

TAQOZ ROM GLOSSARY of WORDS

There are 432 words in TAQOZ ROM, so for mere humans too much to memorise. Instead, it's a good idea to keep this glossary by you when programming. The words are grouped under headings like Maths, Stack Operations to speed up searches. Please note that there are some differences to the ROM version and the enhanced RELOADED version although the bulk of the words work identically. I will try to mark those words that are different in case both versions are being used.

The experienced Forth programmer will notice many words that are unique to TAQOZ. TAQOZ is not intended to meet any international standard; and for a good reason. The non-standard word set is a very close fit around the unique P2 processor and its special circuits. This results in a smaller, more useful and efficient a tool to program P2. TAQOZ is also close enough to many Forth dialects to be quickly assimilated, aided by the glossary.

About the format of the word tables

The WORDS column shows TAQOZ words, with associated words grouped together
The STACK column shows the effect each word has on the data stack
The DESCRIPTION column briefly describes what the word does
The CONSOLE EXAMPLES column(s) show the TAQOZ prompt and user input followed by the response and a final "ok" as indication of acceptance. It may be confusing therefore for those unfamiliar with Forth-like languages as to what constitutes user input and what constitutes the response. Rather than explaining all this **the user input is colored in** as a guide.

Sample word description

Here this row covers both the FOR and the NEXT word since they are used together to form a simple loop.

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
FOR NEXT	(cnt --) (--)	Simple counted loop FOR. However the index I is available starting from 0.	TAQOZ# CRLF 3 FOR ." HELLO! " NEXT HELLO! HELLO! HELLO! ok

Data is referred to as bytes (8 bits), words (16 bits), longs (32 bits) and doubles (64 bits).

Boolean values of FALSE and TRUE are simply 0 or -1 (\$FFFF_FFFF) although in fact any non-zero value can be accepted as a boolean true but not necessarily suitable for bitwise operations such as AND. For instance **4 IF** and **8 IF** would both resolve as true however **4 8 AND IF** fails. Use 0<> to promote any non-zero to a full TRUE in such cases.

LIST OF DEBUGGING CONTROL KEY SHORTCUTS

ESC ESC ESC ESC	Reset (works even if TAQOZ is not checking the console)
ESC	Discard the current console line input
^C	Reset
^W	List WORDS in dictionary
^D	LSD Debugger in ROM (use ESC<CR> to return to TAQOZ)
^Q	Print data stack
^S	Clear data stack
^B	Block memory dump of first 64kB
^X	Re-eXecute last line (very useful)
^Z	Cold start Zap - wipe new code and dictionary - restore settings
^?	DEBUG List registers, current code an dictionary memory, and I/O

