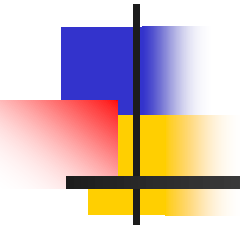


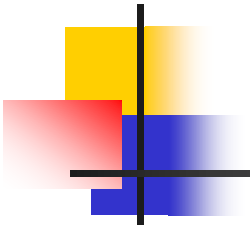
Forth Compiled By Arduino IDE

C Modification Lab

SVFIG

C. H. Ting

June 22, 2019

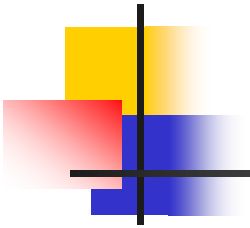


Summary

- Esp32forth v6.1 and v6.2
- Python macro assembler
- Arduino C macro assembler
- Labels in v6.1
- Colon word compiler in v6.2
- Demo of v6.1 and v6.2

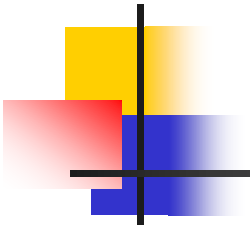
Metamorphosis

Metamorphosis



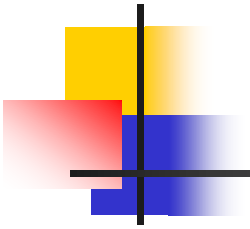
esp32Forth

- esp32Forth emulates eP32, a 32-bit Forth microcontroller.
- Forth Virtual Machine is written in C as an Arduino sketch.
- Forth dictionary is constructed by a metacompiler in F# eForth.



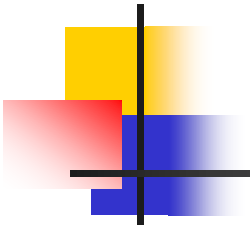
esp32Forth v6.2

- 71 primitive byte-code are coded in C.
- Finite State Machine executes byte code.
- Kernel words contain lists of byte code.
- Colon words contain address lists.
- 256-Level circular stacks.
- Forth is EVAL, called from C.
- One-pass macro assembler/compiler in C.
- Turn-key app stored in flash memory.



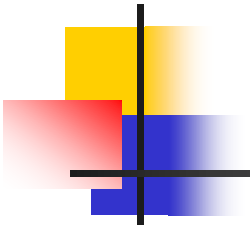
Python Macro Assembler

- Arduino IDE is good at application programming, not for writing an macro assembler.
- Python is a better platform to try out new ideas.
- Forth dictionary is test-constructed by a Python macro assembler.



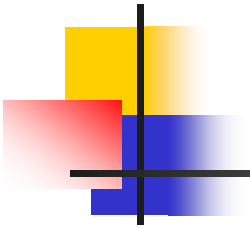
Python Macro Assembler

- Python syntax is close to C. It will be much easier to port a Python macro assembler to a C macro assembler.
- C macro assembler will be hosted on Arduino IDE to build esp32Forth.



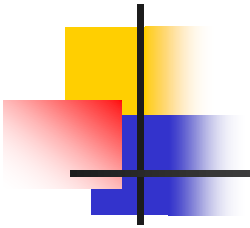
Python Demo

- writeInteger() does , (comma).
- writeByte() does C, (c-comma).
- CODE() assembles a kernel word.
- COLON() compiles a colon word.
- LABEL() compiles a sub-list for branching or looping.



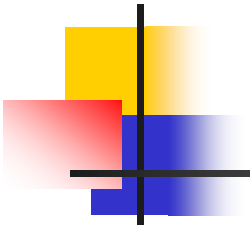
Python Demo

- ceForth_2.py compiles a Forth dictionary and writes it out as a forth.dat file.
- Forth.dat files is compared, byte for byte, to rom_54.h files used in esp32Forth v5.4.



Macro Assembler in C

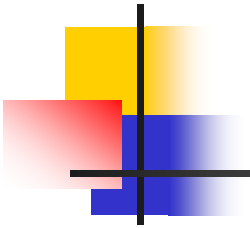
- C is not well equipped to compile an interactive OS like Forth, with variable-length name fields and code fields.
- A macro assembler written in C is forced to construct an eForth dictionary.



