das **interrupts**- Vokabular auswählen , bevor Sie den Interrupt mit **pinchange** konfigurieren . Unterwegs definieren wir die **button** :

```
17 constant button
button input pinMode
: test ." pinvalue: "
    button digitalRead . cr
;
interrupts
' test button pinchange
```

forth

Es funktioniert, aber es gibt einen unerwarteten Effekt, der dazu führt, dass der Interrupt unerwartet ausgelöst wird:

```
pinvalue: 0
pinvalue: 1
pinvalue: 1
pinvalue: 1
pinvalue: 0
pinvalue: 0
pinvalue: 0
pinvalue: 1
pinvalue: 1
pinvalue: 1
pinvalue: 0
pinvalue: 0
pinvalue: 1
```

Die Hardwarelösung würde darin bestehen, einen hochohmigen Widerstand an den Tastenausgang anzuschließen und mit GND zu verbinden.

Softwarekonsolidierung des Interrupts

In der ESP32-Karte können Sie an jedem GPIO-Pin einen Widerstand aktivieren. Diese Aktivierung erfolgt durch das Wort **gpio_pulldown_en**. Dieses Wort akzeptiert als Parameter die GPIO-Pin-Nummer, deren Widerstand aktiviert werden muss. Im Gegenzug gibt dieses Wort 0 zurück, wenn die Aktion erfolgreich war, andernfalls einen Fehlercode:

```
17 Konstanttaste
Button-Eingang pinMode
:test." pinvalue: "
Schaltfläche digitalRead . cr
;
unterbricht
Schaltfläche gpio_pulldown_en ablegen
' Testtaste Pinchange
her
```

Das Ergebnis der Ausführung des Interrupts ist deutlich besser:

```

ok      button digitalRead . cr
ok      ;
ok      interrupts
ok      button gpio_pulldown_en drop
ok      ' test button pinchange
ok      forth
--> pinvalue: 1
pinvalue: 0
pinvalue: 1
pinvalue: 0
pinvalue: 1
pinvalue: 0

```

Bei jeder Zustandsänderung kommt es zu einer Unterbrechung. Auf dem Screenshot oben wird bei jeder Statusänderung **pinvalue: 1** und dann **pinvalue: 0** angezeigt .

Es ist möglich, eine Unterbrechung allein bei der steigenden Flanke zu berücksichtigen. Dies ist möglich durch Angabe von:

```
Schaltfläche GPIO_INTR_POSEDGE gpio_set_intr_type drop
```

Das Wort **gpio_set_intr_type** akzeptiert diese Parameter:

- **GPIO_INTR_ANYEDGE** zur Verwaltung von Interrupts mit steigender oder fallender Flanke
- **GPIO_INTR_NEGEDGE** verarbeitet Interrupts nur bei fallender Flanke
- **GPIO_INTR_POSEDGE** verwaltet Interrupts nur bei steigender Flanke
- **GPIO_INTR_DISABLE** zum Deaktivieren von Interrupts

Vollständiger FORTH-Code mit Erkennung steigender Flanken:

```

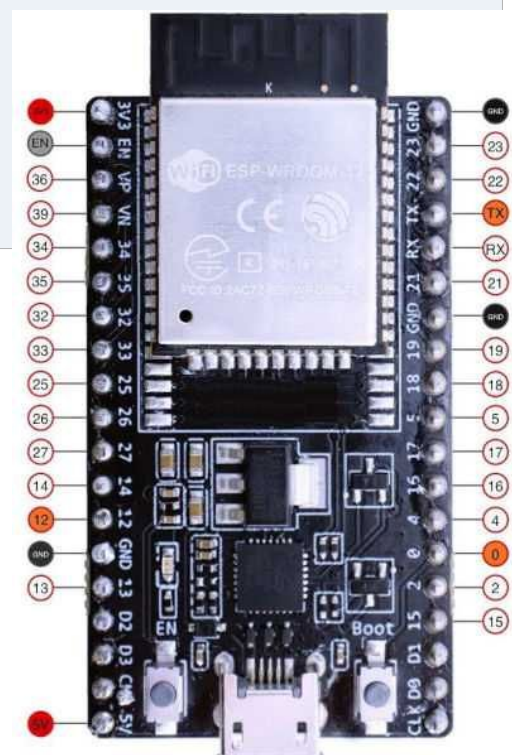
17 constant button
0 constant GPIO_PULLUP_ONLY
button input pinMode
: test ." pinvalue: "
    button digitalRead . cr
;
interrupts
button gpio_pulldown_en drop
button GPIO_INTR_POSEDGE gpio_set_intr_type
drop
' test button pinchange
forth

```

Weitere Informationen

Für ESP32 können alle GPIO-Pins außer GPIO6 bis GPIO11 als Interrupt verwendet werden.

Verwenden Sie keine orange oder roten Stifte. Ihr Programm verhält sich möglicherweise unerwartet,



wenn Sie diese verwenden.

Verwendung des Drehgebers KY-040

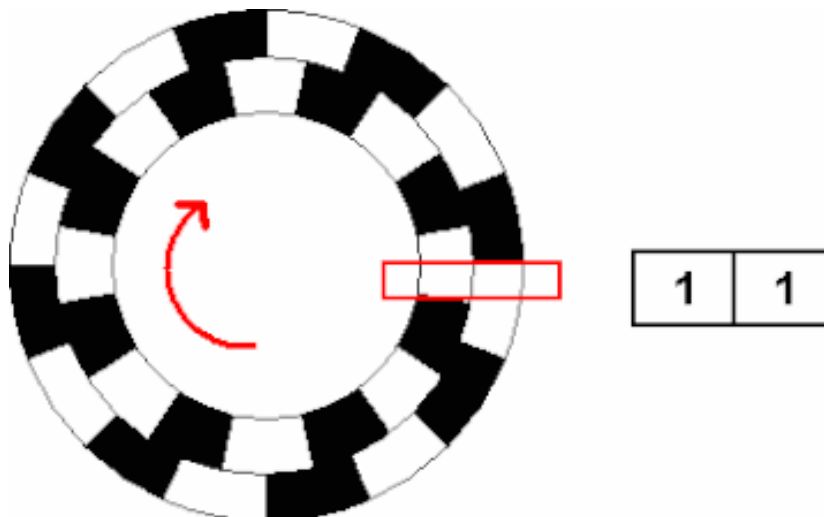
Encoder-Übersicht

Um ein Signal zu variieren, haben wir mehrere Lösungen:

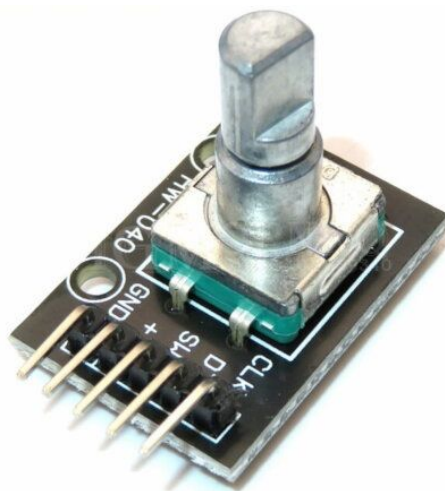
- ein variabler Widerstand in einem Potentiometer
- zwei Tasten zur Verwaltung der Variation per Software
- ein Drehgeber

Der Drehgeber ist eine interessante Lösung. Es kann als Potentiometer verwendet werden, mit dem Vorteil, dass es keinen Start- und Endanschlag hat.

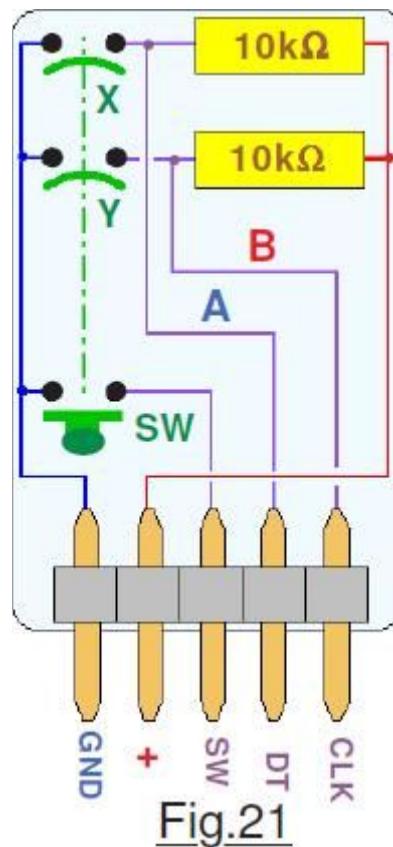
Sein Prinzip ist sehr einfach. Hier sind die Signale, die unser Drehgeber aussendet:



Hier ist unser Encoder:



Internes Funktionsdiagramm:



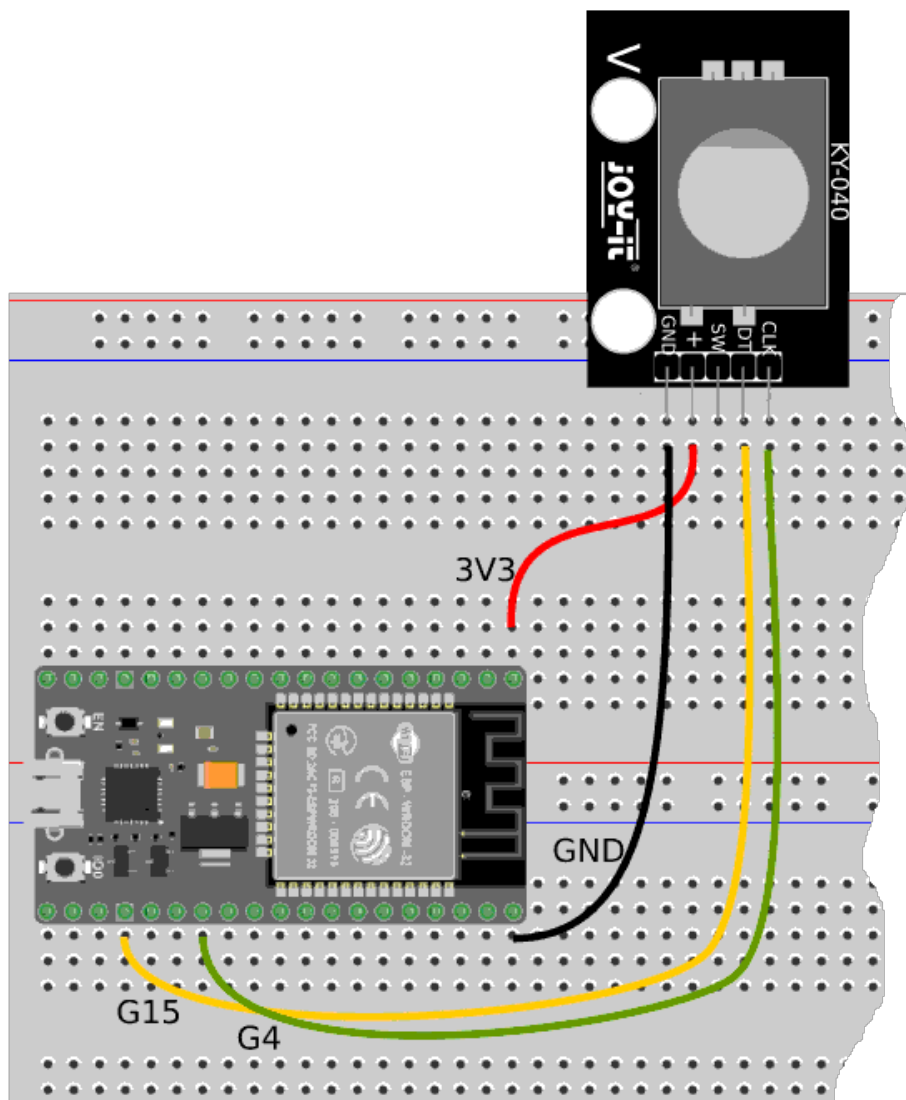
Nach diesem Diagramm interessieren uns zwei Terminals:

- A (DT) -> Schalter X
- B (CLK) -> Schalter Y

Dieser Encoder kann mit 5V oder 3,3V betrieben werden. Das kommt uns entgegen, da die ESP32-Karte über einen 3,3V-Ausgang verfügt.