DEBUGGING

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
@WORDS	(-- adr)	Returns adr, the start of the dictionary	TAQOZ# @WORDS \$20 DUMP --- 0B3F9: 04 44 45 4D 4F 00 10 03 44 55 50 73 00 04 4F 56 '.DEMO...DUPs..OV' 0B409: 45 52 77 00 04 53 57 41 50 7E 00 03 52 4F 54 82 'ERw..SWAP~..ROT.' ok
WORDS	(--)	List the current dictionary of words ^W shortcut	
DUMP:	(--)	Use the following three words for byte, word, and long memory reads during a subsequent DUMP	pub SD DUMP: SDC@ SDW@ SD@ ; 0 \$100 SD DUMP
DUMP DUMPW DUMPL DUMPA DUMPAW	(adr bytes --)	Dump selected memory as:- bytes and ASCII words and ASCII longs and ASCII 64 ASCII characters wide (no hex) 128 ASCII characters wide (no hex)	TAQOZ# \$FC000 \$20 DUMP --- 000F_C000: 00 08 80 FF 3F 00 0C FC 32 C8 06 F6 1F 04 DC FC '....?...2.....' 000F_C010: 40 7E 74 FD 01 CA A6 F0 1F CA 26 F4 00 CA 62 FD '@~t.....&...b.' ok3 TAQOZ# \$FC000 \$20 DUMPL --- 000F_C000: FF80_0800 FC0C_003F F606_C832 FCDC_041F '....?...2.....' 000F_C010: FD74_7E40 F0A6_CA01 F426_CA1F FD62_CA00 '@~t.....&...b.' ok
QD QW	(adr --)	Quick Dump or Quick dump Words. List two lines of memory content	TAQOZ# pub DEMO 10 2* . ; --- ok TAQOZ# ' DEMO QW --- 01000: F80A 00AE DA82 0063 013F 1000 D88A 0063 '.....c.?.....c.' 01010: 0063 0000 0000 0000 0000 0000 0000 0000 'c.....' ok
COG LUT RAM SD SF	(--)	Modifies the next DUMP to use COG or LUT memory instead of the default HUB RAM. Resets to RAM after each DUMP	TAQOZ# 0 4 COG DUMP --- 00.0000: FDA0.0990 FD90.00BC 0000.0000 0000.0000 ok
.S	(--)	List the contents of the data stack Also accessible as the ^Q shortcut	TAQOZ# \$CAFEBABE DUP \$FFFF AND SWAP 16 >> .S --- DATA STACK x2 1 \$0000.CAFE 51966 2 \$0000.BABE 47806 ok
DEBUG	(--)	Dumps the various stacks, task registers, and quick dumps of sections of the current code and dictionary memory. ^? shortcut	
Isio	(--)	List the I/O, displays all 64 I/O pins and indicates what impedance the P2 senses on each pin	TAQOZ# Isio --- P:0000000000011111111122222222223333333333344444444445555555555566 P:01234567890123456789012345678901234567890123456789012345678901 =:~~~~~ ok
LAP .LAP .ms CNT@ LAP@	(--) (--) (-- n) (-- n)	Time the code between each LAP and report results with .LAP Report only milliseconds CNT@ reads the 32-bit CNT register LAP@ is used internally by .LAP	TAQOZ# LAP 1000 FOR NEXT LAP .LAP --- 32146 cycles = 401.825us ok TAQOZ# LAP 1000 FOR NEXT LAP .ms --- 401.825us ok
RESET	(--)	Reset the P2 chip ^C or esc esc esc esc shortcut	

THE STACKS

TAQOZ is based on Tachyon Forth which maintains a **data stack** in LUT memory in each cog but keeps a copy of the top four elements in fixed registers in cog memory for quick and efficient access. Therefore there are no (ugly) PICK and ROLL operators so as to encourage clean and efficient coding using a minimum of stack manipulation. The "top of stack" element or the last one pushed etc is referred to simply as "tos". The stack is a last in, first out store. Functions that require and pass parameters drawn from the top of the stack to usually no more than a depth of four elements.

TAQOZ has a **return stack** in which code return addresses are stored, so that words can be nested within words within words and so on to make a complete program.

In addition to the data and return stack, there is a dedicated **loop parameter stack** and a **branch stack** used internally by looping structures. Since the return stack is only used for return addresses we can then also access loops and parameters from any section of code that is not part of that loop.

In Forth source code, the data **stack effect** of a word is shown within () comment braces. e.g. (n -- w c) means a word consumes an ‘n’ long from the stack when it executes, leaving a ‘w’ word and ‘c’ byte or character on the stack afterwards.

The stacks store everything as 32 bit ‘longs’. In the stack effect comments, the following letters mean:-

- n - signed long (32 bit)
 - u - unsigned long (32 bit)
 - w - signed word (16 bit, sign extended to 32bits)
 - uw - unsigned word (16 bit, padded with leading 0s to 32 bits)
 - c - byte or char (8 bit, padded with leading 0s to 32 bits)
 - d - signed double (64 bit, stored as two longs on the stack, m.s. half nearest tos)
 - ud - unsigned double (64 bit, stored as two longs on the stack, m.s. half nearest tos)
 - adr - 20 bit address
 - str - the address of a zero delimited string
- Other mnemonics are used where useful, but are generally self-explanatory

