



# espForth Browser

---

**Forth Modification Lab**  
**SVFIG**

**C. H. Ting**

**March 23, 2019**



# Summary

---

- **Serial IO in eForth**
- **ESP8266 with UDP**
- **ESP32 HTTP Browser**
- **EVAL**
- **EspForth Browser**
- **Evaluate()**
- **ESP32 Programming**



# Serial IO in eForth

---

- **KEY** receives a character from UART input device and stores it in Terminal Input Buffer (TIB).
- **EMIT** sends a character out to UART output device.
- **HyperTerminal** is my choice to interact with eForth.



# Serial IO in More Sophisticated Forth

---

- **EXPECT** receives a stream of characters from UART input register and stores them in Terminal Input Buffer (TIB).
- **TYPE** streams characters in an output buffer to UART output register.



# ESP8266 with UDP

---

- **UDP is the simplest protocol for WiFi interaction.**
- **ACCEPT is modified to receive input stream from both Serial Monitor and UDP receiver.**
- **EMIT is modified to send characters to UDP output buffer.**
- **CR is modified to send out an assembled UDP packet.**



# ESP8266 with UDP

---

- **I need an UDP terminal emulator on Windows to interface with esp8266forth through WiFi.**
- **Serial Monitor on Arduino works much better than HyperTerminal.**

