

Das grosse Buch für ESP32forth

version 2.0 - Dienstag, 17. März 2026



Installieren Sie die Programmiersprache Forth auf der ESP32-Karte. Lernen Sie, in der Forth-Sprache zu programmieren. Verwenden Sie die Forth-Sprache, um das ESP32-Board anzusteuern. Erfahren Sie, wie das ESP32-Board funktioniert.

Autor

- Marc PETREMAN

Mitarbeiter

- Bob EDWARDS
- Vaclav POSELT
- Thomas SCHREIN
- Martin BITTER

Inhalt

Autor.....	1
Mitarbeiter.....	1
Einführung.....	12
Übersetzungshilfe.....	12
Entdeckung der ESP32-Karte.....	13
Präsentation.....	13
Die Stärken Punkten.....	13
GPIO-Ein-/Ausgänge auf ESP32.....	14
ESP32-Board-Peripheriegeräte.....	15
Die verschiedenen ESP32-Karten.....	17
Endgültige Installation von ESP32forth.....	18
Die ESP32 Wroom 32-Karte.....	18
Anschlussplatine.....	19
Das ESP32 Wrover-Karte.....	20
Anschlussplatine.....	20
Die ESP32 S3-Karte.....	21
Anschlussplatine.....	21
Installieren Sie ESP32Forth.....	23
ESP32forth herunterladen.....	23
ESP32forth kompilieren und installieren.....	23
Parameter für ESP32 WROOM 32.....	25
Starten Sie die Kompilation.....	26
Fehler beim Hochladen der Verbindung beheben.....	28
Warum auf ESP32 in FORTH-Sprache programmieren?.....	30
Präambel.....	30
Grenzen zwischen Sprache und Anwendung.....	31
Was ist ein FORTH-Wort?.....	31
Ein Wort ist eine Funktion?.....	31
FORTH-Sprache im Vergleich zur C-Sprache.....	32
Was FORTH Ihnen im Vergleich zur C-Sprache ermöglicht.....	33
Aber warum ein Stapel statt Variablen?.....	34
Sind Sie überzeugt?.....	34
Gibt es professionelle Bewerbungen, die in FORTH verfasst sind?.....	35
Verwenden von Zahlen mit ESP32Forth.....	36
Zahlen mit dem FORTH-Interpreter.....	36
Eingeben von Zahlen mit unterschiedlichen Zahlenbasen.....	37
Änderung der Zahlenbasis.....	38
Binär und hexadezimal.....	38
Größe der Zahlen im FORTH-Datenstapel.....	40
Speicherzugriff und logische Operationen.....	42
Ein echtes 32-Bit FORTH mit ESP32Forth.....	45
Werte auf dem Datenstapel.....	45

Werte im Gedächtnis.....	45
Textverarbeitung je nach Datengröße oder -typ.....	46
Abschluss.....	47
Kommentare und Erläuterungen.....	49
Schreiben Sie lesbaren FORTH-Code.....	49
Einrückung des Quellcodes.....	50
Kommentare.....	51
Stapelkommentare.....	51
Bedeutung von Stack-Parametern in Kommentaren.....	52
Wortdefinition Wortkommentare.....	53
Textkommentare.....	53
Kommentar am Anfang des Quellcodes.....	54
Diagnose- und Tuning-Tools.....	54
Der Dekompiler.....	54
Speicherauszug.....	55
Stapel monitor.....	55
Wörterbuch / Stapel / Variablen / Konstanten.....	57
Wörterbuch erweitern.....	57
Wörterbuchverwaltung.....	57
Stapel und umgekehrte polnische Notation.....	58
Umgang mit dem Parameterstapel.....	59
Der Return Stack und seine Verwendung.....	60
Speichernutzung.....	60
Variablen.....	61
Konstanten.....	61
Pseudokonstante Werte.....	61
Grundlegende Tools für die Speicherzuweisung.....	62
Textfarben und Anzeigeposition auf dem Terminal.....	63
ANSI-Kodierung von Terminals.....	63
Textfärbung.....	63
Anzeigeposition.....	65
Lokale Variablen mit ESP32Forth.....	67
Einführung.....	67
Der Fake-Stack-Kommentar.....	67
Aktion auf lokale Variablen.....	68
Datenstrukturen für ESP32forth.....	71
Präambel.....	71
Tabellen in FORTH.....	71
Eindimensionales 32-Bit-Datenarray.....	71
Tabellendefinitionswörter.....	72
Lesen und schreiben Sie in eine Tabelle.....	72
Praktisches Beispiel für die Verwaltung eines virtuellen Bildschirms.....	73
Management komplexer Strukturen.....	76
Definition von Sprites.....	78
Reale Zahlen mit ESP32forth.....	80

Die echten mit ESP32forth.....	80
Echte Zahlengenauigkeit mit ESP32forth.....	80
Reale Konstanten und Variablen.....	81
Arithmetische Operatoren für reelle Zahlen.....	81
Mathematische Operatoren für reelle Zahlen.....	81
Logische Operatoren für reelle Zahlen.....	82
Ganzzahlige ↔ reelle Transformationen.....	82
Zahlen und Zeichenfolgen anzeigen.....	84
Änderung der Zahlenbasis.....	84
Definition neuer Anzeigeformate.....	85
Anzeigen von Zeichen und Zeichenfolgen.....	87
String-Variablen.....	89
Wortcode zur Verwaltung von Textvariablen.....	89
Hinzufügen von Zeichen zu einer alphanumerischen Variablen.....	91
Vokabeln mit ESP32forth.....	93
Liste der Vokabeln.....	93
Grundlegende Vokabeln.....	94
Liste der Vokabelinhalte.....	94
Verwendung von Vokabeln.....	94
Verkettung von Vokabeln.....	95
Verzögerte Aktionswörter.....	97
Definition und Verwendung von Wörtern mit defer.....	98
Festlegen einer Vorwärtsreferenz.....	98
Abhängigkeit vom Betriebskontext.....	99
Ein praktischer Fall.....	100
Wortschöpfungswörter.....	103
Die Verwendung des Wortes does>.....	103
Beispiel für Farbmanagement.....	104
Beispiel: Schreiben in Pinyin.....	105
Passen Sie Steckbretter an das ESP32-Board an.....	107
Testplatten für ESP32.....	107
Bauen Sie ein Steckbrett, das für das ESP32-Board geeignet ist.....	107
Stromversorgung der ESP32-Karte.....	109
Wahl der Stromquelle.....	109
Stromversorgung über Mini-USB-Anschluss.....	109
Stromversorgung über 5V-Pin.....	109
Automatischer Start eines Programms.....	111
Installieren und verwenden Sie das Tera Term-Terminal unter Windows.....	113
Installieren Sie Tera Term.....	113
Tera Term einrichten.....	113
Verwendung von Tera Term.....	116
Kompilieren Sie den Quellcode in der Forth-Sprache.....	117
Greifen Sie über TELNET auf ESP32Forth zu.....	119
Ändern Sie den DNS-Namen des ESP32-Boards.....	119
Herstellen einer Verbindung zu ESP32-Boards über deren Hostnamen.....	120

Verwaltung von Quelldateien nach Blöcken.....	123
Die Blöcke.....	123
Öffnen Sie eine Blockdatei.....	123
Bearbeiten Sie den Inhalt eines Blocks.....	124
Blockinhalte zusammenstellen.....	125
Praktisches Schritt-für-Schritt-Beispiel.....	126
Abschluss.....	127
Bearbeiten von Quelldateien mit VISUAL Editor.....	128
Bearbeiten Sie eine FORTH-Quelldatei.....	128
Bearbeiten des FORTH-Codes.....	128
Kompilieren von Dateiinhalten.....	129
RECORDFILE- und FORTH-Projektmanagement.....	130
Speichern Sie RECORDFILE in der Datei autoexec.fs.....	130
Verwenden Sie geänderte Inhalte der Datei autoexec.fs.....	132
Ein Projekt mit ESP32forth aufschlüsseln.....	132
Beispielprojekt.....	133
Die Vorstellung einer Black Box.....	135
Das SPIFFS-Dateisystem.....	138
Zugriff auf das SPIFFS-Dateisystem.....	138
Umgang mit Dateien.....	139
Abschluss.....	140
Bearbeiten und Verwalten von Quelldateien für ESP32forth.....	141
Editoren für Textdateien.....	141
Verwenden Sie eine IDE.....	142
Speicherung auf GitHub.....	144
Einige gute Praktiken.....	145
Die main.fs-Datei.....	146
Eine Ampel mit ESP32 managen.....	148
GPIO-Ports auf der ESP32-Karte.....	148
Montage der LEDs.....	148
Verwaltung von Ampeln.....	150
Abschluss.....	150
Direkter Zugriff auf GPIO-Register.....	152
Verwendung von Wörtern m! und M@.....	152
Das GPIO_OUT_REG-Register.....	155
Aktivierungs- und Deaktivierungsregister.....	156
Hardware-Interrupts mit ESP32forth.....	160
Unterbrechungen.....	160
Montage eines Druckknopfes.....	160
Softwarekonsolidierung des Interrupts.....	161
Weitere Informationen.....	162
Verwendung des Drehgebers KY-040.....	164
Encoder-Übersicht.....	164
Montage des Encoders auf dem Steckbrett.....	165
Analyse von Encodersignalen.....	166

Encoder-Programmierung.....	167
Testen der Kodierung.....	168
Erhöhen und dekrementieren Sie eine Variable mit dem Encoder.....	168
Blinken einer LED pro Timer.....	170
Erste Schritte mit der FORTH-Programmierung.....	170
Blinken nach TIMER.....	171
Hardware- und Software-Interrupts.....	172
Verwenden Sie die Wörter intervall und rerun.....	173
Haushälterin-Timer.....	175
Präambel.....	175
Eine Lösung.....	175
Ein FORTH-Timer für ESP32Forth.....	176
Verwaltung der Licht-Ein-Taste.....	177
Abschluss.....	179
Software-Echtzeituhr.....	180
Verwalten einer Softwareuhr.....	180
Messen der Ausführungszeit eines FORTH-Wortes.....	182
Messung der Leistung von FORTH-Definitionen.....	182
Ein paar Schleifen testen.....	183
Programmieren Sie einen Sonnenscheinanalysator.....	184
Präambel.....	184
Das Miniatur-Solarpanel.....	184
Wiederherstellung eines Miniatur-Solarpanels.....	184
Messung der Solarpanelspannung.....	185
Messung des Solarpanelstroms.....	186
Senkung der Solarpanelspannung.....	186
Programmierung des Solaranalysators.....	187
Verwalten der Aktivierung und Deaktivierung eines Geräts.....	189
Ausgelöst durch Timer-Interrupt.....	190
Vom Sonnenscheinsensor gesteuerte Geräte.....	191
Installieren der OLED-Bibliothek für SSD1306.....	193
Die I2C-Schnittstelle auf ESP32.....	195
Einführung.....	195
Master-Slave-Austausch.....	196
Adressierung.....	197
GPIO-Ports für I2C einstellen.....	198
I2C-Busprotokolle.....	198
Erkennen eines I2C-Geräts.....	198
Das SSD1306 OLED-Display.....	200
Auswahl einer Anzeigeschnittstelle.....	200
Online-Dokumentation.....	201
Anschließen des SSD1306 OLED-Displays.....	201
Gedächtnisorganisation.....	202
Organisieren Sie das SSD1306-Projekt.....	203
Erstellen der Datei main.fs.....	203

Erstellen der Datei config.fs.....	203
Erstellen der Datei oledTools.fs.....	204
Testen Sie unser SSD1306-Projekt.....	204
Verwenden Sie OLED-Vokabular.....	206
Initialisieren des I2C-Busses für das SSD1306 OLED-Display.....	206
Initialisierung der Anzeige für SSD1306.....	207
Erweitern Sie den OLED-Wortschatz.....	209
TEMPVS FVGIT.....	211
Romani non ustulo nulla.....	211
Romani horas et minuta.....	212
Haec Omnia Integramus pro ESP32forth.....	213
Fügen Sie die SPI-Bibliothek hinzu.....	215
Änderungen an der Datei ESP32forth.ino.....	215
Erste Modifikation.....	215
Zweite Modifikation.....	216
Dritte Modifikation.....	216
Vierte Modifikation.....	216
Kommunizieren Sie mit dem Anzeigemodul MAX7219.....	217
Suchen des SPI-Ports auf der ESP32-Karte.....	218
SPI-Anschlüsse am MAX7219-Anzeigemodul.....	218
SPI-Port-Softwareschicht.....	219
Installieren des HTTP-Clients.....	221
Bearbeiten der Datei ESP32forth.ino.....	221
HTTP-Client-Tests.....	222
Verwaltung von digitalen/analogen Ausgängen.....	225
Digital/Analog-Wandlung.....	225
D/A-Wandlung mit R2R-Schaltung.....	225
D/A-Wandlung mit ESP32.....	225
Möglichkeiten der D/A-Wandlung.....	227
Rufen Sie die Uhrzeit von einem WEB-Server ab.....	228
Senden und Empfangen der Uhrzeit von einem Webserver.....	228
Verständnis der Übertragung per GET an einen WEB-Server.....	230
Übertragung von Daten an einen Server per GET.....	230
Parameter in einer URL.....	230
Übergabe mehrerer Parameter.....	230
Verwalten der Parameterübergabe mit ESP32forth.....	231
Datenübertragung an einen WEB-Server.....	233
Datenaufzeichnung auf der Webserverseite.....	233
Zugangsschutz.....	233
Aufgezeichnete Daten anzeigen.....	234
Fügen Sie die zu übertragenden Daten hinzu.....	235
Abschluss.....	237
Klangsynthese mit ESP32Forth.....	238
Einfache Klangsynthese.....	238
Definition der Schallfrequenztabelle.....	238

Abrufen der Frequenz einer Musiknote.....	240
Verwalten der Notendauer.....	240
One-Note-Unterstützung.....	241
Musiknoten erstellen.....	242
Punktetest.....	243
Der Flug der Hummel.....	243
Programm im XTENSA-Assembler.....	246
Präambel.....	246
Kompilieren Sie den XTENSA-Assembler.....	247
Programmierung in Assembler.....	248
Zusammenfassung der grundlegenden Anweisungen.....	248
Ein Bonus-Disassembler.....	249
Erste Schritte im XTENSA-Assembler.....	251
Präambel.....	251
Aufrufen des Xtensa-Assemblers.....	251
Xtensa und der FORTH-Stack.....	251
Schreiben einer Xtensa-Makroanweisung.....	252
Verwalten des FORTH-Stacks im Xtensa-Assembler.....	254
Effizienz der im XTENSA-Assembler geschriebenen Wörter.....	256
Schleifen und Verbindungen im XTENSA-Assembler.....	258
Die LOOP-Anweisung im XTENSA-Assembler.....	258
Verwalten Sie eine Schleife im XTENSA-Assembler mit ESP32forth.....	259
Definieren von Makroanweisungen für die Schleifenverwaltung.....	259
Mit den Makros „For“ und „Next“.....	260
Verbindungsanweisungen im XTENSA-Assembler.....	260
Verzweigungsmakros definieren.....	260
Syntax verzweigter Makroanweisungen.....	261
Definition und Manipulation von Registern.....	263
Definition von Registern.....	263
Zugriff auf Registerinhalte.....	264
Umgang mit Registerbits.....	265
Definition von Masken.....	265
Wechsel von der C-Sprache zur FORTH-Sprache.....	267
Der Zufallszahlengenerator.....	269
Charakteristisch.....	269
Programmiervorgang.....	270
RND-Funktion im XTENSA-Assembler.....	270
Das LoRa-Übertragungssystem.....	272
Verkabelung des LoRa REYAX LR890-Senders.....	272
Der LoRa-Sender für ESP32.....	272
LoRa-Übertragungssicherheit.....	273
Testbericht zum REYAX RYLR890 LoRa-Sender.....	275
Erforderliche Testumgebung.....	275
Bereiten Sie die Kommunikation mit dem LoRa-Sender vor.....	275
Einrichten des REYAX RYLR890 LoRa-Senders.....	278

Wesentliche Parameter.....	278
ADRESSE Definiert die Moduladresse.....	279
AT Testen Sie die LoRa-Verfügbarkeit.....	280
BAND Einstellen der HF-Frequenz.....	280
CPIN Legt das AES128-Netzwerkkenwort fest.....	280
CRFOP Wählt die Ausgangs-HF-Leistung aus.....	281
FACTORY Setzt alle aktuellen Einstellungen auf Standardwerte.....	281
IPR Legt die UART-Baudrate fest.....	282
MODE Wählt den Arbeitsmodus aus.....	282
NETWORKID Wählt die Netzwerk-ID aus.....	283
PARAMETER-Definition von RF-Parametern.....	283
Software-RESET.....	284
SEND sendet Daten an die angegebene Adresse.....	285
VER, um die Firmware-Version anzufordern.....	285
Fehlerergebniscodes.....	285
Vektorisierung von Zeichenemissionen.....	286
Vektorisierung in FORTH verstehen.....	286
Vektorisierung in ESP32Forth.....	287
Vektorisieren Sie den Typ auf den seriellen UART2-Port.....	288
Umschreiben einer kompletten Auflistung.....	289
LoRa-Sender einrichten.....	291
Ermittlung der Adresse von LoRa-Sendern.....	292
Kommunikation zwischen zwei LoRa REYAX RYLR890-Sendern.....	294
Übertragung von BOSS zu SLAV2.....	294
Schnittstelle einer LoRa-Übertragung mit ESP32Forth.....	297
Das LoRa-Senderseitenprogramm namens BOSS.....	298
Empfang und Ausführung von FORTH-Befehlen durch SLAV1.....	299
Ausführen eines von LoRa empfangenen Befehls.....	300
LoRa-Übertragungsverwaltungsschleife.....	301
Eine einfache WEB-Schnittstelle für ESP32Forth.....	304
Detaillierter Inhalt der ESP32forth-Vokabulare.....	308
Version v 7.0.7.21a.....	308
FORTH.....	308
asm.....	309
bluetooth.....	310
editor.....	310
ESP.....	310
espnw.....	310
httpd.....	310
insides.....	310
internals.....	310
interrupts.....	311
ledc.....	311
oled.....	311
recognizers.....	311
registers.....	312

riscv.....	312
rmt.....	312
rtos.....	312
SD.....	312
SD_MMC.....	312
Serial.....	312
sockets.....	313
spi.....	313
SPIFFS.....	313
streams.....	313
structures.....	313
tasks.....	313
telnetd.....	313
timers.....	313
visual.....	314
web-interface.....	314
WiFi.....	314
Wire.....	314
xtensa.....	314
Ressourcen.....	316
Auf Englisch.....	316
Auf Französisch.....	316
GitHub.....	316

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

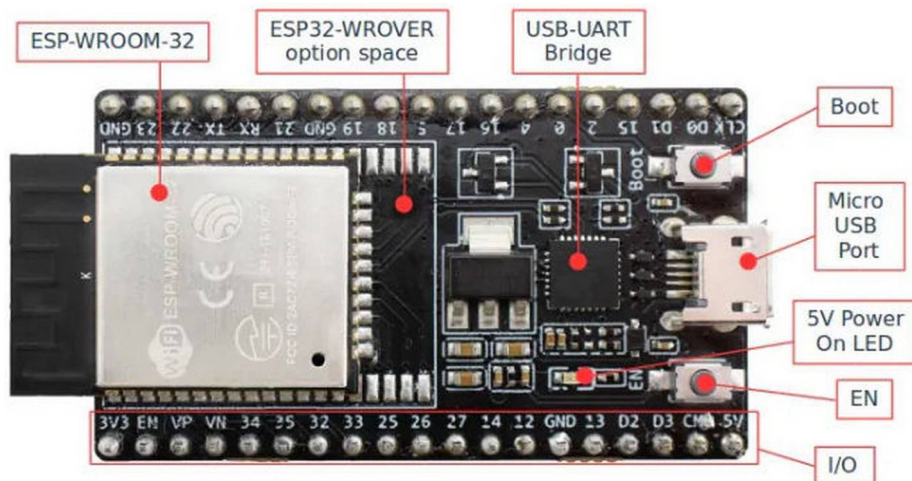
Entdeckung der ESP32-Karte

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

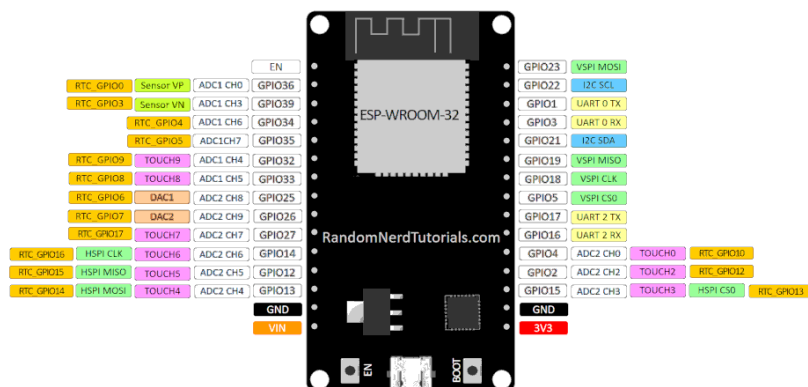
Hier im Foto die ESP32-Karte, anhand derer wir die Rolle der verschiedenen



Figure 1: position des E/S GPIO

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.

Die verschiedenen ESP32-Karten

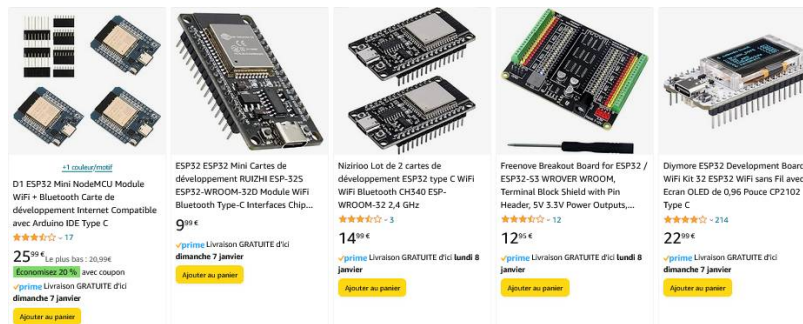


Bild 1: Eine große Auswahl an ESP32 Modulen AMAZON

Wenn Sie auf einer Online-Verkaufsseite eine ESP32-Karte bestellen, steht Ihnen möglicherweise eine sehr große Auswahl an Boards und Modulen zur Verfügung.

Daher stellen sich bei der Auswahl mehrere Fragen:

- Welches Board funktioniert mit ESP32forth?
- Welche Boards eignen sich am besten für meine Projekte?
- Wie hoch ist mein Budget für ein bestimmtes Projekt?

Während der Preis eines gewöhnlichen ESP32-Boards erschwinglich bleibt, können bestimmte Varianten deutlich teurer sein. Wenn Ihr Ziel darin besteht, zunächst kleine Experimente durchzuführen, beginnen Sie mit einem einfachen ESP32-Board. Um richtig experimentieren zu können, benötigen Sie:

- Steckbretter (breadboard), nehmen Sie mindestens 10. Rechnen Sie mit zwei Steckbrettern pro ESP32-Karte;
- flexible Steckverbinder vom Typ DuPont;
- LEDs, Widerstände usw.
- Peripheriegeräte: OLED-Display, LCD, Relais, Synchron- oder Normalmotoren, Servomotoren usw.
- USB-Kabel, das den PC und die ESP32-Karte verbindet. Ein USB-Hub wird empfohlen. Im Falle einer versehentlichen Stromeinspeisung in den USB-Anschluss wird der USB-Anschluss des Hubs anstelle des USB-Anschlusses des PCs beschädigt;

Für rund fünfzig Euro (oder US-Dollar) gibt es fertige Kits, die eine ESP32-Karte sowie Peripheriegeräte und Komponenten enthalten.

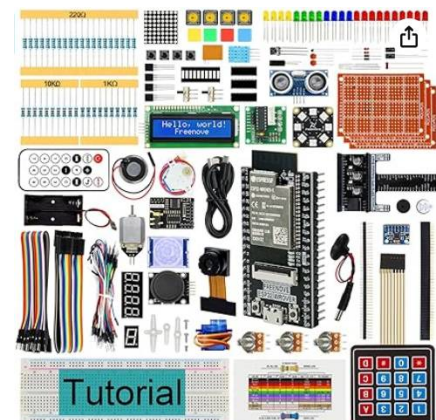


Bild 2: ESP32-Kit

Kein Bausatz ist vollständig. Wenn Sie Experimente durchführen, bestellen Sie am besten ein Kit, dann mehrere ESP32-Karten (mindestens 4), eine Reihe von Steckbrettern, Lochplatinen, ein Batterie-Netzteil usw. ...

Bevor Sie ehrgeizige Projekte wie Fernsteuerung über 3G/4G/5G-Netzwerk, Videoanalyse usw. in Angriff nehmen, beginnen Sie mit einfachen Experimenten in C oder FORTH.

Endgültige Installation von ESP32forth

NEIN ! Die Installation von ESP32forth auf einer ESP32-Karte **ist nicht dauerhaft !**

Wenn Sie ESP32Forth auf einem oder mehreren ESP32-Boards installiert haben, können Sie jederzeit Binärcode von einer beliebigen C-Quelle herunterladen, kompilieren und auf das ESP32-Board flashen, falls Sie ihr Forthexperiment aufgeben wollen.

Auch auf die Gefahr hin, ESP32forth zu bewerben: Viele „Macher“ haben die endgültige Entscheidung getroffen, in der FORTH-Sprache zu programmieren. Nur ein Beispiel, der YouTube-Kanal von **0033mer** :

<https://www.youtube.com/@0033mer>

Er ist einer der produktivsten FORTH-Mitwirkenden auf YouTube. Obwohl er ESP32 nur sehr selten verwendet, verwenden die meisten seiner Beiträge die FORTH-Sprache.

Das Programmieren in der FORTH-Sprache erfordert intellektuelle Anstrengung. Diese Bemühungen sind nicht umsonst, denn sie führen zu bestimmten guten Praktiken, die in anderen Programmiersprachen verwendet werden können.

FORTH ist die einzige Programmiersprache, die auf einer MCU installiert werden kann und einen Interpreter, einen Compiler, einen Assembler sowie beim ESP32 ein SPIFFS-Dateisystem integriert, das alles bietet Ihnen einen erheblichem Entwicklungsspielraum.

Ein ESP32-Board ist eines der sehr seltenen Boards, das auch über serielle Schnittstellen (UART0-Port über den USB-Anschluss) über WLAN oder Bluetooth verfügt. Die meisten ESP32-Boards verfügen zudem über zahlreiche vielseitige GPIO-Ports: Logik, Analog, PWM, UART, SPI, I2C-Ein- und -Ausgang usw. Und das alles mit einem oder zwei Prozessoren mit fast 160 MHz, also 10x schneller als auf einer gewöhnlichen ARDUINO-Karte. .

Und schließlich können die meisten C-Bibliotheken für ARDUINO auf ESP32 verwendet werden. Einige sind über ESP32forth zugänglich.

Die ESP32 Wroom 32-Karte

ESP32 Wroom ist das neueste Mitglied der Familie der ESP-Karten von Espressif. Hierbei handelt es sich um eine besonders modische Reihe von Entwicklungsboards, da sie aufgrund ihres niedrigen Preises, ihres geringen Verbrauchs und ihrer geringen Größe ein ideales Produkt für die Durchführung kleiner IoT-Projekte sind.

- ESP-WROOM-32-Prozessor
- WLAN 802.11 b/g/n
- Bluetooth 4.2 / BLE
- Tensilica L108 32-Bit 160 MHz Prozessor
- 512 KB SRAM und 16 MB Flash-Speicher
- 32 digitale I/O-Pins (3,3 V)
- 6 Analog-Digital-Pins
- 3x UART, 2x SPI, 2x I2C
- USB-zu-UART-CP2102-Schnittstelle

Wenn Sie ESP32forth für dieses Board kompilieren, sind folgende Parameter auf der ARDUINO IDE zu berücksichtigen: TOOLS → BOARD → ESP32 → ESP32 Dev Module:

- **Planke: Partitionsschema des ESP32-Entwicklungsmoduls :**

Kein OTA (2M APP, 2M SPIFFS) ← **Nicht standardmäßige**

Upload-Geschwindigkeit: 921600

CPU-Frequenz: 240 MHz

Flash-Frequenz: 80 MHz

Flash-Modus: QIO

Flash-Größe: 4 MB (32 MB)

Core-Debug-Level: Keine

PSRAM: Deaktivieren

Anschlussplatine

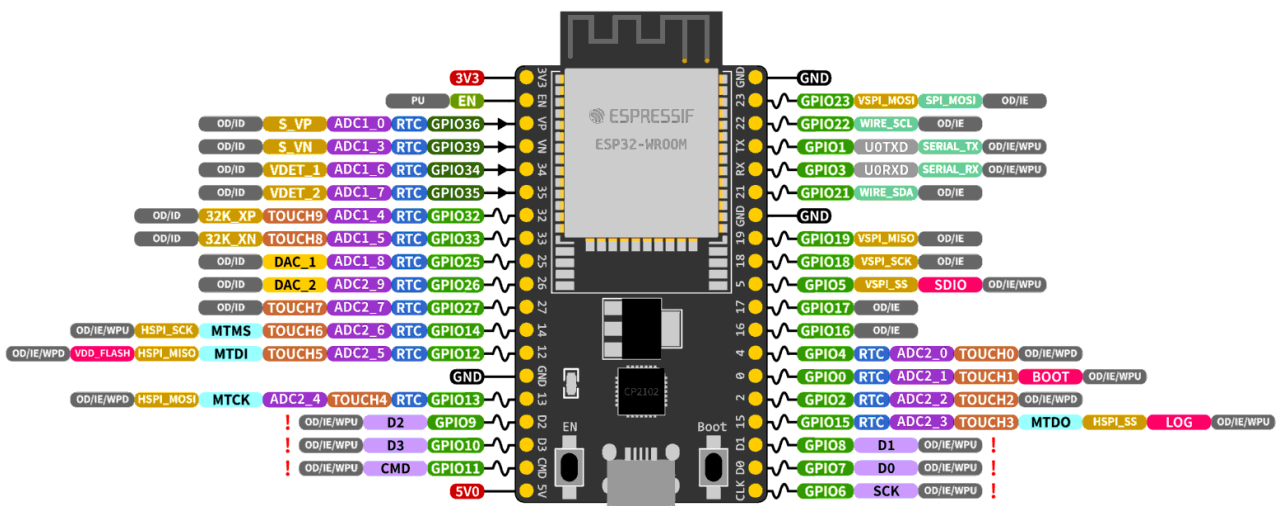


Figure 3: ESP32 Wroom 32 karte

Das ESP32 Wrover-Karte

Die ESP32-WROVER MCU-Module von Espressif Systems sind leistungsstarke und generische Wi-Fi/BT/BLE-MCU-Module, die auf eine Vielzahl von Anwendungen abzielen.

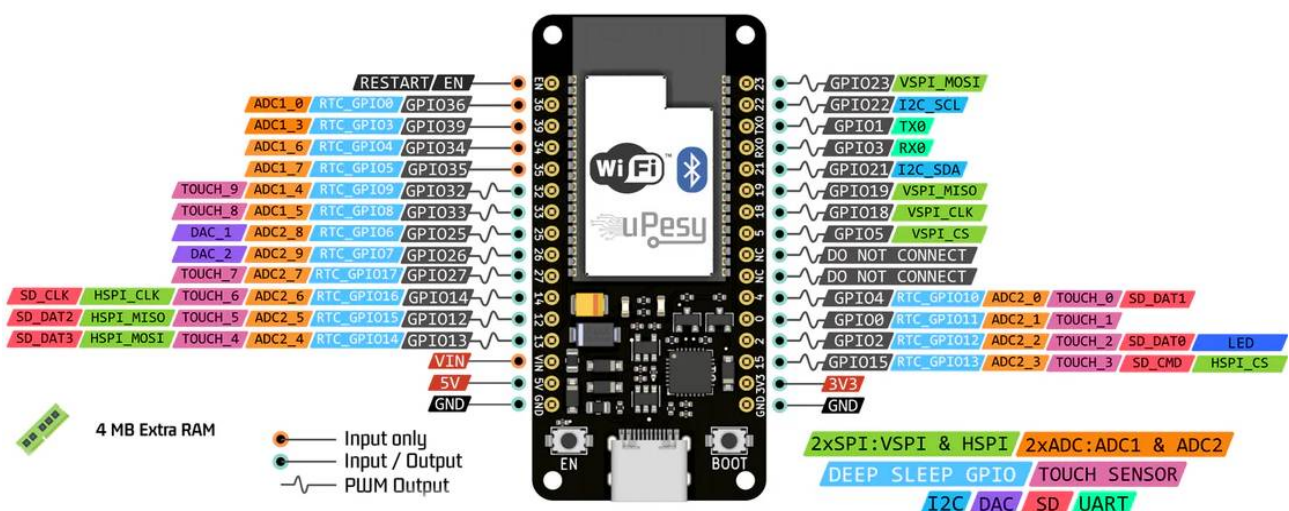
Diese Module zielen auf Anwendungen ab, die von Sensornetzwerken mit geringem Stromverbrauch bis hin zu anspruchsvollsten Aufgaben wie Sprachkodierung, Musik-Streaming und MP3-Dekodierung reichen.

Das ESP32-WROVER-Modul verwendet eine PCB-Antenne, während das ESP32-WROVER-I eine IPEX-Antenne verwendet. Diese Module verfügen über einen externen 4-MB-SPI-Flash, einen externen 4-MB-PSRAM und einen 32-Mbit-SPI-PSRAM.

Wenn Sie ESP32forth für dieses Board kompilieren, sind folgende Parameter auf der ARDUINO IDE zu berücksichtigen: TOOLS → BOARD → ESP32 → ESP32 Dev Module:

- **Planke: Partitionsschema des ESP32-Entwicklungsmoduls :**
Kein OTA (2M APP, 2M SPIFFS) ← Nicht standardmäßige
Upload-Geschwindigkeit: 921600
CPU-Frequenz: 240 MHz
Flash-Frequenz: 80 MHz
Flash-Modus: QIO
Flash-Größe: 4 MB (32 MB)
Core-Debug-Level: Keine
PSRAM: Ermöglicht

Anschlussplatine



Die ESP32 S3-Karte

ESP32-S3 ist ein Entwicklungsboard, das auf einem Espressif ESP32-S3-WROOM-2-Mikrocontroller mit WiFi- und Bluetooth Low Energy-Schnittstellen basiert.

- Xtensa LX7 Dual-Core-32-Bit-Mikroprozessor
PSRAM-Speicher: 8 MB
SRAM-Speicher: 512 KB
ROM-Speicher: 384 KB
SRAM-Speicher (RTC): 16 KB SPI
FLASH-Speicher: 32 MB
WiFi-Schnittstelle: 802.11 b/g/n 2,4 GHz
BLE 5 Mesh-Schnittstelle

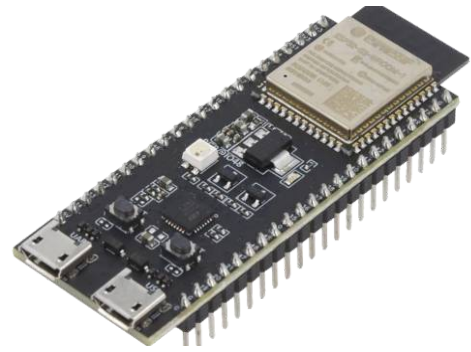


Figure 5: carte ESP32 S3

Wenn Sie ESP32forth für dieses Board kompilieren, sind folgende Parameter auf der ARDUINO IDE zu berücksichtigen: TOOLS → BOARD → ESP32 → ESP32 Dev Module:

- **Planke: Partitionsschema** des ESP32S2-Entwicklungsmoduls :
Kein OTA (2M APP, 2M SPIFFS) ← **Nicht standardmäßige**
Upload-Geschwindigkeit: 921600
USB-CDC beim Booten: Deaktiviert
USB-Firmware-MSC beim Booten: Deaktiviert
USB-DFU beim Booten: Deaktiviert
Upload-Modus: UART0
CPU-Frequenz: 240 MHz
Flash-Frequenz: 80 MHz
Flash-Modus : QIO-
Flash-Größe: 4 MB (32 MB)
Core-Debug-Level: Keine
PSRAM: Ermöglicht

Anschlussplatine

Installieren Sie ESP32Forth

ESP32forth herunterladen

Der erste Schritt besteht darin, den in C geschriebenen Quellcode von ESP32forth abzurufen. Es empfiehlt sich, die neueste Version zu verwenden:

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

Inhalt der heruntergeladenen Datei:

ESP32forth-7.0.7.2x

ESP32forth

readme.txt

esp32forth.ino

optional

SPI-flash.h

serial-blueooth.h

... usw...

ESP32forth kompilieren und installieren

Entpacken Sie die Datei **esp32forth.ino** in ein Arbeitsverzeichnis. Das **optionale Verzeichnis** enthält Dateien, die die ESP32forth-Erweiterung aktivieren. Für die erste Kompilierung und den ersten Upload von ESP32forth werden diese Dateien nicht benötigt.

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

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

Sobald die Arduino IDE installiert ist, starten Sie sie. Die Arduino IDE wird geöffnet. Klicken Sie auf *file* und wählen Sie *Preferences* :

Gehen Sie im daraufhin angezeigten Fenster zum Eingabebereich mit der Bezeichnung

Gehen Sie im daraufhin angezeigten Fenster zum Eingabebereich mit der Bezeichnung *Additional boards manager URLs:* und geben Sie folgende Zeile ein:

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

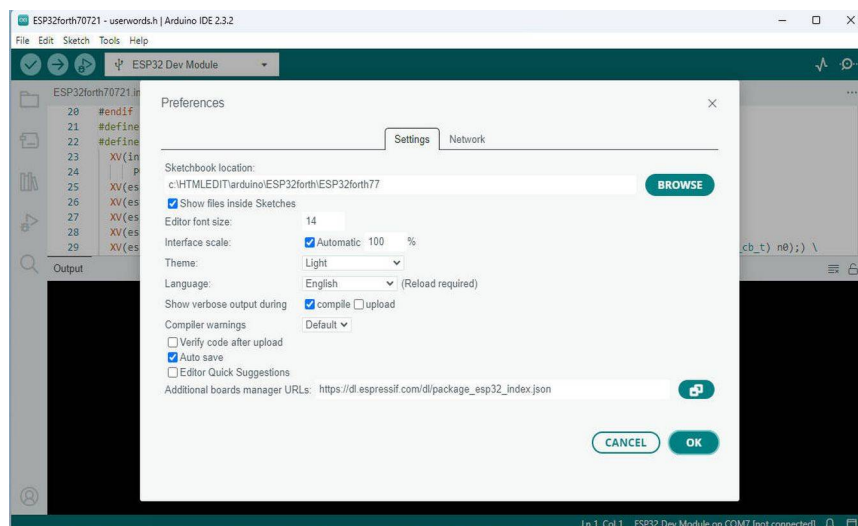


Figure 7: URL-Eingabe Zusätzliche Kartenverwaltung

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

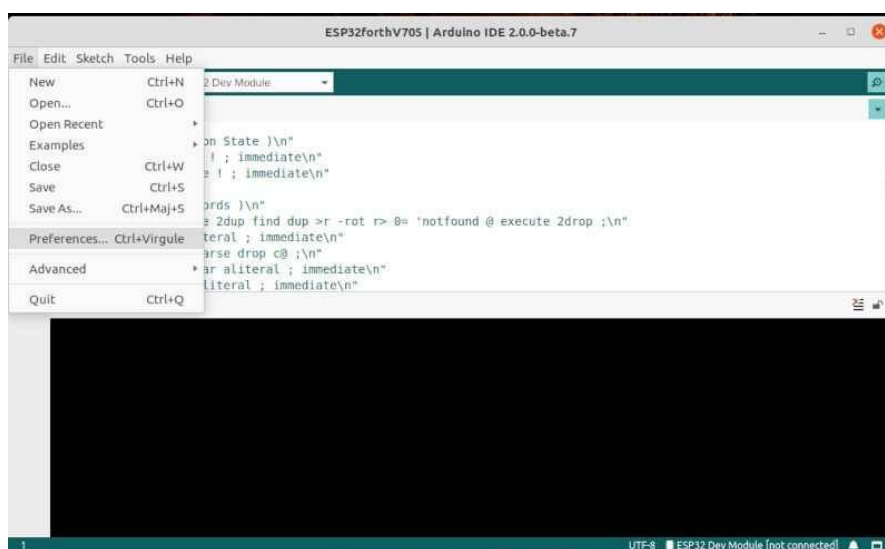


Figure 6: Anpassungen der Vorrechte

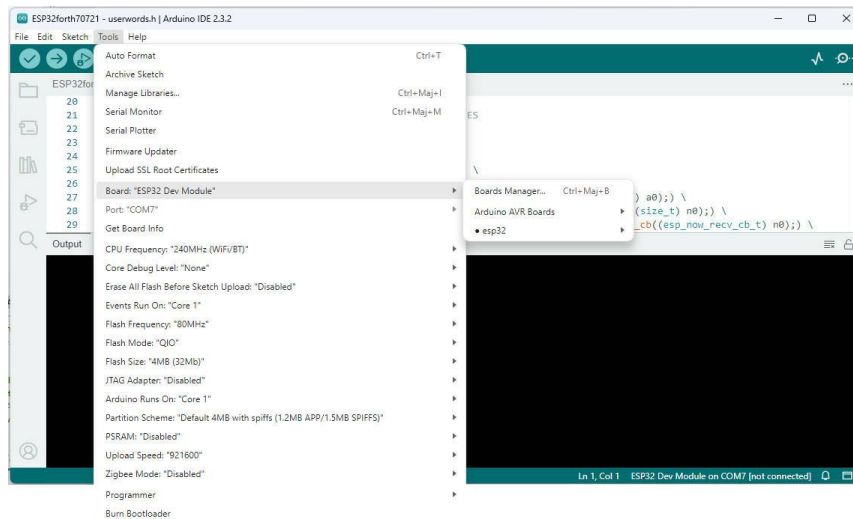


Figure 8: ESP32-Kartenauswahl

Diese Auswahl sollte Sie zur Installation der Pakete für ESP32 auffordern. Akzeptieren Sie diese Installation.

Anschließend sollten Sie auf die ESP32-Kartenauswahl zugreifen können.

Auswahl **des ESP32-Entwicklungsmodul- Boards** :

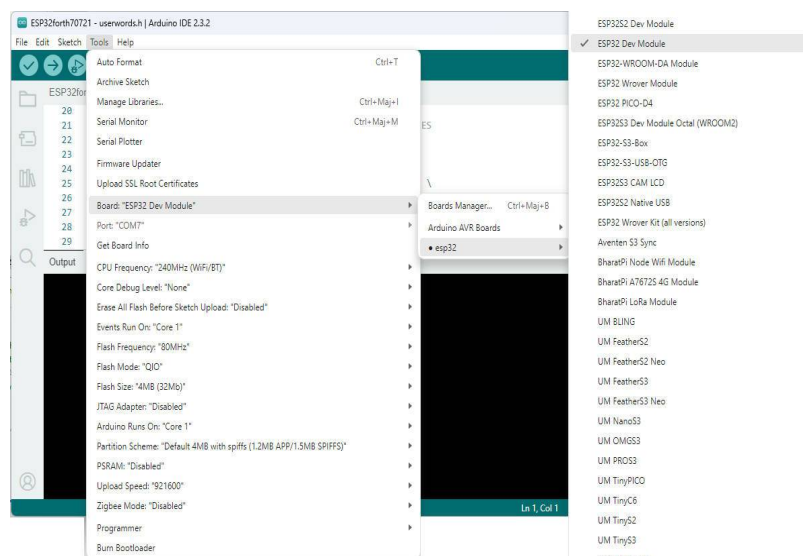


Figure 9: Auswahl des ESP32 Dev Module Boards

Parameter für ESP32 WROOM 32

Hier sind die weiteren Einstellungen, die vor dem Kompilieren von ESP32forth erforderlich sind. Sie erreichen die Einstellungen, indem Sie *erneut auf „Tools“ 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 Kompilation.

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

Wir gehen davon aus, dass Ihr ESP32-Board an einen USB-Anschluss angeschlossen ist. Starten Sie die Kompilierung, indem Sie auf *Sketch klicken* und *Upload auswählen*.

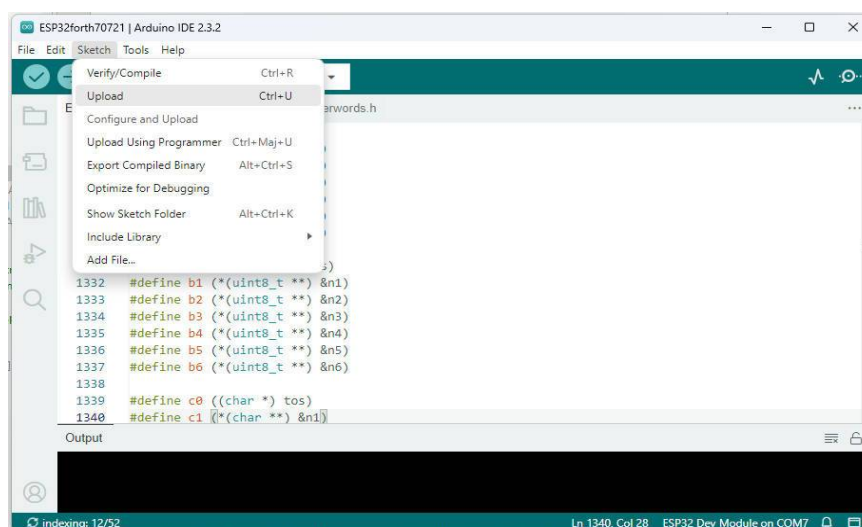


Figure 10: Upload gestartet

Wenn alles reibungslos verläuft, sollte der Binärcode automatisch auf das ESP32-Board übertragen werden. Falls die Kompilierung fehlerfrei abgeschlossen wurde, aber ein Übertragungsfehler auftritt, kompilieren Sie die Datei **esp32forth.ino** erneut. Drücken Sie während der Übertragung die mit **BOOT** gekennzeichnete Taste auf dem ESP32-Board. Dadurch wird das Board für die Übertragung des ESP32forth-Binärcodes bereit.

Video-Tutorial zur Installation und Konfiguration der ARDUINO IDE:

- 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:

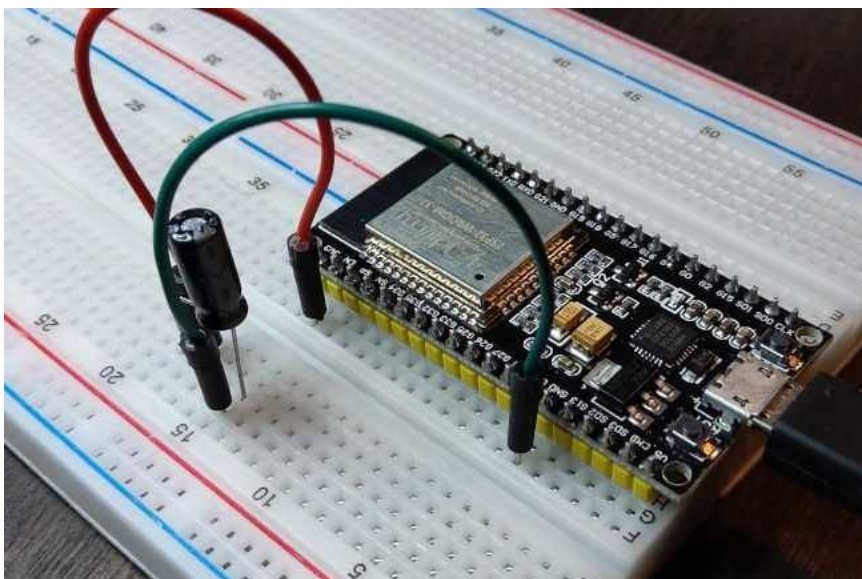


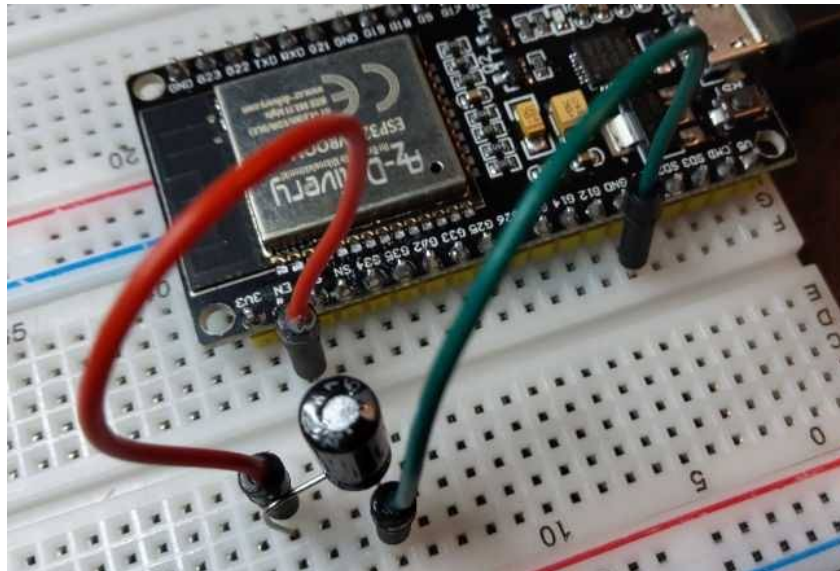
```
Blink
71 | it is attached to digital pin 13, on MKR1000 on pin 6. LED BUILTIN is set to

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 e
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

a1b1d, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 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.

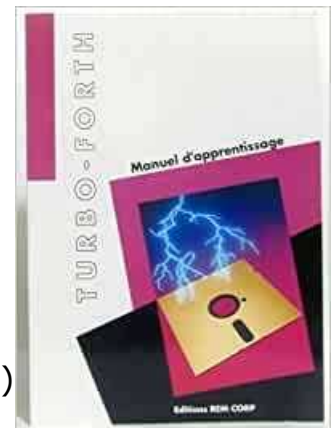
Warum auf ESP32 in FORTH-Sprache programmieren?

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 Loistech - 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!

Gute Programmierung.

Verwenden von Zahlen mit ESP32Forth

Wir haben ESP32Forth ohne Probleme gestartet. Wir werden uns nun mit einigen Manipulationen mit Zahlen befassen, um zu verstehen, wie man den Mikrocontroller in der FORTH-Sprache beherrscht.

Wie in vielen Büchern könnten wir mit einem trivialen Beispielprogramm beginnen, zum Beispiel blinkenden LEDs. So zum Beispiel:

```
\ define LEDs GPIOs
25 constant ledRED
26 constant ledYELLOW
27 constant ledGREEN

\ define masks for red yellow and green LEDs
1 ledRED      defMASK: mLED_RED
1 ledYELLOW   defMASK: mLED_YELLOW
1 ledGREEN    defMASK: mLED_GREEN

\ initialisation GPIO G25 G26 and G27 in output mode
: GPIO.init ( -- )
    1 mLED_RED      GPIO_ENABLE_REG regSet
    1 mLED_YELLOW   GPIO_ENABLE_REG regSet
    1 mLED_GREEN    GPIO_ENABLE_REG regSet
;

\ define a ON and OFF sequence
: GPIO.on.off.sequence { position mask delay -- }
    1 position mask GPIO_OUT_W1TS_REG regSet
    delay ms
    1 position mask GPIO_OUT_W1TC_REG regSet ;
```

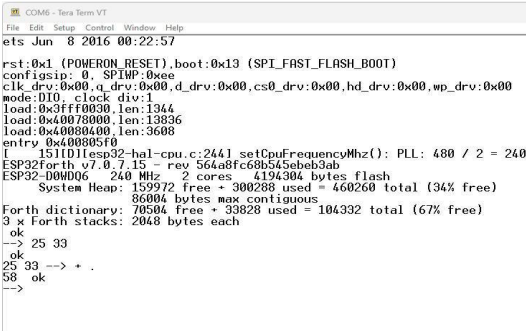
Dieser scheinbar einfache Code erfordert bereits eine Wissensbasis, beispielsweise die Vorstellung von Speicheradresse, Register, Binärmasken und Hexadezimalzahlen.

Wir werden uns daher zunächst mit diesen Grundbegriffen befassen und Sie dazu einladen, einfache Manipulationen vorzunehmen.

Zahlen mit dem FORTH-Interpreter

Wenn ESP32Forth startet, sollte das TERA TERM-Terminalfenster (oder ein anderes Terminalprogramm Ihrer Wahl) anzeigen, dass ESP32Forth verfügbar ist. Drücken Sie ein- oder zweimal die *ENTER-Taste auf der Tastatur*. ESP32Forth antwortet mit einer Bestätigung der erfolgreichen Ausführung **ok**.

Wir werden die Eingabe von zwei Zahlen testen, hier **25** und **33**. Geben Sie diese Zahlen ein und geben Sie dann *die EINGABETASTE* auf der Tastatur ein. ESP32Forth antwortet immer mit **ok**. Sie haben gerade zwei Zahlen auf dem ESP32Forth-Sprachstapel



```
CDM6 - Tera Term VT
File Edit Setup Control Window Help
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configisp: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13836
load:0x40080400,len:3608
entry 0x400805f0
[ 1510]!esp32-hal-cpu.c:244: setCpuFrequencyMhz(): PLL: 480 / 2 = 240
ESP32Forth v7.0.7.15 - rev 564a8fc68b545eb3ab
ESP32-D0W0Q6 240 MHz 2 cores 4194304 bytes flash
System Heap: 159972 free + 300288 used = 460260 total (34% free)
86804 bytes max contiguous
Forth dictionary: 70504 free + 33828 used = 104332 total (67% free)
3 x Forth stacks: 2048 bytes each
ok
--> 25 33
ok
25 33 --> + .
58 ok
-->
```

gestapelt. Geben Sie nun **+** ein. Drücken Sie dann die *ENTER- Taste* . ESP32Forth zeigt das Ergebnis an:

Diese Operation wurde vom FORTH-Interpreter verarbeitet.

ESP32Forth hat, wie alle Versionen der FORTH-Sprache, zwei Zustände:

- **Interpreter** : der Zustand, den Sie gerade getestet haben, indem Sie eine einfache Summe zweier Zahlen gebildet haben;
- **Compiler** : ein Zustand, der die Definition neuer Wörter ermöglicht. Auf diesen Aspekt wird später noch näher eingegangen.

Eingeben von Zahlen mit unterschiedlichen Zahlenbasen

Um die Erläuterungen vollständig zu verstehen, sind Sie herzlich eingeladen, alle Beispiele über das TERA TERM-Terminalfenster zu testen.

Zahlen können natürlich eingegeben werden. Im Dezimalformat handelt es sich IMMER um eine Folge von Zahlen, Beispiel:

```
-1234 5678 + .
```

Das Ergebnis dieses Beispiels wird **4444** sein. FORTH-Zahlen und -Wörter müssen durch mindestens ein *Leerzeichen getrennt sein* . Das Beispiel funktioniert perfekt, wenn Sie pro Zeile eine Zahl oder ein Wort eingeben:

```
-1234
5678
+
.
```

Zahlen können vorangestellt werden, wenn Sie Werte in anderer als der Dezimalform eingeben möchten:

- **\$** -Zeichen , um anzuzeigen, dass es sich bei der Zahl um einen Hexadezimalwert handelt;

Beispiel :

```
255 .      \ display 255
$ff .      \ display 255
```

Der Zweck dieser Präfixe besteht darin, Interpretationsfehler bei ähnlichen Werten zu vermeiden:

```
$0305
0305
```

sind keine **gleichen Zahlen**, wenn die hexadezimale Zahlenbasis nicht explizit definiert ist !

Änderung der Zahlenbasis

ESP32Forth verfügt über Wörter, mit denen Sie die Zahlenbasis ändern können:

- **hex** , um die hexadezimale numerische Basis auszuwählen;
- **binary** , um die binäre Zahlenbasis auszuwählen;
- **decimal** , um die dezimale numerische Basis auszuwählen.

Jede in eine numerische Basis eingegebene Zahl muss die Syntax der Zahlen in dieser Basis respektieren:

```
3E7F
```

führt zu einem Fehler, wenn Sie im Dezimalsystem arbeiten.

