The great book

for ESP32forth

version 1.0 october 2023

Author

- Marc PETREMANN petremann@arduino-forth.com

Collaborators

- Vaclav POSELT

Contents

Author....1
Collaborators....1

Introduction....3
Translation help....3

Discovery of the ESP32 card....4
Presentation....4
The strong points....4
GPIO inputs/outputs on ESP32....5
ESP32 Peripherals....7

Why program in FORTH language on ESP32?....8
Preamble....8
Boundaries between language and application....8
What is a FORTH word?....9
A word is a function?....9
FORTH language compared to C language....10
What FORTH allows you to do compared to the C language....11
But why a stack rather than variables?....12
Are you convinced?....12
Are there any professional applications written in FORTH?....12

A real 32-bit FORTH with ESP32Forth....15
Values on the data stack....15
Values in memory....15
Word processing depending on data size or type....16
Conclusion....17

Ressources....19
in English....19
In french....19
GitHub....19

Introduction

Since 2019, I manage several websites dedicated to FORTH language development for ARDUINO and ESP32 boards, as well as the eForth web version:

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

These sites are available in two languages, French and English. Every year I pay for hosting the main site **arduino-forth.com**.

It will happen sooner or later – and as late as possible – that I will no longer be able to ensure the sustainability of these sites. The consequence will be that the information disseminated by these sites disappears.

This book is the compilation of content from my websites. It is distributed freely from a Github repository. This method of distribution will allow greater sustainability than websites.

Incidentally, if some readers of these pages wish to contribute, they are welcome:

- to suggest chapters ;
- to report errors or suggest changes;
- to help with the translation...

Translation help

Google Translate allows you to translate texts easily, but with errors. So I'm asking for help to correct the translations.

In practice, I provide the chapters already translated in the LibreOffice format. If you want to help with these translations, your role will simply be to correct and return these translations.

Correcting a chapter takes little time, from one to a few hours.