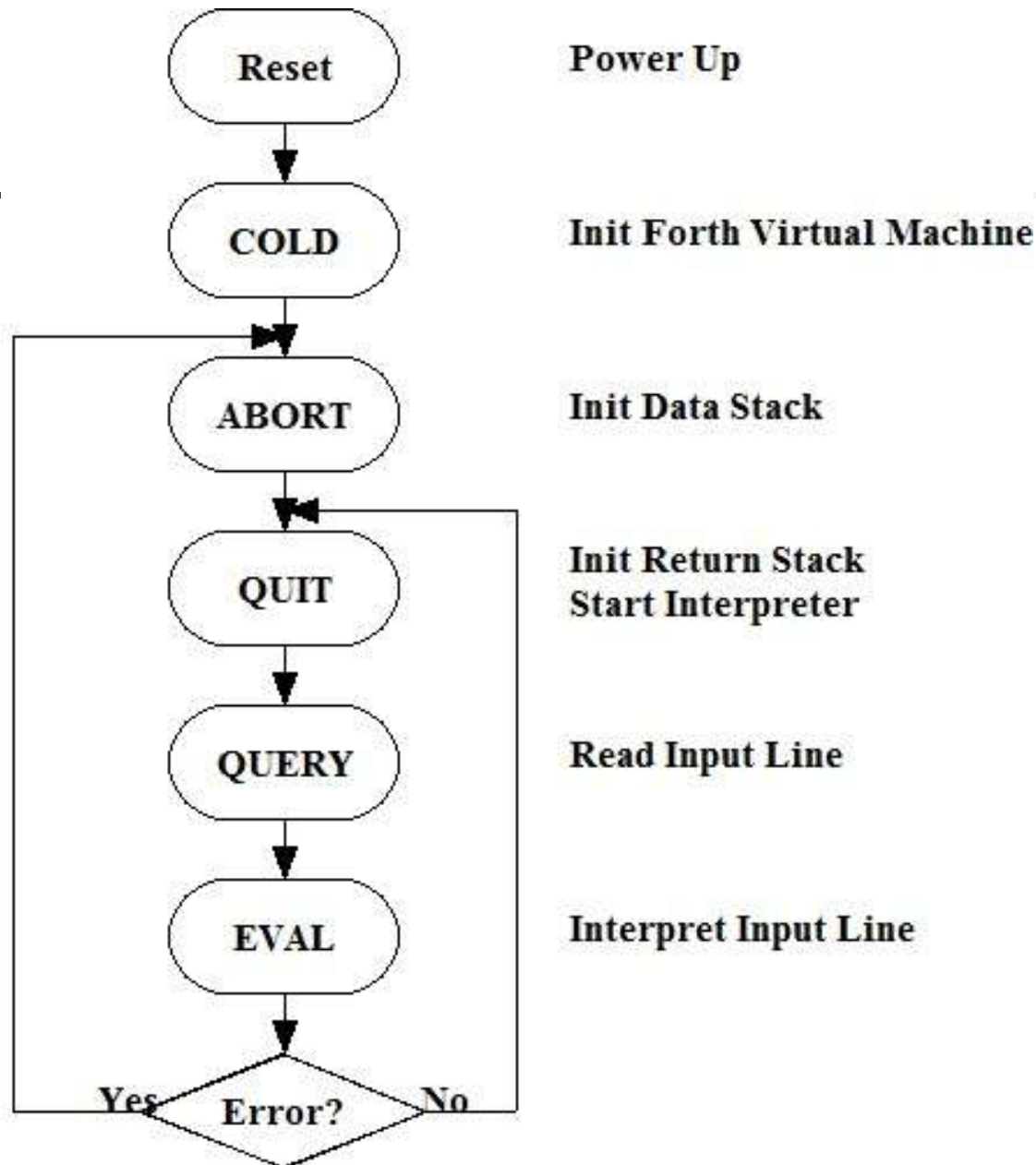


# ESP32 Browser

---

- **Most WiFi LED tutorials for esp32 use browsers to turn LEDs on and off with buttons on a web page.**
- **Web page seems to be a natural interface for eForth, with a few kinks.**

# Forth Interpreter







# Forth Interpreter

---

```
: QUIT ( -- )  
  [ BEGIN QUERY EVAL AGAIN  
:  
: QUERY ( -- )  
  TIB $100 LIT ACCEPT #TIB !  
  DROP 0 LIT >IN ! ;  
:  
: EVAL ( -- )  
  BEGIN TOKEN DUP @  
  WHILE 'EVAL @EXECUTE  
  REPEAT DROP .OK ;
```



# EVAL

---

- **Instead of QUIT, we now focus on EVAL as the essence of Forth interpreter.**
- **Other languages or operating systems can prepare Forth words, stuff them into TIB, and then call EVAL to interpret them.**



# EVAL

---

- **To integrate EVAL into other languages and operating systems, we just have to make sure that:**
  - **EVAL gets all Forth words in TIB**
  - **EVAL can exit Forth and return to other system smoothly.**



# espForth\_53 Loop

---

```
void loop() {  
    bytecode=(unsigned char)cData[P++];  
    primitives[bytecode]();  
}
```



# **espForth\_56 Browser**

---

- **espForth receives and decodes an URL.**
- **Forth words in URL are handed to EVAL to evaluate.**
- **A BREAK instruction in espForth causes Forth to exit and then sends back a web page.**
- **Etc...**



# espForth\_56 Browser

---

```
void loop()  
{  get URL  
    decode URL into TIB  
    EVAL  
    send out web page  
}
```



# Break out loop()

---

- **espForth has an un-used opcode NOP with 0 opcode.**
- **NOP was not used in espForth.**
- **NOP is now used to break out the Arduino loop(), and terminate Forth interpreter.**
- **Forth loop is in evaluate().**



# evaluate()

---

```
void evaluate()  
{ while (true){  
    bytecode=(unsigned char)cData[P++] ;  
    if (bytecode)  
        {primitives[bytecode] () ;}  
    else {break;}  
}}
```





# esp32forth 5.6 Loop

---

```
void loop()  
{  get URL  
    decode URL  
    evaluate();  
    send out web page  
}
```

```
void loop() {  
    WiFiClient client = server.available();  
    if (client) {  
        Serial.println("New Client.");  
        HTTPin = "";  
        HTTPout = "";  
        while (client.connected()) {  
            if (client.available()) {  
                char c = client.read();  
                Serial.write(c);  
                HTTPin += c;  
                if (c == '\n') // end of line
```

```
HTTPin.replace ("%20", " ");
HTTPin.replace ("GET /", "");
HTTPin.replace ("HTTP/1.1", "");
Serial.println (HTTPin);
len=HTTPin.length ();
HTTPin.getBytes (cData, len);
Serial.println ("Enter Forth.");
data[0x66] = 0;           // >IN
data[0x67] = len;         // #TIB
data[0x68] = 0;           // 'TIB
P = 0x180;                // EVAL
WP = 0x184;
evaluate ();
```

```
Serial.println("Return from Forth.");
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.println("<html><head><title>esp32fort  
h_5.6</title></head>");
client.println("<link rel=\"icon\"  
href=\"data:,\"><body><pre>");
client.println(HTTPout);
client.println("</pre></body></html>");
client.println();
break;}}}
client.stop(); }}
```