```
hex 3e7f
```

funktioniert perfekt im Hexadezimalformat. Die neue Zahlenbasis bleibt gültig, solange nicht eine andere Zahlenbasis ausgewählt wird:

```
hex  
$0305  
0305
```

sind gleiche Zahlen !

Sobald eine Zahl in einer numerischen Basis auf dem Datenstapel abgelegt wird, ändert sich ihr Wert nicht mehr. Wenn Sie beispielsweise den Wert **\$ff** auf dem Datenstapel ablegen, ändert sich dieser Wert, der dezimal **255** oder binär **11111111** ist, nicht, wenn wir zur Dezimalzahl zurückkehren:

```
hex ff decimal . \ display: 255
```

Auch wenn ich darauf beharre, dass **255** im Dezimalformat **derselbe Wert ist** wie **\$ff** im Hexadezimalformat!

Im Beispiel zu Beginn des Kapitels definieren wir eine Konstante im Hexadezimalformat:

```
25 constant ledRED
```

Wenn wir Folgendes eingeben:

```
hex ledRED .
```

Dadurch wird der Inhalt dieser Konstante in hexadezimaler Form angezeigt. Der Standortwechsel hat **keine Auswirkungen** auf den endgültigen Betrieb des FORTH-Programms.

Binär und hexadezimal

Das moderne binäre Zahlensystem, die Grundlage des Binärcodes, wurde 1689 von Gottfried Leibniz erfunden und erscheint 1703 in seinem Artikel „Erklärung der binären Arithmetik“.

In seinem Artikel verwendet LEIBNITZ zur Beschreibung aller Zahlen ausschließlich die Zeichen **0** und **1** :

```
: bin0to15 ( -- )
  binary
  $10 0 do
    cr i .
  loop
  cr decimal ;
bin0to15 \ display:
0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
```

Ist es notwendig, die binäre Codierung zu verstehen? Ich werde ja und nein sagen. **Nein** für den täglichen Gebrauch. **Ja**, um die Programmierung von Mikrocontrollern und die Beherrschung logischer Operatoren zu verstehen.

Es war Georges Boole, der die Logik offiziell beschrieb. Seine Arbeit geriet bis zum Erscheinen der ersten Computer in Vergessenheit. Es war Claude Shannon, der erkannte, dass diese Algebra beim Entwurf und der Analyse elektrischer Schaltkreise angewendet werden konnte.

Die Boolesche Algebra beschäftigt sich ausschließlich mit **0** und **1**.

Die grundlegenden Komponenten aller unserer Computer und digitalen Speicher verwenden binäre Codierung und Boolesche Algebra.

Die kleinste Speichereinheit ist das Byte. Es ist ein Raum, der aus 8 Bits besteht. Ein Bit kann nur zwei Zustände haben: **0** oder **1**. Der kleinste Wert, der in einem Byte gespeichert werden kann, ist **00000000**, der größte ist **11111111**. Wenn wir ein Byte in zwei Teile schneiden, erhalten wir:

- vier niederwertige Bits, die die Werte **0000** bis **1111** annehmen können ;

- vier höchstwertige Bits, die einen dieser gleichen Werte annehmen können.

Wenn wir alle Kombinationen zwischen 0000 und 1111, beginnend bei 0, durchnummerieren, kommen wir auf 15:

```
: bin0to15 ( -- )
  binary
  $10 0 do
    cr i .
    i hex . binary
  loop
  cr decimal ;
bin0to15 \ display:
0 0
1 1
10 2
11 3
100 4
101 5
110 6
111 7
1000 8
1001 9
1010 A
1011 B
1100 C
1101 D
1110 E
1111 F
```

Im rechten Teil jeder Zeile zeigen wir den gleichen Wert wie im linken Teil an, jedoch hexadezimal: **1101** und **D** sind die gleichen Werte!

Aus praktischen Gründen wurde die hexadezimale Darstellung zur Darstellung von Zahlen in der Informatik gewählt. Für den höchst- oder niedrigstwertigen Teil eines Bytes, auf 4 Bits, liegen die einzigen Kombinationen der hexadezimalen Darstellung zwischen **0** und **F**. Hier **sind die Buchstaben A bis F hexadezimale Zahlen** !

```
$3E \ ist besser lesbar als 00111110
```

Die hexadezimale Darstellung bietet daher den Vorteil, den Inhalt eines Bytes in einem festen Format von **00** bis **FF** darzustellen. Im Dezimalformat sollten 0 bis 255 verwendet werden.

Größe der Zahlen im FORTH-Datenstapel

ESP32forth verwendet einen Datenstapel mit einer Speichergröße von 32 Bit oder 4 Byte (8 Bit x 4 = 32 Bit). Der kleinste Hexadezimalwert, der auf dem FORTH-Stack gestapelt

werden kann, ist **00000000** , der größte ist **FFFFFFF** . Jeder Versuch, einen größeren Wert zu stapeln, führt zu einer Beschneidung dieses Werts:

```
hex
abcdefabcdef . \ display: EFABCDEF
```

Stapeln wir den größtmöglichen Wert im 32-Bit-Hexadezimalformat (4 Byte):

```
decimal
$ffffffff . \ display: -1
```

Ich sehe Sie überrascht, aber dieses Ergebnis ist **normal** ! Wort **.** Zeigt den Wert an, der sich oben im Datenstapel in seiner vorzeichenbehafteten Form befindet. Um denselben vorzeichenlosen Wert anzuzeigen, müssen Sie das Wort **u.** verwenden :

```
$ffffffff u. \ display: 4294967295
```

Dies liegt daran, dass FORTH 32 Bits zur Darstellung einer Ganzzahl verwendet. Das höchstwertige Bit wird als Vorzeichen verwendet:

- wenn das höchstwertige Bit **0 ist** , ist die Zahl positiv;
- Wenn das höchstwertige Bit **1 ist** , ist die Zahl negativ.

Wenn Sie also richtig gefolgt sind, werden unsere Dezimalwerte 1 und -1 auf dem Stapel im Binärformat in dieser Form dargestellt:

```
binary
000000000000000000000000000000001 \ 1 auf den Stapel schieben
11111111111111111111111111111111 \ -1 auf Stapel schieben
```

Und hier bitten wir unseren Mathematiker Herrn LEIBNITZ, diese beiden Zahlen binär zu addieren. Wenn wir wie in der Schule von rechts beginnen, müssen Sie einfach diese Regel respektieren: $1 + 1 = 10$ im Binärformat. Wir tragen die Ergebnisse in eine dritte Zeile ein:

```
000000000000000000000000000000001
111111111111111111111111111111111
10
```

Nächster Schritt :

```
000000000000000000000000000000001
1111111111111111111111111111111111
10
100
```

Am Ende haben wir das Ergebnis:

```
000000000000000000000000000000001
111111111111111111111111111111111
100000000000000000000000000000000
```

Da dieses Ergebnis jedoch ein 33. höchstwertiges Bit bei 1 hat, ist das Endergebnis **0** , da das Ganzzahlformat streng auf 32 Bits beschränkt ist . Es ist überraschend ? Doch das ist

Zur Erinnerung: GPIOs ¹sind von 0 bis 31 nummeriert. Um einen anderen GPIO zu aktivieren, zum Beispiel GPIO17, hätten wir Folgendes ausgeführt:

```
1 17 lshift GPIO_ENABLE_REG !
```

Angenommen, wir möchten die GPIOs 17 und 25 in einem einzigen Befehl aktivieren. Wir führen Folgendes aus:

```
1 25 lshift
1 17 lshift or GPIO_ENABLE_REG !
```

Was haben wir getan? Hier sind die Details der Operationen:

```
\ 1 25 lshift \ %000000100000000000000000000000
\ 1 17 lshift \ %000000100000000100000000000000
\ or          \ %000000100000000100000000000000
```

Das Wort **or** hat eine Operation ausgeführt, die die beiden Offsets in einer einzigen Binärmaske kombiniert.

Kehren wir zu unserer **Score**- Variable zurück . Wir wollen das niedrigstwertige Byte isolieren. Für uns stehen mehrere Lösungen zur Verfügung. Eine Lösung verwendet die binäre Maskierung mit dem logischen Operator **and** :

```
hex
score @ .          \ display: 76C
score @
$000000FF and .    \ display: 6C
```

So isolieren Sie das zweite Byte von rechts:

```
score @
$0000FF00 and .    \ display: 0700
```

Hier hatten wir Spaß mit dem Inhalt einer Variablen. Um einen Mikrocontroller wie den auf der ESP32-Karte montierten zu beherrschen, unterscheiden sich die Mechanismen kaum. Am schwierigsten ist es, die richtigen Register zu finden. Dies wird Gegenstand eines weiteren Kapitels sein.

Zum Abschluss dieses Kapitels gibt es noch viel zu lernen über binäre Logik und die verschiedenen möglichen digitalen Kodierungen. Wenn Sie die wenigen hier aufgeführten Beispiele getestet haben, verstehen Sie sicherlich, dass FORTH eine interessante Sprache ist:

- dank seines Interpreters, der die interaktive Durchführung zahlreicher Tests ermöglicht, ohne dass eine Neukompilierung durch Hochladen von Code erforderlich ist;
- ein Wörterbuch, dessen Wörter dem Dolmetscher größtenteils zugänglich sind;
- *im Handumdrehen* neue Wörter hinzufügen und diese dann sofort testen können.

1 Allgemeine Eingabe/Ausgabe = Eingabe/Ausgabe für allgemeine Zwecke

Und schließlich ist der FORTH-Code, sobald er kompiliert ist, sicherlich genauso effizient wie sein Äquivalent in der C-Sprache, was nichts verderben soll.

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 $brot . " : " . $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....

Kommentare und Erläuterungen

Es gibt keine IDE², um in der FORTH-Sprache geschriebenen Code strukturiert zu verwalten und darzustellen. Im schlimmsten Fall verwenden Sie einen ASCII-Texteditor, bestenfalls eine echte IDE und Textdateien:

- **edit** oder **Wordpad** unter Windows
- unter Linux **edit**
- **PsPad** unter Windows
- **Netbeans** unter Windows oder Linux...

Hier ist ein Codeausschnitt, der auch von einem Anfänger geschrieben werden könnte:

```
: cycle.stop -1 +to MAX_LIGHT_TIME MAX_LIGHT_TIME 0 = if  
LOW myLIGHTS pin else 0 rerun then ;
```

Dieser Code wird von ESP32forth perfekt kompiliert. Aber bleibt es auch in Zukunft verständlich, wenn es geändert oder in einer anderen Anwendung wiederverwendet werden muss?

Schreiben Sie lesbaren FORTH-Code

Beginnen wir mit dem Namen des zu definierenden Wortes, hier **cycle.stop**. Mit ESP32forth können Sie sehr lange Wortnamen schreiben. Die Größe der definierten Wörter hat keinen Einfluss auf die Leistung der endgültigen Anwendung. Wir haben daher eine gewisse Freiheit, diese Worte zu schreiben:

- wie Objektprogrammierung in JavaScript: **cycle.stop**
- der Camel- **cycleStop** codierung
- für Programmierer, die sehr verständliche Code **cycle-stop-lights**
- **cs1**- Code mag

Es gibt keine Regel. Die Hauptsache ist, dass Sie Ihren FORTH-Code problemlos erneut lesen können. Allerdings haben Computerprogrammierer in der FORTH-Sprache bestimmte Gewohnheiten:

- Konstanten in Großbuchstaben **MAX_LIGHT_TIME_NORMAL_CYCLE**
- Wort, das andere Wörter definiert **defPin:**, d. h. Wort gefolgt von einem Doppelpunkt;
- Adresstransformationswort **>date**, hier wird der Adressparameter um einen bestimmten Wert erhöht, um auf die entsprechenden Daten zu verweisen;

² Integrierte Entwicklungsumgebung = Integrierte Entwicklungsumgebung

- Speicherwort **date@** oder **date!**
- Datenanzeigewort **.date**

Und wie wäre es mit der Benennung von FORTH-Wörtern in einer anderen Sprache als Englisch? Auch hier gilt nur eine Regel: **völlige Freiheit** ! Seien Sie jedoch vorsichtig, ESP32forth akzeptiert keine Namen, die in anderen Alphabeten als dem lateinischen Alphabet geschrieben sind. Sie können jedoch diese Alphabete für Kommentare verwenden:

```
.date    \ Плакат сегодняшней даты
...code... ;
```

Oder

```
: .date    \ 海報今天的日期
...code... ;
```

Einrückung des Quellcodes

Ob der Code zwei Zeilen, zehn Zeilen oder mehr umfasst, hat keinen Einfluss auf die Leistung des Codes nach der Kompilierung. Sie können Ihren Code also auch strukturiert einrücken:

- eine Zeile pro Wort der Kontrollstruktur **if else then** , **begin while repeat...**
Für das Wort if können wir ihm den logischen Test voranstellen, den es verarbeitet wird;
- eine Zeile durch Ausführung eines vordefinierten Wortes, gegebenenfalls vorangestellt durch die Parameter dieses Wortes.

Beispiel :

```
60 constant MAX_LIGHT_TIME_NORMAL_CYCLE
: cycle.stop
  -1 +to MAX_LIGHT_TIME
  MAX_LIGHT_TIME 0 =
  if
    LOW myLIGHTS pin
  else
    0 rerun
  then
;
;
```

Wenn der in einer Kontrollstruktur verarbeitete Code spärlich ist, kann der FORTH-Code komprimiert werden:

```
: cycle.stop
  -1 +to MAX_LIGHT_TIME
  MAX_LIGHT_TIME 0 =
  if          LOW myLIGHTS pin
```

```

else      0 rerun      then
;

```

Dies ist häufig bei **case of endof endcase**-Strukturen der Fall ;

```

: socketError ( -- )
  errno dup
  case
    2 of      ." No such file "      endof
    5 of      ." I/O error "        endof
    9 of      ." Bad file number "   endof
    22 of     ." Invalid argument "  endof
  endcase
  . quit
;

```

Kommentare

Wie jede Programmiersprache ermöglicht die FORTH-Sprache das Hinzufügen von Kommentaren im Quellcode. Das Hinzufügen von Kommentaren hat nach dem Kompilieren des Quellcodes keine Auswirkungen auf die Leistung der Anwendung.

In der FORTH-Sprache haben wir zwei Wörter, um Kommentare abzugrenzen:

- dem Wort **(** muss mindestens ein Leerzeichen folgen. Dieser Kommentar wird durch das Zeichen vervollständigt **)** ;
- Dem Wort **** muss mindestens ein Leerzeichen folgen. Auf dieses Wort folgt zwischen diesem Wort und dem Ende der Zeile ein Kommentar beliebiger Größe.

Das Wort **(** wird häufig für Stapelkommentare verwendet. Beispiele:

```

dup   ( n - n n )
swap  ( n1 n2 - n2 n1 )
drop  ( n -- )
emit  ( c -- )

```

Stapelkommentare

Wie wir gerade gesehen haben, sind sie mit **(** und **)** gekennzeichnet . Ihr Inhalt hat keinen Einfluss auf den FORTH-Code während der Kompilierung oder Ausführung. Wir können also alles zwischen **(** und **)** setzen . Was die Stack-Kommentare betrifft, bleiben wir sehr prägnant. Das **--**-Zeichen symbolisiert die Wirkung eines FORTH-Wortes . Die Angaben vor **--** entsprechen den Daten, die vor der Ausführung des Wortes auf dem Datenstapel abgelegt wurden. Die Angaben nach **-** entsprechen den Daten, die nach der Ausführung des Wortes auf dem Datenstapel verbleiben. Beispiele:

- **words (--)** bedeuten, dass dieses Wort keine Daten im Datenstapel verarbeitet;
- **emit (c --)** bedeutet, dass dieses Wort Daten als Eingabe verarbeitet und nichts auf dem Datenstapel hinterlässt;

- **b1 (--32)** bedeutet, dass dieses Wort keine Eingabedaten verarbeitet und den Dezimalwert 32 auf dem Datenstapel belässt;

Es gibt keine Begrenzung hinsichtlich der Datenmenge, die vor oder nach der Ausführung des Wortes verarbeitet wird. Zur Erinnerung: Die Angaben zwischen (und) dienen nur zur Information.

Bedeutung von Stack-Parametern in Kommentaren

Zunächst ist eine kleine, aber sehr wichtige Klarstellung notwendig. Dies ist die Größe der Daten im Stapel. Bei ESP32Forth belegen die Stack-Daten 4 Bytes. Es handelt sich also um Ganzzahlen im 32-Bit-Format. Einige Wörter verarbeiten Daten jedoch im 8-Bit-Format. Was legen wir also auf den Datenstapel? Mit ESP32Forth werden es **IMMER 32-BIT-DATEN sein** ! Ein Beispiel mit dem Wort **c!** :

```
create myDelemiter
  0 c,
64 myDelimiter c! ( n1 n2 -- )
```

Hier gibt der Parameter **c** an, dass wir einen ganzzahligen Wert im 32-Bit-Format stapeln, dessen Wert jedoch immer im Intervall [0..255] enthalten ist .

Der Standardparameter ist immer **n** . Wenn es mehrere ganze Zahlen gibt, nummerieren wir sie: **n1 n2 n3** usw.

Wir hätten das vorherige Beispiel also auch so schreiben können:

```
create myDelemiter
  0 c,
64 myDelimiter c! ( c addr -- )
```

Aber es ist viel weniger explizit als die vorherige Version. Hier sind einige Symbole, die Sie im gesamten Quellcode sehen werden:

- **addr** gibt eine wörtliche Speicheradresse an oder wird von einer Variablen geliefert;
- **c** gibt einen 8-Bit-Wert im Intervall [0..255] an.
- **d** gibt einen Wert mit doppelter Genauigkeit an. Wird nicht mit ESP32Forth verwendet, das bereits im 32-Bit-Format vorliegt.
- **fl** gibt einen booleschen Wert an, 0 oder ungleich Null;
- **n** gibt eine ganze Zahl an. 32-Bit-Ganzzahl mit Vorzeichen für ESP32Forth;
- **str** gibt eine Zeichenfolge an. Entspricht **addr len --**
- **u** gibt eine vorzeichenlose Ganzzahl an

Nichts hindert uns daran, etwas deutlicher zu werden:

```
: SQUARE ( n -- n-exp2 )
```

```
dup *  
;
```

Wortdefinition Wortkommentare

Definitionswörter verwenden **create** und **does>**. Für diese Wörter empfiehlt es sich, Stapelkommentare wie diesen zu schreiben:

```
\ Definieren Sie einen Befehl oder Datenstrom für SSD1306  
: streamCreate: ( comp: <name> | exec: -- addr len )  
  create  
    here      \ leave current dictionary pointer on stack  
    0 c,      \ initial length data is 0  
  does>  
    dup 1+ swap c@  
    \ Senden Sie ein Datenarray an SSD1306,  
    \ das über den I2C-Bus angeschlossen ist  
    sendDataToSSD1306  
;
```

Hier wird der Kommentar durch das Zeichen **|** in zwei Teile geteilt :

- links der Aktionsteil, wenn das Definitionswort ausgeführt wird, mit dem Präfix **comp:**
- rechts der Aktionsteil des Wortes, der definiert wird, mit dem Präfix **exec:**

Auf die Gefahr hin, darauf zu bestehen, ist dies kein Standard. Es handelt sich lediglich um Empfehlungen.

Textkommentare

Sie werden durch das Wort **** gefolgt von mindestens einem Leerzeichen und einem erläuternden Text gekennzeichnet:

```
\ Speichern Sie die Länge der dazwischen zusammengestellten  
\ Daten unter <WORD> und here  
: ;endStream ( addr-var len --- )  
  dup 1+ here  
  swap -      \ cdata-Länge berechnen  
  \ c im ersten Byte des durch streamCreate: definierten Wortes speichern  
  swap c!  
;
```

Diese Kommentare können in jedem von Ihrem Quellcode-Editor unterstützten Alphabet geschrieben werden:

```
\ 儲存在 <WORD> addr 之間編譯的資料長度  
\ <WORD> 和這裡  
: ;endStream ( addr-var len --- )  
  dup 1+ here  
  swap -      \ 計算 cdata 長度  
  \ 將 c 儲存在由 StreamCreate 定義的字的的第一個位元組中:
```

```
swap c!
```

```
;
```

Kommentar am Anfang des Quellcodes

Mit intensiver Programmierpraxis stehen Sie schnell vor Hunderten oder sogar Tausenden von Quelldateien. Um Fehler bei der Dateiauswahl zu vermeiden, wird dringend empfohlen, den Anfang jeder Quelldatei mit einem Kommentar zu markieren:

```
\ *****  
\ Manage commands for OLED SSD1306 128x32 display  
\  Filename:      SSD10306commands.fs  
\  Date:         21 may 2023  
\  Updated:      21 may 2023  
\  File Version: 1.0  
\  MCU:          ESP32-WROOM-32  
\  Forth:        ESP32forth all versions 7.x++  
\  Copyright:    Marc PETREMAN  
\  Author:       Marc PETREMAN  
\  GNU General Public License  
\ *****
```

Alle diese Informationen liegen in Ihrem Ermessen. Sie können sehr nützlich sein, wenn Sie Monate oder Jahre später auf den Inhalt einer Datei zurückkommen.

Zögern Sie abschließend nicht, Ihre Quelldateien in der FORTH-Sprache zu kommentieren und einzurücken.

Diagnose- und Tuning-Tools

Das erste Tool betrifft die Kompilierungs- oder Interpretationswarnung:

```
3 5 25 --> : TEST ( --- )  
ok  
3 5 25 --> [ HEX ] ASCII A DDUP \ DDUP don't exist
```

Hier existiert das Wort **DDUP** nicht. Jede Kompilierung nach diesem Fehler schlägt fehl.

Der Dekompiler

In einem herkömmlichen Compiler wird der Quellcode in ausführbaren Code umgewandelt, der die Referenzadressen zu einer Bibliothek enthält, mit der der Compiler ausgestattet ist. Um ausführbaren Code zu erhalten, müssen Sie den Objektcode verknüpfen. Der Programmierer kann zu keinem Zeitpunkt allein mit den Ressourcen des Compilers auf den in seiner Bibliothek enthaltenen ausführbaren Code zugreifen.

Mit ESP32Forth kann der Entwickler seine Definitionen dekompile. Um ein Wort zu dekompile, geben Sie einfach **see** gefolgt vom zu dekompilenden Wort ein:

```
: C>F ( °C --- °F ) \ Conversion Celsius in Fahrenheit  
  9 5 */ 32 +  
;
```

```

see c>f
\ display:
: C>F
  9 5 */ 32 +
;

```

Viele Wörter im FORTH-Wörterbuch von ESP32Forth können dekompiert werden.

Durch die Dekompilierung Ihrer Wörter können Sie mögliche Kompilierungsfehler erkennen.

Speicherauszug

Manchmal ist es wünschenswert, die im Speicher befindlichen Werte sehen zu können. Der Wortdump **akzeptiert** zwei Parameter: die Startadresse im Speicher und die Anzahl der anzuzeigenden Bytes:

```

create myDATAS 01 c, 02 c, 03 c, 04 c,
hex
myDATAS 4 dump      \ displays :
3FFEE4EC                                     01 02 03 04

```

Stapel monitor

Der Inhalt des Datenstapels kann jederzeit mit dem Wort **.s** angezeigt werden . Hier ist die Definition des Wortes **.DEBUG** , das **.s** ausnutzt :

```

variable debugStack

: debugOn ( -- )
  -1 debugStack !
;

: debugOff ( -- )
  0 debugStack !
;

: .DEBUG
  debugStack @
  if
    cr ." STACK: " .s
    key drop
  then
;

```

Um **.DEBUG** zu verwenden, fügen Sie es einfach an einer strategischen Stelle im zu debuggenden Wort ein:

```

\ Anwendungsbeispiel:
: mein Test
: myTEST
  128 32 do
    i .DEBUG
  emit

```

```
loop  
;
```

Hier zeigen wir den Inhalt des Datenstapels nach der Ausführung von Wort i in unserer **do loop** an . Wir aktivieren den Fokus und führen **myTEST** aus :

```
debugOn  
myTest  
\ displays:  
\ STACK: <1> 32  
\ 2  
\ STACK: <1> 33  
\ 3  
\ STACK: <1> 34  
\ 4  
\ STACK: <1> 35  
\ 5  
\ STACK: <1> 36  
\ 6  
\ STACK: <1> 37  
\ 7  
\ STACK: <1> 38
```

debugOn aktiviert ist , pausiert jede Anzeige des Inhalts des Datenstacks unsere **do loop**-Schleife . Führen Sie **debugOff** aus , damit das Wort **myTEST** normal ausgeführt wird.

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:

```
decimal 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

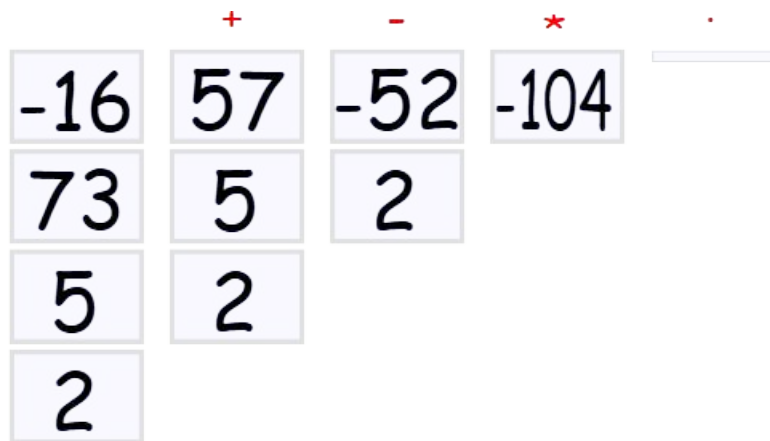
	Zelle	Inhalt	Kommentar
	0	-16	(TOS) Oben rechts
	1	73	(NOS) Als nächstes im Stapel
	2	5	
	3	2	

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 Stapelelemente 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.

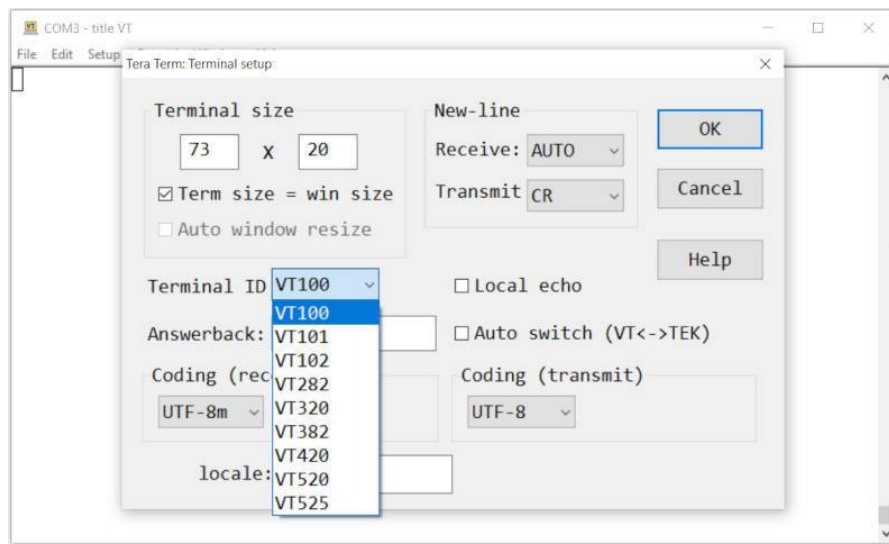
Wir können jetzt mit den zuvor erläuterten Abruf- und Speicherwörtern auf den dem **graphic-array** zugewiesenen Speicher zugreifen. Um die Adresse des dritten Bytes zu berechnen, das dem **graphic-array**, können wir **graphic-array 2 +** schreiben , wobei wir bedenken, dass die Indizes bei 0 beginnen.

```
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

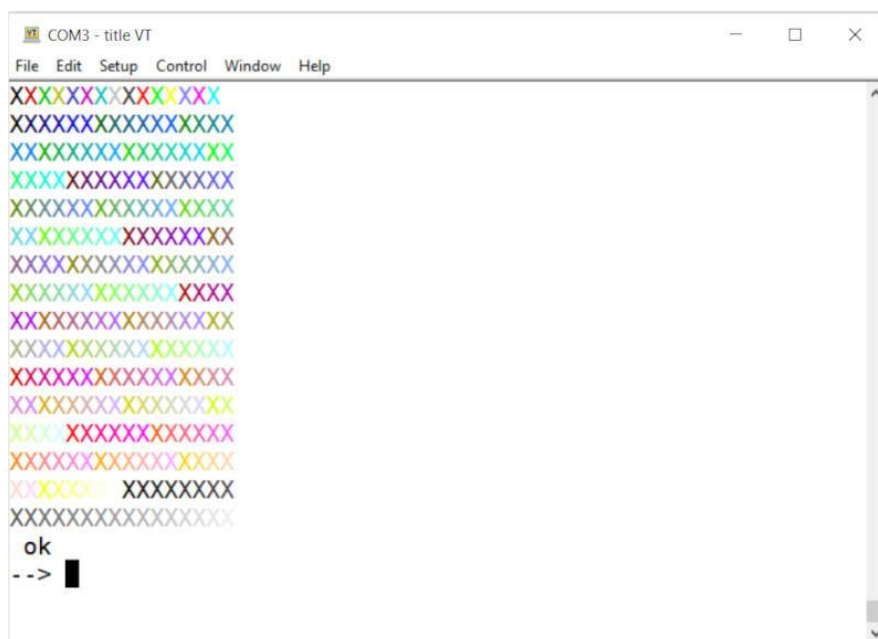
Sehen wir uns zunächst an, wie man den Text einfärbt :

```

: 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
\ ØSTART_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 * { ØSTART_ADDR } \ calc. Adresse für Schleifenstart
  16 / 1+ { LINES }
  base @ { myBASE }            \ aktuelle Basis speichern
  hex
      \ äußere Schleife
  LINES 0 do
    ØSTART_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
      ØSTART_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
        0START_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
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 ;

```

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 `cell` den Wert 2. Wenn Sie eine 64-Bit-Version verwenden, stapelt `cell` 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:

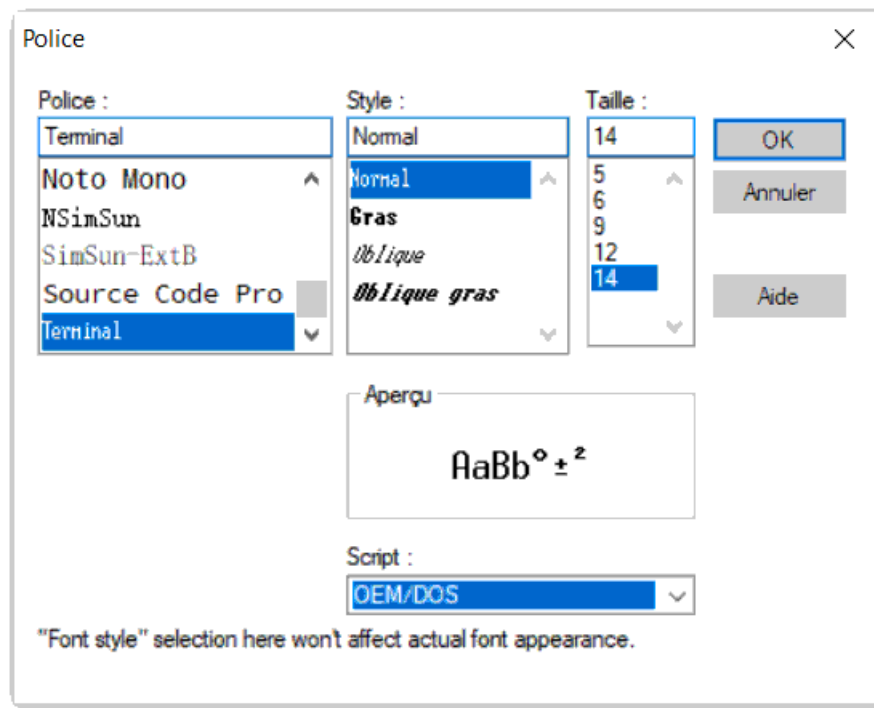
```
: xySCRaddr { x y -- addr }
  SCR_WIDTH y *
  x + mySCREEN +
;
: SCR@ ( x y -- c )
  xySCRaddr c@
;
: SCR! ( c x y -- )
  xySCRaddr c!
;
char X 15 5 SCR!    \ speichert char X am Hals 15 Zeile 5
15 5 SCR@ emit      \ wird angezeigt
```

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:



So zeigen Sie die Tabelle der verfügbaren Zeichen an:

```

: tableChars ( -- )
  base @ >r hex
  128 32 do
    16 0 do
      j i + dup . space emit space space
    loop
    cr
  16 +loop
  256 160 do
    16 0 do
      j i + dup . space emit space space
    loop
    cr
  16 +loop
  cr
  r> base !
;
tableChars

```

Hier ist das Ergebnis der Ausführung von **tableChars**:

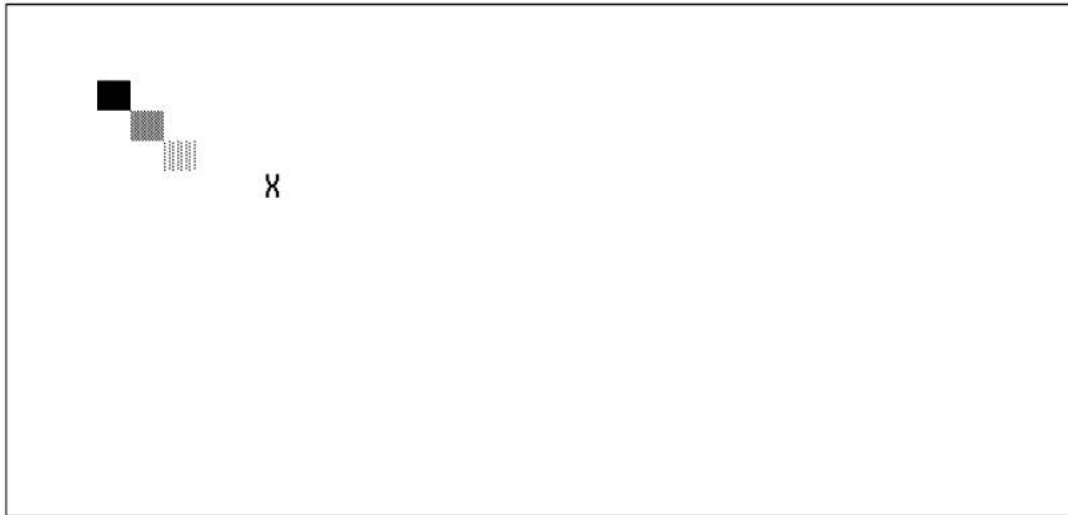
[illegible]

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

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

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

```
: 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
:
```



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:

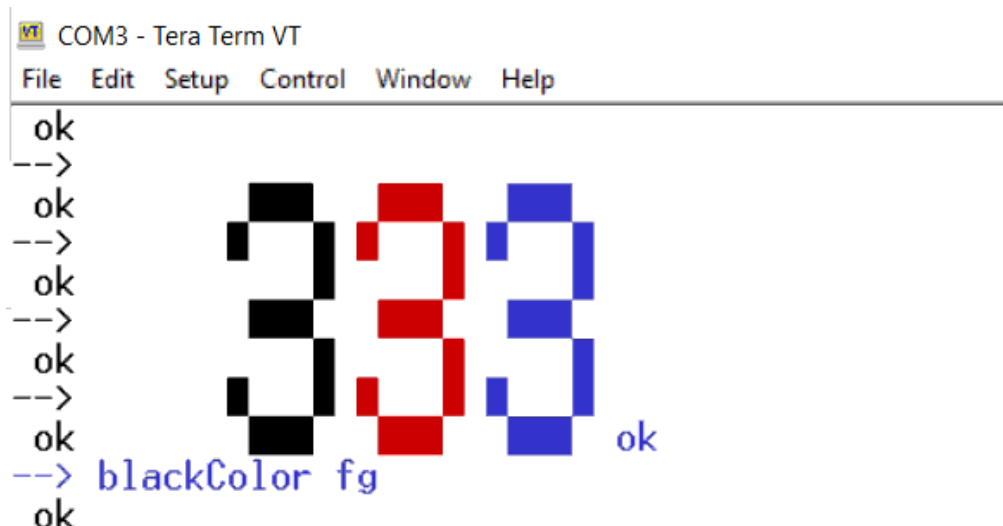
```

: .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 $r1 <> r2$.
- **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 SWAP 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

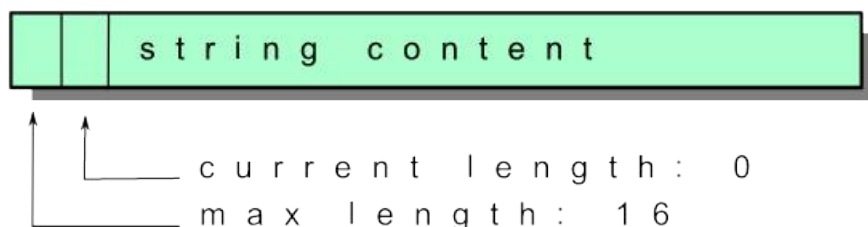
Alphanumerische Textvariablen sind in ESP32forth nativ nicht vorhanden. Hier ist der erste Versuch, die Wortfolge **STRING** zu definieren :

```
\ 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 --- f1)
    str=
    ;

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

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

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

Verzögerte Aktionswörter

Wörter mit verzögerter Aktion werden durch das Definitionswort **defer** definiert . Um die Mechanismen und das Interesse an der Nutzung dieser Wortart zu verstehen, schauen wir uns die Funktionsweise des internen Interpreters der FORTH-Sprache genauer an.

Jede durch **:** (Doppelpunkt) kompilierte Definition enthält eine Folge codierter Adressen, die den Codefeldern der zuvor kompilierten Wörter entsprechen. Im Herzen des FORTH-Systems akzeptiert das Wort **EXECUTE** diese Codefeldadressen als Parameter, Adressen, die wir mit **cfa** für *Code Field Address* abkürzen. Jedes FORTH-Wort hat ein **cfa** und diese Adresse wird vom internen FORTH-Interpreter verwendet:

```
' <Wort>
\ legt das CFA von <Wort> auf dem Datenstapel ab
```

Beispiel:

```
' WORDS
\ stapelt die WORDS cfa.
```

Aus diesem **cfa** , bekannt als einziger Literalwert, kann die Ausführung des Wortes mit **EXECUTE** erfolgen :

```
' WORDS EXECUTE
\ führt WORDS aus
```

Natürlich wäre es einfacher gewesen, **WORDS** direkt einzugeben . Sobald ein **cfa** als einziger Literalwert verfügbar ist, kann er manipuliert und insbesondere in einer Variablen gespeichert werden:

```
variable vector
' WORDS vector !
vector @ .
\ zeigt cfa der in der Vektorvariablen gespeicherten WORDS an
```

Sie können **WORDS** indirekt über den Inhalt von **vector** ausführen :

```
vector @ EXECUTE
```

Dadurch wird die Ausführung des Wortes gestartet, dessen **cfa** in der **vektor** variablen gespeichert und dann vor der Verwendung durch **EXECUTE** wieder auf den Stapel gelegt wurde .

Defer- Definitionsworts ausgenutzt wird . Zur Vereinfachung erstellt **defer** einen Header im Wörterbuch, wie eine **variable** oder **eine constant** , aber anstatt einfach eine Adresse oder einen Wert auf dem Stapel abzulegen, startet es die Ausführung des Wortes, dessen cfa im parametrischen Bereich des definierten Wortes gespeichert **wurde** durch **defer**.

Definition und Verwendung von Wörtern mit defer

Die Initialisierung eines durch **defer** definierten Wortes erfolgt durch **is** :

```
defer vector
' words is vector
```

Durch die Ausführung **des vector** wird das Wort ausgeführt, dessen **cfa** zuvor zugewiesen wurde :

```
vector \ führt words aus
```

Ein von **defer** erstelltes Wort wird verwendet, um ein anderes Wort auszuführen, ohne dieses Wort explizit aufzurufen. Das Hauptinteresse dieser Wortart liegt vor allem in der Möglichkeit, das auszuführende Wort zu modifizieren:

```
' page is vector
```

Der **Vektor** führt jetzt **page** und nicht mehr **words** aus.

Wir verwenden die durch **defer** definierten Wörter im Wesentlichen in zwei Situationen:

- Definition einer Vorwärtsreferenz;
- Definition eines Wortes abhängig vom Betriebskontext.

Im ersten Fall ermöglicht die Definition einer Vorreferenz die Überwindung der Zwänge des unantastbaren Vorrangs von Definitionen.

Im zweiten Fall ermöglicht die Definition eines Wortes in Abhängigkeit vom Betriebskontext, die meisten Schnittstellenprobleme mit einer sich entwickelnden Softwareumgebung zu lösen, die Portabilität von Anwendungen aufrechtzuerhalten und das Verhalten eines Programms an Situationen anzupassen, die von verschiedenen gesteuert werden Parameter ohne Beeinträchtigung der Softwareleistung.

Festlegen einer Vorwärtsreferenz

Im Gegensatz zu anderen Compilern erlaubt FORTH nicht, ein Wort vor seiner Definition in eine Definition zu kompilieren. Dies ist das Prinzip der Vorrangigkeit von Definitionen:

```
: word1 ( ---) word2 ;
: word2 ( ---) ;
```

von word1 zu einem Fehler , da word2 noch nicht definiert ist. So umgehen Sie diese Einschränkung mit **defer** :

```
defer word2
: word1 ( ---) word2 ;
: (word2) ( ---) ;
' (word2) is word2
```

Dieses Mal wurde **word2** ohne Fehler kompiliert. Es ist nicht erforderlich, dem vektorisierten Ausführungswort **word2** ein cfa zuzuweisen . Erst nach der Definition von

(**word2**) wird der Parameterbereich von **word2** aktualisiert. Nach der Zuweisung des vektorisierten Ausführungsworts **word2** kann **word1** den Inhalt seiner Definition fehlerfrei ausführen. Die Ausbeutung von Wörtern, die durch **defer** geschaffen wurden, muss in dieser Situation die Ausnahme bleiben.

Abhängigkeit vom Betriebskontext

ESP32forth nutzt nativ eine Verbindung über den seriellen Port 1 als Ein- und Ausgabestream.

Im ESP32forth-Quellcode finden wir diese Zeilen:

```
defer type
defer key
defer key?
```

Um den seriellen Port zu passieren, initialisiert ESP32forth das **type** wie folgt:

```
' default-type is type
```

Wenn wir ESP32forth im Servermodus aktivieren, wird **type** wie folgt umgeleitet :

```
: server ( port -- )
  server
  ['] serve-key is key
  ['] serve-type is type
  webserver-task start-task
;
```

Hier ist die **type** Umleitung, wenn wir einen TELNET-Fluss verwenden:

```
: connection ( n -- )
  dup 0< if drop exit then to clientfd
  0 echo !
  ['] telnet-key is key
  ['] telnet-type is type quit ;
```

Und wenn wir die Textanzeige auf ein OLED-Display umleiten wollten, müssten wir genauso auf die Schriftart **reagieren**. Im Kapitel *Einrichten des REYAX RYLR890 LoRa-Senders* nutzen wir diese **type** Eigenschaft wie folgt aus:

```
serial \ Seriellles Vokabular auswählen
: serial2-type ( a n -- )
  Serial2.write drop ;
: typeToLoRa ( -- )
  0 echo ! \ Anzeigeecho vom Terminal deaktivieren
  ['] serial2-type ist der Typ
;
: typeToTerm ( -- )
  ['] default-type is type
  -1 echo ! \ Anzeigeecho vom Terminal aktivieren
;
```

Dadurch wird es sehr einfach, einen Textstrom an die serielle Schnittstelle 2 zu übertragen :

```
: optionChoice
    ." choice option:" ;
optionChoice      \ Auswahloptionen anzeigen: auf dem Terminal
typeToLoRa
optionChoice      \ Auswahloptionen anzeigen: bis serial2
typeToTerm        \ stellt die normale Anzeige wieder her
```

In diesem speziellen Fall definieren wir viele Wörter, die es Ihnen ermöglichen, einen LoRa-Sender mit gewöhnlichen Wörtern wie **emit** , **type**, usw... zu steuern. Wenn wir die Übertragung an den seriellen Port 2, also an den LoRa-Sender, nicht aktivieren, können die Wörter, die mit diesem Sender kommunizieren, einfach entwickelt werden :

```
\ Stellen Sie die ADRESSE des LoRa-Senders ein:
\ s" <adress>" Wert im Intervall [0..65535][?] (Standard 0)
: ATaddress ( addr len -- )
    ." AT+ADDRESS="
    type crlf
;
```

ATaddress ausführen , wird der Textstream auf dem Terminal angezeigt. Wenn Sie es richtig befolgt haben, wissen Sie, welches Wort Sie ausführen müssen, um den Fluss von **ATaddress** zum seriellen Port 2 umzuleiten.

Zusammenfassend lässt sich sagen, dass wir dank verzögerter Ausführungswörter auf die Aktion bereits definierter FORTH-Wörter reagieren können.

Ein praktischer Fall

Sie müssen eine Anwendung mit Anzeigen in zwei Sprachen erstellen. Hier ist eine clevere Möglichkeit, ein durch defer definiertes Wort auszunutzen, um Text auf Französisch oder Englisch zu generieren. Zunächst erstellen wir einfach eine Tagedabelle auf Englisch:

```
:noname s" Saturday" ;
:noname s" Friday" ;
:noname s" Thursday" ;
:noname s" Wednesday" ;
:noname s" Tuesday" ;
:noname s" Monday" ;
:noname s" Sunday" ;

create ENdayNames ( --- addr)
    , , , , , , ,
```

Dann erstellen wir eine ähnliche Tabelle für die Tage auf Französisch:

```
:noname s" Samedi" ;
:noname s" Vendredi" ;
```

```

:noname s" Jeudi" ;
:noname s" Mercredi" ;
:noname s" Mardi" ;
:noname s" Lundi" ;
:noname s" Dimanche" ;

create FRdayNames ( --- addr)
  , , , , , , ,

```

Schließlich erstellen wir unser verzögertes Aktionswort **dayNames** und wie man es initialisiert:

```

defer dayNames

: in-ENGLISH
  ['] ENdayNames is dayNames ;

: in-FRENCH
  ['] FRdayNames is dayNames ;

```

Hier sind nun die Wörter, um diese beiden Tabellen zu verwalten:

```

: _getString { array length -- addr len }
  array
  swap cell *
  + @ execute
  length ?dup if
    min
  then
;

10 value dayLength
: getDay ( n -- addr len )      \ n interval [0..6]
  dayNames dayLength _getString
;

```

Folgendes bewirkt die Ausführung **von getDay** :

```

in-ENGLISH 3 getDay type cr \ Anzeige: Wednesday
in-FRENCH 3 getDay type cr \ Anzeige: Mercredi

```

Hier definieren wir das Wort **.dayList** , das den Anfang der Namen der Wochentage anzeigt:

```

: .dayList { size -- }
  size to dayLength
  7 0 do
    i getDay type space
  loop
;

```

```
in-ENGLISH 3 .dayList cr \ display : Sun Mon Tue Wed Thu Fri Sat
in-FRENCH 1 .dayList cr \ display : D L M M J V S
```

In der zweiten Zeile zeigen wir nur den Anfangsbuchstaben jedes Wochentags an. In diesem Beispiel nutzen wir **defer**, um die Programmierung zu vereinfachen. In der Webentwicklung würden wir *Vorlagen verwenden*, um mehrsprachige Websites zu verwalten. In FORTH verschieben wir einfach einen Vektor in einem verzögerten Aktionswort. Hier verwalten wir nur zwei Sprachen. Dieser Mechanismus lässt sich leicht auf andere Sprachen erweitern, da wir die Verwaltung von Textnachrichten vom reinen Anwendungsteil getrennt haben.

Wortschöpfungswörter

FORTH ist mehr als eine Programmiersprache. Es ist eine Metasprache. Eine Metasprache ist eine Sprache, die dazu dient, andere Sprachen zu beschreiben, zu spezifizieren oder zu manipulieren.

Mit ESP32forth können wir die Syntax und Semantik von Programmierwörtern über den formalen Rahmen grundlegender Definitionen hinaus definieren.

constant , **variable** und **value** definierten Wörter gesehen . Diese Wörter werden zur Verwaltung digitaler Daten verwendet.

Im Kapitel *Datenstrukturen für ESP32* haben wir auch das Wort **create** verwendet . Dieses Wort erstellt einen Header, der den Zugriff auf einen im Speicher gespeicherten Datenbereich ermöglicht. Beispiel :

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

Dabei wird jeder Wert im Parameterbereich des Worts **temperatures** mit dem Wort **,** gespeichert .

Mit ESP32forth werden wir sehen, wie wir die Ausführung von durch **create** definierten Wörtern anpassen können.

Die Verwendung des Wortes does>

Den Schlüsselwörtern **CREATE** und **DOES>**, die häufig zusammen verwendet werden, um benutzerdefinierte Wörter (Vokabularwörter) mit bestimmten Verhaltensweisen zu erstellen.

So funktioniert es im Allgemeinen in Forth:

- **CREATE** : Dieses Schlüsselwort wird verwendet, um einen neuen Datenraum im ESP32Forth-Wörterbuch zu erstellen. Es braucht ein Argument, nämlich den Namen, den Sie Ihrem neuen Wort geben;
- **DOES>** : Dieses Schlüsselwort wird verwendet, um das Verhalten des Wortes zu definieren, das Sie gerade mit **CREATE** erstellt haben . Es folgt ein Codeblock, der angibt, was das Wort tun soll, wenn es während der Programmausführung angetroffen wird.

Zusammen sieht es ungefähr so aus:

```
forth
CREATE my-new-word
  \ Code, der ausgeführt werden soll, wenn mein-neues-Wort auftritt
DOES>
;
```

Wenn das Wort **my-new-word** im FORTH-Programm auftritt, wird der im Teil **does>...;** angegebene Code verwendet und durchgeführt.