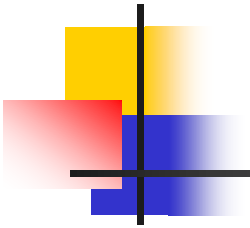
Macro Assembler in C

- HEADER() constructs a link field and a name field.
- CODE() constructs code field filled with byte code.
- COLON() constructs code field filled with address list.
- LABEL() for branching and looping.

Two Memory Views

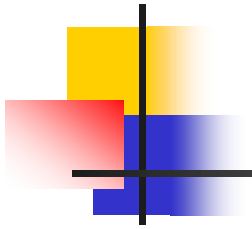


- A large array is allocated for Forth dictionary, viewed both as a word array and a byte array.
- `data[IP++]=n;` assembles a 32-bit word like , (comma).
- `cData[P++]=c;` assembles a byte, like C, (c-comma).
- Pointers: `IP=P/4`



HEADER

```
void HEADER(int lex, char seq[]) {
  P=IP>>2;
  int i;
  int len=lex&31;
  data[P++]=thread;
  IP=P<<2;
  thread=IP;
  cData[IP++]=lex;
  for (i=0;i<len;i++)
     {cData[IP++]=seq[i];}
  while (IP&3) {cData[IP++]=0;}
}
```



CODE

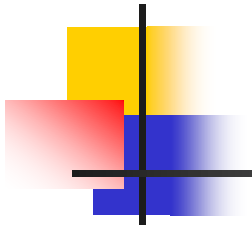
```
int CODE(int len, ... ) {
  int addr=IP;
  int s;
  va_list argList;
  va_start(argList, len);
  for(; len;len--) {
    s= va_arg(argList, int);
    cData[IP++]=s;}
  va_end(argList);
  return addr;
  }
```

Byte Code

```
int as_nop=0;
int as_accept=1;
int as_qrx=2;
int as_txsto=3;
int as_docon=4;
int as_dolit=5;
int as_dolist=6;
int as_exit=7;
int as_execu=8;
int as_donext=9;
int as_qbran=10;
int as_bran=11;
int as_store=12;
int as_at=13;
int as_cstor=14;
int as_cat=15;
int as_rpat=16;
int as_rpsto=17;
int as_rfrom=18;
int as_rat=19;
int as_tor=20;
int as_spat=21;
int as_spsto=22;
int as_drop=23;
int as_dup=24;
int as_swap=25;
int as_over=26;
int as_zless=27;
int as_andd=28;
int as_orr=29;
int as_xorr=30;
int as_uplus=31;
int as_next=32;
int as_qdup=33;
int as_rot=34;
int as_ddrop=35;
int as_ddup=36;
int as_plus=37;
int as_inver=38;
int as_negat=39;
int as_dnega=40;
int as_subb=41;
int as_abss=42;
int as_equal=43;
int as_uless=44;
int as_less=45;
int as_ummod=46;
int as_msmod=47;
int as_slmod=48;
int as_mod=49;
int as_slash=50;
int as_umsta=51;
int as_star=52;
int as_mstar=53;
int as_ssmod=54;
int as_stasl=55;
int as_pick=56;
int as_pstor=57;
int as_dstor=58;
int as_dat=59;
;
int as_count=60;
int as_dovar=61;
int as_max=62;
int as_min=63;
int as_tone=64;
int
as_sendPacket=65;
int as_poke=66;
int as_peek=67;
int as_adc=68;
int as_pin=69;
int as_duty=70;
int as_freq=71;
```