STACK OPERATIONS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES	
ISP	(--)	Initialise the data stack to empty Shortcut: ^S		
2DUP	(n1 n2 -- n1 n2 n1 n2) or (d -- d d)	Make a copy of a double at tos		
DUP OVER 3RD 4TH	(n -- n n) (n1 n2 -- n1 n2 n1) (n1 n2 n3 -- n1 n2 n3 n1) (n1 n2 n3 n4 -- n1 n2 n3 n4 n1)	Make a copy of the 1st, 2nd, 3rd, or 4th tos	TAQOZ# 1234 5678 9012 3456 .S DATA STACK x4 1 \$0000.0D80 3456 2 \$0000.2334 9012 3 \$0000.162E 5678 4 \$0000.04D2 1234 ok	TAQOZ# 4TH .S DATA STACK x5 1 \$0000.04D2 1234 2 \$0000.0D80 3456 3 \$0000.2334 9012 4 \$0000.162E 5678 5 \$0000.04D2 1234
?DUP	(n1 -- n1 n1) else (n -- n)	Duplicate the tos only if it is non-zero (doing this can sometimes save having to specially drop a false flag as in: ?DUP IF <do something with the flag> THEN	TAQOZ# 1234 ?DUP 0 ?DUP .S DATA STACK x3 1 \$0000.0000 0 2 \$0000.04D2 1234 3 \$0000.04D2 1234 ok	
DEPTH	(-- n)	n = the current depth of the data stack (not including n)	TAQOZ# 1 2 3 ok	TAQOZ# DEPTH . 3 ok
DUPC@				
DROP 2DROP 3DROP	(n --) (n1 n2 --) or (d --) (n1 n2 n3 --)	Drop and discard the top elements of the data stack, either 1, 2, or 3 levels	TAQOZ# 1234 5678 9012 .S DATA STACK x3 1 \$0000.2334 9012 2 \$0000.162E 5678 3 \$0000.04D2 1234 ok	TAQOZ# 2DROP .S DATA STACK x1 1 \$0000.04D2 1234 ok
NIP	(n1 n2 -- n2)	Nip out the second element	TAQOZ# 1234 5678 9012 NIP .S DATA STACK x2 1 \$0000.2334 9012 2 \$0000.04D2 1234 ok	
SWAP	(n1 n2 -- n2 n1)	Swap the top two data elements	TAQOZ# 1234 5678 .S DATA STACK x2 1 \$0000.162E 5678 2 \$0000.04D2 1234 ok	TAQOZ# SWAP .S DATA STACK x2 1 \$0000.04D2 1234 2 \$0000.162E 5678 ok
2SWAP	(n1 n2 n3 n4 -- n3 n4 n1 n2) or (d1 d2 -- d2 d1)	Swap the top two doubles (pairs of longs)		
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotate the 3rd element to the top	TAQOZ# 1234 5678 9012 .S DATA STACK x3	TAQOZ# ROT .S DATA STACK x3

			1 \$0000.2334 9012 2 \$0000.162E 5678 3 \$0000.04D2 1234 ok	1 \$0000.04D2 1234 2 \$0000.2334 9012 3 \$0000.162E 5678 ok
-ROT	(n1 n2 n3 -- n2 n1 n2)	Rotate the top element to the 3rd position (Reverse ROT)	TAQOZ# 1234 5678 9012 .S DATA STACK x3 1 \$0000.2334 9012 2 \$0000.162E 5678 3 \$0000.04D2 1234 ok	TAQOZ# -ROT .S DATA STACK x3 1 \$0000.162E 5678 2 \$0000.04D2 1234 3 \$0000.2334 9012 ok
>L L>	(n --) (-- n)	Push n onto the loop stack Pop n from the loop stack		
>R R>	(n --) (-- n)	Push n onto the return stack Pop n from the return stack		

Note: The data stack is 4 levels deep in the cog and then implemented as a non-addressable LIFO stack in cog memory. TAQOZ words are optimized for these four fixed cog registers and to encourage efficient stack use no messy PICK and ROLL words are implemented. There are many words that also avoid pushing and popping the stack as this slows execution speed too. Try to factor words so that they use four or less parameters.

