

Wydział Informatyki, Elektroniki i Telekomunikacji

Kierunek: Elektronika i Telekomunikacja



# Projekt

## Implementacja interpretera Forth na platformie ESP-WROOM-32D

**Prowadzący:**

prof. dr hab. inż. Bogusław Cyganek

**Autorzy:**

Tomasz Sokołowski

Piotr Prusak

## Spis treści

1. Cel projektu .....	3
2. Realizacja projektu .....	3
3. Język Forth .....	3
4. Platforma sprzętowa ESP-WROOM-32D .....	4
5. Wykorzystane oprogramowanie .....	5
6. Implementacja projektu .....	6
6.1. Uruchomienie projektu z wykorzystaniem Microsoft Visual Studio .....	7
6.2. Uruchomienie programu na platformie Linux (Ubuntu) .....	7
6.3. Uruchomienie na platformie ESP32 .....	9
7. Podsumowanie i wnioski .....	15
8. Bibliografia.....	16

## 1. Cel projektu

---

Celem projektu jest implementacja języka Forth na platformie sprzętowej ESP-WROOM-32D. Projekt ma być dostępny na małej przenośnej platformie embedded. Komunikacja z platformą będzie odbywać się poprzez port szeregowy.

## 2. Realizacja projektu

---

Wstępnie zapoznano się z dokumentacją do kodu a następnie uruchomiono skompilowany kod w programie Visual Studio 2019 na system Windows 10. Kolejnym krokiem było uruchomienie interpretera na systemie Linux w dystrybucji Ubuntu 20.04. Ostatnim etapem projektu było uruchomienie interpretera Forth na platformie ESP-WROOM-32D. Do realizacji projektu wykorzystano kod źródłowy z repozytorium <https://github.com/BogCyg/BCForth>. Projekt został umiejscowiony w repozytrium: [https://github.com/thomastomicio/BCForth/tree/esp\\_32](https://github.com/thomastomicio/BCForth/tree/esp_32).

## 3. Język Forth

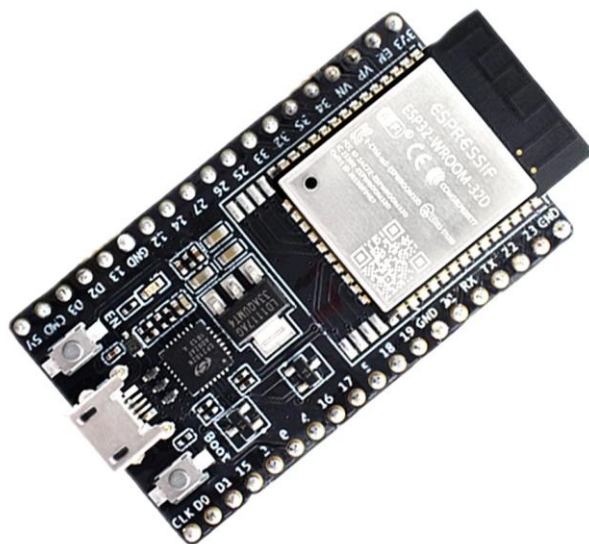
---

Język Forth to wysokopoziomowy język programowania, który wykorzystuje stos do wykonywania operacji. Jego twórcą jest Charles Moore. Język Forth jest znany z tego, że jego składnia jest prosta i zwięzła, a także z tego, że pozwala na łatwe tworzenie własnych słów - funkcji, które są przetwarzane przez interpreter. Jego naczelną zasadą jest "zasada minimalizmu", co oznacza, że jego składnia jest bardzo zwięzła i pozwala na szybkie i łatwe tworzenie programów. Jego głównym celem jest zminimalizowanie liczby linii kodu potrzebnych do napisania programu. Język Forth jest często używany w przemyśle, w szczególności w przemyśle motoryzacyjnym, elektronicznym i telekomunikacyjnym. Program napisany w Forth składa się z sekwencji słów rozdzielonych separatorami. W trakcie swojego działania interpreter języka może znajdować się w jednym ze stanów: definiowania lub wykonywania.