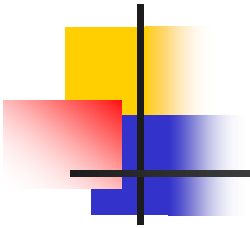
Kernel Words

```
IP=512;
HEADER(3,"HLD");
int HLD=CODE(8,as_docon,as_next,0,0,0X90,1,0,0);
HEADER(4,"SPAN");
int SPAN=CODE(8,as_docon,as_next,0,0,0X94,1,0,0);
HEADER(3,">IN");
int INN=CODE(8,as_docon,as_next,0,0,0X98,1,0,0);
HEADER(4,"#TIB");
int NTIB=CODE(8,as_docon,as_next,0,0,0X9C,1,0,0);
HEADER(4,"'TIB");
int TTIB=CODE(8,as_docon,as_next,0,0,0XA0,1,0,0);
HEADER(4,"BASE");
int BASE=CODE(8,as_docon,as_next,0,0,0XA4,1,0,0);
HEADER(7,"CONTEXT");
int CNTXT=CODE(8,as_docon,as_next,0,0,0XA8,1,0,0);
```



COLON

```
int COLON(int len, ... ) {
  int addr=IP;
  P=IP>>2;
  data[P++]=6; // dolist
  va_list argList;
  va_start(argList, len);
  for(; len;len--) {
    int j=va_arg(argList, int);
    data[P++]=j;}
  IP=P<<2;
  va_end(argList);
  return addr;
  }
```



LABEL

```
int LABEL(int len, ... ) {
int addr=IP;
P=IP>>2;
va_list argList;
va_start(argList, len);
for(; len;len--) {
  int j=va_arg(argList, int);
  data[P++]=j;}
IP=P<<2;
va_end(argList);
return addr;
}
```

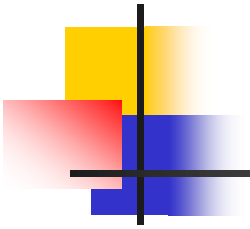
Labels

```
int KEY=0x780;
int TCHA1=0x800;
int EXE1=0x8AC;
int CMOV1=0x8CC;
int CMOV2=0x8EC;
int MOVE1=0x91C;
int MOVE2=0x93C;
int FILL1=0x970;
int FILL2=0x97C;
int DIGS=0xA6C;
int DIGS2=0xA88;
int SIGN1=0xAB4;
int TOUP1=0xBB0;
int DGTQ1=0xC10;
int NUMQ1=0xC7C;
int NUMQ2=0xCC0;
int NUMQ3=0xD18;
int NUMQ4=0xD24;
int NUMQ5=0xD3C;
int NUMQ6=0xD40;
```

```
int CHAR1=0xDA0;
int CHAR2=0xDA8;
int TYPE1=0xDF0;
int TYPE2=0xE04;
int DOT1=0xF54;
int PARS1=0xFC8;
int PARS2=0x100C;
int PARS3=0x1010;
int PARS4=0x101C;
int PARS5=0x104C;
int PARS6=0x1070;
int PARS7=0x1084;
int PARS8=0x109C;
int SAME1=0x1200;
int SAME2=0x125C;
int FIND1=0x12A8;
int FIND2=0x12F8;
int FIND3=0x1308;
int FIND4=0x1310;
int FIND5=0x1328;
int FIND6=0x1340;
```

```
int ABOR1=0x141C;
int ABOR2=0x142C;
int INTER1=0x14B8;
int INTER2=0x14D0;
int INTER3=0x14D4;
int DOTOK1=0x1550;
int DOTOK2=0x1568;
int EVAL1=0x1580;
int EVAL2=0x15A4;
int QUIT1=0x15C8;
int UNIQ1=0x16F4;
int SNAM1=0x1750;
int TICK1=0x1774;
int SCOM1=0x180C;
int SCOM2=0x1810;
int SCOM3=0x1814;
int SCOM4=0x1828;
int DMP1=0x18E4;
int DMP2=0x18FC;
int DUMP1=0x194C;
int DUMP2=0x1974;
```

```
int TNAM1=0x19A4;
int TNAM2=0x19D4;
int TNAM3=0x19E0;
int TNAM4=0x19E8;
int WORS1=0x1A44;
int WORS2=0x1A98;
int WORS3=0x1AAC;
int WORS4=0x1AB4;
int FORG1=0x1ADC;
int FORG2=0x1B0C;
int LINE1=0x1B64;
int PP1=0x1BA0;
int PP2=0x1BBC;
int EMITT1=0x1DBC;
int TYPEE1=0x1E0C;
int PPPP1=0x1E48;
int PPPP2=0x1E70;
```

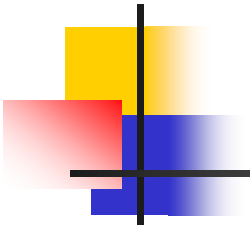


Forward Referencing

- Labels are initialized to 0. After first pass, labels are all resolved properly.
- New values must be copied to the label table. This is equivalent to the second pass, done manually.