```
\ Definieren Sie ein Register, ähnlich einer Konstante
: defREG:
  create ( addr1 -- <name> )
  ,
  does> ( -- regAddr )
    @
;

```

Hier definieren wir das Definitionswort **defREG:**, das genau die gleiche Wirkung hat wie **constant**. Aber warum sollte man ein Wort erschaffen, das die Wirkung eines bereits existierenden Wortes nachahmt?

```
$3FF44004 constant GPIO_OUT_REG
```

Oder

```
$3FF44004 defREG: GPIO_OUT_REG
```

sind ähnlich. Durch die Erstellung unserer Register mit **defREG:** haben wir jedoch folgende Vorteile:

- ein besser lesbarer ESP32forth-Quellcode. Wir erkennen leicht alle Konstanten, die ein ESP32-Register benennen;
- **does>-** Teil von **defREG:** zu ändern, ohne dann die Codezeilen neu schreiben zu müssen, die **defREG:** nicht verwenden würden.

Hier ist ein klassischer Fall, bei dem eine Datentabelle verarbeitet wird:

```
\ Definitionswort für eindimensionales Array
: array ( comp: -- <name> | exec: index <name> -- addr )
  create
  does>
    swap cell * +
;
array temperatures
  21 ,    32 ,    45 ,    44 ,    28 ,    12 ,
0 temperatures @ .  \anzeige 21
5 temperatures @ .  \anzeige 12

```

Der Ausführung von **temperatures** muss die Position des zu extrahierenden Werts in dieser Tabelle vorangestellt werden. Hier erhalten wir nur die Adresse, die den zu extrahierenden Wert enthält.

Beispiel für Farbmanagement

In diesem ersten Beispiel definieren wir das Wort **color:**, das die auszuwählende Farbe abrufen und in einer Variablen speichern:

```

0 value currentCOLOR

\ Wort als COLOR-Konstante definieren
: color: ( n -- <name> )
  create
    ,
  does>
    @ to currentCOLOR
;

$00 color: setBLACK
$ff color: setWHITE

```

Das Ausführen des Wortes **setBLACK** oder **setWHITE** vereinfacht den ESP32forth-Code erheblich. Ohne diesen Mechanismus hätte eine dieser Zeilen regelmäßig wiederholt werden müssen:

```
$00 currentCOLOR !
```

Oder

```

$00 constant BLACK
BLACK currentCOLOR !

```

Beispiel: Schreiben in Pinyin

Pinyin wird auf der ganzen Welt häufig verwendet, um die Aussprache des Mandarin-Chinesisch zu lehren, und es wird auch in verschiedenen offiziellen Kontexten in China verwendet, beispielsweise auf Straßenschildern, Wörterbüchern und Lernbüchern. Es erleichtert Menschen, deren Muttersprache das lateinische Alphabet verwendet, Chinesisch zu lernen.

Um Chinesisch auf einer QWERTZ-Tastatur zu schreiben, verwenden Chinesen im Allgemeinen ein System namens „Pinyin-Eingabe“. Pinyin ist ein System zur Romanisierung des Mandarin-Chinesisch, das das lateinische Alphabet verwendet, um die Laute des Mandarin darzustellen.

Auf einer QWERTZ-Tastatur geben Benutzer Mandarin-Lautlaute in Pinyin-Romanisierung ein. Wenn jemand beispielsweise das Zeichen „你“ schreiben möchte („nǐ“, was auf Englisch „tu“ oder „toi“ bedeutet), kann er „ni“ eingeben.

In diesem sehr vereinfachten Code können Sie Pinyin-Wörter so programmieren, dass sie in Mandarin geschrieben werden. Der folgende Code funktioniert nur mit dem PuTTY-Terminal:

```

\ Funktioniert nur mit dem PuTTY-Terminal
Intern
internals
: chinese:
  create ( c1 c2 c3 -- )

```

```
    c, c, c,  
  does>  
    3 serial-type  
  ;  
forth
```

Um den UTF8-Code eines chinesischen Schriftzeichens zu finden, kopieren Sie das chinesische Schriftzeichen, beispielsweise aus Google Translate. Beispiel :

Guten Morgen -> 早安 (Zao an)

Kopieren Sie 早, gehen Sie zum PuTTY-Terminal und geben Sie Folgendes ein:

key key key \ gefolgt von der Taste <Enter>

Fügen Sie das Zeichen 早 ein. ESP32forth sollte die folgenden Codes anzeigen:

230 151 169

Für jedes chinesische Schriftzeichen verwenden wir diese drei Codes wie folgt:

169 151 230 chinese: Zao
137 174 229 chinese: An

Verwenden :

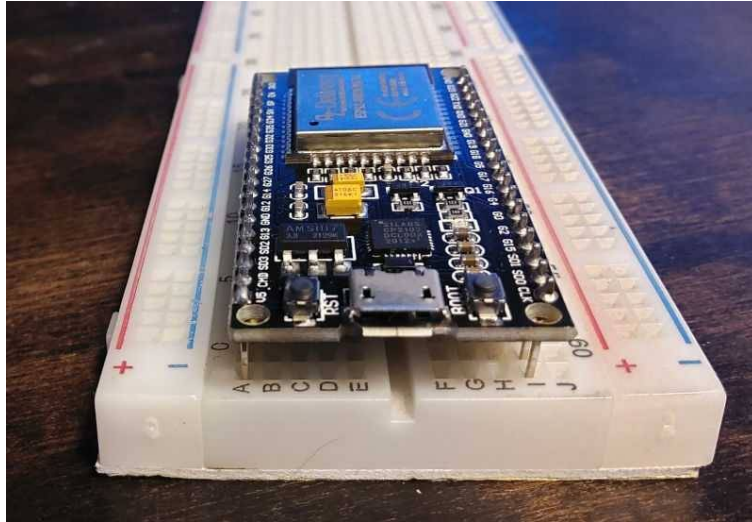
Zao An \ display 早安

Geben Sie zu, dass eine solche Programmierung etwas anderes ist als das, was Sie in der Sprache C tun können. Nein?

Passen Sie Steckbretter an das ESP32-Board an

Testplatten für ESP32

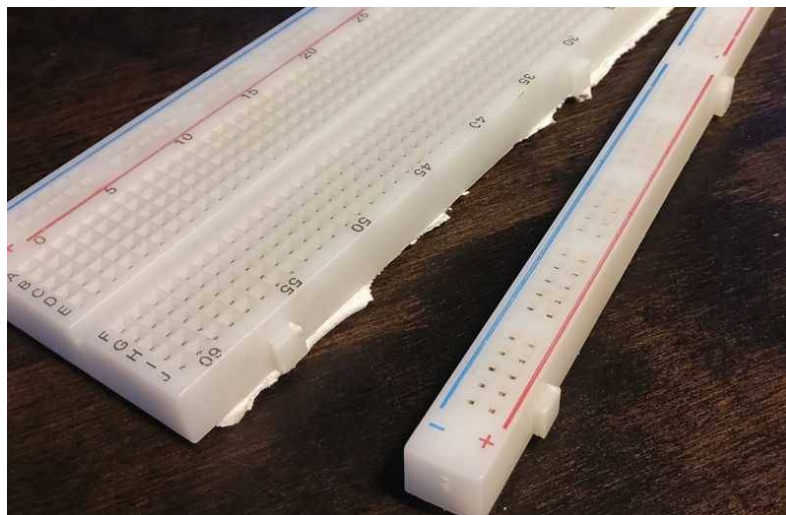
Sie haben gerade Ihre ESP32-Karten erhalten. Und die erste böse Überraschung: Diese Karte passt sehr schlecht auf das Testboard:



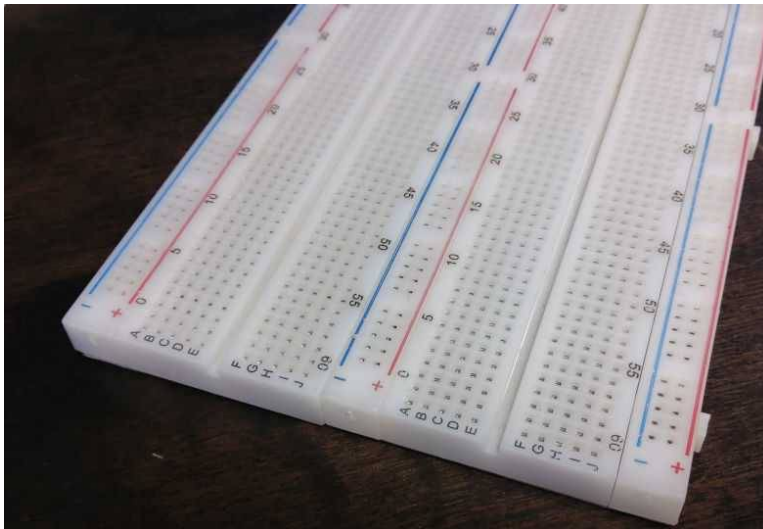
Es gibt kein Steckbrett, das speziell für ESP32-Boards geeignet ist.

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

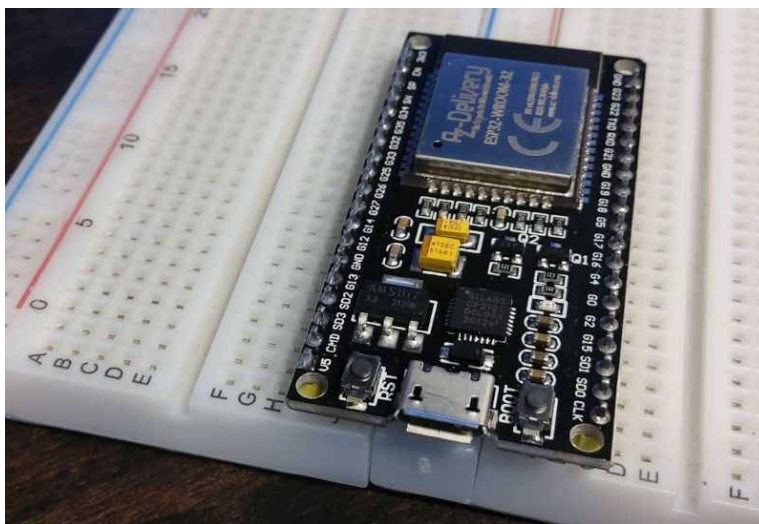
Wir werden unsere eigene Testplatte bauen. Dazu benötigen Sie zwei identische Testplatten. Auf einer der Platten werden wir eine Stromleitung entfernen. Verwenden Sie dazu einen Cutter und schneiden Sie von unten. Sie sollten diese Stromleitung folgendermaßen trennen können:



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

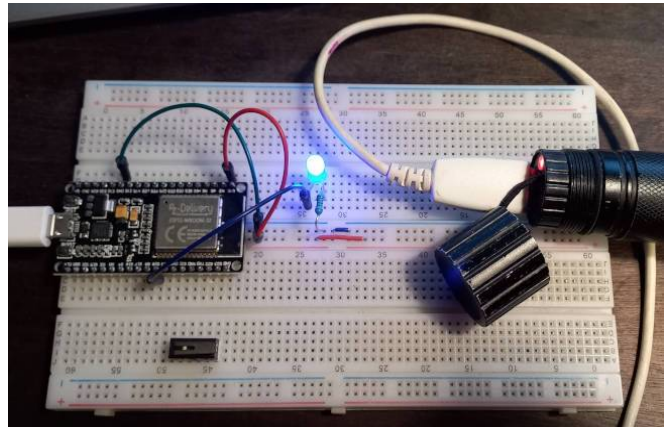
Wahl der Stromquelle

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).



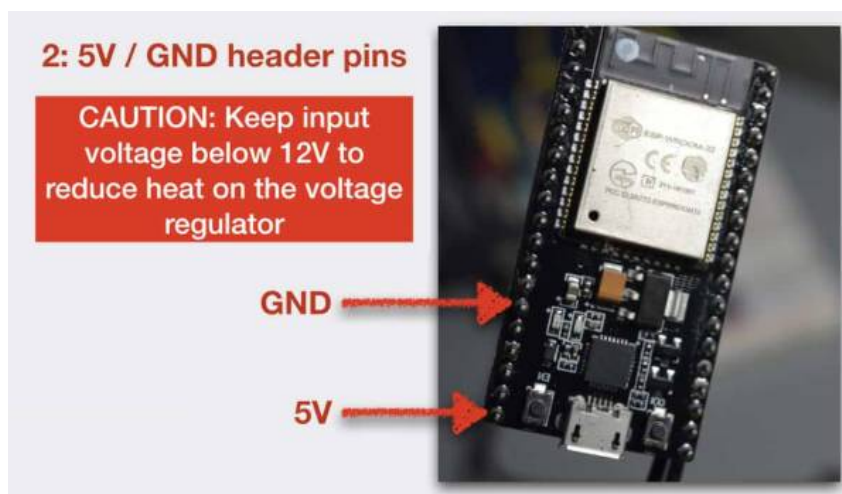
Hier versorgen wir unser ESP32-Board mit einem Backup-Akku für mobile Geräte.

Stromversorgung über 5V-Pin

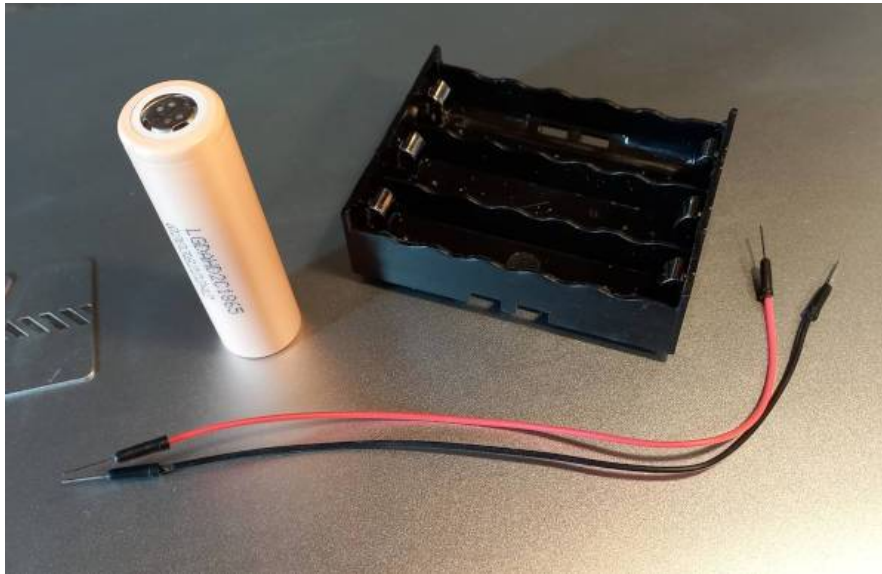
Die zweite Möglichkeit besteht darin, eine externe unregelte Stromversorgung an den 5V-Pin und Masse anzuschließen. Alles zwischen 5 und 12 Volt sollte funktionieren.

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.

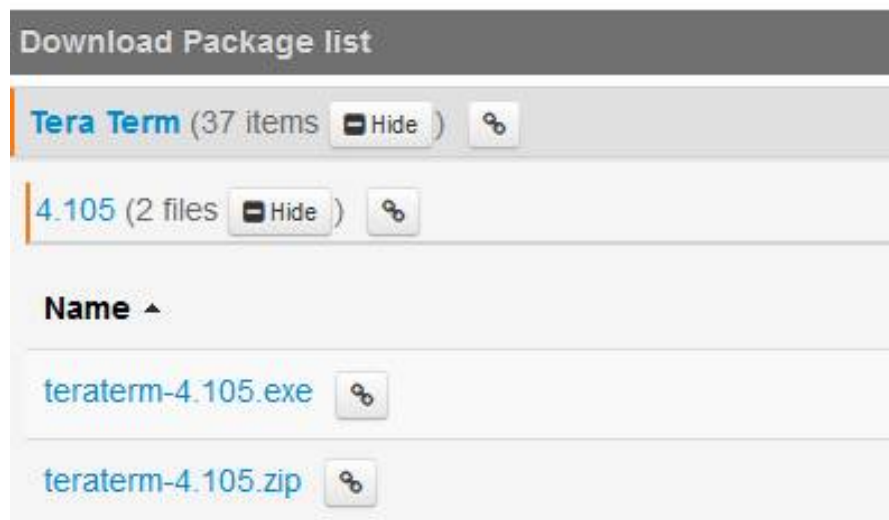
Installieren und verwenden Sie das Tera Term- Terminal unter Windows

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:

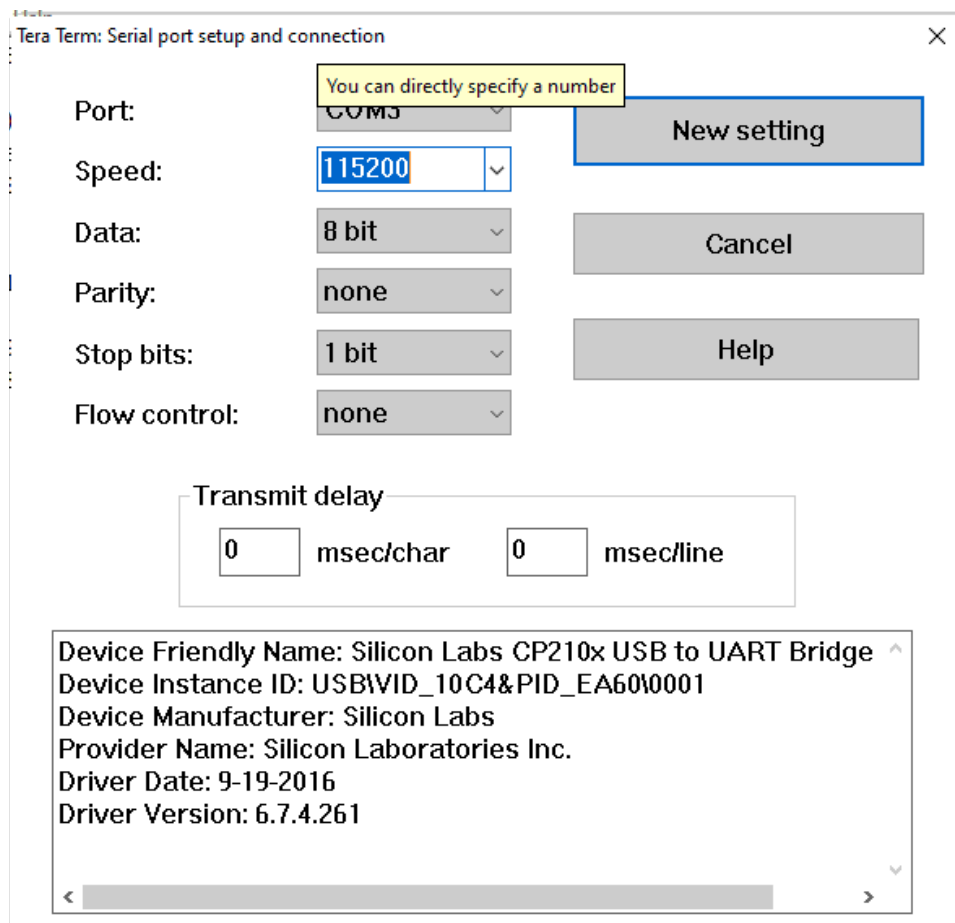


Installieren Sie Tera Term. Die Installation ist schnell und einfach.

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

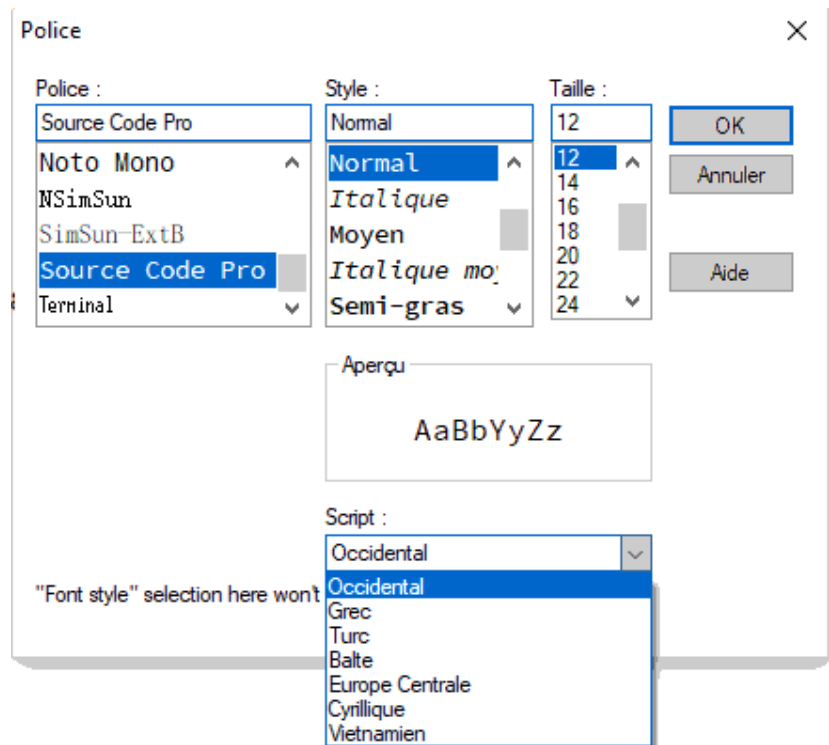
G: 0 < >

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** .

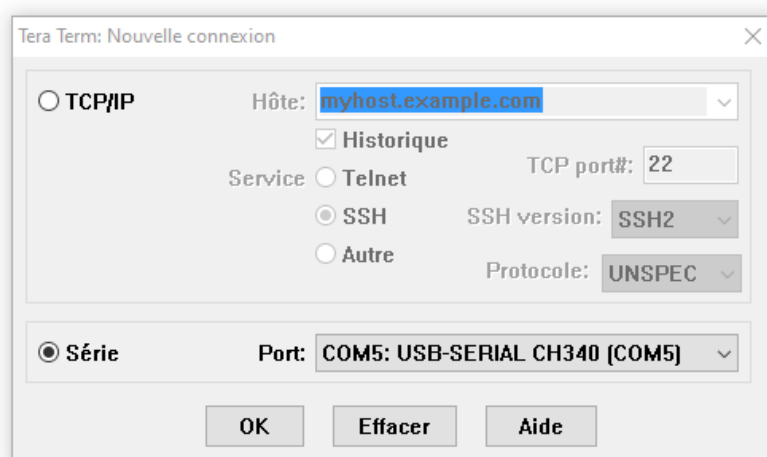
Verwendung von Tera Term

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:



```
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:

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.

Greifen Sie über TELNET auf ESP32Forth zu

Bevor Sie eine Verbindung verwalten, müssen Sie eine Netzwerkverbindung herstellen. Das ESP32-Board verfügt über eine WiFi-Schnittstelle. Um eine WLAN-Verbindung herzustellen, müssen Sie:

- über ein Modem/Router verfügen, der WLAN-Verbindungen verwaltet
- Sie müssen über die SSID des verfügbaren WLAN-Ports und seinen Zugriffsschlüssel verfügen

Die Verbindung zum WLAN-Netzwerk wird durch das Wort **login** sichergestellt :

```
\ Verbindung zum lokalen WLAN-LAN
: myWiFiConnect ( -- )
  z" Mariloo"
  z" 1925144D91DE5373C3XXXXXXXX"
  login
;
```

Beim Ausführen **von myWiFiConnect** wird Folgendes angezeigt:

```
-> myWiFiConnect
192.168.1.8
MDNS gestartet
```

Ändern Sie den DNS-Namen des ESP32-Boards

Um eine Verbindung zu einem ESP32-Board herzustellen, gibt es zwei Methoden:

- indem es seine IP-Adresse im internen Netzwerk kennt. Im obigen Fall lautet die IP-Adresse 192.168.1.8. Diese Adresse kann sich ändern, wenn sie nicht vom WLAN-Router gesperrt ist;
- durch den DNS-Namen, der bei der Verbindung mit dem WLAN-Netzwerk angegeben wurde. Standardmäßig weist ESP32forth den Namen **forth** , die eine Verbindung zum WLAN-Netzwerk herstellt.

forth Hostnamen anstelle der IP-Adresse zu verwenden :

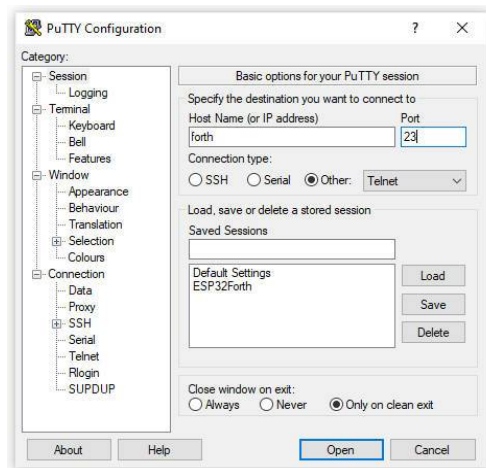


Figure 12: Verwenden Sie den DNS-Namen mit PuTTY

Wenn Sie mit mehreren ESP32-Karten im selben Netzwerk kommunizieren möchten, muss jede Karte einen eindeutigen Hostnamen deklarieren. Beispielcode für zwei ESP32-Karten:

```
\ forthCOM3 für die erste ESP32-Karte
z" Mariloo"
z" 1925144D91DE5373C3C2D7XXXX"
login
z" forthCOM3" MDNS.begin
cr telnetd 552 server
```

Code für die zweite ESP32-Karte:

```
\ forthCOM6 für die zweite ESP32-Karte
z" Mariloo"
z" 1925144D91DE5373C3C2D7959F"
login
z" forthCOM6" MDNS.begin
cr telnetd 552 server
```

Das Ausführen dieses Codes auf jeder Karte wirkt sich auf die Hostnamen **forthCOM3** und **forthCOM6** im internen Netzwerk aus.

Herstellen einer Verbindung zu ESP32-Boards über deren Hostnamen

Starten Sie PuTTY. Wir geben den Hostnamen und den offenen Port ein, um auf **forthCOM3** zuzugreifen :

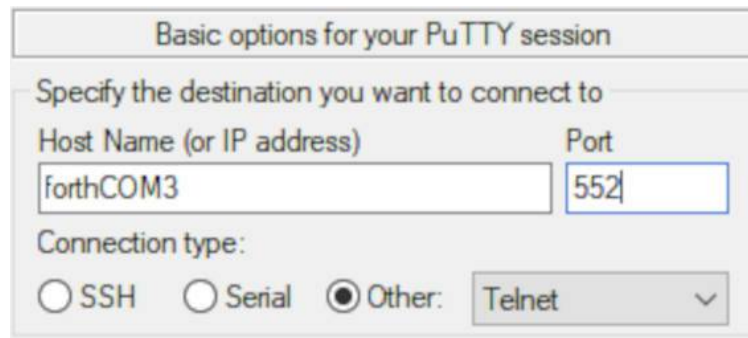


Figure 14: accès de PuTTY à forthCOM3

Dann starten wir eine neue PuTTY-Sitzung und ändern einfach den Hostnamen für diese Sitzung, hierher **forthCOM6** . Hier sind zwei PuTTY-Sitzungen, mit denen Sie mit diesen beiden ESP32-Karten kommunizieren können:

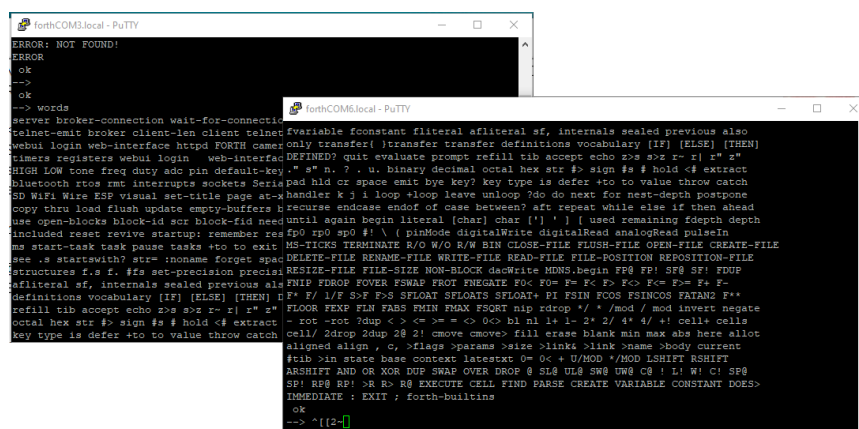


Figure 13: PuTTY greift auf zwei separate ESP32-Boards zu

Um den TELNET-Client automatisch auf der ESP32-Karte zu starten, integrieren wir unseren Verbindungscode in **autoexec.fs** . Hier ist der Code, den Sie über das Terminal eingeben müssen. Erster Typ:

```
visual edit /spiffs/autoexec.fs
```

Geben Sie dann diese wenigen Zeilen ein:

```
\ forthCOM3 für die erste ESP32-Karte
z" Mariloo"
z" 1925144D91DE5373C3C2DXXXXX"
login
z" forthCOM3" MDNS.begin
cr telnetd 552 server
forth
```

Anschließend CTRL-X und Y ausführen. Der Code wird gespeichert und beim nächsten Start von ESP32forth geladen. Der TELNET-Client wird automatisch neu gestartet, wenn ESP32forth startet. Es ist nicht mehr notwendig, das Terminal für die Kommunikation mit der mit dem Hostnamen **forthCOM3** deklarierten ESP32-Karte zu verwenden :

- Trennen Sie die ESP32-Karte.
- Schließen Sie die ESP32-Karte wieder an, aber öffnen Sie das Terminal nicht!
- warte ein paar Sekunden...
- Starten Sie puTTY und aktivieren Sie eine TELNET-Verbindung mit **forthCOM3** auf Port 552

Der TELNET-Zugriff über PuTTY ermöglicht die gleichen Vorgänge wie über das Terminal. Einzige Einschränkung: Wenn Sie FORTH-Code per Kopieren/Einfügen übertragen, begrenzen Sie die Größe des übertragenen Codes.

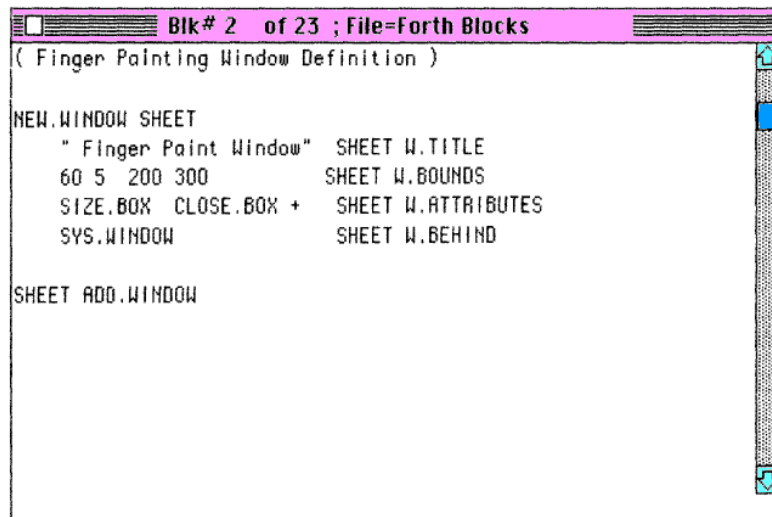
HINWEIS: Auf so konfigurierte ESP32-Karten kann aus dem Internet zugegriffen werden, wenn die Konfiguration des WLAN-Routers dies zulässt.

Verwaltung von Quelldateien nach Blöcken

Verwenden Sie den **editor** nur für Blockdateibearbeitungen im SPIFFS-Dateisystem.
Verwenden Sie zuerst **RECORDFILE**. Siehe Kapitel *RECORDFILE*.

Die Blöcke

Hier ein Block auf einem alten Computer :



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:

- **Mit 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 **l** . 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

Verwenden Sie den **visual edit** nur für Blockdateibearbeitungen im SPIFFS-Dateisystem. Verwenden Sie zuerst **RECORDFILE**. Siehe Kapitel *RECORDFILE*.

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/MPETREMAN11/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.

RECORDFILE- und FORTH-Projektmanagement

Dieses Kapitel ist einem einzigen Schlüsselement gewidmet: **RECORDFILE** . Dieses Wort ermöglicht das schnelle Speichern von Dateien im SPIFFS-Dateisystem.

Ich empfehle Ihnen, es sorgfältig zu lesen, bevor Sie versuchen, Quelldateien mit **visual** oder **editor** zu bearbeiten.

Hier erfahren Sie Schritt für Schritt, wie Sie die Definition von **RECORDFILE** speichern und dann effektiv verwenden.

Speichern Sie RECORDFILE in der Datei autoexec.fs

Wenn ESP32forth startet, prüft das System, ob die Datei **autoexec.fs** vorhanden ist . Wenn diese Datei vorhanden ist, wird ihr Inhalt interpretiert.

Hier ist der Quellcode für **RECORDFILE**. Diese Definition wurde von Bob EDWARDS entwickelt. Kopieren Sie diesen Code und fügen Sie ihn in das Terminalfenster ein, um ihn zu kompilieren. Dieses Manöver muss nur einmal durchgeführt werden :

```
\ These chars terminate all text lines in a file
create crlf 13 C, 10 C,

\ Records the input stream to a spiffs file until
\ an <EOF> marker is encountered, then close file
: RECORDFILE ( "filename" "filecontents" "<EOF>" -- )
  bl parse          \ read the filename ( a n )
  W/O CREATE-FILE throw >R    \ create the file to record to -
                                \ put file id on R stack
  BEGIN
    \ read a line of the file from the input stream
    tib #tib accept
    tib over
    S" <EOF>" startswith? \ does the line start with <EOF> ?
    DUP IF
      \ Yes, so drop the end line of the file containing <EOF>
      swap drop
    ELSE
      swap
      tib swap
      \ No, so write the line to the open file
      R@ WRITE-FILE throw
      \ and terminate line with cr-lf
      crlf 2 R@ WRITE-FILE throw
    THEN
  UNTIL                \ repeat until <EOF> found
```

```
R> CLOSE-FILE throw          \ Close the file
;
```

Sobald dieses Wort kompiliert ist, werden wir sehen, wie wir vorgehen, damit dieses Wort dauerhaft in **autoexec.fs** verfügbar ist.

Erstellen Sie auf Ihrem PC in Ihrem ESP32Forth-Entwicklungsbereich eine **autoexec.fs**-Datei.

Kopieren Sie den **RECORDFILE**- Code wie oben angegeben in diese **autoexec.fs**- Datei. Fügen Sie diese beiden Codezeilen hinzu:

```
RECORDFILE /spiffs/autoexec.fs
\ These chars terminate all text lines in a file
create crlf 13 C, 10 C,

\ Records the input stream to a spiffs file until
\ an <EOF> marker is encountered, then close file
: RECORDFILE ( "filename" "filecontents" "<EOF>" -- )
  bl parse          \ read the filename ( a n )
  W/O CREATE-FILE throw >R    \ create the file to record to -
                                \ put file id on R stack
  BEGIN
    \ read a line of the file from the input stream
    tib #tib accept
    tib over
    S" <EOF>" startswith? \ does the line start with <EOF> ?
    DUP IF
      \ Yes, so drop the end line of the file containing <EOF>
      swap drop
    ELSE
      swap
      tib swap
      \ No, so write the line to the open file
      R@ WRITE-FILE throw
      \ and terminate line with cr-lf
      crlf 2 R@ WRITE-FILE throw
    THEN
  UNTIL              \ repeat until <EOF> found
  R> CLOSE-FILE throw \ Close the file
;
```

<EOF>

Kopieren Sie diesen Quellcode erneut, einschließlich der roten Codezeilen. Fügen Sie diesen Code erneut in das Terminalfenster ein. Übertragen Sie diesen Code an die ESP32-Karte.

Im Gegensatz zur ersten Manipulation, bei der der Code kompiliert wurde, wird dieser Code dieses Mal in der Datei **/sipffs/autoexec.fs** gespeichert.

Datei **autoexec.fs** gespeichert ist, führen Sie **ls** aus :

```
ls /sipffs/
```

Datei **autoexec.fs** in der Dateiliste erscheinen. Um den Inhalt von **autoexec.fs** zu überprüfen, geben Sie Folgendes ein :

```
cat /sipffs/autoexec.fs
```

Dadurch sollte der Inhalt von **autoexec.fs** angezeigt werden.

Verwenden Sie geänderte Inhalte der Datei autoexec.fs

Starten Sie ESP32forth neu. Wenn alles gut gelaufen ist, ist **RECORDFILE** jetzt verfügbar, wenn ESP32forth startet. **Wörter** ausführen . Sie sollten **RECORDFILE** in den ersten Wörtern des FORTH-Wörterbuchs finden :

```
RECORDFILE crlf FORTH spi oled telnetd registers webui login  web-interface  
httpd ok LED OUTPUT INPUT HIGH LOW tone freq duty adc pin default-key?  
default-key default-type visual set-title page at-xy normal bg fg ansi...
```

Überladen Sie **autoexec.fs** nicht mit anderen Definitionen. Wir werden sehen, wie man ein Projekt erstellt.

Ein Projekt mit ESP32forth aufschlüsseln

Auf Ihrem PC wird ein FORTH-Entwicklungsprojekt für ESP32forth erstellt:

- Bearbeiten des Quellcodes mit dem Texteditor Ihrer Wahl oder einer IDE (z. B. Netbeans);
- über ein Terminal verfügen, das über USB mit der ESP32-Karte verbunden ist;
- Habe ESP32forth auf der ESP32-Karte aktiviert.

Arbeiten Sie am PC strukturiert. Die folgenden Erläuterungen sind lediglich Empfehlungen.

Beginnen Sie mit der Definition des allgemeinen Arbeitsverzeichnisses für alle ESP32forth-Entwicklungen. Zum Beispiel ein Ordner mit dem Namen **ESP32forth** developments.

Erstellen Sie dann in diesem Ordner zwei weitere Ordner:

- **_Meine Projekte** , das alle Ihre Projekte aufnehmen soll;
- **_sandbox** , die alle kleinen zu testenden Programme aufnehmen soll, die keinen bestimmten Zweck haben;
- **Tools** , die alle Quelldateien von allgemeinem Interesse aufnehmen sollen. Dabei handelt es sich um getestete Dateien, die keiner Anpassung bedürfen;
- **Dokumentation** , die für Dokumente jeglicher Art bestimmt ist.

Beispielprojekt

Ich werde den TEMPVS FVGIT-Quellcode als Beispielprojekt verwenden. Die vollständigen Quellcodes sind hier verfügbar:

https://github.com/MPETREMAN11/ESP32forth/tree/main/_my%20projects/display/OLED%20SSD1306%20128x32/TEMPVS%20FVGIT

Die erste zu erstellende Datei heißt **main.fs**. Diese Datei muss in einen TEMPVS FVGIT-Ordner geschrieben werden:

```
ESP32forth developments
+-----> _my Projects
      +-----> TEMPVS FVGIT
            +-----> main.fs
                        config.fs
                        strings.fs
```

Auch hier handelt es sich lediglich um Empfehlungen. Das Hauptinteresse besteht darin, alle Komponenten eines einzigen Projekts zusammenzuführen. Inhalt der Datei **main.fs** :

```
RECORDFILE /spiffs/main.fs
DEFINED? --tempusFugit [if] forget --tempusFugit [then]
create --tempusFugit

s" /spiffs/strings.fs"      included
s" /spiffs/RTClock.fs"      included

s" /spiffs/clepsydra.fs"    included

s" /spiffs/config.fs"       included
s" /spiffs/oledTools.fs"    included
( part of code removed here )
<EOF>
```

In Rot finden wir unser Wort **RECORDFILE**. Um den Code von **main.fs** im SPIFFS-Dateisystem auf dem ESP32-Board zu speichern, kopieren Sie einfach diesen Quellcode und übergeben Sie ihn mit dem Terminalprogramm an ESP32forth.

In Blau im obigen Code ruft der Inhalt von **main.fs** die Datei strings.fs auf. Der Quellcode für diese Datei stammt aus dem **Tools**- Ordner. Es handelt sich um eine Kopie von **strings.fs**, die dann wie folgt geändert wird:

```
RECORDFILE /spiffs/strings.fs
structures
struct __STRING
    ptr field >maxLength    \ point to max length of string
    ptr field >realLength    \ real length of string
    ptr field >strContent    \ string content
```

```

forth
( ... removed part of file )
\ work only with strings. Don't use with other arrays
: input$ { addr len -- }
  addr len maxlen$ nip accept
  addr __STRING - cell+ >realLength !
;
<EOF>

```

Durch Kopieren und Übergeben dieses Quellcodes wird die Datei **strings.fs** im SPIFFS-Dateisystem auf der ESP32-Karte erstellt.

Zu diesem Zeitpunkt beginnen wir, mehrere Dateien auf der ESP32-Karte zu haben. Um alle übertragenen Dateien zu kompilieren, führen wir einfach Folgendes aus:

```
/spiffs/main.fs einschließen
```

Abgesehen von der physischen Speicherplatzbeschränkung gibt es keine Begrenzung für die Dateien, die auf der ESP32-Karte gespeichert werden können. Der verfügbare Speicherplatz im SPIFFS-Dateisystem übersteigt 1 MB Aufnahmespeicher.

Wenn Sie den Inhalt einer allgemeinen Softwarekomponente ändern müssen, tun Sie dies immer in einer Kopie der Quelldatei für diese Komponente. Denken Sie daran, diese Änderungen zu versionieren und zu datieren.

Für jede der Dateien in diesem Projekt integrieren wir **RECORDFILE** und seinen **<EOF>**-Terminator .

In jedem Projekt finden wir die Dateien **main.fs** und **config.fs** . Ihr Inhalt wird jedoch an jedes Projekt angepasst. Für ein bestimmtes Projekt werden alle **fs**- Erweiterungsdateien im SPIFFS-Dateisystem auf das ESP32-Board geladen. Die Zusammenstellung ihrer Inhalte geht unglaublich schnell. Vor allem aber **bleibt der Inhalt** dieser Dateien zwischen zwei Neustarts der ESP32-Karte erhalten. Bei der geringsten Blockierung von FORTH ist es

einfach, die Karte neu zu starten und alle Wortdefinitionen des Projekts zu finden, ohne dass eine erneute Übertragung über das Terminal erforderlich ist.

Die Vorstellung einer Black Box

Es ist ein altes Konzept, das aus der Zeit stammt, als wir hauptsächlich in Assembler auf Mikrocontrollerkarten entwickelten. Wir finden es auch bei Klassen in der Objektprogrammierung. Im Konzept der „Black Box“ müssen wir ein Unterprogramm, eine Funktion, eine Methode als Black Box betrachten. Wir wissen, was wir dort hineingesteckt haben. Wir wissen, was dabei herauskommen kann oder wie diese Box funktioniert, machen uns aber keine Sorgen um ihre interne Funktionsweise. Wir vertrauen denen, die die „Black Box“ programmiert haben.

In der FORTH-Sprache hat ein Wort eine Definition. Wenn Sie Parameter über den Stapel übergeben, muss letztendlich nur der Designer der Definition dafür sorgen, dass die Definition ordnungsgemäß funktioniert. Und um das ordnungsgemäße Funktionieren einer Definition sicherzustellen, wird dringend empfohlen, keine zu langen Definitionen zu erstellen.

Mein Tipp zum Markieren von verifiziertem Code ist einfach, direkt vor der Definition eine Kommentarzeile einzufügen. Beispiel für nicht verifizierten Code:

```
: fpi* ( fn - fn*pi )
  pi f*
;
```

Der Code wird in einer *Sandbox* oder in einer Projektdatei getestet. Egal. Nachdem ich es mit verschiedenen

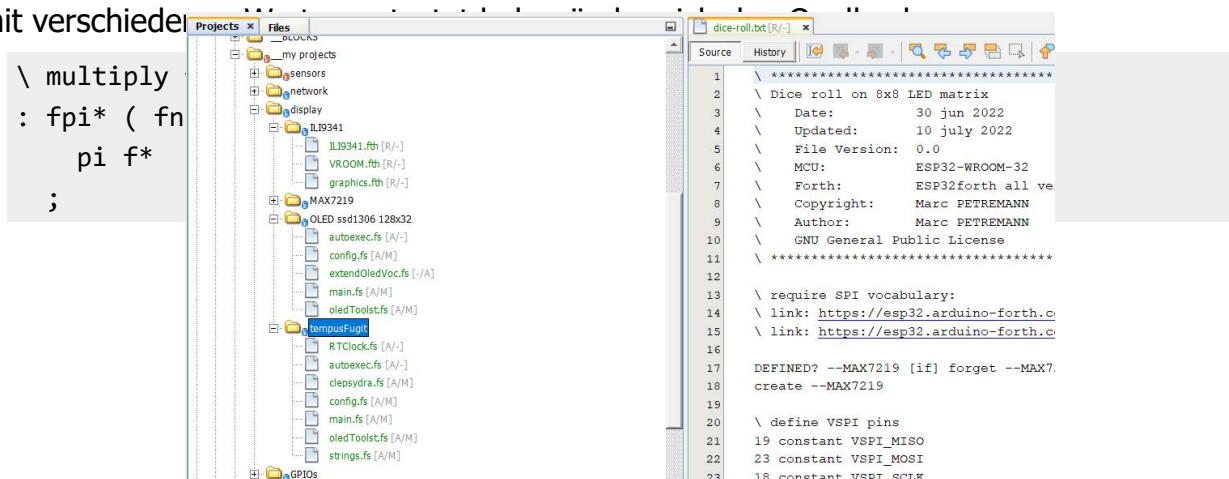


Figure 15: Strukturierung von Projekten mit der Netbeans IDE

Wenn Sie Zweifel an der Zuverlässigkeit Ihres Codes haben, können Sie eine Datei **tests.fs** definieren.

Diese Datei ist nur für die Durchführung von Unit-Test-Batterien gedacht. Sehen Sie sich die Definition des Wortes **asserte**(an , mit dem diese Tests durchgeführt werden. Die Definition ist hier sichtbar:

<https://github.com/MPETREMAN11/ESP32forth/blob/main/tools/assert.fs>

Und hier ist ein Beispiel für Tests, die in unserer Datei **tests.fs** gespeichert sind :

```
assert( 0 >gray 0 = )
assert( 1 >gray 1 = )
assert( 2 >gray 3 = )
assert( 3 >gray 2 = )
assert( 4 >gray 6 = )
assert( 5 >gray 7 = )
assert( 6 >gray 5 = )
assert( 7 >gray 4 = )
```

assert(generiert eine Warnung, wenn sich das getestete Wort nicht wie erwartet verhält.

Für eine so einfache Definition wie die von **fpi*** ist möglicherweise eine Reihe von Tests erforderlich, wenn wir das Wort **f*** nicht zuverlässig gemacht haben . Wir integrieren diese Tests in die Datei **main.fs** :

```
RECORDFILE /spiffs/main.fs
DEFINED? --tempusFugit [if] forget --tempusFugit [then]
create --tempusFugit

s" /spiffs/strings.fs"      included
s" /spiffs/RTClock.fs"      included

s" /spiffs/clepsydra.fs"    included

s" /spiffs/config.fs"       included
s" /spiffs/oledTools.fs"    included

s" /spiffs/tests.fs"       included
<EOF>
```

Selbstverständlich muss auch der Inhalt der Datei **tests.fs** auf die ESP32-Karte übertragen werden.

Auf diese Weise umfasst der gesamte Kompilierungszyklus auch eine Reihe von Tests. Tests garantieren nicht, dass der Code zuverlässig ist. Sie ermöglichen es nur, mögliche Nebenwirkungen zu erkennen, wenn wir Teile des Anwendungscodes ändern müssen.

Zusammenfassend empfehle ich, Ihren Code zu fragmentieren, indem Sie diese Dateien systematisch integrieren:

- **main.fs** , die Hauptdatei. Unabhängig vom Namen des Projekts kompilieren Sie es normalerweise mit einer einfachen Ausführung von **include /spiffs/main.fs**
- **config.fs** , das die globalen Konfigurationsparameter enthält, zum Beispiel WLAN-Zugangskennwörter;
- **tests.fs** , das eine Reihe von Tests enthält. Wenn Sie keine Tests durchführen, ist die Erstellung dieser Datei nicht erforderlich.

Alle anderen Dateien haben die Erweiterung **fs** , mit Ausnahme der Dateien, die nicht von ESP32forth verarbeitet werden.

Wenn Sie diese wenigen Richtlinien befolgen, wird es Ihnen leichter fallen, komplexe Anwendungen zu verwalten. Das Hauptargument für die Einführung **von RECORDFILE** ist die Zeitersparnis bei der Kompilierung und Speicherung zuverlässiger Teile des Codes in Dateien im SPIFFS-Dateisystem.