## 4. Platforma sprzętowa ESP-WROOM-32D

---

Platforma sprzętowa ESP-WROOM-32D pozwala na łatwe wykorzystanie możliwości układu ESP32 (Wi-Fi + Bluetooth BLE) firmy Espressif. Na płytce znajduje się konwerter USB/UART (Silabs CP2102), oraz liczne wyprowadzenia pinów mikrokontrolera na złącza goldpin.



*Rys. 1 Platforma sprzętowa ESP-WROOM-32D*

Główne cechy tej platformy to:

- ❖ Procesor Xtensa LX6 o dwóch rdzeniach z taktowaniem do 240 MHz
- ❖ 448 kB pamięci ROM
- ❖ 520 kB pamięci SRAM
- ❖ Możliwość obsługi magistral cyfrowych, m.in.: I2C, SPI, I2S, PWM, UART, IR, CAN
- ❖ 12-bitowy przetwornik analogowo-cyfrowy SAR (do 18 kanałów)
- ❖ Dwa przetworniki cyfrowo-analogowe o rozdzielczości 8-bit
- ❖ Protokoły BT: Bluetooth v4.2 BR/EDR oraz Bluetooth BLE
- ❖ Obsługa Bluetooth Audio: CVSD oraz SBC
- ❖ Protokoły Wi-Fi: 802.11 b/g/n/d/e/i/k/r (802.11n o przepływności do 150 Mbit/s)
- ❖ Wbudowany konwerter USB-UART
- ❖ Wyprowadzenia GPIO o rozstawie 2,54 mm
- ❖ Zasilanie z USB (gniazdo typu microUSB)

## 5. Wykorzystane oprogramowanie

---

### ❖ **Microsoft Visual Studio**

to zintegrowane środowisko programistyczne firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacji Windows Forms, WPF, Web Sites, Web Applications i inne. Aplikacje mogą być pisane m. in. na platformy: Microsoft Windows, Windows Phone, Linux oraz MacOS.

### ❖ **Visual Studio Code**

to darmowy edytor kodu źródłowego z kolorowaniem składni dla wielu języków, stworzony przez Microsoft, o otwartym kodzie źródłowym. Oprogramowanie ma wsparcie dla debugowania kodu, zarządzania wersjami kodu źródłowego za pośrednictwem systemu kontroli wersji Git, automatycznego uzupełniania kodu IntelliSense, zarządzania wycinkami kodu oraz jego refaktoryzacji. Funkcjonalność aplikacji można rozbudować za pomocą rozszerzeń zainstalowanych z dedykowanego repozytorium rozszerzeń, w naszym projekcie skorzystaliśmy z rozszerzenia ESP-IDF.

### ❖ **ESP-IDF**

to natywna platforma programistyczna firmy Espressif dla ESP-32. Zawiera biblioteki oprogramowania i kod źródłowy w interfejsie aplikacji dla ESP32 oraz skrypty do obsługi Toolchain. ESP-IDF zapewnia klasyczne podejście do pisania programu za pomocą interfejsu wiersza poleceń (CLI).

### ❖ **CMake**

to wieloplatformowe narzędzie do automatycznego zarządzania procesem kompilacji programu. Jego główna cecha to niezależność od używanego kompilatora oraz platformy sprzętowej. CMake nie kompiluje programu samodzielnie, lecz tworzy pliki z regułami kompilacji dla konkretnego środowiska; przykładowo w systemie GNU/Linux będą to pliki Makefile, natomiast pod Windowsem — pliki projektu dla Microsoft Visual Studio.

#### ❖ Github

GitHub to serwis, w którym możesz przechowywać swój kod źródłowy, jak i zarządzać całym procesem wytwarzania oprogramowania: od ukrywania plików źródłowych przed publicznością dzięki repozytorium prywatnym, przez możliwość ich pobierania dzięki komendzie git clone, po prośbę o wprowadzenie zmian dzięki opcji tworzenia pull request. Co ciekawe, możemy również utworzyć tzw. fork, czyli kopię istniejącego projektu, i wprowadzić do niego własne zmiany.

#### ❖ Putty

PuTTY to oprogramowanie dedykowane dla systemów operacyjnych Windows oraz Linux. PuTTY pełni funkcję programowego emulatora terminala. W naszym projekcie został wykorzystany do komunikacji szeregowej z platformą ESP-WROOM-32D.