# Serial Monitor



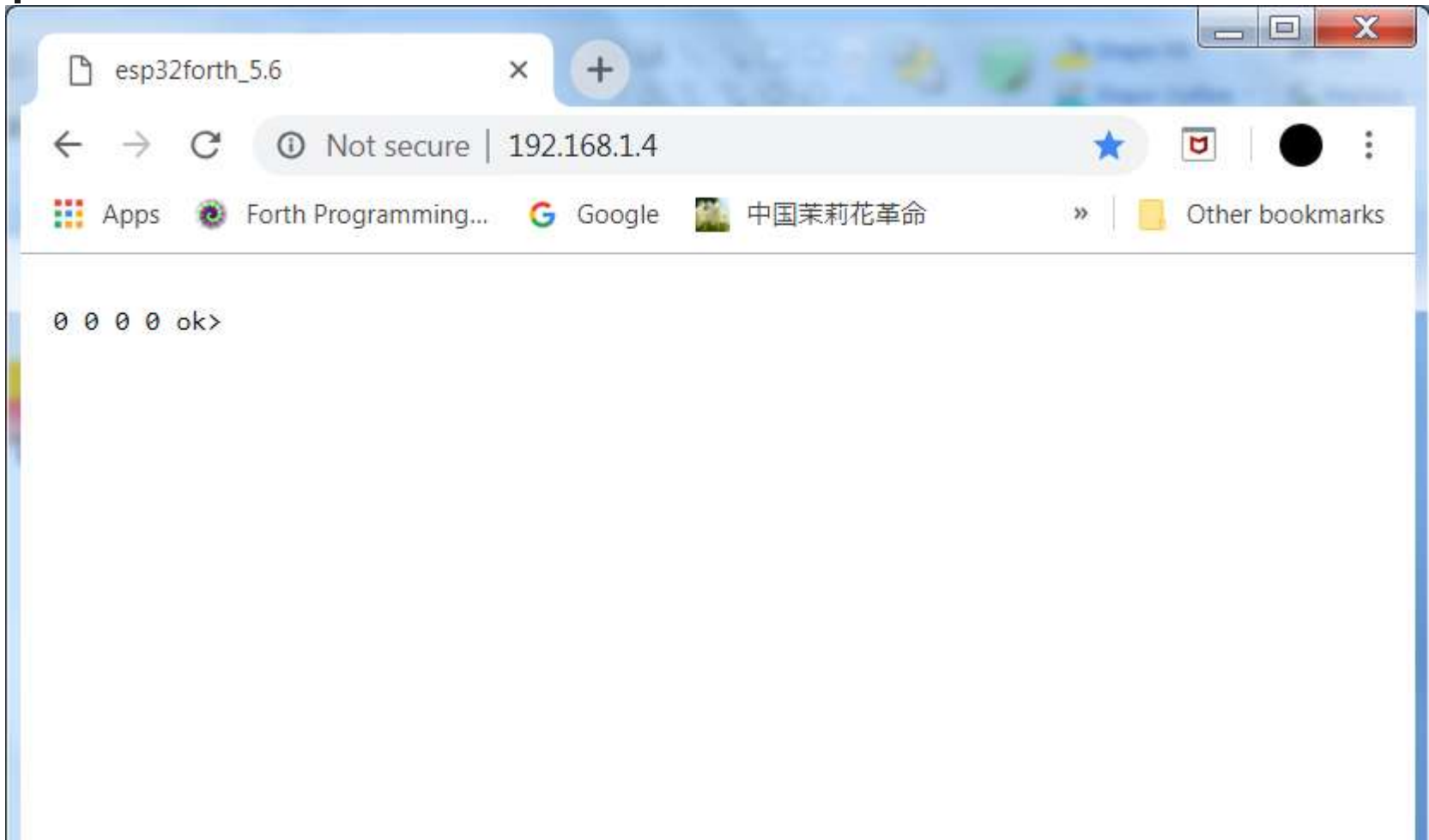
COM6

```
WiFi connected
IP Address: 192.168.1.4
Booting esp32Forth_56 ...
Load file.
  LINE reDef  PP reDef  PO reDef  P1 reDef  POEN reDef  P1EN reDef  P(