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
```

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.

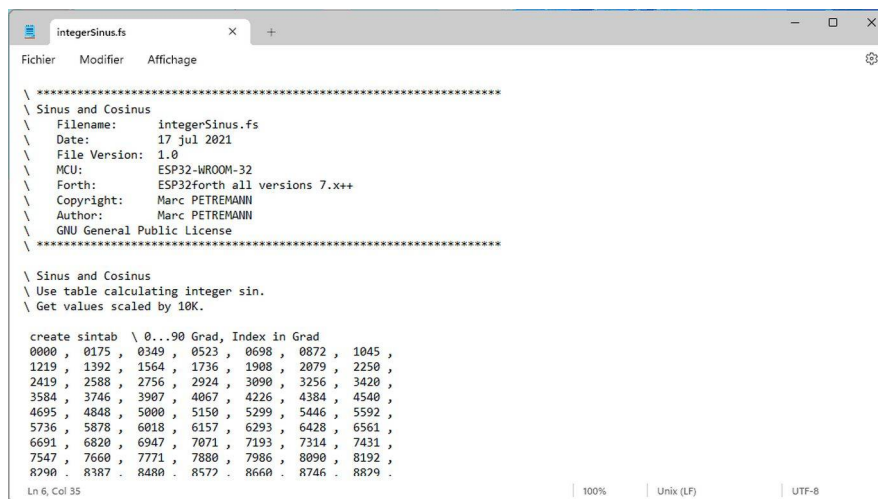
Bearbeiten und Verwalten von Quelldateien für ESP32forth

Wie bei den meisten Programmiersprachen liegen in der FORTH-Sprache geschriebene Quelldateien im einfachen Textformat vor. Die Erweiterung von Dateien in der FORTH-Sprache ist kostenlos:

- **txt-** Erweiterung für alle Textdateien;
- **forth** wird von einigen FORTH-Programmierern verwendet;
- **fth** komprimierte Form für FORTH;
- **4th** andere komprimierte Form für FORTH;
- **fs** unsere Lieblingserweiterung...

Editoren für Textdateien

Editor edit unter Windows, der einfachste:



```
\
\ Sinus and Cosinus
\
\ Filename: integerSinus.fs
\ Date: 17 jul 2021
\ File Version: 1.0
\ MCU: ESP32-WROOM-32
\ FORTH: ESP32FORTH all versions 7.x++
\ Copyright: Marc PETREMANN
\ Author: Marc PETREMANN
\ GNU General Public License
\
\ Sinus and Cosinus
\ Use table calculating integer sin.
\ Get values scaled by 10K.

create sintab \ 0...90 Grad, Index in Grad
0000 , 0175 , 0349 , 0523 , 0698 , 0872 , 1045 ,
1219 , 1392 , 1564 , 1736 , 1908 , 2079 , 2250 ,
2419 , 2588 , 2756 , 2924 , 3090 , 3256 , 3420 ,
3584 , 3746 , 3907 , 4067 , 4226 , 4384 , 4540 ,
4695 , 4848 , 5000 , 5150 , 5299 , 5446 , 5592 ,
5736 , 5878 , 6018 , 6157 , 6293 , 6428 , 6561 ,
6691 , 6820 , 6947 , 7071 , 7193 , 7314 , 7431 ,
7547 , 7660 , 7771 , 7880 , 7986 , 8090 , 8192 ,
8290 , 8387 , 8480 , 8572 , 8660 , 8746 , 8829 ,
```

Bearbeiten mit Bearbeiten unter Windows 11

Andere Editoren wie **WordPad** werden nicht empfohlen, da Sie das Risiko eingehen, den Quellcode der FORTH-Sprache in einem Dateiformat zu speichern, das nicht mit ESP32forth kompatibel ist.

Unter Linux heißt das Äquivalent **gEdit**. MacOS verfügt außerdem über einen einfachen Texteditor.

Wenn Sie eine benutzerdefinierte Dateierweiterung wie **fs** für Ihre FORTH-Sprachquelldateien verwenden, muss diese Dateierweiterung von Ihrem System erkannt werden, damit sie vom Texteditor geöffnet werden können.

Verwenden Sie eine IDE

Nichts hindert Sie daran, eine IDE zu verwenden³. Ich für meinen Teil bevorzuge **Netbeans**, das ich auch für PHP, MySQL, Javascript, C, Assembler verwende ... Es ist eine sehr leistungsstarke IDE und genauso effizient wie **Eclipse** :

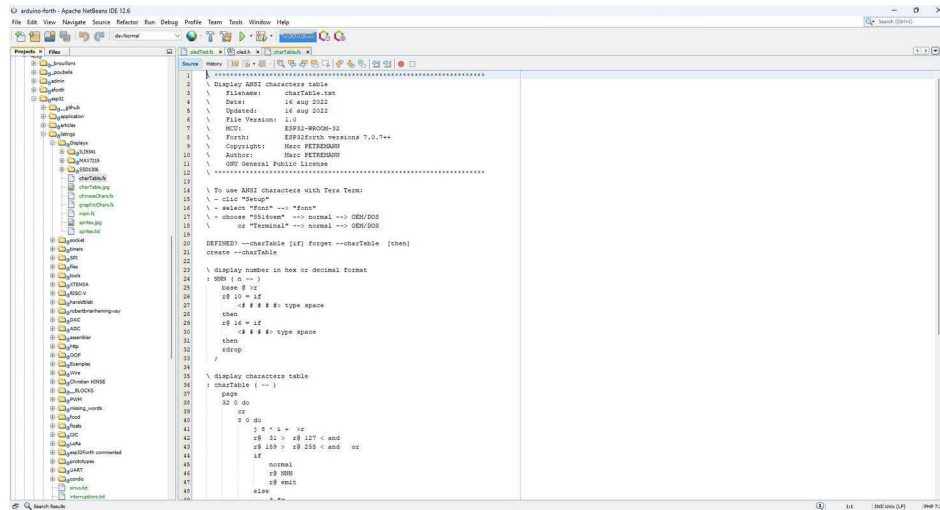


Figure 16: Bearbeiten mit Netbeans

Netbeans bietet mehrere interessante Funktionen:

- Versionsverwaltung mit **GIT** ;
- Wiederherstellung früherer Versionen geänderter Dateien;
- Dateivergleich mit **Diff** ;
- One-Click- **FTP**- Übertragung zum Online-Hosting Ihrer Wahl;

3 Integrierte Entwicklungsumgebung

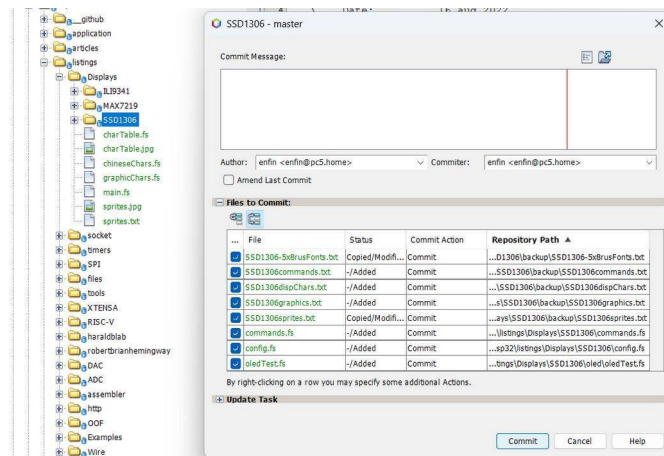
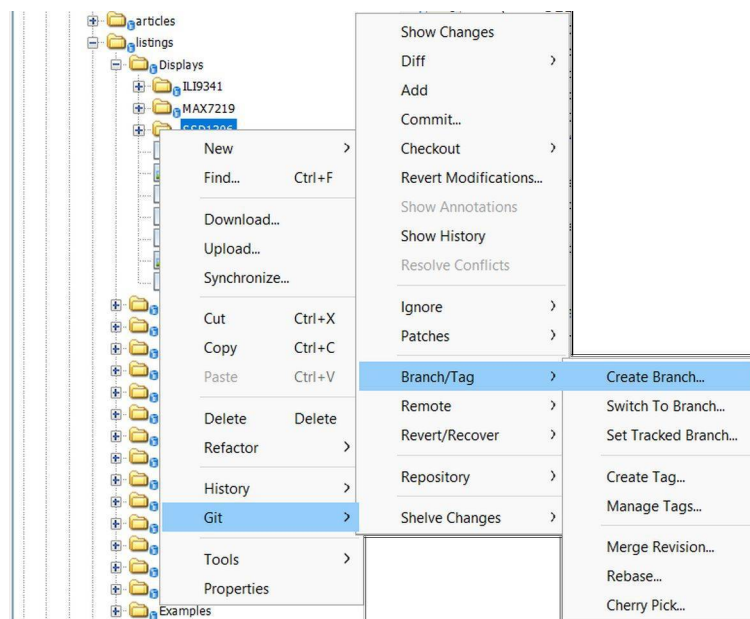


Figure 17: GIT-Commit-Vorgang in Netbeans eines Ordners

GIT- Option besteht die Möglichkeit, Dateien in einem Repository zu teilen und die Zusammenarbeit bei komplexen Projekten zu verwalten. Mit **GIT** können Sie lokal oder kollaborativ verschiedene Versionen desselben Projekts verwalten und diese Versionen dann zusammenführen. Sie können Ihr lokales GIT-Repository erstellen. Jedes *Mal, wenn* eine Datei oder ein komplettes Verzeichnis festgeschrieben wird, bleiben die Entwicklungen unverändert. Auf diese Weise können Sie alte Versionen derselben Datei oder desselben Dateionders finden.

Mit NetBeans können Sie einen Entwicklungszweig für ein komplexes Projekt definieren. Hier erstellen wir einen neuen Zweig:



Erstellen einer Verzweigung für ein Projekt

Beispiel einer Situation, die die Gründung einer Zweigstelle rechtfertigt:

- Sie haben ein funktionierendes Projekt;
- Sie planen, es zu optimieren;

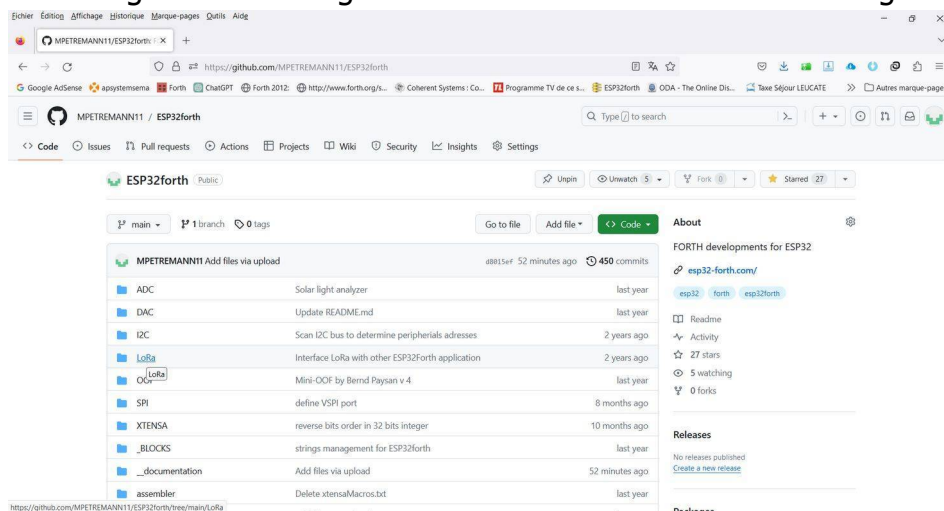
- Erstellen Sie einen Zweig und führen Sie die Optimierungen in diesem Zweig durch.

Änderungen an Quelldateien in einem Zweig haben keinen Einfluss auf Dateien im *main*.

Im Übrigen ist es mehr als ratsam, über physische Backup-Medien zu verfügen. Eine SSD-Festplatte kostet für 300 GB Speicherplatz etwa 50 €. Die Lese- oder Schreibzugriffsgeschwindigkeit von SSD-Medien ist einfach erstaunlich!

Speicherung auf GitHub

GitHub⁴-Website ist neben **SourceForge**⁵ einer der besten Orte zum Speichern von Quelldateien. Auf GitHub können Sie einen Arbeitsordner mit anderen Entwicklern teilen und komplexe Projekte verwalten. Der Netbeans-Editor kann eine Verbindung zum Projekt herstellen und ermöglicht das Übergeben oder Abrufen von Dateiänderungen.

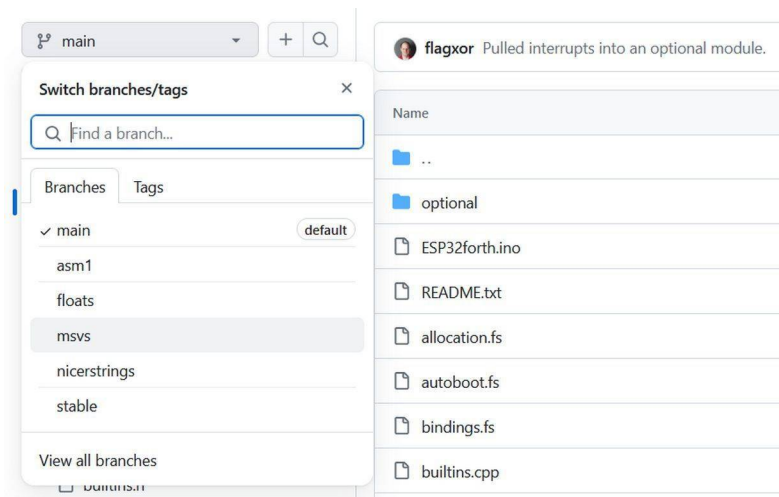


Speichern von Dateien auf GitHub

Auf **GitHub** können Sie Projektzweige (*fork*). Sie können bestimmte Teile Ihrer Projekte auch vertraulich behandeln. Hier die Zweige in den **flagxor/ueforth**-Projekten:

⁴ <https://github.com/>

⁵ <https://sourceforge.net/>



Zugriff auf einen Projektzweig

Einige gute Praktiken

Die erste gute Vorgehensweise besteht darin, Ihre Arbeitsdateien und Ordner richtig zu benennen. Sie entwickeln für ESP32Forth. Erstellen Sie daher einen Ordner mit dem Namen **ESP32forth**.

Erstellen Sie für verschiedene Tests einen **Sandbox**- Unterordner in diesem Ordner.

Erstellen Sie für gut strukturierte Projekte einen Ordner pro Projekt. Wenn Sie beispielsweise einen Roboter steuern möchten, erstellen Sie einen **robot**- Unterordner.

Tools- Unterordner. Wenn Sie eine Datei aus diesem Tools-Ordner in einem Projekt verwenden, kopieren Sie diese Datei und fügen Sie sie in den Ordner dieses Projekts ein. Dadurch wird verhindert, dass eine Änderung einer Datei in **Tools** Ihr Projekt später stört.

Die zweite Best Practice besteht darin, den Quellcode eines Projekts in mehrere Dateien zu verteilen:

- **config.fs** zum Speichern von Projekteinstellungen;
- **documentation** Verzeichnis zum Speichern von Dateien im Format Ihrer Wahl, die sich auf die Projektdokumentation beziehen;
- **myApp.fs** für Ihre Projektdefinitionen. Wählen Sie einen ziemlich eindeutigen Dateinamen. Um beispielsweise einen Roboter zu verwalten, verwenden Sie den Namen **robot-commands.fs**.

..	
LOTTOinterface.jpg	Add files via upload
README.md	Create README.md
euroMillionFR.fs	LOTO wining combinaisons numbers
generalWords.fs	general words for LOTTO program
gridsManage.fs	Manage content of LOTTO grids
interface.fs	text interface for LOTTO program
main.fs	LOTTO game main file
numbersFrequency.fs	stats frequency for LOTTO numbers

Figure 18: Beispiel für die Benennung einer FORTH-Quelldatei

Der Inhalt dieser Dateien muss über das Terminal auf die ESP32-Karte übertragen werden, damit ESP32forth den FORTH-Code interpretiert und kompiliert.

Die main.fs-Datei

ESP32forth verwaltet ein SPIFFS-Dateisystem ⁶. Siehe das Kapitel *Das SPIFFS-Dateisystem*.

Diese Dateien werden daher auf der ESP32-Karte gespeichert und können von ESP32forth gelesen werden. Wenn Sie eine **config.fs- Datei** im SPIFFS-Dateisystem geschrieben haben, ist hier die Codezeile, die Sie in **main.fs schreiben müssen, um auf den Inhalt von config.fs** zuzugreifen :

```
s" /spiffs/config.fs" included
```

Ab diesem Zeitpunkt haben Sie zwei Möglichkeiten, den Inhalt von **config.fs zu interpretieren** . Vom Terminal aus:

```
include /spiffs/config.fs
```

Oder

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

Der Punkt ist, dass **main.fs** andere Dateien aufrufen kann. Beispiel :

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

Die Verarbeitung vieler Dateien dauert weniger als eine Sekunde. Diese Strategie vermeidet die wiederholte Übertragung des Quellcodes per serieller Verbindung über das Terminal.

Und wenn Sie mehrere Projekte auf mehreren ESP32-Boards verwalten, ist es einfacher, jedes Projekt mit einem einfachen Befehl **include /spiffs/main.fs** zu testen .

⁶ Flash-Dateisystem für serielle Peripherieschnittstellen

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:



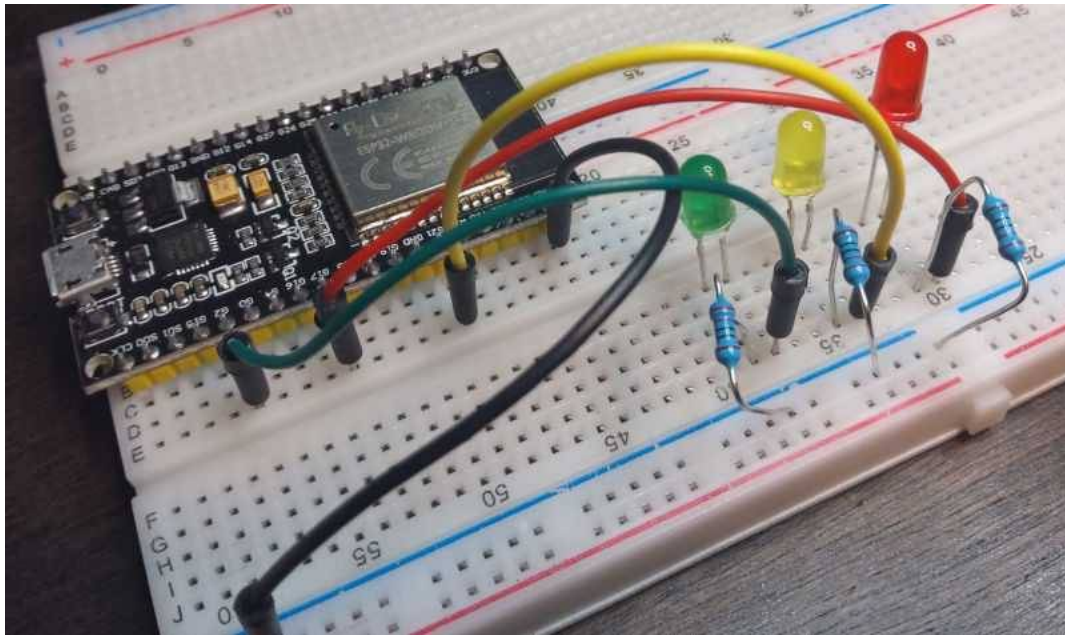
Jeder Anschluss ist durch eine Reihe von Buchstaben und Zahlen gekennzeichnet, hier von links nach rechts auf unserem Foto: G22 TXD RXD G21 GND G19 G18 usw.

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.

Direkter Zugriff auf GPIO-Register

In manchen Situationen ist es viel vorteilhafter, direkten Zugriff auf die GPIO-Register auf dem ESP32-Board zu haben. Beispielsweise zur Verwaltung komplexer Aktivierungs- oder Deaktivierungssequenzen.

Mit ESP32forth erfolgt der Zugriff auf ein ESP32-Register mit m Wörtern **m!** und **M@** . Diese Wörter ermöglichen insbesondere den direkten Zugriff auf GPIO-Register.

Das GPIO-Register, das die ersten 32 Ein-/Ausgänge verwaltet, befindet sich an der hexadezimalen Adresse 3ff44004:

```
$3ff44004 defREG: GPIO_OUT_REG
```

Wenn wir eine LED an den GPIO2-Port anschließen, können wir sie wie folgt ein- und ausschalten:

```
0 GPIO_OUT_REG m! \ LED an G2 ausschalten
4 GPIO_OUT_REG m! \ LED an G2 einschalten
```

Der Nachteil, in der Reihenfolge **4 GPIO_OUT_REG m!** , dadurch wird nur der G2-Port aktiviert. Wenn andere Ports aktiv waren, werden diese deaktiviert. Die Lösung, die mir in den Sinn kommt, wäre daher, den Zustand des **GPIO_OUT_REG**-Registers mit **m@** zu lesen und logische Operationen mit dem Wert durchzuführen, bevor er ihn mit **m!** erneut einfügt.

Es stellt sich heraus, dass die ESP32-Karte über dedizierte Register verfügt, die diese selektiven Aktivierungs- und Deaktivierungsvorgänge durchführen, ohne diese logischen Operationen an diesem **GPIO_OUT_REG**-Register durchzuführen.

Verwendung von Wörtern m! und M@

Diese beiden Wörter sind im **Register** vokabular definiert. Dies sind auch die einzigen in diesem Vokabular definierten Wörter :

- **m!** (val shift mask addr --)
ändert den Inhalt eines Registers, auf das **addr** zeigt , wendet eine logische Maske mit **mask** an und verschiebt **val** entsprechend der **shift** um n Bits ;
- **m@** (shift mask addr -- val)
liest den Inhalt eines Registers, auf das **addr** zeigt , wendet eine logische Maske mit **mask** an und verschiebt entsprechend **shift** um n Bits.

Um die Funktionsweise dieser beiden Wörter vollständig zu verstehen, definieren wir zunächst ein Wort, das den 32-Bit-Inhalt einer beliebigen Adresse anzeigt :

```
\ n im Format bbbbbbbb bbb..... anzeigen
: .binDisp ( n -- )
  base @ >r binary \ Binärbasis auswählen
```

```

<#                \Nummernformatierung starten
4 for
  aft
    8 for
      aft # then
    next
  bl hold        \füge 'Leerzeichen' in der Zahlenformatierung hinzu
then
next
#>
cr space ." 33222222 22221111 11111100 00000000"
cr space ." 10987654 32109876 54321098 76543210"
cr type          \zeigt n im Binärformat an
r> base !        \aktuelle numerische Basis wiederherstellen
;

```

Definieren wir einen beliebigen Speicherplatz, initialisiert mit einem Nullwert:

```

create myReg
0 ,

```

So zeigen Sie den Inhalt von **myReg** mit **.binDisp** an :

```

myReg @ .binDisp    \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 00000000 00000000 00000000 ok

```

Bit b22 wurde hervorgehoben. Hier liegt er bei Null. ACHTUNG: Die Bits sind in der von **.binDisp** gerenderten Anzeige von 0 bis 31 nummeriert . So setzen Sie dieses einzelne B22-Bit auf eins:

```

registers
1 22 $ffffffff myReg m!
forth
myReg @ .binDisp    \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 01000000 00000000 00000000 ok

```

Es ist fertig. Aber wir haben den 32-Bit-Wert **\$ffffffff** als Maske genommen . Die Wahl dieser Maske ermöglicht leider eine Aktion auf alle Teile des Inhalts von **myReg** . Wenn wir nur auf ein oder mehrere Bits einwirken wollen, müssen wir eine Maske wählen, die die Wirkung des Wortes **m!** begrenzt. Um beispielsweise das einzelne Bit b07 auf 1 zu setzen, müssen Sie eine Maske verwenden. Hier ist diese Maske im Binärformat:

```

00000000 00000000 00000000 10000000

```

Was hexadezimal zu **\$00000080** übersetzt wird . Wenn Sie Zweifel haben, können Sie dies mit **.binDisp** überprüfen :

```
$00000080 .binDisp      \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 00000000 00000000 10000000 ok
```

Diese Binärmaske entspricht Bit b07. Wir werden dieses Bit nun in **myReg** auf 1 setzen, ohne das andere Bit b22 zu ändern, das bereits auf eins gesetzt ist :

```
registers
1 7 $00000080 myReg m!
forth
myReg @ .binDisp      \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 01000000 00000000 10000000 ok
```

Zu den Parametern, die für **m!**, wir haben das Paar Verschiebungsadr. Um den ESP32forth-Code zu vereinfachen, erstellen wir ein Wort **defMASK:**:

```
\ definiere eine Maske für Register
: defMASK: ( comp: mask0 position -- <name> | exec: -- position mask1 )
  create
    dup ,
    lshift ,
  does>
    dup @
    swap cell + @
;
```

Um eine Maske zu definieren, benötigen Sie lediglich zwei Parameter:

- **mask0** : 1 ist die minimale Maske für ein Bit, 3 für 2 Bits, 7 für 3 Bits usw.
- **position** : Gibt die Position im Intervall [0..31] an.

Um beispielsweise das einzelne Bit b12 zu modifizieren, können wir unsere Maske wie folgt definieren :

```
1 12 defMASK: mB12
```

Um dieses einzelne Bit b12 auf 1 zu setzen:

```
registers
1 mB12 myREG m!
forth
myREG @ .binDisp      \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 01000000 00010000 10000000 ok
```

Um dieses Bit b12 zurückzusetzen:

```

registers
0 mB12 myREG m!
forth
myREG @ .binDisp      \ display :
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 01000000 00000000 10000000 ok

```

defMASK: definierte Masken gelten für jedes Register. Das werden Sie später sehen. Diese Masken sind auch sehr nützlich, um den Zustand eines oder mehrerer bestimmter Bits zu bestimmen. Testen wir den Zustand unseres Bits b12 mit **m@** :

```

registers
mB12 myREG m@ .      \ display : 0
forth

```

Lassen Sie uns dieses Bit b12 auf 1 zurücksetzen und es erneut testen :

```

registers
1 mB12 myREG m!
1 mb12 myREG m@ .      \ display : 1
forth

```

Wir haben jetzt die Schlüssel, um Stück für Stück auf den Inhalt eines beliebigen Registers zu reagieren. In diesem Kapitel konzentrieren wir uns insbesondere auf das

GPIO_OUT_REG-Register:

```

\ GPIO 0-31 Ausgangsregister R/W
$3FF44004 defREG: GPIO_OUT_REG

```

Das GPIO_OUT_REG-Register

Mit diesem Register können Sie die GPIO-Ports G0 bis G31 steuern. Wir werden drei farbige Dioden an die Pins G25, G26 und G27 anschließen. Wir definieren daher drei Konstanten, die mit diesen Pins verbunden sind:

```

\ definierte LEDs GPIOs
25 constant ledRED
26 constant ledYELLOW
27 constant ledGREEN

```

Im Kapitel *Eine Ampel mit ESP32 verwalten* haben wir dieselben Wörter mit **defPIN** definiert:.. Hier machen wir es mit **constant**, was das gleiche Verhalten hat. Wir werden diese Konstanten verwenden, um die Masken zu definieren:

```

\ Definieren Sie Masken für rote, gelbe und grüne LEDs
1 ledRED      defMASK: mLED_RED
1 ledYELLOW   defMASK: mLED_YELLOW
1 ledGREEN    defMASK: mLED_GREEN

```

Um die rote LED an G25 zu aktivieren, geben Sie Folgendes ein :

```
registers
1 mLED_RED GPIO_OUT_REG m!
forth
```

Um die wiederkehrende Auswahl des Registervokabulars zu vermeiden, definieren wir zwei Wörter, die uns die Programmierung erleichtern :

```
\ set Maske in Adr
: regSet ( val shift mask addr -- )
  [ registers ] m! [ forth ]
;

\ Testmaske in Adr
: regTst ( shift mask addr -- val )
  [ registers ] m@ [ forth ]
;
```

Aktivierung der roten LED am G25 :

```
1 mLED_RED GPIO_OUT_REG regSet
```

Dies funktioniert, sofern die GPIO-Pins korrekt initialisiert sind. Das werden wir besprechen.

Aktivierungs- und Deaktivierungsregister

Die Registernamen stammen aus der Dokumentation zum *ESP32 Technical Reference Manual* (PDF):

https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

Bevor wir unsere an die GPIO-Ports G25, G26 und G27 angeschlossenen LEDs ein- und ausschalten, beginnen wir mit der Initialisierung dieser Ports. Dies geschieht durch Einwirkung auf das **GPIO_ENABLE_REG**-Register:

```
: GPIO.init ( -- )
  1 mLED_RED      GPIO_ENABLE_REG regSet
  1 mLED_YELLOW   GPIO_ENABLE_REG regSet
  1 mLED_GREEN    GPIO_ENABLE_REG regSet
;
```

Durch Ausführen des Wortes **GPIO.init** werden die Ausgangsports G25, G26 und G27 initialisiert. Testen wir die Beleuchtung der LEDs :

```
GPIO.init
1 mled_red      GPIO_OUT_REG regSet
1 mled_yellow   GPIO_OUT_REG regSet
1 mled_green    GPIO_OUT_REG regSet
```

Wenn die LEDs richtig verdrahtet sind, sollten sie leuchten. Hier erfolgt das Aufleuchten der LEDs sequentiell durch dreimaliges Wiederholen des Wortes **regSet**. Indem wir direkt

auf den Inhalt des **GPIO_OUT_REG**-Registers einwirken, können wir die drei LEDs in einer einzigen Ausführung von **regSet** zum Leuchten bringen:

```
GPIO.init
7 mled_red
  mled_yellow nip +
    mled_green nip + GPIO_OUT_REG regSet
```

Sehen wir uns an, wie man die beiden Register **GPIO_OUT_W1TS_REG** und **GPIO_OUT_W1TC_REG** verwendet, um indirekt auf den Zustand eines oder mehrerer GPIO-Ports einzuwirken :

- **GPIO_OUT_W1TS_REG** : *GPIO Output Write to Set Register* wird verwendet, um die Bits, die den GPIO-Pins im „Output“-Modus entsprechen, auf logisch hoch (1) zu setzen. Sie können die angegebenen GPIO-Ausgänge aktivieren, indem Sie die entsprechenden Bits auf 1 setzen.
- **GPIO_OUT_W1TC_REG** : *GPIO Output Write to Clear Register* wird verwendet, um bestimmte Bits im GPIO-Ausgangsregister zu löschen (auf Null). Jedes Bit dieses Registers ist einem bestimmten GPIO-Pin zugeordnet, und **indem Sie eine 1** in ein bestimmtes Bit dieses Registers schreiben, können Sie den entsprechenden Ausgang des GPIO-Pins auf Null setzen (löschen).

Wir können daher alle unsere LEDs in einer einzigen **RegSet**-Sequenz wie folgt zum Leuchten bringen :

```
7 mled_red
  mled_yellow nip +
    mled_green nip + GPIO_OUT_W1TS_REG regSet
```

Und um alle LEDs in einer einzigen **RegSet**-Sequenz auszuschalten, führen wir Folgendes aus :

```
7 mled_red
  mled_yellow nip +
    mled_green nip + GPIO_OUT_W1TC_REG regSet
```

Wenn wir einen zeitgesteuerten Ein- und Ausschaltzyklus einer LED verwalten möchten, erstellen wir ein Wort, das einen Ein- und Ausschaltzyklus verwaltet:

```
\ Definieren Sie eine EIN- und AUS-Sequenz für eine LED
: GPIO.on.off.sequence { position mask delay -- }
  1 position mask GPIO_OUT_W1TS_REG regSet
  delay ms
  1 position mask GPIO_OUT_W1TC_REG regSet
;
```

Wir werden unser Wort **GPIO.on.off.sequence** :

```
mLED_RED 1000 GPIO.on.off.sequence
```

Diese Sequenz sollte die rote LED eine Sekunde lang aufleuchten lassen. Definieren wir nun einen vollständigen Zyklus, der einen Brand an einer Straßenkreuzung simuliert:

```
: traffic-light ( -- )
  mLED_GREEN 3000 GPIO.on.off.sequence
  mLED_YELLOW 1000 GPIO.on.off.sequence
  mLED_RED    3000 GPIO.on.off.sequence
;
```

Das Anfahren **traffic-light** simuliert eine klassische Straßenampel. Allerdings können wir keine Straßenampel simulieren, bei der die roten und orangefarbenen Lichter gleichzeitig leuchten. Wir werden daher zunächst das Wort **TRAFFIC.sequence** schreiben, das den Code von **GPIO.on.off.sequence** verwendet, mit dem Unterschied, dass wir zusätzlich zu den anderen Parametern auch einen Wert verwenden müssen :

```
\ define a ON and OFF sequence
: TRAFFIC.sequence { val position mask delay -- }
  val position mask GPIO_OUT_W1TS_REG regSet
  delay ms
  val position mask GPIO_OUT_W1TC_REG regSet
;
```

Dann definieren wir diese vier Wörter, die es ermöglichen, einen komplexen Straßenbrandzyklus zu bewältigen:

```
: TRAFFIC.red ( -- )
  1 mLED_RED    2500 TRAFFIC.sequence ;
: TRAFFIC.yellow ( -- )
  1 mLED_YELLOW 1000 TRAFFIC.sequence ;
: TRAFFIC.green ( -- )
  1 mLED_GREEN  3000 TRAFFIC.sequence ;
: TRAFFIC.red-yellow ( -- )
  3 mLED_RED
  mLED_YELLOW nip + 500 TRAFFIC.sequence ;
```

Wir definieren nun einen Ampelzyklus, wie wir ihn in Deutschland antreffen könnten :

```
: TRAFFIC.german.cycle ( -- )
  TRAFFIC.red
  TRAFFIC.red-yellow
  TRAFFIC.green
  TRAFFIC.yellow
;
```

In Deutschland haben Ampeln vier Zyklen. Die Besonderheit dieser Lichter besteht darin, dass sie gleichzeitig das rote und das gelbe Licht aufleuchten lassen, bevor sie wieder auf das grüne Licht umschalten.

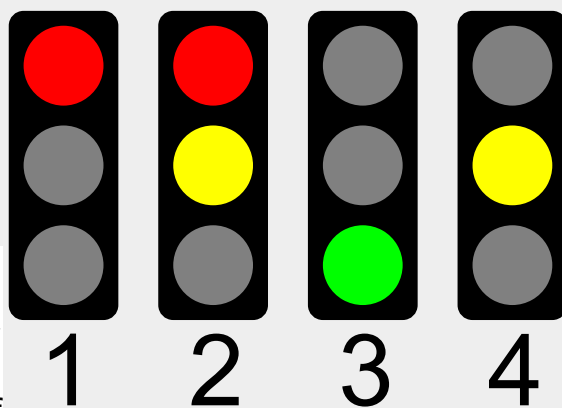


Figure 19: cycle de feux à quatre états

Und zum Abschluss testen wir unseren Straßenlaternenzyklus im Loop:

```
: TRAFFIC.loop ( -- )
  begin
    TRAFFIC.german.cycle
  key? until
;
```

Durch Ausführen von **TRAFFIC.loop** wird unsere Straßenlaterne simuliert. Wenn die Sequenz **TRAFFIC.red-yellow** eintrifft, können wir sehen, dass die gelben und roten Lichter gleichzeitig leuchten.

In diesem Kapitel haben wir gesehen, wie man auf mehrere GPIO-Ausgänge gleichzeitig einwirkt, indem man direkt auf die GPIO-Register einwirkt.

Allerdings ist es bei ESP32-Karten nicht empfehlenswert, zu viele LEDs gleichzeitig zu verwalten. Die ESP32-Karte sollte für die Signalverwaltung an sehr energieeffiziente Geräte reserviert sein. Wir werden daher auf Bestückungen wie 8 bis 16 LEDs verzichten, um beispielsweise einen Lauflichteffekt zu erzielen, es sei denn, wir verwenden Steuerelemente mit separater Stromversorgung.

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:

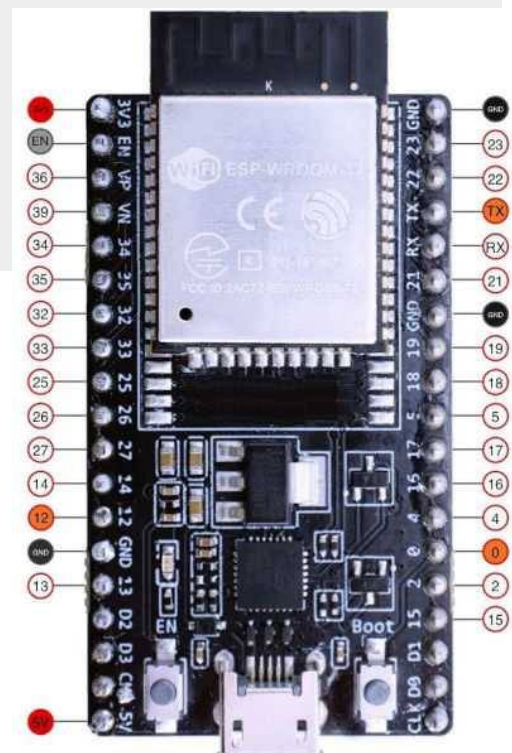
```

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

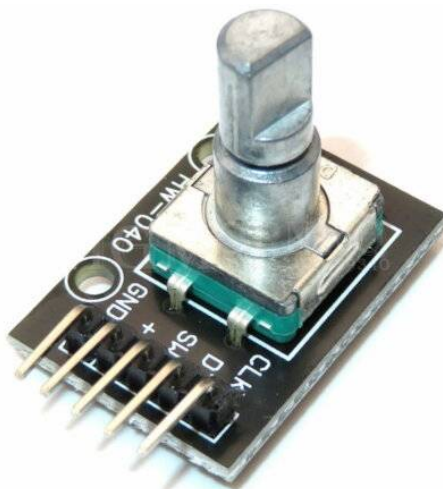
Um ein Signal zu variieren, haben wir mehrere Lösungen:

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

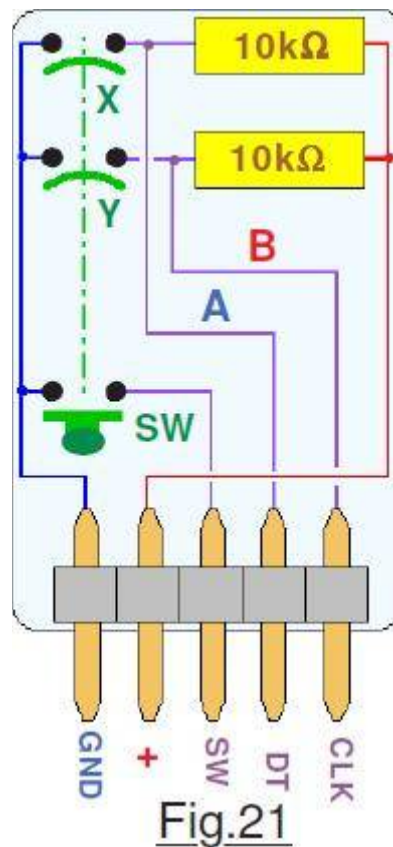
Der Drehgeber ist eine interessante Lösung. Es kann als Potentiometer verwendet werden, mit dem Vorteil, dass es keinen Start- und Endanschlag hat.

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

Hier ist unser Encoder:



Internes Funktionsdiagramm:



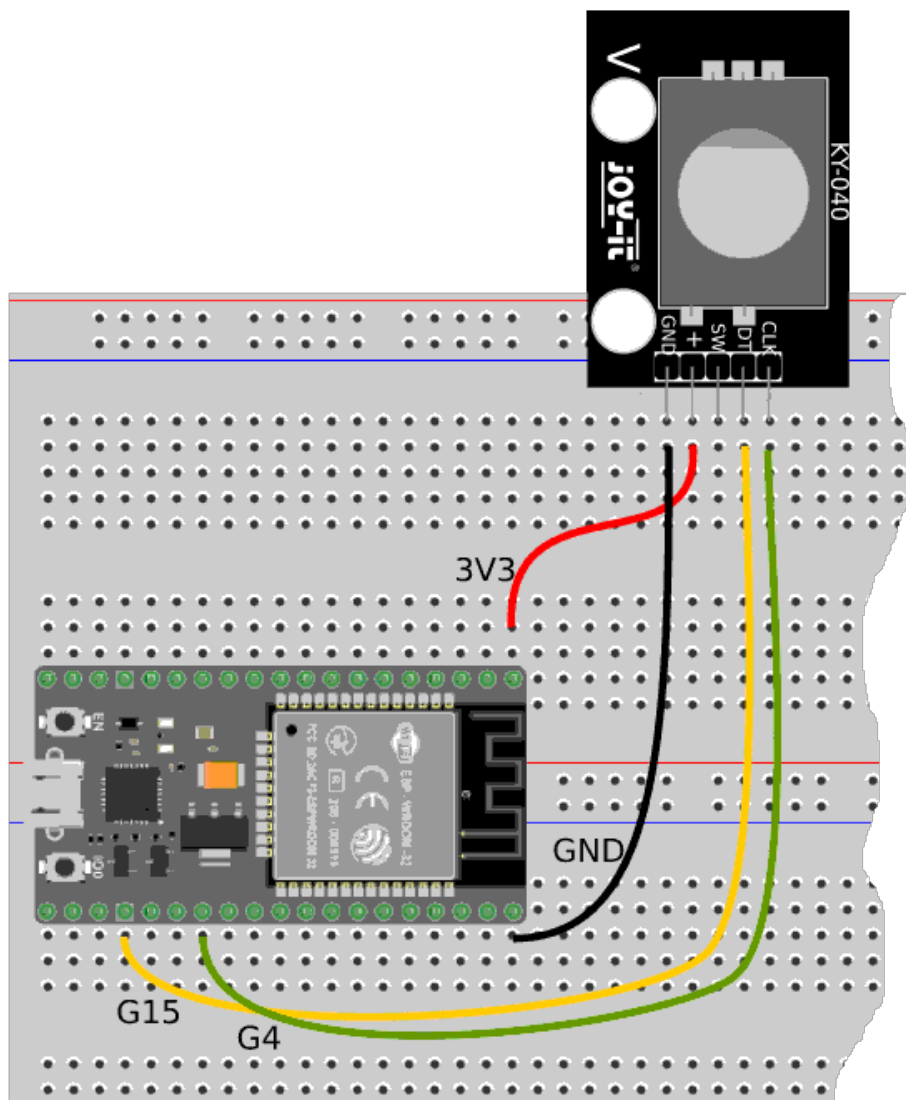
Nach diesem Diagramm interessieren uns zwei Terminals:

- 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:

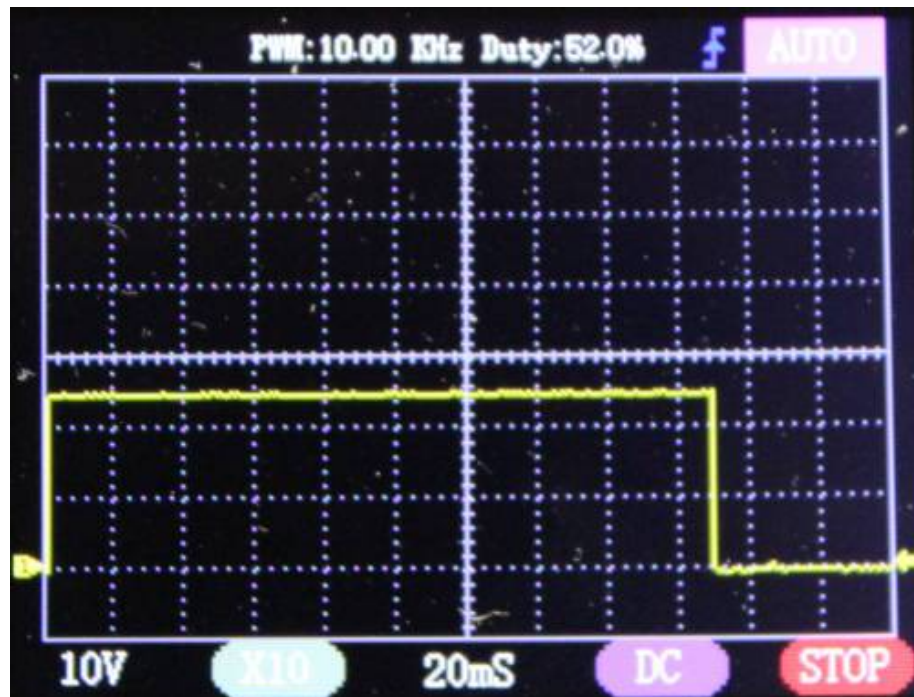


BITTE BEACHTEN : Die Position der Pins G4 und G15 kann je nach Version Ihrer ESP32-Karte variieren.

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 Wartezeigers 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** 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.

Programmieren Sie einen Sonnenscheinanalysator

Präambel

Im Rahmen eines Solarprojekts, bei dem mehrere Solarmodule und deren Mikrowechselrichter zum Einsatz kommen, treten einige Probleme bei der Verwaltung der erzeugten elektrischen Energie auf.

Das Hauptanliegen besteht darin, große Verbrauchergeräte nur dann zu aktivieren, wenn die Solarmodule bei voller Sonne produzieren. Besonders betroffen ist ein Gerät, der Heißwasser-Cumulus:

- Aktivieren Sie das Gerät, wenn die Paneele direktem Sonnenlicht ausgesetzt sind.
- Deaktivieren Sie das Gerät, wenn Wolken vorziehen.

Mikrowechselrichter speisen Strom in das allgemeine Stromnetz ein. Wenn ein Gerät, das viel Strom verbraucht, bei Wolkendurchzug aktiv ist, wird dieses Gerät hauptsächlich über das allgemeine Netz mit Strom versorgt.

In diesem Artikel stellen wir eine Lösung vor, die die Wolkenerkennung mithilfe eines Miniatur-Solarpanels und einer ESP32-Karte ermöglicht.

Den vollständigen Code finden Sie hier:

<https://github.com/MPETREMAN11/ESP32forth/blob/main/ADC/solarLightAnalyzer.txt>

Das Miniatur-Solarpanel

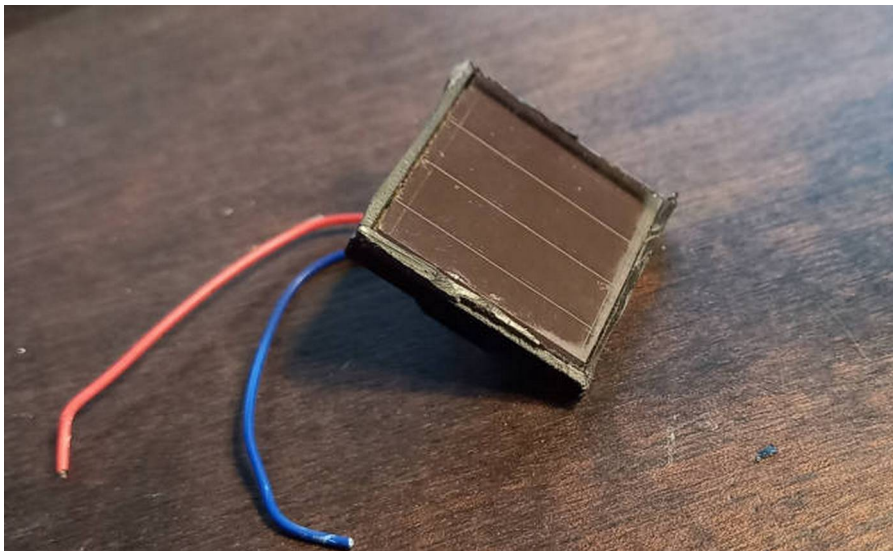
Zur Herstellung unseres Wolkendetektors verwenden wir ein sehr kleines Solarpanel, hier ein 25 mm x 25 mm großes Panel.

Wiederherstellung eines Miniatur-Solarpanels

Dieses Miniatur-Solarpanel stammt aus einer defekten Gartenlampe:



Hier ist unser Mini-Solarpanel aus dieser Gartenlampe:



Wir opfern zwei Dupont-Anschlüsse, um verschiedene Messungen an unserer Prototypenplatte durchführen zu können. Diese Anschlüsse werden an die beiden roten und blauen Drähte angelötet, die aus dem Mini-Solarpanel kommen.

Messung der Solarpanelspannung

Wir beginnen mit der Messung der Leerlaufspannung unseres Mini-Solarpanels, hier mit einem Oszilloskop. Diese Spannungsmessung kann auch mit einem Voltmeter durchgeführt werden:



Bei hellem Licht beträgt die gemessene Spannung 14,2 Volt!

Bei diffusem Licht sinkt die Spannung auf 5,8 Volt.

Durch das Abdecken des Mini-Solarpanels mit der Hand sinkt die Spannung auf nahezu 0 Volt.

Messung des Solarpanelstroms

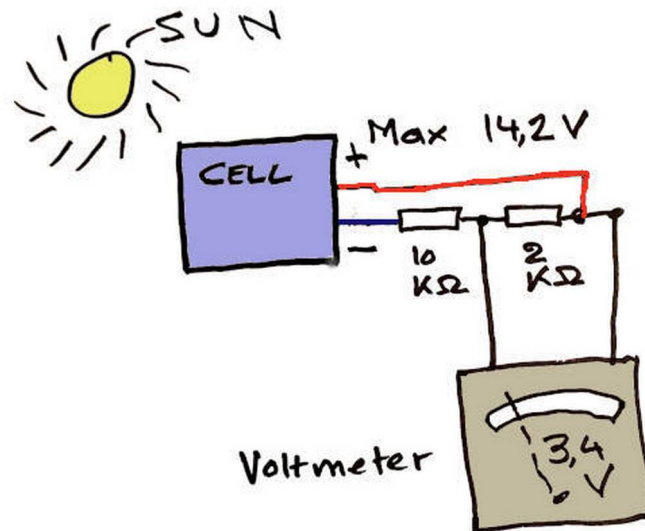
Der Strom, also die Intensität, muss mit einem Amperemeter gemessen werden. Geeignet ist die Amperemeterfunktion eines Universalreglers. Durch Kurzschließen des Mini-Solarpanels bei hellem Licht kann ein Strom von 10 mA gemessen werden.

Unser Mini-Solarpanel hat daher eine Leistung von ca. 0,2 Watt.

Bevor Sie unser Mini-Solarpanel an die ESP32-Karte anschließen, müssen Sie unbedingt die Ausgangsspannung senken. Es kommt nicht in Frage, diese Spannung von 14,2 Volt in einen Eingang der ESP32-Karte einzuspeisen. Eine solche Spannung würde die internen Schaltkreise der ESP32-Platine zerstören.

Senkung der Solarpanelspannung

Die Idee besteht darin, die Spannung an unserem Mini-Solarpanel zu senken. Nach einigen Tests wählen wir zwei Widerstände aus, einen mit 220 Ohm, den anderen mit 1K Ohm. Montage dieser Widerstände:



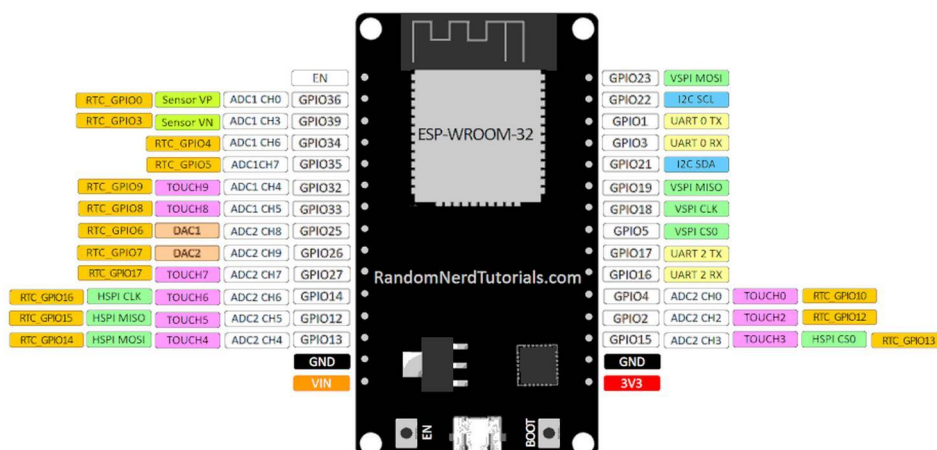
Die Spannungsmessung erfolgt zwischen den beiden Widerständen und dem Pluspol des Solarpanels.

Das Voltmeter zeigt nun bei hellem Licht eine maximale Spannung von 3,2V an, bei diffusem Licht eine Spannung von 0,35V.

Programmierung des Solaranalysators

Die ESP32-Karte verfügt über 18 12-Bit-Kanäle, die eine Analog-Digital-Umwandlung (ADC) ermöglichen. Um die Spannung unseres Mini-Solarpanels zu analysieren, ist ein einzelner ADC-Kanal notwendig und ausreichend.

nur 15 ADC-Kanäle zur Verfügung:



Wir werden einen verwenden, den **ADC1_CH6- Kanal, der an Pin G34** angeschlossen ist :

```

34 constant SOLAR_CELL

: init-solar-cell ( -- )
    SOLAR_CELL input pinMode
    ;

init-solar-cell

```

Um die Spannung am Punkt zwischen den beiden Widerständen abzulesen, führen Sie einfach **SOLAR_CELL analogRead aus** . Diese Sequenz lässt einen Wert zwischen 0 und 4095 fallen. Der niedrigste Wert entspricht einer Nullspannung. Der höchste Wert entspricht einer Spannung von 3,3 Volt.

solar-cell-read definition zur Wiederherstellung dieser Spannung:

```

: solar-cell-read ( -- n )
    SOLAR_CELL analogRead
    ;

```

Testen wir diese Definition in einer Schleife:

```

: solar-cell-loop ( --)
    init-solar-cell
    begin
        solar-cell-read cr .
        200 ms
    key? until
    ;

```

Beim Ausführen von **solar-cell-loop** wird alle 200 Millisekunden der ADC-Spannungsumwandlungswert angezeigt:

```

...
322
331
290
172
39
0
0
0
0
19
79
86
...

```

Hier wurden die Werte durch Beleuchten des Mini-Solarpanels mit einer Hochleistungslampe ermittelt. Nullwerte entsprechen dem Fehlen von Beleuchtung.

Tests mit der echten Sonne zeigen Messungen über 300.

Verwalten der Aktivierung und Deaktivierung eines Geräts

Zunächst definieren wir zwei Pins, von denen ein Pin für die Verwaltung eines Aktivierungssignals und der andere für ein Deaktivierungssignal reserviert ist:

- G17-Pin mit einer grünen LED verbunden. Mit diesem Pin wird ein Gerät aktiviert.
- G16-Pin mit einer roten LED verbunden. Mit dieser PIN wird ein Gerät deaktiviert.

```
17 constant DEVICE_ON      \ green LED
16 constant DEVICE_OFF     \  red LED

: init-device-state ( -- )
  DEVICE_ON  output pinMode
  DEVICE_OFF output pinMode
;
```

Wir hätten einen einzigen Pin verwenden können, um das Remote-Gerät zu verwalten. Aber einige Geräte, wie zum Beispiel bistabile Relais, haben zwei Spulen:

- Die erste Spule wird mit Strom versorgt, so dass die Kontakte schalten. Der Zustand ändert sich nicht, wenn die Spule nicht mehr bestromt wird;
- Um in den Ausgangszustand zurückzukehren, wird die zweite Spule mit Strom versorgt.

Aus diesem Grund wird unsere Programmierung diesen Gerätetyp berücksichtigen.

```
\ Definieren Sie die High-State-Verzögerung des Triggers
500 value DEVICE_DELAY

\ setze HIGH-Level des Triggers
: device-activation { trigger -- }
  trigger HIGH digitalWrite
  DEVICE_DELAY ?dup
  if
    ms
    trigger LOW  digitalWrite
  then
;
```

Hier wird die Pseudokonstante **DEVICE_DELAY** verwendet, um die Verzögerung anzugeben, während der das Steuersignal hoch gehalten werden soll. Nach dieser Zeit kehrt das Steuersignal in den Low-Zustand zurück.

Wenn der Wert von **DEVICE_DELAY** Null ist, bleibt das Steuersignal hoch.