Montage des Encoders auf dem Steckbrett

Für die Verkabelung unseres Encoders mit der ESP32-Karte sind nur 4 Drähte erforderlich:

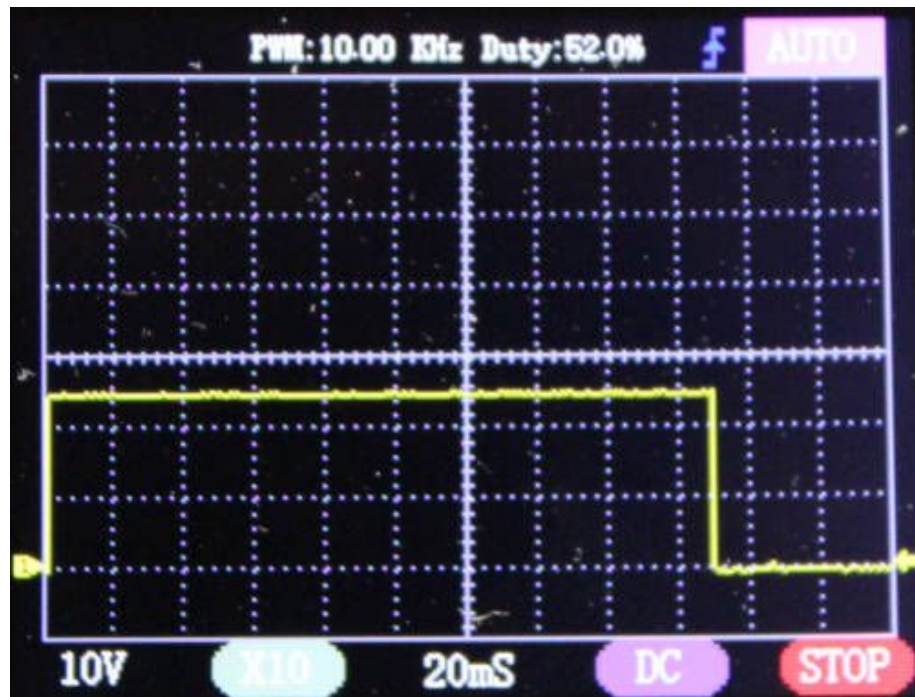


BITTE BEACHTEN : Die Position der Pins G4 und G15 kann je nach Version Ihrer ESP32-Karte variieren.

Analyse von Encodersignalen

Da unser Encoder angeschlossen ist, erhält jeder Anschluss A oder B eine Spannung, hier 3,3V, deren Intensität durch einen Widerstand von 10KOhm begrenzt wird.

Die Analyse des Signals an Klemme G15 zeigt deutlich das Vorhandensein der 3,3-V-



Spannung:

Bei dieser Signalerfassung erscheint der Low-Pegel an Klemme G15 beim Betätigen der Encoder-Steuerstange. Im Leerlauf liegt das Signal an Klemme G15 auf High-Pegel.

Das ändert alles, denn auf Programmiererebene müssen wir den G15-Interrupt als fallende Flanke verarbeiten.

Encoder-Programmierung

Der Encoder wird per Interrupt verwaltet. Interrupts lösen das Programm nur aus, wenn ein bestimmtes Signal einen genau definierten Pegel erreicht.

Wir werden einen einzelnen Interrupt auf dem GPIO G15-Terminal verwalten:

```
interrupts

\ enable interrupt on GPIO G15
: intG15enable ( -- )
  15 GPIO_INTR_POSEDGE gpio_set_intr_type drop
  ;

\ disable interrupt on GPIO G15
: intG15disable ( -- )
  15 GPIO_INTR_DISABLE gpio_set_intr_type drop
  ;

: pinsInit ( -- )
  04 input pinmode          \ G04 as an input
```

```

04 gpio_pulldown_en drop    \ Enable pull-down on GPIO 04
15 input pinmode            \ G15 as an input
15 gpio_pulldown_en drop    \ Enable pull-down on GPIO 15
intG15enable
;

```

Im Wort **pinsInit** initialisieren wir die GPIO-Pins G4 und G15 als Eingang. Dann bestimmen wir den Interrupt-Modus von G15 bei fallender Flanke mit **15 GPIO_INTR_POSEDGE gpio_set_intr_type drop** .

Testen der Kodierung

Dieser Teil des Codes darf nicht in einer Endassembly verwendet werden. Es dient lediglich der Überprüfung, ob der Encoder korrekt angeschlossen ist und ordnungsgemäß funktioniert:

```

: test ( -- )
  cr ." PIN: "
  cr ." - G15: " 15 digitalRead .
  cr ." - G04: " 04 digitalRead .
;

pinsInit    \ initialisiert G4 und G15
' test 15 pinchange

```

Es ist die Sequenz „**Test 15 Pinchange**“, die ESP32Forth anweist, den Testcode auszuführen, wenn durch die Aktion von Terminal G15 ein Interrupt ausgelöst wird.

Ergebnis der Aktion auf unserem Encoder. Wir haben nur die Ergebnisse der am Stopp eintreffenden Aktionen gespeichert, einmal gegen den Uhrzeigersinn, dann im Uhrzeigersinn:

```

PIN:
- G15: 1  \ Rückwärtsdrehung im Uhrzeigersinn
- G04: 1
PIN:
- G15: 0  \ Rechtsdrehung
- G04: 1

```

Erhöhen und dekrementieren Sie eine Variable mit dem Encoder

Nachdem wir den Encoder nun per Hardware-Interrupt getestet haben, können wir den Inhalt einer Variablen verwalten. Dazu definieren wir unsere Variable **KYvar** und die Wörter, mit denen wir ihren Inhalt ändern können:

```

0 value KYvar    \ content is incremented or decremented

\ increment content of KYvar
: incKYvar ( n -- )
  1 +to KYvar
;

\ decrement content of KYvar

```

```

: decKYvar ( n -- )
  -1 +to KYvar
; ;

```

Das Wort **incKYvar** erhöht den Inhalt von **KYvar** . Das Wort **decKYvar** dekrementiert den Inhalt von **KYvar** .

Wir testen die Änderung des Inhalts der Variablen **KYvar** über dieses Wort **testIncDec** , das wie folgt definiert ist:

```

\ wird durch Unterbrechung verwendet, wenn G15 aktiviert ist
: testIncDec ( -- )
  intG15disable
  15 digitalRead if
    04 digitalRead if
      decKYvar
    else
      incKYvar
    then
      cr ." KYvar: " KYvar .
  then
  1000 0 do loop \ small wait loop
  intG15enable
;

pinsInit
' testIncDec 15 pinchange

```

Durch Drehen des Encoder-Reglers nach rechts (im Uhrzeigersinn) wird der Inhalt der Variablen **KYvar** erhöht. Eine Drehung nach links dekrementiert den Inhalt der **KYvar**-Variablen:

```

pinsInit
' testIncDec 15 pinchange
-->
KYvar: 1    \ rotate Clockwise
KYvar: 2
KYvar: 3
KYvar: 4
KYvar: 3    \ rotate Contra Clockwise
KYvar: 2
KYvar: 1
KYvar: 0
KYvar: -1
KYvar: -2

```

Blinken einer LED pro Timer

Erste Schritte mit der FORTH-Programmierung

Jeder Programmieranfänger kennt dieses mehr als klassische Beispiel sehr gut: das Blinken einer LED. Hier ist der Quellcode in C-Sprache für ESP32:

```
/*
 * This ESP32 code is created by esp32io.com
 * This ESP32 code is released in the public domain
 * For more detail (instruction and wiring diagram),
 * visit https://esp32io.com/tutorials/esp32-led-blink
 */

// the code in setup function runs only one time when ESP32 starts
void setup() {
  // initialize digital pin GPIO18 as an output.
  pinMode(18, OUTPUT);
}

// the code in loop function is executed repeatedly infinitely
void loop() {
  digitalWrite(18, HIGH); // turn the LED on
  delay(500);             // wait for 500 milliseconds
  digitalWrite(18, LOW);  // turn the LED off
  delay(500);             // wait for 500 milliseconds
}
```

In der FORTH-Sprache ist es nicht viel anders:

```
18 constant myLED

: led.blink ( -- )
  myLED output pinMode
  begin
    HIGH myLED pin
    500 ms
    LOW myLED pin
    500 ms
  key? until
;
```

Wenn Sie diesen FORTH-Code kompilieren, während ESP32forth auf Ihrem ESP32-Board installiert ist, und am Terminal **led.blink eingeben**, blinkt die mit dem GPIO18-Port verbundene LED.

Um in C-Sprache geschriebenen Code einzuschleusen, muss er auf dem PC kompiliert und dann auf die ESP32-Karte hochgeladen werden, was einige Zeit in Anspruch nimmt. Bei der FORTH-Sprache hingegen ist der Compiler auf unserem ESP32-Board bereits betriebsbereit. Der Compiler kompiliert das in der FORTH-Sprache geschriebene Programm in zwei bis drei Sekunden und ermöglicht seine sofortige Ausführung, indem er einfach das Wort eingibt, das diesen Code enthält, hier **led.blink** für unser Beispiel.

In der FORTH-Sprache können wir Hunderte von Wörtern zusammenstellen und sie sofort einzeln testen, was in der C-Sprache überhaupt nicht möglich ist.

Wir faktorisieren unseren FORTH-Code wie folgt:

```
18 constant myLED

: led.on ( -- )
  HIGH myLED pin
;

: led.off ( -- )
  LOW myLED pin
;

: waiting ( -- )
  500 ms
;

: led.blink ( -- )
  myLED output pinMode
  begin
    led.on      waiting
    led.off     waiting
  key? until
;
```

led.on eingeben, und ausschalten, indem wir **led.off eingeben** . Die Ausführung von **led.blink** bleibt weiterhin möglich.

Das Ziel der Faktorisierung besteht darin, eine komplexe und schwer lesbare Funktion in eine Reihe einfacherer und besser lesbarer Funktionen zu unterteilen. Bei FORTH wird die Faktorisierung empfohlen, um einerseits das Debuggen zu erleichtern und andererseits die Wiederverwendung faktorisierter Wörter zu ermöglichen.

Für diejenigen, die die FORTH-Sprache kennen und beherrschen, mögen diese Erklärungen trivial erscheinen. Dies ist für Leute, die in C programmieren, alles andere als offensichtlich, da sie gezwungen sind, Funktionsaufrufe in der allgemeinen Funktion **loop()** zu gruppieren .

Nachdem dies nun erklärt ist, werden wir alles vergessen! Weil...

Blinken nach TIMER

Wir werden alles vergessen, was zuvor erklärt wurde. Denn dieses LED-Blinkbeispiel hat einen großen Nachteil. Unser Programm macht genau das und nichts anderes. Kurz gesagt, es ist eine echte Verschwendung von Hardware und Software, eine LED auf unserer ESP32-Karte zum Leuchten zu bringen. Wir werden eine ganz andere Art und Weise sehen, dieses Blinken zu erzeugen, ausschließlich in der FORTH-Sprache.

ESP32forth verfügt über zwei Wörter, die sehr nützlich sind, um dieses LED-Blinken zu verwalten: **interval** und **rerun** .

Aber bevor wir diskutieren, wie diese beiden Wörter funktionieren, werfen wir einen Blick auf den Begriff der Unterbrechung ...

Hardware- und Software-Interrupts

Wenn Sie vorhaben, Mikrocontroller zu verwalten, ohne sich Gedanken über Hardware- oder Software-Interrupts machen zu müssen, dann geben Sie die Computerentwicklung für ESP32-Boards auf!

Sie haben das Recht zu starten und keine Unterbrechungen zu erleben. Und wir erklären Ihnen Interrupts und die Verwendung von Timer-Interrupts.

Hier ist ein Nicht-Computer-Beispiel dafür, was ein Interrupt ist:

- Sie erwarten ein wichtiges Paket;
- Sie gehen jede Minute zum Tor Ihres Hauses, um zu sehen, ob der Postbote angekommen ist.

In diesem Szenario verbringen Sie tatsächlich Ihre Zeit damit, nach unten zu gehen, nachzusehen und wieder nach oben zu gehen. Tatsächlich hat man kaum Zeit, etwas anderes zu tun ...

In Wirklichkeit sollte Folgendes passieren:

- du bleibst in deinem Zuhause;
- der Postbote kommt und klingelt;
- Du gehst runter und holst dein Paket ab...

Ein Mikrocontroller, der die ESP32-Karte enthält, verfügt über zwei Arten von Interrupts:

- **Hardware-Interrupts** : Sie werden durch eine physische Aktion an einem der GPIO-Eingänge der ESP32-Karte ausgelöst;
- **Software-Interrupts** : Sie werden ausgelöst, wenn bestimmte Register vordefinierte Werte erreichen.

Dies ist bei Timer-Interrupts der Fall, die wir als Software-Interrupts definieren.

Verwenden Sie die Wörter intervall und rerun.

