El gran libro
por ESP32forth

versión 1.2 - 29. octubre 2023

Autor

- Marc PETREMANN petremann@arduino-forth.com

Colaboradores

- Vaclav POSSELT
-

Indice

Autor....1
Colaboradores....1

Introducción....5
Ayuda de traducción....5

Descubrimiento de la tarjeta ESP32....6
Presentación....6
Puntos fuertes....6
Entradas/salidas GPIO en ESP32....7
Periféricos ESP32....9

Instalar ESP32Forth....10
Descargar ESP32forth....10
Compilando e instalando ESP32forth....10
Configuraciones para ESP32 WROOM....12
Iniciar la compilación....13

Solucionar el error de conexión de carga....14

¿Por qué programar en lenguaje FORTH en ESP32?....16
Preámbulo....16
Límites entre lenguaje y aplicación....16
¿Qué es una CUARTA palabra?....17
¿Una palabra es una función?....17
Lenguaje FORTH comparado con el lenguaje C....18
Qué le permite hacer FORTH en comparación con el lenguaje C....19
Pero ¿por qué una pila en lugar de variables?....20
¿Estás convencido?....20
¿Hay alguna solicitud profesional escrita en FORTH?....20

Un verdadero FORTH de 32 bits con ESP32Forth....23
Valores en la pila de datos....23
Valores en la memoria....23
Procesamiento de textos dependiendo del tamaño o tipo de datos....24
Conclusión....25

Diccionario / Pila / Variables / Constantes....27
Ampliar diccionario....27
Gestión de diccionarios....27
Pilas y notación polaca inversa....28
Manejo de la pila de parámetros....29
La pila de retorno y sus usos....29
Uso de memoria....30
Variables....30
Constantes....31
Valores pseudoconstantes....31
Herramientas básicas para la asignación de memoria....31

Colores de texto y posición de visualización en el terminal.....33
Codificación ANSI de terminales.....33
Coloración de texto.....34
Posición de visualización.....35
Variables locales con ESP32Forth.....37
Introducción.....37
El comentario de la pila falsa.....37
Acción sobre variables locales.....38
Estructuras de datos para ESP32forth.....41
Preámbulo.....41
Tablas en FORTH.....41
Matriz de datos unidimensional de 32 bits.....41
Palabras de definición de tabla.....42
Leer y escribir en una tabla.....42
Ejemplo práctico de gestión de una pantalla virtual.....43
Gestión de estructuras complejas.....46
Definición de sprites.....48
Instalación de la biblioteca OLED para SSD1306.....51
Números reales con ESP32 en adelante.....54
Los reales con ESP32 en adelante.....54
Precisión de números reales con ESP32forth.....54
Constantes y variables reales.....55
Operadores aritméticos en números reales.....55
Operadores matemáticos sobre números reales.....56
Operadores lógicos en números reales.....56
Entero ↔ transformaciones reales.....56
El generador de números aleatorios.....58
Característica.....58
Procedimiento de programación.....59
Función RND en ensamblador XTENSA.....59
Contenido detallado de los vocabularios ESP32forth.....61
Version v 7.0.7.15.....61
FORTH.....61
asm.....62
bluetooth.....63
editor.....63
ESP.....63
httpd.....63
insides.....63
internals.....63
interrupts.....64
ledc.....64
oled.....64
registers.....64
riscv.....64

rtos....65
SD....65
SD_MMC....65
Serial....65
sockets....65
spi....65
SPIFFS....65
streams....65
structures....66
tasks....66
telnetd....66
visual....66
web-interface....66
WiFi....66
xtensa....66
Recursos....67
En inglés....67
En francés....67
GitHub....67

Introducción

Desde 2019 he gestionado varios sitios web dedicados al desarrollo del lenguaje FORTH para placas ARDUINO y ESP32, así como la versión web eForth :

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

Estos sitios están disponibles en dos idiomas, francés e inglés. Cada año pago por el alojamiento del sitio principal. **arduino-forth.com**.

Tarde o temprano –y lo más tarde posible– sucederá que ya no podré garantizar la sostenibilidad de estos lugares. La consecuencia será que la información difundida por estos sitios desaparecerá.

Este libro es la recopilación del contenido de mis sitios web. Se distribuye gratuitamente desde un repositorio de Github. Este método de distribución permitirá una mayor sostenibilidad que los sitios web.

De paso, si algunos lectores de estas páginas desean hacer su aporte, son bienvenidos. :

- para sugerir capítulos ;
- para informar errores o sugerir cambios ;
- para ayudar con la traducción...

Ayuda de traducción

Google Translate te permite traducir textos fácilmente, pero con errores. Por eso pido ayuda para corregir las traducciones.

En la práctica, proporciono los capítulos ya traducidos en formato LibreOffice. Si desea ayudar con estas traducciones, su función será simplemente corregir y devolver estas traducciones.

Corregir un capítulo lleva poco tiempo, de una a unas pocas horas.

Para contactar conmigo : petremann@arduino-forth.com

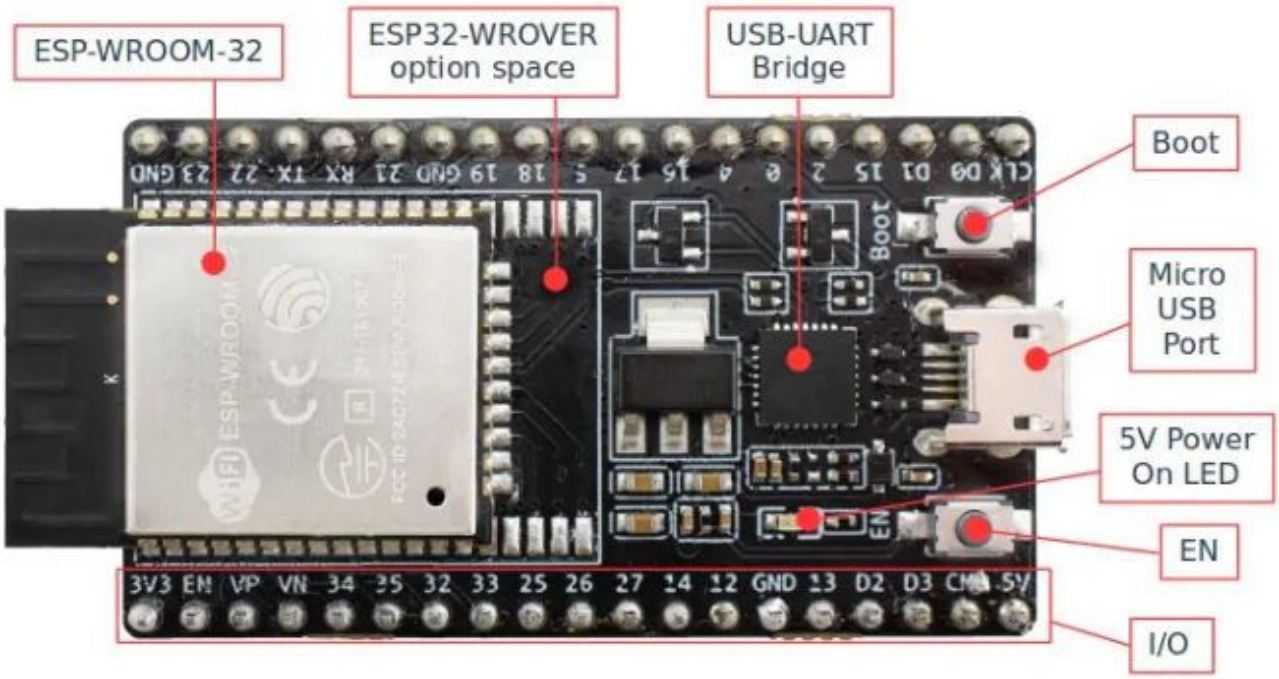
Descubrimiento de la tarjeta ESP32

Presentación

La placa ESP32 no es una placa ARDUINO. Sin embargo, las herramientas de desarrollo aprovechan ciertos elementos del ecosistema ARDUINO, como ARDUINO IDE.

Puntos fuertes

En cuanto al número de puertos disponibles, la tarjeta ESP32 se sitúa entre un ARDUINO



NANO y un ARDUINO UNO. El modelo básico tiene 38 conectores :

Los dispositivos ESP32 incluyen :

- 18 canales de convertidor analógico a digital. (ADC)
- 3 interfaces SPI
- 3 interfaces UART
- 2 interfaces I2C
- 16 canales de salida PWM
- 2 convertidores de digital a analógico (DAC)
- 2 interfaces I2S

- 10 GPIO de detección capacitiva

La funcionalidad ADC (convertidor analógico a digital) y DAC (convertidor digital a analógico) están asignadas a pines estáticos específicos. Sin embargo, puedes decidir qué pines son UART, I2C, SPI, PWM, etc. Sólo necesitas asignarlos en el código. Esto es posible gracias a la función de multiplexación del chip ESP32.

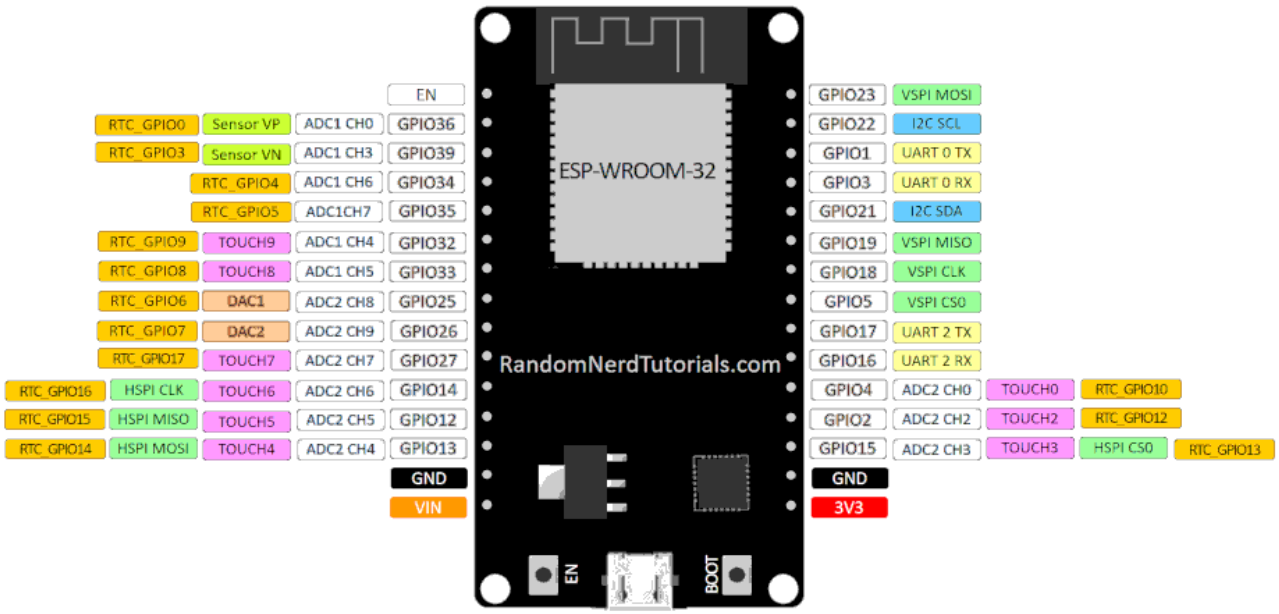
La mayoría de los conectores tienen múltiples usos.

Pero lo que distingue a la placa ESP32 es que está equipada de serie con soporte WiFi y Bluetooth, algo que las placas ARDUINO sólo ofrecen en forma de extensiones.

Entradas/salidas GPIO en ESP32

Aquí, en foto, la tarjeta ESP32 desde la que explicaremos el papel de las diferentes entradas/salidas GPIO :

La posición y la cantidad de E/S GPIO pueden cambiar según la marca de la tarjeta. Si este es el caso, sólo son auténticas las indicaciones que aparecen en el mapa físico. En la foto, fila inferior, de izquierda a derecha : CLK, SD0, SD1, G15, G2, G0, G4, G16.....G22, G23, GND.



En este diagrama, vemos que la fila inferior comienza con 3V3 mientras que en la foto, esta E/S está al final de la fila superior. Por lo tanto, es muy importante no confiar en el diagrama y, en su lugar, verificar la correcta conexión de los periféricos y componentes en la tarjeta física ESP32.

Las placas de desarrollo basadas en un ESP32 generalmente tienen 33 pines aparte de los de la fuente de alimentación. Algunos pines GPIO tienen funciones un tanto particulares :

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

Si tu tarjeta ESP32 tiene E/S GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, definitivamente no debes usarlas porque están conectadas a la memoria flash del ESP32. Si los usas el ESP32 no funcionará.

Las E/S GPIO1(TX0) y GPIO3(RX0) se utilizan para comunicarse con la computadora en UART a través del puerto USB. Si los utilizas, ya no podrás comunicarte con la tarjeta.