Es ist das Wort **trigger-activation**, das die Aktivierung des entsprechenden Pins verwaltet:

- **TRIGGER_ON trigger-activation** setzt den an die grüne LED angeschlossenen Pin dauerhaft oder vorübergehend auf High;
- **TRIGGER_OFF trigger-activation** setzt den an die rote LED angeschlossenen Pin dauerhaft oder vorübergehend auf High.

Wir definieren nun zwei Wörter, **device-ON** und **device-OFF**, die jeweils für die Aktivierung und Deaktivierung des Geräts verantwortlich sind, das über die Pins G16 und G17 gesteuert werden soll:

```
\ Gerätestatus definieren: 0=LOW, -1=HIGH
0 value DEVICE_STATE

: enable-device ( -- )
  DEVICE_STATE invert
  if
    DEVICE_OFF LOW digitalWrite
    DEVICE_ON device-activation
    -1 to DEVICE_STATE
  then
;

: disable-device ( -- )
  DEVICE_STATE
  if
    DEVICE_ON LOW digitalWrite
    DEVICE_OFF device-activation
    0 to DEVICE_STATE
  then
;
```

Der Gerätestatus wird in **DEVICE_STATE** gespeichert. Dieser Status wird getestet, bevor versucht wird, den Status zu ändern. Wenn das Gerät aktiv ist, wird es nicht wiederholt reaktiviert. Das Gleiche gilt, wenn das Gerät inaktiv ist.

```
\ Triggerwert für sonnigen oder bewölkten Himmel definieren
300 value SOLAR_TRIGGER

\ wenn Solarlicht > SOLAR_TRIGGER, Aktion aktivieren
: action-light-level ( -- )
  solar-cell-read SOLAR_TRIGGER >=
  if
    enable-device
  else
    disable-device
  then
;
```

Ausgelöst durch Timer-Interrupt

Der eleganteste Weg ist die Verwendung eines Timer-Interrupts. Wir verwenden Timer 0:

```
0 to DEVICE_DELAY
```

```

200 to SOLAR_TRIGGER
init-solar-cell
init-device-state

timers
: action ( -- )
    action-light-level
    0 rerun
;

' action 1000000 0 interval

```

Von nun an analysiert der Timer jede Sekunde den Lichtfluss und reagiert entsprechend.
 Link zum Video: <https://youtu.be/IAjeev2u9fc>

Bei diesem Video berücksichtigen wir zwei Parameter:

- **0 to DEVICE_DELAY** leuchten die LEDs dauerhaft. Die rote LED zeigt an, dass das Gerät deaktiviert ist. Die grüne LED zeigt die Geräteaktivierung an;
- **200 to SOLAR_TRIGGER** bestimmt den Schwellenwert zum Auslösen des Sonnenscheinzustands. Dieser Parameter ist einstellbar, um ihn an die Eigenschaften des Mini-Solarmoduls anzupassen.

Das **action**-wort funktioniert per Timer-Interrupt. Für den Betrieb des Detektors ist daher keine allgemeine Schleife erforderlich.

Vom Sonnenscheinsensor gesteuerte Geräte

Zusammenfassend haben wir zwei Steuerleitungen, eine Leitung entspricht der grünen LED im Video, die andere Leitung entspricht der roten LED. Das Programm ist so konzipiert, dass nicht beide Steuerleitungen gleichzeitig aktiv sein können.

Um ein kontinuierliches Signal auf einer der Steuerleitungen zu haben, muss der **DEVICE_DELAY**- Wert nur Null sein. So initialisieren Sie dieses Szenario:

```

\ Beginnen Sie mit einem konstanten Befehlssignal
: start-CCS ( -- )
    0 to DEVICE_DELAY
    200 to SOLAR_TRIGGER
    init-solar-cell
    init-device-state
    disable-device
    [ timers ] ['] action 1000000 0 interval
;

```

Und um zeitgesteuerte Befehle zu erhalten, weisen wir **DEVICE_DELAY** die Verzögerung des Aktivierungs- oder Deaktivierungsbefehls des Geräts zu.

```

\ Beginnen Sie mit einem temporären Befehlssignal

```

```
: start-TCS ( -- )
: start-TCS ( -- )
  300 to DEVICE_DELAY
  200 to SOLAR_TRIGGER
  init-solar-cell
  init-device-state
  disable-device
  [ timers ] ['] action 1000000 0 interval
;
```

start-TCS- Szenario ist typisch für eine impulsbetriebene bistabile Relaissteuerung. Das Relais wird aktiviert, wenn es einen Aktivierungsbefehl erhält. Auch wenn das Aktivierungssignal abfällt, bleibt das bistabile Relais aktiv. Um das bistabile Relais zu deaktivieren, muss ihm auf der Deaktivierungsleitung ein Deaktivierungsbefehl übermittelt werden.

Zusammenfassend lässt sich sagen, dass unser Solarlichtanalysator eine Vielzahl von Geräten steuern kann. Es reicht aus, die Steuerschnittstellen dieser Geräte an die Eigenschaften der GPIO-Ports der ESP32-Karte anzupassen.

Installieren der OLED-Bibliothek für SSD1306

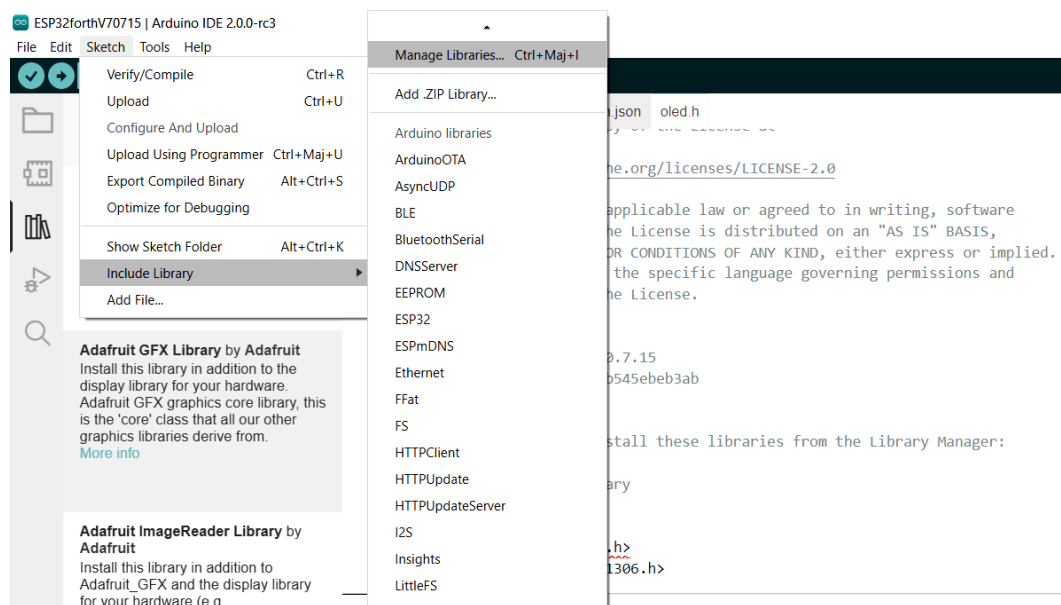
Seit ESP32forth Version 7.0.7.15 sind die Optionen im **optional** Ordner verfügbar:

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
```

Die I2C-Schnittstelle auf ESP32

Einführung

I2C (bedeutet auf Englisch: Inter-Integrated Circuit) ist ein Computerbus, der aus dem „Krieg der Standards“ hervorgegangen ist, den die Akteure der elektronischen Welt begonnen haben. Es wurde von Philips für Heimautomatisierungs- und Haushaltselektronikanwendungen entwickelt und ermöglicht den einfachen Anschluss eines Mikroprozessors und verschiedener Schaltkreise, insbesondere eines modernen Fernsehers: Fernbedienungsempfänger, Einstellungen des Niederfrequenzverstärkers, Tuner, Uhr, Verwaltung der Scart-Buchse usw. .

Es gibt unzählige Peripheriegeräte, die diesen Bus nutzen, er kann sogar per Software in jeden Mikrocontroller implementiert werden. Das Gewicht der Unterhaltungselektronikindustrie hat dank dieser zahlreichen Komponenten sehr niedrige Preise ermöglicht.

Dieser Bus wird von bestimmten Herstellern manchmal als TWI (Two Wire Interface) oder TWSI (Two Wire Serial Interface) bezeichnet.

Der Austausch findet immer zwischen einem einzelnen Master und einem (oder allen) Slave(s) statt, immer auf Initiative des Masters (niemals von Master zu Master oder von Slave zu Slave). Allerdings hindert nichts eine Komponente daran, vom Master- in den Slave-Status und umgekehrt zu wechseln.

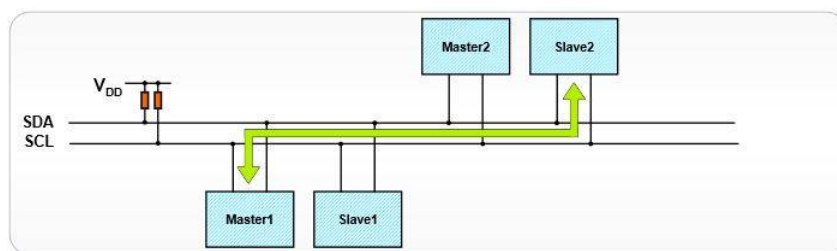


Figure 20: Prinzip eines I2C-Busses

Die Verbindung erfolgt über zwei Leitungen:

- SDA (Serial Data Line): bidirektionale Datenleitung,
- SCL (Serial Clock Line): bidirektionale Synchronisationstaktleitung.

Wir dürfen die Masse nicht vergessen, die den Geräten gemeinsam sein muss.

Beide Leitungen werden über Pull-up-Widerstände (RP) auf den Spannungspegel VDD gezogen.

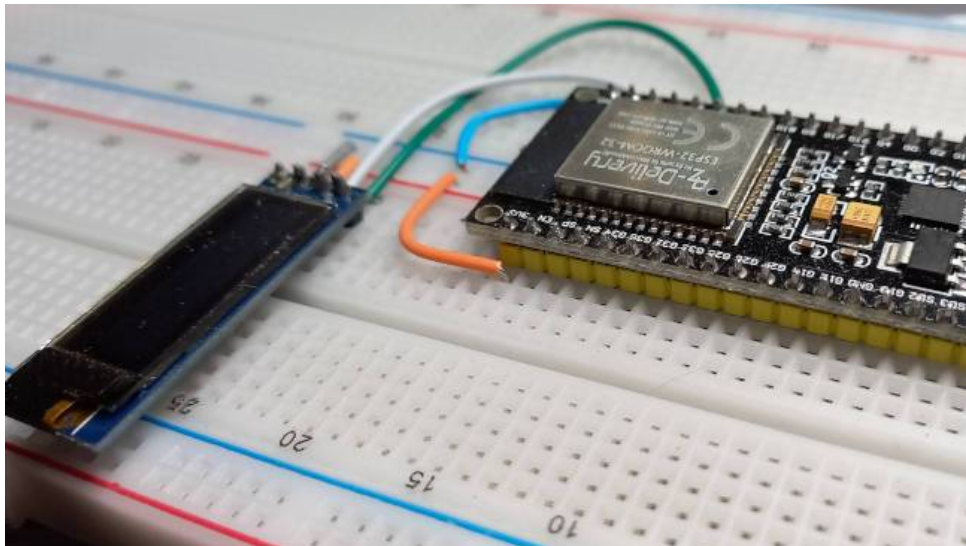


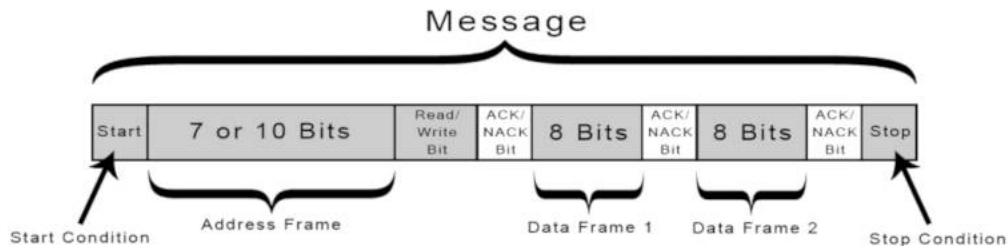
Figure 21: An den I2C-Bus angeschlossenes OLED-Display

Master-Slave-Austausch

Die Nachricht kann in zwei Teile unterteilt werden:

- Der Master ist der Sender, der Slave ist der Empfänger:
 - Aussenden einer START-Bedingung durch den Master („S“),
 - Übertragung des Adressbytes oder der Adressbytes durch den Master zur Bezeichnung eines Slaves, wobei das R/W-Bit auf 0 steht (siehe Abschnitt zur Adressierung weiter unten),
 - Antwort vom Slave mit einem ACK-Bestätigungsbit (oder NACK-Nichtbestätigungsbit),
 - Nach jeder Quittierung kann der Slave eine Pause („PA“) anfordern.
 - Senden eines Befehlsbytes durch den Master für den Slave,
 - Antwort vom Slave mit einem ACK-Bestätigungsbit (oder NACK-Nichtbestätigungsbit),
 - Aussenden einer RESTART-Bedingung durch den Master („RS“),
 - Übertragung des Adressbytes oder der Adressbytes durch den Master zur Bezeichnung desselben Slaves, wobei das R/W-Bit auf 1 steht.
 - Antwort vom Slave mit einem ACK-Bestätigungsbit (oder NACK-Nichtbestätigungsbit).
- Der Master wird zum Empfänger, der Slave wird zum Sender:
 - Senden eines Datenbytes durch den Slave für den Master,

- Antwort vom Master mit einem ACK-Bestätigungsbit (oder NACK-Nichtbestätigungsbit),
- Übertragung weiterer Datenbytes durch den Slave mit Quittung vom Master,
- für das letzte vom Master erwartete Datenbyte antwortet er mit einem NACK, um den Dialog zu beenden,
- Ausgabe einer STOP-Bedingung durch den Master („P“).



Startbedingung : Die SDA-Leitung wechselt von einem hohen Spannungspegel auf einen niedrigen Spannungspegel, bevor die SCL-Leitung von hoch auf niedrig wechselt.

Abschaltbedingung : Die SDA-Leitung wechselt von einem niedrigen Spannungspegel auf einen hohen Spannungspegel, nachdem die SCL-Leitung von niedrig auf hoch wechselt.

Adressrahmen : eine eindeutige 7- oder 10-Bit-Sequenz für jeden Slave, die den Slave identifiziert, wenn der Master mit ihm kommunizieren möchte.

Lesen-/Schreibbit : Ein einzelnes Bit, das angibt, ob der Master Daten an den Slave sendet (niedriger Spannungspegel) oder Daten von ihm anfordert (hoher Spannungspegel).

ACK/NACK-Bit : Auf jeden Frame einer Nachricht folgt ein Bestätigungs-/Nichtbestätigungsbit. Wenn ein Adressrahmen oder Datenrahmen erfolgreich empfangen wurde, wird ein ACK-Bit an den Absender zurückgegeben.

Adressierung

I2C hat keine Slave-Auswahlleitungen wie SPI, daher braucht es eine andere Möglichkeit, dem Slave mitzuteilen, dass Daten an ihn und nicht an einen anderen Slave gesendet werden. Dies geschieht durch Adressierung. Der Adressrahmen ist immer der erste Rahmen nach dem Startbit in einer neuen Nachricht.

Der Master sendet jedem mit ihm verbundenen Slave die Adresse des Slaves, mit dem er kommunizieren möchte. Anschließend vergleicht jeder Slave die vom Master gesendete Adresse mit seiner eigenen. Wenn die Adresse übereinstimmt, wird ein ACK-Bit mit

niedriger Spannung an den Master zurückgegeben. Wenn die Adresse nicht übereinstimmt, unternimmt der Slave nichts und die SDA-Leitung bleibt hoch.

Auf diese Weise erkennt das **wire.detect**- Wort Geräte, die an den i2c-Bus angeschlossen sind.

Sie können mehrere verschiedene Geräte an den i2c-Bus anschließen. Sie können nicht mehrere Kopien desselben Geräts an denselben i2c-Bus anschließen.

GPIO-Ports für I2C einstellen

Das Einrichten der GPIO-Ports für den I2C-Bus ist sehr einfach:

```
\ Aktivieren Sie das Wire-Vokabular
Wire
\ Starten Sie die I2C-Schnittstelle über Pin 21 und 22 am ESP32 DEVKIT V1
\ wobei 21 als SDA und 22 als SCL verwendet werden.
21 22 wire.begin
```

I2C-Busprotokolle

Der Dialog findet nur zwischen einem Meister und einem Sklaven statt. Dieser Dialog wird immer vom Master initiiert (Startbedingung): Der Master sendet die Adresse des Slaves, mit dem er auf dem I2C-Bus kommunizieren möchte.

Der Dialog wird immer vom Master beendet (Stop-Bedingung).

Das Taktsignal (SCL) wird vom Master generiert.

Erkennen eines I2C-Geräts

Dieser Teil wird verwendet, um das Vorhandensein eines an den I2C-Bus angeschlossenen Geräts zu erkennen.

Sie können diesen Code kompilieren, um zu testen, ob angeschlossene und aktive Module am I2C-Bus vorhanden sind.

```
\ aktiviert das Wire-Vokabular
Wire
\ startet die I2C-Schnittstelle über Pin 21 und 22 auf ESP32 DEVKIT V1
\ mit 21 für sda und 22 für scl.
21 22 wire.begin

: spaces ( n -- )
  for
    space
  next
;


```

```

: .## ( n -- )
  <# # # #> type
;

\ Nicht alle Bitmuster sind gültige 7-Bit-i2c-Adressen
: Wire.7bitaddr? ( a -- f )
  dup $07 >=
  swap $77 <= and
;

: Wire.detect ( -- )
  base @ >r hex
  cr
  ."      00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f"
  $80 $00 do
    i $0f and 0=
    if
      cr i .## ." : "
    then
    i Wire.7bitaddr? if
      i Wire.beginTransaction
      -1 Wire.endTransmission 0 =
      if
        i .## space
      else
        ." -- "
      then
    else
      2 spaces
    then
  loop
  cr r> base !
;

```

Hier zeigt die Ausführung des Worts **Wire.detect** an, dass das OLED-Anzeigegerät an der Hexadezimaladresse 3c vorhanden ist:

```

Wire.detect
      00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00 :      -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
20 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
30 : -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
50 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
60 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
70 : -- -- -- -- -- -- -- -- -- -- -- -- -- --

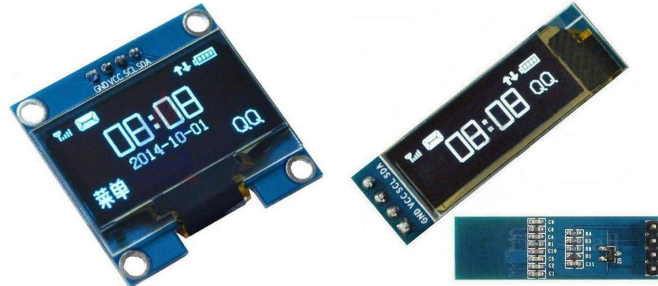
```

Hier haben wir ein Modul an der Hexadezimaladresse 3c erkannt. Dies ist die Adresse, die wir zur Adressierung dieses Moduls verwenden werden...

Das SSD1306 OLED-Display

Das OLED-Display gibt es in zwei Definitionen:

- 128 x 64 Pixel, Monochrom- oder Farbbildschirm. Wenn der Bildschirm farbig ist, bleiben die Pixel monochrom.
- 128 x 32 Pixel, monochromer Bildschirm.



Diese Displays sind mit SPI- oder I2C-Schnittstelle verfügbar.

Bevorzugen Sie die I2C-Schnittstelle, die den Anschluss mehrerer I2C-Schnittstellen an demselben I2C-Gerät ermöglicht. Das **OLED**-Vokabular ist darauf ausgelegt, die Übertragung über I2C an diese OLED-Displays zu verwalten.

Auswahl einer Anzeigeschnittstelle

Die Wahl einer Anzeigeschnittstelle unterliegt mehreren Bedingungen:

- sein Preis;
- sein Stromverbrauch;
- seine Robustheit;
- seine einfache Programmierung und Verwendung.

Eine Anzeigeschnittstelle ist bei einem eigenständigen Setup sehr nützlich, um sehr klare Text- oder Grafikinformatoren bereitzustellen.

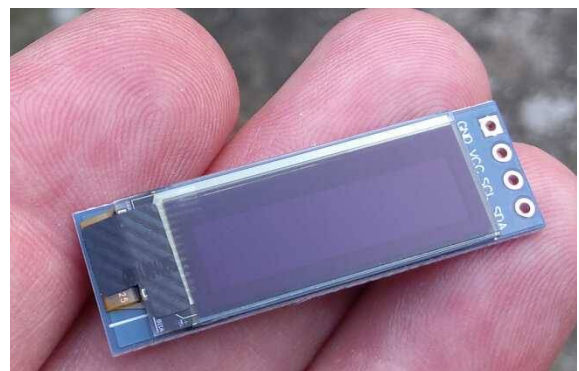
Nach mehreren Recherchen fiel die Wahl auf ein OLED-Display dieser Art

Es kostet nur ein paar Euro.

Dieses Display nutzt OLED-Technologie, also ohne Hintergrundbeleuchtung.

Die Displayauflösung beträgt 128x32 Pixel. Es kann Text und Bilder anzeigen, jedoch nur in Schwarzweiß.

Im **DISPLAYOFF**-Modus ist der Stromverbrauch nahezu Null.



Es handelt sich um ein sehr weit verbreitetes und recht gut dokumentiertes Produkt.

Online-Dokumentation

- Adafruit: Technische Dokumentation und Steuerung des OLED-Displays
<https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>
- Adafruit: SSD1306 C-Bibliothek
https://adafruit.github.io/Adafruit_SSD1306/html/files.html
- Adafruit: SSD1306 microPython
<https://github.com/adafruit/micropython-adafruit-ssd1306>
- Punyforth: SSD1306 SPI Forth
<https://github.com/zeroflag/punyforth/blob/master/arch/esp8266/forth/ssd1306-spi.forth>
- TG9541: Forth Oled-Display
<https://github.com/TG9541/forth-oled-display/blob/master/ssd1306.fs>
- Yunfan: SSD1306 128x32 i2c von
<https://gist.github.com/yunfan/2d3ee14697f3ebd3cb43ae411216d9aa>

Anschließen des SSD1306 OLED-Displays

Das SSD1306 128x32 OLED-Display muss am I2C-Bus der ESP32-Karte verwendet werden.

Dieser I2C-Bus ist auf allen ESP32-Boards vorhanden.

Verbindung zu einer ESP32-Karte:

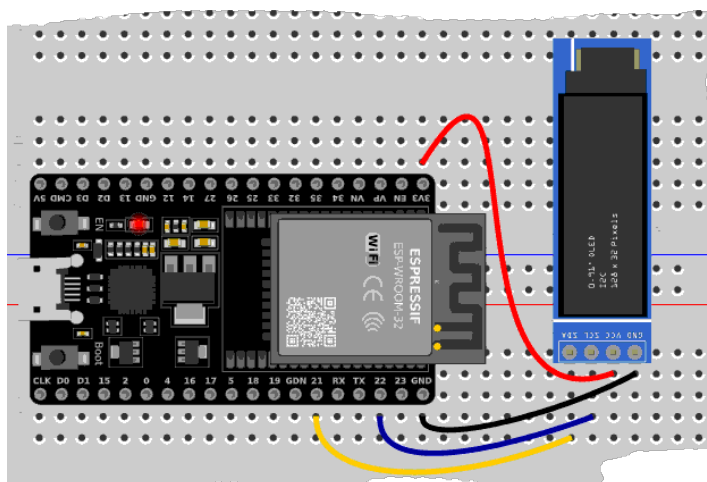


Figure 22: branchement de l'afficheur OLED SSD1306

Wie wir sehen, reichen 4 Drähte aus: 2 für die Stromversorgung des SSD1306 OLED-Displays (schwarze und rote Drähte), 2 für die Verbindung zum I2C-Bus (blaue und gelbe Drähte).

Die Stromversorgung des Displays erfolgt über die ESP32-Karte. Es ist nicht erforderlich, eine Hilfsstromversorgung zu verwenden. Der sehr geringe Stromverbrauch dieses Displays ermöglicht dies. Das OLED-Display SSD1306 verfügt über einen integrierten Schaltkreis, der die für seinen Betrieb erforderliche 5-V-Spannung zurückbringt.

Gedächtnisorganisation

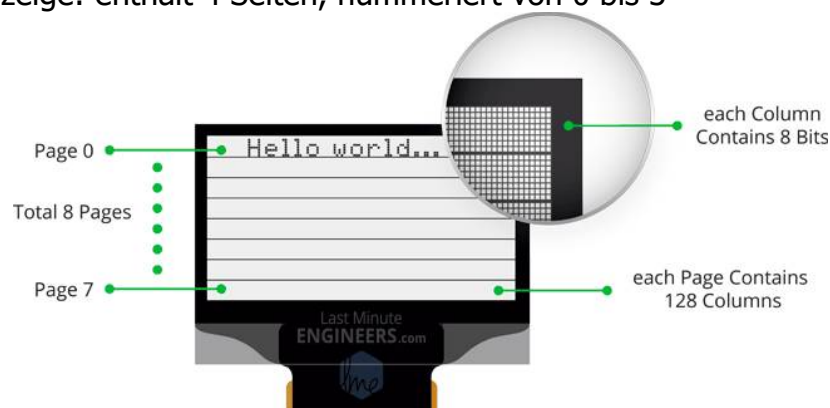
Der Bildschirm SSD1306 128x32 verwendet die gleichen internen Komponenten wie der Bildschirm SSD1306 128x64. Der interne Speicher ist beiden Modellen gemeinsam, der 128x32-Bildschirm nutzt nur einen Teil dieses Speichers.

Der interne Speicher des Displays verfügt über 1 KB RAM.

In diesem Diagramm ist hier die Organisation des Bildschirms für eine Auflösung von 128 x 64 Pixeln dargestellt:

Jede Spalte enthält 8 Bits. Pro Seite ist eine Zeile vorgesehen:

- Die 128x64-Anzeige: enthält 8 Seiten, nummeriert von 0 bis 7
- Die 128x32-Anzeige: enthält 4 Seiten, nummeriert von 0 bis 3



Jede Seite ist in Segmente unterteilt:

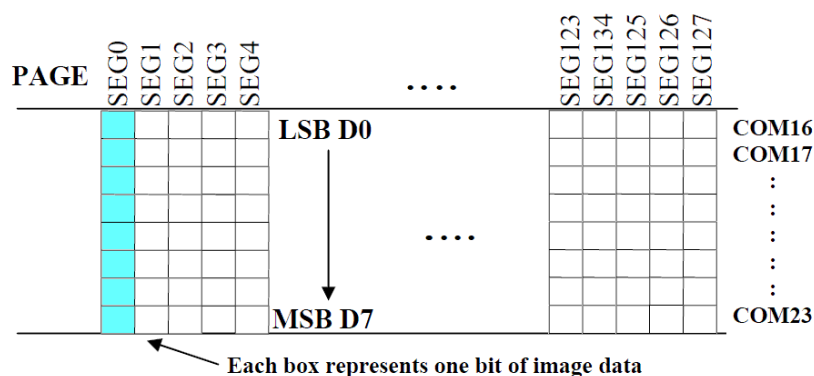


Figure 23: segmentation de l'espace mémoire de l'afficheur

Hier, in der Abbildung blau dargestellt, stellt ein Segment ein Byte dar. Das niederwertigste Bit steht oben.

Wir müssen nicht weiter gehen, um dieses Display mit dem **OLED**- Vokabular zu verwalten .

Organisieren Sie das SSD1306-Projekt

Bevor wir zum Kern der Sache kommen, schauen wir uns an, wie wir unser Projekt organisieren werden. Erstellen Sie auf Ihrem Computer einen Arbeitsordner mit dem Namen **display** . Erstellen Sie in diesem Ordner einen Unterordner **SSD1306** .

Wir werden die Verwaltung von SPIFFS-Dateien und die Erstellung unseres FORTH-Codes in einem echten Projekt voll ausschöpfen.

Erstellen der Datei main.fs

Unterordner **SSD1306** die Datei **main.fs** und kopieren Sie diesen FORTH-Code hinein:

```
RECORDFILE /spiffs/main.fs
DEFINED? --oledTest [if] forget --oledTest [then]
create --oledTest
s" /spiffs/config.fs"      included
s" /spiffs/oledTools.fs"   included
<EOF>
```

Kopieren Sie diesen Code erneut und starten Sie das Terminal, das mit dem ESP32 und der ESP32forth-Karte kommuniziert. Kopieren Sie diesen Code in das Terminal. Starte es. Am Ende der Ausführung sollten Sie Ihre main.fs-Datei auf der ESP32-Karte finden:

```
--> ls /spiffs/
autoexec.fs
main.fs
```

So überprüfen Sie, ob der Inhalt von **main.fs** im SPIFFS-Dateisystem gespeichert wurde:

```
cat /spiffs/main.fs
```

sollte den Inhalt der Datei **/spiffs/main.fs** anzeigen .

Erstellen der Datei config.fs

Unterordner **SSD1306** die Datei **config.fs** und kopieren Sie diesen FORTH-Code hinein:

```
RECORDFILE /spiffs/config.fs
\ Oled SSD1306-Abmessungen festlegen
alt
oled
128 to WIDTH
32 to HEIGHT
forth
\set Adresse der OLED SSD1306 Anzeige 128x32 Pixel
$3c constant I2C_SSD1306_ADDRESS
<EOF>
```

wie bei **main.fs** über das Terminal, um die neue Datei **config.fs** zu speichern.

Erstellen der Datei oledTools.fs

Hier ist vorerst unsere letzte **oledTools.fs**- Datei, die Sie auf dem Computer erstellen und an die ESP32-Karte übertragen können :

```
RECORDFILE /spiffs/oledTools.fs
oled
: Oled128x32Init
  OledAddr @ 0=
  if
    WIDTH HEIGHT OledReset OledNew
    SSD1306_SWITCHCAPVCC I2C_SSD1306_ADDRESS OledBegin drop
  then
  OledCLS
  1 OledTextsize      \ Zeichne 2x skalierten Text
  WHITE OledTextc      \ Zeichne weißen Text
  0 0 OledSetCursor    \ Beginnen Sie in der oberen linken Ecke
  z" *Esp32forth*" OledPrintln OledDisplay
;
forth
<EOF>
```

Testen Sie unser SSD1306-Projekt

Hier ist die Struktur unseres Projekts auf unserer Computerfestplatte.

Alle FS-Erweiterungsdateien wurden an die ESP32-Karte übergeben, um im SPIFFS-Dateisystem gespeichert zu werden.

Um den Inhalt der Datei /SPIFFS/config.fs zu kompilieren, können Sie dies über das Terminalfenster testen, das mit ESP32forth kommuniziert :

```
include /spiffs/config.fs
```

Wenn Sie den Inhalt der Datei **config.fs** niemals ändern , steht er ESP32forth immer zur Verfügung, sobald das ESP32-Board eingeschaltet wird.

main.fs- Datei übergeben haben ? Der Inhalt dieser Datei muss für den Aufruf der verschiedenen Projektdateien reserviert werden. Erinnerung an den Inhalt unserer main.fs-Datei, wie er auf der ESP32-Karte im SPIFFS-Dateisystem aufgezeichnet ist:

```
EFINED? --oledTest [if] forget --oledTest [then]
```

```
create --oledTest
```

```
s" /spiffs/config.fs"      included
```

```
s" /spiffs/oledTools.fs"   included
```

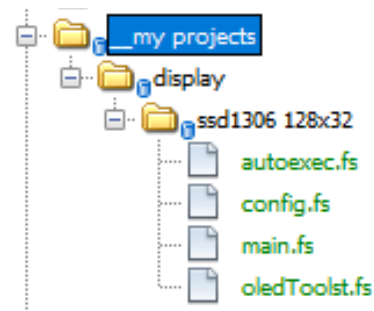


Figure 24: fichiers du projet

In den ersten beiden Zeilen können Sie einen Marker verwalten. Jedes Mal, wenn der Inhalt von **main.fs** interpretiert wird, testet ESP32forth, ob ein **--oledTest**-Wort vorhanden ist. Wenn dieses Wort existiert, wird es aus dem Wörterbuch gelöscht. Alle nach **--oledTest** kompilierten Wörter werden aus dem Wörterbuch entfernt.

In der zweiten Zeile erstellen wir das Wort **--oledTest** neu. Es ist nicht überraschend, dieses Wort zu entfernen und neu zu erstellen. Auf diese Weise wird jedes Mal, wenn der Inhalt von **main.fs** interpretiert wird, das Wörterbuch von ESP32forth mit Inhalten neu gestartet, die unser Projekt nicht stören.

Schließlich wird ESP32forth in den letzten beiden Zeilen von **main.fs** aufgefordert, den Inhalt der Dateien **config.fs** und **oledTools.fs** zu verarbeiten. Um diese globale Verarbeitung zu starten, können wir Folgendes eingeben:

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

Wenn Sie ein komplexes Projekt haben, müssen Sie diese Behandlung möglicherweise Dutzende oder sogar Hunderte Male eingeben. Erinnern Sie sich, dass wir in der Datei **autoexec.fs** das Wort **MAIN** definiert haben? Erinnerung an die Definition dieses Wortes :

```
: MAIN
  s" /spiffs/main.fs"      included
;
```

Sehr gut ! Wir können also einfach **MAIN** eingeben, anstatt **include /spiffs/main.fs** auszuführen...

Sollen wir den Test machen? OK. Schalten Sie die ESP32-Karte aus. Schalten Sie es wieder ein. Öffnen Sie das Terminal, das mit ESP32forth kommuniziert, und geben Sie **MAIN** ein. Alle Projektinhalte werden fast sofort ausgeführt und kompiliert! Geben Sie **Wörter ein**. Alle Wörter in unserem Projekt sind im **vierten Vokabular enthalten**.

Wir überprüfen, ob es funktioniert, indem wir jetzt Folgendes eingeben:

```
Oled128x32Init
```

Esp32forth anzeigen :

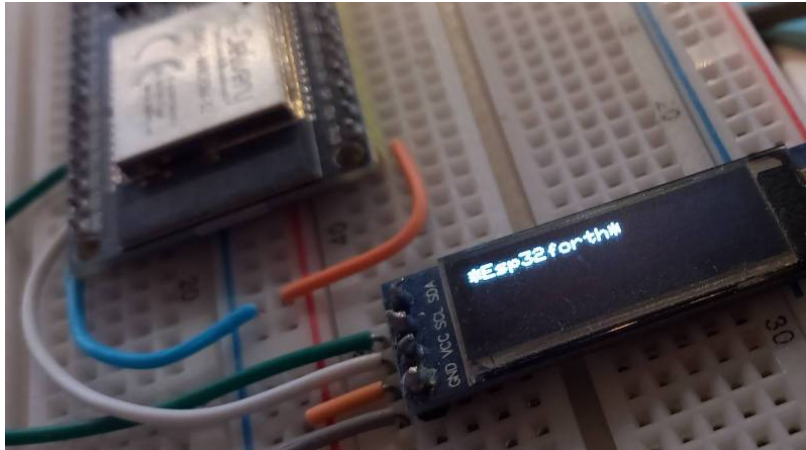


Figure 25: résultat de l'exécution du mot `Oled128x32Init`

Von diesem Moment an können wir alle anderen Wörter unseres Projekts schreiben und umsetzen.

Verwenden Sie OLED-Vokabular

Die Wörter, die wir zur Verwendung unseres SSD1306 128x32 OLED-Displays verwenden werden, sind im **OLED**- Vokabular verfügbar :

```
--> oled vlist
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 OledFastHLine OledFastVLine OledCirc
OledCircF OledRect OledRectF OledRectR OledRectRF oled-builtins
```

Initialisieren des I2C-Busses für das SSD1306 OLED-Display

Unser OLED-Display ist mit dem I2C-Bus verkabelt. Es ist daher an der hexadezimalen Adresse 3c auf diesem I2C-Bus verfügbar. Als erstes muss daher eine Konstante definiert werden :

```
\ set Adresse der OLED SSD1306 Anzeige 128x32 Pixel
$3c constant I2C_SSD1306_ADDRESS
```

Im Oled-Vokabular gibt es eine **OledAddr**- Variable , die für die Speicherung dieser 3c-Adresse verantwortlich ist. Wenn diese Adresse einen Nullwert enthält, liegt das daran, dass der I2C-Bus keine Verbindung mit unserem SSD1306-Display hergestellt hat. Es ist dieser Nullwert, der die Initialisierung dieser Verbindung bedingt:

```
OledAddr @ 0=
if
  \ init I2C communication with SSD1306
then
```

Hier ist der Initialisierungsteil unserer Kommunikation über den I2C-Bus zum SSD1306 OLED-Display :

```
WIDTH HEIGHT OledReset OledNew
SSD1306_SWITCHCAPVCC I2C_SSD1306_ADDRESS OledBegin drop
```

- **WIDTH HEIGHT OledReset OledNew**- Sequenz instanziiert eine neue OLED-Sitzung für unser SSD1306-Display.
- Dem Wort **OledBegin** werden zwei Parameter vorangestellt:
 - **SSD1306_SWITCHCAPVCC** , was die 3,3-V-Versorgung von der ESP32-Karte bestätigt, was in unserer Baugruppe der Fall ist. Wenn wir ein externes Netzteil verwendet hätten, ersetzen wir dieses Wort durch **SSD1306_EXTERNALVCC** .
 - **I2C_SSD1306_ADDRESS** , die die Adresse des OLED-Displays auf dem I2C-Bus angibt.

Diese Initialisierung des I2C-Busses wird bei unserem SSD1306 OLED-Display nur einmal durchgeführt.

Initialisierung der Anzeige für SSD1306

Um eine Anzeige zu starten, initialisieren wir die Anzeige:

- **OledCLS** , das die Löschung des Bildschirminhalts anfordert;
- **1 OledTextsize** , das die Größe des anzuzeigenden Textes angibt;
- **WHITE OledTextc**, der die Farbe des anzuzeigenden Texts angibt. Für das SSD1306-Display gibt es nur die Farben **WEISS** und **SCHWARZ** .

Hier ist das Wort **Oled128x32Init**, das eine ordnungsgemäße Initialisierung ermöglicht:

```
oled
: Oled128x32Init
  OledAddr @ 0=
  if
    WIDTH HEIGHT OledReset OledNew
    SSD1306_SWITCHCAPVCC I2C_SSD1306_ADDRESS OledBegin drop
  then
  OledCLS
  1 OledTextsize      \ Draw 1x Scale Text
  WHITE OledTextc     \ Draw white text
  0 0 OledSetCursor   \ Start at top-left corner
  z" *Esp32forth*" OledPrintln OledDisplay
;
```

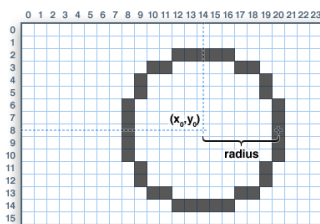
Beim Ausführen **von Oled128x32Init** sollte der Text ***Esp32forth*** auf dem OLED-Bildschirm angezeigt werden.

Hier ist eine Zusammenfassung der Textanzeige-Verwaltungsbefehle für das OLED-Display:

- **OledCLS** (--)
Löscht den Inhalt des OLED-Bildschirms
- **OledDisplay** (--)
überträgt die auf die Anzeige wartenden Befehle an das OLED-Display
- **OledHOME** (--)
Positioniert den Cursor in Zeile 0, Spalte 0 auf dem OLED-Display. Diese Position ist pixelgenau.
- **OledInvert** (--)
Kehrt die OLED-Bildschirmanzeige um
- **OledNum** (n --)
Zeigt die Zahl n als Zeichenfolge auf dem OLED-Bildschirm an
- **OledNumln** (n --)
Zeigt eine Ganzzahl auf dem OLED-Display an und wechselt zur nächsten Zeile
- **OledPrint** (z-string --)
Zeigt Z-String-Text auf dem OLED-Bildschirm an
- **OledPrintln** (z-string --)
Druckt Z-String-Text auf dem OLED-Bildschirm und wechselt zur nächsten Zeile
- **OledTextc** (WEISS|SCHWARZ --)
Legt die Farbe des anzuzeigenden Textes fest
- **OledSetCursor** (xy --)
Legt die Cursorposition fest
- **OledTextsize** (Größe --)
Legt die Größe des Texts fest, der auf dem OLED-Bildschirm angezeigt wird. Der Wert von n muss im Intervall [1..3] liegen. Für Text in normaler Größe ist size=1. Wenn Sie den Wert 4 überschreiten, wird der Text bei einer 4-zeiligen Anzeige abgeschnitten.

Und hier ist eine Zusammenfassung der Befehle zur Verwaltung der Grafikanzeige:

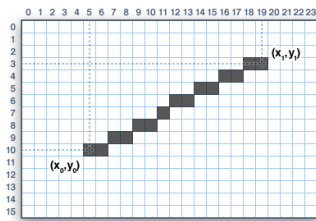
- **OledCirc** (xy radius color --)
Zeichnet einen Kreis mit Mittelpunkt xy, mit Radius radius und Farbe color (0|1)



- **OledCircF** (xy-Radiusfarbe --)
Zeichnet einen Vollkreis mit Mittelpunkt bei xy, mit Radius und Farbe (0|1)

- **OledDrawL** (x0 y0 x1 y1 color --)

Zeichnet eine Linie von x0 y0 bis x1 y1 der Farbe color.



- **OledFastHLine** (xy length color --)
Zeichnet eine horizontale Linie von xy mit der Dimensionslänge und der Farbe color.
- **OledFastVLine** (xy length color --)
Zeichnet eine vertikale Linie von xy mit der Dimensionslänge und der Farbe color.
- **OledPixel** (xy color)
Aktiviert ein Pixel an der Position x y. Der Farbparameter bestimmt die Farbe des Pixels.
- **OledRect** (xy width height color --)
Zeichnet ein leeres Rechteck aus der xy-Position von size width height und color color.
- **OledRectF** (xy width height color --)
Zeichnet ein ausgefülltes Rechteck aus der xy-Position von size width height und color color.
- **OledRectR** (xy width height radius color --)
Zeichnet ein leeres Rechteck mit abgerundeten Ecken, ausgehend von der xy-Position, der Dimension width height, in der Farbe color, mit einem Radius radius.
- **OledRectRF** (xy width height radius color --)
Zeichnet von der xy-Position aus ein festes Rechteck mit abgerundeten Ecken der Bemaßung width height in der Farbe color mit einem Radius radius nach.

Das OLED-Display ermöglicht die Ausführung von Text- und Grafikverwaltungswörtern im gleichen Anzeigemodus. Kurz gesagt, Sie können Text und Grafiken mischen.

Erweitern Sie den OLED-Wortschatz

OledTriangle- Erweiterung wie diese definieren kann, um die Definitionen in der Datei **oled.h** zu erweitern :

```
YV(oled, OledTriangle, oled_display->drawTriangle(n5, n4, n3, n2, n1, n0 ); DROPn(6)) \
```

Dabei ist jedoch nicht berücksichtigt, dass wir in der FORTH-Sprache programmieren und dass wir mit dieser Sprache unser OLED-Vokabular erweitern können. Wir werden daher auf unserem Computer in unserem Projektverzeichnis eine neue Datei mit dem Dateinamen **extendOledVoc.fs** erstellen. Inhalt :

```

RECORDFILE /spiffs/extendOledVoc.fs
oled definitions
: OledTriangle { x0 y0 x1 y1 x2 y2 color -- }
  x0 y0 x1 y1 color OledDrawL
  x1 y1 x2 y2 color OledDrawL
  x2 y2 x0 y0 color OledDrawL
;
forth definitions
<EOF>

```

Anschließend kopieren wir diesen Code, fügen ihn ein und übermitteln ihn über das Terminal, das mit der ESP32-Karte kommuniziert, an ESP32. Wir sollten am Ende eine Datei **extendOledVoc.fs** im SPIFFS-Speicherraum haben. Wir ändern nun den Inhalt der Datei **main.fs** :

```

s" /spiffs/config.fs"      included
s" /spiffs/extendOledVoc.fs"  included
s" /spiffs/oledTools.fs"    included

```

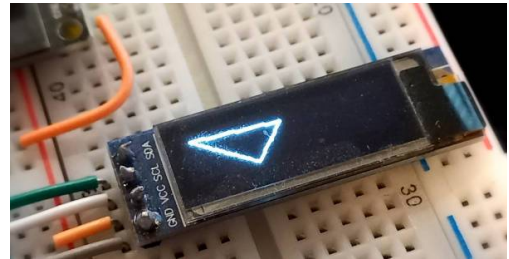
Wir trennen die ESP32-Karte und schließen sie wieder an. Wir geben **MAIN** in das Terminal ein. Wenn alles gut gelaufen ist, sollten Sie das Wort **OledTriangle** finden , indem Sie einfach **oled** vlist eingeben.

Test unseres neuen Wortes **OledTriangle**:

```

oled
OledCLS OledDisplay
5 5 60 8 40 30 WHITE OledTriangle OledDisplay

```



TEMPVS FVGIT⁷

Was wäre, wenn die Römer die Zeitanzeige digital programmieren könnten?

Dies ist ein interessantes Projekt, das mehrere Dateien kombiniert. In diesem Kapitel werden wir nicht den gesamten hier verwendeten Code angeben. Es wäre zu lang.

Die Quellcodes für dieses Kapitel befinden sich in dieser Datei:

- **ESP32forth-book.zip** → Projekte → tempusFugit

Link: <https://github.com/MPETREMAN11/ESP32forth/blob/main/documentation/ESP32forth-book.zip>

Das gesamte Projekt enthält diese Dateien:

- von **autoexec.fs** wird geladen, wenn ESP32forth startet
- **clepsydra.fs** konvertiert Zahlen in römische Ziffern
- **config.fs** globale Konfigurationseinstellungen
- **main.fs** lädt die anderen Projektdateien
- **oledTools.fs** vervollständigt das **Oled- Vokabular**
- **RTClock.fs** verwaltet die Echtzeituhr
- **strings.fs** übernimmt die Verarbeitung alphanumerischer Zeichenfolgen

Die Reihenfolge des Ladens von Projektdateien ist in **main.fs** geschrieben :

```
s" /spiffs/strings.fs"      included
s" /spiffs/RTClock.fs"     included
s" /spiffs/clepsydra.fs"   included
s" /spiffs/config.fs"      included
s" /spiffs/oledTools.fs"   included
```

Die meisten Dateien in diesem Projekt sind unabhängig, mit Ausnahme von **clepsydra.fs**, das von **strings.fs** abhängig ist.

Romani non ustulo nulla⁸

Die Römer kannten die Zahl 0 nicht. Wie können wir also **13:00** oder **00:15** in römischen Ziffern anzeigen?

Um das Problem der Stunden nach Mitternacht, zum Beispiel 00:15 Uhr, zu lösen, werden uns die Japaner (Einwohner



⁷ Tempus fugit = Zeit vergeht

⁸ Romani non ustulo nulla = Die Römer kannten die Null nicht

JAPANS) eine große Hilfe sein. Wenn Sie jemals in dieses Land reisen, werden Sie erstaunt sein, dass die Geschäfte bis **25:00 Uhr geöffnet sind** !

Dieser Laden ist von 09:00 bis 25:00 Uhr geöffnet! Oh ja. Allerdings haben auch die Uhren in Japan das 24-Stunden-Format. Wir wussten, dass die Japaner fleißig waren, aber wenn man bedenkt, dass sie 25 Stunden am Tag arbeiten, haben wir das Recht, einige Zweifel zu haben ...

Tatsächlich gibt es eine sehr logische Erklärung. Nach 12:00 Uhr ist es 12:01 Uhr usw. Und so ist es nach 23:59 Uhr 24:00 Uhr und dann 24:01 Uhr. Wenn also ein Geschäft um 25:00 Uhr schließt, müssen wir verstehen, dass es für uns um 01:00 Uhr schließt.

Wenn wir dies auf unsere römische Uhr übertragen, können wir bei **00:00** Uhr **XXIV** oder besser **XXIII:IX** (23:60) anzeigen.

Romani horas et minuta⁹

Um den Fall von Stunden wie 01:00, 02:00 ... 23:00 aufzulösen, können wir logischerweise nach 12:59 sehr gut 12:60 anzeigen, dann die Minute nach 13:01 ... 12:60 Römische Ziffern: **XXII:IX** .

Dies wird mit dem Wort **tempusTo\$** erreicht:

```
: tempusTo$ { HH MM -- }
  HH 0 = MM 0= AND if
    60 to MM
    23 to HH
  THEN
  HH 0 > MM 0= AND if
    60 to MM
    -1 +to HH
  then
  HH 0 <= if
    24 to HH
  then
  HH roman tempus $!
  [char] : tempus c+$! \ add char :
  MM roman tempus append$
  tempus
;
```

Im ersten if..then-Test testen wir, ob wir bei **00:00 sind** . In diesem Fall erzwingen wir die Stunde auf **23** und die Minuten auf **60** .

Wenn im zweiten Test die Stunde größer als 00 und die Minuten größer als 00 sind, dekrementieren wir die Stunde und erzwingen die Minuten auf **60** . Der Nachteil besteht darin, dass wir die Zeit, wenn sie 00 ist, auf -1 ändern.

9 Romani horas et minuta = römische Stunden und Minuten

Wenn die Stunde im letzten Test Null oder negativ ist, erzwingen wir sie auf **24**.

Wir können das Wort **.tempus** verwenden, das für die Entwicklung verwendet wurde, um diesen korrekten Vorgang zu überprüfen:

```
--> 23 59 .tempus
XXIII:LIX ok
--> 0 0 .tempus
XXIII:LX ok
--> 0 1 .tempus
XXIV:I ok
--> 1 0 .tempus
XXIV:LX ok
--> 1 1 .tempus
I:I ok
```

Haec Omnia Integramus pro ESP32forth¹⁰

Im aktuellen Stand des Projekts müssen Sie die Anfangszeit manuell eingeben:

```
22 19 start
```

Bedeutet, dass wir die Zeit auf **22:19** initialisieren. Diese Initialisierung erfolgt ganz einfach:

```
0 RTC.setTime
```

Dann initialisieren wir das 128x32 OLED-Display:

```
Oled128x32Init
1 OledTextsize
WHITE OledTextc
```

Und schließlich rufen wir die aktuelle Uhrzeit ab und zeigen sie an:

```
OledCLS OledDisplay
16 20 OledSetCursor
RTC.getTime drop tempusTo$ s>z OledPrintln OledDisplay
```

Ich habe Tests mit einem größeren Display durchgeführt. Das Problem besteht darin, dass für den Text **XXIII:XXVIII** nicht genügend Platz zur Anzeige dieses Texts vorhanden ist.

Hier ist die letzte Schleife, die ausgeführt werden muss, um mit der Anzeige der Zeit in römischen Ziffern zu beginnen:

```
oled
: start ( HH MM -- )
  0 RTC.setTime      \ define current time
  Oled128x32Init
  1 OledTextsize
  WHITE OledTextc
  begin
```

¹⁰ Haec omnia integramus pro ESP32forth = Wir integrieren das alles für ESP32forth

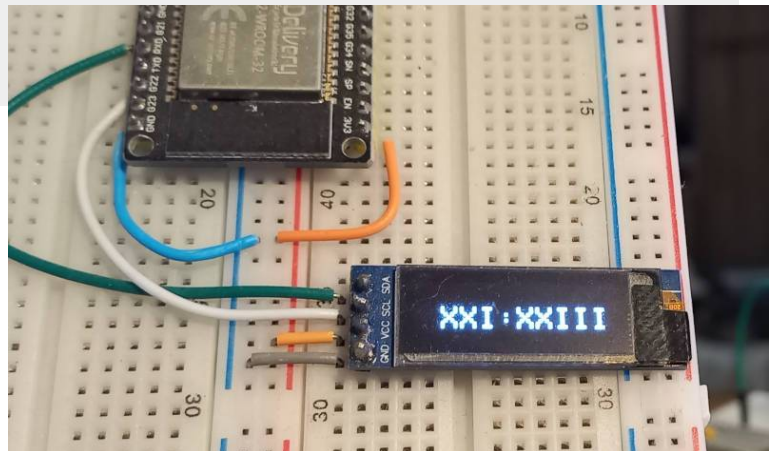
```