## 6. Implementacja projektu

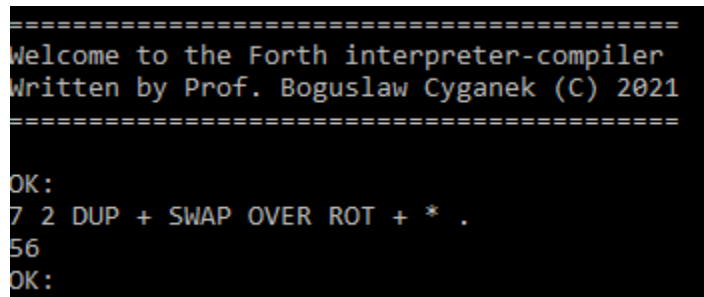
---

Cel projektu to implementacja projektu na platformie ESP-WROOM-32D. Pośrednimi etapami były kompilacja i uruchomienie projektu z wykorzystaniem środowiska Microsoft Visual Studio na systemie Windows. Kolejnym krokiem było uruchomienie projektu z wykorzystaniem systemu Linux w dystrybucji Ubuntu. Finalnym etapem było uruchomienie interpretera Forth na małej przenośnej platformie embedded.

## 6.1. Uruchomienie projektu z wykorzystaniem Microsoft Visual Studio

---

Początkowo sklonowano projekt z repozytorium <https://github.com/BogCyg/BCForth>. Następnie wykorzystując narzędzie CMake zbudowano projekt w katalogu build. Otworzono projekt w programie Microsoft Visual Studio 2019. Skompilowano i uruchomiono program. Przetestowano poprawność działania programu wykorzystując sekwencję opisaną w dokumentacji do projektu udostępnionej lokalnie za pośrednictwem skrzynki pocztowej. Wynik działania programu został przedstawiony poniżej.



```
=====
Welcome to the Forth interpreter-compiler
Written by Prof. Boguslaw Cyganek (C) 2021
=====
OK:
7 2 DUP + SWAP OVER ROT + * .
56
OK:
```

*Rys. 2 Działający program w konsoli Visual Studio*

Opisana wyżej sekwencja umieszcza na stosie liczbę 7 następnie dodaje na wierzchu stosu cyfrę 2, operacja DUP podwaja górną liczbę ze stosu (2) następnie dwie górne wartości są sumowane i na stosie pozostaje 4 i 7. Operacja SWAP zamienia miejscami 2 najwyższe liczby na stosie. Operacja OVER powiela drugą od góry liczbę ze stosu i dodaje ją na wierzchu. Po tej operacji na stosie pozostają cyfry 4, 7 i 4. Operacja ROT zmienia kolejność cyfr na stosie umieszczając 7 na spodzie, a 4 na wierzchu. Następnie dwie cyfry 4 z wierzchu stosu są sumowane i na stosie pozostaje 8 i 7, te dwie liczby są następnie ze sobą mnożone i na stosie pozostaje liczba 56. Kropka ściąga cyfrę z wierzchu stosu i wypisuje na standardowe wyjście. Operacje opisane powyżej zostały graficznie przedstawione w dokumentacji do interpretera.

## 6.2. Uruchomienie programu na platformie Linux (Ubuntu)

---

### Środowisko programistyczne

Etap ten zrealizowano dzięki wykorzystaniu, wspieranego wewnątrz przez system Windows, funkcji podsystemu Linux - WSL(Windows Subsystem for Linux) w wersji 2. Podsystem wspiera obsługę przewidzianej dystrybucji Ubuntu 20.04.5 LTS. Dodatkowo skorzystano z możliwości, jakie oferuje edytor tekstowy Visual Code. Pozwala on na kompilację uprzednio napisanego programu w systemie operacyjnym zainstalowanym w ramach WSL.