To contact me : petremann@arduino-forth.com

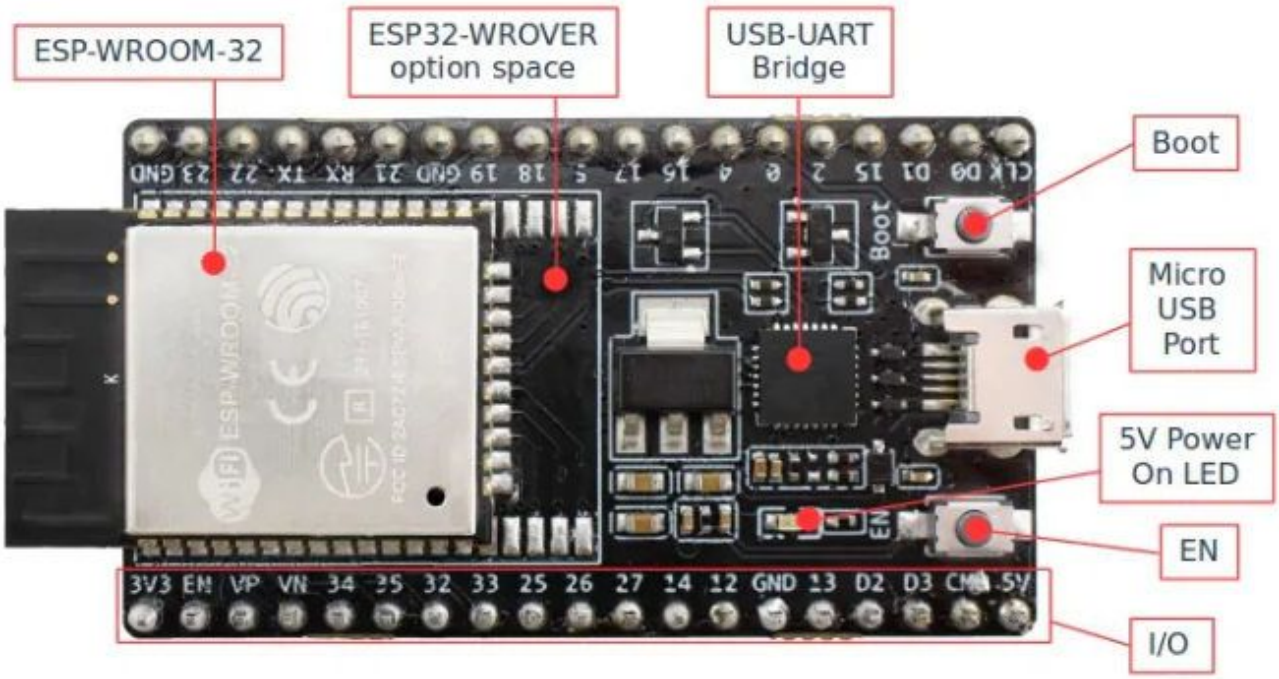
Discovery of the ESP32 card

Presentation

The ESP32 board is not an ARDUINO board. However, development tools leverage certain elements of the ARDUINO eco-system, such as the ARDUINO IDE.

The strong points

In terms of the number of ports available, the ESP32 card is located between an ARDUINO



NANO and ARDUINO UNO. The basic model has 38 connectors:

ESP32 devices include :

- 18 analog-to-digital converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 digital-to-analog converters (DAC)
- 2 I2S interfaces

- 10 capacitive sensing GPIOs

The ADC (analog-to-digital converter) and DAC (digital-to-analog converter) functionality are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc. You just need to assign them in the code. This is possible thanks to the multiplexing function of the ESP32 chip.

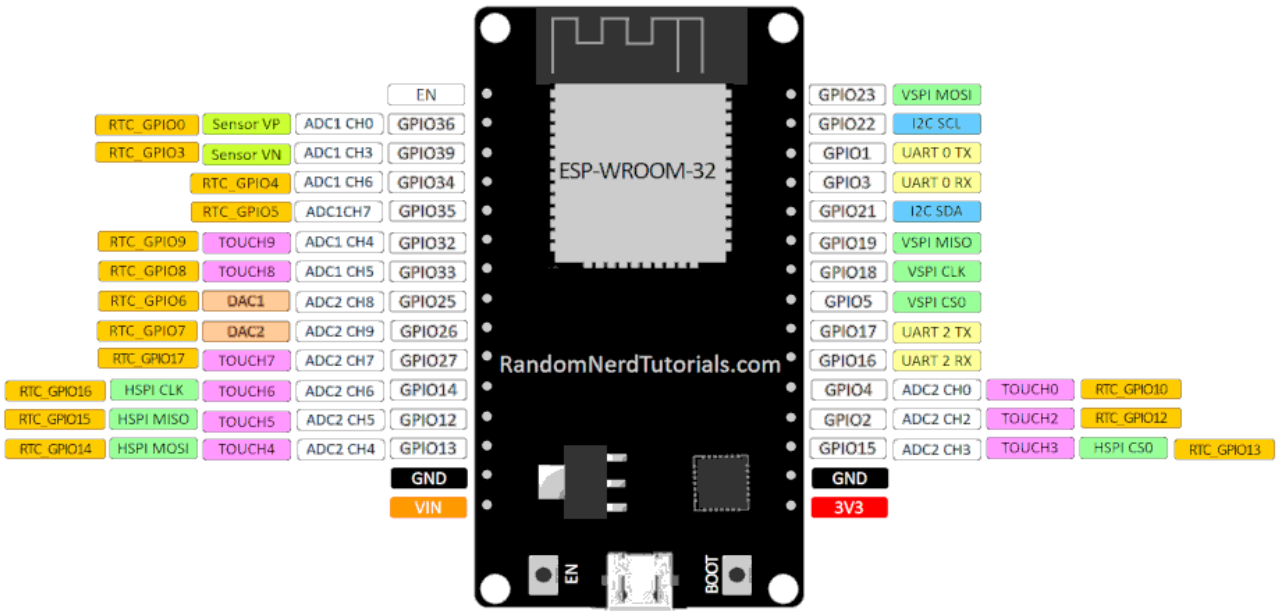
Most connectors have multiple uses.