GPIO36(VP), GPIO39(VN), GPIO34, GPIO35 I/O se pueden utilizar solo como entrada. Tampoco tienen resistencias pullup y pulldown internas incorporadas.

El terminal EN le permite controlar el estado de encendido del ESP32 a través de un cable externo. Está conectado al botón EN de la tarjeta. Cuando el ESP32 está encendido, está a 3,3 V. Si conectamos este pin a tierra el ESP32 se apaga. Puedes usarlo cuando el ESP32 está en una caja y quieres poder encenderlo/apagarlo con un interruptor.

Periféricos ESP32

Para interactuar con módulos, sensores o circuitos electrónicos, el ESP32, como cualquier microcontrolador, dispone de multitud de periféricos. Hay más que en una placa Arduino clásica.

ESP32 tiene los siguientes periféricos:

- 3 interfaces UART
- 2 interfaces I2C
- 3 interfaces SPI
- 16 salidas PWM
- 10 sensores capacitivos
- 18 entradas analógicas (ADC)
- 2 salidas DAC

ESP32 ya utiliza algunos periféricos durante su funcionamiento básico. Por lo tanto, hay menos interfaces posibles para cada dispositivo.

Instalar ESP32Forth

Descargar ESP32forth

El primer paso consiste en recuperar el código fuente, en lenguaje C, de ESP32forth. Preferiblemente utilice la versión más reciente:
https://esp32forth.appspot.com/ESP32forth.html

Contenido del archivo descargado:

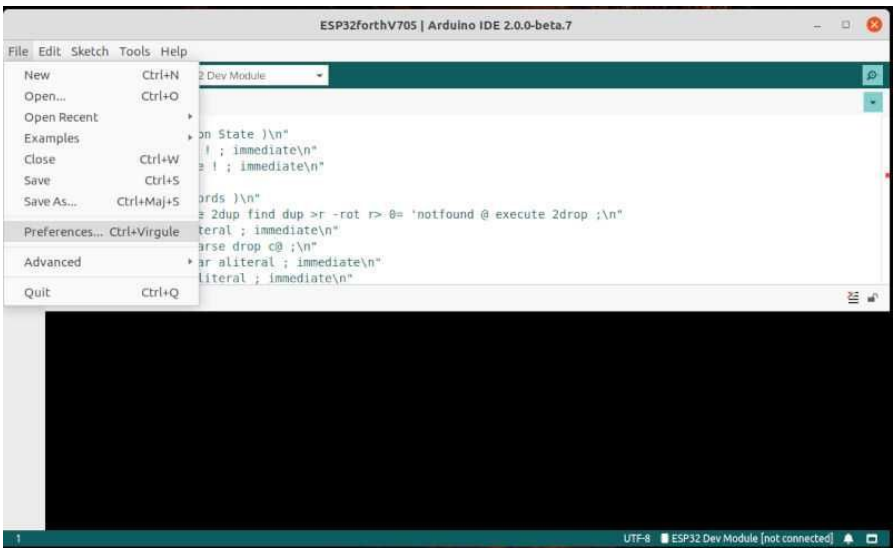
```
ESP32forth-7.0.x.x
     ESP32forth
          readme.txt
          esp32forth.ino
          optional
               SPI-flash.h
               serial-blueooth.h
               ...etc...
```

Compilando e instalando ESP32forth

archivo **esp32forth.ino** en un directorio de trabajo. El directorio opcional contiene archivos que permiten la extensión de ESP32 en adelante. Para nuestra primera compilación y carga de ESP32 en adelante, estos archivos no son necesarios.

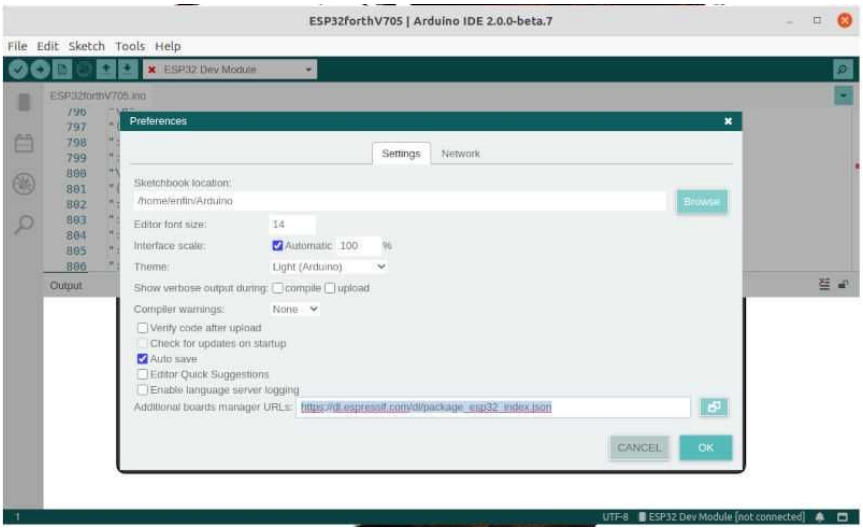
Para compilar ESP32 en adelante, debe tener ARDUINO IDE ya instalado en su computadora:
https://docs.arduino.cc/software/ide-v2

Una vez instalado ARDUINO IDE, ejecútelo. ARDUINO IDE está abierto, aquí la versión 2.0. Haga clic en *file* y seleccione *Preferences* :

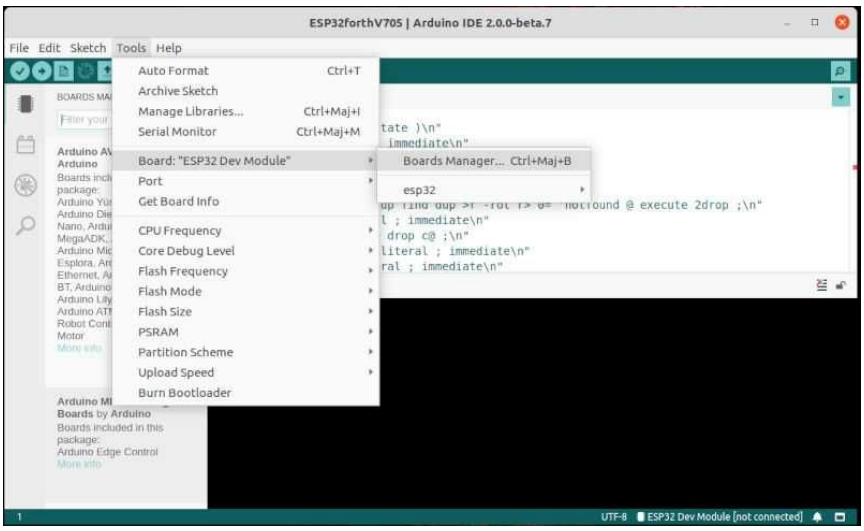


En la ventana que aparece, vaya al cuadro de entrada marcado *Additional boards manager URLs*: e ingrese esta línea:

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



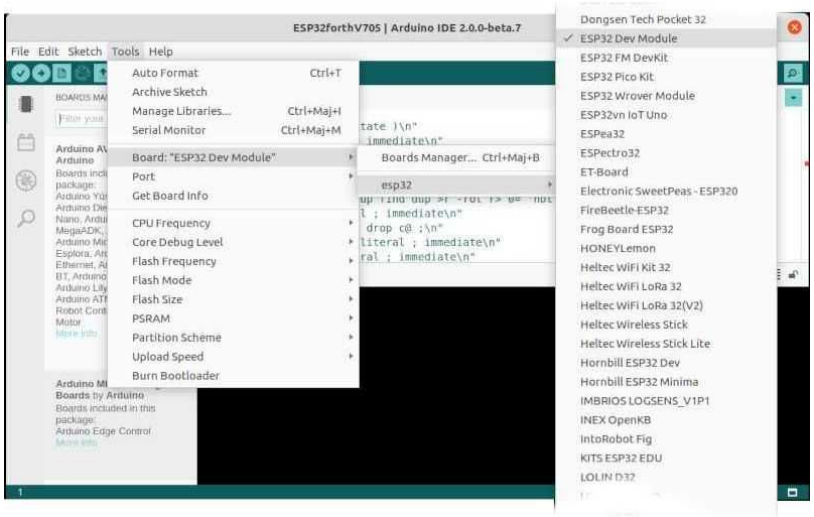
A continuación, haga clic en *Tools* y seleccione *Board* :.



Esta selección debería ofrecerle la instalación de paquetes para ESP32. Acepta esta instalación.

Entonces deberías poder acceder a la selección de tarjetas ESP32:

Selección de placa **ESP32 Dev Module** :



Configuraciones para ESP32 WROOM

Aquí están las otras configuraciones necesarias antes de compilar ESP32 en adelante. Accede a la configuración haciendo clic nuevamente en *Tools* :

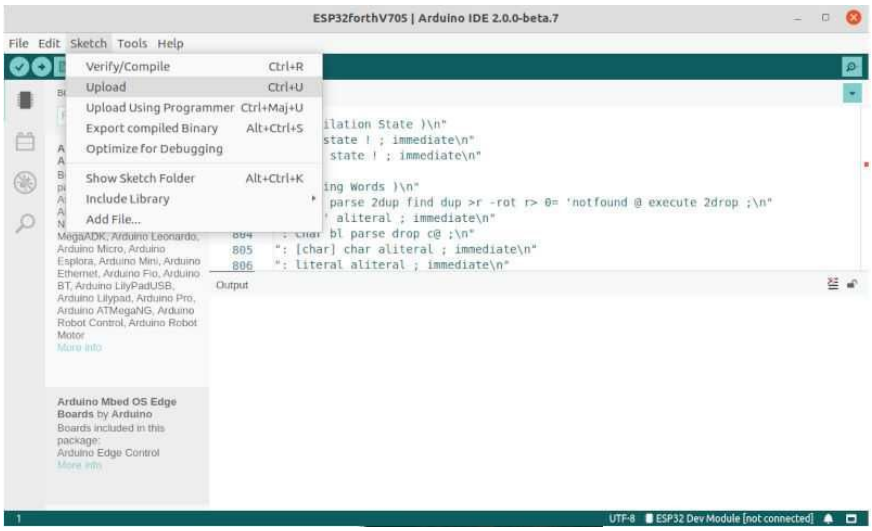
```
-- TOOLS----+-- BOARD    ----+-- ESP32 ------+-- ESP32 Dev Module
            +-- Port: -------+-- COMx
            |
            +-- CPU Frequency ---------+-- 240 Mhz
            +-- Core Debug Level ------+-- None
```

```
+-- 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
```

Iniciar la compilación

Todo lo que queda es compilar ESP32. Cargue el código fuente mediante *File* y *Open*.

Se supone que su placa ESP32 está conectada a un puerto USB. Inicie la compilación haciendo clic en *Sketch* y seleccionando Upload :



Si todo va correctamente, deberías transferir el código binario automáticamente a la placa ESP32. Si la compilación se realiza sin errores, pero hay un error de transferencia, vuelva a compilar el archivo **esp32forth.ino** . En el momento de la transferencia, presione el botón marcado **BOOT** en la placa ESP32. Esto debería hacer que la tarjeta esté disponible para transferir el código binario ESP32forth.