## Instalacja potrzebnych pakietów

Ubuntu dostępne za pośrednictwem funkcji należało w pierwszej kolejności odpowiednio skonfigurować. Pakiety wymagane do instalacji kompilatora g++ w wersji 10 (obsługującego standard C++20) zostały zainstalowane w wyniku użycia poniższych komend:

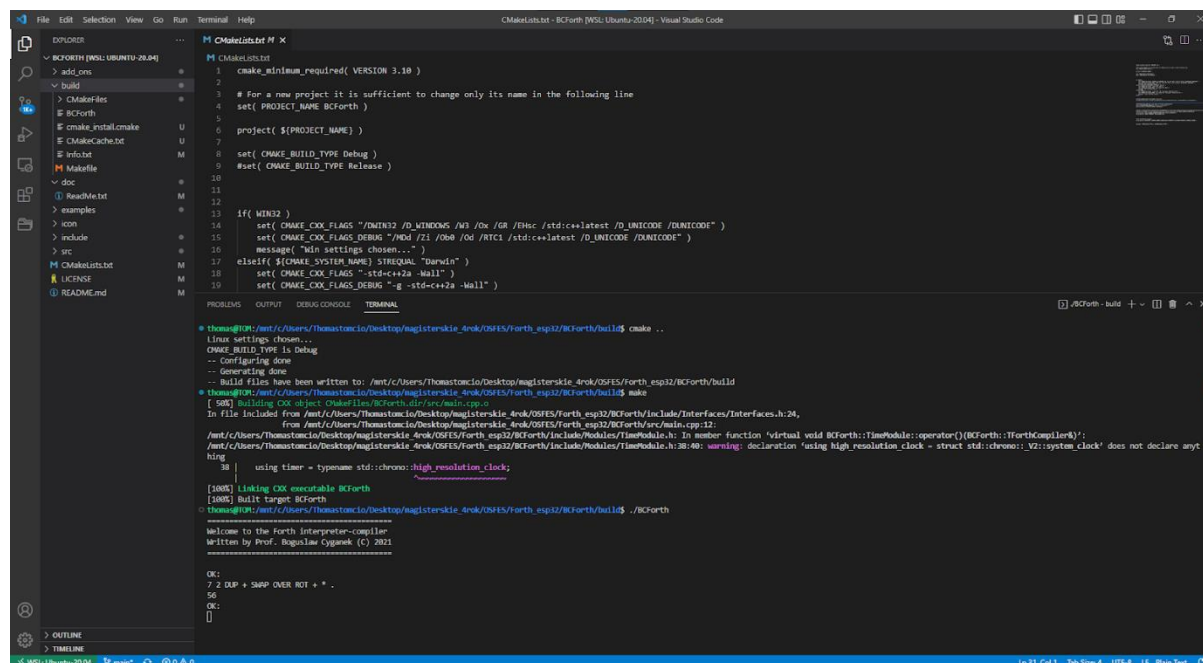
```
sudo apt update -y
sudo apt upgrade -y
sudo apt install -y build-essential
sudo apt install -y gcc-10 g++-10 cpp-10
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-10 100 --
slave /usr/bin/g++ g++ /usr/bin/g++-10 --slave /usr/bin/gcov gcov
/usr/bin/gcov-10
```

Wymagana również była instalacja narzędzia konfiguracji budowania projektu Cmake dostępnego natywnie w omawianym systemie Linux:

```
apt install -y cmake
```

## Efekt końcowy

Udało się uzyskać ten sam rezultat, co w przypadku implementacji na platformie Windows. Poniżej przedstawiono widok na środowisko pracy z uruchomionym terminalem podsystemu WSL, w którym działa aplikacja interpretera języka Forth.



Rys. 3 Interpreter forth działający na systemie Linux



### 6.3. Uruchomienie na platformie ESP32

#### Środowisko programistyczne

Pierwszym etapem, jaki został zrealizowany było przygotowanie środowiska programistycznego. Według specyfikacji otrzymanej implementacji interpretera Forth wymagany do kompilacji projektu był standard języka C++ w wersji std++20.

Budowanie projektów języka C++ dla platformy ESP32 w standardzie std++20 umożliwia środowisko programistyczne *ESP-IDF* w wersji v5. Stąd w projekcie wykorzystano narzędzia właśnie w tej wersji.

#### Narzędzie Cmake

W katalogu głównym projektu utworzono plik CMakeLists.txt, w który prezentuje się następująco:

```
cmake_minimum_required(VERSION 3.16)

include($ENV{IDF_PATH}/tools/cmake/project.cmake)

set( PROJECT_NAME BCForth_for_ESP32 )

project( ${PROJECT_NAME} )
```

Istotna jest linijka ze słowem kluczowym *include*. Jest ona odpowiedzialna za importowanie plików środowiska programistycznego ESP-IDF niezbędnych do kompilacji programu na platformę ESP32.