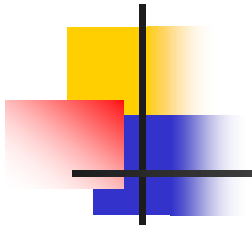
Colon Words

```
HEADER(3,"KEY");
KEY=COLON(4,QKEY,QBRAN,KEY+4,EXITT);
HEADER(6,"WITHIN");
int WITHI=COLON(7,OVER,SUBBB,TOR,SUBBB,RFROM,ULESS,EXITT);
HEADER(5,">CHAR");
int TCHAR=COLON(13,DOLIT,0x7F,ANDD,DUPP,DOLIT,127,BLANK,
  WITHI,QBRAN,TCHA1,DROP,DOLIT,0X5F);
TCHA1=LABEL(1,EXITT);
HEADER(7,"ALIGNED");
int ALIGN=COLON(7,DOLIT,3,PLUS,DOLIT,0XFFFFFFFC,ANDD,EXITT);
HEADER(4,"HERE");
int HERE=COLON(3,CP,AT,EXITT);
HEADER(3,"PAD");
int PAD=COLON(5,HERE,DOLIT,80,PLUS,EXITT);
```



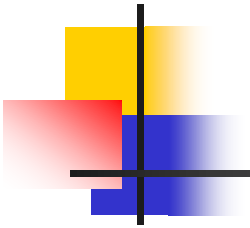
find

```
HEADER(4,"find");
  int FIND=COLON(10,SWAP,DUPP,AT,TEMP,STORE,DUPP,AT,TOR,
       CELLP,SWAP);
  FIND1=LABEL(20,AT,DUPP,QBRAN,FIND4,DUPP,AT,DOLIT,0
       xFFFFFF3F,ANDD,UPPER,RAT,UPPER,XORR,
       QBRAN,FIND2,CELLP,DOLIT,0XFFFFFFFF,BRAN,FIND3);
  FIND2=LABEL(4,CELLP,TEMP,AT,SAMEQ);
  FIND3=LABEL(2,BRAN,FIND5);
  FIND4=LABEL(6,RFROM,DROP,SWAP,CELLM,SWAP,EXITT);
  FIND5=LABEL(6,QBRAN,FIND6,CELLM,CELLM,BRAN,FIND1);
  FIND6=LABEL(9,RFROM,DROP,SWAP,DROP,CELLM,DUPP,
       NAMET,SWAP,EXITT);
```



Symbol Table

- **While assembling Forth, macro assembler spills names and code of all Forth words to the Serial Monitor.**
- **Names and code are compared to contents of rom_54.h.**

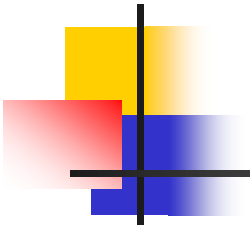


Intel Hex Dump

- To facilitate byte-for-byte comparison, macro assembler also produces a hex dump file with line checksums.
- F# metacompiler also produces a hex dump file for checking.

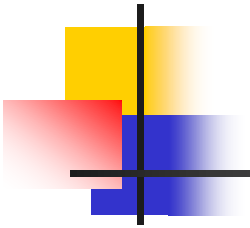
Intel Hex Dump

```
void CheckSum() { int I;  char sum=0;
  Serial.println();
  Serial.printf("%4x ",IP);
  for (i=0;i<32;i++) {sum += cData[IP];
    Serial.printf("%2x",cData[IP++]); }
  Serial.printf(" %2x",sum);}
IP=0;
for (len=0;len<0x120;len++){CheckSum();}
```