Instalación y configuración de ARDUINO IDE en vídeo:

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

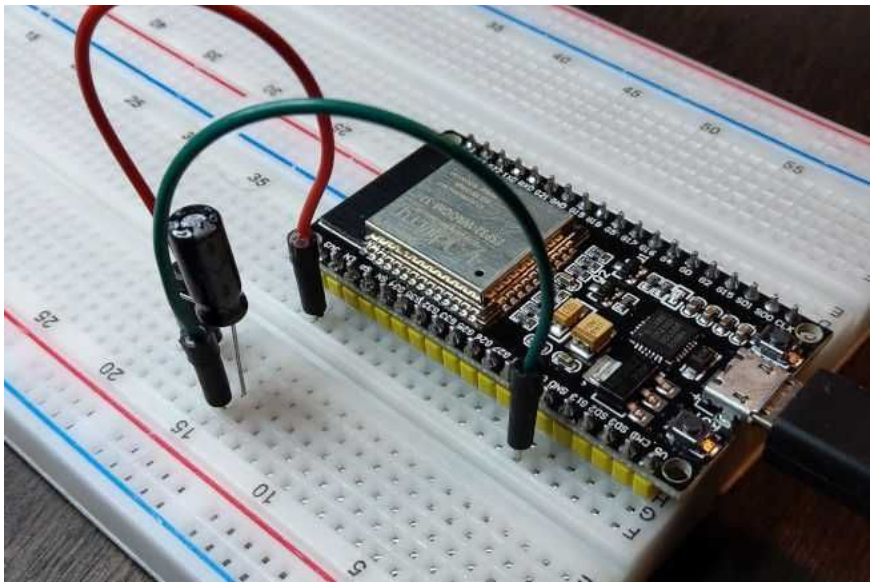
Solucionar el error de conexión de carga

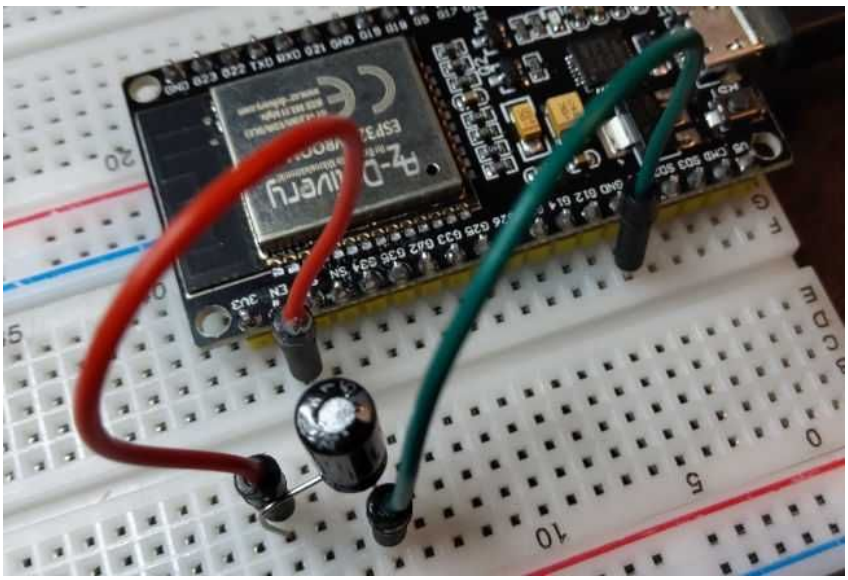
Aprenda cómo solucionar el error fatal que ocurrió: "Failed to connect to ESP32: Timed out waiting for packet header" al intentar cargar un nuevo código a su tarjeta ESP32 de una vez por todas.

Algunas placas de desarrollo ESP32 (lea Las mejores placas ESP32) no ingresan al modo flash/carga automáticamente al descargar código nuevo.

Esto significa que cuando intentas cargar un nuevo boceto en tu placa ESP32, ARDUINO IDE no se conecta a tu placa y recibes el siguiente mensaje de error:

Para hacer que la placa ESP32 cambie automáticamente al modo flash/descarga, podemos conectar un condensador electrolítico de 10uF entre el pin EN y GND:





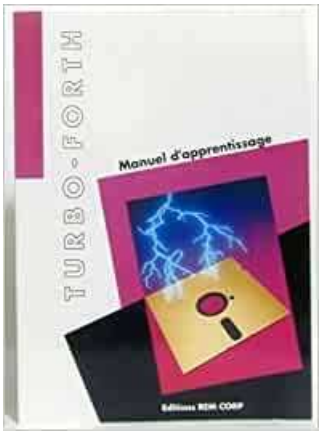
Esta manipulación sólo es necesaria si estás en la fase de carga de ESP32 adelante desde ARDUINO IDE. Una vez que ESP32forth está instalado en la placa ESP32, el uso de este condensador ya no es necesario.

¿Por qué programar en lenguaje FORTH en ESP32?

Preámbulo

Llevo programando en FORTH desde 1983. Dejé de programar en FORTH en 1996. Pero nunca he dejado de seguir la evolución de este lenguaje. Reanudé la programación en 2019 en ARDUINO con FlashForth y luego ESP32forth.

Soy coautor de varios libros sobre el idioma FORTH:



- 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 Loisitech - 1986)
- TURBO-Forth, manuel d'apprentissage (ed Rem CORP - 1990)
- TURBO-Forth, guide de référence (ed Rem CORP - 1991)

Programar en el lenguaje FORTH siempre fue un hobby hasta que en 1992 me contactó el gerente de una empresa que trabajaba como subcontratista para la industria del automóvil. Tenían inquietudes por el desarrollo de software en lenguaje C. Necesitaban encargar un autómata industrial.

Los dos diseñadores de software de esta empresa programaron en lenguaje C: TURBO-C de Borland para ser precisos. Y su código no podía ser lo suficientemente compacto y rápido como para caber en los 64 kilobytes de memoria RAM. Corría el año 1992 y no existían las ampliaciones de tipo memoria flash. ¡En estos 64 KB de RAM teníamos que meter MS-DOS 3.0 y la aplicación!

Durante un mes, los desarrolladores del lenguaje C habían estado dando vuelta al problema en todas direcciones, incluso aplicando ingeniería inversa con SOURCER (un desensamblador) para eliminar partes no esenciales del código ejecutable.

Analicé el problema que se me presentó. Partiendo de cero, creé, solo, en una semana, un prototipo perfectamente operativo y que cumplía con las especificaciones. Durante tres años, de 1992 a 1995, creé numerosas versiones de esta aplicación que se utilizó en las líneas de montaje de varios fabricantes de automóviles.

Límites entre lenguaje y aplicación

Todos los lenguajes de programación se comparten de la siguiente manera:

- un intérprete y código fuente ejecutable: BASIC, PHP, MySQL, JavaScript, etc... La aplicación está contenida en uno o más archivos que serán interpretados cuando sea necesario. El sistema debe alojar permanentemente al intérprete que ejecuta el código fuente;
- un compilador y/o ensamblador: C, Java, etc. Algunos compiladores generan código nativo, es decir ejecutable específicamente sobre un sistema. Otros, como Java, compilan código ejecutable en una máquina Java virtual.

El lenguaje FORTH es una excepción. Integra:

- un intérprete capaz de ejecutar cualquier palabra en el CUARTO idioma
- un compilador capaz de ampliar el diccionario de CUARTAS palabras

¿Qué es una CUARTA palabra?

Una FORTH palabra designa cualquier expresión de diccionario compuesta por caracteres ASCII y utilizable en interpretación y/o compilación: palabras le permite enumerar todas las palabras en el FORTH diccionario.

Ciertas palabras FORTH solo se pueden usar en la compilación: `if else then` por ejemplo.

Con el lenguaje FORTH, el principio esencial es que no creamos una aplicación. ¡En ADELANTE, ampliamos el diccionario ! Cada nueva palabra que defina será una parte tan importante del diccionario FORTH como todas las palabras predefinidas cuando se inicie FORTH. Ejemplo :

```
: typeToLoRa ( -- )
    0 echo !    \ desactive l'echo d'affichage du terminal
    ['] serial2-type is type
  ;
: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ active l'echo d'affichage du terminal
  ;
```

Creamos dos nuevas palabras: `typeToLoRa` y `typeToTerm` que completarán el diccionario de palabras predefinidas.

¿Una palabra es una función?

Si y no. De hecho, una palabra puede ser una constante, una variable, una función... Aquí, en nuestro ejemplo, la siguiente secuencia:

```
: typeToLoRa ...código... ;
```

tendría su equivalente en lenguaje C:

`void typeToLoRa() { ...código... }`

En FORTH idioma, no hay límite entre el idioma y la aplicación.

En FORTH, como en el lenguaje C, puede utilizar cualquier palabra ya definida en la definición de una nueva palabra.

Sí, pero entonces ¿por qué FORTH en lugar de C?

Estaba esperando esta pregunta.

En lenguaje C, solo se puede acceder a una función a través de la función principal `main()` . Si esta función integra varias funciones adicionales, resulta difícil encontrar un error de parámetro en caso de un mal funcionamiento del programa.

Por el contrario, con FORTH es posible ejecutar - a través del intérprete - cualquier palabra predefinida o definida por usted, sin tener que pasar por la palabra principal del programa.

Se puede acceder inmediatamente al intérprete FORTH en la tarjeta ESP32 a través de un programa tipo terminal y un enlace USB entre la tarjeta ESP32 y la PC.

La compilación de programas escritos en lenguaje FORTH se realiza en la tarjeta ESP32 y no en el PC. No hay carga. Ejemplo :

```
: >gray (n -- n')
    dup 2/ xor \ n' = n xor ( 1 desplazamiento lógico a la derecha )
  ;
```

Esta definición se transmite copiando/pegando en el terminal. El intérprete/compilador FORTH analizará la secuencia y compilará la nueva palabra `>gray` .

En la definición de `>gray` , vemos la secuencia `dup 2/ xor` . Para probar esta secuencia, simplemente escríbala en la terminal. Para ejecutar `>gray` , simplemente escriba esta palabra en la terminal, precedida por el número a transformar.

Lenguaje FORTH comparado con el lenguaje C

Esta es la parte que menos me gusta. No me gusta comparar el lenguaje FORTH con el lenguaje C. Pero como casi todos los desarrolladores usan el lenguaje C, voy a probar el ejercicio.

Aquí hay una prueba con `if()` en lenguaje C:

```
if(j > 13){              // Si tous les bits sont recus
    rc5_ok = 1;          // Le processus de decodage est OK
    detachInterrupt(0);  // Desactiver l'interruption externe (INT0)
    return;
}
```

Pruebe con `if` en el lenguaje FORTH (fragmento de código):

```
var-j @ 13 >        \ Si tous les bits sont recus
    if
        1 rc5_ok !  \ Le processus de decodage est OK
        di          \ Desactiver l'interruption externe (INT0)
        exit
    then
```

Aquí está la inicialización de registros en lenguaje C :

```
void setup() {
  //  Configuration du module Timer1
  TCCR1A = 0;
  TCCR1B = 0;          // Desactive le module Timer1
  TCNT1  = 0;          // Definit valeur préchargement Timer1 sur 0
(reset)
  TIMSK1 = 1;          // activer interruption de debordement Timer1
}
```

La misma definición en CUARTO idioma:

```
: setup ( -- )
  \ Configuration du module Timer1
  0 TCCR1A !
  0 TCCR1B !     \ Desactive le module Timer1
  0 TCNT1  !     \ Définit valeur préchargement Timer1 sur 0 (reset)
  1 TIMSK1 !     \ activer interruption de debordement Timer1
;
```

Qué le permite hacer FORTH en comparación con el lenguaje C

Entendemos que FORTH da acceso inmediatamente a todas las palabras del diccionario, pero no sólo eso. A través del intérprete también accedemos a toda la memoria de la tarjeta ESP32. Conéctese a la placa ARDUINO que tiene instalado FlashForth, luego simplemente escriba:

```
hex here 100 dump
```

Deberías encontrar esto en la pantalla del terminal :

```
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 CA 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 CE 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
```

Esto corresponde al contenido de la memoria flash.

¿Y el lenguaje C no podía hacer eso?

Sí, pero no tan simple e interactivo como en el lenguaje FORTH.

Veamos otro caso que destaca la extraordinaria compacidad del lenguaje FORTH...

Pero ¿por qué una pila en lugar de variables?

La pila es un mecanismo implementado en casi todos los microcontroladores y microprocesadores. Incluso el lenguaje C aprovecha una pila, pero no tienes acceso a ella.

Sólo el lenguaje FORTH brinda acceso completo a la pila de datos. Por ejemplo, para hacer una suma, apilamos dos valores, ejecutamos la suma, mostramos el resultado: `2 5 + .` muestra `7 `.

Es un poco desestabilizador, pero cuando comprendes el mecanismo de la pila de datos, aprecias enormemente su formidable eficiencia.

La pila de datos permite pasar datos entre palabras ADELANTE mucho más rápidamente que procesando variables como en el lenguaje C o cualquier otro lenguaje que use variables.

¿Estás convencido?

Personalmente, dudo que este único capítulo lo convierta irremediablemente a programar en el lenguaje FORTH. Cuando buscas dominar las placas ESP32, tienes dos opciones :

- programar en lenguaje C y utilizar las numerosas bibliotecas disponibles. Pero permanecerá encerrado en las capacidades de estas bibliotecas. Adaptar códigos al lenguaje C requiere conocimientos reales de programación en lenguaje C y dominar la arquitectura de las tarjetas ESP32. Desarrollar programas complejos siempre será un problema.
- prueba la FORTH aventura y explora un mundo nuevo y emocionante. Por supuesto, no será fácil. Necesitará comprender en profundidad la arquitectura de las tarjetas ESP32, los registros y las banderas de registro. A cambio, tendrás acceso a una programación perfectamente adaptada a tus proyectos.

¿Hay alguna solicitud profesional escrita en FORTH?

¡Oh sí! Empezando por el telescopio espacial HUBBLE, algunos de cuyos componentes fueron escritos en lenguaje FORTH.

El TGV ICE alemán (Intercit y Express) utiliza procesadores RTX2000 para controlar motores mediante semiconductores de potencia. El lenguaje de máquina del procesador RTX2000 es el lenguaje FORTH.

Este mismo procesador RTX2000 se utilizó para la sonda Philae que intentó aterrizar en un cometa.