Das Wort **interval** ist im **timers**- Vokabular definiert . Es akzeptiert drei Parameter:

- **xt** , das den Code für das Wort ausführt, das beim Auslösen des Interrupts ausgelöst werden soll;
- **usec** ist die Wartezeit in Mikrosekunden, bevor der Interrupt ausgelöst wird.
- **t** ist die Nummer des auszulösenden Timers. Dieser Parameter muss im Bereich [0..3] liegen.

Nehmen wir teilweise den faktorisierten Code unseres LED-Blinkens:

```
18 constant myLED

0 value LED_STATE

: led.on ( -- )
  HIGH dup myLED pin
  to LED_STATE
;

: led.off ( -- )
  LOW dup myLED pin
  to LED_STATE
;

timers \ select timers vocabulary
: led.toggle ( -- )
  LED_STATE if
    led.off
  else
    led.on
  then
    0 rerun
;

' led.toggle 500000 0 interval

: led.blink
  myLED output pinMode
  led.toggle
;
```

Dem Wort **rerun** wird die Anzahl der vor der Definition von interval aktivierten Timer vorangestellt . Das Wort **rerun** muss in der Definition des vom Timer ausgeführten Wortes verwendet werden.

Das Wort **led.blink** initialisiert den von der LED verwendeten GPIO-Ausgang und führt dann **led.toggle** aus .

In dieser Sequenz FORTH ' **led.toggle 500000 0 interval** initialisieren wir Timer 0, indem wir den Ausführungscode des Wortes mit **rerun** wiederherstellen, gefolgt vom Zeitintervall, hier 500 Millisekunden, und dann der Nummer des auszulösenden Timers.

Das LED-Blinken beginnt unmittelbar nach der Ausführung des Wortes **led.blink** .

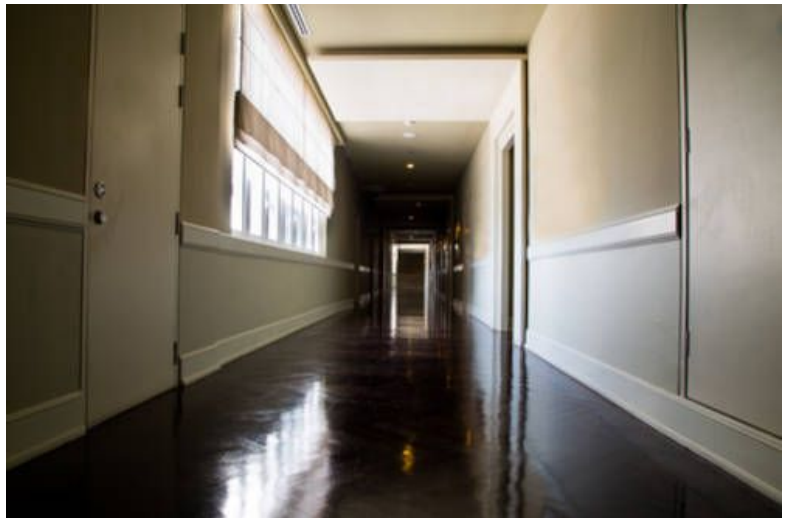
Der FORTH-Interpreter von ESP32forth bleibt zugänglich, während die LED blinkt, was in der C-Sprache unmöglich ist!

Haushälterin-Timer

Präambel

Wir schreiben das Jahr 1990. Er ist ein Computerprogrammierer, der viel arbeitet. Deshalb verlässt er manchmal etwas spät sein Büro.

Und während eines seiner späten Verlassen des Büros betrat er den Korridor, einen dieser Korridore mit einem Timer-Knopf an jedem Ende. Das Licht ist schon an. Doch aus Reflex drückt unser befreundeter Programmierer den Schalter und sticht sich in den Finger. Um den Timer zu blockieren, wird eine Holzspitze in den Schalter eingeführt.



Es ist die Putzfrau, die den Boden reinigt und ihm erklärt: „Ja. Der Timer läuft nur eine Minute der Knopf mit dieser kleinen Holzspitze“...

Eine Lösung

Diese Anekdote löste im Kopf unseres Programmierers eine Idee aus. Da er einige Kenntnisse über Mikrocontroller hatte, machte er sich daran, eine Lösung für die Putzfrau zu finden.

Die Geschichte sagt nicht, in welcher Sprache er seine Lösung programmierte. Sicherlich im Assembler.

Die Steuerung der Lichter leitete er von seiner Schaltung ab:

- Durch einfaches Drücken wird der Timer für eine Minute gestartet.
- Wenn das Licht eingeschaltet ist, verkürzt ein kurzer Tastendruck die Zündverzögerung auf eine Minute.
- Das Geheimnis unseres Programmierers besteht darin, einen langen Druck von 3 Sekunden oder mehr geplant zu haben. Durch langes Drücken wird der Timer für 10 Minuten Beleuchtung gestartet;

- Wenn sich der Timer im Langlauf befindet, verringert ein weiterer langer Druck die Timer-Verzögerung auf eine Minute.
- Ein kurzer Piepton bestätigt die Aktivierung oder Deaktivierung eines langen Timerzyklus.

Die Putzfrau schätzte diese Verbesserung des Timers sehr. Sie musste den Knopf in keiner Weise mehr blockieren.

Was ist mit den anderen Arbeitern? Da diese Funktion niemandem bekannt war, nutzten sie den Timer weiter, indem sie kurz den Aktivierungsschalter drückten.

Ein FORTH-Timer für ESP32Forth

Sie verstehen, wir werden **Timer verwenden**, um einen Timer zu verwalten, indem wir das zuvor beschriebene Szenario integrieren.

```
\ myLIGHTS mit GPIO18 verbunden
18 constant myLIGHTS

\ definiert maximale Zeit für den normalen oder
\ erweiterten Zyklus in Sekunden
60 constant MAX_LIGHT_TIME_NORMAL_CYCLE
600 constant MAX_LIGHT_TIME_EXXTENDED_CYCLE

\ maximale Zeit für normalen oder verlängerten Zyklus, in Sekunden
0 value MAX_LIGHT_TIME

timers
\ Beleuchtung wird ausgeschaltet, wenn MAX_LIGHT_TIME gleich 0 ist
: cycle.stop ( -- )
  -1 +to MAX_LIGHT_TIME          \ décrémente temps max de 1 seconde
  MAX_LIGHT_TIME 0 = if
    LOW myLIGHTS pin            \ coupe éclairage
  else
    0 rerun
  then
;

\ Timer 0 initialisieren
' cycle.stop 1000000 0 interval

\ startet einen Beleuchtungszyklus, n ist die Verzögerung in Sekunden
: cycle.start ( n -- )
  1+ to MAX_LIGHT_TIME          \ sélect. Temps max
  myLIGHTS output pinMode
  HIGH myLIGHTS pin            \ active éclairage
  0 rerun
;
```

Wir können unseren Timer bereits testen:

```
3 cycle.start \ schaltet die Lichter für 3 Sekunden ein
10 cycle.start \ schaltet die Lichter für 10 Sekunden ein
```

Wenn wir **Cycle.start neu starten** , während das Licht an ist, starten wir erneut für einen neuen Beleuchtungszyklus von n Sekunden.

Daher müssen wir die Aktivierung dieser Zyklen noch über einen Schalter verwalten.

Verwaltung der Licht-Ein-Taste

Das ist keine Raketenwissenschaft. Wir werden einen Druckknopf verwalten. Da wir über eine ESP32-Karte verfügen, die mit ESP32Forth programmierbar ist, werden wir diese nutzen, um diese Taste durch Interrupts zu verwalten. Die Interrupts, die die GPIO-Terminals auf der ESP32-Karte verwalten, sind Hardware-Interrupts.

Unser Taster ist am GPIO17 (G17)-Terminal montiert.

Wir definieren zwei Wörter, **intPosEdge** und **intNegEdge** , die die Art der Auslösung des Interrupts bestimmen:

- **intPosEdge** , um den Interrupt bei steigender Flanke auszulösen;
- **intNegEdge** , um den Interrupt bei fallender Flanke auszulösen.

```
17 constant button \ Mount-Taste auf GPIO17

Unterbrechungen\Interrupt-Vokabular auswählen

\Interrupt zum Auslösen des Signals aktiviert
: intPosEdge ( -- )
  button #GPIO_INTR_POSEDGE gpio_set_intr_type drop
;

\interrupt für Falldown-Signal aktiviert
: intNegEdge ( -- )
  button #GPIO_INTR_NEGEDGE gpio_set_intr_type drop
;
```

Anschließend müssen wir einige Variablen und Konstanten definieren:

- zwei Konstanten, **CYCLE_SHORT** und **CYCLE_LONG** , die verwendet werden, um die Dauer des Leuchtens der Lichter zu definieren. Hier haben wir für unsere Tests 3 und 10 Sekunden gewählt.
- die **msTicksPositiveEdge**- Variable , die die Position des von ms-ticks gelieferten Wartezählers speichert
- Konstante **DELAY_LIMIT** , die den Schwellenwert für die Bestimmung eines kurzen oder langen Tastendrucks bestimmt. Hier sind es 3000 Millisekunden oder 3 Sekunden. Ein normaler Benutzer wird die Licht-Ein-Taste NIEMALS 3 Sekunden lang drücken. Nur die Putzfrau kennt das Manöver, eine lange Dauerbeleuchtung zu haben...

```

03 constant CYCLE_SHORT      \ Leuchtdauer für kurzes Drücken, in Sekunden
10 constant CYCLE_LONG       \ Leuchtdauer für langes Drücken

\ speichert den Wert der MS-Ticks bei steigender Flanke
Variable msTicksPositiveEdge

\ Fristbegrenzung: wenn Verzögerung < DELAY_LIMIT, kurzer Zyklus
3000 constant DELAY_LIMIT

```

Das Wort **getButton** wird bei jedem Interrupt gestartet, der durch Drücken des an GPIO17 (G17) angeschlossenen Druckknopfs auf unserem ESP32-Board ausgelöst wird.

getButton- Ausführung sind Interrupts auf G17 deaktiviert. Diese Unterbrechung wird am Ende der Definition wieder aktiviert. Diese Deaktivierung ist notwendig, um Interrupt-Stacking zu verhindern.

Auf die Deaktivierung folgt die **70000 0 do loop**. Diese Schleife wird zur Verwaltung von Kontakt-Bounces verwendet. Hier verwalten wir die Entprellung per Software.

```

\ Wort wird durch Interrupt ausgeführt
: getButton ( -- )
  button gpio_intr_disable drop
  70000 0 do loop \ anti rebound
  button digitalRead 1 =
  if
    ms-ticks msTicksPositiveEdge !
    intNegEdge
  else
    intPosEdge
    ms-ticks msTicksPositiveEdge @ -
    DELAY_LIMIT >
    if CYCLE_LONG cr ." BEEP"
    else CYCLE_SHORT cr ." ----"
  then
  cycle.start
  button gpio_intr_enable drop
;

```

Bei der steigenden Flanke zeichnet das Wort **getButton** den Zustand des Verzögerungszählers auf und positioniert die Interrupts bei der fallenden Flanke. Dann verlassen wir dieses Wort, indem wir die Unterbrechungen reaktivieren.

Bei der fallenden Flanke berechnet das Wort **getButton die seit der steigenden Flanke verstrichene Zeit**. Wenn diese Verzögerung größer als **DELAY_LIMIT ist**, wird ein langer Zündzyklus eingeleitet. Andernfalls wird ein kurzer Zündzyklus eingeleitet.

Der Beginn eines langen Zündzyklus wird durch die Anzeige von „BEEP“ am Terminal angezeigt.

Im Originalszenario wird dies durch einen kurzen Piepton signalisiert.

Abschließend initialisieren wir den Button und den Hardware-Interrupt auf diesem Button:

```
\ Schaltflächen- und Interrupt-Vektoren initialisieren
button input pinMode          \ wählt G17 im Eingabemodus aus
button gpio_pulldown_en drop  \ aktiviert den Innenwiderstand von G17
' getButton button pinchange
intPosEdge

forth
```

Abschluss

Sehen Sie sich das Montagevideo an: https://www.youtube.com/watch?v=OHWMh_bIWz0

Diese sehr einfache Fallstudie zeigt, wie man den Timer und einen Hardware-Interrupt gleichzeitig verwaltet.

Diese beiden Mechanismen haben nur eine sehr geringe präventive Wirkung. Der Timer lässt den Zugriff auf den FORTH-Interpreter frei. Der Hardware-Interrupt ist auch dann betriebsbereit, wenn FORTH einen anderen Prozess ausführt.

Wir machen kein Multitasking. Es ist wichtig, es zu sagen!

Ich hoffe nur, dass dieser Lehrbuchkoffer Ihnen nun viele Anregungen für Ihre Entwicklungen gibt...