But what sets the ESP32 board apart is that it is equipped as standard with WiFi and Bluetooth support, something that ARDUINO boards only offer in the form of extensions.

GPIO inputs/outputs on ESP32

Here, in photo, the ESP32 card from which we will explain the role of the different GPIO inputs/outputs :

The position and number of GPIO I/Os may change depending on the card brand. If this is the case, only the indications appearing on the physical map are authentic. Pictured, bottom row, left to right: CLK, SD0, SD1, G15, G2, G0, G4, G16.....G22, G23, GND.



In this diagram, we see that the bottom row begins with 3V3 while in the photo, this I/O is at the end of the top row. It is therefore very important not to rely on the diagram and instead to double check the correct connection of the peripherals and components on the physical ESP32 card.

Development boards based on an ESP32 generally have 33 pins apart from those for the power supply. Some GPIO pins have somewhat particular functions :

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

If your ESP32 card has I/O GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, you should definitely not use them because they are connected to the flash memory of the ESP32. If you use them the ESP32 will not work.

GPIO1(TX0) and GPIO3(RX0) I/O are used to communicate with the computer in UART via USB port. If you use them, you will no longer be able to communicate with the card.

GPIO36(VP), GPIO39(VN), GPIO34, GPIO35 I/O can be used as input only. They also do not have built-in internal pullup and pulldown resistors.

The EN terminal allows you to control the ignition status of the ESP32 via an external wire. It is connected to the EN button on the card. When the ESP32 is turned on, it is at 3.3V. If we connect this pin to ground, the ESP32 is turned off. You can use it when the ESP32 is in a box and you want to be able to turn it on/off with a switch.

ESP32 Peripherals

To interact with modules, sensors or electronic circuits, the ESP32, like any micro-controller, has a multitude of peripherals. There are more of them than on a classic Arduino board.

ESP32 has the following peripherals :

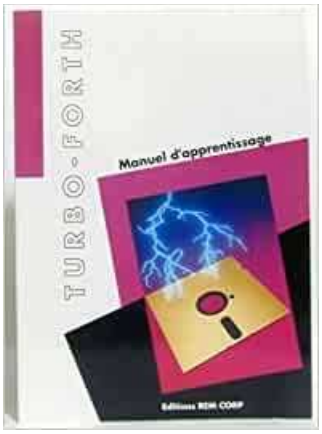
- 3 UART interface
- 2 I2C interfaces
- 3 SPI interfaces
- 16 PWM outputs
- 10 capacitive sensors
- 18 analog inputs (ADC)
- 2 DAC outputs

Some peripherals are already used by ESP32 during its basic operation. There are therefore fewer possible interfaces for each device.

Why program in FORTH language on ESP32?

Preamble

I have been programming in FORTH since 1983. I stopped programming in FORTH in 1996. But I have never stopped monitoring the evolution of this language. I resumed programming in 2019 on ARDUINO with FlashForth then ESP32forth.



I am co-author of several books concerning the FORTH langage :

- 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)

Programming in the FORTH language was always a hobby until 1992 when the manager of a company working as a subcontractor for the automobile industry contacted me. They had a concern for software development in C language. They needed to order an industrial automaton.

The two software designers of this company programmed in C language: TURBO-C from Borland to be precise. And their code couldn't be compact and fast enough to fit into the 64 kilobytes of RAM memory. It was 1992 and flash memory type expansions did not exist. In these 64 KB of RAM, we had to fit MS-DOS 3.0 and the application!

For a month, C language developers had been twisting the problem in all directions, even reverse engineering with SOURCER (a disassembler) to eliminate non-essential parts of executable code.

I analyzed the problem that was presented to me. Starting from scratch, I created, alone, in a week, a perfectly operational prototype that met the specifications. For three years, from 1992 to 1995, I created numerous versions of this application which was used on the assembly lines of several automobile manufacturers.