La elección del lenguaje FORTH para aplicaciones profesionales resulta interesante si consideramos cada palabra como una caja negra. Cada palabra debe ser simple, por lo tanto tener una definición bastante corta y depender de pocos parámetros.

Durante la fase de depuración, resulta fácil probar todos los valores posibles procesados por esta palabra. Una vez convertida en perfectamente fiable, esta palabra se convierte en una caja negra, es decir, una función en la que tenemos absoluta confianza en su correcto funcionamiento. De palabra en palabra, es más fácil hacer que un programa complejo sea confiable en FORTH que en cualquier otro lenguaje de programación.

Pero si nos falta rigor, si construimos plantas de gas, también es muy fácil que una aplicación funcione mal, ¡o incluso que falle por completo!

Finalmente, es posible, en FORTH idioma, escribir las palabras que definas en cualquier idioma humano. Sin embargo, los caracteres utilizables están limitados al conjunto de caracteres ASCII entre 33 y 127. Así es como podríamos reescribir simbólicamente las palabras alto y bajo:

```
\ Pin de puerto activo, no cambie otros.
: __/ ( pinmask portadr -- )
    mset
  ;
\ Deshabilite un pin de puerto, no cambie los demás.
: \__ ( pinmask portadr -- )
    mclr
  ;
```

A partir de este momento, para encender el LED, puedes escribir :

```
_O_ __/ \ luces LED
```

¡Sí! ¡ La secuencia `_O_ __/` está en FORTH idioma!

Con ESP32forth, aquí están todos los caracteres a tu disposición que pueden componer una FORTH palabra:

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

Buena programación.

Un verdadero FORTH de 32 bits con ESP32Forth

ESP32Forth es un FORTH real de 32 bits. Qué significa eso ?

El lenguaje FORTH favorece la manipulación de valores enteros. Estos valores pueden ser valores literales, direcciones de memoria, contenidos de registros, etc.

Valores en la pila de datos

Cuando se inicia ESP32Forth, el intérprete FORTH está disponible. Si ingresa cualquier número, se colocará en la pila como un entero de 32 bits:

```
35
```

Si apilamos otro valor, también se apilará. El valor anterior será empujado hacia abajo una posición:

```
45
```

Para sumar estos dos valores, usamos una palabra, aquí `+` :

```
+
```

Nuestros dos valores enteros de 32 bits se suman y el resultado se coloca en la pila. Para mostrar este resultado, usaremos la palabra `.` :

```
. \ mostrar 80
```

En el lenguaje FORTH podemos concentrar todas estas operaciones en una sola línea:

```
35 45 +. \ mostrar 80
```

A diferencia del lenguaje C, no definimos un tipo **int8** , **int16** o **int32** .

Con ESP32Forth, un carácter ASCII será designado por un entero de 32 bits, pero cuyo valor estará acotado [32..256[. Ejemplo :

```
67 emit         \ mostrar C
```

Valores en la memoria

ESP32Forth le permite definir constantes y variables. Su contenido siempre estará en formato de 32 bits. Pero hay situaciones en las que eso no necesariamente nos conviene. Tomemos un ejemplo sencillo: definamos un alfabeto en código Morse. Sólo necesitamos unos pocos bytes:

- uno para definir el número de signos del código morse

- uno o más bytes por cada letra del código Morse

```
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,
```

Aquí definimos solo 3 palabras, `mA` , `mB` y `mC` . En cada palabra se almacenan varios bytes. La pregunta es: ¿cómo recuperaremos la información contenida en estas palabras?

La ejecución de una de estas palabras deposita un valor de 32 bits, valor que corresponde a la dirección de memoria donde almacenamos nuestra información en código Morse. Es la palabra `c@` la que usaremos para extraer el código Morse de cada letra:

```
mA c@ .  \ muestra 2
mB c@ .  \ muestra 4
```

El primer byte extraído así se utilizará para gestionar un bucle para mostrar el código Morse de una letra:

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

Hay muchos ejemplos ciertamente más elegantes. Aquí, es para mostrar una forma de manipular valores de 8 bits, nuestros bytes, mientras usamos estos bytes en una pila de 32 bits.

Procesamiento de textos dependiendo del tamaño o tipo de datos.

En todos los demás idiomas tenemos una palabra genérica, como `echo` (en PHP) que muestra cualquier tipo de datos. Ya sea un número entero, real o una cadena, siempre usamos la misma palabra. Ejemplo en lenguaje PHP:

```
$bread = "Pain cuit";
```

```
$price = 2.30;
echo $bread . " : " . $price;
// affiche   Pain cuit: 2.30
```

¡Para todos los programadores, esta forma de hacer las cosas es EL ESTÁNDAR! Entonces, ¿cómo haría FORTH este ejemplo en PHP?

```
: pain s" Pain cuit" ;
: prix s" 2.30" ;
pain type   s" : " type    prix type
\ affiche   Pain cuit: 2.30
```

Aquí, el `tipo de palabra` nos dice que acabamos de procesar una cadena de caracteres.

Cuando PHP (o cualquier otro lenguaje) tiene una función genérica y un analizador, FORTH lo compensa con un único tipo de datos, pero con métodos de procesamiento adaptados que nos informan sobre la naturaleza de los datos procesados.

Aquí hay un caso absolutamente trivial para FORTH, que muestra una cantidad de segundos en formato HH:MM:SS:

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

Me encanta este ejemplo porque, hasta la fecha, **NINGÚN OTRO LENGUAJE DE PROGRAMACIÓN** es capaz de realizar esta conversión HH:MM:SS de forma tan elegante y concisa.

Lo has entendido, el secreto de FORTH está en su vocabulario.

Conclusión

FORTH no tiene tipificación de datos. Todos los datos pasan a través de una pila de datos. ¡Cada posición en la pila es SIEMPRE un entero de 32 bits!

Eso es todo lo que hay que saber.

Los puristas de los lenguajes hiperestructurados y prolijos, como C o Java, ciertamente gritarán herejía. Y aquí me permitiré responderlas: ¿ por qué necesitas escribir tus datos?

Porque es en esa simplicidad donde reside el poder de FORTH: una única pila de datos con un formato sin tipo y operaciones muy sencillas.

Y les voy a mostrar lo que muchos otros lenguajes de programación no pueden hacer: definir nuevas palabras de definición:

```
: 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    \ display   .- -... -.-.
```

Aquí, la palabra **`morse:`** se ha convertido en una palabra de definición, del mismo modo que **`constante`** o **`variable`** …

Porque FORTH es más que un lenguaje de programación. Es un metalenguaje, es decir un lenguaje para construir tu propio lenguaje de programación....

Diccionario / Pila / Variables / Constantes

Ampliar diccionario

Forth pertenece a la clase de lenguajes interpretativos tejidos. Esto significa que puede interpretar comandos escritos en la consola, así como compilar nuevas subrutinas y programas.

El compilador Forth es parte del lenguaje y se utilizan palabras especiales para crear nuevas entradas de diccionario (es decir, palabras). Los más importantes son `:` (iniciar una nueva definición) y `;` (termina la definición). Probemos esto escribiendo:

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

¿Lo que pasó? La acción de: es crear una nueva entrada de diccionario llamada `*+` y cambiar del modo de interpretación al modo de compilación. En modo de compilación, el intérprete busca palabras y, en lugar de ejecutarlas, instala punteros a su código. Si el texto es un número, en lugar de colocarlo en la pila, ESP32forth construye el número en el espacio del diccionario asignado para la nueva palabra, siguiendo un código especial que coloca el número almacenado en la pila cada vez que se ejecuta la palabra. La acción de ejecución de `*+` es por tanto ejecutar secuencialmente las palabras previamente definidas `*` y `+` .

La palabra `;` es especial. Es una palabra inmediata y siempre se ejecuta, incluso si el sistema está en modo compilación. ¿Qué hace `?` es doble. Primero, instala código que devuelve el control al siguiente nivel externo del intérprete y, segundo, regresa del modo de compilación al modo de interpretación.

Ahora prueba tu nueva palabra:

```
decimal 5 6 7 *+ . \ muestra 47 ok<#,ram>
```

Este ejemplo ilustra dos actividades de trabajo principales en Forth: agregar una nueva palabra al diccionario y probarla tan pronto como se haya definido.

Gestión de diccionarios

La palabra `forget` seguida de la palabra a eliminar eliminará todas las entradas del diccionario que haya realizado desde esa palabra:

```
: test1 ;
: test2 ;
: test3 ;
forget test2  \ borrar test2 y test3 del diccionario
```

Pilas y notación polaca inversa

Forth tiene una pila explícitamente visible que se utiliza para pasar números entre palabras (comandos). Usar Forth efectivamente te obliga a pensar en términos de la pila. Esto puede resultar difícil al principio, pero como todo, se vuelve mucho más fácil con la práctica.

En FORTH, la pila es análoga a una pila de cartas con números escritos en ellas. Los números siempre se agregan en la parte superior de la pila y se eliminan de la parte superior de la pila. ESP32forth integra dos pilas: la pila de parámetros y la pila de retroalimentación, cada una de las cuales consta de una cantidad de celdas que pueden contener números de 16 bits.

La FORTH línea de entrada:

```
decimal 2 5 73 -16
```

deja la pila de parámetros como está

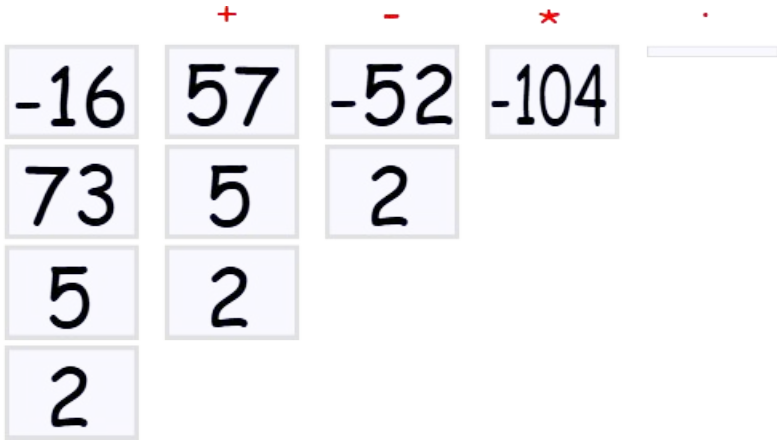
Célula	contenido	comentario
0	-dieciséis	(TOS) Arriba a la derecha
1	73	(NOS) El siguiente en la pila
2	5	
3	2	

Normalmente usaremos numeración relativa de base cero en estructuras de datos Forth, como pilas, matrices y tablas. Tenga en cuenta que cuando se ingresa una secuencia de números de esta manera, el número más a la derecha se convierte en *TOS* y el número más a la izquierda está en la parte inferior de la pila.

Supongamos que seguimos la línea de entrada original con la línea

```
+ - * .
```

Las operaciones producirían operaciones de pila sucesivas:



Después de las dos líneas, la consola muestra:

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

Tenga en cuenta que ESP32 en adelante muestra convenientemente los elementos de la pila al interpretar cada línea y que el valor de -16 se muestra como un entero sin signo de 32 bits. Además, la palabra `.` consume el valor de datos -104, dejando la pila vacía. Si ejecutamos. en la pila ahora vacía, el intérprete externo aborta con un error de puntero de pila STACK UNDERFLOW ERROR.

La notación de programación donde los operandos aparecen primero, seguidos por los operadores se llama notación polaca inversa (RPN).

Manejo de la pila de parámetros

Al ser un sistema basado en pilas, ESP32forth debe proporcionar formas de poner números en la pila, eliminarlos y reorganizar su orden. Ya hemos visto que podemos poner números en la pila simplemente escribiéndolos. También podemos integrar números en la definición de una CUARTA palabra.

La palabra **`drop`** limina un número de la parte superior de la pila y coloca así el siguiente en la parte superior. La palabra **`swap`** intercambia los 2 primeros números. **`dup`** copia el número en la parte superior, empujando todos los demás números hacia abajo. **`rot`** rota los primeros 3 números. Estas acciones se presentan a continuación.

La pila de retorno y sus usos

Al compilar una nueva palabra, ESP32forth establece vínculos entre la palabra que llama y las palabras previamente definidas que serán invocadas por la ejecución de la nueva palabra. Este mecanismo de vinculación, en tiempo de ejecución, utiliza rstack. La dirección de la siguiente palabra que se invocará se coloca en la pila trasera para que cuando la palabra actual haya terminado de ejecutarse, el sistema sepa dónde pasar a la siguiente palabra. Dado que las palabras se pueden anidar, debe haber una pila de estas direcciones de retorno.

Además de servir como reserva de direcciones de retorno, el usuario también puede almacenar y recuperar de la pila de retorno, pero esto debe hacerse con cuidado porque la pila de retorno es esencial para la ejecución del programa. Si utiliza la batería de retorno para almacenamiento temporal, debe devolverla a su estado original; de lo contrario, es probable que bloquee el sistema ESP32forth. A pesar del peligro, hay ocasiones en las que usar backstack como almacenamiento temporal puede hacer que su código sea menos complejo.