Software-Echtzeituhr

Das Wort MS-TICKS

Das Wort **MS-TICKS** wird in der Definition des Wortes **ms verwendet** :

```
DEFINED? ms-ticks [IF]
: ms ( n -- )
  ms-ticks >r
  begin
    pause ms-ticks r@ - over
  >= until
  rdrop drop
;
[THEN]
```

Dieses Wort **MS-TICKS** steht im Mittelpunkt unserer Untersuchungen. Wenn wir die ESP32-Karte starten, stellt ihre Ausführung die Anzahl der Millisekunden wieder her, die seit dem Start der ESP32-Karte vergangen sind. Dieser Wert wächst immer noch. Der Sättigungswert dieser Zählung beträgt $2^{32}-1$ oder 4294967295 Millisekunden oder ungefähr 49 Tage ...

Bei jedem Neustart der ESP32-Karte beginnt dieser Wert wieder bei Null.

Verwalten einer Softwareuhr

Aus den **HH MM SS**- Daten (Stunden, Minuten, Sekunden) lässt sich leicht ein ganzzahliger Wert in Millisekunden rekonstruieren, der der seit 00:00:00 verstrichenen Zeit entspricht. Wenn wir von dieser Zeit den Wert von **MS-TICKS subtrahieren** , erhalten wir einen Startzeitwert zur Bestimmung der tatsächlichen Zeit . Wir initialisieren daher einen Basiszähler **currentTime** aus dem Wort **RTC.set-time** :

```
0 value currentTime

\ store current time
: RTC.set-time { hh mm ss -- }
  hh 3600 *
  mm 60 *
  ss + + 1000 *
  MS-TICKS - to currentTime
;
```

Initialisierungsbeispiel: **22 52 00 RTC.set-time** initialisiert die Zeitbasis für 22:52:00...

Zur ordnungsgemäßen Initialisierung bereiten Sie die drei Werte **HH MM SS** gefolgt vom Wort **RTC.set-time vor** und achten Sie auf Ihre Uhr. Wenn die erwartete Zeit erreicht ist, führen Sie die Initialisierungssequenz aus.

Die umgekehrte Operation stellt die **HH MM-** und **SS -Werte** der aktuellen Zeit unter Verwendung dieses Wortes wieder her:

```
\ aktuelle Zeit in Sekunden abrufen
: RTC.get-time ( -- hh mm ss )
  currentTime MS-TICKS + 1000 /
  3600 /mod swap 60 /mod swap
;
```

Abschließend definieren wir das Wort **RTC.display-time** , mit dem Sie nach der Initialisierung unserer Softwareuhr die aktuelle Uhrzeit anzeigen können:

```
\ wird für die SS- und MM-Teilzeitanzeige verwendet
: :## ( n -- n' )
  # 6 base ! # decimal [char] : hold
;

\ zeigt die aktuelle Uhrzeit an
: RTC.Anzeigezeit (--)
: RTC.display-time ( -- )
  currentTime MS-TICKS + 1000 /
  <# :## :## 24 MOD #S #> type
;
```

Der nächste Schritt wäre die Verbindung zu einem Zeitserver mit dem NTP-Protokoll, um unsere Software-Uhr automatisch zu initialisieren.

Messen der Ausführungszeit eines FORTH-Wortes

Messung der Leistung von FORTH-Definitionen

Beginnen wir mit der Definition des Wortes „ **measure:**“ , das diese Ausführungszeitmessungen durchführt:

```
: measure: ( exec: -- <word> )
  ms-ticks >r
  ' execute
  ms-ticks r> -
  cr ." execution time: "
  <# # # # [char] . hold #s #> type ." sec." cr
;
```

In diesem Wort rufen wir die Zeit durch **ms-ticks ab** , dann rufen wir den Ausführungscode des Wortes ab, das auf „ **measure:**“ folgt: Wenn wir dieses Wort ausführen, rufen wir den neuen Zeitwert durch **ms-ticks ab** . Wir machen den Unterschied, der der verstrichenen Zeit in Millisekunden entspricht, die das Wort zur Ausführung benötigt. Beispiel :

```
measure: words
\ zeigt an: execution time: 0.210sec.
```

Das Wort **words** wurde in 0,2 Sekunden ausgeführt. In dieser Zeit sind Übertragungsverzögerungen durch das Endgerät nicht berücksichtigt. Diese Zeit berücksichtigt auch nicht die Verzögerung, die durch **measure:** um den Ausführungscode des zu messenden Wortes abzurufen.

measure: gestapelt werden, gefolgt von dem zu messenden Wort:

```
: SQUARE ( n -- n-exp2 )
  dup *
;
3 measure: SQUARE
\ Zeigt an:
\ execution time: 0.000sec.
```

Dieses Ergebnis bedeutet, dass unsere **SQUARE**- Definition in weniger als einer Millisekunde ausgeführt wird.

Wir werden diesen Vorgang einige Male wiederholen:

```
: test-square ( -- )
  1000 for
    3 SQUARE drop
  next
;
3 measure: test-square
\ Zeigt an:
\ execution time: 0.001sec.
```

Indem wir das Wort **SQUARE** 1000 Mal ausführen , gefolgt von einem Stapeln von Werten und einem Entstapeln des Ergebnisses, kommen wir zu einer Ausführungszeit von 1 Millisekunde. Wir können vernünftigerweise folgern, dass **SQUARE** in weniger als einer Mikrosekunde ausgeführt wird!

Ein paar Schleifen testen

Wir werden einige Schleifen mit 1 Million Iterationen testen. Beginnen wir mit einer **do-loop** :

```
: test-loop ( -- )
  1000000 0 do
    loop
  ;
measure: test-loop
\ Anzeige:
\ execution time: 1.327sec.
```

Sehen wir uns nun mit einer **for-next**- Schleife an :

```
: test-for ( -- )
  1000000 for
    next
;
```



```
measure: test-for
\ Anzeigt:
\ execution time: 0.096sec.
```

Die **for-next**- Schleife läuft fast 14-mal schneller als die **do-loop** .

Sehen wir uns an, was eine **Begin-Until**- Schleife zu bieten hat:

```
: test-begin ( -- )
  1000000 begin
    1- dup 0=
    until
  ;
measure: test-begin
\ Anzeigt:
\ execution time: 0.273sec.
```

Dies ist effizienter als die **do-loop** , aber immer noch dreimal langsamer als die **for-next**- Schleife.

Jetzt sind Sie in der Lage, noch effizientere FORTH-Programme zu erstellen.

Installieren der OLED-Bibliothek für SSD1306

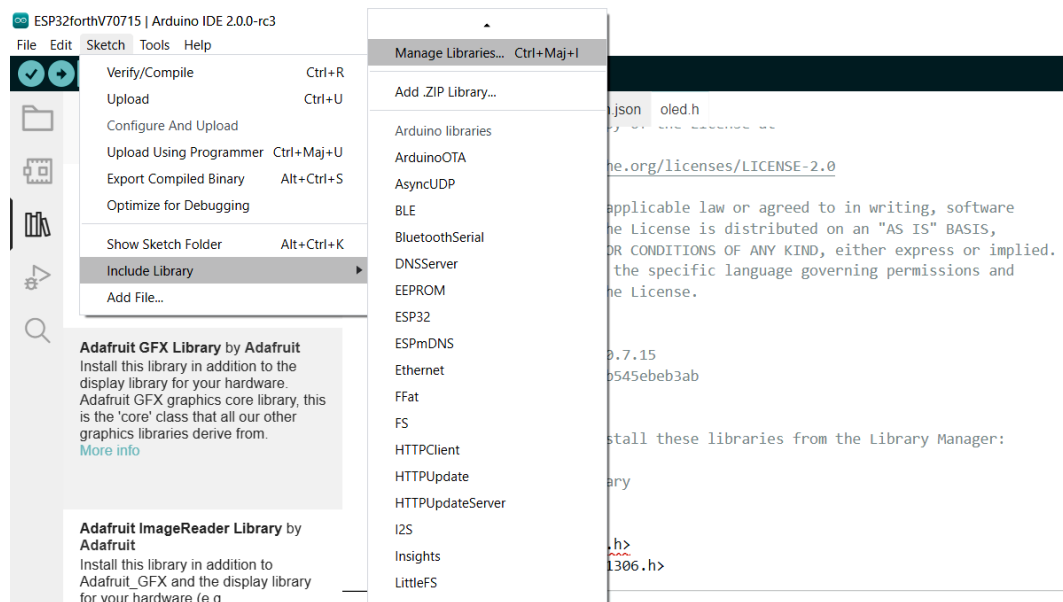
Seit ESP32forth Version 7.0.7.15 sind die Optionen im **optional** Ordner verfügbar:

Téléchargements > ESP32forth-7.0.7.15(1).zip > ESP32forth > optional		
	Nom	Type
✦	assemblers.h	Fichier H
✦	camera.h	Fichier H
✦	interrupts.h	Fichier H
✦	oled.h	Fichier H
✦	README-optional.txt	Document texte
	rmt.h	Fichier H
	serial-bluetooth.h	Fichier H
	spi-flash.h	Fichier H

Um das **oled** Vokabular zu erhalten, kopieren Sie die Datei **oled.h** in den Ordner, der die Datei **ESP32forth.ino** enthält.

Starten Sie dann ARDUINO IDE und wählen Sie die neueste **ESP32forth.ino**-Datei aus.

Wenn die OLED-Bibliothek nicht installiert wurde, klicken Sie in der ARDUINO IDE auf *Sketch* und wählen Sie *Include Library* und dann *Manage Libraries*.



Suchen Sie in der linken Seitenleiste nach der Bibliothek **Adafruit SSD1306 by Adafruit**.

Sie können nun den Sketch compilieren und uploaden. Klicken Sie dazu auf *Sketch* und dann *Upload*.

Wenn der Sketch auf das ESP32-Board hochgeladen ist, starten Sie das TeraTerm-Terminal. Überprüfen Sie, ob das **oled**-Vokabular vorhanden ist :

```
oled vlist \ display:
OledInit SSD1306_SWITCHCAPVCC SSD1306_EXTERNALVCC WHITE BLACK OledReset
HEIGHT WIDTH OledAddr OledNew OledDelete OledBegin OledHOME OledCLS OledTextc
OledPrintln OledNumln OledNum OledDisplay OledPrint OledInvert OledTextsize
OledSetCursor OledPixel OledDrawL OledCirc OledCircF OledRect OledRectF
OledRectR OledRectRF oled-builtins
```

Installieren des HTTP-Clients

Bearbeiten der Datei ESP32forth.ino

ESP32Forth wird als Quelldatei bereitgestellt, geschrieben in der Sprache C. Diese Datei muss mit ARDUINO IDE oder einem anderen C-Compiler kompiliert werden, der mit der ARDUINO-Entwicklungsumgebung kompatibel ist.

Hier sind die Teile des Codes, die geändert werden müssen. Erster zu ändernder Teil:

```
#define ENABLE_SD_SUPPORT
#define ENABLE_SPI_FLASH_SUPPORT
#define ENABLE_HTTP_SUPPORT
// #define ENABLE_HTTPS_SUPPORT
```

Zweiter zu ändernder Teil:

```
// .....
#define VOCABULARY_LIST \
  V(forth) V(internals) \
  V(rtos) V(SPIFFS) V(serial) V(SD) V(SD_MMC) V(ESP) \
  V(ledc) V(http) V(Wire) V(WiFi) V(bluetooth) V(sockets) V(oled) \
  V(rmt) V(interrupts) V(spi_flash) V(camera) V(timers)
```

Dritter zu ändernder Teil:

```
OPTIONAL_RMT_SUPPORT \
OPTIONAL_OLED_SUPPORT \
OPTIONAL_SPI_FLASH_SUPPORT \
OPTIONAL_HTTP_SUPPORT \
FLOATING_POINT_LIST

#ifndef ENABLE_HTTP_SUPPORT
# define OPTIONAL_HTTP_SUPPORT
#else

# include <HTTPClient.h>
  HTTPClient http;

# define OPTIONAL_HTTP_SUPPORT \
  XV(http, "HTTP.begin", HTTP_BEGIN, tos = http.begin(c0)) \
  XV(http, "HTTP.doGet", HTTP_DOGET, PUSH http.GET()) \
  XV(http, "HTTP.getPayload", HTTP_GETPL, String s = http.getString(); \
    memcpy((void *) n1, (void *) s.c_str(), n0); DROPn(2)) \
  XV(http, "HTTP.end", HTTP_END, http.end())
#endif
```

Vierter zu ändernder Teil:

```
vocabulary ledc ledc definitions
```

```

transfer ledc-builtins
forth definitions

vocabulary http  http definitions
transfer http-builtins
forth definitions

vocabulary Serial  Serial definitions
transfer Serial-builtins
forth definitions