OledCLS OledDisplay
16 20 OledSetCursor
RTC.getTime drop tempusTo$ s>z OledPrintln OledDisplay
1000 ms
key? until
;
forth

```

Das Programm kann verbessert werden, indem die Uhrzeit von einem Zeitserver abgerufen wird. Siehe Kapitel *Uhrzeit von einem WEB-Server abrufen* .

Timerfunktionen zu nutzen, um den Interpreter freizugeben. Siehe Kapitel *„Blinken einer LED per Timer“* .



Schließlich dauerte das Zusammenstellen der verschiedenen Dateien für dieses Projekt und die wenigen Tests und Anpassungen einen Nachmittag.

Ich möchte jedoch einige Punkte hervorheben:

- Erstellen Sie in Ihrem Projektordner immer eine Kopie einer Allzweckdatei. Zum Beispiel für die Datei **strings.fs** ;
- Wenn Sie eine allgemeine Komponente kopieren, zum Beispiel **strings.fs** oder **RTClock.fs** , nehmen Sie nur Änderungen an den Dateien vor, die in Ihren Projektordner kopiert wurden. Versionieren Sie diese Änderungen und geben Sie das Änderungsdatum im Kopfkomentar der geänderten Datei an.

Während Sie Ihre Projekte ausführen, stellen Sie möglicherweise fest, dass dieselbe Datei in verschiedene Ordner kopiert und geändert wird. Diese Lösung ist einer einzelnen Datei voller Anpassungsparameter vorzuziehen.

Fügen Sie die SPI-Bibliothek hinzu

Die SPI-Bibliothek ist in ESP32forth nicht nativ implementiert. Um es zu installieren, müssen Sie zunächst die Datei **spi.h** erstellen, die im selben Ordner installiert werden muss wie der Ordner, der die Datei **ESP32forth.ino** enthält.

Inhalt der **spi.h**- Datei (in C-Sprache) :

```
# include <SPI.h>

#define OPTIONAL_SPI_VOCABULARY V(spi)
#define OPTIONAL_SPI_SUPPORT \
    XV(internals, "spi-source", SPI_SOURCE, \
        PUSH spi_source; PUSH sizeof(spi_source) - 1) \
    XV(spi, "SPI.begin", SPI_BEGIN, SPI.begin((int8_t) n3, (int8_t) n2, (int8_t) n1, (int8_t) n0); DROPn(4)) \
    XV(spi, "SPI.end", SPI_END, SPI.end();) \
    XV(spi, "SPI.setHwCs", SPI_SETHWCS, SPI.setHwCs((boolean) n0); DROP) \
    XV(spi, "SPI.setBitOrder", SPI_SETBITORDER, SPI.setBitOrder((uint8_t) n0); DROP) \
    XV(spi, "SPI.setDataMode", SPI_SETDATAMODE, SPI.setDataMode((uint8_t) n0); DROP) \
    XV(spi, "SPI.setFrequency", SPI_SETFREQUENCY, SPI.setFrequency((uint32_t) n0); DROP) \
    XV(spi, "SPI.setClockDivider", SPI_SETCLOCKDIVIDER, SPI.setClockDivider((uint32_t) n0); DROP) \
    XV(spi, "SPI.getClockDivider", SPI_GETCLOCKDIVIDER, PUSH SPI.getClockDivider();) \
    XV(spi, "SPI.transfer", SPI_TRANSFER, SPI.transfer((uint8_t *) n1, (uint32_t) n0); DROPn(2)) \
    XV(spi, "SPI.transfer8", SPI_TRANSFER_8, PUSH (uint8_t) SPI.transfer((uint8_t) n0); NIP) \
    XV(spi, "SPI.transfer16", SPI_TRANSFER_16, PUSH (uint16_t) SPI.transfer16((uint16_t) n0); NIP) \
    XV(spi, "SPI.transfer32", SPI_TRANSFER_32, PUSH (uint32_t) SPI.transfer32((uint32_t) n0); NIP) \
    XV(spi, "SPI.transferBytes", SPI_TRANSFER_BYTES, SPI.transferBytes((const uint8_t *) n2, (uint8_t *) n1, (uint32_t) n0); DROPn(3)) \
    XV(spi, "SPI.transferBits", SPI_TRANSFER_BITES, SPI.transferBits((uint32_t) n2, (uint32_t *) n1, (uint8_t) n0); DROPn(3)) \
    XV(spi, "SPI.write", SPI_WRITE, SPI.write((uint8_t) n0); DROP) \
    XV(spi, "SPI.write16", SPI_WRITE16, SPI.write16((uint16_t) n0); DROP) \
    XV(spi, "SPI.write32", SPI_WRITE32, SPI.write32((uint32_t) n0); DROP) \
    XV(spi, "SPI.writeBytes", SPI_WRITE_BYTES, SPI.writeBytes((const uint8_t *) n1, (uint32_t) n0); DROPn(2)) \
    XV(spi, "SPI.writePixels", SPI_WRITE_PIXELS, SPI.writePixels((const void *) n1, (uint32_t) n0); DROPn(2)) \
    XV(spi, "SPI.writePattern", SPI_WRITE_PATTERN, SPI.writePattern((const uint8_t *) n2, (uint8_t) n1, (uint32_t) n0); DROPn(3))

const char spi_source[] = R"""(
vocabulary spi    spi definitions
transfer spi-builtins
forth definitions
)""";
```

Die vollständige Datei ist auch hier verfügbar:

<https://github.com/MPETREMANN11/ESP32forth/blob/main/optional/spi.h>

Änderungen an der Datei ESP32forth.ino

Datei **spi.h** kann nicht in ESP32forth integriert werden, ohne dass einige Änderungen an der Datei **ESP32forth.ino vorgenommen werden**. Hier sind die wenigen Änderungen, die an dieser Datei vorgenommen werden müssen. Diese Änderungen wurden in Version 7.0.7.15 vorgenommen, sollten aber auch auf andere neuere oder zukünftige Versionen anwendbar sein.

Erste Modifikation

Code in Rot hinzugefügt:

```
#define VOCABULARY_LIST \
    V(forth) V(internals) \
```

```
V(rtos) V(SPIFFS) V(serial) V(SD) V(SD_MMC) V(ESP) \
V(ledc) V(Wire) V(WiFi) V(sockets) \
OPTIONAL_CAMERA_VOCABULARY \
OPTIONAL_BLUETOOTH_VOCABULARY \
OPTIONAL_INTERRUPTS_VOCABULARIES \
OPTIONAL_OLED_VOCABULARY \
OPTIONAL_SPI_VOCABULARY \
OPTIONAL_RMT_VOCABULARY \
OPTIONAL_SPI_FLASH_VOCABULARY \
USER_VOCABULARIES
```

Zweite Modifikation

Zusatz in Rot nach diesem Code:

```
// Hook to pull in optional Oled support.
# if __has_include("oled.h")
# include "oled.h"
# else
# define OPTIONAL_OLED_VOCABULARY
# define OPTIONAL_OLED_SUPPORT
# endif

// Hook to pull in optional SPI support.
# if __has_include("spi.h")
# include "spi.h"
# else
# define OPTIONAL_SPI_VOCABULARY
# define OPTIONAL_SPI_SUPPORT
# endif
```

Dritte Modifikation

Zusatz in Rot:

```
#define EXTERNAL_OPTIONAL_MODULE_SUPPORT \
OPTIONAL_ASSEMBLERS_SUPPORT \
OPTIONAL_CAMERA_SUPPORT \
OPTIONAL_INTERRUPTS_SUPPORT \
OPTIONAL_OLED_SUPPORT \
OPTIONAL_SPI_SUPPORT \
OPTIONAL_RMT_SUPPORT \
OPTIONAL_SERIAL_BLUETOOTH_SUPPORT \
OPTIONAL_SPI_FLASH_SUPPORT
```

Vierte Modifikation

Zusatz in Rot:

```
internals DEFINED? oled-source [IF]
  oled-source evaluate
[THEN] forth
```

```
internals DEFINED? spi-source [IF]
  spi-source evaluate
[THEN] forth
```

Wenn Sie diese Anweisungen sorgfältig befolgen, können Sie ESP32forth mit ARDUINO IDE kompilieren und auf das ESP32-Board hochladen. Sobald diese Vorgänge abgeschlossen sind, starten Sie das Terminal. Sie müssen die ESP32forth-Willkommensaufforderung finden. Typ:

```
spi vlist
```

spi- Vokabular definierten Wörter finden :

```
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
```

Sie können jetzt Erweiterungen über den SPI-Port ansteuern, beispielsweise die LED-Anzeigen MAX7219.

Kommunizieren Sie mit dem Anzeigemodul MAX7219

Bei der SPI-Kommunikation gibt es immer einen Master , *der* die Peripheriegeräte (auch *Slaves genannt*) steuert. Daten können gleichzeitig gesendet und empfangen werden. Das bedeutet, dass der Master Daten an einen Slave senden kann und ein Slave gleichzeitig Daten an den Master senden kann.

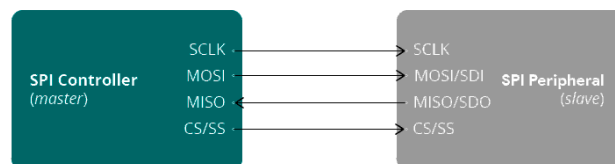


Figure 26: Steuern eines SPI-Geräts

Sie können mehrere Sklaven haben. Ein Slave kann ein Sensor, ein Display, eine microSD-Karte usw. oder ein anderer Mikrocontroller sein. Das bedeutet, dass Sie Ihren ESP32 mit **mehreren Geräten** verbinden können .

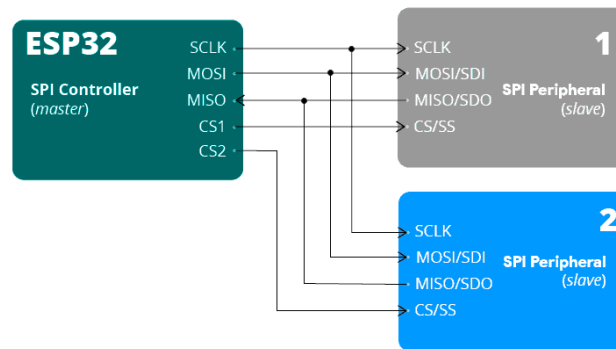


Figure 27: Steuerung von zwei SPI-Geräten

Ein Slave wird ausgewählt, indem der CS1- oder CS2-Selektor auf den niedrigen Pegel gestellt wird. Es werden so viele CS-Selektoren benötigt, wie Slaves verwaltet werden müssen.

Suchen des SPI-Ports auf der ESP32-Karte

Auf einem ESP32-Board gibt es zwei SPI-Ports: HSPI und VSPI. Der SPI-Port, den wir verwalten werden, ist derjenige, dessen Pins das Präfix VSPI haben:

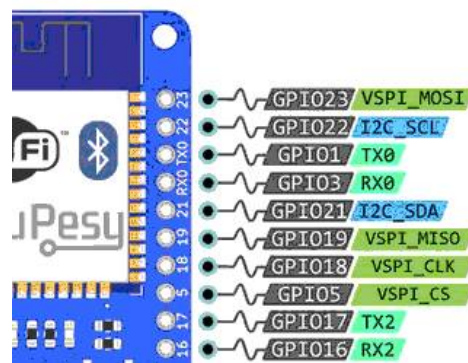


Figure 28: die beiden SPI-Ports auf der ESP32-Karte

Mit ESP32forth können wir daher die Konstanten definieren, die auf diese VSPI-Pins zeigen:

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

Um mit dem Anzeigemodul MAX7219 zu kommunizieren, müssen wir nur die Pins VSPI_MOSI, VSPI_SCLK und VSPI_CS verdrahten.

SPI-Anschlüsse am MAX7219-Anzeigemodul

Hier ist die SPI-Port-Anschlussbelegung auf dem MAX7219-Modul:



Figure 29: SPI-Anschlüsse am MAX7219-Modul

Verbindung zwischen dem MAX7219-Modul und der ESP32-Karte:

MAX7219		ESP32
DIN	<---->	VSPI_MOSI
CS	<---->	VSPI_CS
CLK	<---->	VSPI_SCLK

Die VCC- und GND-Anschlüsse sind mit einer externen Stromversorgung verbunden:

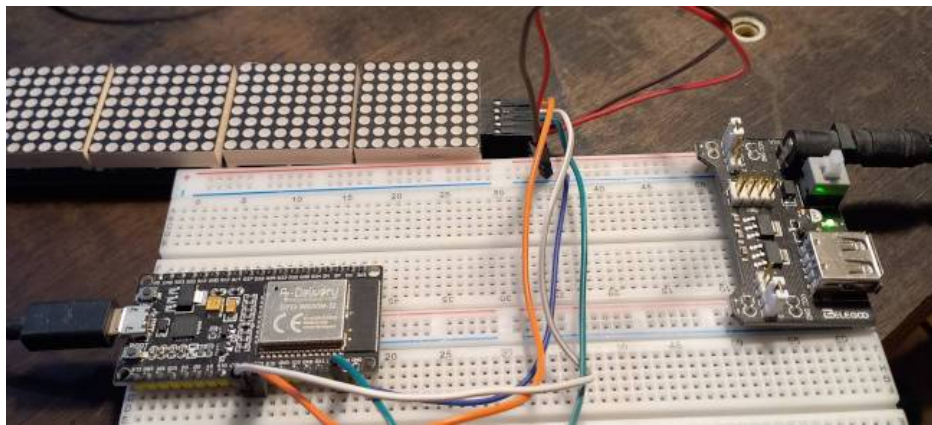


Figure 30: über ein externes Netzteil

Der GND-Teil dieser externen Stromversorgung wird mit dem GND-Pin der ESP32-Karte geteilt.

SPI-Port-Softwareschicht

Alle Wörter zur Verwaltung des SPI-Ports sind bereits im **SPI**- Vokabular verfügbar .

Das einzige, was definiert werden muss, ist die Initialisierung des SPI-Ports:

```
\ Definieren Sie die SPI-Portfrequenz
4000000 constant SPI_FREQ

\ SPI-Vokabular auswählen
only FORTH SPI also

\ SPI-Port initialisieren
: init.VSPI ( -- )
  VSPI_CS OUTPUT pinMode
  VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
```

```
SPI_FREQ SPI.setFrequency  
;
```

Jetzt können wir unser Anzeigemodul MAX7219 verwenden.

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

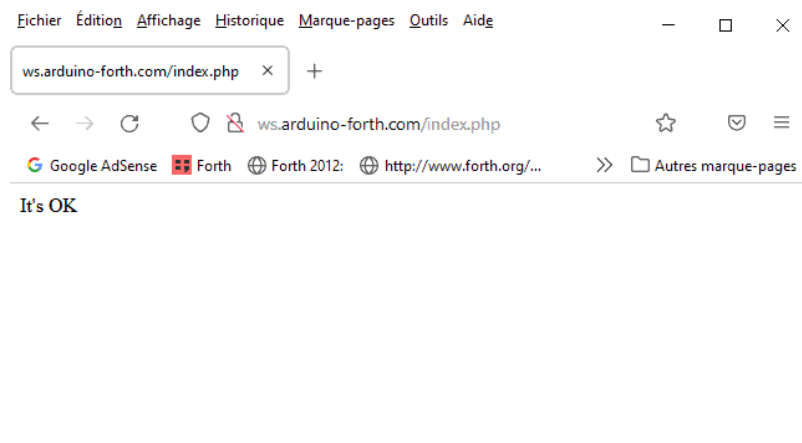
Um unseren HTTP-Client zu testen, können wir dies tun, indem wir einen beliebigen Webserver abfragen. Aber für das, was wir später betrachten, benötigen Sie einen persönlichen Webserver. Auf diesem Server erstellen wir eine Subdomain:

- 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.

Verwaltung von digitalen/analogen Ausgängen

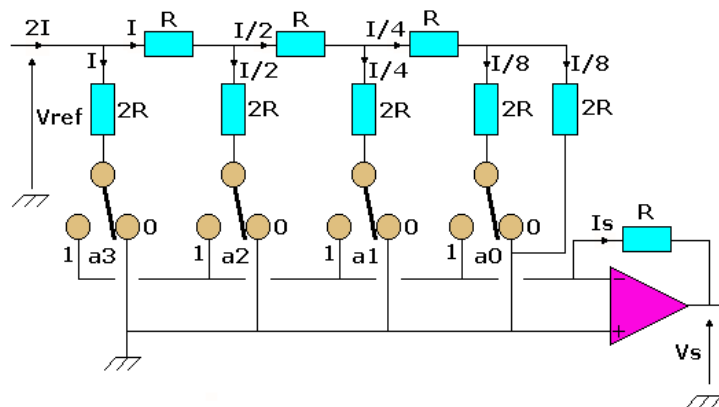
Digital/Analog-Wandlung

Die Umwandlung einer digitalen Größe in eine zu dieser digitalen Größe proportionale elektrische Spannung ist eine sehr interessante Funktionalität auf einem Mikrocontroller.

Wenn Sie das Internet nutzen und einen VoIP-Telefonanruf tätigen, wird Ihre Stimme in numerische Werte umgewandelt. Das Ihres Gesprächspartners wird umgekehrt von Zahlen in Tonsignale umgewandelt. Dieser Prozess nutzt die Analog-Digital-Umwandlung und umgekehrt.

D/A-Wandlung mit R2R-Schaltung

Hier ist das Grunddiagramm eines 4-Bit-Digital-Analog-Wandlers:



Der zu konvertierende Wert wird auf 4 Bits auf die 4 Pins a0 bis a3 verteilt. Die Referenzspannung wird oben links in die Schaltung eingespeist. Diese Spannung erzeugt eine Intensität von $2I$, wenn dieser Strom keinen Widerstand durchfließt.

Abhängig von den aktivierten Bits wird für jedes Bit die Spannung geteilt und zu der der anderen aktiven Bits addiert. Wenn beispielsweise a2 und a0 aktiv sind, entspricht der Ausgangsstrom I_s der Summe $I/2$ und $I/8$.

Für diese 4-Bit-Schaltung beträgt der Konvertierungsschritt $I/16$. Bei ESP32 erfolgt die Konvertierung auf 8 Bit. Der Konvertierungsschritt wird daher $I/256$ sein.

D/A-Wandlung mit ESP32

Kein ARDUINO-Board verfügt über einen D/A-Wandlungsausgang. Um eine D/A-Wandlung mit einer ARDUINO-Karte durchzuführen, müssen Sie eine externe Komponente verwenden.

Bei der ESP32-Karte haben wir zwei Pins, G25 und G26, die den D/A-Wandlungsausgängen entsprechen.

Für unser erstes D/A-Umwandlungsexperiment mit der ESP32-Karte werden wir zwei LEDs an die Pins G25 und G26 anschließen:

```
\ definiere Gx für LEDs
25 constant ledBLUE      \ blaue LED auf G25
26 constant ledWHITE     \ weiße LED auf G26
```

Bevor wir eine D/A-Wandlung durchführen, planen wir, die Pins G25 und G26 zu initialisieren :

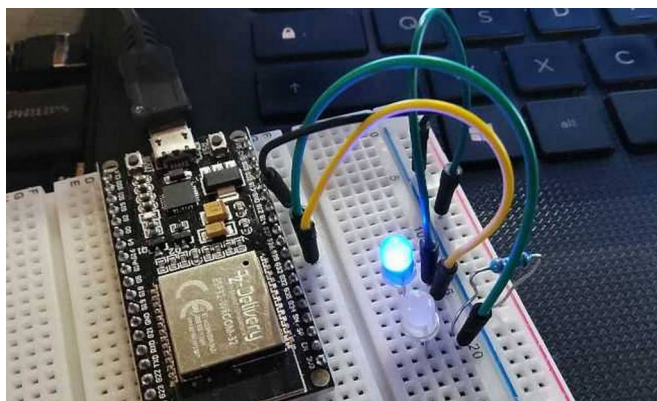
```
\ init Gx als Ausgabe
: initLeds ( -- )
    ledBLUE  output pinMode
    ledWHITE output pinMode
;
```

Und wir definieren zwei Wörter, mit denen wir die Intensität unserer beiden LEDs steuern können:

```
\ Intensität der BLAUEN LED einstellen
: BLset ( val -- )
    ledBLUE  swap dacWrite
;

\ Intensität der WEISSEN LED einstellen
: WHset ( val -- )
    ledWHITE swap dacWrite
;
```

Die Wörter **BLset** und **WHset** akzeptieren als Parameter einen numerischen Wert im Bereich 0..255.



Auf dem Foto lässt die Sequenz **200 BLset** nach **initLeds** die blaue LED mit reduzierter Leistung leuchten.

Um es mit voller Leistung einzuschalten, verwenden wir die Sequenz **255 BLset**

Um es vollständig auszuschalten, senden wir dieser Sequenz **0 BLset**

Möglichkeiten der D/A-Wandlung

Hier haben wir mit unseren beiden LEDs eine einfache und uninteressante Baugruppe geschaffen.

Diese Montage hat den Vorzug, dass sie zeigt, dass die D/A-Wandlung perfekt funktioniert. Die D/A-Wandlung ermöglicht:

- Leistungssteuerung über einen speziellen Schaltkreis, beispielsweise einen Variator für einen Elektromotor ;
- Erzeugung von Signalen: Sinus, Quadrat, Dreieck usw.
- Konvertierung von Sounddateien
- Klangsynthese...

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>

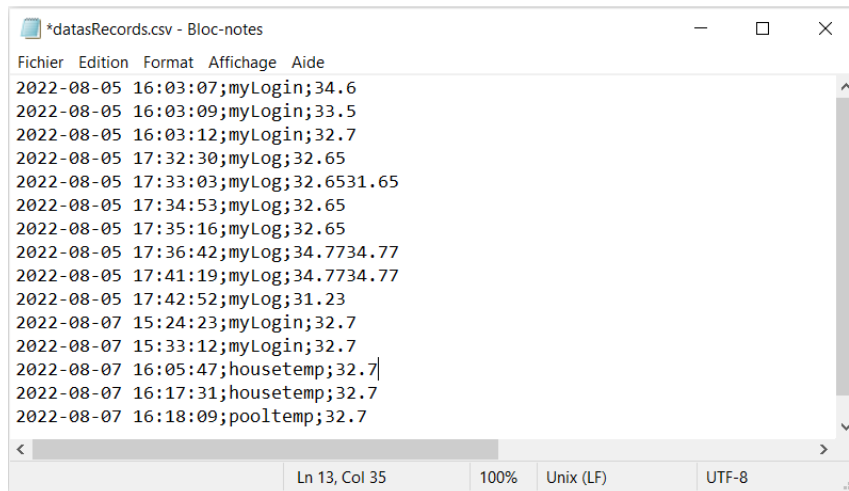
log pwd Paar , dessen Werte vom Datenaufzeichnungsskript getestet und genehmigt werden.

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=pool12022&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.

Klangsynthese mit ESP32Forth

Für erste Klangexperimente benötigen Sie einen Lautsprecher, den Sie an einen GPIO-Ausgang anschließen. Da die Impedanz der Lautsprecher jedoch sehr niedrig ist, muss ein Transistor angeschlossen werden. Hier ist das empfohlene Diagramm für einen kleinen Lautsprecher.

In diesem Diagramm wird der GPIO4-Pin erwähnt. Tatsächlich kann diese Baugruppe an jedem GPIO-Ausgang der ESP32-Karte verwendet werden. Die beiden Ausgänge, die uns besonders interessieren werden, sind GPIO25 und GPIO26, die für DAC-Ausgänge (Digital to Analog Conversion) reserviert sind.

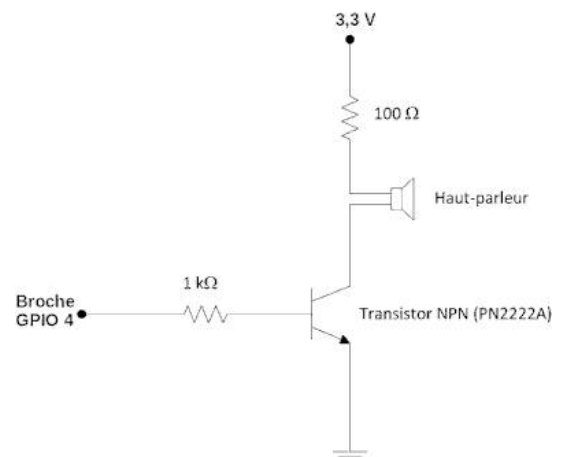


Figure 31: Anschließen eines Lautsprechers

Einfache Klangsynthese

Wir werden die PWM-Signalerzeugung verwenden, jedoch an den DAC-Ausgängen.

Unser Lautsprecher ist über den PN2222A-Transistor, der als Impedanzadapter dient, mit dem GPIO25-Ausgang verbunden.

```
0 constant CHANNEL0      \define PWM-Kanal 0
25 constant BUZZER        \ Lautsprecher angeschlossen an GPIO25

ledc                        \ wählen Sie das ledc-Vokabular aus
: initTones ( -- )
    BUZZER CHANNEL0 ledcAttachPin
;
```

Das Wort **initTones** verbindet den GPIO25-Ausgang mit dem PWM-Kanal 0. Das Erzeugen eines Sounds geschieht folgendermaßen:

```
CHANNEL0 freq ledcWriteTone drop
```

Dabei ist freq die gewünschte Frequenz, multipliziert mit 1000. Um also die Note LA (A in englischer Notation) zu erzeugen, deren Frequenz 440 Hz beträgt, müssen Sie den Wert 440*1000 verwenden:

```
CHANNEL0 440000 ledcWriteTone drop
```

Definition der Schallfrequenztafel

Um die Klangfrequenzen von Musiknoten zu finden, haben wir Wikipedia aufgesucht. Wir erstellen eine Frequenztafel, in der jede Frequenz in ihrer für **ledcWriteTone** verwendbaren Form aufgezeichnet wird :

```

\ frequency notes
\ source: https://fr.wikipedia.org/wiki/Note_de_musique
\ frequency is multiplied by 1000
create NOTES
\ octave -1
15350 , 17330 , 18360 , 19450 , 20600 , 21830 ,
23130 , 24500 , 25960 , 27500 , 29140 , 30870 ,
\ octave 0
32700 , 34650 , 36710 , 38890 , 41200 , 43650 ,
46250 , 49000 , 51910 , 55000 , 58270 , 61740 ,
\ octave 1
65410 , 69300 , 73420 , 77780 , 82410 , 87310 ,
92500 , 98000 , 103830 , 110000 , 116540 , 123470 ,
\ octave 2
130810 , 138590 , 146830 , 155560 , 164810 , 174610 ,
185000 , 196000 , 207650 , 220000 , 233080 , 246940 ,
\ octave 3
261630 , 277180 , 293660 , 311130 , 329630 , 349230 ,
369990 , 392000 , 415300 , 440000 , 466160 , 493880 ,
\ octave 4
523250 , 554370 , 587330 , 622250 , 659260 , 698460 ,
739990 , 783990 , 830610 , 880000 , 932330 , 987770 ,
\ octave 5
1046500 , 1108730 , 1174660 , 1244510 , 1318510 , 1396910 ,
1479980 , 1567980 , 1661220 , 1760000 , 1864660 , 1975530 ,
\ octave 6
2093000 , 2217460 , 2349320 , 2489020 , 2637020 , 2793830 ,
2959960 , 3135960 , 3322440 , 3520000 , 3729310 , 3951070 ,
\ octave 7
4186010 , 4434920 , 4698640 , 4978030 , 5274040 , 5587650 ,
5919910 , 6271930 , 6644880 , 7040000 , 7458620 , 7902130 ,
\ octave 8
8372020 , 8869840 , 9397280 , 9956060 , 10548080 , 11175300 ,
11839820 , 12543860 , 13289760 , 14080000 , 14917240 , 15804260 ,

```

Es gibt zwölf Töne pro Oktave, daher die Definition von 12 Werten pro Oktave. Hier nehmen wir nur 10 Zeilen oder 10 Oktaven auf. Denn ab 15Khz wären die Geräusche nicht mehr hörbar.

Um eine Note zu finden, müssen Sie lediglich ihre Position in einer Oktave kennen. Beispielsweise lautet unsere A-Note in Oktave 3: $((\text{Oktave}+1)*12)+\text{Position}$. Wenn sich A in der 10. Position in Oktave 3 befindet, lautet die zu bestimmende Adresse $\text{NOTES}+4*((\text{OCTAVE}+1*12)+\text{position})$

Abrufen der Frequenz einer Musiknote

Wir erstellen zunächst ein Wort **set.octave** , mit dem wir die gewünschte Oktave auswählen können. Dann definieren wir **get.note** , das die Häufigkeit der gewünschten Note abrufen:

```
3 value OCTAVE
\ Oktave im Intervall -1..8 auswählen
: set.octave ( n[-1..8] )
  to OCTAVE
;

\ Note im Intervall 1..12 auswählen
: get.note ( n[1..12] -- )
  1- OCTAVE 1+ 12 * + cell *      \ calc. offset in NOTES array
  NOTES + @                        \ fetch frequency of selected note
;

3 value OCTAVE
\ Oktave im Intervall -1..8 auswählen
: set.octave ( n[-1..8] )
  to OCTAVE
;

: OCT6 ( -- )    6 set.octave ;
: OCT5 ( -- )    5 set.octave ;
: OCT4 ( -- )    4 set.octave ;
: OCT3 ( -- )    3 set.octave ;
: OCT2 ( -- )    2 set.octave ;
: OCT1 ( -- )    1 set.octave ;
```

Wir werden später sehen, wie man Notizen verwaltet, indem man sie aus ihrer Notation heraus aufruft.

Verwalten der Notendauer

Die Dauer einer Note ist das Zeitintervall zwischen dem Auslösen zweier aufeinanderfolgender Noten.

Eine Basisverzögerung wird durch die **WHOLE-NOTE-DURATION**- Konstante definiert .

Die Dauern sind in einem neuen **music** vokabular definiert :

```
1600 constant WHOLE-NOTE-DURATION
WHOLE-NOTE-DURATION value duration

vocabulary music
music definitions
music also
```

```

\ set duration of a whole note
: o ( -- )
  WHOLE-NOTE-DURATION to duration
;

\ set duration of a white note
: o| ( -- )
  WHOLE-NOTE-DURATION 2/ to duration
;

\ set duration of a black note
: .| ( -- )
  WHOLE-NOTE-DURATION 2/ 2/ to duration
;

\ set duration of a half black note
: .|' ( -- )
  WHOLE-NOTE-DURATION 2/ 2/ 2/ to duration
;

\ set duration of a quarter black note
: .|" ( -- )
  WHOLE-NOTE-DURATION 2/ 2/ 2/ 2/ to duration
;

```

Wir definieren Wörter, die die gewünschte Dauer symbolisieren: **o** für eine ganze Note, **\o** für eine halbe Note, **\.** für eine schwarze Notiz usw.

One-Note-Unterstützung

Der Sustain einer Note ist die Zeitspanne, die die Note während ihrer Spielzeit hörbar ist. Wir definieren einen **sustain**- Wert, der den Prozentsatz des Emissions-Sustains der Note während ihrer Gesamtdauer ausdrückt. Wenn dieser Wert 100 beträgt, folgen die Noten aufeinander, ohne dass es zwischen den Noten zu einer Pause kommt.

```

\ Sustain der Note, im Intervall [0..100]
90 value SUSTAIN

ledc
\ Note im Intervall [0..100] halten
: sustain.note ( -- )
  duration SUSTAIN 100 */ ms
  CHANNEL0 0 ledcWriteTone drop
  duration 100 SUSTAIN - 100 */ ms
;

```

Das Wort **sustain.note** erzeugt zwei Verzögerungen. Die erste Verzögerung entspricht der Dauer der Notenpflege. Die zweite Verzögerung entspricht einer Stille-

Aufrechterhaltungsverzögerung. Die Summe dieser beiden Verzögerungen entspricht immer der in der Dauer definierten Verzögerung.

Musiknoten erstellen

Wir kommen zum interessantesten Teil und definieren die Noten anhand ihres Namens:

```
: create-note
  \ compile position in octave
  create      ( position -- )
  ,
  \ get note frequency in current octave
  does>
    @ 1- get.note
    CHANNEL0 swap ledcWriteTone drop
    sustain.note
;

\ notes in english notation
1 create-note C
2 create-note C#
3 create-note D
4 create-note D#
5 create-note E
6 create-note F
7 create-note F#
8 create-note G
9 create-note G#
10 create-note A
11 create-note A#
12 create-note B

\ notes in french notation
1 create-note DO
2 create-note DO#
3 create-note RE
4 create-note RE#
5 create-note MI
6 create-note FA
7 create-note FA#
8 create-note SOL
9 create-note SOL#
10 create-note LA
11 create-note LA#
12 create-note SI
```

```

: SIL ( -- )
  CHANNEL0 0 ledcWriteTone drop
  duration ms
;

forth definitions

```

Zusätzlich zu den zwölf Noten, von **DO** bis **SI** , definieren wir unser **SIL** , das eine Stille ist.

Punktetest

Wir testen alle Noten Tonleiter für Tonleiter:

```

forth definitions
: music-scale ( -- )
  C C# D D# E F F# G G# A A# B
;

initTones
forth also music also
.|
80 to SUSTAIN
OCT1 music-scale
OCT2 music-scale
OCT3 music-scale
OCT4 music-scale
OCT5 music-scale
OCT6 music-scale

```

Wenn alles gut geht, müssen wir alle Noten in Halbtonschritten von Oktave 1 bis zur höchsten Oktave, hier 6, entfalten. Wir definieren keine zusätzliche Oktave. Es ist machbar. Die abgegebenen Geräusche erreichen jedoch einen Grenzbereich, um hörbar zu sein.

Der Flug der Hummel

Dies ist ein erster Test zur Transponierung einer Partitur. Dazu sammeln wir ein besonders schwieriges Musikstück, **THE FLIGHT OF THE BOURDON** von **Rimsky KORSAKOV** . Hier ist der erste Takt der ersten Zeile:

So kodieren wir diesen ersten Takt in französischer Notation:

So kodieren wir diesen ersten Takt in französischer Notation:

```
OCT5 MI RE# RE DO#      RE DO# DO  OCT4 SI
```

Oder in englischer Schreibweise:

```
OCT5 E D# D C#      D C# C OCT4 B
```

Hier ist der Code für die erste Zeile dieser Partition:

```
: 1stLine ( -- )
  .|" ( duration of a quarter black note )
  OCT5 MI RE# RE DO#      RE DO# DO  OCT4 SI
  OCT5 DO  OCT4 SI LA# LA      SOL# SOL FA# FA
  MI RE# RE DO#      RE DO# DO OCT3 SI
;
```

Ich entschuldige mich, wenn mir bei der Übersetzung der Partitur Fehler unterlaufen sind. In diesem Stadium ist es einfach, diese Musiklinie zu testen:

```
: flightBumbleBee ( -- )
  initTones
  1stLine
;
flightBumbleBee
```

Wir codieren zwei weitere Zeilen:

```
: 2ndLine ( -- )
  .|" ( duration of a quarter black note )
  OCT4 DO OCT3 SI FA# FA      SOL# SOL FA# FA
  MI RE# RE DO#      RE DO DO# OCT2 SI OCT3
  MI RE# RE DO#      RE DO DO OCT2 SI OCT3
  MI RE# RE DO#      DO FA FA RE#
;

: 3rdLine ( -- )
  .|" ( duration of a quarter black note )
  MI RE# RE DO#      DO DO# RE RE#
  MI RE# RE DO#      DO FA FA RE#
  MI RE# RE DO#      DO DO# RE RE#
  MI RE# RE DO#      RE DO DO# OCT2 SI OCT3
;
```



Figure 32: erster Takt – Der Hummelflug –
Rimsky KORSAKOV

```
: flightBumbleBee ( -- )  
  initTones  
  1stLine  
  2ndLine  
  3rdLine  
  ;  
flightBumbleBee
```

Wir lassen Sie die anderen drei Zeilen der Partitur kodieren.

Programm im XTENSA-Assembler

Präambel

Für diejenigen, die mit Assemblersprache nicht vertraut sind: Es handelt sich um die unterste Ebene der Programmierung. Im Assembler sprechen wir den Prozessor direkt an.

Es ist auch eine schwierige Sprache, nicht sehr gut lesbar. Andererseits ist die Leistung außergewöhnlich.

Wir programmieren in Assembler:

- wenn es keine andere Lösung gibt, um auf bestimmte Funktionen eines Prozessors zuzugreifen;
- um bestimmte Teile des Programms schneller zu machen. Von einem Assembler generierter Code ist am schnellsten!
- zum Spass. Die Assembler-Programmierung ist eine intellektuelle Herausforderung;
- weil keine entwickelte Sprache alles kann. Manchmal können Sie in Assembler Funktionen programmieren, die zu komplex sind, um sie in einer anderen Sprache zu schreiben.

Als Beispiel ist hier der Huffman-Dekodierungscode, der im XTENSA-Assembler ausgeführt wurde:

```
/* input in t0, value out in t1, length out in t2 */
    srl t1, t0, 6
    li t3, 3
    beq t3, t4, 2f
    li t2, 2
    andi t3, t0, 0x20
    beq t3, r0, 1f
    li t2, 3
    andi t3, t0, 0x10
    beq t3, r0, 1f
    li t2, 4
    andi t3, t0, 0x08
    beq t3, r0, 1f
    li t2, 5
    andi t3, t0, 0x04
    beq t3, r0, 1f
    li t2, 6
    andi t3, t0, 0x02
```

```

    beq t3, r0, 1f
    li t2, 7
    andi t3, t0, 0x01
    beq t3, r0, 1f
    li t2, 8
    b 2f
    li t1, 9
1: /* length = value */
    move t1, t2
2: /* done */

```

Seit Version 7.0.7.4 enthält ESP32forth einen vollständigen XTENSA-Assembler. Dieser Assembler verwendet die Infix-Notation:

```

\ im herkömmlichen Assembler:
\   andi t3, t0, 0x01

\ im XTENSA-Assembler mit ESP32forth:
a3 a0 $01 ANDI,

```

ESP32forth ist die **allererste höhere Programmiersprache** für ESP32, die einen XTENSA-Assembler integriert.

Mit dieser Funktion kann der Programmierer seine Assembly-Makros definieren.

Jedes in der XTENSA-Assemblersprache von ESP32forth geschriebene Wort kann sofort in jeder Definition in der FORTH-Sprache verwendet werden.

Kompilieren Sie den XTENSA-Assembler

Seit Version 7.0.7.15 bietet ESP32forth den XTENSA-Assembler als Option an. Um diese Option zu kompilieren:

- **optionalen** Ordner in dem Ordner, in dem Sie die ZIP-Datei der ESP32forth-Version entpackt haben
- Datei „**assemblers.h**“ in den Stammordner, der die Datei „**ESP32forth.ino**“ enthält
- Führen Sie die ARDUINO-IDE aus, kompilieren Sie **ESP32forth.ino** und laden Sie es auf die ESP32-Karte hoch

Wenn alles gut geklappt hat, greifen Sie auf den XTENSA-Assembler zu, indem Sie einmal Folgendes eingeben:

```
xtensa-assembler
```

So überprüfen Sie die korrekte Verfügbarkeit des XTENSA-Befehlssatzes:

Programmierung in Assembler

Um das zuvor Gesagte klar zu verstehen, hier eine beispielhafte Definition von Brad NELSON:

```
\ Beispiel vorgeschlagen von Brad NELSON
code my2*
  a1 32 ENTRY,
  a8 a2 0 L32I.N,
  a8 a8 1 SLLI,
  a8 a2 0 S32I.N,
  RETW.N,
end-code
```

Wir haben gerade das Wort **my2*** definiert, das genau die gleiche Wirkung hat wie das Wort **2***. Die Assemblierung des Codes erfolgt sofort. Wir können daher unsere Definition von **my2*** vom Terminal aus testen:

```
--> 3 my2*
ok
6 --> 21 my2*
ok
6 42 -->
```

Diese Möglichkeit, einen zusammengestellten Code sofort zu testen, ermöglicht es, ihn vor Ort zu testen. Wenn wir etwas komplexen Code schreiben müssen, ist es einfach, ihn in Fragmente zu schneiden und jeden Teil dieses Codes mit dem ESP32forth-Interpreter zu testen.

Der XTENSA-Assemblercode wird nach dem zu definierenden Wort platziert. Es ist die Codesequenz **my2***, die das Wort **my2*** erzeugt.

Die folgenden Zeilen enthalten den XTENSA-Assemblercode. Die Assemblydefinition endet mit der Ausführung von **end-code**.

Zusammenfassung der grundlegenden Anweisungen

Liste der grundlegenden Anweisungen, die in allen Versionen der Xtensa-Architektur enthalten sind. Der Rest dieses Abschnitts bietet einen Überblick über die grundlegenden Anweisungen.

Laden / Laden

```
L8UI, L16SI, L16UI, L32I, L32R,
```

Lagern / Lagern

S8I, S16I, S32I,

Speicherbestellung

MEMW, EXTW,

Sprünge

CALL0, CALLX0, RET, J, JX,

Bedingte Verzweigung

BALL, BNALL, BANY, BNONE, BBC, BBCI, BBS, BBSI, BEQ, BEQI, BEQZ, BNE, BNEI, BNEZ, BGE, BGEI, BGEU, BGEUI, BGEZ, BLT, BLTI, BLTU, BLTUI, BLTZ,

Schicht

MOVI, MOVEQZ, MOVGEZ, MOVLTZ, MOVNEZ,

Arithmetik

ADDMI, ADD, ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, NEG, ABS,

Binäre Logik

UND, ODER,

Lücke

EXTUI, SRLI, SRAI, SLLI, SRC, SLL, SRL, SRA, SSL, SSR, SSAI, SSA8B, SSA8L,

Prozessorsteuerung

RSR, WSR, XSR, RUR, WUR, ISYNC, RSYNC, ESYNC, DSYNC, NOP,

Ein Bonus-Disassembler

Ein Assembler ist sehr gut. Die einfache Integration von Code in FORTH-Definitionen ist wunderbar. Aber einen XTENSA-Disassembler zu haben, ist königlich!

Nehmen wir die zuvor zusammengestellte Definition von **my2*** . Die Demontage ist einfach:

```
' my2* cell+ @ 20 disasm
\ zeigt an:
\ 1074338656 -- a1 32 ENTRY,          -- 004136
\ 1074338659 -- a8 a2 0 L32I.N,       -- 0288
\ 1074338661 -- a8 a8 1 SLLI,         -- 1188F0
\ 1074338664 -- a8 a2 0 S32I.N,       -- 0289
\ 1074338666 -- RETW.N,              -- F01D
\ 1074338668 -- .....
```

Auf den Code unseres Wortes **my2*** kann nur indirekt zugegriffen werden, dessen Adresse im Parameterfeld angegeben wird.

In jeder Zeile wird Folgendes angezeigt:

- die Adresse des zusammengestellten Codes
- der zerlegte Code an dieser Adresse auf 2 oder 3 Bytes
- der Hexadezimalcode, der dem zerlegten Code entspricht

Der Disassembler kann auch auf den gesamten bereits kompilierten oder assemblierten Code einwirken. Sehen wir uns den Code für Wort **2* an** :

```
' 2* @ 20 disasm
\ display:
\ 1074606252 -- a12 a3 0 L32I.N,          -- 03C8
\ 1074606254 -- a5 a5 1 SLLI,             -- 1155F0
\ 1074606257 -- a15 a12 0 L32I.N,         -- 0CF8
\ 1074606259 -- a3 a3 4 ADDI.N,           -- 334B
\ 1074606261 -- 1074597318 J,             -- F74346
```

Die Demontage zeigt an, dass der Code zu einem bedingungslosen Sprung **1074597318 J** führt . Die Demontage kann problemlos an dieser neuen Adresse fortgesetzt werden:

```
1074597318 20 disasm
\ display:
\ 1074597318 -- a15 JX,                    -- 000FA0
\ 1074597321 -- a10 64672 L32R,            -- FCA0A1
\ 1074597324 -- a5 a7 1 S32I,             -- 016752
\ 1074597327 -- 1074633168 CALL8,         -- 08C025
\ 1074597330 -- a12 a3 0 L32I,            -- 0023C2
\ 1074597333 -- a2 a7 4 ADDI,             -- 04C722
\ 1074597336 .....
```

Erste Schritte im XTENSA-Assembler

Präambel

Der Assembler-Code ist nicht in eine andere Umgebung übertragbar oder erfordert enorme Anstrengungen, den Assembler-Code zu verstehen und anzupassen.

Eine FORTH-Version ist nicht vollständig, wenn sie keinen Assembler hat.

Eine Assembler-Programmierung ist nicht erforderlich. Aber in manchen Fällen kann das Erstellen einer Definition in Assembler viel einfacher sein als eine Version in C-Sprache oder in reiner FORTH-Sprache.

Aber vor allem bietet eine in Assembler geschriebene Definition eine unübertroffene Ausführungsgeschwindigkeit.

Wir werden anhand sehr einfacher und sehr kurzer Beispiele sehen, wie man die Programmierung von FORTH-Definitionen beherrscht, die im Xtensa-Assembler geschrieben wurden.

Aufrufen des Xtensa-Assemblers

xtensa-assembler aufzurufen . Dieses Wort lädt den Inhalt des **xtensa**- Vokabulars . Dieses Wort darf nur einmal beim Starten von ESP32forth und vor jeder Definition eines Wortes im xtensa-Code aufgerufen werden:

```
forth
DEFINED? code invert [IF] xtensa-assembler [THEN]
```

Wenn wir nun **order** eingeben , zeigt ESP32forth Folgendes an:

```
xtensa >> asm >> FORTH
```

Es ist diese Reihenfolge der Vokabeln, die beachtet werden muss, wenn wir ein neues Wort in der Xtensa-Assembly mithilfe der Definitionswörter **code** und **end-code** definieren möchten .

Xtensa und der FORTH-Stack

Der Xtensa-Prozessor verfügt über 16 Register, a0 bis a15. In Wirklichkeit gibt es 64 Register, aber wir können von diesen 64 Registern nur auf ein Fenster mit 16 Registern zugreifen, das im Intervall 00..15 zugänglich ist.

Register a2 enthält den FORTH-Stapelzeiger.

Jedes Mal, wenn ein Wert gestapelt wird, wird der Stapelzeiger um vier Einheiten erhöht:

```
SP@. \ zeigt 1073632236 an
1
```

```

SP@. \display 1073632240
2
SP@. \display 1073632244
drop drop
SP@. \1073632236

```

So könnten wir dieses Wort **SP@** im Xtensa-Assembler umschreiben:

```

\ SP-Stack-Zeiger abrufen - entspricht SP@
code mySP@
    a1 32      ENTRY,
    a8 a2      MOV.N, \ kopiere den Inhalt von a2 nach a8
    a2 a2 4    ADDI,  \ a2 erhöhen
    a8 a2 0    S32I.N, \ kopiere a8 in die Adresse, auf die a2+0 zeigt
                RETW.N,
end-code

```

Testen wir dieses neue Wort **mySP@** :

```

mySP@ .
\ zeigt 1073632240 an
SP@ .
\ zeigt 1073632240 an

```

Schreiben einer Xtensa-Makroanweisung

In unserer Definition des Wortes **mySP@** **erhöht** die Sequenz **a2 a2 4 ADDI**, den Stapelzeiger um vier Einheiten. Ohne dieses Inkrement ist es unmöglich, einen Wert an die Spitze des FORTH-Stacks zurückzugeben. Mit FORTH schreiben wir ein Makro, das diesen Vorgang automatisiert.

Zunächst erweitern wir das **ASM**-Vokabular:

```

asm definitions

: macro:
    :
;

```

Unsere **macro:** definition ist mit **:** überflüssig, hat aber den Vorteil, dass der FORTH-Code dann etwas lesbarer wird, wenn wir eine Makroanweisung definieren, die das **xtensa**-Vokabular erweitert :

```

xtensa definitions

macro: sp++,
    a2 a2 4    ADDI,
;

```

sp++ Makroanweisung können wir die Definition von **mySP@** neu schreiben :

```

forth definitions

```

```
asm xtensa
```

```
\ Stack Pointer SP abrufen – Äquivalent für SP@
```

```
code mySP@
```

```
    a1 32      ENTRY,
```

```
    a8 a2      MOV.N, \ Inhalt von a2 in a8 kopieren
```

```
    sp++,
```

```
    a8 a2 0     S32I.N, \ kopiere a8 in die Adresse, auf die a2+0 zeigt  
                    RETW.N,
```

```
end-code
```

Es ist durchaus möglich, ein Makro in ein anderes zu integrieren. Im **mySP@** -Code kopiert die Codezeile **a8 a2 0 S32I.N** den Inhalt des Registers a8 an die Adresse, auf die a2 zeigt. Hier ist diese neue Makroanweisung:

```
xtensa definitions
```

```
\ Erhöhen Sie den Stapelzeiger und speichern Sie den Inhalt
```

```
\ von ar in der Adresse, auf die der Stapelzeiger zeigt
```

```
Makro: arPUSH, { ar -- }
```

```
    sp++,
```

```
    ar a2 0 S32I.N,
```

```
;
```

Diese Makroanweisung verwendet eine lokale Variable **ar** . Wir hätten darauf verzichten können, aber der Vorteil dieser Variable besteht darin, dass der Makrocode besser lesbar ist.

mySP@ -Code mit dieser Makroanweisung.

```
forth definitions
```

```
asm xtensa
```

```
\ Stack Pointer SP abrufen – entspricht SP@
```

```
code mySP@3
```

```
    a1 32      ENTRY,
```

```
    a8 a2      MOV.N,
```

```
    a8 arPUSH,
```

```
                    RETW.N,
```

```
end-code
```

Vervollständigen wir unsere Liste der Makroanweisungen:

```
xtensa definitions
```

```
\ Stapelzeiger dekrementieren
```

```
macro: sp--, ( -- )
```

```
    a2 a2 -4     ADDI,
```

```
;
```

```

\ Speichere den Inhalt der Adresse, auf die der Stapelzeiger
\ zeigt, in ar und dekrementiere den Stapelzeiger
macro: arPOP,    { ar -- }
    ar a2 0      L32I.N,
    sp--,
;

```

Mit diesen neuen Makros schreiben wir **swap** neu :

```

forth definitions
asm xtensa

code mySWAP
    a1 32      ENTRY,
    a9 arPOP,
    a8 arPOP,
    a9 arPUSH,
    a8 arPUSH,
                RETW.N,
end-code

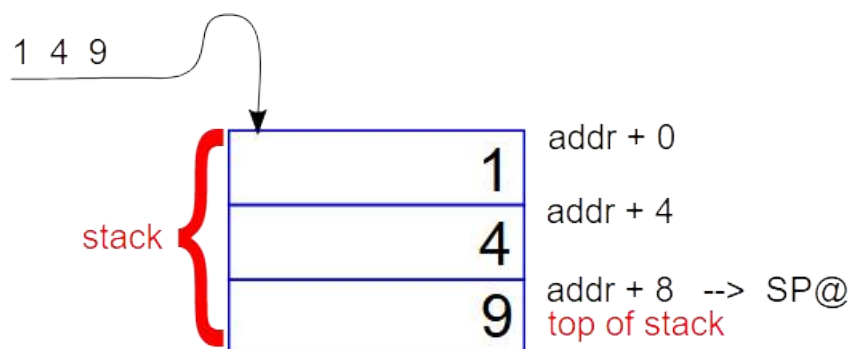
17 24 mySWAP

```

Verwalten des FORTH-Stacks im Xtensa-Assembler

Auf die Position des FORTH-Stapelzeigers kann mit **SP@** **zugegriffen werden** . Durch das Stapeln einer 32-Bit-Ganzzahl (Standardgröße für ESP32forth) wird dieser Stapelzeiger um vier Einheiten erhöht.

Wir haben besprochen, wie das Inkrementieren oder Dekrementieren dieses Stapelzeigers mithilfe der Makroanweisungen **sp++** und **sp--** verwaltet wird. Diese Makroanweisungen bewegen den Stapelzeiger um vier Einheiten.



Hier haben wir drei Werte gestapelt, **1** **4** und **9** . Bei jedem Stapeln wird der Stapelzeiger automatisch erhöht. Im Xtensa-Assembler befindet sich der Stapelzeiger im Register a2. Wir haben gesehen, dass wir den Inhalt dieses Registers mit den Makrobefehlen **sp++** und

sp-- manipulieren können . Die Manipulation dieses Registers hat eine direkte Auswirkung auf den von ESP32forth verwalteten Stapelzeiger.

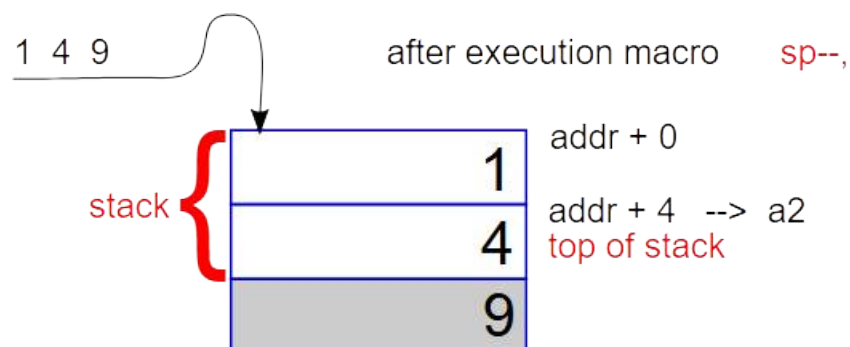
So haben wir das Wort **+** in Assembler umgeschrieben , indem wir den Stapelzeiger durch unsere **arPOP** und **arPUSH** -Makroanweisungen manipuliert haben :

```
code my+
  a1 32      ENTRY,
  a7 arPOP,
  a8 arPOP,
  a7 a8 a9    ADD,
  a9 arPUSH,
              RETW.N,
end-code
```

Es gibt eine andere Möglichkeit, Daten vom Stapel mithilfe der **L32I.N**- Anweisung abzurufen . Diese Anweisung verwendet einen unmittelbaren Index:

```
code my+
  a1 32 ENTRY,
    sp--,
  a7 a2 0     L32I.N,
  a8 a2 1     L32I.N,
  a7 a8 a9    ADD,
  a9 a2 0     S32I.N,
  RETW.N,
end-code
```

Bevor wir die Daten vom Stapel abrufen, dekrementieren wir den Stapelzeiger mit unserer Makroanweisung **sp--** ,. Dadurch bewegt sich der Zeiger um 4 Einheiten zurück.



Aber nur weil sich der Zeiger zurückbewegt, bedeutet das nicht, dass die zuvor gestapelten Daten verschwinden. Sehen wir uns diese Codezeile im Detail an:

```
a7 a2 0     L32I.N,
```

Dieser Befehl lädt Register a7 mit dem Inhalt der Adresse, auf die (a2)+n*4 zeigt. Hier ist n 0. Diese Anweisung fügt den Wert 4 in unser Register a7 ein.

Sehen wir uns die folgende Zeile an:

```
a8 a2 1      L32I.N,
```

Register a8 wird mit dem Inhalt geladen, auf den (a2)+1*4 zeigt. Diese Anweisung fügt den Wert 9 in unser Register a8 ein.