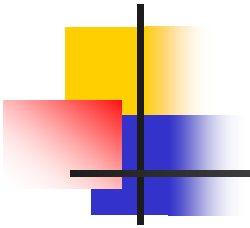
esp32Forth v6.1

- **Macro assembler in esp32Forth v6.1 now produces an identical Forth dictionary as the one in rom_54.h in esp32Forth v5.4.**
- **esp32Forth v6.1 is completely self-sufficient as one sketch file in Arduino IDE.**



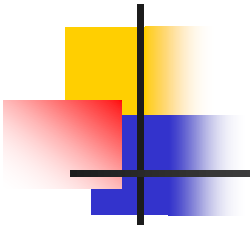
Conclusion I

- **Finally, a complete eForth system is produced by a single C file.**
- **Labeling is still a complicated process. Some smart person will have to figure out how to implement the Forth-like single-pass compiler in C.**



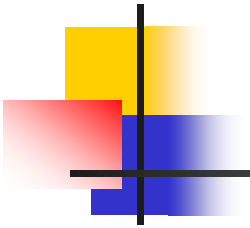
Conclusion I

Guess who that smart person is?



esp32Forth v6.2

- A Forth-like singe-pass macro assembler is now coded in C.
- Instead of LABEL(), I coded these routines:
- IF(), ELSE(), THEN(), FOR(), NEXT(), BEGIN(), UNTIL(), AGAIN(), WHILE(), REPEAT, AFT()

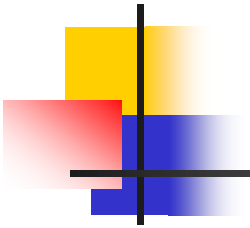


esp32Forth v6.2

- High level Forth words can now be coded similar to native Forth words.
- All words can be assembled in a single pass, without labels.
- Return stack is used to compile nested structures.

Colon Words

```
HEADER(3,"KEY");
  int KEY=COLON(0);
  BEGIN(1,QKEY);
  UNTIL(1,EXITT);
HEADER(6,"WITHIN");
  int WITHI=COLON(7,OVER,SUBBB,TOR,SUBBB,RFROM,ULESS,EXITT);
HEADER(5,">CHAR");
  int TCHAR=
       COLON(8,DOLIT,0x7F,ANDD,DUPP,DOLIT,127,BLANK,WITHI);
  IF(3,DROP,DOLIT,0X5F);
  THEN(1,EXITT);
```

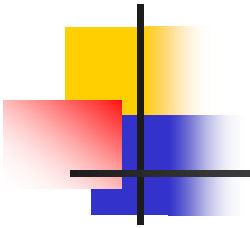


find in v6.2

```
HEADER(4,"find");
  int FIND=COLON(10,SWAP,DUPP,AT,TEMP,STORE,DUPP,
       AT,TOR,CELLP,SWAP);
  BEGIN(2,AT,DUPP);
  IF(9,DUPP,AT,DOLIT,0xFFFFFF3F,ANDD,UPPER,RAT,UPPER,XORR);
  IF(3,CELLP,DOLIT,0XFFFFFFFF);
  ELSE(4,CELLP,TEMP,AT,SAMEQ);
  THEN(0);
  ELSE(6,RFROM,DROP,SWAP,CELLM,SWAP,EXITT);
  THEN(0);
  WHILE(2,CELLM,CELLM);
  REPEAT(9,RFROM,DROP,SWAP,DROP,CELLM,DUPP,NAMET,SWAP,EXITT);
```

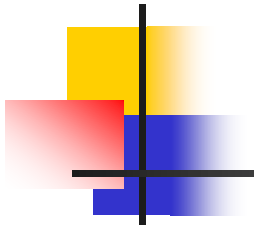
find in eForth

```
:: find ( a va -- xt na | a 0 )
  SWAP          \ va a
  DUP @ tmp !   \ va a  \ get cell count
  DUP @ >R      \ va a  \ #XOR --- count and 1st 3 char
  cell+ SWAP      \ a' va  a'=a(#XOR)+4
  BEGIN @ DUP   \ a' na na
    IF DUP @ $FFFFFF3F LIT AND wupper
      R@ wupper XOR \ ignore lexicon bits
      IF cell+ -1 LIT
      ELSE cell+ tmp @ SAME?
      THEN
    ELSE R> DROP SWAP cell- SWAP EXIT \ a 0
    THEN
  WHILE cell- cell-   \ a' la
  REPEAT R> DROP SWAP DROP
  cell- DUP NAME> SWAP ;;
```