Boundaries between language and application

All programming languages are shared like this :

- an interpreter and executable source code: BASIC, PHP, MySQL, JavaScript, etc... The application is contained in one or more files which will be interpreted whenever necessary. The system must permanently host the interpreter running the source code;
- a compiler and/or assembler: C, Java, etc. Some compilers generate native code, that is to say executable specifically on a system. Others, like Java, compile executable code on a virtual Java machine.

The FORTH language is an exception. It integrates :

- an interpreter capable of executing any word in the FORTH language
- a compiler capable of extending the dictionary of FORTH words

What is a FORTH word?

A FORTH word designates any dictionary expression composed of ASCII characters and usable in interpretation and/or compilation: words allows you to list all the words in the FORTH dictionary.

Certain FORTH words can only be used in compilation: **`if else then`** for example.

With the FORTH language, the essential principle is that we do not create an application. In FORTH, we extend the dictionary! Each new word you define will be as much a part of the FORTH dictionary as all the words pre-defined when FORTH starts. Example:

```
: typeToLoRa ( -- )
    0 echo !    \ disable display echo from terminal
    ['] serial2-type is type
  ;
: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ enable display echo from terminal
  ;
```

We create two new words: **`typeToLoRa`** and **`typeToTerm`** which will complete the dictionary of pre-defined words.

A word is a function?

Yes and no. In fact, a word can be a constant, a variable, a function... Here, in our example, the following sequence :

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

would have its equivalent in C langage :

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

In FORTH language, there is no limit between language and application.

In FORTH, as in C language, you can use any word already defined in the definition of a new word.

Yes, but then why FORTH rather than C?

I was expecting this question.

In C language, a function can only be accessed through the main function **`main()`**. If this function integrates several additional functions, it becomes difficult to find a parameter error in the event of a malfunction of the program.

On the contrary, with FORTH it is possible to execute - via the interpreter - any word pre-defined or defined by you, without having to go through the main word of the program.

The FORTH interpreter is immediately accessible on the ESP32 card via a terminal type program and a USB link between the ESP32 card and the PC.

The compilation of programs written in FORTH language is carried out in the ESP32 card and not on the PC. There is no upload. Example:

```
: >gray ( n -- n' )
    dup 2/ xor      \ n' = n xor ( 1 time right shift logic )
  ;
```

This definition is transmitted by copy/paste into the terminal. The FORTH interpreter/compiler will parse the stream and compile the new word **`>gray`**.

In the definition of **`>gray`**, we see the sequence **`dup 2/ xor`**. To test this sequence, simply type it in the terminal. To execute **`>gray`**, simply type this word in the terminal, preceded by the number to transform.

FORTH language compared to C language

This is my least favorite part. I don't like to compare the FORTH language to the C language. But as almost all developers use the C language, I'm going to try the exercise.

Here is a test with **`if()`** in C language:

```
if(j > 13){              // If all bits are received
    rc5_ok = 1;          // Decoding process is OK
    detachInterrupt(0);  // Disable external interrupt (INT0)
    return;
}
```

Test with if in FORTH language (code snippet) :

```
var-j @ 13 >        \ If all bits are received
    if
        1 rc5_ok !  \ Decoding process is OK
```

```
        di          \ Disable external interrupt (INT0)
        exit
    then
```

Here is the initialization of registers in C langage :

```
void setup() {
  // Timer1 module configuration
  TCCR1A = 0;
  TCCR1B = 0;          // Disable Timer1 module
  TCNT1  = 0;          // Set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;          // enable Timer1 overflow interrupt
}
```

The same definition in FORTH langage :

```
: setup {
  \ Timer1 module configuration
  0 TCCR1A !
  0 TCCR1B !     \ Disable Timer1 module
  0 TCNT1  !     \ Set Timer1 preload value to 0 (reset)
  1 TIMSK1 !     \ enable Timer1 overflow interrupt
}
```

What FORTH allows you to do compared to the C language

We understand that FORTH immediately gives access to all the words in the dictionary, but not only that. Via the interpreter, we also access the entire memory of the ESP32 card. Connect to the ESP32 board that has ESP32forth installed, then simply type :

```
hex here 100 dump
```

You should find this on the terminal screen :

```
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
```

This corresponds to the contents of flash memory.