```

Sobald die Datei **ESP32forth.ino** geändert wurde, kompilieren Sie sie und laden sie auf das ESP32-Board hoch. Wenn alles richtig gelaufen ist, sollten Sie über ein neues **http**-Vokabular verfügen :

```

http
vlist    \ displays :
HTTP.begin HTTP.doGet HTTP.getPayload HTTP.end http-builtins

```

HTTP-Client-Tests

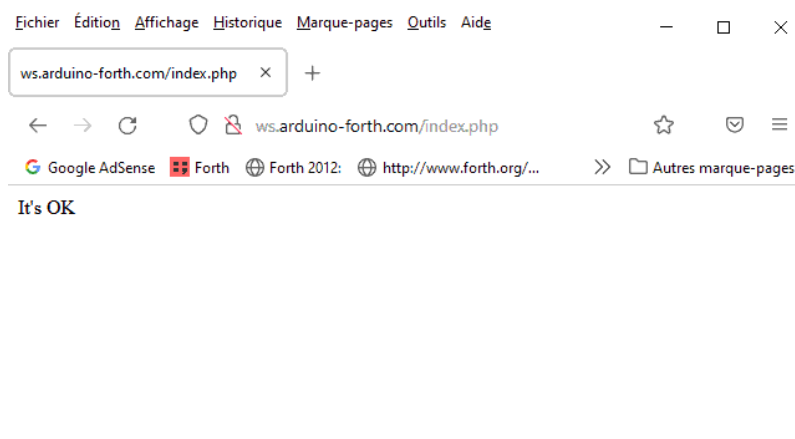
Um unseren HTTP-Client zu testen, können wir dies tun, indem wir einen beliebigen Webserver abfragen. Aber für das, was wir später betrachten, benötigen Sie einen persönlichen Webserver. Auf diesem Server erstellen wir eine Subdomain:

- Unser Server ist arduino-forth.com
- **ws**- Subdomain
- Wir greifen auf diese Subdomain mit der URL <http://ws.arduino-forth.com> zu

Da diese Subdomain erstellt wird, enthält sie kein auszuführendes Skript. Wir erstellen die Seite **index.php** und fügen dort diesen Code ein:

```
It's OK
```

Um zu überprüfen, ob unsere Subdomain funktioniert, fragen Sie sie einfach in unserem bevorzugten Webbrowser ab:



Wenn alles wie geplant verläuft, sollte in unserem bevorzugten Webbrowser der Text **It's OK** angezeigt werden. Sehen wir uns nun an, wie man dieselbe Serverabfrage von ESP32Forth aus durchführt ...

Hier ist der FORTH-Code, der schnell geschrieben wurde, um den HTTP-Client-Test durchzuführen:

```
WiFi

\ connection to local WiFi LAN
: myWiFiConnect
  z" mySSID"
  z" myWiFiCode"
  login
;

Forth

create httpBuffer 700 allot
  httpBuffer 700 erase

HTTP

: run
  cr
  z" http://ws.arduino-forth.com/" HTTP.begin
  if
    HTTP.doGet dup ." Get results: " . cr 0 >
    if
      httpBuffer 700 HTTP.getPayload
      httpBuffer z>s dup . cr type
    then
  then
  HTTP.end
;
```

Wir aktivieren die WLAN-Verbindung, indem wir **myWiFiConnect** ausführen und dann **Folgendes** ausführen :

```
--> myWiFiConnect
192.168.1.23
MDNS started
ok
--> run

Get results: 200
8
It's OK
ok
```

Unser HTTP-Client hat den Webserver perfekt abgefragt und den gleichen Text angezeigt, den er von unserem Webbrowser abgerufen hat.

Dieser kleine erfolgreiche Test eröffnet den Weg zu enormen Möglichkeiten.

Rufen Sie die Uhrzeit von einem WEB-Server ab

Kapitel *Software-Echtzeituhr* haben wir untersucht, wie man eine Echtzeituhr mithilfe der Eigenschaften des **timers** verwaltet .

Die Initialisierung dieser Echtzeituhr muss jedoch manuell erfolgen. Nachdem wir nun die Möglichkeit haben, mit einem Webserver zu kommunizieren, werden wir sehen, wie diese Initialisierung über einen Webserver durchgeführt wird.

Senden und Empfangen der Uhrzeit von einem Webserver

Für den Serverteil erstellen wir ein neues **gettime.php**- Skript , dessen Inhalt wie folgt lautet:

```
<?php
echo date('H i s')." RTC.set-time";
```

Wenn wir dieses Skript <http://ws.arduino-forth.com/gettime.php> in einem Webbrowser ausführen, wird Folgendes angezeigt:

```
15 25 30 RTC.set-time
```

Wir haben die Arbeit so vorbereitet, dass der ESP32Forth-Interpreter nur diese Zeile ausführen muss. Hier ist der FORTH-Code zum Abrufen der Uhrzeit:

```
WiFi
\ connection to local WiFi LAN
: myWiFiConnect
  z" mySSID"
  z" myWiFiCode"
  login
;

Forth

0 value currentTime

\ store current time
: RTC.set-time { hh mm ss -- }
  hh 3600 *
  mm 60 *
  ss + + 1000 *
  MS-TICKS - to currentTime
;

\ used for SS and MM part of time display
: :## ( n -- n' )
  # 6 base ! # decimal [char] : hold
;

\ display current time
: RTC.display-time ( -- )
  currentTime MS-TICKS + 1000 /
```



```

    <# :## :## 24 mod #S #> type
;

700 constant bufferSize
create httpBuffer
    bufferSize allot

0 buffer 700 erase

HTTP

: getTime
    cr
    z" http://ws.arduino-forth.com/gettime.php" HTTP.begin
    if
        HTTP.doGet
        if
            httpBuffer bufferSize HTTP.getPayload
            httpBuffer z>s evaluate
        then
    then
    HTTP.end
;

myWiFiConnect
getTime
RTC.display-time

```

Im Wort **getTime** ruft diese Sequenz **httpBuffer z>s equal** den Inhalt des Web-Transaktionspuffers ab und wertet seinen Inhalt aus. Dies ist möglich, weil der Webserver eine mit unserem FORTH-Interpreter kompatible Sequenz übermittelt hat. Wenn Sie die letzten drei Zeilen dieses Codes ausführen, wird Folgendes angezeigt:

```

--> myWiFiConnect
192.168.1.23
MDNS started
ok
--> getTime
ok
--> RTC.display-time
15:33:09 ok

```

Diese Initialisierung kann nur einmal durchgeführt werden, im Allgemeinen beim Starten von ESP32Forth. Diese Technik der Abfrage unseres eigenen Webserver vermeidet die Verhandlung mit einem Zeitserver.

Die meisten Zeitserver liefern Informationen in Formaten, die von FORTH nur schwer verarbeitet werden können: CSV, JSON, XML usw.

Verständnis der Übertragung per GET an einen WEB-Server

Übertragung von Daten an einen Server per GET

Es gibt zwei Methoden, Daten von einer Webseite an einen Webserver zu übertragen:

- **POST** ist die Methode, die im Allgemeinen für Formulare verwendet wird
- **GET**, das ist die Methode, die wir untersuchen werden

Es gibt andere Methoden, diese sind jedoch im Allgemeinen Maschine-zu-Maschine-Transaktionen über Webdienste vorbehalten.

Parameter in einer URL

Beginnen wir damit, zu erklären, was eine URL ist: <http://my-website.com/> (z. B. URL).

Wir analysieren eine URL beginnend am Ende:

- **.com** ist die TLD (Top-Level-Domain)
- **my-website** ist der Domainname
- **http://** ist das Kommunikationsprotokoll.

Wir werden keinen erschöpfenden Kurs zu diesen Elementen durchführen. Das Einzige, was es zu wissen gibt, kommt jetzt.

Auf diese URL kann das Skript oder die HTML-Seite folgen, Beispiel: <http://my-website.com/index.php>

Wir können diese URL mit einer Parameterübergabe vervollständigen:

```
http://my-website.com/index.php?temp=32.7
```

Hier übergeben wir einen **temporären Parameter** mit dem Wert **32,7**.

Die Übergabe von Parametern mit der GET-Methode ist durch das **?**-Zeichen gekennzeichnet.

Übergabe mehrerer Parameter

Mehrere Parameter können übertragen werden, indem man sie durch das Zeichen **&** trennt :

```
http://my-website.com/index.php?log=myLog&pwd=myPassWd&temp=32.7
```

Hier übergeben wir drei Parameter:

- **log** mit dem myLog-Wert
- **pwd** mit dem Wert myPassWd
- **temp** mit dem Wert 32,7

Um zu verstehen, wie der Server diese Daten erhält, erstellen wir ein **record.php- Skript** , das vorläufig einfach Folgendes enthält:

```
<?php  
var_dump($_GET);
```

und was dies anzeigt, wenn wir dieses Skript mit unserem bevorzugten Webbrowser abfragen:

```
array(3) {  
  ["log"]=>  
  string(7) "myLogin"  
  ["pwd"]=>  
  string(10) "mypassword"  
  ["temp"]=>  
  string(4) "32.7"  
}
```

Das ist so ziemlich alles, was wir brauchen, um die Daten abzurufen und auf dem Server zu speichern. Das werden wir entdecken...

Verwalten der Parameterübergabe mit ESP32forth

Zunächst ist es notwendig, Wörter zur Verwaltung von Zeichenfolgen zu haben. Sie finden diese Wörter im *Kapitel Darstellung von Zahlen und Zeichenfolgen*, Teil *Code von Wörtern zur Verwaltung von Textvariablen* .

Wir beginnen mit der Erstellung einer Zeichenfolge:

```
256 string myUrl  
s" http://ws.arduino-forth.com/record.php?log=myLog&pwd=myPassWd&temp=" myUrl $!
```

myUrl definiert . Diese Variable ist fast vollständig. Es fehlt lediglich der Wert des **temp**-Parameters . Um diesen Wert hinzuzufügen, führen wir **append\$** aus :

```
s" 32.5" myUrl append$  
myUrl type  
\ display: http://ws.arduino-forth.com/record.php?  
log=myLog&pwd=myPassWd&temp=32.5
```

Dies ist die URL, die wir in dieser Definition verwenden werden:

```

: sendData ( str -- )
  s" http://ws.arduino-forth.com/record.php?log=myLog&pwd=myPassWd&temp=
  myUrl $!
  myUrl append$
  \ cr myUrl type
  myUrl s>z HTTP.begin
  if
    HTTP.doGet dup 200 =
    if drop
      httpBuffer bufferSize HTTP.getPayload
      httpBuffer z>s type
    else
      cr ." CNX ERR: " .
    then
  then
  HTTP.end
;

myWiFiConnect
s" 32.65" sendData

```

Das Wort **sendData** ruft den Inhalt der Zeichenfolge ab, hier **32.65** , verkettet diesen Inhalt mit **myUrl** und initiiert dann eine Web-Client-Transaktion mit dem in **myUrl** genannten Server .

Sie werden feststellen, dass in der URL ein Parameter **log** vorhanden ist. Dieser Parameter kann für jede ESP32-Karte, die eine Transaktion zum Webserver initiiert, unterschiedlich sein. Es ist möglich, dass zehn, zwanzig oder sogar tausend ESP32-Karten ihre Daten auf einem einzigen Webserver speichern.

Datenübertragung an einen WEB-Server

Datenaufzeichnung auf der Webserverseite

Im vorherigen Kapitel „Übertragung per GET an einen WEB-Server verstehen“ haben wir erklärt, wie ESP32Forth Informationen an einen Webserver überträgt.

Sehen wir uns nun an, wie wir die Daten serverseitig speichern. Hier ist ein erstes Skript in PHP, das diese Aufzeichnung durchführt:

```
<?php
// echo "<pre>"; var_dump($_GET);
$handle = fopen("datasRecords.csv","a");
$myDatas = array(
    'currentDateTime' => date("Y-m-d H:i:s"),
    'currentLogin'     => $_GET['log'],
    'currentTemp'      => $_GET['temp'],
);
fwrite($handle, implode(';', $myDatas)."\n");
fclose($handle);
echo "DATAs recorded";
```

Dieses Skript ist sehr einfach:

- Wir öffnen eine **dataRecords.csv -Datei** mit **fopen** .
- Wir bereiten die Daten für die Speicherung in einer **myDatas**- Tabelle vor
- Diese Daten speichern wir mit **fwrite**
- **implode** in das CSV-Format übertragen
- Wir schließen die Datei mit **fclose**

Datei **im CSV**- Format lässt sich leicht mit einer Tabellenkalkulation abrufen oder mit einem einfachen Texteditor lesen.