Para almacenar en la pila, use `>r` para mover la parte superior de la pila de parámetros a la parte superior de la pila de retorno. Para recuperar un valor, `r>` mueve el valor superior de la pila nuevamente a la parte superior de la pila de parámetros. Para simplemente eliminar un valor de la parte superior de la pila, existe la palabra `rdrop` . La palabra `r@` copia la parte superior de la pila nuevamente en la pila de parámetros.

Uso de memoria

En ESP32 en adelante, los números de 32 bits se recuperan de la memoria a la pila mediante la palabra `@` (fetch) y se almacenan desde arriba en la memoria mediante la palabra `.` (ciego). `@` espera una dirección en la pila y reemplaza la dirección con su contenido. `!` espera un número y una dirección para almacenarlo. Coloca el número en la ubicación de memoria a la que hace referencia la dirección, consumiendo ambos parámetros en el proceso.

Los números sin signo que representan valores de 8 bits (bytes) se pueden colocar en caracteres del tamaño de un carácter. celdas de memoria usando `c@` y `c!` .

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

Variables

Una variable es una ubicación con nombre en la memoria que puede almacenar un número, como el resultado intermedio de un cálculo, fuera de la pila. Por ejemplo:

```
variable x
```

crea una ubicación de almacenamiento llamada `x` , que se ejecuta dejando la dirección de su ubicación de almacenamiento en la parte superior de la pila:

```
X .       \ muestra la dirección
```

Luego podremos recoger o almacenar en esta dirección:

```
variable   x
```

```
3 x !
x @ .     \ muestra: 3
```

Constantes

Una constante es un número que no desea cambiar mientras se ejecuta un programa. El resultado de ejecutar la palabra asociada a una constante es el valor de los datos que quedan en la pila.

```
\ define los pines VSPI
19 constant VSPI_MISO
23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS

\ establece la frecuencia del puerto SPI
4000000 constant SPI_FREQ

\ seleccionar vocabulario SPI
only FORTH  SPI also

\ inicializa el puerto SPI
: init.VSPI ( -- )
    VSPI_CS OUTPUT pinMode
    VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
    SPI_FREQ SPI.setFrequency
  ;
```

Valores pseudoconstantes

Un valor definido con valor es un tipo híbrido de variable y constante. Establecemos e inicializamos un valor y se invoca como lo haríamos con una constante. También podemos cambiar un valor como podemos cambiar una variable.

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

La palabra **`to`** también funciona en definiciones de palabras, reemplazando el valor que le sigue con lo que esté actualmente en la parte superior de la pila. Debe tener cuidado de que **`a`** vaya seguido de un valor definido por **`value`** y no de otra cosa.

Herramientas básicas para la asignación de memoria.

Las palabras **`create`** y **`allot`** son las herramientas básicas para reservar espacio en la memoria y colocarle una etiqueta. Por ejemplo, la siguiente transcripción muestra una nueva entrada del diccionario de **`graphic-array`** :

```
create graphic-array ( --- addr )
```

```
    %00000000 c,
    %00000010 c,
    %00000100 c,
    %00001000 c,
    %00010000 c,
    %00100000 c,
    %01000000 c,
    %10000000 c,
```

Cuando se ejecuta, la palabra **`graphic-array`** insertará la dirección de la primera entrada.

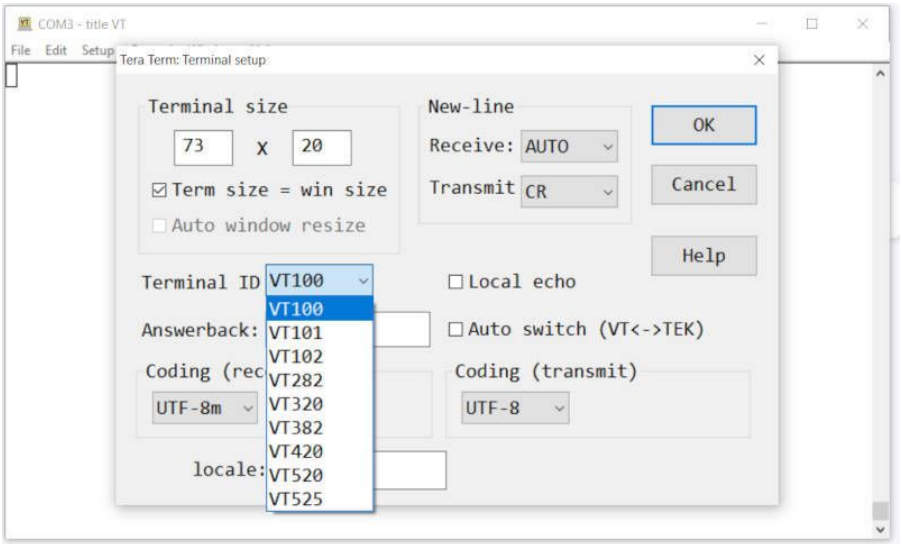
Ahora podemos acceder a la memoria asignada a la **`graphic-array`** usando las palabras de búsqueda y almacenamiento explicadas anteriormente. Para calcular la dirección del tercer byte asignado a **`graphic-array`** podemos escribir **`graphic-array 2 +`** , recordando que los índices comienzan en 0.

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

Colores de texto y posición de visualización en el terminal

Codificación ANSI de terminales.

Si utiliza software de terminal para comunicarse con ESP32 en adelante, es muy probable que este terminal emule un terminal tipo VT o equivalente. Aquí, TeraTerm configurado para emular un terminal VT100:



Estos terminales cuentan con dos características interesantes:

- colorear el fondo de la página y el texto que se mostrará
- posicionar el cursor de visualización

Ambas funciones están controladas por secuencias ESC (escape). Así es como se definen las palabras **bg** y **fg** en ESP32 en adelante:

orth 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" ;
```

Palabra **normal** anula las secuencias de color definidas por **bg** y **fg** .

La palabra **page** borra la pantalla del terminal y coloca el cursor en la esquina superior izquierda de la pantalla.

Coloración de texto

Veamos primero cómo colorear el texto:

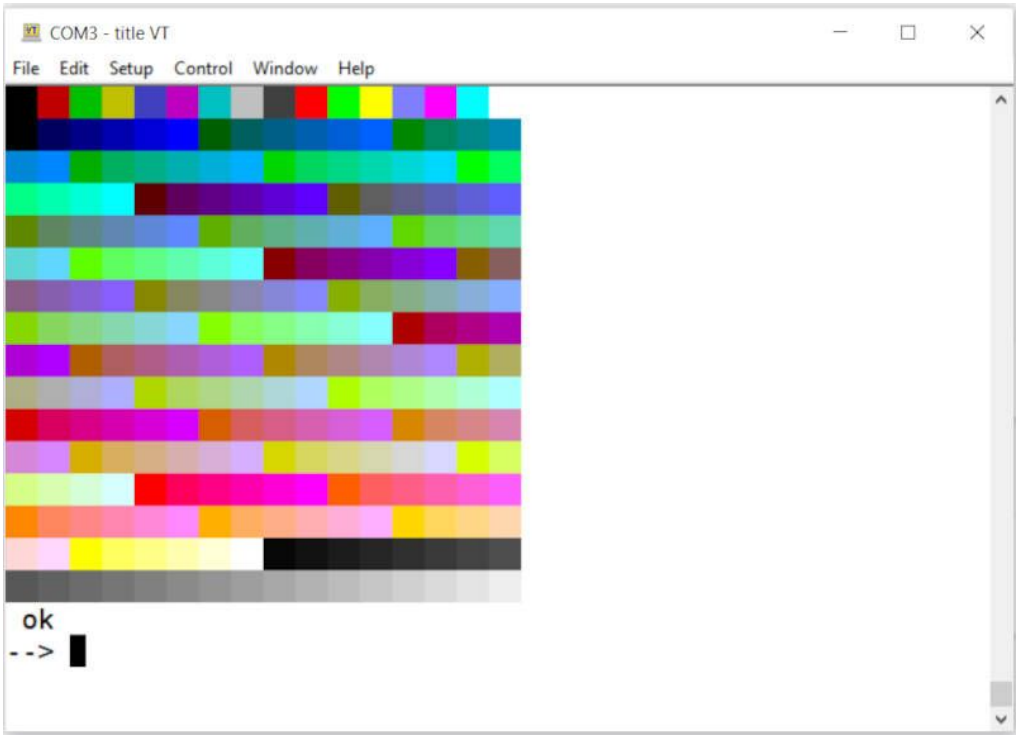
```
: testFG ( -- )
    page
    16 0 do
        16 0 do
            j 16 * i + fg
            ." X"
        loop
        cr
    loop
    normal
  ;
```

Al ejecutar **testFG** se muestra esto:

Para probar los colores de fondo, procederemos de la siguiente manera:

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

Al ejecutar **testBG** se muestra esto:

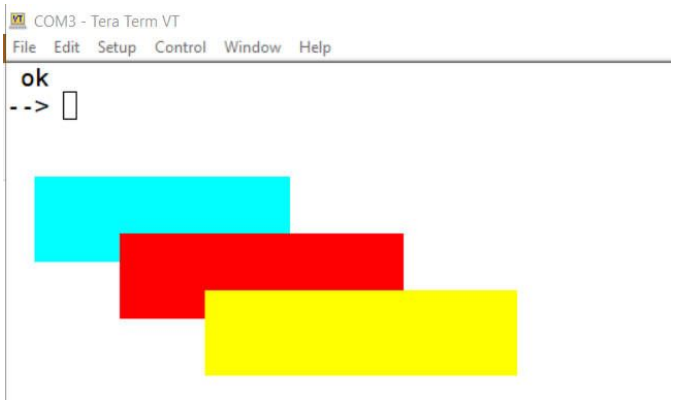


Posición de visualización

El terminal es la solución más sencilla para comunicarse con ESP32forth. Con secuencias de escape ANSI es fácil mejorar la presentación de datos.

```
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
  ;
```

Ejecutar `3boxes` muestra esto:



Ahora está equipado para crear interfaces simples y efectivas que permitan la interacción con las definiciones FORTH compiladas por ESP32forth.

Variables locales con ESP32Forth

Introducción

El lenguaje FORTH procesa datos principalmente a través de la pila de datos. Este mecanismo muy simple ofrece un rendimiento inigualable. Por el contrario, seguir el flujo de datos puede volverse complejo rápidamente. Las variables locales ofrecen una alternativa interesante.

El comentario de la pila falsa

Si sigues los diferentes ejemplos FORTH, habrás notado los comentarios de la pila enmarcados por `(` y `)` . Ejemplo:

```
\ suma dos valores sin signo, deja la suma y la lleva a la pila
: um+ ( u1 u2 -- sum carry )
\ aquí la definición
  ;
```

Aquí, el comentario `(u1 u2 - sum carry)` no tiene absolutamente ninguna acción sobre el resto del código FORTH. Esto es puro comentario.

Al preparar una definición compleja, la solución es utilizar variables locales enmarcadas por `{` y `}` . Ejemplo:

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

Definimos cuatro variables locales `a b c` y `d` .

Las palabras `{` y `}` se parecen a las palabras `(` y `)` pero no tienen el mismo efecto en absoluto. Los códigos colocados entre `{` y `}` son variables locales. La única restricción: no utilice nombres de variables que puedan ser palabras FORTH del diccionario FORTH. También podríamos haber escrito nuestro ejemplo así:

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

Cada variable tomará el valor de la pila de datos en el orden de su depósito en la pila de datos. aquí, 1 entra en `varA` , 2 en `varB` , etc.:

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

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

Nuestro comentario de pila falsa se puede completar así:

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

Los caracteres siguientes `--` tienen ningún efecto. El único punto es hacer que nuestro comentario falso parezca un comentario de pila real.

Acción sobre variables locales.

Las variables locales actúan exactamente como pseudovariables definidas por valor. Ejemplo:

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

Tiene el mismo efecto que este:

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

En este ejemplo, var se define explícitamente por valor.

Asignamos un valor a una variable local con la palabra `to` o `+to` para incrementar el contenido de una variable local. En este ejemplo, agregamos una variable local `result` inicializada a cero en el código de nuestra palabra:

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

¿No es más legible que esto?

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

Aquí hay un ejemplo final, la definición de la palabra **um+** que suma dos enteros sin signo y deja la suma y el valor de desbordamiento de esta suma en la pila de datos:

```
\ suma dos enteros sin signo, deja la suma y la lleva a la pila
: 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
  ;
```

Aquí hay un ejemplo más complejo, reescribiendo **DUMP** usando variables locales:

```
\ variables locales en DUMP:
\ START_ADDR    \ primera dirección para el volcado
\ END_ADDR      \ última dirección para el volcado
\ 0START_ADDR   \ primera dirección del bucle en el volcado
\ LINES         \ número de líneas para el bucle de volcado
\ myBASE        \ base numérica actual
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 }             \ store latest address to dump
    swap { START_ADDR }             \ store START address to dump
    START_ADDR 16 / 16 * { 0START_ADDR } \ calc. addr for loop start
    16 / 1+ { LINES }
    base @ { myBASE }               \ save current base
    hex
    \ outer loop
    LINES 0 do
        0START_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
            0START_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
```

El uso de variables locales simplifica enormemente la manipulación de datos en pilas. El código es más legible. Tenga en cuenta que no es necesario declarar previamente estas variables locales, basta con designarlas al utilizarlas, por ejemplo: **`base @ { myBASE }`** .

ADVERTENCIA: si utiliza variables locales en una definición, no utilice más las palabras **`>r`** y **`r>`** , de lo contrario corre el riesgo de alterar la gestión de las variables locales. Basta con mirar la descompilación de esta versión de **`DUMP`** para comprender el motivo de esta advertencia:

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

Estructuras de datos para ESP32forth

Preámbulo

ESP32forth es una versión de 32 bits del lenguaje FORTH. Quienes han practicado FORTH desde sus inicios han programado con versiones de 16 bits. Este tamaño de datos está determinado por el tamaño de los elementos depositados en la pila de datos. Para conocer el tamaño en bytes de los elementos, debes ejecutar la palabra celda. Ejecutando esta palabra para ESP32 en adelante:

```
cell .  \ muestra 4
```

El valor 4 significa que el tamaño de los elementos colocados en la pila de datos es de 4 bytes, o 4x8 bits = 32 bits.

Con una versión FORTH de 16 bits, la celda apilará el valor 2. Del mismo modo, si usa una versión de 64 bits, la celda apilará el valor 8.

Tablas en FORTH

Comencemos con estructuras bastante simples: tablas. Sólo discutiremos matrices uni o bidimensionales.

Matriz de datos unidimensional de 32 bits

Este es el tipo de mesa más simple. Para crear una tabla de este tipo utilizamos la palabra `create` seguida del nombre de la tabla a crear:

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

En esta tabla almacenamos 6 valores: 34, 37....12. Para recuperar un valor, simplemente use la palabra `@` incrementando la dirección apilada por `temperatures` con el desplazamiento deseado:

```
temperatures         \ dirección de pila
    0 cell *         \ calcular desplazamiento 0
    +                \ agregar desplazamiento a la dirección
    @ .              \ muestra 34

temperatures         \ dirección de pila
    1 cell *         \ calcula el desplazamiento 1
    +                \ agregar desplazamiento a la dirección
    @ .              \ muestra 37
```

Podemos factorizar el código de acceso al valor deseado definiendo una palabra que calculará esta dirección:

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

Notarás que para n valores almacenados en esta tabla, aquí 6 valores, el índice de acceso siempre debe estar en el intervalo [0..n-1].

Palabras de definición de tabla

A continuación se explica cómo crear una definición de palabra de matrices de enteros unidimensionales:

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

En nuestro ejemplo almacenamos 6 valores entre 0 y 255. Es fácil crear una variante de `matriz` para gestionar nuestros datos de una forma más compacta:

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

Con esta variante, los mismos valores se almacenan en cuatro veces menos espacio de memoria.

Leer y escribir en una tabla.

Es completamente posible crear una matriz vacía de n elementos y escribir y leer valores en esta matriz:

```
arrayC myCTemps
    6 allot             \ reservar 6 bytes
    0 myCTemps 6 0 fill \ llenar estos 6 bytes con valor 0
```

```
    32 0 myCTemps c!    \ almacena 32 en myCTemps[0]
    25 5 myCTemps c!    \ almacena 25 en myCTemps[5]
    0 myCTemps c@.      \ muestra 32
```

En nuestro ejemplo, la matriz contiene 6 elementos. Con ESP32 en adelante, hay suficiente espacio de memoria para procesar matrices mucho más grandes, con 1.000 o 10.000 elementos, por ejemplo. Es fácil crear tablas multidimensionales. Ejemplo de una matriz bidimensional:

```
63 constant SCR_WIDTH
16 constant SCR_HEIGHT
create mySCREEN
    SCR_WIDTH SCR_HEIGHT * allot            \ reservar 63 * 16 bytes
    mySCREEN SCR_WIDTH SCR_HEIGHT * bl fill
    \ llenar este espacio con 'espacio'
```

Aquí, definimos una tabla bidimensional llamada **mySCREEN** que será una pantalla virtual de 16 filas y 63 columnas.

Simplemente reserva un espacio de memoria que es el producto de las dimensiones X e Y de la tabla a utilizar. Ahora veamos cómo gestionar esta matriz bidimensional:

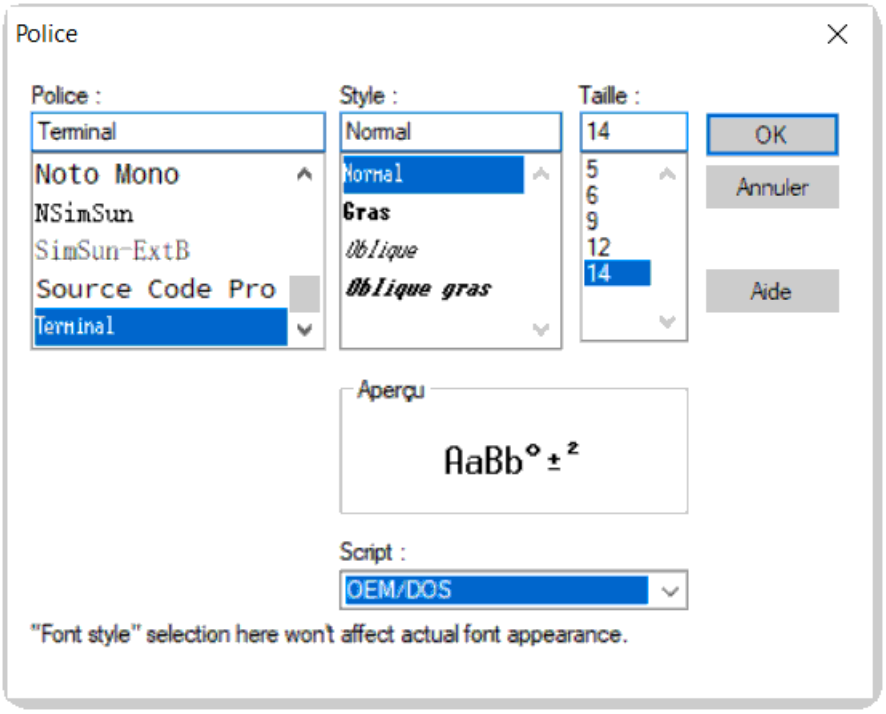
```
: xySCRaddr { x y -- addr }
    SCR_WIDTH y *
    x + mySCREEN +
  ;
: SCR@ ( x y -- c )
    xySCRaddr c@
  ;
: SCR! ( c x y -- )
    xySCRaddr c!
  ;
char X 15 5 SCR!    \ almacena el carácter X en el cuello 15 línea 5
15 5 SCR@ emit      \ visualizaciones
```

Ejemplo práctico de gestión de una pantalla virtual

Antes de continuar en nuestro ejemplo de gestión de una pantalla virtual, veamos cómo modificar el juego de caracteres del terminal TERA TERM y mostrarlo.

Inicie TERA TERM:

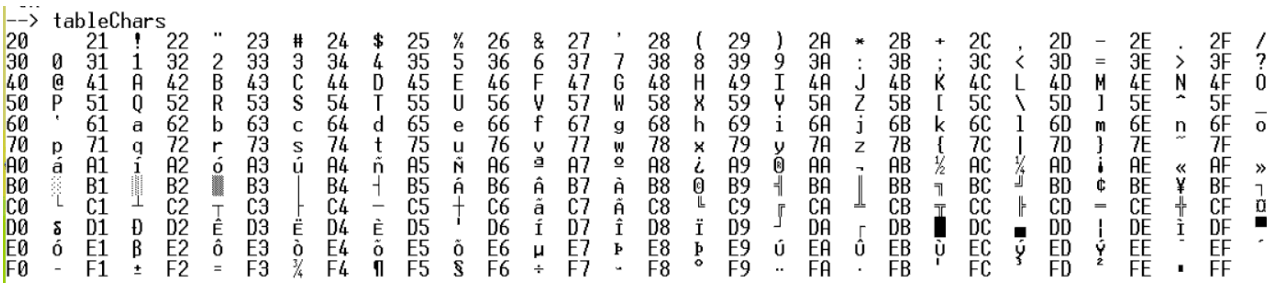
- en la barra de menú, haga clic en *Setup*
- seleccione *Font* y *Font*...
- configure la fuente a continuación:



A continuación se explica cómo mostrar la tabla de caracteres disponibles:

```
: 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
```

Aquí está el resultado de ejecutar `tableChars`:



Estos caracteres son los del conjunto ASCII de MS-DOS. Algunos de estos personajes son semigráficos. Aquí tienes una inserción muy sencilla de uno de estos personajes en nuestra pantalla virtual:

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

Ahora veamos cómo mostrar el contenido de nuestra pantalla virtual. Si consideramos cada línea de la pantalla virtual como una cadena alfanumérica, solo necesitamos definir esta palabra para mostrar una de las líneas de nuestra pantalla virtual:

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

En el camino, crearemos una definición que permita mostrar el mismo carácter n veces:

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

Y ahora, definimos la palabra que nos permitirá mostrar el contenido de nuestra pantalla virtual. Para ver claramente el contenido de esta pantalla virtual, la enmarcamos con caracteres especiales:

```
: 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
  ;
```

Al ejecutar nuestra palabra **`dispScreen`** se muestra esto:

En nuestro ejemplo de pantalla virtual, mostramos que administrar una matriz bidimensional tiene una aplicación concreta. Nuestra pantalla virtual es accesible para escribir y leer. Aquí mostramos nuestra pantalla virtual en la ventana de terminal. Esta pantalla está lejos de ser eficiente. Pero puede ser mucho más rápido en una pantalla OLED real.

Gestión de estructuras complejas.

ESP32 en adelante tiene el vocabulario de estructuras. El contenido de este vocabulario permite definir estructuras de datos complejas.

Aquí hay una estructura de ejemplo trivial:

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

Aquí, definimos la estructura YMDHMS. Esta estructura gestiona los punteros **`>year >month >day >hour >min`** y **`>sec`** .

de la palabra **`YMDHMS`** es inicializar y agrupar los punteros en la estructura compleja. Así es como se utilizan estos consejos:

```
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
```

Hemos definido la palabra **DateTime** que es una tabla simple de 6 celdas consecutivas de 32 bits. El acceso a cada celda se realiza mediante el puntero correspondiente. Podemos redefinir el espacio asignado de nuestra estructura **YMDHMS** usando la palabra **i8** para señalar bytes:

```
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    \ muestra:
\  YEAR: 2022
\ MONTH: 3
\   DAY: 21
\    HH: 22
```

```
\    MM: 36
\    SS: 15
```

En esta estructura cYMDHMS, mantuvimos el año en formato de 32 bits y redujimos todos los demás valores a enteros de 8 bits. Vemos, en el código .cDate, que el uso de punteros permite un fácil acceso a cada elemento de nuestra compleja estructura....

Definición de sprites

Anteriormente definimos una pantalla virtual como una matriz bidimensional. Las dimensiones de esta matriz están definidas por dos constantes. Recordatorio de la definición de esta pantalla virtual:

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

La desventaja de este método de programación es que las dimensiones se definen en constantes y, por tanto, fuera de la tabla. Sería más interesante incrustar las dimensiones de la mesa en la mesa. Para ello definiremos una estructura adaptada a este caso:

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

create myVscreen    \ define una pantalla de 8x32 bytes
    32 c,           \ ancho de compilación
    08 c,           \ altura de compilación
    myVscreen >width  c@
    myVscreen >height c@ * allot
```

Para definir un sprite de software, simplemente compartiremos esta definición:

```
: sprite: ( width height -- )
    create
        swap c, c,  \ compilar ancho y alto
    does>
  ;
2 1 sprite: blackChars
    $db c, $db c,
2 1 sprite: greyChars
    $b2 c, $b2 c,
blackChars >content 2 type   \ muestra el contenido del sprite
blackChars
```

Aquí se explica cómo definir un sprite de 5 x 7 bytes:

```
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,
```

Para mostrar el sprite, desde una posición xy en la ventana de terminal, basta con un simple bucle:

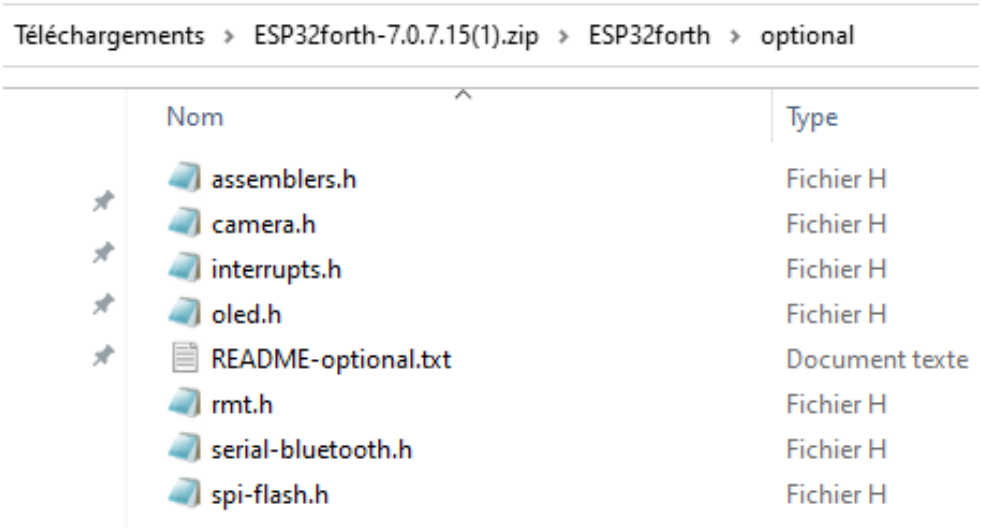
```
: .sprite { xpos ypos sprAddr -- }
    sprAddr >height c@ 0 do
        xpos ypos at-xy
        sprAddr >width c@ i *
        \ calcular el desplazamiento en los datos del sprite
        sprAddr >content +
        \ calcular la dirección real para la línea n en datos de
sprites
        sprAddr >width c@ type  \ línea de visualización
        1 +to ypos              \ incrementar y posición
    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
```

Resultado de mostrar nuestro sprite:

Espero que el contenido de este capítulo te haya dado algunas ideas interesantes que te gustaría compartir...

Instalación de la biblioteca OLED para SSD1306

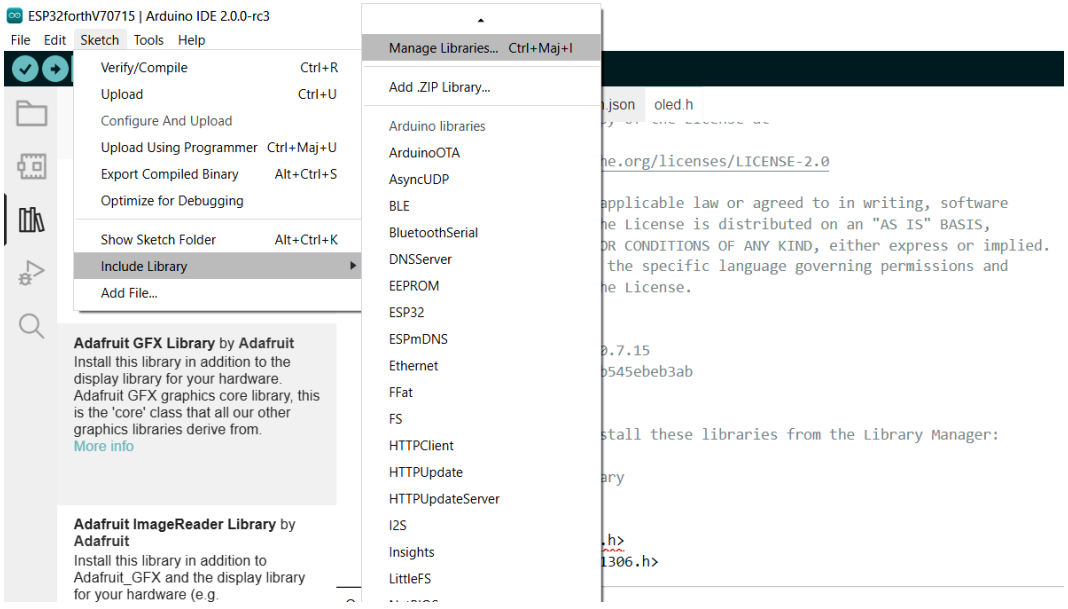
Desde ESP32 en adelante versión 7.0.7.15, las opciones están disponibles en la carpeta **optional**:



Para tener el vocabulario `oled`, copie el archivo **oled.h** a la carpeta que contiene el archivo **ESP32forth.ino**.

Luego inicie ARDUINO IDE y seleccione el archivo **ESP32forth.ino** más reciente.

Si la biblioteca OLED no se ha instalado, en ARDUINO IDE, haga clic en *Sketch* y seleccione *Include*, luego seleccione *Manage Libraries*.



En la barra lateral izquierda, busque la biblioteca **Adafruit SSD1306 by Adafruit**.

Vous pouvez maintenant lancer la compilation du croquis en cliquant sur *Sketch* et en sélectionnant *Upload*.

Ahora puede comenzar a compilar el boceto haciendo clic en *Sketch* y seleccionando *Upload*.

Una vez que el boceto esté cargado en la placa ESP32, inicie la terminal TeraTerm. Compruebe que el vocabulario **oled** esté presente:

```
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
```

Números reales con ESP32 en adelante

Si probamos la operación `1 3 /` en FORTH idioma, el resultado será 0.

No es sorprendente. Básicamente, ESP32forth solo usa enteros de 32 bits a través de la pila de datos. Los números enteros ofrecen ciertas ventajas:

- velocidad de procesamiento;
- resultado de cálculos sin riesgo de deriva en caso de iteraciones;
- Adecuado para casi todas las situaciones.

Incluso en cálculos trigonométricos podemos utilizar una tabla de números enteros. Simplemente crea una tabla con 90 valores, donde cada valor corresponde al seno de un ángulo, multiplicado por 1000.

Pero los números enteros también tienen límites:

- resultados imposibles para cálculos de división simples, como nuestro ejemplo de 1/3;
- Requiere manipulaciones complejas para aplicar fórmulas físicas.

Desde la versión 7.0.6.5, ESP32 incluye operadores que tratan con números reales.

Los números reales también se llaman números de coma flotante.

Los reales con ESP32 en adelante

Para distinguir los números reales, deben terminar en la letra "e":

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

Es la palabra con `f.` lo que le permite mostrar un número real ubicado en la parte superior de la pila de reales.

Precisión de números reales con ESP32forth

La palabra `set-precision` le permite indicar el número de decimales que se mostrarán después del punto decimal. Veamos esto con la constante `pi` :

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

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

La precisión límite para procesar números reales con ESP32 en adelante es de seis decimales:

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

Si reducimos la precisión de visualización de los números reales por debajo de 6, los cálculos se seguirán realizando con una precisión de 6 decimales.

Constantes y variables reales

Una constante real se define con la palabra **fconstant** :

```
0.693147e fconstante ln2 \ logaritmo natural de 2
```

Una variable real se define con la palabra **fvariable** :

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

ATENCIÓN: todos los números reales pasan por la **pila de números reales** . En el caso de una variable real, sólo la dirección que apunta al valor real pasa a través de la pila de datos.

La palabra **SF!** almacena un valor real en la dirección o variable señalada por su dirección de memoria. La ejecución de una variable real coloca la dirección de memoria en la pila de datos clásica.

La palabra **SF@** apila el valor real al que apunta su dirección de memoria.

Operadores aritméticos en números reales

ESP32Forth tiene cuatro operadores aritméticos **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 también tiene estas palabras:

- **1/F** calcula el inverso de un número real;
- **fsqrt** calcula la raíz cuadrada de un número real.

```
5e 1/F f.       \ mostrar 0.200000      1/5
5e fsqrt f.     \ mostrar 2.236068      sqrt(5)
```

Operadores matemáticos sobre números reales

ESP32forth tiene varios operadores matemáticos:

- **F**** eleva un r_val real a la potencia r_exp
- **FATAN2** calcula el ángulo en radianes a partir de la tangente.
- **FCOS** (r1 -- r2) Calcula el coseno de un ángulo expresado en radianes.
- **FEXP** (ln-r -- r) calcula el real correspondiente a e EXP r
- **FLN** (r -- ln-r) calcula el logaritmo natural de un número real.
- **FSIN** (r1 -- r2) calcula el seno de un ángulo expresado en radianes.
- **FSINCOS** (r1 -- rcos rsin) calcula el coseno y el seno de un ángulo expresado en radianes.

Algunos ejemplos :

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

4.605170e FEXP F.     \ mostrar 100.000018

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

Operadores lógicos en números reales

ESP32forth también le permite realizar pruebas lógicas en datos reales:

- **F0<** (r -- fl) prueba si un número real es menor que cero.
- **F0=** (r -- fl) indica verdadero si el real es cero.
- **f<** (r1 r2 -- fl) fl es verdadero si r1 < r2.
- **f<=** (r1 r2 -- fl) fl es verdadero si r1 <= r2.
- **f<>** (r1 r2 -- fl) fl es verdadero si r1 <> r2.
- **f=** (r1 r2 -- fl) fl es verdadera si r1 = r2.
- **f>** (r1 r2 -- fl) fl es verdadero si r1 > r2.
- **f>=** (r1 r2 -- fl) fl es verdadero si r1 >= r2.

Entero ↔ transformaciones reales

ESP32forth tiene dos palabras para transformar números enteros en reales y viceversa:

- **F>S** (r -- n) convierte un real en un número entero. Deje la parte entera en la pila de datos si el real tiene partes decimales.
- **S>F** (n -- r: r) convierte un número entero en un número real y transfiere este número real a la pila de reales.

Ejemplo :

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

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

El generador de números aleatorios

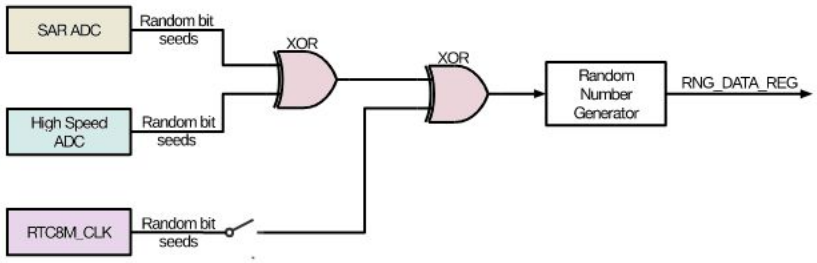
Característica

El generador de números aleatorios genera números aleatorios verdaderos, lo que significa un número aleatorio generado a partir de un proceso físico, en lugar de mediante un algoritmo. Ningún número generado dentro del rango especificado tiene más o menos probabilidad de aparecer que cualquier otro número.

Cada valor de 32 bits que el sistema lee del registro RNG_DATA_REG del generador de números aleatorios es un número aleatorio verdadero. Estos números aleatorios verdaderos se generan en función del ruido térmico en el sistema y del desfase del reloj asíncrono.

El ruido térmico proviene del ADC de alta velocidad, del SAR ADC o de ambos. Siempre que se active el ADC de alta velocidad o el ADC SAR, los flujos de bits se generarán y se introducirán en el generador de números aleatorios a través de una puerta lógica XOR como semillas aleatorias.

Cuando el reloj RTC8M_CLK está habilitado para el núcleo digital, el generador de números aleatorios también tomará muestras de RTC8M_CLK (8 MHz) como una semilla binaria aleatoria. RTC8M_CLK es una fuente de reloj asíncrona y aumenta la entropía del RNG al introducir la metaestabilidad del circuito. Sin embargo, para garantizar la máxima entropía, también se recomienda habilitar siempre una fuente ADC.



Cuando hay ruido del SAR ADC, el generador de números aleatorios se alimenta con una entropía de 2 bits en un ciclo de reloj de RTC8M_CLK (8 MHz), que se genera a partir de un oscilador RC interno (consulte el capítulo Restablecer y reloj para obtener más detalles).). Por tanto, es recomendable leer el registro **RNG_DATA_REG** a una velocidad máxima de 500 kHz para obtener la máxima entropía.

Cuando hay ruido del ADC de alta velocidad, el generador de números aleatorios recibe entropía de 2 bits en un ciclo de reloj APB, que normalmente es de 80 MHz. Por tanto, es recomendable leer el registro **RNG_DATA_REG** a una velocidad máxima de 5 MHz para obtener la máxima entropía.

Se probó una muestra de datos de 2 GB, que se lee del generador de números aleatorios a una frecuencia de 5 MHz con solo el ADC de alta velocidad habilitado, utilizando el conjunto de pruebas Dieharder Random Number (versión 3.31.1). La muestra pasó todas las pruebas.

Procedimiento de programación

Cuando utilice el generador de números aleatorios, asegúrese de que se permita al menos SAR ADC, High Speed ADC o RTC8M_CLK. De lo contrario, se devolverán números pseudoaleatorios.