  0 0 0 0 ok> Done loading.
New Client.
GET / HTTP/1.1

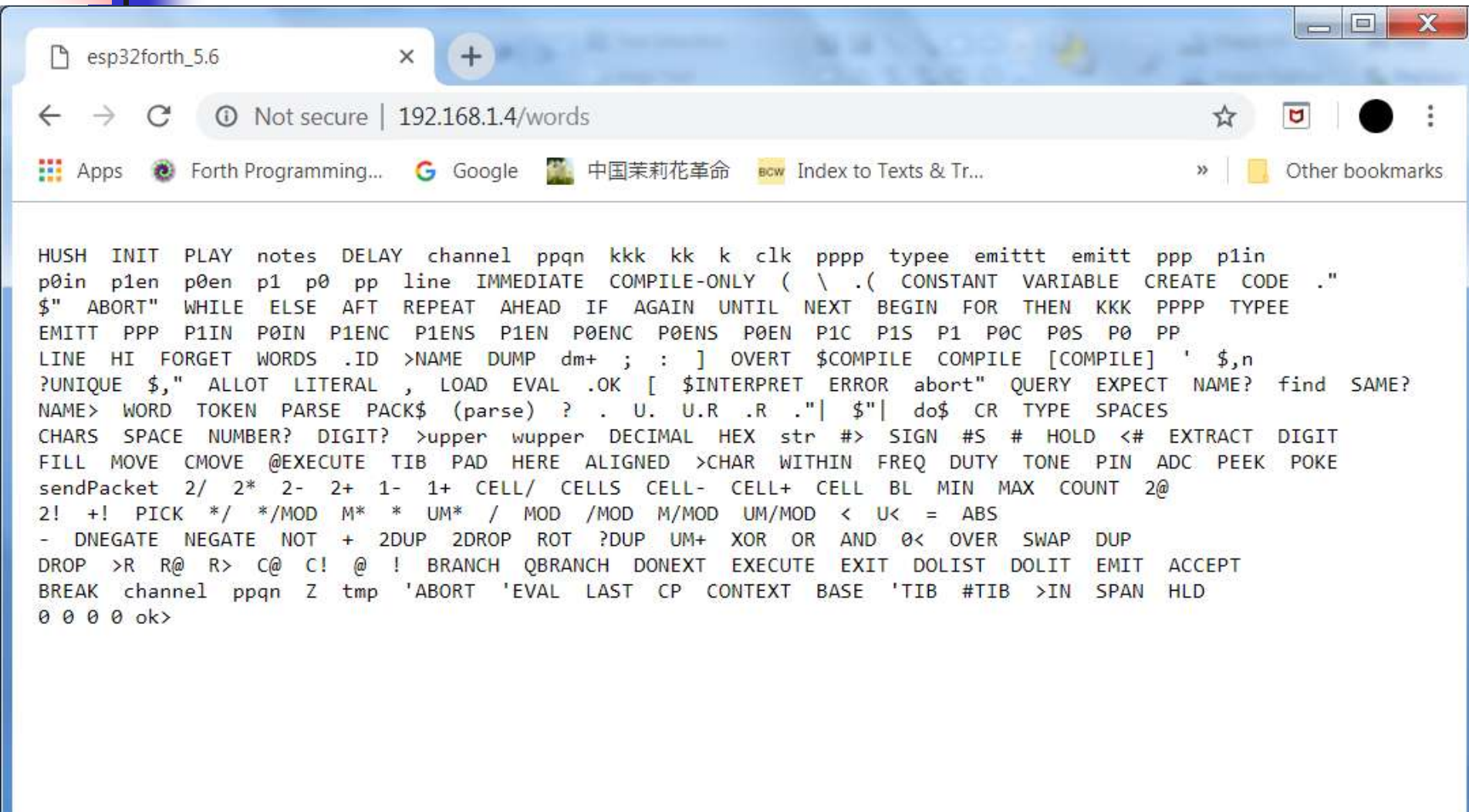
Enter Forth.

  0 0 0 0 ok>
Return from Forth.
Client disconnected.
New Client.
```