Zugangsschutz

Wenn Sie unsere Erläuterungen aufmerksam verfolgt haben, werden Sie feststellen, dass wir zwei Parameter **log** und **pwd** übermitteln . Diese beiden Parameter dienen zunächst als Zugangsschlüssel zu unserem Datenaufzeichnungsskript.

Diesen Schutz haben wir eingerichtet, um den Zugriff auf das Skript durch einen unbefugten Sender zu verhindern. Hier akzeptieren wir zwei Sender :

```
<?php
```

```
// echo "<pre>"; var_dump($_GET);
$myAuths = array(
    'pooltemp' => 'pool2022',
    'housetemp' => 'house2022',
);

/**
 * Test authorization access
 * @param array $auths
 * @return boolean
 */
function testAuths($auths){
    if(array_key_exists($_GET['log'], $auths) &&
    $auths[$_GET['log']]==$_GET['pwd']) {
        return true;
    }
    return false;
}

// Recording datas in CSV file format
if (testAuths($myAuths)) {
    $handle = fopen("datasRecords.csv","a");
    $myDatas = array(
        'currentDateTime' => date("Y-m-d H:i:s"),
        'currentLogin'     => $_GET['log'],
        'currentTemp'      => $_GET['temp'],
    );
    fwrite($handle, implode(';', $myDatas)."\n");
    fclose($handle);
    echo "DATAS recorded";
} else {
    echo "AUTH failed";
}
```

Dieses Skript dient als Beispiel. Es ist bewusst einfach gehalten. Bei einer professionellen Anwendung würden die Schlüssel und Passwörter in der Datenbank gespeichert.

Hier ist eine Transaktion, die erfolgreich ausgeführt wird :

```
http://ws.arduino-forth.com/record.php?log=pooltemp&pwd=pool2022&temp=27.5
```

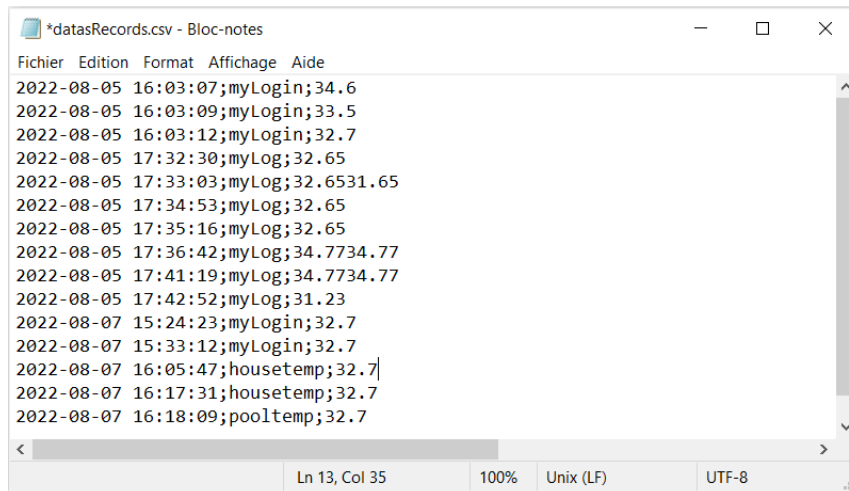
log pwd Paar , dessen Werte vom Datenaufzeichnungsskript getestet und genehmigt werden.

Aufgezeichnete Daten anzeigen

Nom de fichier	Taille d...	Type de ...	Dernière modification
..			
datasRecords.csv	672	Fichier C...	07/08/2022 16:18:09
record.php	927	Fichier P...	07/08/2022 16:17:22
gettime.php	41	Fichier P...	04/08/2022 15:25:24
index.php	8	Fichier P...	03/08/2022 20:14:10

Für den Zugriff auf die erfassten Daten nutzen wir einen FTP-Client (Filezilla):

Dort finden wir unsere Datei **datasRecords.csv** . Laden Sie es einfach herunter, um den Inhalt mit einem beliebigen Texteditor anzuzeigen:



In den letzten Zeilen finden wir unsere Übertragungstests mit zwei verschiedenen Logins. Das Skript **record.php** kann Transaktionen mit Hunderten verschiedener ESP32-Karten verarbeiten, jede mit einem anderen Login.

Fügen Sie die zu übertragenden Daten hinzu

Wenn Sie einen Sensor vom Typ DHT11 oder DHT22 (Temperatur- und Feuchtigkeitssensor) verwalten, könnten Sie versucht sein, die Temperatur- und Feuchtigkeitswerte in einer einzigen Transaktion aufzuzeichnen. Nichts könnte einfacher sein, als dies zu tun. Hier ist der Aspekt der Transaktion, der dies ermöglicht:

```
http://ws.arduino-forth.com/record.php?log=pooltemp&pwd=pool2022&temp=27.5&hygr=62.2
```

Damit es funktioniert, müssen Sie jedoch auf das PHP-Skript record.php reagieren:

```
<?php
// Recording datas in CSV file format
if (testAuths($myAuths)) {
    $handle = fopen("datasRecords.csv", "a");
    $myDatas = array(
        'currentDateTime' => date("Y-m-d H:i:s"),
        'currentLogin'    => $_GET['log'],
        'currentTemp'     => $_GET['temp'],
        'currentHygr'     => $_GET['hygr'],
    );
    fwrite($handle, implode(';', $myDatas). "\n");
    fclose($handle);
    echo "DATAS recorded";
} else {
    echo "AUTH failed";
}
```

Hier fügen wir einfach eine Zeile zur Tabelle **\$myDatas** hinzu .

Auf der FORTH-Seite werden wir die URL-Verwaltung verbessern:

```
256 string myUrl    \ declare string variable

: addTemp ( strAddrLen -- )
  s" &temp=" myUrl append$
  myUrl append$
;

: addHygr ( strAddrLen -- )
  s" &hygr=" myUrl append$
  myUrl append$
;

: sendData ( strHygr strTemp -- )
  s" http://ws.arduino-forth.com/record.php?log=myLog&pwd=myPassWd" myUrl
  $!
  addTemp
  addHygr
  cr myUrl type
  myUrl s>z HTTP.begin
  if
    HTTP.doGet dup 200 =
    if drop
      httpBuffer bufferSize HTTP.getPayload
      httpBuffer z>s type
    else
      cr ." CNX ERR: " .
    then
  then
  HTTP.end
;

\ for test:
myWiFiConnect
s" 64.2"    \ hygrometry
s" 31.23"   \ temperature
sendData
```

Wir haben zwei Wörter hinzugefügt, **addTemp** und **addHygr** . Jedes dieser Wörter verknüpft einen Parameter und seinen Wert mit der URL, die für die Webtransaktion zwischen Ihrer ESP32-Karte und dem Webserver verwendet wird.

Es gibt nur zwei Einschränkungen hinsichtlich der Anzahl der von der GET-Methode übergebenen Parameter:

- die Länge unserer URL wie in FORTH definiert, hier 256 Zeichen. Wenn Sie dieses Limit erhöhen möchten, stellen Sie einfach unsere URL mit einer längeren Anfangslänge ein: **512 string myUrl**
- die maximale Länge von URLs, die vom HTTP-Protokoll akzeptiert werden. Diese Länge kann nach aktuellen Standards 8000 Zeichen erreichen.

In Bezug auf FORTH haben wir andere Einschränkungen. Insbesondere, wenn wir Textdaten übermitteln möchten. Bestimmte Zeichen, zum Beispiel „&“, müssen codiert werden. Sie müssen diese Kodierung in FORTH durchführen.

Abschluss

FRAGE: Wofür kann das alles genutzt werden?

Eine ESP32-Karte kostet jeweils weniger als 10 €/\$. Noch eher 5 €/\$, wenn Sie in großen Mengen kaufen. Wenn Sie einen Temperatursensor und ein Relais integrieren, können Sie beispielsweise Temperaturmessungen durchführen und Befehle vom Server übertragen, um ein Relais zu aktivieren/deaktivieren. Die Temperaturregelung mehrerer Räume wird ganz einfach. Das Gleiche gilt für die intelligente Bewässerung eines Gewächshauses.

Außerdem können Sie ganz einfach Zutritte überwachen und Lichter oder Alarme auslösen. Nehmen wir den Fall eines Portals. Sie genehmigen Durchgänge zwischen bestimmten Zeiten und verriegeln dasselbe Tor (magnetischer Saugnapf) über die ESP32-Karte.

Wir vertrauen auf Ihre Vorstellungskraft, um praktische Lösungen zu finden, die diese Datenübertragung zwischen ESP32-Karten und einem Webserver nutzen.

UND WARUM EIN WEBSERVER?

Mit einem Webserver ist die Abfrage einfach von überall aus möglich, mit einem auf Ihrem PC installierten Webbrowser, einem digitalen Tablet, einem Smartphone. Und ein einzelner Webserver kann eine unbegrenzte Anzahl verschiedener Skripte integrieren.

Der Zufallszahlengenerator

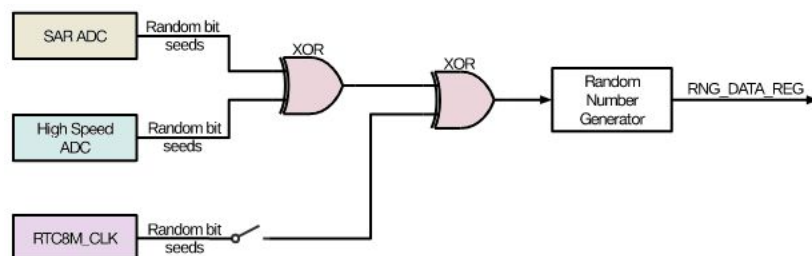
Charakteristisch

Der Zufallszahlengenerator generiert echte Zufallszahlen, also eine Zufallszahl, die durch einen physikalischen Prozess und nicht durch einen Algorithmus generiert wird. Keine innerhalb des angegebenen Bereichs generierte Zahl erscheint mit größerer oder geringerer Wahrscheinlichkeit als jede andere Zahl.

Jeder 32-Bit-Wert, den das System aus dem RNG_DATA_REG-Register des Zufallszahlengenerators liest, ist eine echte Zufallszahl. Diese echten Zufallszahlen werden basierend auf thermischem Rauschen im System und asynchronem Taktversatz generiert.

Das thermische Rauschen kommt vom Hochgeschwindigkeits-ADC oder vom SAR-ADC oder von beiden. Immer wenn der Hochgeschwindigkeits-ADC oder SAR-ADC aktiviert wird, werden die Bitströme generiert und über ein XOR-Logikgatter als Zufalls-Seeds in den Zufallszahlengenerator eingespeist.

Wenn der RTC8M_CLK-Takt für den digitalen Kern aktiviert ist, tastet der Zufallszahlengenerator auch RTC8M_CLK (8 MHz) als zufälligen binären Startwert ab. RTC8M_CLK ist eine asynchrone Taktquelle und erhöht die RNG-Entropie durch die Einführung von Schaltungsmetastabilität. Um maximale Entropie zu gewährleisten, wird jedoch auch empfohlen, immer eine ADC-Quelle zu aktivieren.



Wenn vom SAR-ADC Rauschen auftritt, wird der Zufallszahlengenerator mit einer Entropie von 2 Bits in einem Taktzyklus von RTC8M_CLK (8 MHz) gespeist, die von einem internen RC-Oszillator erzeugt wird (weitere Einzelheiten finden Sie im Kapitel „Reset und Takt“). Daher ist es ratsam, das **RNG_DATA_REG**- Register mit einer maximalen Rate von 500 kHz zu lesen, um die maximale Entropie zu erhalten.

Wenn Rauschen vom Hochgeschwindigkeits-ADC auftritt, wird dem Zufallszahlengenerator in einem APB-Taktzyklus, der normalerweise 80 MHz beträgt, 2-Bit-Entropie zugeführt.

Daher ist es ratsam, das **RNG_DATA_REG- Register** mit einer maximalen Rate von **5 MHz** zu lesen , um die maximale Entropie zu erhalten.

Eine 2-GB-Datenprobe, die vom Zufallszahlengenerator mit einer Frequenz von 5 MHz gelesen wird, wobei nur der Hochgeschwindigkeits-ADC aktiviert ist, wurde mit der Dieharder Random Number-Testsuite (Version 3.31.1) getestet. Die Probe hat alle Tests bestanden.

Programmiervorgang

Stellen Sie bei Verwendung des Zufallszahlengenerators sicher, dass mindestens SAR ADC, High Speed ADC oder RTC8M_CLK zulässig sind. Andernfalls werden Pseudozufallszahlen zurückgegeben.