LOGIC (BITWISE)

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
AND ANDN OR XOR	(n1 n2 -- n3) (n1 n2 -- n3) (n1 n2 -- n3) (n1 n2 -- n3)	n3 = n1 and n2 n3 = n1 and not n2 n3 = n1 or n2 n3 = n1 xor n2	TAQOZ# 1011b 101b AND .BIN %1 ok TAQOZ# 1011b 101b NAND .BIN %1010 ok TAQOZ# %1000 %11 OR .BIN %1011 ok TAQOZ# %1010 %11 XOR .BIN %1001 ok
NOT	(n1 -- n2)	n2 = n1 all bits inverted	TAQOZ# %11 NOT .BIN %11111111111111111111111111111100 ok

SHIFTING

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
>> <<	(n1 cnt -- n2)	Shift n1 right or left cnt places	TAQOZ# \$F 2 >> . 3 ok TAQOZ# 3 2 << . 12 ok
ROL ROR	(n1 cnt -- n2) (n1 cnt -- n2)	n2 = rotate n1 left cnt times, with the ms bit rotating into the ls bit n2 = rotate n1 right cnt times, with the ls bit rotating into the ms bit	TAQOZ# \$F0000000 2 ROL HEX . \$C0000003 ok TAQOZ# \$15 3 ROR HEX . \$E0000001 ok
SAR	(n1 cnt -- n2)	Shift n1 cnt places right, arithmetically	TAQOZ# -255 4 SAR . -31 ok
2/ 4/ 8>> 9>> 16>>	(n1 -- n2)	Shift tos right by 1, 2, 8, 9, or 16 ** (2/ is equivalent to 1 >>) These are fast operations since no pushing or dropping is done.	TAQOZ# 32 4/ . 8 ok
2* 4* 8<< 9<<	(n1 -- n2)	Shift tos left by 1, 2, 8, or 9 **	TAQOZ# 3 8<< . 48 ok

** These instructions work with a single operand and return with a single operand. Because they do not push or pop they are useful, being both faster and more compact. For instance the word 8>> might simply be equivalent to 8 >> except it only needs a single wordcode and saves time by not having to push 8 and then pop it to discard afterwards since it is essentially a single PASM instruction.

CONVERSION AND MASKING

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
>9	(n -- 9bits)	9bits = n and \$1FF	
>B	(n -- byte)	to byte = n and \$FF	
B>L	(a b c d -- dcba)	byte to long : Merge four bytes into a long	
B>W	(a b -- word)	byte to word : Merge two bytes into a word	
W>L	(word1 word2 -- long)	words to long : Merge words into a long	
W>B	(word - - wordl wordh)	word to bytes : Split word into two bytes	
L>W	(long -- wordl wordh)	long to words : Split long into two words	
>N	(n -- nibble)	to nibble = n and \$F (n -- nibble)	
>W	(n -- word)	to word = n and \$FFFF	
BITS	(n1 cnt -- n2)	Extract lsb cnt bits from n1	TAQOZ# \$F0F0F0F0 5 BITS . 32 ok

L>S	(n -- lsb9 h)	Specialized operation for file system addresses	
REV	(n1 -- n2)	Reverse bits 0..32 -> 32..0 (n1 -- n2) Very useful for array address twiddling in fast fourier transforms	TAQOZ# 5 REV . 167772160 ok
>	(bitmask -- bitpos)	Encode - bitpos = the position of the msb set in bitmask	TAQOZ# %0100 > . 2 ok TAQOZ# 64 > . 6 ok
<	(bitpos -- bitmask)	Decode - bitmask = long with one bit set at bitpos position	TAQOZ# 3 < .BIN %1000 ok TAQOZ# 6 < .BIN %1000000 ok
S>#	(str -- u digits false)	Converts string at str to a number u and digit count digits, else returns false on fail	TAQOZ# " %1011" \$># . SPACE . --- 4 11 ok TAQOZ# " 12345678" \$># . SPACE . --- 8 12345678 ok