W katalogu main umieszczone plik CMakeLists.txt o następującej zawartości:

```
idf_component_register(
SRCS
"main.cpp"
INCLUDE_DIRS
"."
"./include"
"./include/Auxiliary"
"./include/Interfaces"
"./include/Modules"
"./include/Words"
"./include/ESP32")
```

Stanowi on informację dla środowiska ESP-IDF o wymaganych do budowy projektu plikach źródłowych (lokalizacje wypisane po słowie kluczowym *SRCS*) oraz nagłówkowych (lokalizacje wypisane po słowie kluczowym *INCLUDE\_DIRS*).

## Kompilacja kodu

Otrzymana bazowa implementacja interpretera Forth w języku C++ nie nadawała się do pomyślnego ukończenia etapu budowy projektu, w efekcie którego powstanie plik binarny z przeznaczeniem do wgrania na platformę ESP32.

Do osiągnięcia pozytywnego rezultatu kompilacji projektu należało zmodyfikować dotychczasowy kod źródłowy programu. Wprowadzono po kolei przedstawione zmiany. Typy o nazwach *CellType* oraz *SignedType* będące pierwotnie aliasem na typ *size\_t* zostały zamienione na alias na tym *unsigned long long* w pliku *BaseDefinitions.h*. W ten sposób możliwe było podczas kompilacji przejście statycznych asercji, które widoczne są w niżej przytoczonym fragmencie kodu.

```

34 using CellType = unsigned long long; //size_t;
35 using RawByte = unsigned char;
36 using Char = char;
37
38 using size_type = unsigned long long; //size_t;
39
40 using SignedIntType = long long; // it is good and efficient to have the SignedIntType
41 // the same length as the basic CellType (otherwise e.g. address
42 // arithmetic causes address crop to 4 bytes if int is 4, etc.)
43 static_assert( sizeof( CellType ) == sizeof( SignedIntType ) );
44
45 using FloatType = double;
46 static_assert( sizeof( CellType ) == sizeof( FloatType ) );

```

Rys. 4 Zmiany w pliku *BaseDefinitions.h*

Ze względu na restrykcyjne podejście kompilatora do analizy kodu należało dokonać drobnych poprawek w zapisie dostarczonego kodu bazowego. Dzięki temu projekt udało się zbudować pod postacią pliku binarnego.

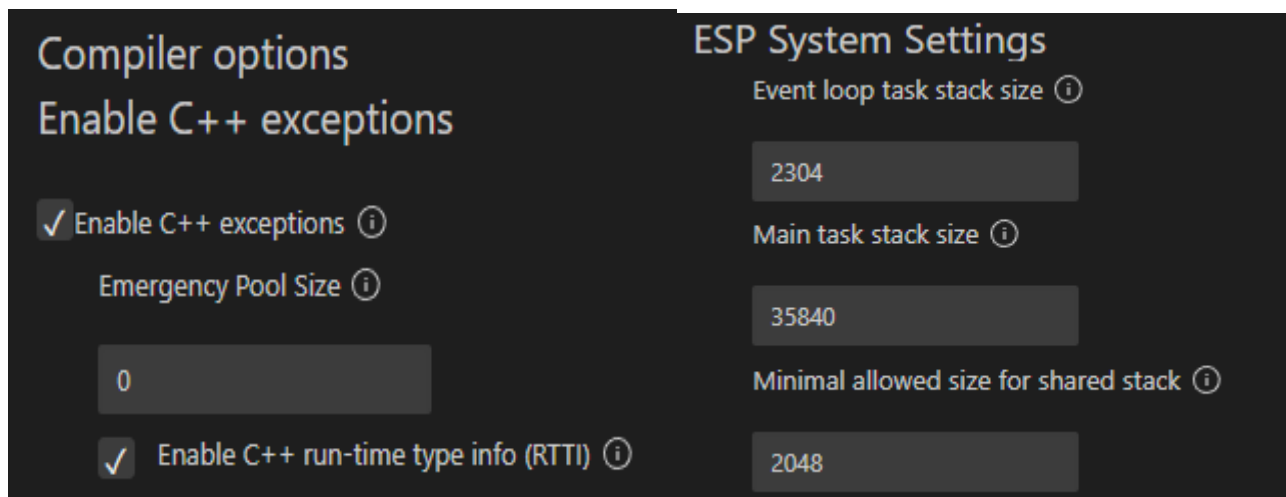
Na poprawki składają się następujące czynności:

- ❖ dodano nawiasy klamrowe około 120 linijki w pliku "SystemWords.h" dla zagnieżdżonych if-ów,
- ❖ zakomentowano linijkę 70 w pliku "CoreModule.h", w której zadeklarowane było niewykorzystywane później w kodzie wyrażenie,
- ❖ zainicjowano zmienną w linijce 67 pliku "TimeModule.h" (*CellType* *\_pad\_addr* = 0;)

## Ustawienie parametrów kompilacji

W udostępnionym przez środowisko ESP-IDF narzędziu do konfiguracji projektu pod nazwą "Menuconfig" wykonano następujące operacje:

- ❖ Zaznaczono opcję "Enable C++ exceptions" w sekcji "Compiler options"
- ❖ Zaznaczono opcję "Enable C++ run-time type info (RTTI)" w sekcji "Compiler options"
- ❖ Zwiększono rozmiar stosu taska głównego z 3584 na 35840 bajtów w sekcji "ESP System Settings".



Rys. 5 Konfiguracja środowiska ESP-IDF

## Przygotowanie pliku konfiguracyjnego platformę ESP32

Plik, o którym mowa nosi nazwę “ESP32\_config.h” i znajduje się w folderze [BCForth/include/ESP32/](#) projektu. Jego zawartość definiuje 3 funkcje:

- ❖ configure
- ❖ register\_spiffs
- ❖ esp\_vfs\_spiffs\_register

Rolą pierwszej z nich (*configure*) jest instalacja sterownika modułu UART, co jest niezbędne do uzyskania blokującego standardowego wejścia STDIN, które domyślnie na platformie ESP32 jest nieblokujące. Dwie ostatnie funkcje są związane z obsługą przewidzianego systemu plików. Zostaną one lepiej opisane w kolejnym rozdziale.

## Funkcja odczyt plików rozszerzeń

Dostarczona implementacja interpretera Forth zawiera obsługę funkcji czytania z pliku. Co więcej plik o nazwie “AddOns.txt” jest wczytywany zaraz po rozpoczęciu pracy programu. Na zapewnienie tych funkcjonalności pozwoliło wykorzystanie wspieranego natywnie przez ESP32 systemu plik o nazwie SPIFFS.

## Wstępna konfiguracja

- ❖ przygotowano odpowiednio plik z mapowaniem partycji "partition.csv" w katalogu projektu z dodanym wpisem partycji o wielkości 1MB przeznaczonej na system plików SPIFFS. Jego zawartość przedstawiona jest poniżej.

```
nvs,      data, nvs,      ,      0x6000,  
phy_init, data, phy,      ,      0x1000,  
factory,  app,  factory,  ,      1M,  
storage,  data, spiffs,   ,      1M
```

Rys. 6 Plik SPIFFS

- ❖ w "Menuconfig" ustawiono w "Partition Table" opcje "Partition Table" : "Custom partition table CSV" i "Custom partition csv file" : "partition.csv"
- ❖ w "Menuconfig" ustawiono w "Serial flasher config" opcję "Flash size" : "4 MB" oraz zaznaczono opcję "Detect flash size when flashing bootloader"
- ❖ dodano linię do pliku Cmake w katalogu głównym ( z plikiem main.cpp ):  
**spiffs\_create\_partition\_image(storage ../add\_ons FLASH\_IN\_PROJECT)**
- ❖ w pliku "Interfaces.h" w linijce 90 ścieżkę do pliku zmieniono na "/spiffs/AddOns.txt"
- ❖

## Instalacja i deinstalacja systemu plików

Za instalację systemu plików SPIFFS odpowiedzialna jest funkcja `register_spiffs`, która ustawia folder nadrzędny jako `"/spiffs"`.

Za deinstalację SPIFFS odpowiada funkcja `unregister_spiffs`.