# Esp32forth Browser



# Esp32forth Browser



```
HUSH INIT PLAY notes DELAY channel ppqn kkk kk k clk pppp typee emittt emitt ppp p1in
p0in p1en p0en p1 p0 pp line IMMEDIATE COMPILE-ONLY ( \ .( CONSTANT VARIABLE CREATE CODE ."
$" ABORT" WHILE ELSE AFT REPEAT AHEAD IF AGAIN UNTIL NEXT BEGIN FOR THEN KKK PPPP TYPEE
EMITT PPP P1IN P0IN P1ENC P1ENS P1EN P0ENC P0ENS P0EN P1C P1S P1 P0C P0S P0 PP
LINE HI FORGET WORDS .ID >NAME DUMP dm+ ; : ] OVERT $COMPILE COMPILE [COMPILE] ' $,n
?UNIQUE $," ALLOT LITERAL , LOAD EVAL .OK [ $INTERPRET ERROR abort" QUERY EXPECT NAME? find SAME?
NAME> WORD TOKEN PARSE PACK$ (parse) ? . U. U.R .R ."| $"| do$ CR TYPE SPACES
CHARS SPACE NUMBER? DIGIT? >upper wupper DECIMAL HEX str #> SIGN #S # HOLD <# EXTRACT DIGIT
FILL MOVE CMOVE @EXECUTE TIB PAD HERE ALIGNED >CHAR WITHIN FREQ DUTY TONE PIN ADC PEEK POKE
sendPacket 2/ 2* 2- 2+ 1- 1+ CELL/ CELLS CELL- CELL+ CELL BL MIN MAX COUNT 2@
2! +! PICK */ */MOD M* * UM* / MOD /MOD M/MOD UM/MOD < U< = ABS
- DNEGATE NEGATE NOT + 2DUP 2DROP ROT ?DUP UM+ XOR OR AND 0< OVER SWAP DUP
DROP >R R@ R> C@ C! @ ! BRANCH QBRANCH DONEXT EXECUTE EXIT DOLIST DOLIT EMIT ACCEPT
BREAK channel ppqn Z tmp 'ABORT 'EVAL LAST CP CONTEXT BASE 'TIB #TIB >IN SPAN HLD
0 0 0 0 ok>
```



# EVAL

---

- **EVAL is very powerful. It is used to load a file in flash memory to customized espForth for specific applications.**

```
: LOAD ( a n --, load text string )  
  #TIB ! 'TIB ! 0 >IN ! EVAL ;
```





# espForth\_53/56

---

- **EspForth\_53 with Serial Monitor is for programming and testing. Program can be store in flash for loading.**
- **EspForth\_56 with browser is for controlling and showcasing. Application program in flash is loaded automatically on boot.**



# espForth Programming

---

- **Only a few audio commands were tested on NodeMCU ESP32 kit.**
- **Programming and testing methods are developed.**
- **When AI Robot is ready, real program will be developed.**



# espForth Programming

---

- **Serial Monitor Level**
- **Load.txt Level**
- **cEF level**
- **Arduino Level**



# Serial Monitor Level

---

- **Serial Monitor in Arduino is better than HyperTerminal, with very large output display panel**
- **It does not accept text file**
- **256 Byte Terminal Input Buffer**
- **Long line of text works like blocks**



# Load.txt Level

---

- **ESP32 has 4MB flash memory**
- **SPIFFS and FS libraries implement a simple flash file system.**
- **I need only one file: Load.txt.**
- **Load.txt can be compiled manually in espForth\_53, or automatically in espForth\_56.**



# cEF Level

---

- **Use cefMETA\_56.fex to meta-compile esp32eforth:**
  - **cefMeta\_56.f**
  - **cefASM\_56.f**
  - **cefKERN\_56.f**
  - **cEF\_56.f**
- **Rom\_56.h is generated for Arduino.**



# Arduino Level

---

- **New Byte Code**
- **Forth Virtual Machine**
- **Finite State Machine**
- **Call Arduino Libraries**

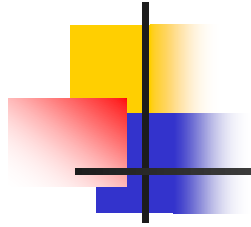


# Conclusion

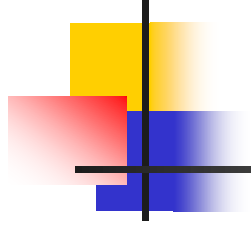
---

- **EspForth browser shows that there is a much more natural and pleasant interface for Forth to the Internet universe.**
- **We need to put a Forth like it online, to attract more followers.**





# Questions?



---

**Thank you.**