MATHS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
*/	(n1 n2 n3 -- res)	Multiply n1 by n2 and divide the 64-bit product by n3 for a 32-bit result	TAQOZ# 355 100000000 113 */ . --- 314159292 ok
+ - * /	(n1 n2 -- n3)	Add, subtract, multiply, divide	TAQOZ# 2 3 + . 5 ok TAQOZ# 12 -4 / . -3 ok
ABS	(n1 -- n2)	Absolute value of n1 - if n1 is negative then negate it to a positive number	TAQOZ# 4 ABS . 4 ok TAQOZ# -33 ABS . 33 ok
BOUNDS	(n1 n2 -- n1+n2 n1)	Add n1,n2 leaving n1 at tos	
MAX MIN	(u1 u2 -- u3)	Leave the maximum, minimum of u1 and u2, all unsigned	TAQOZ# 4 5 MAX . 5 ok TAQOZ# 33 99 MIN . 33 ok
MAXS MINS	(n1 n2 -- n3)	Leave the maximum, minimum of n1 and n2, all signed	
NEGATE -NEGATE ?NEGATE	(n1 -- n2) (n1 n2 -- n3) (n1 flag -- n2)	n2 = -n1 n3 = -n1 if n2 is negative, else n3=n1 n2 = -n1 if flag is true, else n2=n1	TAQOZ# 45 NEGATE . -45 ok TAQOZ# 54 -2 -NEGATE . -54 ok TAQOZ# -2 TRUE ?NEGATE . 2 ok
GETRND	(-- n)	Returns pseudorandom number *	
RND	(-- n)	Generate n, a pseudo-random long enhanced with the system counter	
SQRT	(u1 -- u2)	u2 = unsigned square root of u1	TAQOZ# 9 3 3 * * SQRT . 9 ok
UM*	(u1 u2 -- ud1)	Unsigned u1 * u2, resulting in unsigned 64 bit ud1	
UM//	(ud1 u2 -- rem quot)	Unsigned 64 bit ud1 divided by long u2 resulting in rem and quotient unsigned longs	
U/ U//	(u1 u2 -- u3) (u1 u2 -- rem quot)	Divide u1 by u2, all unsigned Divide u1 by u2, all unsigned	
//	(n1 mod -- rem)	Unsigned divide and return with remainder	
W*	(w1 w2 -- n)	n = w1*w2 (fast single PASM instruction)	
1+ 2+ 4+	(n1 -- n2)	Increment the tos element by 1, 2, or 4 *	TAQOZ# 1234 2+ . 1236 ok
1- 2-	(n1 -- n2)	Decrement the tos element 1 or 2 *	TAQOZ# 1234 2- . 1232 ok

* These instructions work with a single operand and return with a single operand. Because they do not push or pop they are useful, both faster and more compact. For instance the word 1+ might simply be equivalent to 1 + except it only needs a single wordcode and saves time by not having to push 1 and then pop it to discard afterwards since it is essentially a single PASM instruction.

MODIFIERS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
KB MB M	n1 -- n2 n1 -- n2 n1 -- n2	n2 = n1*1,024 n2 = n1*1,048,576 n2 = n1*1,000,000	TAQOZ# 128 KB . --- 131072 ok

STRINGS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
[“]	(--)	During compilation, displays the following string in the input stream until a terminating “	
“	(-- str1)	Compile a string in the input stream up to the trailing " and leave the address of the null terminated string on the stack.	TAQOZ# " hello world" \$10 DUMP --- 01002: 68 65 6C 6C 6F 20 77 6F 72 6C 64 00 10 F8 44 D8 'hello world...D.' ok
\$!	(str1 str2 --)	Store string at hub memory address str1 at address str2	TAQOZ# 16 BYTES mystring ok TAQOZ# “ Hello” mystring \$! ok
GET\$	(-- str)	Build a delimited word in wordbuf for wordcnt and return immediately upon a valid delimiter	
LEN\$	(str - n)	n = length of string at addr str	TAQOZ# mystring LEN\$. 5 ok
NULL\$	(-- str)	An empty string at address str	
\$=	(str1 str2 -- flag)	flag = true if string at hub memory address str1 is identical to string at str2, else flag = false	TAQOZ# mystring mystring \$= . -1 ok
\$>#	(str -- n digits) or (str -- flag)	Try to convert string at address str to number n and digits else return flag = false	TAQOZ# " \$123.456" \$># . SPACE .L --- 6 \$0012_3456 ok