Definicje obu tych funkcji przedstawia poniższa grafika:

```
void register_spiffs(void)  
{  
    esp_vfs_spiffs_conf_t config = {  
        .base_path = "/spiffs",  
        .partition_label = NULL,  
        .max_files = 5,  
        .format_if_mount_failed = true,  
    };  
    esp_vfs_spiffs_register(&config);  
}  
  
void unregister_spiffs(void)  
{  
    esp_vfs_spiffs_unregister(NULL);  
}
```

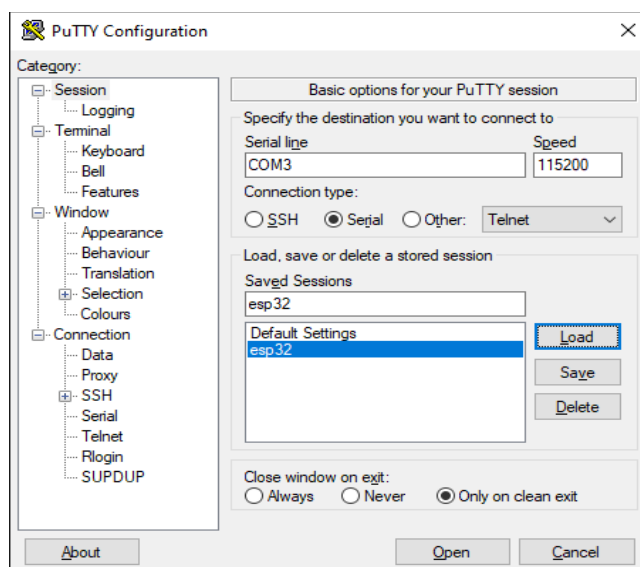
Rys. 7 Definicje funkcji do instalacji i deinstalacji systemu

## Ustawienie klienta komunikacji szeregowej

W celu połączenia z płytką ESP-WROOM-32D wykorzystano klienta komunikacji szeregowej Putty na komputerze, do której płytkę podłączono.

## Ustawienie parametrów połączenia

Wybrano typ transmisji na szeregowy (opcja **Serial**), wybrano odpowiedni port szeregowy COM, pod którym dostępna była płytka, ustawiono prędkość transmisji na 115200 bit/s.

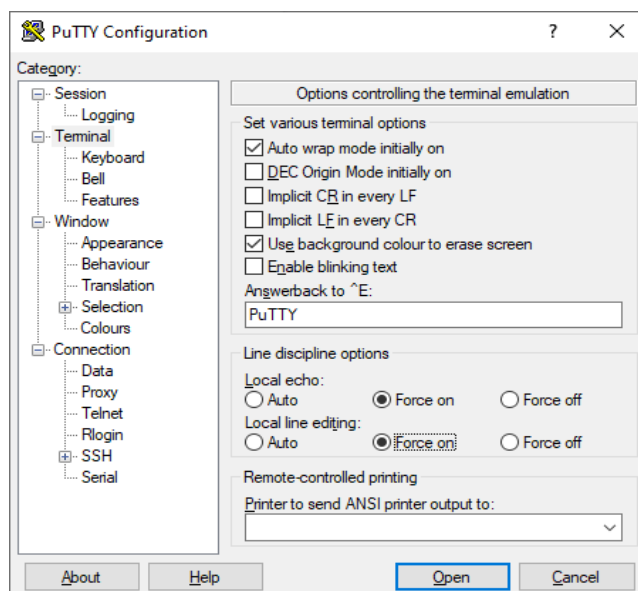


Rys. 8 Konfiguracja programu PuTTY

## Ustawienie wyświetlania

Programu *Putty* domyślnie nie wyświetla wprowadzanych z klawiatury znaków (tzw. lokalne echo). Jest to istotne z uwagi na kontrolę nad przesyłaniem do platformy docelowej (ESP32) znaków. Z tego powodu w interfejsie graficznym programu *Putty* w zakładce **Terminal** w polu **Category** przeprowadzono następujące operacje:

- ❖ Wybrano opcję **Local echo -> Force on**
- ❖ Wybrano opcję **Local line editing -> Force on**