- SAR ADC se puede activar utilizando el controlador DIG ADC.
- El ADC de alta velocidad se habilita automáticamente cuando los módulos Wi-Fi o Bluetooth están habilitados.
- RTC8M_CLK se habilita configurando el bit RTC_CNTL_DIG_CLK8M_EN en el registro RTC_CNTL_CLK_CONF_REG.

Cuando utilice el generador de números aleatorios, lea el registro **`RNG_DATA_REG`** varias veces hasta que se generen suficientes números aleatorios.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	$3FF75144	RO

```
\ Datos de números aleatorios
$3FF75144 constant RNG_DATA_REG

\ obtener 32 bits aleatorio b=número
: rnd  ( -- x )
    RNG_DATA_REG L@
  ;

\ obtener número aleatorio en el intervalo [0..n-1]
: random ( n -- 0..n-1 )
    rnd swap mod
  ;
```

Función RND en ensamblador XTENSA

Desde la versión 7.0.7.4, ESP32 en adelante tiene un ensamblador XTENSA. Es posible reescribir nuestra **`rnd`** palabra en el ensamblador XTENSA:

```
forth definitions
asm xtensa
$3FF75144 constant RNG_DATA_REG

code myRND ( -- [addr] )
    a1 32           ENTRY,
```

```
    a8 RNG_DATA_REG L32R,        \ a8 = RNG_DATA_REG
    a9 a8 0         L32I.N,      \ a9 = [a8]
    a9           arPUSH,         \ push a9 on stack
                    RETW.N,
end-code
```

Contenido detallado de los vocabularios ESP32forth

ESP32forth proporciona numerosos vocabularios:

- **FORTH** es el vocabulario principal;
- Ciertos vocabularios se utilizan para la mecánica interna de ESP32Forth, como **internals** , **asm...**
- Muchos vocabularios permiten la gestión de puertos o accesorios específicos, como **bluetooth** , **oled** , **spi** , **wifi** , **wire…**

Aquí encontrarás la lista de todas las palabras definidas en estos diferentes vocabularios. Algunas palabras se presentan con un enlace de color:

align es una FORTH palabra ordinaria;

CONSTANT es palabra de definición;

begin marca una estructura de control;

key es una palabra de ejecución diferida;

LED es una palabra definida por **constant** , **variable** o **value** ;

registers marca un vocabulario.

Las palabras del vocabulario **FORTH** se muestran en orden alfabético. Para otros vocabularios, las palabras se presentan en su orden de visualización.

Version v 7.0.7.15

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 1 o bits bit skip advance advance-operand reset reset-operand for-operands operands >printop >inop >next >opmask& bit! mask pattern length demask enmask >>1 odd? high-bit end-code code, code4, code3, code2, code1, callot chere reserve

code-at code-start

bluetooth

SerialBT.new SerialBT.delete SerialBT.begin SerialBT.end SerialBT.available
SerialBT.readBytes SerialBT.write SerialBT.flush SerialBT.hasClient
SerialBT.enableSSP SerialBT.setPin SerialBT.unpairDevice SerialBT.connect
SerialBT.connectAddr SerialBT.disconnect SerialBT.connected
SerialBT.isReady bluetooth-builtins

editor

a r d e wipe p n l

ESP

getHeapSize getFreeHeap getMaxAllocHeap getChipModel getChipCores getFlashChipSize
getCpuFreqMHz getSketchSize deepSleep getEfuseMac esp_log_level_set ESP-builtins

httpd

notfound-response bad-response ok-response response send path method **hasHeader**
handleClient read-headers completed? body content-length header crnl= eat
skipover skipto in@<> end< **goal#** **goal** strcase= upper server client-cr client-emit
client-read client-type **client-len** client httpd-port clientfd **sockfd** body-read
body-1st-read body-chunk body-chunk-size **chunk-filled** chunk **chunk-size**
max-connections

insides

run normal-mode raw-mode step ground handle-key quit-edit save load backspace
delete handle-esc insert update crtype cremit ndown down nup up caret length
capacity text start-size fileh filename# filename max-path

internals

assembler-source xtensa-assembler-source MALLOC SYSFREE REALLOC heap_caps_malloc
heap_caps_free heap_caps_realloc heap_caps_get_total_size heap_caps_get_free_size
heap_caps_get_minimum_free_size heap_caps_get_largest_free_block RAW-YIELD
RAW-TERMINATE READDIR CALLCODE CALL0 CALL1 CALL2 CALL3 CALL4 CALL5 CALL6
CALL7 CALL8 CALL9 CALL10 CALL11 CALL12 CALL13 CALL14 CALL15 DOFLIT S>FLOAT?
fill32 'heap 'context 'latestxt 'notfound 'heap-start 'heap-size 'stack-cells
'boot 'boot-size 'tib 'argc 'argv 'runner 'throw-handler NOP BRANCH 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 OledCirc OledCircF OledRect OledRectF
OledRectR OledRectRF oled-builtins

registers

m@ m!

riscv

C.FSWSP, C.SWSP, C.FSDSP, C.ADD, C.JALR, C.EBREAK, C.MV, C.JR, C.FLWSP,
C.LWSP, C.FLDSP, C.SLLI, BNEZ, BEQZ, C.J, C.ADDW, C.SUBW, C.AND, C.OR,
C.XOR, C.SUB, C.ANDI, C.SRAI, C.SRLI, C.LUI, C.LI, C.JAL, C.ADDI, C.NOP,
C.FSW, C.SW, C.FSD, C.FLW, C.LW, C.FLD, C.ADDI4SP, C.ILL, EBREAK, ECALL,
AND, OR, SRA, SRL, XOR, SLTU, SLT, SLL, SUB, ADD, SRAI, SRLI, SLLI, ANDI,

ORI, XORI, SLTIU, SLTI, ADDI, SW, SH, SB, LHU, LBU, LW, LH, LB, BGEU, BLTU, BGE, BLT, BNE, BEQ, JALR, JAL, AUIPC, LUI, J-TYPE U-TYPE B-TYPE S-TYPE I-TYPE R-TYPE rs2' rs2#' rs2 rs2# rs1' rs1#' rs1 rs1# rd' rd#' rd rd# offset ofs ofs. >ofs iiii i numeric register' reg'. reg>reg' register reg. nop x31 x30 x29 x28 x27 x26 x25 x24 x23 x22 x21 x20 x19 x18 x17 x16 x15 x14 x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 zero

rtos

vTaskDelete xTaskCreatePinnedToCore xPortGetCoreID rtos-builtins

SD

SD.begin SD.beginFull SD.beginDefaults SD.end SD.cardType SD.totalBytes SD.usedBytes SD-builtins

SD_MMC

SD_MMC.begin SD_MMC.beginFull SD_MMC.beginDefaults SD_MMC.end SD_MMC.cardType SD_MMC.totalBytes SD_MMC.usedBytes SD_MMC-builtins

Serial

Serial.begin Serial.end Serial.available Serial.readBytes Serial.write Serial.flush Serial.setDebugOutput Serial2.begin Serial2.end Serial2.available Serial2.readBytes Serial2.write Serial2.flush Serial2.setDebugOutput serial-builtins

sockets

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

spi

SPI.begin SPI.end SPI.setHwCs SPI.setBitOrder SPI.setDataMode SPI.setFrequency SPI.setClockDivider SPI.getClockDivider SPI.transfer SPI.transfer8 SPI.transfer16 SPI.transfer32 SPI.transferBytes SPI.transferBits SPI.write SPI.write16 SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern SPI-builtins

SPIFFS

SPIFFS.begin SPIFFS.end SPIFFS.format SPIFFS.totalBytes SPIFFS.usedBytes SPIFFS-builtins

streams

stream> >stream stream>ch ch>stream wait-read wait-write empty? full? stream# >offset >read >write **stream**

structures

field struct-align align-by last-struct struct long ptr i64 i32 i16 i8
typer last-align

tasks

.tasks main-task task-list

telnetd

server broker-connection wait-for-connection connection telnet-key telnet-type
telnet-emit broker client-len client telnet-port clientfd sockfd

visual

edit insides

web-interface

server webserver-task do-serve handle1 serve-key serve-type handle-input handle-index out-string output-stream input-stream out-size webserver index-html index-html#

WiFi

Wire.begin Wire.setClock Wire.getClock Wire.setTimeout Wire.getTimeout Wire.beginTransmission Wire.endTransmission Wire.requestFrom Wire.write Wire.available Wire.read Wire.peek Wire.flush Wire-builtins

xtensa

WUR, WSR, WITLB, WER, WDTLB, WAITI, SSXU, SSX, SSR, SSL, SSIU, SSI, SSAI,
SSA8L, SSA8B, SRLI, SRL, SRC, SRAI, SRA, SLLI, SLL, SICW, SICT, SEXT, SDCT,
RUR, RSR, RSIL, RFI, ROTW, RITLB1, RITLB0, RER, RDTLB1, RDTLB0, PITLB,
PDTLB, NSAU, NSA, MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULS.DD
MULA.DA.HH, MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULS.DA MULA.AD.HH, MULA.AD.LH,
MULA.AD.HL, MULA.AD.LL, MULS.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL,
MULS.AA MULA.DD.HH.LDINC, MULA.DD.LH.LDINC, MULA.DD.HL.LDINC, MULA.DD.LL.LDINC,
MULA.DD.LDINC MULA.DD.HH.LDDEC, MULA.DD.LH.LDDEC, MULA.DD.HL.LDDEC,
MULA.DD.LL.LDDEC,
MULA.DD.LDDEC MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULA.DD
MULA.DA.HH.LDINC,
MULA.DA.LH.LDINC, MULA.DA.HL.LDINC, MULA.DA.LL.LDINC, MULA.DA.LDINC
MULA.DA.HH.LDDEC,
MULA.DA.LH.LDDEC, MULA.DA.HL.LDDEC, MULA.DA.LL.LDDEC, MULA.DA.LDDEC MULA.DA.HH,
MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULA.DA MULA.AD.HH, MULA.AD.LH, MULA.AD.HL,
MULA.AD.LL, MULA.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL, MULA.AA
MUL16U, MUL16S, MUL.DD.HH, MUL.DD.LH, MUL.DD.HL, MUL.DD.LL, MUL.DD MUL.DA.HH,
MUL.DA.LH, MUL.DA.HL, MUL.DA.LL, MUL.DA MUL.AD.HH, MUL.AD.LH, MUL.AD.HL,
MUL.AD.LL, MUL.AD MUL.AA.HH, MUL.AA.LH, MUL.AA.HL, MUL.AA.LL, MUL.AA MOVT,
MOVSP, MOVT.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,
S32C1I, 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, DPFRO, 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
```

Recursos

En inglés

- **ESP32forth** Página mantenida por Brad NELSON, el creador de ESP32forth. Allí encontrarás todas las versiones (ESP32, Windows, Web, Linux...) https://esp32forth.appspot.com/ESP32forth.html
-

En francés

- **ESP32 Forth** sitio en dos idiomas (francés, inglés) con muchos ejemplos https://esp32.arduino-forth.com/

GitHub

- **Ueforth** Recursos mantenidos por Brad NELSON. Contiene todos los archivos fuente en lenguaje Forth y C para ESP32forth https://github.com/flagxor/ueforth
- **ESP32forth** códigos fuente y documentación para ESP32 en adelante. Recursos mantenidos por Marc PETREMANN https://github.com/MPETREMANN11/ESP32forth

- **ESP32forthStation** recursos mantenidos por Ulrich HOFFMAN. Computadora Forth independiente con computadora de placa única LillyGo TTGO VGA32 y ESP32forth
  https://github.com/uho/ESP32forthStation
- **ESP32Forth** recursos mantenidos por F. J. RUSSO
  https://github.com/FJRusso53/ESP32Forth
- **esp32forth-addons** recursos mantenidos por Peter FORTH
  https://github.com/PeterForth/esp32forth-addons
- **Esp32forth-org** Repositorio de código para miembros de los grupos Forth2020 y ESp32forth
  https://github.com/Esp32forth-org
-

índice léxico

asm.....62
bluetooth.....63
editor.....63
ESP.....63
f.....54
fconstant.....55
FORTH.....61
fvariable.....55
httpd.....63
insides.....63
internals.....63
interrupts.....64
ledc.....64
números aleatorios.....58
oled.....51, 64
pi.....54
random.....59
registers.....64
riscv.....64
rnd.....59
RNG_DATA_REG.....59
rtos.....65
SD.....65
SD_MMC.....65
Serial.....65
set-precision.....54
sockets.....65
spi.....65
SPIFFS.....65
streams.....65
struct.....46
structures.....46, 66
.....48
tasks.....66
telnetd.....66
to.....38
visual.....66
web-interface.....66
WiFi.....66
xtensa.....66
{.....37
}.....37
+to.....38