Note: All TAQOZ strings are 0 delimited

DEFINING NEW VARIABLES

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES	
byte <variable name> word <variable name> long <variable name>	(--) (--) (--)	Define new byte variable Define new word variable Define new long variable	TAQOZ# --- defining vars TAQOZ# byte MYBYTE ok TAQOZ# word TIMEOUT ok TAQOZ# long SAMPLE ok	TAQOZ# --- writing to a long var. TAQOZ# 2000 SAMPLE ! ok TAQOZ# --- reading a long var. TAQOZ# SAMPLE @ . 2000 ok
bytes <variable name> words <variable name> longs <variable name>	(n --) (n --) (n --)	Define new byte array variable Define new word array variable Define new long array variable All these create an array of size n bytes, words or longs	TAQOZ# 4 longs BUFFER ok	TAQOZ# pub INITBUFFER (--) TAQOZ# --- set all entries to 100 TAQOZ# 4 0 DO TAQOZ# 100 BUFFER I 4* + ! TAQOZ# LOOP ; ok
org	(adr --)	Set data pointer used for defining variables to hub memory adr (for example to reclaim data space when reloading source code)	TAQOZ# IFDEF *MANDELBROT* TAQOZ# Cx org TAQOZ# --- Cx is the 1st variable defined in the code below this TAQOZ# FORGET *MANDELBROT* TAQOZ# } ok	
res	(n --)	Takes next word in input stream as a name and reserves n bytes of hub memory	TAQOZ# 8 res fnam ok	
VAR <variable name>		Create a long variable in code space (special use)		

Note 1: When variables like BUFFER execute, they leave the address of the variable at tos (-- adr)

Note2: Remember that addresses point to bytes in memory, so members of an array defined with 'longs' have addresses spaced 4 apart, 'words' 2 apart, 'bytes' 1 apart. Hence the 4* word in the example above to calculate the long address.

Note 3: None of these words initialise the variable, they just allot the required memory.

DEFINING NEW CONSTANTS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
<code>:=</code>	<code>(n --)</code>	Create a long constant using the name next in the input stream	TAQOZ# 1 := RED ok TAQOZ# RED . 1 ok TAQOZ# 2 ' RED ==! Ok
<code>==!</code>	<code>(n adr --)</code>	Change the value of constant at adr to n	TAQOZ# RED . 2 ok
DATCON		Same as <code>:=</code> to compile a constant but recognized by FORGET	

DEFINING NEW LITERALS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
	(c --)	Compile byte c into code memory	
	(w --)	Compile word w into code memory	
,	(n --)	Compile long n into code memory	

PREDEFINED TASK VARIABLES

Each cog may have its own set of variables that are offset from a base address (that address is obtained by using REG - see below)
This is so that any cog running TAQOZ may have different I/O devices selected etc.

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
REG	(index -- adr)	Find the address of register 'index'	
char	(-- adr)	Points to the last character sent via KEY	
delim	(-- adr)	Points to the word delimiter (normally space) plus backup byte with delimiter detected (SP,TAB,CR etc)	
names	(-- adr)	Points to the start of the latest name field in the dictionary (builds down)	
uemit	(-- adr)	Vector that points to cfa of current EMIT routine (0=console=(EMIT))	
ukey	(-- adr)	Vector that points to cfa of current KEY routine (0=console=(KEY))	

Note: There are more registers available, the above are all that are defined in TAQOZ ROM. The user may create more words to access the extra registers with the aid of the ROM assembly code listing. TAQOZ RELOADED includes more words for register access too.

PREDEFINED CONSTANTS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
ON TRUE -1 OFF FALSE	(-- n) (-- n)	n = -1 n = 0	TAQOZ# ON OFF .S DATA STACK x2 1 \$0000.0000 0 2 \$FFFF.FFFF -1 ok
CLKHZ	(-- n)	n = 20,000,000	

Note: Constants can be changed using ==!