And the C language couldn't do that?

Yes, but not as simple and interactive as in FORTH language.

But why a stack rather than variables?

The stack is a mechanism implemented on almost all microcontrollers and microprocessors. Even the C language leverages a stack, but you don't have access to it.

Only the FORTH language gives full access to the data stack. For example, to make an addition, we stack two values, we execute the addition, we display the result: `2 5 + .` displays 7.

It's a little destabilizing, but when you understand the mechanism of the data stack, you greatly appreciate its formidable efficiency.

The data stack allows data to be passed between FORTH words much more quickly than by processing variables as in C language or any other language using variables.

Are you convinced?

Personally, I doubt that this single chapter will irremediably convert you to programming in the FORTH language. When trying to master ESP32 cards, you have two options :

- program in C language and use the numerous libraries available. But you will remain locked into the capabilities of these libraries. Adapting codes to C language requires real knowledge of programming in C language and mastering the architecture of ESP32 cards. Developing complex programs will always be a problem.
- try the FORTH adventure and explore a new and exciting world. Of course, it won't be easy. You will need to understand the architecture of ESP32 cards, the registers, the register flags in depth. In return, you will have access to programming perfectly suited to your projects.

Are there any professional applications written in FORTH?

Oh yes! Starting with the HUBBLE space telescope, certain components of which were written in FORTH language.

The German TGV ICE (Intercity Express) uses RTX2000 processors to control motors via power semiconductors. The machine language of the RTX2000 processor is the FORTH language.

This same RTX2000 processor was used for the Philae probe which attempted to land on a comet.

The choice of the FORTH language for professional applications turns out to be interesting if we consider each word as a black box. Each word must be simple, therefore have a fairly short definition and depend on few parameters.

During the debugging phase, it becomes easy to test all the possible values processed by this word. Once made perfectly reliable, this word becomes a black box, that is to say a function in which we have absolute confidence in its proper functioning. From word to word, it is easier to make a complex program reliable in FORTH than in any other programming language.

But if we lack rigor, if we build gas plants, it is also very easy to get an application that works poorly, or even to completely crash FORTH!

Finally, it is possible, in FORTH language, to write the words you define in any human language. However, the usable characters are limited to the ASCII character set between 33 and 127. Here is how we could symbolically rewrite the words high and low:

```
\ Turn a port pin on, dont change the others.
: __/ ( pinmask portadr -- )
    mset
  ;
\ Turn a port pin off, dont change the others.
: \__ ( pinmask portadr -- )
    mclr
  ;
```

From this moment, to turn on the LED, you can type:

```
_O_  __/     \ turn LED on
```

Yes! The sequence `_o_ __/` is in FORTH language!

With ESP32forth, here are all the characters at your disposal that can compose a FORTH word :

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

Good programming.

A real 32-bit FORTH with ESP32Forth

ESP32Forth is a real 32-bit FORTH. What does it mean?

The FORTH language favors the manipulation of integer values. These values can be literal values, memory addresses, register contents, etc.

Values on the data stack

When ESP32Forth starts, the FORTH interpreter is available. If you enter any number, it will be dropped onto the stack as a 32-bit integer :

```
35
```

If we stack another value, it will also be stacked. The previous value will be pushed down one position :

```
45
```

To add these two values, we use a word, here **+**:

```
+
```

Our two 32-bit integer values are added together and the result is dropped onto the stack. To display this result, we will use the word **.**:

```
. \ display 80
```

In FORTH language, we can concentrate all these operations in a single line :

```
35 45 + .  \ display 80
```

Unlike the C language, we do not define an **int8** or **int16** or **int32** type.