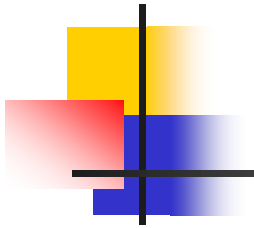
find in v6.1

```
HEADER(4,"find");
  int FIND=COLON(10,SWAP,DUPP,AT,TEMP,STORE,DUPP,AT,TOR,
       CELLP,SWAP);
  FIND1=LABEL(20,AT,DUPP,QBRAN,FIND4,DUPP,AT,DOLIT,0
       xFFFFFF3F,ANDD,UPPER,RAT,UPPER,XORR,
       QBRAN,FIND2,CELLP,DOLIT,0XFFFFFFFF,BRAN,FIND3);
  FIND2=LABEL(4,CELLP,TEMP,AT,SAMEQ);
  FIND3=LABEL(2,BRAN,FIND5);
  FIND4=LABEL(6,RFROM,DROP,SWAP,CELLM,SWAP,EXITT);
  FIND5=LABEL(6,QBRAN,FIND6,CELLM,CELLM,BRAN,FIND1);
  FIND6=LABEL(9,RFROM,DROP,SWAP,DROP,CELLM,DUPP,
       NAMET,SWAP,EXITT);
```



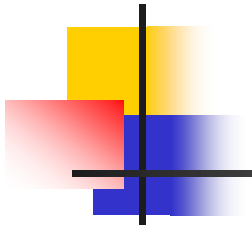
IF

```
void IF(int len, ... ) {
  P=IP>>2;
  data[P++]=QBRAN;
  pushR=P;
  data[P++]=0;
  va_list argList;
  va_start(argList, len);
  for(; len;len--) {
    int j=va_arg(argList, int);
    data[P++]=j; }
  IP=P<<2;
  va_end(argList);
  }
```



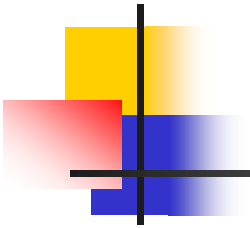
ELSE

```
void ELSE(int len, ... ) {
  P=IP>>2;
  data[P++]=BRAN;
  data[P++]=0;
  data[popR]=P<<2;
  pushR=P-1;
  va_list argList;
  va_start(argList, len);
  for(; len;len--) {
    int j=va_arg(argList, int);
    data[P++]=j;}
  IP=P<<2;
  va_end(argList);
  }
```



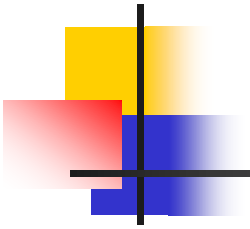
THEN()

```
void THEN(int len, ... ) {
  P=IP>>2;
  data[popR]=P<<2;
  va_list argList;
  va_start(argList, len);
  for(; len;len--) {
    int j=va_arg(argList, int);
    data[P++]=j;}
  IP=P<<2;
  va_end(argList);
  }
```



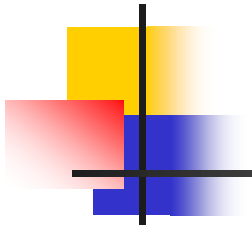
Conclusion II

- The macro assembler in C is now extended to be a Forth meta-compiler.
- Intricate Forth words and control structures can be compiled faithfully in C, in a single pass.

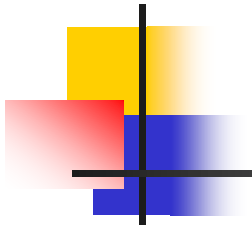


Demo

- Arduino IDE 1.8.9
- esp32Forth v6.1/v6.2
  - Forth Virtual Machine
  - Assembler and kernel words
  - Compiler and colon words
  - Serial Monitor interface



Questions?



Thank you.