DEFINING NEW WORDS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES	
[]		flag that we have entered a definition end definition and lock allocated bytes		
pub :	(--)	Create new word as public using the next word in the input stream as the name	TAQOZ# pub HITHERE ." Hello World" ; ok TAQOZ# : HITHERE2 ." Hello World" ; ok	TAQOZ# HITHERE Hello World ok
pri	(--)	Create a new word using the next word in the input stream as the name; It is marked private so a RECLAIM can remove them from the dictionary when required. The private words will still function, but can no longer be used to make new words. Keeps the dictionary uncluttered from unnecessary detail	TAQOZ# pri HIAGAIN ." Hello the World" ; ok	TAQOZ# HIAGAIN Hello the World ok TAQOZ# RECLAIM ok TAQOZ# HIAGAIN ?? hiagain not found ok
pre	(--)	Create a new preemptive word that has the "immediate" attribute set so that it executes immediately at compile time (words like IF ELSE THEN etc)	TAQOZ# pre <new word name> more TAQOZ words ; ok	
[C]	(--)	Force compilation of the next word in the input stream even if it was defined as preemptive		
;	(--)	Compile an EXIT word and ends definition - does not execute	See examples above	
ALLOT	(bytes --)	Allot n bytes of code memory - advances "here"		

ASM		Execute following code as hubexec assembly code.		
CREATE\$ CREATE	(str --) (--)	Creates a new word in the dictionary using the string at address str. If string is empty no new word is created Creates a new word in the dictionary using the name that follows in the input stream		
CFA	(nfa -- cfa)			
CPA	(nfa -- cpa)	The CPA is the address of the word code stored in the name field header that points to the code to execute		
GRAB		IMMEDIATE --- executes preceding code to make it available for any immediate words following		
NFA'	(-- nfa false)	Takes the next word in the input stream and finds the word address in the dictionary. If not found returns false (0)	TAQOZ# NFA' OVER \$10 DUMP 00.B406: 04 4F 56 45 52 74 00 04 53 57 41 50 7B 00 03 52 .OVERT..SWAP{..R ok	
TAQOZ END	(--) (--)	Marks the start of a block of source code to be compiled in block mode Marks end of block load mode		
[W]		append this wordcode to next free code location + append EXIT (without counting)		

CONDITIONAL DEFINING

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
IFDEF IFDEF	(--)	If the next name in the input stream is already DEFINED then process all input stream that follows up to the terminating curly brace } If next name in the input stream is NOT DEFINED then process all source between here and the matching curly brace }	TAQOZ# IFDEF *BREAKOUT* TAQOZ# bk org TAQOZ# FORGET *BREAKOUT* TAQOZ# } ok

DICTIONARY

WORDS	STACK	DESCRIPTION
HERE	(-- adr)	Point to the next compilation location
@CODES @DATA @WORD		At temp code pointer (builds up in memory) At data pointer (builds up) At word pointer (dictionary builds down)
@HERE	(-- atradr)	Variable which holds the code pointer HERE
@WORDS	(-- namadr)	Point to the latest entry in the names dictionary TAQOZ# @WORDS QD --- 0B400: 03 44 55 50 73 00 04 4F 56 45 52 77 00 04 53 57 '.DUPs..OVERw..SW' 0B410: 41 50 7E 00 03 52 4F 54 82 00 04 2D 52 4F 54 81 'AP~..ROT...-ROT.' ok
FORGET	(--)	Takes next word in input stream, and forgets all names in the dictionary from this name onwards - from most recent instance of the name. Also resets code pointers to the code pointer of the word.
SEARCH	(cstr -- nfaptr)	Search the dictionaries for cstr which points to a counted word string constructed as count+string+null

COMPARISON

WORDS		DESCRIPTION	CONSOLE EXAMPLES
0= 0<> 0<	(n -- flag)	Compare tos with zero	TAQOZ# 0 0= . -1 ok TAQOZ# 0 0<> . 0 ok
= <> < > U< U> <= =>	(n1 n2 -- flag) (u1 u2 -- flag) (n1 n2 -- flag)	Compare top two stack values Compare two unsigned longs Compare top two stack values	TAQOZ# 1 1 = . -1 ok TAQOZ# 1 1 <> . 0 ok
WITHIN	(val min max -- flag)	Return with flag true if val is within min and max (inclusive, not ANSI)	TAQOZ# 3 1 5 WITHIN . -1 ok TAQOZ# 3 3 7 WITHIN . -1 ok TAQOZ# 67 3 7 WITHIN . 0 ok

BRANCHING

Branching words are preemptive and compile conditional and unconditional relative jumps. At compile time words such as IF ELSE BEGIN WHILE will leave encoded address values on the stack but these are checking by their corresponding words.