```
a9 a2 0      S32I.N,
```

Hier wird der Inhalt des Registers a9 an der Adresse gespeichert, auf die (a2)+1*0 zeigt. Tatsächlich überschreiben wir den Wert 4 mit dem Ergebnis der Addition der Inhalte der Register a7 und a8.

Sehen wir uns ein letztes Beispiel an, in dem wir zwei Parameter verarbeiten und zwei davon auf den Datenstapel ausgeben. In diesem Beispiel schreiben wir das Wort **/MOD** um :

```
code my/MOD ( n1 n2 -- rem quot )
  a1 32      ENTRY,
  a7 arPOP,      \ divisor in a7
  a8 arPOP,      \ Wert, der durch a8 geteilt werden soll
  a7 a8 a9      REMS, \ a9 = a8 MOD a7
  a9 arPUSH,
  a7 a8 a9      QUOS, \ a9 = a8 / a7
  a9 arPUSH,
              RETW.N,
end-code

5 2 my/MOD . . \ Anzeige 2 1
-5 -2 my/MOD . . \ Anzeige 2 -1
```

Im Wort **my/MOD** verwenden wir dieselben Daten n1 und n2, die jeweils in den Registern a8 und a7 platziert sind. Es sind dann die **REMS**- und **QUOT**- Anweisungen , die die Berechnung der von **my/MOD** zurückgegebenen Ergebnisse ermöglichen .

Effizienz der im XTENSA-Assembler geschriebenen Wörter

In unserem allerletzten Beispiel oben haben wir das Wort **/MOD** umgeschrieben . Die zu stellende Frage lautet: „Ist das Wort **my/MOD** wirklich schneller in der Ausführung als das Wort **/MOD** ?“

Dazu verwenden wir das Wort **measure** , dessen FORTH-Code im Kapitel „ *Messung der Ausführungszeit eines FORTH-Worts*“ erläutert wird .

```
: test1
  1000000 for
    5 2 /MOD
    drop drop
  next
;

: test2
```

```
1000000 for
  5 2 my/MOD
  drop drop
next
;

measure: test1 \ Anzeige: Ausführungszeit: 0,856 Sekunden.
measure: test2 \ Anzeige: Ausführungszeit: 0,600 Sek.
```

Die Wörter **test1** und **test2** sind ähnlich, außer dass **test2 my/MOD** ausführt . Über 1 Million Iterationen beträgt die Zeitersparnis 0,144 Sekunden. Es ist nicht viel, aber das Verhältnis scheint dennoch signifikant zu sein.

Umgekehrt sehen wir, dass die Ausführungszeit der FORTH-Sprache sehr schnell ist.

Schleifen und Verbindungen im XTENSA-Assembler

Die LOOP-Anweisung im XTENSA-Assembler

Die LOOP-Schleife im XTENSA-Assembler verwendet die **LOOP- Anweisung** , um den Prozessor anzuweisen, einen Befehlsblock zu wiederholen, bis ein bestimmter Zähler Null erreicht. Die Schleife wird initialisiert, indem der Anfangswert des Zählers festgelegt und dann die **LOOP**-Anweisung mit diesem Wert als Argument ausgeführt wird. Bei jeder Iteration der Schleife wird der Zähler um 1 verringert, bis er Null erreicht. An diesem Punkt stoppt die Schleife. Im klassischen Assembler:

```
; Initialisierung des Zählers auf 10
MOVI a0, 10

; Beginn der LOOP-Schleife
loop:
; Anweisung(en) zur Wiederholung
...
; Dekrementieren Sie den Zähler und testen Sie die Stoppbedingung
LOOP a0, loop
```

Hier wiederholt die LOOP-Schleife die Anweisungen zwischen **loop:** und **LOOP a0**, führt eine zehnmale Schleife durch und dekrementiert den Zähler a0 bei jeder Iteration. Wenn der Zähler Null erreicht, stoppt die Schleife.

Wenn der XTENSA-Prozessor auf den **LOOP**- Befehl trifft , initialisiert er drei Sonderregister:

- **LCOUNT** $\leftarrow$ **AR[s]** – 1

Das Sonderregister LCOUNT wird mit dem Inhalt des Registers initialisiert, hier a0 in unserem Beispiel, dekrementiert um eine Einheit. Wenn der Zähler den Wert 0 erreicht, schließt der LOOP-Befehl die Schleife ab;

- **LBEG** $\leftarrow$ **PC** + 3

Das LBEG-Sonderregister enthält die Startadresse der aktuell ausgeführten LOOP-Schleife. Diese Adresse wird durch den LOOP-Befehl definiert.

- **LEND** $\leftarrow$ **PC** + (**024** | **imm8**) + 4 Das LEND-Sonderregister enthält die Endadresse der aktuell ausgeführten LOOP-Schleife. Diese Adresse wird durch den LOOP-Befehl definiert.

Im XTENSA-Assembler lässt die LOOP-Anweisung zwei Parameter zu:

```
LOOP as, label
```

Die Bezeichnung entspricht einem 8-Bit-Offset nach dem **LOOP**- Befehl . Sie können einen Code mit einer Länge von mehr als 256 Byte nicht wiederholen.

Hier ist ein zerlegter XTENSA-Code mithilfe einer LOOP-Schleife:

.data:00000000	004136	entry a1, 32
.data:00000003	01a082	movi a8, 1
.data:00000006	04a092	movi a9, 4
.data:00000009	048976	loop a9, 0x00000011
.data:0000000c	04c222	addi a2, a2, 4
.data:0000000f	0289	s32i.n a8, a2, 0
.data:00000011	f01d	retw.n



Der Disassembler gibt eine Filialadresse an. In Wirklichkeit enthält der zusammengesetzte Code diesen durch das Etikett angegebenen Offset nur in Form eines positiven 8-Bit-Werts.

Verwalten Sie eine Schleife im XTENSA-Assembler mit ESP32forth

Die FORTH-Sprache kann einen Vorwärtsverweis nicht auflösen. Sofern Sie nicht herumfummeln, ist es schwierig, die **LOOP**-Anweisung zu verwenden, ohne einen Trick zu finden.

Definieren von Makroanweisungen für die Schleifenverwaltung

LOOP-Anweisung einfacher verwenden zu können, definieren wir zwei Makroanweisungen, **For** und **Next**, von denen hier der Code in der FORTH-Sprache ist:

```
: For, { as n -- }
  as n MOVI,
  as 0 LOOP,
  chere 1- to LOOP_OFFSET
;

: Next, ( -- )
  chere LOOP_OFFSET - 2 -
  LOOP_OFFSET [ internals ] ca! [ asm xtensa ]
;
```

Der **For**-Makrobefehl akzeptiert die gleichen Parameter wie der **LOOP**-Befehl :

```
as n For,
```

- ebenso wie das Register, das die Anzahl der Iterationen der Schleife enthält;
- n ist die Anzahl der Iterationen.

Mit den Makros „For“ und „Next“

Wir definieren ein **myLOOP**- Wort , um die **LOOP**-Anweisung über die Makroanweisungen **For**, **Next**, zu testen :

```
code myLOOP ( n -- n' )
  a1 32          ENTRY,
  a8 1           MOVI,
  a9 4           For,           \ LOOP start here
    a8 a8 1      ADDI,
    a8           arPUSH,       \ push result on stack
  Next,
                                RETW.N,
end-code
```

Register a8 wird mit dem Wert 1 initialisiert. Die **For**, **Next** -Schleife erhöht den Inhalt von a8 und stapelt seinen Inhalt. Folgendes bietet das Ausführen von **MyLOOP** :

```
ok
--> myLoop
ok
2 3 4 5 -->
```

ACHTUNG : Wenn die Anzahl der Iterationen Null ist, erhöht sich die Anzahl der Iterationen auf 232.

Verbindungsanweisungen im XTENSA-Assembler

Der XTENSA-Assembler im **xtensa**- Vokabular verfügt über mehrere Arten von Verzweigungsanweisungen:

- Verbindungen mit booleschen Flags, die im Sonderregister **BR** definiert sind : **BF**, **BT**,
- Die Verbindungen führen Tests an den Registern durch: **BALL**, **BANY**, **BBC**, **BBS**, **BEQ**, **BGE**, **BLT**, **BNE**, **BNONE**,

Es ist diese zweite Kategorie von Verbindungen, die uns interessiert.

Verzweigungsmakros definieren

Der ESP32forth xtensa Assembler verfügt nicht über einen Label-Management-Mechanismus wie ein klassischer Assembler. Um effektiv zu sein, muss die Etikettenverwaltung in mehreren Schritten funktionieren, wenn Vorwärtsverzweigungen aufgelöst werden müssen. Dies ist nicht kompatibel mit der Funktionsweise der FORTH-Sprache, die in einem einzigen Durchgang kompiliert oder assembliert.

Wir überwinden diese Schwierigkeit, indem wir zwei Makroanweisungen definieren, **If**, und **Then**, die diese Vorwärtsverbindungen verwalten :

```
: If, ( -- BRANCH_OFFSET )
    chere 1-
;

: Then, { BRANCH_OFFSET -- }
    chere BRANCH_OFFSET - 2 -
    BRANCH_OFFSET [ internals ] ca! [ asm xtensa ]
;
```

Der Makroanweisung muss eine andere Makroanweisung vorausgehen. Für unseren ersten Test definieren wir das Makro **<**, das einen ungelösten Zweig zusammenstellt :

```
: <, ( as at -- )
    0 BGE,
;
```

Verwendung dieser Makros in unserem ersten Beispiel :

```
code my< ( n1 n2 -- f1 )    \ f1=1 if n1 < n2
    a1 32                ENTRY,
    a8                arPOP,                \ a8 = n2
    a9                arPOP,                \ a9 = n1
    a7 0                MOVI,                \ a7 = 1
    a8 a9 <, If,
        a7 1            MOVI,                \ a7 = 0
    Then,
    a7                arPUSH,
                        RETW.N,
end-code
```

Syntax verzweigter Makroanweisungen

In unserem Beispiel haben wir die Makroanweisung **<**, verwendet, die mit der BGE-Verzweigungsanweisung verbunden ist und deren Bedeutung ist: „Verzweigen, wenn größer oder gleich“. Normalerweise würde es mit „>=“ übersetzt werden. Warum wurde „<“ verwendet?

Dies liegt daran, dass unsere Makroanweisung **If, Then** eine Logik hat, die der des auszuführenden Zweigs entgegengesetzt ist. Der in **If, Then** eingeschlossene Code wird ausgeführt, wenn die erforderliche Bedingung nicht gültig ist. Hier ist die Tabelle, die diese umgekehrte Logik zusammenfasst und die Wahl des Namens dieser Makroanweisungen erläutert, die vor **If, ... Then** , verwendet wurden:

XTENSA branch instruction			Macro
BEQ	Branch if Equal	AR[s] = AR[t]	<=>,
BGE	Branch if Greater Than or Equal	AR[s] ≥ AR[t]	<=,
BLT	Branch if Less Than	AR[s] < AR[t]	>=,
BNE	Branch if Not Equal	AR[s] ≠ AR[t]	=,

Kehren wir zu unserem **my<**-Assembly-Beispiel zurück. Folgendes ergibt die Ausführung des Wortes **my<** :

```

10 20 my< .      \ zeigt an: 1
20 20 my< .      \ zeigt an: 0
20 10 my< .      \ zeigt an: 0
-5 35 my< .      \ zeigt an: 1
-10 -3 my< .     \ zeigt an: 1
-3 -10 my< .     \ zeigt an: 0

```

Wir sehen, dass diese umgekehrte Logik respektiert wird.

Sobald diese Logik verstanden ist, können wir eine neue Makroanweisung **>=**, definieren :

```

: >=, ( as at -- )
    0 BLT,
;

```

Und testen Sie diese Makroanweisung :

```

code my>= ( n1 n2 -- f1 )    \ f1=1 if n1 < n2
    a1 32                    ENTRY,
    a8                      arPOP,          \ a8 = n2
    a9                      arPOP,          \ a9 = n1
    a7 0                     MOVI,          \ a7 = 1
    a8 a9 >=, If,
        a7 1                 MOVI,          \ a7 = 0
    Then,
    a7                      arPUSH,
                                RETW.N,
end-code

10 20 my>= .      \ zeigt an: 0
20 20 my>= .      \ zeigt an: 1
20 10 my>= .      \ zeigt an: 1
-5 35 my>= .      \ zeigt an: 0
-10 -3 my>= .     \ zeigt an: 0
-3 -10 my>= .     \ zeigt an: 1

```

Definition und Manipulation von Registern

Im technischen Dokument von ESP32 finden wir eine sehr große Menge an Registern. Mit diesen Registern können Sie alle Peripheriegeräte und GPIO-Ports auf dem ESP32-Board steuern.

SAR ADC2 control registers			
SENS_SAR_READ_CTRL2_REG	SAR ADC2 data and sampling control	0x3FF48890	R/W
SENS_SAR_MEAS_START2_REG	SAR ADC2 conversion control and status	0x3FF48894	RO
ULP coprocessor configuration register			
SENS_ULP_CP_SLEEP_CYC0_REG	Sleep cycles for ULP coprocessor	0x3FF48818	R/W
Pad attenuation configuration registers			
SENS_SAR_ATTEN1_REG	2-bit attenuation for each pad	0x3FF48834	R/W
SENS_SAR_ATTEN2_REG	2-bit attenuation for each pad	0x3FF48838	R/W
DAC control registers			
SENS_SAR_DAC_CTRL1_REG	DAC control	0x3FF48898	R/W
SENS_SAR_DAC_CTRL2_REG	DAC output control	0x3FF4889C	R/W

Figur 33: Auszug aus dem Technischen Referenzhandbuch

Im Allgemeinen erfolgt die Manipulation dieser Register durch die von ESP32forth angebotene Anwendungsschicht. Ein direkter Zugriff ist daher nicht erforderlich.

In manchen Fällen kann es interessant sein, Register direkt zu verwalten:

- um auf Funktionen zuzugreifen, die ESP32forth nicht bietet
- um FORTH-Code schneller auszuführen

Definition von Registern

Das Definieren eines Registers ist sehr einfach:

```
$3FF48898 constant SENS_SAR_DAC_CTRL1_REG \ DAC-Steuerung
```

Der erste Nachteil der Definition eines Registers als Konstante besteht darin, dass wir beim Lesen des Quellcodes das Register nicht von anderen Konstanten unterscheiden können. Wir werden daher ein Registererstellungswort wie folgt definieren :

```
\ Definieren Sie ein Register, ähnlich einer Konstante
: defREG:
  create ( addr1 -- )
  ,
  does> ( -- regAddr )
  @
;
```

```
$3FF48898 defREG: SENS_SAR_DAC_CTRL1_REG \ DAC-Steuerung
```

Auf diese Weise wissen wir durch erneutes Lesen unseres Codes, dass es sich bei dem erstellten Wort um ein Register handelt. Der andere Vorteil besteht darin, dass wir **defREG:** modifizieren können, um sein Verhalten zu ändern: Kontrolltests hinzufügen, Parameter initialisieren usw.

Zugriff auf Registerinhalte

Beispiel, Bitwerte im Register **SENS_SAR_DAC_CTRL1_REG** :

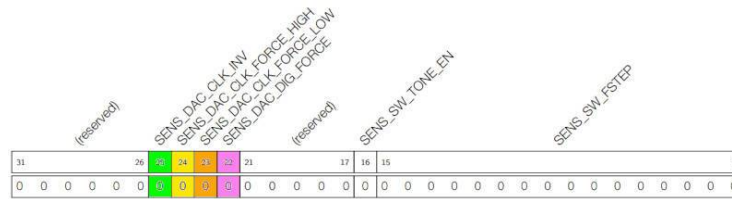


Figure 34: bits dans le registre
ENS_SAR_DAC_CTRL1_REG

Dieses Register enthält Bits oder Bitblöcke mit definierten Funktionen.

Wir erstellen zunächst ein Wort, mit dem wir den Inhalt eines Registers visualisieren können:

```
\Registrierungsinhalt anzeigen
: .reg ( reg -- )
  base @ >r
  binary
  @ <#
  4 for
    aft
      8 for
        aft # then
      next
    bl hold
  then
next
#>
cr space ." 33222222 22221111 11111100 00000000"
cr space ." 10987654 32109876 54321098 76543210"
cr type
r> base !
;
```

Mal sehen, was der Inhalt unseres **SENS_SAR_DAC_CTRL1_REG** -Registers ergibt :

```
SENS_SAR_DAC_CTRL1_REG .reg
\ Anzeige:
33222222 22221111 11111100 00000000
10987654 32109876 54321098 76543210
00000000 00000000 00000000 00000000 ok
```

In den ersten beiden Zeilen können Sie vertikal den Rang eines Bits in diesem Register ablesen, hier in Rot 25, dessen Inhalt 0 ist. Um dieses Bit auszulesen, gehen wir wie folgt vor:

```
SENS_SAR_DAC_CTRL1_REG @
```

```
1 25 lshift and
```

Um dieses Bit zu ändern und auf 1 zu setzen:

```
1 25 lshift
SENS_SAR_DAC_CTRL1_REG @
or
SENS_SAR_DAC_CTRL1_REG !
```

Schauen wir mal mit **.reg** nach :

```
SENS_SAR_DAC_CTRL1_REG .reg
\ Anzeige:
33222222 22221111 11111100 00000000
10987654 32109876 54321098 76543210
00000010 00000000 00000000 00000000 ok
```

Wenn es nur einmal ist, hilft es. Mal sehen, wie wir es effizienter machen können ...

Umgang mit Registerbits

Fahren wir mit der Änderung von Bit 25 unseres Registers **SENS_SAR_DAC_CTRL1_REG** fort . So setzen Sie Bit b25 auf 1:

```
SENS_SAR_DAC_CTRL1_REG .reg      \ display:
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 10000000 00000000 00000000 ok

registers
1 25 $02000000 SENS_SAR_DAC_CTRL1_REG m!
SENS_SAR_DAC_CTRL1_REG .reg      \ display:
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000010 00000000 00000000 00000000 ok
```

Wir verwenden das m!-Wort (val shift mask addr --), das vier Parameter akzeptiert :

- **val** ist der zu ändernde Wert, hier 1
- **shift** , die dem auf diesen Wert anzuwendenden Offset entspricht, hier 25
- **Mask** , die der logischen Maske des zu ändernden Registerteils entspricht, hier \$02000000
- **Addr** ist die Adresse des Registers, hier SENS_SAR_DAC_CTRL1_REG

Definition von Masken

Eine Maske wird verwendet, um anzuzeigen, welche Bits geändert werden können. Im vorherigen Beispiel haben wir Bit b25 geändert. In der Espressif-Dokumentation ist Bit b25

mit der Bezeichnung **SENS_DAC_CLK_INV** gekennzeichnet . Die einfachste Lösung wäre, eine Konstante wie diese zu erstellen:

```
1 25 lshift constant SENS_DAC_CLK_INV
```

Dadurch wird jedoch nicht der Wertoffset angepasst, der mit dem Wert der Binärmaske übereinstimmen muss.

Sehen wir uns eine elegantere Möglichkeit zum Definieren von Masken an:

```
: defMASK:
  create ( mask0 position -- )
    dup ,
    lshift ,
  does> ( -- position mask1 )
    dup @
    swap cell + @
;
```

```
1 25 defMASK: mSENS_DAC_CLK_INV
```

Beachten Sie nebenbei, dass dem Namen der Maske der Buchstabe „ **m** “ (für Maske) vorangestellt ist. Dies ist keineswegs verpflichtend. Wenn Sie jedoch viele Register und Masken zusammengestellt haben, können Sie sich mit dem Präfix „ **m** “ zwischen Registern und Masken zurechtfinden :

```
--> words
mSENS_SW_FSTEP mSENS_SW_TONE_EN mSENS_DAC_DIG_FORCE mSENS_DAC_CLK_FORCE_LOW
mSENS_DAC_CLK_FORCE_HIGH mSENS_DAC_CLK_INV defMask: SENS_SAR_DAC_CTRL2_REG
SENS_SAR_DAC_CTRL1_REG GPIO_ENABLE_W1TC_REG GPIO_ENABLE_W1TS_REG GPIO_ENABLE_REG
GPIO_OUT_W1TC_REG GPIO_OUT_W1TS_REG GPIO_OUT_REG DR_REG_GPIO_BASE PIN_DAC2
PIN_DAC1 CONFIG_IDF_TARGET_ESP32S3 CONFIG_IDF_TARGET_ESP32S2 .reg AdcREG:
mtst mset mclr --DAdirect SENS_DAC_CLK_INV defMASK: input$ c+$! mid$ left$
right$ 0$! $! maxlen$ string $= FORTH camera-server camera telnetd bterm
.....
```

defMASK: definierte Wort platziert den Maskenoffset auf dem Stapel, hier 25 für **mSENS_DAC_CLK_INV** und den Wert der anzuwendenden Binärmaske.

Beginnen wir mit der Änderung von Bit b25 mit dieser Maskendefinition:

```
1 mSENS_DAC_CLK_INV SENS_SAR_DAC_CTRL1_REG m!
SENS_SAR_DAC_CTRL1_REG .reg \ display:
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000010 00000000 00000000 00000000

0 mSENS_DAC_CLK_INV SENS_SAR_DAC_CTRL1_REG m!
SENS_SAR_DAC_CTRL1_REG .reg \ display:
```

```
\ 33222222 22221111 11111100 00000000
\ 10987654 32109876 54321098 76543210
\ 00000000 00000000 00000000 00000000
```

Wechsel von der C-Sprache zur FORTH-Sprache

Bei ESP32Forth gibt es zwei Lösungen zum Hinzufügen von Grundelementen zum Wörterbuch:

- Schreiben Sie den ESP32Forth-Quellcode neu, indem Sie die gewünschten Grundelemente hinzufügen.
- Schreiben Sie diese C-Sprachwörter in FORTH um

Die erste Lösung in C-Sprache ist hier geschrieben:

```
#ifndef ENABLE_DAC_SUPPORT
# define OPTIONAL_DAC_SUPPORT
# else
# include <driver/dac.h>
# include <driver/dac_common.h>
# include <soc/rtc_io_reg.h>
# include <soc/rtc_cntl_reg.h>
# include <soc/sens_reg.h>
# include <soc/rtc.h>
# define OPTIONAL_DAC_SUPPORT \
Y(dac_output_enable, n0 = dac_output_enable( (dac_channel_t) n0 ) ) \
Y(dac_output_disable, n0 = dac_output_disable( (dac_channel_t) n0 ) ) \
Y(dac_output_voltage, n0 = dac_output_voltage((dac_channel_t) n1, (gpio_num_t) n0); NIP ) \
Y(dac_cw_generator_enable, PUSH dac_cw_generator_enable() ) \
Y(dac_cw_generator_disable, PUSH dac_cw_generator_disable() ) \
Y(dac_i2s_enable, PUSH dac_i2s_enable() ) \
Y(dac_i2s_disable, PUSH dac_i2s_disable() ) \
Y(rtc_freq_div_set, REG_SET_FIELD(RTC_CNTL_CLK_CONF_REG, RTC_CNTL_CK8M_DIV_SEL, n0 ); DROP ) \
Y(dac_freq_step_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL1_REG, SENS_SW_FSTEP, n0, SENS_SW_FSTEP_S); DROP ) \
Y(dac1_scale_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_SCALE1, 0, SENS_DAC_SCALE1_S); ) \
Y(dac2_scale_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_SCALE2, 0, SENS_DAC_SCALE2_S); ) \
Y(dac1_offset_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_DC1, n0, SENS_DAC_DC1_S); DROP ) \
Y(dac2_offset_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_DC2, n0, SENS_DAC_DC2_S); DROP ) \
Y(dac1_invert_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_INV1, n0, SENS_DAC_INV1_S); DROP ) \
Y(dac2_invert_set, SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_INV2, n0, SENS_DAC_INV2_S); DROP ) \
Y(dac1_cosine_enable, SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN1_M); ) \
Y(dac2_cosine_enable, SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN2_M); ) \
Y(dacWrite, dacWrite(n1, n0); DROPn(2))
#endif
```

Die zweite Lösung besteht darin, sich den Quellcode einer in der Sprache C geschriebenen Datei anzusehen und zu verstehen, wie Register in dieser Sprache manipuliert werden.

Auszug aus der Datei **dac-cosine.c** :

```
/*
 * Enable cosine waveform generator on a DAC channel
 */
void dac_cosine_enable(dac_channel_t channel)
{
    // Enable tone generator common to both channels
    SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL1_REG, SENS_SW_TONE_EN);
```

```

switch(channel) {
    case DAC_CHANNEL_1:
        // Enable / connect tone generator on / to this channel
        SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN1_M);
        // Invert MSB, otherwise part of waveform will have inverted
        SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_INV1, 2, SENS_DAC_INV1_S);
        break;
    case DAC_CHANNEL_2:
        SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN2_M);
        SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_INV2, 2, SENS_DAC_INV2_S);
        break;
    default :
        printf("Channel %d\n", channel);
}
}

```

Eine der häufig vorkommenden C-Funktionen ist **SET_PERI_REG_MASK** . Diese Funktion setzt die durch eine Maske in einem Register bezeichneten Bits auf 1. Beispiel:

```
SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN1_M);
```

Die C-Funktion, die durch eine Maske in einem Register bezeichnete Bits auf 0 setzt, ist **CLEAR_PERI_REG_MASK** .

Wir werden daran interessiert sein, wie wir

dac_cosine_enable(dac_channel_tchannel) in der FORTH-Sprache umschreiben werden. Wir sehen, dass das Register `SENS_SAR_DAC_CTRL2_REG` erwähnt wird. Wir werden dieses Register definieren:

```
$3FF4889c defREG: SENS_SAR_DAC_CTRL2_REG \ DAC output control
```

SENS_SAR_DAC_CTRL2_REG- Register sind die beiden Bits, die uns interessieren, b24 und b25. Definieren wir die entsprechenden Masken:

```

1 24 defMASK: mSENS_DAC_CW_EN1 \ selects CW generator as source for PDAC1
1 25 defMASK: mSENS_DAC_CW_EN2 \ selects CW generator as source for PDAC2

```

Bei der „Bare-Metal“-Programmierung, die direkt auf die ESP32-Register einwirkt, müssen in FORTH nicht alle Register und Registermasken definiert werden, wie dies bei der C-Sprache der Fall ist. Beschränken Sie sich auf die für Ihre Anwendung wesentlichen Register und Masken.

Es wird empfohlen, die Namen der Register und Registermasken zu verwenden, wie sie in der Espressif-Dokumentation erscheinen, oder, falls dies nicht der Fall ist, die Registernamen, die in den Quellcodes der C-Sprache verwendet werden.

Die Verwaltung einiger Geräte ist sehr komplex. Die Espressif-Dokumentation geizt mit Beispielen zur direkten Verwendung von Registern.

Der Zufallszahlengenerator

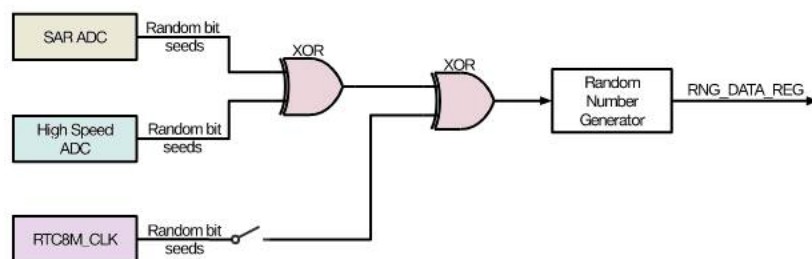
Charakteristisch

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 UL@
;

\ 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]
```

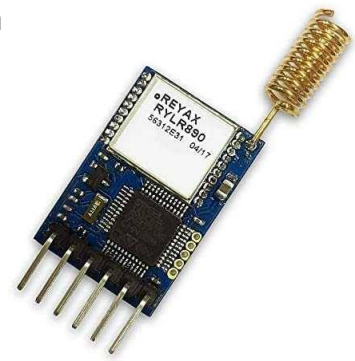
a9	arPUSH,	\ push a9 on stack
end-code	RETW.N,	

Das LoRa-Übertragungssystem

LoRa ist eine Kommunikationstechnologie, die ein Weitverkehrsnetzwerk mit geringem Stromverbrauch nutzt. Mit LoRa können Sie Geräte und Gateways drahtlos verbinden.

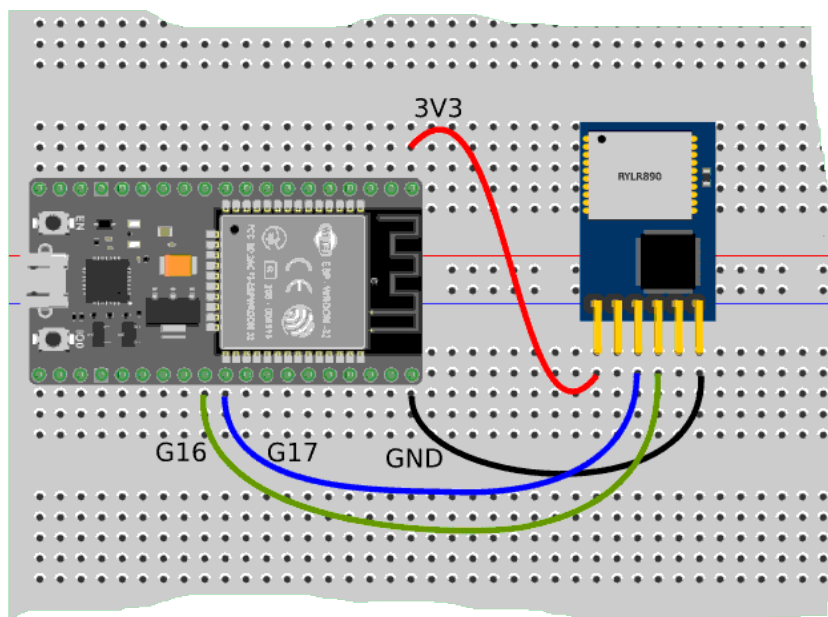
Für diesen Standard ist noch kein Abonnement erforderlich. Es bietet **Peer-to-Peer**-Kommunikation.

LoRa, WLAN und Bluetooth ergänzen sich und überschneiden sich nicht. Im Vergleich zu Wi-Fi und Bluetooth, die eine sehr kurze Reichweite bieten, profitiert LoRa von einer sehr geringen Bandbreite. Gateways oder Hubs werden kaum 1 % der Zeit von angeschlossenen Geräten genutzt. Dadurch wird die Bandbreite erheblich reduziert. Der Verkehr zwischen den Sensoren und dem Gateway ist langsam und unidirektional. LoRa ist die beste Möglichkeit, über mehrere Kilometer hinweg zu kommunizieren, mit sehr wenig Strom und auf sehr einfache Weise!



Verkabelung des LoRa REYAX LR890-Senders

Der Sender wird folgendermaßen mit der ESP32-Karte verbunden:



ACHTUNG: Überprüfen Sie die Position der Pins G16 und G17 auf Ihrer ESP32-Karte. Diese kann je nach Version Ihrer ESP32-Karte unterschiedlich sein.

Der LoRa-Sender für ESP32

Das REYAX LR890-Modul kostet rund 15 €. Es wiegt 7 Gramm.

Sein Verbrauch bei der Übertragung beträgt 43 mA (3,3 V). Beim Empfang beträgt er 16,5 mA und kann im SLEEP-Modus auf 0,5 mA sinken.

Um eine Punkt-zu-Punkt-Übertragung zu gewährleisten, sind zwei LoRa-Module erforderlich. Jedes Modul ist Sender und Empfänger.

Die ESP32-Karte kommuniziert über ihren seriellen Port mit dem LoRa-Modul. Alle Übertragungen zwischen der ESP32-Karte und dem LoRa-Sender werden über AT-Befehle verarbeitet. Beispiel:

```
AT+SEND=50,5,HALLO
```

Diese text wird von der ESP32-Karte an das LoRa-Übertragungsmodul übertragen :

- Das LoRa-Modul wechselt in den Sendemodus und sendet diese Zeichenfolge
- Unmittelbar nach der Übertragung kehrt das LoRa-Modul in den Empfangsmodus zurück
- Das entfernte LoRa-Modul empfängt die Zeichenfolge.
- Das entfernte LoRa-Modul kann diesen Empfang mit **+OK** bestätigen

Ein LoRa-Modul kann mit einem LoRaWan-Gateway kommunizieren. Im Allgemeinen handelt es sich um eine Box, die über eine Ethernet-Verbindung mit einem Router verbunden ist. Daher ist es möglich, eine Webanwendung zu haben, die mit einem oder mehreren LoRa-Modulen kommuniziert.

LoRa-Übertragungssicherheit

Ein einzelnes LoRa-Modul kann mit mehreren entfernten LoRa-Modulen kommunizieren.

Diese LoRa-Module müssen durch ihre **NETWORKID** unterschieden werden . Sender und Empfänger müssen die gleiche **NETWORKID** haben.

Anschließend erhält jedes Modul eine **ADDRESS** , standardmäßig 0. Diese Adresse liegt zwischen 0 und 65535.

Die Übertragung kann mit einem 32-stelligen **AES-Schlüssel** verschlüsselt werden. Die Sender- und Empfänger-LoRa-Module müssen über denselben **AES-Schlüssel** verfügen . Wenn ein Modul eine mit einem unbekannten **AES-Schlüssel** verschlüsselte Nachricht empfängt , ignoriert es die Nachricht.

Und schließlich wird jedem Modul eine Sendefrequenz zugewiesen. Die Sende- und Empfangsmodule müssen auf der gleichen Frequenz arbeiten.

Beispiel für die Frequenzwahl 868,5 MHz.

```
\ Wählen Sie die Frequenz 865,5 MHz für die LoRa-Übertragung
32 string AT_BAND
s" AT+BAND=868500000" AT_BAND $! \ Frequenz auf 868,5 MHz einstellen
$0a AT_BAND c+$!
$0d AT_BAND c+$! \CR LF-Code am Ende des Befehls hinzufügen
AT_BAND Serial2.write drop
```

Auf derselben Frequenz können wir eine Flotte von 65.535 LoRa-Modulen verwalten, wobei jedes Modul seine Adresse hat. Wenn wir mit der Adresse 0 senden, sprechen wir alle LoRa-Module an.

Wenn wir den AES-Verschlüsselungsschlüssel hinzufügen, gibt es Hunderttausende LoRa-Module, die in einem Umkreis von wenigen Kilometern nebeneinander existieren können!

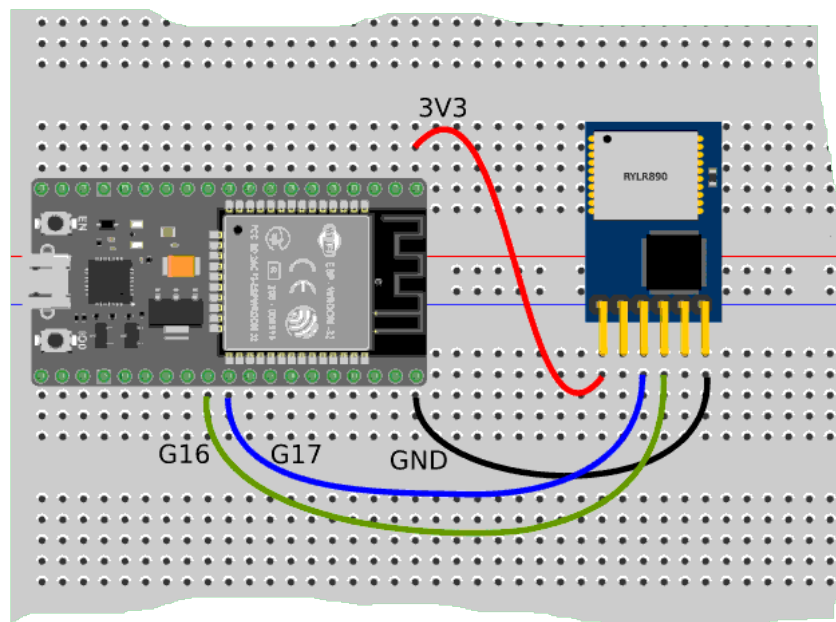
Durch Veränderung der Sendeleistung kann die Reichweite der Module erhöht werden. Wir können auch auf die Empfangsantenne einwirken. Mit einer Richtantenne erreichen Sie eine Reichweite von **20 bis 30 Kilometern** ...

Testbericht zum REYAX RYLR890 LoRa-Sender

Erforderliche Testumgebung

Um unseren REYAX RYLR890 LoRa-Sender zu testen, müssen Sie:

- Verwenden Sie Zeichenfolgenverwaltungswörter... @todo: Referenz in der Datei
- Verwenden Sie eine ESP32Forth-Version mit Zugriff auf den UART2-Port
- Verkabeln Sie den REYAX RYLR890 LoRa-Sender wie folgt:



Bereiten Sie die Kommunikation mit dem LoRa-Sender vor

Alle Programmierer, die den UART2-Port verwalten, definieren einen Speicherbereich, der als Puffer dient. Wir für unseren Teil erstellen direkt eine alphanumerische Variable:

```
128 string LoRaTX \ buffer ESP32 -> LoRa transmitter
```

Hier werden wir nur Tests zum Senden von Befehlen an den REYAX RYLR890 LoRa-Sender durchführen und sehen, wie wir wiederherstellen können, was dieser selbe Sender zurücksendet. Für den Empfang benötigen wir daher eine weitere alphanumerische Variable :

```
128 string LoRaRX \ buffer LoRa transmitter -> ESP32
```

Beginnen wir mit der Initialisierung der seriellen Übertragung zum LoRa-Sender. Hier beträgt die Geschwindigkeit 115200 Baud. Dies ist die Standardübertragungsgeschwindigkeit des LoRa-Senders :

```

Serial \ Serielles Vokabular auswählen

\ Serial2 initialisieren
: Serial2.init ( -- )
    #SERIAL2_RATE Serial2.begin
;

```

Für unseren Testbefehl an den LoRa-Sender wählen wir die Arbeitsfrequenz des Senders, hier 868,5 Mhz:

```

\ Richten Sie die LoRa-Frequenz ein
: .band8685 ( -- )
    s" AT+BAND=868500000" LoRaTX $!
    $0d LoRaTX c+$!
    $0a LoRaTX c+$!      \ add CR LF code at end of command
    LoRaTX Serial2.write drop
;

```

Schließlich definieren wir ein Wort, mit dem wir die Antwort vom LoRa-Sender abrufen können :

```

\ Eingabe vom LoRa-Sender
: LoRaInput ( -- n )
    Serial2.available dup if
        LoRaRX maxlen$ nip
        Serial2.readBytes
        LoRaRX drop cell - !
    then
;

```

Das **LoRaInput** -Wort testet, ob eine serielle Linkübertragung vom seriellen UART2-Port empfangen wurde:

- wenn kein Empfang vorhanden ist, wird 0 zurückgegeben
- Wenn es Zeichen enthält, speichert es diese Zeichen in der alphanumerischen LoRaRX-Zeichenfolge und aktualisiert die Größe dieser Zeichenfolge.

Beispiel für Senden und Empfangen:

```

Serial2.init
.band8685
LoRaInput

```

Folgendes ergibt ein **LoRaRX**- Speicherauszug :

```

LoRaRX-Dump
--addr--- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F -----chars-----
3FFF-87A0 05 00 00 00 2B 4F 4B 0D 0A 31 0D 0A 2B 4F 4B 0D ....+OK..1..+OK.

```

Wenn wir **LoRaRX** ausführen , stellen wir die Adresse und insbesondere die Länge aller empfangenen Zeichen wieder her, einschließlich der Zeichen CR+LF (\$0d \$0a). Um nur Zeichen zu verwalten, die strikt im Intervall [0..0A..Za..z] enthalten sind, müssen Sie die Größe der Zeichenfolge um zwei Einheiten reduzieren:

```
: LoRaType ( -- )
  LoRaRX dup 0 > if
    2 - type
  else
    2drop
  then
;
LoRaType \ zeigt an: +OK
```

Hier wird beim Ausführen von **LoRaType** +OK angezeigt, was die Antwort auf unseren Testbefehl **AT+BAND=868500000** ist.

Einrichten des REYAX RYLR890 LoRa-Senders

Bevor wir Befehle für unseren REYAX RYLR896 LoRa-Sender definieren, definieren wir das Wort **cr^lf** :

```
: crlf ( -- )      \ gleiche Aktion wie cr, aber angepasst für LoRa
  $0d emit
  $0a emit
;
```

Der Zweck dieses CRLF-Worts besteht darin, die Übertragung auf dem UART2-Port von der ESP32-Karte zum LoRa-Sender abzuschließen. Die Definition dieses Wortes verwendet emit. Wundere dich nicht. Wir werden später sehen, wie man die vektorisierte Ausführung von Wörtern in der FORTH-Sprache nutzt, um die gewünschte Aktion beim Emitten auszuführen. Diese Lösung wird Einsteiger in die Programmiersprache FORTH überraschen. Es wird auch gezeigt, dass FORTH viel flexibler ist als viele andere Programmiersprachen.

Wesentliche Parameter

Hier ist die Liste der wesentlichen Parameter zur Konfiguration Ihres LoRa-Moduls.

Die Reihenfolge der Verwendung des **AT-** Befehls :

- Verwenden Sie **AT+ADDRESS**, um ADDRESS festzulegen. Als ADRESSE gilt die Identifikation des angegebenen Senders oder Empfängers.
- Verwenden Sie **AT+NETWORKID** , um die Lora-Netzwerk-ID festzulegen. Dies ist eine Gruppenfunktion. Nur durch die Einstellung derselben NETWORKID können Module miteinander kommunizieren. Wenn die ADRESSE des angegebenen Empfängers zu einer anderen Gruppe gehört, ist eine Kommunikation untereinander nicht möglich. Der empfohlene Wert: 1 ~ 15
- Verwenden Sie **AT+BAND** , um die Mittenfrequenz des Funkbandes anzupassen. Sender und Empfänger müssen die gleiche Frequenz nutzen, um miteinander zu kommunizieren.
- Verwenden Sie **AT+PARAMETER** , um die RF-Wireless-Einstellungen anzupassen. Sender und Empfänger müssen die gleichen Parameter einstellen, um miteinander kommunizieren zu können. Die Parameter sind wie folgt zu definieren:
 - **<Spreading Factor>** : Je größer der SF, desto besser die Empfindlichkeit. Die Übertragungszeit wird jedoch länger dauern.
 - **<Bandbreite>** : Je kleiner die Bandbreite, desto besser die Empfindlichkeit. Die Übertragungszeit wird jedoch länger dauern.

- **<Kodierungsrate>** : Die Kodierungsrate ist am schnellsten, wenn Sie sie auf 1 einstellen.
- **<Programmierte Präambel>** : Präambelcode. Wenn der Präambelcode größer ist, verringert sich die Wahrscheinlichkeit eines Datenverlusts. Der Präambelcode kann im Allgemeinen auf über 10 eingestellt werden, sofern die Übertragungszeit zulässig ist.
 - * Kommunikation bis zu 3 km: empfohlene Einstellung „AT+PARAMETER=10,7,1,7“
 - * Über 3 km: empfohlene Einstellung „AT+PARAMETER=12,4,1,7“
- Verwenden Sie **AT+SEND** , um Daten an die angegebene ADRESSE zu senden. Aufgrund des vom Modul verwendeten Programms erhöht sich der Nutzdatenanteil um mehr als 8 Byte gegenüber der tatsächlichen Datenlänge.

AT- Befehle muss **crLf übergeben werden** .

+OK antwortet, damit Sie den nächsten **AT- Befehl ausführen können** .

ADRESSE Definiert die Moduladresse

Jedes LoRa-Übertragungsmodul muss eine persönliche Adresse haben.

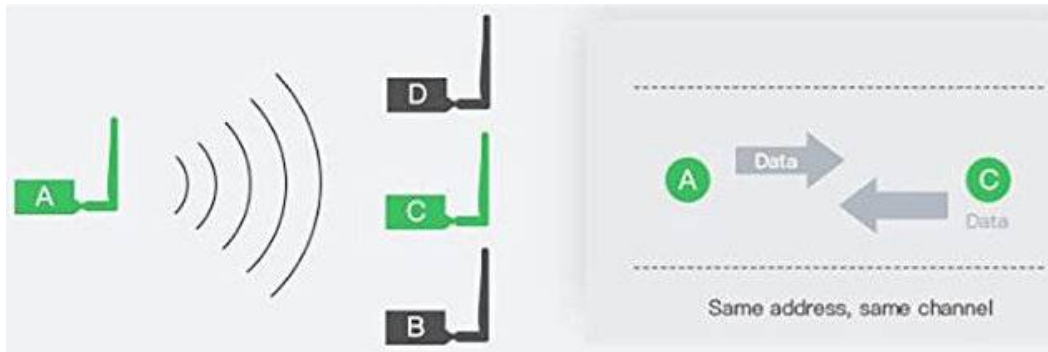
Syntax	Antwort
AT+ADDRESS=<address>	+OK
AT+ADRESSE=?	+ADDRESS=22

```
\ Stellen Sie die ADRESSE des LoRa-Senders ein:
\ s" " Wert im Intervall [0..65535][?] (Standard 0)
: ATaddress ( addr len -- )
  ." AT+ADDRESS="
  type crLf
;
```

<Adresse>=0~65535 (Standard 0)

Beispiel: Moduladresse auf **22 setzen** . Die Einstellungen werden in LoRa gespeichert.

```
s" 22" Ataddress
```



AT Testen Sie die LoRa-Verfügbarkeit

Syntax	Antwort
BEI.	+OK

```
\ LoRa-Testverfügbarkeit
: AT_ ( -- )
  ." AT"
  type crlf
;
```

BAND Einstellen der HF-Frequenz

Syntax	Antwort
AT+BAND=<Parameter>	+OK
AT+BAND=?	+BAND=868500000

```
\ Stellen Sie das BAND des LoRa-Senders ein:
\s" " Der Wert ist die HF-Frequenz, Einheit Hz
: ATband ( addr len -- )
  ." AT+BAND="
  type crlf
;
```

Parameter ist HF-Frequenz, Einheit ist Hz: 915000000Hz (Standard: RYLY89x)

Beispiel: Wählen Sie die Frequenz 868500000Hz:

```
s" 868500000" ATband
```

CPIN Legt das AES128-Netzwerkennwort fest

Syntax	Antwort
AT+CPIN=<Passwort>	+OK

Syntax	Antwort
AT+CPIN=?	+CPIN=FABC0002EEDCA.....

Passwort: AES-Passwort mit 32 Zeichen von 00000000000000000000000000000001 bis FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF.

Der Austausch wird akzeptiert, wenn beide Module das gleiche Passwort haben. Nach dem Zurücksetzen wird das bisherige Passwort gelöscht.

```
\Legen Sie das AES32-Passwort fest:
\ Der Wert \s"<Parameter>" ist ein 32 Zeichen langes AES-Passwort
\ von 00000000000000000000000000000001 bis FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
: ATcpin ( addr len -- )
    ." AT+CPIN="
    type crlf
;
```

Beispiel: Wählen Sie dieses Passwort: FABC0002EEDCAA90FABC0002EEDCAA90

```
s" FABC0002EEDCAA90FABC0002EEDCAA90" ATcpin
```

CRFOP Wählt die Ausgangs-HF-Leistung aus

Syntax	Antwort
AT+CRFOP=<Leistung>	+OK
AT+CRFOP=?	+CRFOP=10

Leistung: zwischen 0..15 und 15 dBm (Standard)

```
\Stellen Sie die CRFOP-HF-Ausgangsleistung ein:
\ Der Wert ist die HF-Ausgangsleistung zwischen 0 und 15
: ATcrfop ( addr len -- )
    ." AT+CRFOP="
    type crlf
;
```

Wählen Sie beispielsweise die Ausgangsleistung bei 10 dBm:

```
s" 10" ATcrfop
```

FACTORY Setzt alle aktuellen Einstellungen auf Standardwerte

Setzt alle aktuellen Einstellungen auf die Herstellerstandards.

Syntax	Antwort
IN+FABRIK	+FABRIK

```
\Setzen Sie den LoRa-Sender mit den WERKS-Parametern zurück
: ATfactory ( -- )
```

```

." AT+FACTORY"
crLf
;

```

IPR Legt die UART-Baudrate fest

Syntax	Antwort
AT+IPR=<Parameter>	+OK
AT+IPR=?	+IPR=38400

UART-Baud-Parameter:

- 300
- 1200
- 4800
- 9600
- 19200
- 28800
- 38400
- 57600
- 115200 (Standard)

Die Einstellungen werden im EEPROM gespeichert.

MODE Wählt den Arbeitsmodus aus

Syntax	Antwort
AT+MODE=<Parameter>	+OK
AT+MODE=?	+MODE=1

Einstellung:

- 0: Sende- und Empfangsmodus (Standard).
- 1: Schlafmodus.

```

\ Arbeitsmodus einstellen:
\ s" " Wert ist [0,1]
\ 0 (Standard) Sende- und Empfangsmodus
\ 1 Schlafmodus
: ATmode ( addr len -- )
  ." AT+MODE"
  type crLf
;

```

NETWORKID Wählt die Netzwerk-ID aus

Syntax	Antwort
AT+NETWORKID=<Netzwerk-ID>	+OK
AT+NETWORKID=?	+NETWORKID=6

```
\ NETZWERK-ID festlegen:  
\ s" " Der Wert ist [0..16] (0 Bay-StandardEinstellung)  
: ATnetworkid ( addr len -- )  
  ." AT+NETWORKID"  
  type crlf  
;
```

Netzwerk-ID: 0–16 (standardmäßig 0)

Beispiel: Wählen Sie die Netzwerk-ID 6 aus

Die Einstellungen werden im EEPROM gespeichert.

Die „0“ ist die öffentliche Kennung für LoRa. Es wird nicht empfohlen, 0 zum Festlegen der NETWORKID zu verwenden.

```
s" 6" Atnetworkid
```

PARAMETER-Definition von RF-Parametern

Syntax	Antwort
AT+PARAMETER=<Spreading Factor>, <Bandwidth>,<Coding Rate>, <Programmed Preamble>	+OK
AT+PARAMETER=?	+PARAMETER=7,3,4,5

Parameter:

- Spreading-Faktor
 - 7~12, (Standard 12)
- Bandbreite / Bandbreite (0~9):
 - 0: 7,8 kHz (nicht empfohlen, über Spezifikation)
 - 1: 10,4 kHz (nicht empfohlen, über Spezifikation)
 - 2: 15,6 kHz
 - 3: 20,8 kHz
 - 4: 31,25 kHz
 - 5: 41,7 kHz
 - 6: 62,5 kHz

- 7: 125 kHz (Standard).
- 8: 250 kHz
- 9: 500 kHz
- Codierungsrate
 - 1~4, (Standard 1)
- Programmierte Präambel
 - 4~7 (Standard 4)

Spreading-Faktor

Spreading- Faktor	Bitrate/Flow
7	5469 bps
8	3125 bps
9	1758 bps
10	977 bps
11	537 bps
12	293 bps

Codierungsrate

Die LoRa-Modulation fügt außerdem jeder Datenübertragung eine Vorwärtsfehlerkorrektur (FEC) hinzu. Diese Implementierung erfolgt durch Codierung von 4-Bit-Daten mit Redundanzen in 5 Bit, 6 Bit, 7 Bit oder sogar 8 Bit. Durch die Verwendung dieser Redundanz kann das LoRa-Signal Störungen abdecken. Der Wert der Kodierungsrate sollte entsprechend den Bedingungen des für die Datenübertragung verwendeten Kanals angepasst werden. Wenn der Kanal zu stark gestört ist, wird empfohlen, den Wert der Codierungsrate zu erhöhen.