- SAR ADC kann mit dem DIG ADC-Controller aktiviert werden.
- Der Hochgeschwindigkeits-ADC wird automatisch aktiviert, wenn Wi-Fi- oder Bluetooth-Module aktiviert sind.
- RTC8M_CLK wird durch Setzen des RTC_CNTL_DIG_CLK8M_EN-Bits im RTC_CNTL_CLK_CONF_REG-Register aktiviert.

Wenn Sie den Zufallszahlengenerator verwenden, lesen Sie das **RNG_DATA_REG - Register** mehrmals, bis genügend Zufallszahlen generiert wurden.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	\$3FF75144	RO

```
\ Zufallszahlendaten
$3FF75144 constant RNG_DATA_REG

\ Holen Sie sich eine 32-Bit-Zufallszahl b=Zahl
: rnd ( -- x )
  RNG_DATA_REG L@
;

\ Zufallszahl im Intervall [0..n-1] erhalten
: random ( n -- 0..n-1 )
  rnd swap mod
;
```

RND-Funktion im XTENSA-Assembler

Seit Version 7.0.7.4 verfügt ESP32forth über einen XTENSA-Assembler. Es ist möglich, unser **drittes Wort** im XTENSA-Assembler umzuschreiben :

```
forth definitions
asm xtensa
$3FF75144 constant RNG_DATA_REG

code myRND ( -- [addr] )
```

```

a1 32          ENTRY,
a8 RNG_DATA_REG L32R,      \ a8 = RNG_DATA_REG
a9 a8 0        L32I.N,     \ a9 = [a8]
a9            arPUSH,      \ push a9 on stack
                    RETW.N,
end-code

```

Detaillierter Inhalt der ESP32forth-Vokabulare

ESP32forth bietet zahlreiche Vokabulare:

- **FORTH** ist das Hauptvokabular;
- Bestimmte Vokabulare werden für interne Mechaniken für ESP32Forth verwendet, wie zum Beispiel **internals** , **asm...**
- Viele Vokabulare ermöglichen die Verwaltung bestimmter Anschlüsse oder Zubehörteile, wie **Bluetooth** , **OLED** , **SPI** , **WiFi wire**.

Hier finden Sie die Liste aller in diesen verschiedenen Vokabeln definierten Wörter. Einige Wörter werden mit einem farbigen Link dargestellt :

[align](#) ist ein gewöhnliches FORTH-Wort;

CONSTANT ist Definitionswort;

begin markiert eine Kontrollstruktur;

key ist ein verzögertes Ausführungswort;

LED ist ein Wort, das durch eine Konstante , eine Variable oder einen Wert definiert ist ;

registers markiert einen Wortschatz.

FORTH- Vokabularwörter werden in alphabetischer Reihenfolge angezeigt. Bei anderen Vokabularen werden die Wörter in ihrer Anzeigereihenfolge angezeigt.

Version v 7.0.7.15

FORTH

=	-rot	_	:	:	:noname	!
?	?do	?dup	.	."	.s	'
(local)	[[']	[char]	[ELSE]	[IF]	[THEN]
l	f	f	}transfer	@	*	*/
*/MOD	/	/mod	#	#!	#>	#fs
#s	#tib	+	+!	+loop	+to	<
<#	<=	<>	=	>	>=	>BODY
>flags	>flags&	>in	>link	>link&	>name	>params
>R	>size	0<	0<>	0=	1-	1/F
1+	2!	2@	2*	2/	2drop	2dup
4*	4/	abort	abort"	abs	accept	adc
afliteral	aft	again	ahead	align	aligned	allocate
allot	also	analogRead	AND	ansi	ARSHIFT	asm
assert	at-xy	base	begin	bq	BIN	binary
bl	blank	block	block-fid	block-id	buffer	bye
c,	C!	C@	CASE	cat	catch	CELL
cell/	cell+	cells	char	CLOSE-DIR	CLOSE-FILE	cmove

cmove>	CONSTANT	context	copy	cp	cr	CREATE
CREATE-FILE	current	dacWrite	decimal	default-key	default-key?	
default-type		default-use	defer	DEFINED?	definitions	DELETE-FILE
depth	digitalRead	digitalWrite		do	DOES>	DROP
dump	dump-file	DUP	duty	echo	editor	else
emit	empty-buffers		ENDCASE	ENDOF	erase	ESP
ESP32-C3?	ESP32-S2?	ESP32-S3?	ESP32?	evaluate	EXECUTE	exit
extract	F-	f.	f.s	F*	F**	F/
F+	F<	F<=	F<>	F=	F>	F>=
F>S	F0<	F0=	FABS	FATAN2	fconstant	FCOS
fdepth	FDROP	FDUP	FEXP	fq	file-exists?	
FILE-POSITION		FILE-SIZE	fill	FIND	fliteral	FLN
FLOOR	flush	FLUSH-FILE	FMAX	FMIN	FNEGATE	FNIP
for	forget	FORTH	forth-builtins		FOVER	FP!
FP@	fp0	free	freq	FROT	FSIN	FSINCOS
FSQRT	FSWAP	fvariable	handler	here	hex	HIGH
hld	hold	httpd	I	if	IMMEDIATE	include
included	included?	INPUT	internals	invert	is	J
K	key	key?	L!	latestxt	leave	LED
ledc	list	literal	load	login	loop	LOW
ls	LSHIFT	max	MDNS.begin	min	mod	ms
MS-TICKS	mv	n.	needs	negate	nest-depth	next
nip	nl	NON-BLOCK	normal	octal	OF	ok
only	open-blocks	OPEN-DIR	OPEN-FILE	OR	order	OUTPUT
OVER	pad	page	PARSE	pause	PI	pin
pinMode	postpone	precision	previous	prompt	PSRAM?	pulseIn
quit	r"	R@	R/O	R/W	R>	rl
r~	rdrop	read-dir	READ-FILE	recurse	refill	registers
remaining	remember	RENAME-FILE	repeat	REPOSITION-FILE		required
reset	resize	RESIZE-FILE	restore	revive	RISC-V?	rm
rot	RP!	RP@	rp0	RSHIFT	rtos	s"
S>F	s>z	save	save-buffers		scr	SD
SD_MMC	sealed	see	Serial	set-precision		set-title
sf,	SF!	SF@	SFLOAT	SFLOAT+	SFLOATS	sign
SL@	sockets	SP!	SP@	sp0	space	spaces
SPIFFS	start-task	startswith?	startup:	state	str	str=
streams	structures	SW@	SWAP	task	tasks	telnetd
terminate	then	throw	thru	tib	to	tone
touch	transfer	transfer	type	u.	U/MOD	UL@
UNLOOP	until	update	use	used	UW@	value
VARIABLE	visual	vlist	vocabulary	W!	W/O	web-
interface						
webui	while	WiFi	Wire	words	WRITE-FILE	XOR
Xtensa?	z"	z>s				

asm

```
xtensa disasm disasm1 matchit address istep sextend m. m@ for-ops op >operands
>mask >pattern >length >xt op-snap opcodes coden, names operand l o bits
bit skip advance advance-operand reset reset-operand for-operands operands
>printop >inop >next >opmask& bit! mask pattern length demask enmask >>1
odd? high-bit end-code code, code4, code3, code2, code1, callot chere reserve
```

```
code-at code-start
```

bluetooth

```
SerialBT.new SerialBT.delete SerialBT.begin SerialBT.end SerialBT.available  
SerialBT.readBytes SerialBT.write SerialBT.flush SerialBT.hasClient  
SerialBT.enableSSP SerialBT.setPin SerialBT.unpairDevice SerialBT.connect  
SerialBT.connectAddr SerialBT.disconnect SerialBT.connected  
SerialBT.isReady bluetooth-builtins
```

editor

```
a r d e wipe p n l
```

ESP

```
getHeapSize getFreeHeap getMaxAllocHeap getChipModel getChipCores getFlashChipSize  
getCpuFreqMHz getSketchSize deepSleep getEfuseMac esp_log_level_set ESP-builtins
```

httpd

```
notfound-response bad-response ok-response response send path method hasHeader  
handleClient read-headers completed? body content-length header crnl= eat  
skipover skipto in@<> end< goal# goal strcase= upper server client-cr client-emit  
client-read client-type client-len client httpd-port clientfd sockfd body-read  
body-1st-read body-chunk body-chunk-size chunk-filled chunk chunk-size  
max-connections
```

insides

```
run normal-mode raw-mode step ground handle-key quit-edit save load backspace  
delete handle-esc insert update crtype cremit ndown down nup up caret length  
capacity text start-size fileh filename# filename max-path
```

internals

```
assembler-source xtensa-assembler-source MALLOC SYSFREE REALLOC heap_caps_malloc  
heap_caps_free heap_caps_realloc heap_caps_get_total_size heap_caps_get_free_size  
heap_caps_get_minimum_free_size heap_caps_get_largest_free_block RAW-YIELD  
RAW-TERMINATE READDIR CALLCODE CALL0 CALL1 CALL2 CALL3 CALL4 CALL5 CALL6  
CALL7 CALL8 CALL9 CALL10 CALL11 CALL12 CALL13 CALL14 CALL15 DOFLIT S>FLOAT?  
fill32 'heap' 'context' 'latestxt' 'notfound' 'heap-start' 'heap-size' 'stack-cells'  
'boot' 'boot-size' 'tib' 'argc' 'argv' 'runner' 'throw-handler' NOP BRANCH OBRANCH  
DONEXT DOLIT DOSET DOCOL DOCON DOVAR DOCREATE DODOES ALITERAL LONG-SIZE  
S>NUMBER? 'SYS' YIELD EVALUATE1 'builtins internals-builtins autoexec  
arduino-remember-filename  
arduino-default-use esp32-stats serial-key? serial-key serial-type yield-task  
yield-step e' @line grow-blocks use?! common-default-use block-data block-dirty  
clobber clobber-line include+ path-join included-files raw-included include-file  
sourcedirname sourcefilename! sourcefilename sourcefilename# sourcefilename&  
starts../ starts./ dirname ends/ default-remember-filename remember-filename
```

```

restore-name save-name forth-wordlist setup-saving-base 'cold park-forth
park-heap saving-base crtype cremit cases \(+to\) \(to\) --? }? ?room scope-create
do-local scope-clear scope-exit local-op scope-depth local+! local! local@
<>locals locals-here locals-area locals-gap locals-capacity ?ins. ins.
vins. onlines line-pos line-width size-all size-vocabulary vocs. voc. voclist
voclist-from see-all >vocnext see-vocabulary nonvoc? see-xt ?see-flags
see-loop see-one indent+! icr see. indent mem= ARGS_MARK -TAB +TAB NONAMED
BUILTIN_FORK SMUDGE IMMEDIATE_MARK relinquish dump-line ca@ cell-shift
cell-base cell-mask MALLOC_CAP_RTDRAM MALLOC_CAP_RETENTION MALLOC_CAP_IRAM_8BIT
MALLOC_CAP_DEFAULT MALLOC_CAP_INTERNAL MALLOC_CAP_SPIRAM MALLOC\_CAP\_DMA
MALLOC\_CAP\_8BIT MALLOC\_CAP\_32BIT MALLOC\_CAP\_EXEC #f+s internalized BUILTIN_MARK
zplace $place free. boot-prompt raw-ok \[SKIP\]' \[SKIP\] ?stack sp-limit input-limit
tib-setup raw.s $@ digit parse-quote leaving, leaving )leaving leaving(
value-bind evaluate&fill evaluate-buffer arrow ?arrow. ?echo input-buffer
immediate? eat-till-cr wascr *emit *key notfound last-vocabulary voc-stack-end
xt-transfer xt-hide xt-find& scope

```

interrupts

```

pinchange #GPIO\_INTR\_HIGH\_LEVEL #GPIO\_INTR\_LOW\_LEVEL #GPIO\_INTR\_ANYEDGE
#GPIO\_INTR\_NEGEDGE #GPIO\_INTR\_POSEDGE #GPIO\_INTR\_DISABLE ESP\_INTR\_FLAG\_INTRDISABLED
ESP\_INTR\_FLAG\_IRAM ESP\_INTR\_FLAG\_EDGE ESP\_INTR\_FLAG\_SHARED ESP\_INTR\_FLAG\_NMI
ESP\_INTR\_FLAG\_LEVELn ESP\_INTR\_FLAG\_DEFAULT gpio\_config gpio\_reset\_pin gpio\_set\_intr\_type
gpio\_intr\_enable gpio\_intr\_disable gpio\_set\_level gpio\_get\_level gpio\_set\_direction
gpio\_set\_pull\_mode gpio\_wakeup\_enable gpio\_wakeup\_disable gpio\_pullup\_en
gpio pulldown\_en gpio pulldown\_dis gpio\_hold\_en gpio\_hold\_dis
gpio\_deep\_sleep\_hold\_en gpio\_deep\_sleep\_hold\_dis gpio\_install\_isr\_service
gpio\_isr\_handler\_add gpio\_isr\_handler\_remove
gpio\_set\_drive\_capability gpio\_get\_drive\_capability esp\_intr\_alloc esp\_intr\_free
interrupts-builtins

```

ledc

```

ledcSetup ledcAttachPin ledcDetachPin ledcRead ledcReadFreq ledcWrite ledcWriteTone
ledcWriteNote ledc-builtins

```

oled

```

OledInit SSD1306\_SWITCHCAPVCC SSD1306\_EXTERNALVCC WHITE BLACK OledReset HEIGHT
WIDTH OledAddr OledNew OledDelete OledBegin OledHOME OledCLS OledTextc
OledPrintln OledNumln OledNum OledDisplay OledPrint OledInvert OledTextsize
OledSetCursor OledPixel OledDrawL OledCirc OledCircF OledRect OledRectF
OledRectR OledRectRF oled-builtins

```

registers

```

m@ m!