*Rys. 9 Konfiguracja ustawień wyświetlania*

## Efekt końcowy

Uruchomiony terminal programu *Putty* wyświetla oczekiwane rezultaty, które uzyskano przy budowaniu kodu interpretera języka Forth dla platform innych niż ESP32.

```

[0] COM1_PUTTY
load:0xffff0030, len:6982
load:0x00078000, len:15452
load:0x00000400, len:3840
entry 0x0000044c
(127) boot: ESP-IDF v5.0 2nd stage bootloader
(127) boot: compile time 21:51:51
(127) boot: chip revision: V1.0
(130) boot: chip revision: 1, min. bootloader chip revision: 0
(130) boot: esp32: SPI Speed : 40MHz
(142) boot: esp32: SPI Mode : DIO
(146) boot: esp32: SPI Flash Size : 4MB
(151) boot: Flashing ROM early entry source...
(161) boot: Partition Table:
(161) boot: # Label Usage Type St Offset Length
(167) boot: 0 nvs NVS data 01 02 00001000 00001000
(174) boot: 1 phy_init RF data 01 01 00001000 00001000
(181) boot: 2 factory Factory img 01 00 00000000 00000000
(189) boot: 3 storage Unknown data 01 02 00110000 00100000
(193) boot: End of partition table
(193) boot: min. chip revision: 1, min. application chip revision: 0
(198) esp_image: segment 0: paddr=00100020 vaddr=0f000000 size=104ach (101494)
esp
(154) esp_image: segment 1: paddr=00030114 vaddr=0f000000 size=0294ch (10146)
load
(157) esp_image: segment 2: paddr=0003e641 vaddr=40000000 size=011a4ch (4532)
load
(161) esp_image: segment 3: paddr=00040020 vaddr=40000020 size=7a950ch (502096)
load
(172) esp_image: segment 4: paddr=0004b878 vaddr=400011b4 size=0c134ch (49460)
load
(181) esp_image: segment 5: paddr=0004cab7 vaddr=40000000 size=00010ch (16)
load
(182) boot: Loaded app from partition at offset 0x10000
(182) boot: Flashing ROM early entry source...
(205) cpu_start: Pre cpu init.
(205) cpu_start: Starting app cpu, entry point is 0x00001124
(0) cpu_start: App cpu up.
(321) cpu_start: Pre cpu start user code
(321) cpu_start: App start 0x00000034
(321) cpu_start: Application Information:
(328) cpu_start: Project name: RCForth_for_ESP32
(332) cpu_start: App version: 216477e-dirty
(337) cpu_start: Compile time: Jan 18 2023 21:52:10
(344) cpu_start: ELF file SHA256: sha0c4f962000ee...
(350) cpu_start: ESP-IDF: v5.0
(355) heap_init: Initializing. RAM available for dynamic allocation:
(362) heap_init: AS 0FFFA000 len 00001320 (4 KiB): DRAM
(368) heap_init: AS 0FFFA000 len 00000008 (1 KiB): DRAM
(374) heap_init: AS 0FFFA040 len 00003A00 (14 KiB): D/RAM
(380) heap_init: AS 0FFFA000 len 00000000 (1 KiB): D/RAM
(387) heap_init: AS 00000000 len 00001218 (78 KiB): IRAM
(394) spi_flash: detected chip: generic
(398) spi_flash: flash size: 4M
(416) cpu_start: Starting scheduler on PRO CPU.
(0) cpu_start: Starting scheduler on APP CPU.
=====
Welcome to the Forth interpreter-compiler
Written by Prof. Bogusław Cyganek (C) 2021

"/spiffs/AddOns.txt" file successfully loaded
OK:

```

Rys. 10 Finalnie działający interpreter na ESP-WROOM-32

## 7. Podsumowanie i wnioski

---

W pełni zrealizowano postawiony w projekcie cel. Podczas pracy nad projektem poszerzono wiedzę z zakresu programowania platformy ESP-WROOM-32D, oraz szczegółowo zapoznano się z językiem Forth. Wdrożenie projektu na małą przenośną platformę embedded pokazało możliwość implementacji interpretera języka Forth na platformie embedded. Takie rozwiązanie może być stosowane w dziedzinie automatyki do sterowania urządzeniami.

## 8. Bibliografia

---

- ❖ Załączona dokumentacja interpretera Forth
- ❖ [Espressif ESP32 get started](#)
- ❖ [Espressif ESP32 reference manual](#)
- ❖ [Espressif ESP32 c++ support](#)
- ❖ [Język Forth](#)
- ❖ [Repozytorium projektu](#)