With ESP32Forth, an ASCII character will be designated by a 32-bit integer, but whose value will be bounded [32..256[. Example :

```
67 emit  \ display C
```

Values in memory

ESP32Forth allows you to define constants and variables. Their content will always be in 32-bit format. But there are situations where that doesn't necessarily suit us. Let's take a simple example, defining a Morse code alphabet. We only need a few bytes :

- un pour définir le nombre de signes du code morse
- un ou plusieurs octets pour chaque lettre du code 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,
```

Here we define only 3 words, **mA**, **mB** and **mC**. In each word, several bytes are stored. The question is: how will we retrieve the information in these words ?

The execution of one of these words deposits a 32-bit value, a value which corresponds to the memory address where we stored our Morse code information. It is the word **c@** that we will use to extract the Morse code from each letter :

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

The first byte placed on the stack will be used to manage a loop to display the code of a character in Morse code :

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

There are plenty of certainly more elegant examples. Here we show a way to manipulate 8-bit values, our bytes, while operating these bytes on a 32-bit stack.

Word processing depending on data size or type

In all other languages, we have a generic word, like **echo** (in PHP) which displays any type of data. Whether integer, real, string, we always use the same word. Example in PHP language:

```
$bread = "Baked bread";
$price = 2.30;
echo $bread . " : " . $price;
// display   Baked bread: 2.30
```

For all programmers, this way of doing things is THE STANDARD! So how would FORTH do this example in PHP?

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

Here, the word **type** tells us that we have just processed a character string.

Where PHP (or any other language) has a generic function and a parser, FORTH compensates with a single data type, but adapted processing methods which inform us about the nature of the data processed.

Here is an absolutely trivial case for FORTH, displaying a number of seconds in HH:MM:SS format:

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

I love this example because, to date, **NO OTHER PROGRAMMING LANGUAGE** is capable of achieving this HH:MM:SS conversion so elegantly and concisely.

You have understood, the secret of FORTH is in its vocabulary.

Conclusion

FORTH has no data typing. All data passes through a data stack. Each position in the stack is ALWAYS a 32-bit integer!

That's all there is to know.

Purists of hyper-structured and verbose languages, such as C or Java, will certainly cry heresy. And here, I will allow myself to answer them : why do you need to type your data ?

Because it is in this simplicity that the power of FORTH lies : a single stack of data with an untyped format and very simple operations.

And I'm going to show you what many other programming languages can't do, define new definition words :

```
: 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   .- -... -.-.
```

Here, the word **morse:** has become a definition word, in the same way as constant or variable...

Because FORTH is more than a programming language. It is a meta-language, that is to say a language to build your own programming language....

Ressources

in English

- **ESP32forth** page maintained by Brad NELSON, the creator of ESP32forth. You will find all versions there (ESP32, Windows, Web, Linux...)
  https://esp32forth.appspot.com/ESP32forth.html
-

In french

- **ESP32 Forth** site in two languages (French, English) with lots of examples
  https://esp32.arduino-forth.com/

GitHub

- **Ueforth** resources maintained by Brad NELSON. Contains all Forth and C language source files for ESP32forth
  https://github.com/flagxor/ueforth
- **ESP32forth** source codes and documentation for ESP32forth. Resources maintained by Marc PETREMANN
  https://github.com/MPETREMANN11/ESP32forth
- **ESP32forthStation** resources maintained by Ulrich HOFFMAN. Stand alone Forth computer with LillyGo TTGO VGA32 single board computer and ESP32forth.
  https://github.com/uho/ESP32forthStation
- **ESP32Forth** resources maintained by F. J. RUSSO
  https://github.com/FJRusso53/ESP32Forth
- **esp32forth-addons** resources maintained by Peter FORTH
  https://github.com/PeterForth/esp32forth-addons
- **Esp32forth-org** Code repository for members of the Forth2020 and ESp32forth groups
  https://github.com/Esp32forth-org
-

Index

FORTH word.....9 ressources.....19 type.....17