```

riscv

```

C.FSWSP, C.SWSP, C.FSDSP, C.ADD, C.JALR, C.EBREAK, C.MV, C.JR, C.FLWSP,
C.LWSP, C.FLDSP, C.SLLI, BNEZ, BEQZ, C.J, C.ADDW, C.SUBW, C.AND, C.OR,
C.XOR, C.SUB, C.ANDI, C.SRAI, C.SRLI, C.LUI, C.LI, C.JAL, C.ADDI, C.NOP,
C.FSW, C.SW, C.FSD, C.FLW, C.LW, C.FLD, C.ADDI4SP, C.ILL, EBREAK, ECALL,
AND, OR, SRA, SRL, XOR, SLTU, SLT, SLL, SUB, ADD, SRAI, SRLI, SLLI, ANDI,

```



```
ORI, XORI, SLTIU, SLTI, ADDI, SW, SH, SB, LHU, LBU, LW, LH, LB, BGEU, BLTU,
BGE, BLT, BNE, BEQ, JALR, JAL, AUIPC, LUI, J-TYPE U-TYPE B-TYPE S-TYPE
I-TYPE R-TYPE rs2' rs2#' rs2 rs2# rs1' rs1#' rs1 rs1# rd' rd#' rd rd# offset
ofs ofs. >ofs iiii i numeric register' reg'. reg>reg' register reg. nop
x31 x30 x29 x28 x27 x26 x25 x24 x23 x22 x21 x20 x19 x18 x17 x16 x15 x14
x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 zero
```

rtos

```
vTaskDelete xTaskCreatePinnedToCore xPortGetCoreID rtos-builtins
```

SD

```
SD.begin SD.beginFull SD.beginDefaults SD.end SD.cardType SD.totalBytes
SD.usedBytes SD-builtins
```

SD_MMC

```
SD\_MMC.begin SD_MMC.beginFull SD_MMC.beginDefaults SD_MMC.end SD_MMC.cardType
SD_MMC.totalBytes SD\_MMC.usedBytes SD_MMC-builtins
```

Serial

```
Serial.begin Serial.end Serial.available Serial.readBytes Serial.write
Serial.flush Serial.setDebugOutput Serial2.begin Serial2.end Serial2.available
Serial2.readBytes Serial2.write Serial2.flush Serial2.setDebugOutput serial-
builtins
```

sockets

```
ip. ip# ->h_addr ->addr! ->addr@ ->port! ->port@ sockaddr l, s, bs, SO\_REUSEADDR
SOL\_SOCKET sizeof\(sockaddr\_in\) AF\_INET SOCK\_RAW SOCK\_DGRAM SOCK\_STREAM
socket setsockopt bind listen connect sockaccept select poll send sendto
sendmsg recv recvfrom recvmsg gethostbyname errno sockets-builtins
```

spi

```
SPI.begin SPI.end SPI.setHwCs SPI.setBitOrder SPI.setDataMode SPI.setFrequency
SPI.setClockDivider SPI.getClockDivider SPI.transfer SPI.transfer8 SPI.transfer16
SPI.transfer32 SPI.transferBytes SPI.transferBits SPI.write SPI.write16
SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern SPI-builtins
```

SPIFFS

```
SPIFFS.begin SPIFFS.end SPIFFS.format SPIFFS.totalBytes SPIFFS.usedBytes
SPIFFS-builtins
```

streams

```
stream> >stream stream>ch ch>stream wait-read wait-write empty? full? stream#
>offset >read >write stream
```

structures

```
field struct-align align-by last-struct struct long ptr i64 i32 i16 i8
```

```
typer last-align
```

tasks

```
.tasks main-task task-list
```

telnetd

```
server broker-connection wait-for-connection connection telnet-key telnet-type  
telnet-emit broker client-len client telnet-port clientfd sockfd
```

visual

```
edit insides
```

web-interface

```
server webserver-task do-serve handle1 serve-key serve-type handle-input  
handle-index out-string output-stream input-stream out-size webserver index-html  
index-html#
```

WiFi

```
Wire.begin Wire.setClock Wire.getClock Wire.setTimeout Wire.getTimeout  
Wire.beginTransaction Wire.endTransmission Wire.requestFrom Wire.write  
Wire.available Wire.read Wire.peek Wire.flush Wire-builtins
```

xtensa

```
WUR, WSR, WITLB, WER, WDTLB, WAITI, SSXU, SSX, SSR, SSL, SSIU, SSI, SSAI,  
SSA8L, SSA8B, SRLI, SRL, SRC, SRAI, SRA, SLLI, SLL, SICW, SICT, SEXT, SDCT,  
RUR, RSR, RSIL, RFI, ROTW, RITLB1, RITLB0, RER, RDTLB1, RDTLB0, PITLB,  
PDTLB, NSAU, NSA, MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULS.DD  
MULA.DA.HH, MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULS.DA MULA.AD.HH, MULA.AD.LH,  
MULA.AD.HL, MULA.AD.LL, MULS.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL,  
MULS.AA MULA.DD.HH.LDINC, MULA.DD.LH.LDINC, MULA.DD.HL.LDINC, MULA.DD.LL.LDINC,  
MULA.DD.LDINC MULA.DD.HH.LDDEC, MULA.DD.LH.LDDEC, MULA.DD.HL.LDDEC,  
MULA.DD.LL.LDDEC,  
MULA.DD.LDDEC MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULA.DD  
MULA.DA.HH.LDINC,  
MULA.DA.LH.LDINC, MULA.DA.HL.LDINC, MULA.DA.LL.LDINC, MULA.DA.LDINC  
MULA.DA.HH.LDDEC,  
MULA.DA.LH.LDDEC, MULA.DA.HL.LDDEC, MULA.DA.LL.LDDEC, MULA.DA.LDDEC MULA.DA.HH,  
MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULA.DA MULA.AD.HH, MULA.AD.LH, MULA.AD.HL,  
MULA.AD.LL, MULA.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL, MULA.AA  
MUL16U, MUL16S, MUL.DD.HH, MUL.DD.LH, MUL.DD.HL, MUL.DD.LL, MUL.DD MUL.DA.HH,  
MUL.DA.LH, MUL.DA.HL, MUL.DA.LL, MUL.DA MUL.AD.HH, MUL.AD.LH, MUL.AD.HL,  
MUL.AD.LL, MUL.AD MUL.AA.HH, MUL.AA.LH, MUL.AA.HL, MUL.AA.LL, MUL.AA MOVLT,  
MOVSP, MOVLT.S, MOVFT.S, MOVGEZ.S, MOVLTZ.S, MOVNEZ.S, MOVEQZ.S, ULE.S, OLE.S,  
ULT.S, OLT.S, UEQ.S, OEQ.S, UN.S, CMPSP NEG.S, WFR, RFR, ABS.S, MOV.S,  
ALU2.S UTRUNC.S, UFLOAT.S, FLOAT.S, CEIL.S, FLOOR.S, TRUNC.S, ROUND.S,  
MSUB.S, MADD.S, MUL.S, SUB.S, ADD.S, ALU.S MOVFT, MOVGEZ, MOVLTZ, MOVNEZ,  
MOVEQZ, MAXU, MINU, MAX, MIN, CONDOP MOV, LSXU, LSX, L32E, LICW, LICT,  
LDCT, JX, IITLB, IDTLB, LSIU, LSI, LDINC, LDDEC, L32R, EXTUI, S32E, S32RI,
```

S32CI, [ADDMI](#), [ADDI](#), L32AI, L16SI, S32I, S16I, S8I, L32I, L16UI, L8UI,
 LDSTORE [MOVI](#), IIU, IHU, IPFL, DIWBI, DIWB, DIU, DHU, DPFL, CACHING2 III,
 IHI, IPF, DII, DHI, DHWBI, DHWB, DPFWO, DPFR, DPFR, CACHING1 CLAMPS,
 BREAK, CALLX12, CALLX8, CALLX4, CALLX0, CALLXOP CALL12, CALL8, CALL4, [CALL0](#),
 CALLOP LOOPGTZ, LOOPNEZ, [LOOP](#), BT, BF, BRANCH2b [J](#), BGEUI, BGEI, BGEZ, BLTUI,
 BLTI, BLTZ, BNEI, BNEZ, [ENTRY](#), BEQI, [BEQZ](#), BRANCH2e BRANCH2a BRANCH2 BBSI,
 BBS, BNALL, BGEU, BGE, BNE, BANY, BBCI, BBC, [BALL](#), BLTU, BLT, [BEQ](#), BNONE,
 BRANCH1 [REMS](#), REMU, [QUOS](#), QUOU, MULSH, MULUH, [MULL](#), XORB, ORBC, ORB, [ANDBC](#),
[ANDB](#), ALU2 [ALL8](#), [ANY8](#), [ALL4](#), [ANY4](#), ANYALL SUBX8, SUBX4, SUBX2, SUB, [ADDX8](#),
[ADDX4](#), [ADDX2](#), [ADD](#), [XOR](#), [OR](#), [AND](#), ALU XSR, [ABS](#), [NEG](#), RFDO, RFDD, SIMCALL,
 SYSCALL, RFWU, RFWO, RFDE, RFUE, RFME, RFE, [NOP](#), [EXTW](#), MEMW, EXCW, DSYNC,
 ESYNC, RSYNC, ISYNC, RETW, [RET](#), ILL, ILL.N, [NOP.N](#), [RETW.N](#), [RET.N](#), BREAK.N,
 MOV.N, MOVI.N, BNEZ.N, BEQZ.N, [ADDI.N](#), [ADD.N](#), [S32I.N](#), [L32I.N](#), tttt t ssss
 s rrrr r bbbb b y w iiii [i](#) xxxx x sa sa. >sa entry12 entry12' entry12.
 >entry12 coffset18 cofs cofs. >cofs offset18 offset12 offset8 ofs18 ofs12
 ofs8 ofs18. ofs12. ofs8. >ofs sr imm16 imm8 imm4 im numeric register reg.
 nop [a15](#) [a14](#) [a13](#) [a12](#) [a11](#) [a10](#) [a9](#) [a8](#) [a7](#) [a6](#) [a5](#) [a4](#) [a3](#) [a2](#) [a1](#) [a0](#)

Ressourcen

Auf Englisch

- **ESP32forth** Seite verwaltet von Brad NELSON, dem Erfinder von ESP32forth. Dort finden Sie alle Versionen (ESP32, Windows, Web, Linux...)
<https://esp32forth.appspot.com/ESP32forth.html>

•

Auf Französisch

- **ESP32 Forth** Website in zwei Sprachen (Französisch, Englisch) mit vielen Beispielen
<https://esp32.arduino-forth.com/>

GitHub

- **Ueforth** Ressourcen verwaltet von Brad NELSON. Enthält alle Forth- und C-Sprach Quelldateien für ESP32forth
<https://github.com/flagxor/ueforth>
- **ESP32forth** Quellcodes und Dokumentation für ESP32forth. Ressourcen verwaltet von Marc PETREMANN
<https://github.com/MPETREMANN11/ESP32forth>
- **ESP32forthStation** Ressourcen verwaltet von Ulrich HOFFMAN. Eigenständiger Forth-Computer mit LillyGo TTGO VGA32-Einplatinencomputer und ESP32forth
<https://github.com/uho/ESP32forthStation>
- **ESP32Forth** Ressourcen verwaltet von F. J. RUSSO
<https://github.com/FJRusso53/ESP32Forth>
- **esp32forth-addons** Ressourcen verwaltet von Peter FORTH
<https://github.com/PeterForth/esp32forth-addons>
- **Esp32forth-org** Code-Repository für Mitglieder der Forth2020- und ESP32forth-Gruppen
<https://github.com/Esp32forth-org>

•