Eine Erhöhung des CR-Wertes führt jedoch auch zu einer Verlängerung der Übertragungsdauer.

Beispiel: Parameter wie folgt einstellen:

<Spreading Factor> 7,<Bandwidth> 20,8KHz, <Coding Rate> 4,<Programmed Preamble>5,

s" 7,3,4,5" Atparameter

Software-RESET

Syntax	Antwort
AT+RESET	+OK

\Setzen Sie den LoRa-Sender zurück

```
: ATreset ( -- )
  ." AT+RESET"
  crlf
;
```

SEND sendet Daten an die angegebene Adresse

Syntax	Antwort
AT+SEND=<Adresse>,<Nutzdatenlänge>,<Daten>	+OK
AT+SEND=?	+SEND=50.5,HALLO

<Adresse>0~65535, wenn <Adresse> 0 ist, werden Daten an alle Adressen gesendet (von 0 bis 65535).

<Nutzlastlänge>Maximal 240 Bytes

<Daten>ASCII-Format

Forth-Code für ESP32Forth:

```
\ eine Zahl in eine Dezimalzeichenfolge umwandeln
: .n ( n ---)
  base @ >r decimal
  <# #s #> type
  r> base !
;
\SENDEN Daten an die Terminadresse senden
: ATsend { addr len address -- }
  ." AT+SEND="
  address .n [char] , emit
  len      .n [char] , emit
  addr len type crlf
;
```

Beispiel: Sendet die Zeichenfolge **HALLO** an Adresse 50:

```
s „HELLO“ 50 ATsend \ Anzeige: AT+SEND=50;5;HELLO
```

VER, um die Firmware-Version anzufordern

```
\ VER, um die Firmware-Version abzufragen
: ATver ( -- )
  ." AT+VER"
  crlf
;
```

Fehlerergebniscodes

- +ERR=1 Es gibt kein „Enter“ oder \$0D \$0A am Ende des AT-Befehls
- +ERR=2 Der Kopf des AT-Befehls ist keine „AT“-Zeichenfolge

- +ERR=3 Der AT-Befehl enthält kein „=“-Symbol
- +ERR=4 unbekannter Befehl
- +ERR=10 TX ist pünktlich
- +ERR=11 RX wurde überschritten
- +ERR=12 CRC-Fehler
- +ERR=13 Sendedaten mit mehr als 240 Byte
- +ERR=15 Unbekannter Fehler

Vektorisierung von Zeichenemissionen

Wenn Sie die Entwicklung unserer Worte zur Konfiguration des REYAX RYLR890 LoRa-Senders bis zu diesem Punkt verfolgt haben, hat Sie sicherlich etwas überrascht:

```
\ Stellen Sie die ADRESSE des LoRa-Senders ein:
\s"<Adresse>" Wert im Intervall [0..65535][?] (Standard 0)
: ATaddress ( addr len -- )
  ." AT+ADDRESS="
  type crlf
;
```

Denn wenn ich mich nicht irre, sendet diese Sequenz **." AT+ADDRESS="** die Zeichenfolge an unser Terminal und nicht an den Sender über den seriellen UART2-E-Port, also an den LoRa-Sender!

Wir verstehen Ihre Überraschung. Und wir werden sehen, wie wir den Zeichenfluss zum LoRa-Sender umleiten können, ohne etwas an der Definition unseres Wortes ATaddress **zu ändern**.

Vektorisierung in FORTH verstehen

Die FORTH-Sprache hat bestimmte Vorteile, die in vielen anderen Programmiersprachen überhaupt nicht vorhanden sind. Unter diesen Vermögenswerten erwähnen wir das Wort „**aufschieben**“. Mit diesem Wort können Sie ein Wort erstellen, dessen Aktion verzögert wird:

```
defer myWords
```

defer erstellt ein **myWords**- Wort , das NICHTS bewirkt!!!

Ja!

Es liegt nun an uns, dies in die Tat umzusetzen. Sehen wir uns diese Definition an :

```
: (myWords) ( -- )
  cr ." I display my words: "
;
```

Damit myWords ausgeführt wird (myWords), erhalten wir den Aktionscode von (myWords) und weisen ihn myWords zu:

```
' (myWords) is myWords
```

Wenn wir von nun an **myWords** eingeben, wird die in **(myWords)** definierte Aktion ausgeführt.

OK. Aber ist hier ein solcher Code-Overhead notwendig, wenn wir einfach **(myWords) ausführen können** ?

Und Sie haben völlig Recht, diese Frage zu stellen. Aber Sie können die Aktion von **myWords** ändern :

```
' vlist ist myWords
```

Wenn wir nun **myWords eingeben** , wird das Wort **vlist** ausgeführt.

Wir werden sehen, wie wir diesen Mechanismus nutzen können, um das Verhalten von ESP32Forth zu ändern ...

Vektorisierung in ESP32Forth

Beginnen wir mit einem kleinen Reverse Engineering. Wenn wir uns im ESP32Forth-Code umsehen, finden wir Folgendes für das Wort **."**:

```
: ."
  postpone s" state @ if postpone type else type then ; immediate
```

Hier ist das Wort, das uns interessiert, **type** , dessen Definition lautet :

```
defer type
```

Ahhhh.... Beginnen Sie zu verstehen?

Welche Aktion führt **type** aus? Wir finden dies im Quellcode von ESP32Forth :

```
: serial-type ( a n -- ) Serial.write drop ;
: default-type serial-type ;

' default-type is type
```

Wenn wir uns für das Wort **emit** interessieren, finden wir diese Definition im ESP32Forth-Quellcode :

```
: emit ( n -- )
  >r rp@ 1 type rdrop ;
```

Auch hier finden wir **type**.

Daher werden wir auf diesen Worttyp reagieren, um die Ausgabe von Zeichen auf den seriellen UART2-Port umzuleiten.

Vektorisieren Sie den Typ auf den seriellen UART2-Port

Indem wir uns die Definition des **serial-type** ansehen, definieren wir unsere Version für die Übertragung an den seriellen UART2-Port:

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

Wie weit ist es gegangen? Ist das nicht zu schwierig?

Um nun den Zeichenübertragungsstrom von ESP32Forth zum seriellen UART2-Port umzuleiten, führen Sie einfach die Sequenz '**serial2-type is type**' aus.

Aber wenn Sie das tun, werden Sie ein wenig Schwierigkeiten haben, zum normalen Verhalten von ESP32Forth zurückzukehren, es sei denn, Sie stellen **type** mit der Sequenz '**default-type is type**' auf seine ursprüngliche Aktion zurück .

Fassen wir diese Sequenzen in diesen beiden Worten zusammen:

```
: typeToLoRa ( -- )  
  ['] serial2-type is type  
  ;  
  
: typeToTerm ( -- )  
  ['] default-type is type  
  ;
```

Und jetzt, um unser **ATaddress**- Wort auszuführen , indem es die Zeichen an den seriellen UART2-Port überträgt, geben Sie einfach Folgendes ein:

```
typeToLoRa  
s" 45" ATaddress \ AT+ADDRESS=45 an UART2 senden  
typeToTerm
```

Und da warte ich auf Ihre Bemerkung: „Aber welchen Sinn hat die Vektorisierung?“

In unserem Fall bietet die Vektorisierung viele Vorteile:

- Schreiben Sie einfachen Code mit Wörtern, die bereits aus dem FORTH-Wörterbuch von ESP32Forth bekannt sind.
- bietet die Möglichkeit, alle Konfigurationswörter des LoRa-Senders gegenüber dem Terminal zu testen
- Möglichkeit, den Fluss auf ein anderes Gerät umzuleiten, zum Beispiel I2S oder UART1, ohne diese Parameterdefinitionen neu schreiben zu müssen...

Unabhängig von unserer Verwaltung der LoRa-Senderparameter verstehen wir, dass es ausreicht, denselben Vektorisierungsmechanismus für die vom seriellen UART2-Port empfangenen Zeichen zu nutzen, um problemlos die Steuerung von ESP32Forth über diesen seriellen Port zu übernehmen!

Genau das macht ESP32Forth, wenn Sie den WLAN- oder Bluetooth-Port aktivieren! Ich lade Sie ein, den Quellcode von ESP32Forth zu erkunden. Schauen Sie sich die **server** definition an:

```
: server ( port -- )
  server
  ['] serve-key is key
  ['] serve-type is type
  webserver-task start-task
;
```

Umschreiben einer kompletten Auflistung

Die Mindestparameter für die Kommunikation zwischen ESP32+LoRa-Karten sind:
Frequenz und Adresse:

```
\ *** LoRa-Setup-Wörter definieren *****

create $crlf
  $0d c, $0a c,

: crlf ( -- )      \ same action as cr, but adapted for LoRa
  $crlf 2 type
;

\ Set the ADDRESS of LoRa transmitter:
\ s" " value in interval [0..65535][?] (default 0)
: ATaddress ( addr len -- )
  ." AT+ADDRESS="
  type crlf ;

\ Set the BAND of LoRa transmitter:
\ s" " value is RF frequency, unit Hz
: ATband ( addr len -- )
  s" AT+BAND=" type
  type crlf ;
```

Aus der Originalverpackung entnommene LoRa-Sender kommunizieren theoretisch mit 115200 Baud mit der ESP32-Karte:

```
\ 115200 Geschwindigkeitskommunikation für LoRa REYAX
115200 value #SERIAL2_RATE

\Definition der OUTput- und INput-Puffer
128 string LoRaRX \ LoRa-Sender puffern -> ESP32

serial \Serielles Vokabular auswählen

\ Serial2 initialisieren
: Serial2.init ( -- )
```

```
#SERIAL2_RATE Serial2.begin
;
```

Wir stellen auch das Wort **LoRaInput** wieder her , das die vom LoRa-Sender am UART2-Port zurückgegebenen Nachrichten liest. Das Wort **rx.** Zur Erleichterung der Handhabung wurde hinzugefügt:

```
\ Eingabe vom LoRa-Sender
: LoRaInput ( -- n )
  Serial2.available if
    LoRaRX maxlen$ nip
    Serial2.readBytes
    LoRaRX drop cell - !
  else
    0 LoRaRX drop cell - !
  then
;

: rx.
  LoRaINPUT
  loRaRX type
;
```

Hier werden die Wörter **typeToLoRa** und **typeToTerm** verwendet, um die Textanzeige vom Terminal auf den UART2-Port zu übertragen:

```
\ *** Definition verzögerter Wörter *****

serial \ Select Serial vocabulary

: serial2-type ( a n -- )
  Serial2.write drop ;

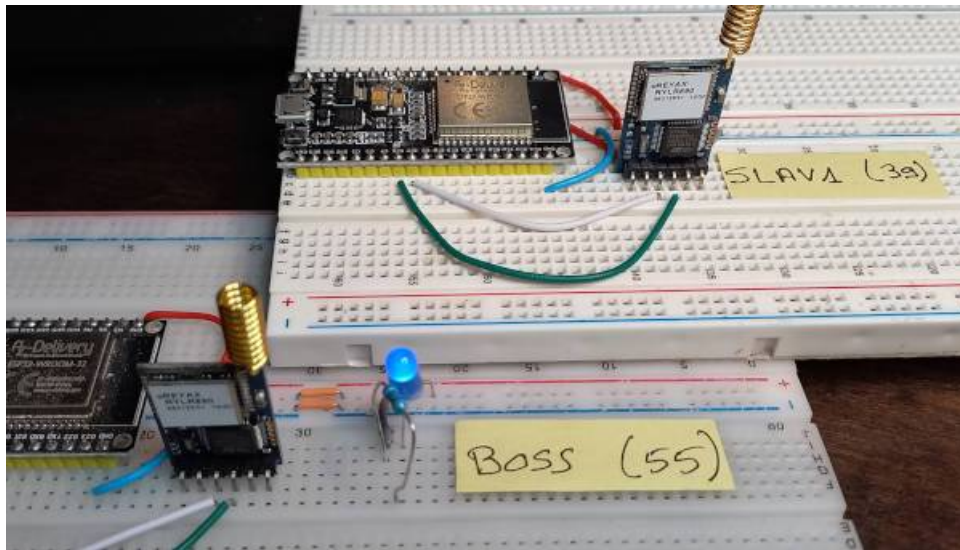
: typeToLoRa ( -- )
  0 echo ! \ disable display echo from terminal
  ['] serial2-type is type
;

: typeToTerm ( -- )
  ['] default-type is type
  -1 echo ! \ enable display echo from terminal
;
```

Hier haben wir die notwendigen und ausreichenden FORTH-Wörter, um unsere drei REYAX RYLR890 LoRa-Sender zu konfigurieren.

LoRa-Sender einrichten

Auf diesem Foto sind die Bezeichnungen BOSS und SLAV1 zu sehen. Dabei handelt es sich um einfache selbstklebende Post-its, die auf die Testplatten geklebt werden.



Wir werden drei Konstanten erstellen, die diesen Bezeichnungen zugeordnet sind:

```
55 constant LoRaBOSS
39 constant LoRaSLAV1
40 constant LoRaSLAV2
```

Um miteinander zu kommunizieren, müssen unsere LoRa-Sender auf die gleiche Frequenz eingestellt sein. Die gewählte Frequenz ist 868,5 MHz bzw. 868500000 Hz:

```
: emptyRX ( -- )
  LoRaINPUT
;

: SETband ( -- )
  emptyRX
  typeToLoRa
  s" 868500000" ATband
  typeToTerm
;
```

Beginnen wir mit der Konfiguration unseres ersten LoRa-Senders:

```
serial2.init
SETband
rx. \ display +OK
```

Wenn alles gut geht, wird **rx.** zeigt **+OK an** .

Dies ist die vom LoRa-Sender zurückgegebene Nachricht. Es ist möglich, eine Fehlermeldung wie +ERR=1 zu erhalten. Wiederholen Sie den Einstellungsbefehl.

Die Häufigkeit wird in 9 Ziffern ohne Trennzeichen oder Leerzeichen angegeben. Die Einheit ist Hz ¹¹.

Das REYAX RYLR896 LoRa-Modul kann Frequenzen von 862 MHz bis 1020 MHz betreiben.

ACHTUNG : Die Antenne muss auf die verwendete Frequenz abgestimmt sein! Die im REYAX LoRa-Modul verbaute Antenne ist auf Frequenzen um 868 MHz abgestimmt. Die Verwendung einer schlecht abgestimmten Antenne verringert die Übertragungseffizienz des LoRa-Moduls erheblich.

LoRa-Module senden schmalbandig. Wählen Sie eine der in Ihrem Land zugelassenen Frequenzen aus.

Ermittlung der Adresse von LoRa-Sendern

Um betriebsbereit zu sein, müssen sich alle Sender in einem Netzwerk auf derselben Frequenz befinden. Wenn Sie eine Nachricht an einen bestimmten Sender übermitteln möchten, müssen Sie die Adresse des Empfängersenders angeben. Wenn **BOSS** beispielsweise eine Nachricht an **SLAV1** senden möchte, übermitteln wir die Nachricht an den Sender mit der Adresse 39.

ACHTUNG : Sie können nicht zwei Sender mit derselben Adresse auf derselben Frequenz haben!

Hier Definition des Wortes, mit dem die Adresse 55 für den **BOSS -Sender konfiguriert werden kann** :

```
: SETaddress ( n -- )
  emptyRX
  typeToLoRa
  str ATband
  typeToTerm
;

LoRaBOSS SETaddress
rx. \ display +OK
```

Wir können für jeden Sender einen beliebigen Wert im Intervall [1..65535] annehmen. Adresse 0 ist für Übertragungen an alle LoRa-Sender reserviert, die auf derselben Frequenz hören.

Da unser **BOSS- Sender nun** konfiguriert ist, können wir ihn vom PC trennen und den Sender mit der Bezeichnung **SLAV1 anschließen** . Wir kompilieren das Quellskript und beginnen mit der Konfiguration **von SLAV1** :

```
serial2.init
SETband
```

11 Für FRANKREICH reicht das freie Frequenzband von 863 MHz bis 868,6 MHz. Quelle: ARCEP Das „Free-Bands-Portal“

```
rx. \ display +OK
```

```
LoRaSLAV1 SETaddress
```

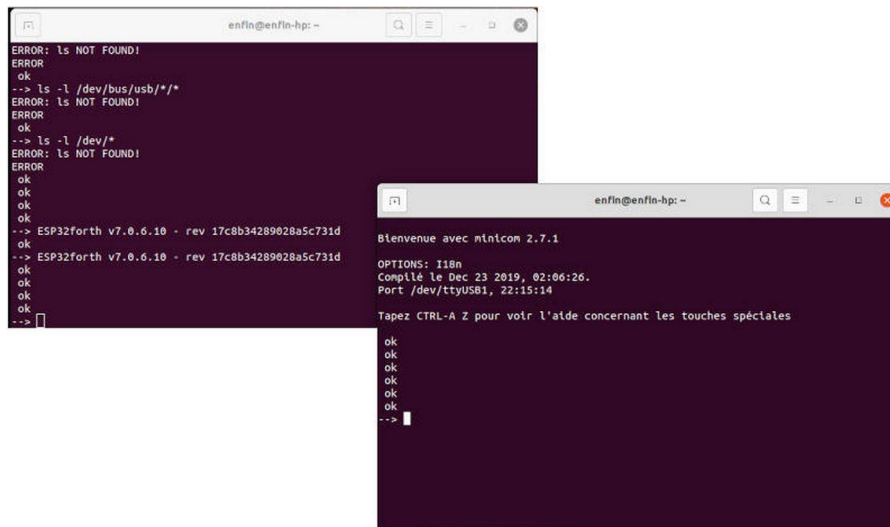
```
rx. \ display +OK
```

Kommunikation zwischen zwei LoRa REYAX RYLR890-Sendern

Um unsere REYAX RYLR890 LoRa-Sender zu initialisieren, müssen Sie:

- Ich habe zwei REYAX RYLR890 LoRa-Sender mit der ESP32-Karte
- Verbinden Sie jedes ESP32-Board mit einem verfügbaren USB-Port Ihres PCs

Hier haben wir unter Linux zwei Minicom-Terminals geöffnet:



The image shows two terminal windows. The left window is a standard Linux terminal with a dark purple background. It shows the user 'enfin' at the prompt 'enfin@enfin-hp: ~'. The user enters several 'ls' commands to check for USB devices, but they all fail with 'ERROR: ls NOT FOUND!'. Then, the user enters two 'ESP32forth' commands: 'ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d' and 'ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d'. Both commands succeed, indicated by 'ok' responses. The right window is a Minicom terminal window, also with a dark purple background. It shows the Minicom version '2.7.1' and the options 'I18n', 'Compiled le Dec 23 2019, 02:06:26.', and 'Port /dev/ttyUSB1, 22:15:14'. It also displays the instruction 'Tapez CTRL-A Z pour voir l'aide concernant les touches spéciales'. The user has entered several 'ok' responses, and the prompt '-->' is visible at the bottom.

Linux-Befehle zum Öffnen dieser beiden Terminals:

- Starten Sie über die Tastatur das Befehlsterminal mit STRG-ALT-T
- Wenn Sie im Befehlsfenster „sudo minicom“ eingeben, wird das mit /dev/ttyUSB0 verbundene Minicom-Terminal gestartet
- Geben Sie in der Eingabeaufforderung das Linux-Administratorkennwort ein. Das erste Terminal ermöglicht den Zugriff auf Ihre erste ESP32-Karte
- Starten Sie erneut über die Tastatur das Befehlsterminal mit STRG-ALT-T
- Geben Sie in diesem neuen Befehlsfenster sudo minicom -D /dev/ttyUSB1 ein, wodurch ein weiteres mit /dev/ttyUSB1 verbundenes Minicom-Terminal gestartet wird
- Geben Sie in der Eingabeaufforderung das Linux-Administratorkennwort ein. Dieses andere Terminal ermöglicht den Zugriff auf die zweite ESP32-Karte

Übertragung von BOSS zu SLAV2

Die Auflistung unseres FORTH-Codes unterscheidet sich nur geringfügig von der im vorherigen Kapitel. Wir haben einfach die Wörter **ATaddress** und **ATband** entfernt. Diese

Wörter sind nicht mehr erforderlich, um unsere LoRa **BOSS-** , **SLAV1-** und **SLAV2-Sender zu konfigurieren** .

Einmal konfiguriert, behält ein LoRa-Sender diese Einstellung bei, auch wenn er ausgeschaltet ist. Durch das Einschalten einer ESP32-Karte und ihres LoRa-Senders werden die Einstellungen des LoRa-Senders nicht geändert.

Die Wörter **ATaddress** und **ATband** werden durch **ATsend** ersetzt :

```
\SENDEN Daten an die Terminadresse senden
: ATsend { addr len address -- }
    ." AT+SEND="
    address n. [char] , emit
    len      n. [char] , emit
    addr len type crlf
;
```

Jetzt integrieren wir dieses Wort in **toSLAV2** :

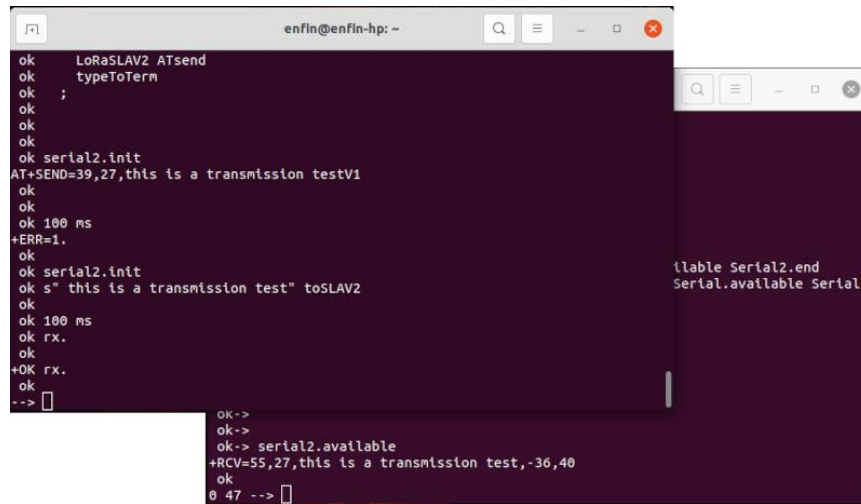
```
: toSLAV2 ( addr len -- )
    emptyRX
    typeToLoRa
    LoRaSLAV2 ATsend
    typeToTerm
;
```



Wir kompilieren das gleiche Programm auf jedem ESP32 (**BOSS** und **SLAV2**). Es ist sehr leicht. Einfach aus der Liste kopieren und in das Terminal einfügen. Jede ESP32-Karte erstellt ihre Liste in etwa zehn Sekunden.

Hier die beiden Fenster unseres Minicom-Terminals. Im linken Fenster können Sie **BOSS** steuern .

Das rechte Fenster steuert **SLAV2** :



```
enfin@enfin-hp: ~
ok LoRaSLAV2 ATsend
ok typeToTerm
ok ;
ok
ok
ok
ok serial2.init
AT+SEND=39,27,this is a transmission testV1
ok
ok
ok 100 ms
+ERR=1.
ok
ok serial2.init
ok s" this is a transmission test" toSLAV2
ok
ok 100 ms
ok rx.
ok
+OK rx.
ok
->

OK->
OK->
OK-> serial2.available
+RCV=55,27,this is a transmission test,-36,40
ok
0 47 ->
```

Testen wir die Übertragung von **BOSS** zu **SLAV2** . Platzieren Sie dazu den Mauszeiger im linken Fenster und geben Sie direkt ein:

```
serial2.init
s" this is a transmission test" toSLAV2
rx. \ display: +OK
```

Sind wir sicher, dass **SLAV2** die Nachricht erhalten hat? Nichts einfacher.

Wir platzieren den Mauszeiger im rechten Fenster und geben einfach Folgendes ein:

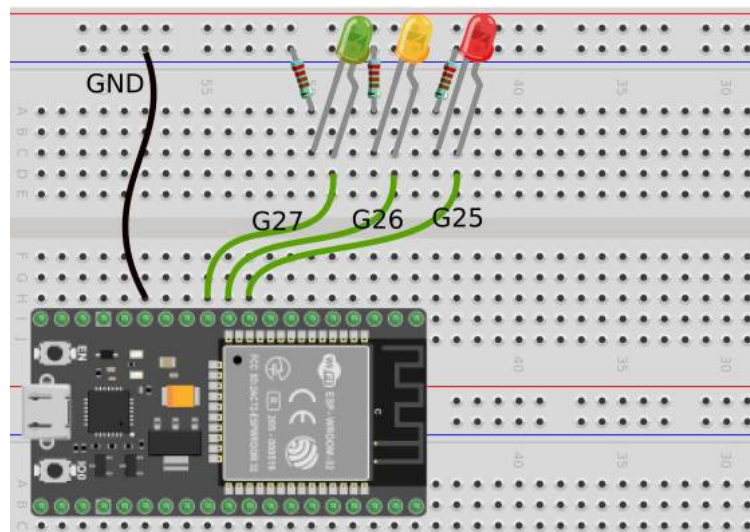
```
serial2.init
rx. \ display: +RCV=55,27,this is a transmission test,-36,40
```

Das Ergebnis dieser Übertragung durch **SLAV2** wird in der alphanumerischen Variablen **LoRaRX** gespeichert .

Schnittstelle einer LoRa-Übertragung mit ESP32Forth

Um die unglaubliche Flexibilität der FORTH-Sprache und insbesondere der ESP32Forth-Version auf ESP32 zu demonstrieren, verwenden wir das im Kapitel „*Eine Ampel mit ESP32 verwalten*“ verwendete Programm .

Das Problem bei den Definitionen in diesem Kapitel besteht darin, dass die Steuerung der LEDs die GPIO-Anschlüsse der seriellen Verbindung verwendet. Wir verschieben den LED-Anschluss daher wie folgt:



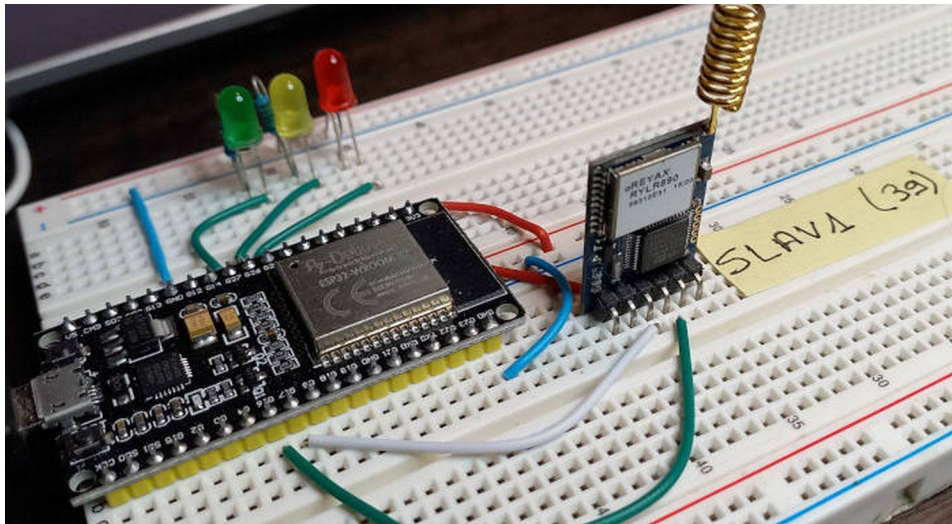
Hier ist die einzige Code-Anpassung, die zur Anpassung an den neuen Anschluss der LEDs vorgenommen wird :

```
\ neuer Code
27 constant ledGREEN      \ green  LED on GPIO2
26 constant ledYELLOW     \ yellow LED on GPIO21
25 constant ledRED        \ red    LED on GPIO17
```

Hier sind die Codesequenzen in FORTH-Sprache, um jede LED selektiv ein- oder auszuschalten. Diese Sequenzen können über das mit der ESP32-Karte verbundene Terminalfenster ausgeführt werden :

```
LEDinit
ledGREEN high      \ set GREEN led on
ledRED  high       \ set RED led on
ledGREEN low       \ set GREEN led off
```

Und es sind diese Sequenzen, und nur diese, die von den LoRa-Sendern gesendet und empfangen werden. Hier ist die Montage der LEDs und des LoRa-Senders auf unserer Testplatte namens **SLAV1** :



Das LoRa-Senderseitenprogramm namens BOSS

Wir werden das umfangreiche Kommunikationsprogramm abschließen. Wir beginnen mit dem, was im vorherigen Kapitel beschrieben wurde.

Die Auflistung umfasst die wesentlichen Komponenten, die eine LoRa-Übertragung von der mit BOSS gekennzeichneten ESP32-Karte ermöglichen.

Wir fügen einfach ein paar einfache Definitionen hinzu, um das Ein- und Ausschalten der LEDs, die sich auf der mit **SLAV1 gekennzeichneten Karte befinden, aus der Ferne auszuführen :**

```
55 constant LoRaBOSS
39 constant LoRaSLAV1
\ 40 constant LoRaSLAV2

: toSLAV1 ( addr len -- )
  emptyRX
  typeToLoRa
  LoRaSLAV1 ATsend
  typeToTerm
;

: REDhigh ( -- )
  s" LEDred high"    toSLAV1
;

: REDlow ( -- )
  s" LEDred low"     toSLAV1
;
```

```

: YELLOWhigh ( -- )
  s" ledYELLOW high" toSLAV1
;

: YELLOWlow ( -- )
  s" ledYELLOW low" toSLAV1
;

: GREENhigh ( -- )
  s" ledGREEN high" toSLAV1
;

: GREENlow ( -- )
  s" ledGREEN low" toSLAV1
;

```

Wir erstellen pro Befehl eine Definition mit dem Ziel, diese so einfach wie möglich zu gestalten. Es steht Ihnen frei, eine interaktivere Möglichkeit zu schaffen. Dies ist nicht der Zweck dieses Kapitels. Sobald die mit **BOSS** gekennzeichnete Karte angeschlossen und der Code kompiliert ist und wir einen Befehl an **SLAV1** senden möchten, geben wir zunächst einfach Folgendes in das Terminal ein:

```

serial2.init
REDhigh

```

Dadurch wird die **LEDred high**-Meldung an SLAV1 übermittelt. Die letzte Phase besteht darin, diesen Befehl so auszuführen, als ob er von einem an **SLAV1** angeschlossenen Terminal eingegeben würde.

Empfang und Ausführung von FORTH-Befehlen durch SLAV1

Fassen wir zusammen: Der BOSS-Sender sendet eine Nachricht, zum Beispiel **LEDred high**, an den SLAV1-Sender. Sender SLAV1 empfängt die Nachricht und führt **RXdecode** aus, um diesen FORTH-Befehl in der alphanumerischen Variablen **RCVdata** zu speichern.

```

      x . . . . "LEDred high" . . . . . x
      |                                     |
+-----+--+                                     +-----+--+
| BOSS  |                                     | SLAV1 |
+-----+--+                                     +-----+--+
                                           | RXdecode
                                           +-----> RXdata: LEDred high

```

Ausführen eines von LoRa empfangenen Befehls

In unserem Diagramm haben wir zwei ESP32-Boards mit jeweils einem LoRa-Sender:

- **BOSS** (Adresse 55), der FORTH-Befehle sendet
- **SLAV1** (Adresse 39), der diese FORTH-Befehle empfängt

Der einzige Unterschied zu FORTH-Befehlen, die direkt auf der PC-Tastatur eingegeben und vom an SLAV1 angeschlossenen Terminalprogramm übertragen werden, betrifft die von LoRa übertragenen und in **RXdata** gespeicherten Befehle.

Wie führe ich diese in **Rxdata gespeicherten Befehle aus** ? Die Antwort ist erschreckend einfach:

```
RCVdata evaluate
```

Wir brauchen absolut nichts anderes, um die LoRa-Übertragung mit einem Programm zu verbinden, das in einer ESP32-Karte eingebettet ist!!!

Hier ist eine sichere Definition dieser Schnittstelle:

```
: RXinterface ( -- )
  RCVdata ?dup if
    evaluate
  else
    2drop
  then
;
```

Hier sind einige Manipulationen in FORTH, um diese Schnittstelle zu testen :

```
LEDinit          \ initialize GPIOs
s" LEDred high" RCVdata $!
RCVdata RXinterface \ turn RED led on
s" LEDred low" RCVdata $!
RCVdata RXinterface \ turn RED led off
```

Wir haben unser Versprechen gehalten: von LoRa aus auf jedes Programm zu reagieren, ohne eine einzige Codezeile zu ändern.

In jeder ESP32-Karte können Sie mit dem Terminal ganz einfach alle Funktionalitäten Ihrer Programme kompilieren und testen.

Um diese Programme zu verbinden, reicht es dann aus, die LoRa-Übertragungsschicht und ihren Schnittstellencode hinzuzufügen.

Der Fernsender muss nur FORTH-Befehle senden, um auf Ihre Programme zu reagieren.

Nur die FORTH-Sprache ermöglicht eine so einfache Übertragung ->
Anwendungsschnittstelle!

Kommen wir nun zum letzten Punkt: Regelmäßiges Auslesen des Empfangspuffers des LoRa-Senders....

LoRa-Übertragungsverwaltungsschleife

Der LoRa-Sender ist mit dem seriellen UART2-Port verbunden. Wenn eine Übertragung empfangen wird, gibt das Wort **Serial2.available** die Anzahl der Bytes an, die im seriellen UART2-Puffer warten. Wenn keine Übertragung erfolgt, ist der von **Serial2.available** gemeldete Wert **Null**. Hier ist der Code zum Testen des Vorhandenseins von Zeichen, die von UART2 empfangen werden:

```
Serial
\ final loop
: LoRaLoop ( -- )
  begin
    Serial2.available \ not 0 if chars available
    if
      100 ms          \ ensures that entire transmission is received
      LoRaRX maxlen$ nip
      Serial2.readBytes
      LoRaRX drop cell - !
      RXdecode         \ analyse content of LoRa message
      RXinterface      \ interpret content or RCVdata
    then
      pause            \ skip to next task
    again
  ;
```

LoRaLoop- Code verwendet eine Endlosschleife. Es wird nicht empfohlen, dieses Wort so auszuführen, wie es ist. Wenn Sie dies tun, haben Sie keine Kontrolle mehr über den FORTH-Interpreter von ESP32Forth.

Um **LoRaLoop** zu verwenden , ohne den FORTH-Interpreter zu blockieren, definieren wir eine neue **my-Loop**- Aufgabe wie diese:

```
' LoRaLoop 100 100 task my-loop
my-loop start-task
```

Von diesem Moment an wird jede Übertragung, die von der mit **BOSS** gekennzeichneten Karte erfolgt, auf dieser mit **SLAV1** gekennzeichneten Karte interpretiert .

Damit unsere gesamte Programmierung in unserer **SLAV1**- Karte erhalten bleibt , schließen wir die allgemeine Initialisierung ab :

```

\ 115200 speed communication for LoRa REYAX
115200 value #SERIAL2_RATE

Serial
: mainInit ( -- )
  cr ." Starting SLAV1 LoRa" cr
  LEDinit
  #SERIAL2_RATE Serial2.begin    \ initialise Serial2
  my-loop start-task
;
startup: mainInit

```

SLAV1 gekennzeichneten ESP32-Karte kompiliert wurde , beim Neustart der Karte das Wort mainInit ausgeführt, was durch die Meldung Starting **SLAV1** LoRa bestätigt wird, die normalerweise angezeigt werden sollte.

Hier ist ein Foto der von der BOSS-Karte ausgeführten Aktionen, unten links eingefügt:



Auf der mit **SLAV1** gekennzeichneten Karte reagieren die LEDs mit einer Latenz von ein bis zwei Sekunden. Diese Verzögerung ist normal. Dies resultiert aus dem LoRa-Protokoll, das zwar langsam, aber äußerst robust ist. Auf dem Foto oben wurden die Tests mit einem Abstand von einem Meter durchgeführt. Für das Foto wurden die **BOSS-** und **SLAV1-**Karten zusammengebracht.

Wenn Sie **SLAV1** am Terminal angeschlossen lassen, haben Sie weiterhin die Kontrolle über den FORTH-Interpreter. Auch das ist normal! Das **LoRaLoop** -Wort wird im Multitasking ausgeführt.

Seit ihren Anfängen ist die FORTH-Sprache multitaskingfähig. Es war bereits in Versionen unter MS-DOS vorhanden, als MS-DOS kein Multitasking ermöglichte.

Mit ESP32Forth führen wir die Funktionalitäten von FORTH fort, einschließlich der Möglichkeiten, gleichzeitige Aufgaben zu aktivieren. In unserem speziellen Fall gibt Ihnen die Monitor-Aufgabe die Kontrolle über den Interpreter, während Sie die LEDs über die **LoRaLoop**- Aufgabe verwalten .

• Eine einfache WEB-Schnittstelle für ESP32Forth

Autor: Václav POSELT

Ich habe Forth wieder verwendet, nachdem ich jahrelang nicht mit FlashFORTH auf Atmega 328 und Arduino programmiert hatte. Nachdem ich meinen ersten Build erstellt hatte, war es notwendig, auch ein Bedienfeld für die Elektronik, Tasten und das Display zu bauen. Ich dachte, es ist Zeit für IoT und drahtlose Steuerung, also sparen Sie sich lieber die Bauarbeiten und steuern Sie alles drahtlos. Dafür bin ich auf ESP32 mit WiFi und BT umgestiegen, ich habe Dutzende Beispiele für Webinterface-Programme in Arduino C mit JavaScript gefunden, aber nichts in ESP32Forth auf ESP32. Für mich als Anfänger war das ein Problem.

Hier ist das Ergebnis meiner Bemühungen – ein einfaches Beispiel einer Webschnittstelle auf einem Webserver, der auf ESP32Forth läuft. Der Code basiert auf dem Beispiel von Peter Forth (peter-webpage-dht11-graphic-example.txt). Der gesamte Code befindet sich in der angehängten Datei **example_web.fs**.

Der Webserver läuft auf dem ESP32-Board bei aktivierter WLAN-Verbindung und antwortet auf Anfragen von Clients (Browser auf PC, Handy etc.).

Die grundlegende Weboberfläche ist daher einfach:

```
: runpage begin handleClient if serve-page 100 ms then 500 ms again ;
```

Dabei erkennt **handleClient**, ob Anfragen vom Client vorliegen, löst die Anfrage auf und übergibt dem Client HTML-Inhalte mit der Word-Dienstseite. Der **ms**- Timeout verbessert die Stabilität der WLAN-Verbindung in meinem Heimnetzwerk.

```
: serve-page ( --) \ simple parsing and action of client respond
  path s" /" str= if
    htmlpagesend exit \ exit leaves from serve-page
  then
  path s" /26/on" str= if
    cr ." ACTION for /26/on " cr \ here put action word
    0 to GPIO26 htmlpagesend exit
  then
  path s" /26/off" str= if
    cr ." ACTION for /26/off " cr
    1 to GPIO26 htmlpagesend exit
  then
  path s" /27/on" str= if
    cr ." ACTION for /27/on " cr
    0 to GPIO27 htmlpagesend exit
  then
  path s" /27/off" str= if
    cr ." ACTION for /27/off " cr
```

```

1 to GPIO27 htmlpagesend exit
then
path respond          \ actions for html forms
htmlpagesend exit    \ resend html page
;

```

Das Wort **serve-page** übernimmt den Client-Anfragetext aus dem Wortpfad in der Form „addr len“ und vergleicht ihn mit möglichen Client-Antworten. Bei jeder Übereinstimmung wird die entsprechende Aktion aktiviert und der HTML-Seiteninhalt mit dem Wort **htmlpagesend** aktualisiert. Aktionswörter können anstelle von Ersatzwörtern eingesetzt werden, wie z . B. **." ACTION for /26/on "**.

```

: htmlpagesend \ send whole html page
s" text/html" ok-response
htmlpage      \ create html page in webintstream buffer
webcontent send \ and send it to client
;

```

Das Wort **htmlpagesend** gibt den ersten Statuscode und HTML-Datentyp an den Client (Browser) zurück. Anschließend wird der HTML-Seitencode dynamisch als Text erstellt und schließlich zur Anzeige im Browser an den Client gesendet.

Kurz gesagt, es ist ein ziemlicher Prozess.

Für den praktischen Gebrauch habe ich mich auf drei Arten von Informationen konzentriert, die von der ESP32-Webschnittstelle generiert werden:

- einfache passive Textdaten wie die Ergebnisse bestimmter Messungen, beispielsweise von einer Wetterstation
- Tasten zum Ein-/Ausschalten, um etwas über die ESP32-Schaltung zu steuern
- HTML-Formulare zum Anpassen bestimmter Parameter in einem Programm, das auf ESP32 läuft.

Dazu habe ich dieses einfache Beispiel einer auf ESP32 generierten Webseite mit der IP-Adresse 192.168.1.6 erstellt.

The screenshot shows a web browser window with the address bar displaying '192.168.1.6/27/off'. The page title is 'ESP32FORTH Web Server'. Below the title, it shows 'GPIO 26 STATUS is: 0' and 'GPIO 27 STATUS is: 1'. There are two large buttons: a red one labeled 'GPIO26' and a green one labeled 'GPIO27'. Below these are several input forms: a text input field with 'Text Text' and a 'Submit' button; a number input field with a range of '1 and 10' and a 'Submit' button; a volume slider set to '51' with a 'Submit' button; a time selection field with 'Submit' button; a birthday field with 'Submit' button; a car selection dropdown with 'Volvo' selected and a 'Submit' button; and a button labeled 'Click me to display Date and Time.'. At the bottom, it shows the date and time: 'Wed Nov 01 2023 22:42:13 GMT+0100 (středoevropský standardní čas)'.

Figure 35: page web générée par ESP32forth

Somit ist der Status des Text-GPIO 26: Textinformationen und der Wert 0 oder 1 ist der FORTH-Wert für GPIO26, der beim Generieren der HTML-Seite in die Webseite einbezogen wird.

Mit den Tasten GPIO26 und GPIO27 können Sie den entsprechenden Wert ändern, um etwas zu steuern, beispielsweise ein mit den ESP32-GPIOs verbundenes Relais.

Andere HTML-Formulare können eine erweiterte Anpassung der Programmeinstellungen steuern.

Der letzte *Click me to display ...* generiert nur tatsächliche Datums-/Uhrzeitinformationen vom Client-Browser ohne programmgesteuerte Verbindung zum ESP32forth-Code.

Skript **example_web.fs** in der Datei **ESP32forth-book.zip** , die hier verfügbar ist:
<https://github.com/MPETREMAN11/ESP32forth/blob/main/documentation/ESP32forth-book.zip>

Dann nur kurz:

Die Zeilen 8–29 erstellen das **mvbar**- Hilfewort , das als **mvbar** für jeden mehrzeiligen Text | verwendet wird um eine temporäre Zeichenfolge wie `addr len` über mehrere Textzeilen zu erstellen.

Der Puffer für den HTML-Seitentext wird in Zeile 31 per **stream webintstream** und verwendet **Word >stream** , um Textteile zusammenzufügen.

Das Wort **htmlpage** in den Zeilen 46 bis 135 erstellt nach jeder Aktivierung dynamisch HTML-Text. Die Zeilen 67 bis 71 erstellen Text mit den tatsächlichen Werten von FORTH GPIO26, GPIO27. Bei der kontinuierlichen Anzeige bestimmter Messwerte ist es notwendig, im generierten HTML Code für die automatische HTML-Seitenaktualisierung zu hinterlegen.

Die Zeilen 72 bis 88 erstellen rote oder grüne Schaltflächen basierend auf GPIO-Werten mit Kundeninformationen `/26/on` oder `/26/off` zur Erkennung im Service-Seitenwort.

Die Zeilen 91–127 generieren HTML-Formulardaten, um bestimmte Werte wie Daten, Zeit, Text oder Bereich anzupassen. Die Zeilen 128–131 generieren nur tatsächliche Datums-/Uhrzeitinformationen mit JavaScript-Code.

Am Ende des Codes steht die Aktivierung des Servers mit WLAN und das Starten der Weboberfläche als Hintergrundaufgabe.

Ich präsentiere diesen Code als Grundlage für Experimente. Ich bin sicher, dass es verbessert werden kann, Feedback ist willkommen.

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 7.0.7.21a

FORTH

-	-rot	/	/	:	:noname	!
?	?do	?dup	.	."	.s	'
(local)	[	[']	[char]	[ELSE]	[IF]	[THEN]
]	{	{	}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	bg	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	fg	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>	r
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](#)

espnow

[esp_now_init](#) [esp_now_deinit](#) [esp_now_get_version](#) [esp_now_send](#)
[esp_now_register_rcv_cb](#)
[esp_now_unregister_rcv_cb](#) [esp_now_register_send_cb](#) [esp_now_unregister_send_cb](#)
[esp_now_add_peer](#) [esp_now_del_peer](#) [esp_now_mod_peer](#) [esp_now_is_peer_exist](#)
[esp_now_get_peer_num](#) [esp_now_set_pmk](#) [ESP_NOW_MAX_DATA_LEN](#) [ESP_NOW_ETH_ALEN](#)
[espnow-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?](#)
[fill132](#) ['heap](#) ['context](#) ['latestxt](#) ['notfound](#) ['heap-start](#) ['heap-size](#) ['stack-cells](#)
['boot](#) ['boot-size](#) ['tib](#) ['argc](#) ['argv](#) ['runner](#) ['throw-handler](#) [NOP](#) [BRANCH](#) [0BRANCH](#)
[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_RTCRAM 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 OledFastHLine OledFastVLine
OledCirc OledCircF OledRect OledRectF OledRectR OledRectRF oled-builtins

```

recognizers

```

REC-FNUM RECOGNIZE RECSTACK +RECOGNIZER -RECOGNIZER GET-RECOGNIZERS SET-RECOGNIZERS
RECTYPE-NUM RECTYPE-IMM RECTYPE-WORD RECTYPE-NONE RECTYPE: REC-NUM REC-FIND

```

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](#)

rmt

[rmt_set_clk_div](#) [rmt_get_clk_div](#) [rmt_set_rx_idle_thresh](#) [rmt_get_rx_idle_thresh](#) [rmt_set_mem_block_num](#) [rmt_get_mem_block_num](#) [rmt_set_tx_carrier](#) [rmt_set_mem_pd](#) [rmt_get_mem_pd](#) [rmt_tx_start](#) [rmt_tx_stop](#) [rmt_rx_start](#) [rmt_rx_stop](#) [rmt_tx_memory_reset](#) [rmt_rx_memory_reset](#) [rmt_set_memory_owner](#) [rmt_get_memory_owner](#) [rmt_set_tx_loop_mode](#) [rmt_get_tx_loop_mode](#) [rmt_set_rx_filter](#) [rmt_set_source_clk](#) [rmt_get_source_clk](#) [rmt_set_idle_level](#) [rmt_get_idle_level](#) [rmt_get_status](#) [rmt_set_rx_intr_en](#) [rmt_set_err_intr_en](#) [rmt_set_tx_intr_en](#) [rmt_set_tx_thr_intr_en](#) [rmt_set_gpio](#) [rmt_config](#) [rmt_isr_register](#) [rmt_isr_deregister](#) [rmt_fill_tx_items](#) [rmt_driver_install](#) [rmt_driver_uninstall](#) [rmt_get_channel_status](#) [rmt_get_counter_clock](#) [rmt_write_items](#) [rmt_wait_tx_done](#) [rmt_get_ringbuf_handle](#) [rmt_translator_init](#) [rmt_translator_set_context](#) [rmt_translator_get_context](#) [rmt_write_sample](#) [rmt-builtins](#)

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 !field field-op field struct-align max! align-by struct last-struct  
long ptr i64 u32 i32 u16 i16 u8 i8 sc@ typer last-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
```

timers

```
interval onalarm int-enable! alarm-enable! divider! autoreload! increase!  
enable! alarm! alarm@ timer! timer@ tmp t>nx timer\_isr\_callback\_add timer\_init\_null  
timer\_get\_counter\_value timer\_set\_counter\_value timer\_start timer\_pause  
timer\_set\_counter\_mode timer\_set\_auto\_reload timer\_set\_divider  
timer\_set\_alarm\_value
```

```
timer_get_alarm_value timer_set_alarm timer_group_intr_enable
timer_group_intr_disable
timer_enable_intr timer_disable_intr timers-builtins
```

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

[WiFi MODE APSTA](#) [WiFi MODE AP](#) [WiFi MODE STA](#) [WiFi MODE NULL](#) [WiFi.config](#) [WiFi.begin](#)
[WiFi.disconnect](#) [WiFi.status](#) [WiFi.macAddress](#) [WiFi.localIP](#) [WiFi.mode](#) [WiFi.setTxPower](#)
[WiFi.getTxPower](#) [WiFi.softAP](#) [WiFi.softAPIP](#) [WiFi.softAPBroadcastIP](#)
[WiFi.softAPNetworkID](#)
[WiFi.softAPConfig](#) [WiFi.softAPdisconnect](#) [WiFi.softAPgetStationNum](#) [WiFi-builtins](#)

Wire

[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, WTLB, WER, WDTLB, WAITL, SSX, SSU, SSR, SSL, 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 MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL, MULA.AA MOVT,
MOVSP, MOV~~T~~.S, MOVF.S, MOVGEZ.S, MOVLTZ.S, MOVNEZ.S, MOVEQZ.S, ULE.S, OLE.S,
ULT.S, OLT.S, UEQ.S, OEQ.S, UN.S, CMPSOP 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 MOVF, 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,
 S32CI1, [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, DPFW, 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>

•

Index

also.....	95	ansi.....	94	Automatischer Start.....	111
analogRead.....	188	Anzeigeposition.....	63	BASE.....	84
and.....	43	asm.....	309	bg.....	63

binary.....	38	Kommentar.....	67	SPI.....	215
bluetooth.....	310	ledc.....	238, 311	SPIFFS.....	138, 313
c@.....	60	ledcAttachPin.....	238	startup:.....	111
cat.....	139	ledcWriteTone.....	238	streams.....	313
commande AT.....	278	load.....	125	string.....	89
constant.....	61	login.....	119	Stromversorgung.....	109
cp.....	139	ls.....	138	struct.....	76
create.....	103	ml.....	152, 265	structures.....	76, 78, 313
decimal.....	38	m@.....	155	tasks.....	313
DECIMAL.....	84	ms-ticks.....	180	telnetd.....	120, 313
defer.....	98	mv.....	139	Tera Term-Terminal.....	113
DOES>.....	103	Netbeans.....	142	Textfarben.....	63
dump.....	55	normal.....	63	thru.....	126
edit.....	128	oled.....	193, 311	timers.....	313
editor.....	94, 123, 310	OledReset.....	207	to.....	68
ESP.....	310	only.....	95	touch.....	139
espnw.....	310	page.....	63	u.....	41
EXECUTE.....	97	pi.....	80	value.....	61
F.....	80	recognizers.....	311	variable.....	61
F>S.....	82	RECORDFILE.....	130, 204	visual.....	314
fconstant.....	81	registers.....	312	vlist.....	94
fg.....	63	rerun.....	172	voclist.....	93
flush.....	125	riscv.....	312	web-interface.....	314
FORTH.....	308	rm.....	139	WiFi.....	314
FORTH-Wort.....	31	rmt.....	312	wipe.....	124
fvariable.....	81	rtos.....	312	Wire.....	314
GIT.....	143	S>F.....	83	Wire.detect.....	198
handleClient.....	304	save-buffers.....	126	xtensa.....	314
hex.....	38	SD.....	312	xtensa-assembler.....	247, 251
HEX.....	84	SD_MMC.....	312	:noname.....	100
HOLD.....	85	see.....	55	.s.....	55
httpd.....	310	Serial.....	312	{.....	67
include.....	129, 138	server.....	120	}.....	67
insides.....	310	set-precision.....	80	#.....	85
internals.....	310	SF!.....	81	#>.....	85
interrupts.....	311	SF@.....	81	#S.....	85
interval.....	172	shift.....	42	+to.....	68
is.....	98	sockets.....	313	<#.....	85
		spi.....	313		