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
IF ELSE THEN	(cond --) (--) (--)	Conditional execution IF	
BEGIN AGAIN UNTIL WHILE REPEAT	(--) (--) (cond --) (cond --) (--)	Start a repeated conditional loop Repeat forever Repeat until condition false Repeat while condition true End a WHILE loop	BEGIN xxx AGAIN BEGIN xxx cond UNTIL BEGIN xxx cond WHILE yyy REPEAT
GOTO			
EXIT	(--)	Exit word now - pops return stack into IP	
0EXIT ?EXIT	(cond --) (cond --)	Exit word if cond = 0 / false Exit word if cond = true	

CASE

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
BREAK	(--)	Stop executing the CASE code and EXIT	TAQOZ# pub CASETEST (val --) TAQOZ# --- print 'val' as a word TAQOZ# SWITCH TAGOZ# 1 CASE ." one" BREAK TAQOZ# 2 CASE ." two" BREAK TAQOZ# 3 CASE ." three" BREAK ; ok TAQOZ# 2 CASETEST two ok TAQOZ# 1 CASETEST one ok
CASE	(val1 --)	Execute the following code up to BREAK if val1 = SWITCH val2	
SWITCH	(val2 --)	Stores val2 ready for comparison with a number of CASEs	
CASE@	(-- val2)	Returns the value stored by the last SWITCH word	
CASE=	(val -- flag)	flag set true if val = value stored by the last SWITCH word	
CASE>	(from to -- flag)	flag set true if val stored by the last SWITCH word is within range 'from' - 'to', inclusive	

VECTORED EXECUTION

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
AUTO	<name>	Set the autostart vector to the following name in the input stream (Remember to backup the code & settings)	
‘	(-- adr false)	Takes the next word in the input stream, looks it up in the dictionary and returns the execution adr of the word else if not found returns false	TAQOZ# ' DUP .W --- \$0073 ok TAQOZ# ' !qwq!# .W --- \$0000 ok
CALL	(adr --)	Call the code field address on the top of stack	
JUMP	(adr --)	Jump to code field address on top of the data stack, doesn't save return address	
NOP	(--)	No Operation - convenient place holder or very short delay	TAQOZ# LAP NOP LAP .LAP --- 25 cycles = 312ns ok

LOOPING

Loop words compile directly like every other non-preemptive word so they do all their work at run-time. DO will stack the parameters onto the loop stack and also push the IP onto the branch stack so that LOOP can simply use this copy to loop back to. FOR NEXT works in a similar fashion.

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
DO	(limit start --)	Execute code between DO LOOP until the index equals the limit.	TAQOZ# 8 0 DO I . LOOP 01234567 ok
ADO	(start cnt --)	There is not much ADO about this, just combines BOUNDS and DO	TAQOZ# 0 8 ADO I . LOOP 01234567 ok
LOOP	(--)	Increment the index and loop if not at limit yet, otherwise unstack and continue.	
+LOOP	(cnt --)	+LOOP adds a cnt to the index instead of 1.	TAQOZ# 0 8 ADO I . 2 +LOOP 0246 ok
FOR NEXT	(cnt --) (--)	Simple counted loop FOR	TAQOZ# 3 FOR ." N" I . SPACE NEXT --- N3 N2 N1 ok
LEAVE	(--)	Set the index to limit-1 so that it will LEAVE the loop at the next LOOP or +LOOP executed	
?NEXT	(flag --)	Exit FOR NEXT loop early if flag is true (non-zero)	
I J	(-- n) (-- n)	I returns the current loop index J returns the index for the next outer loop.	
I+	(n -- n+l)	Add loop index to offset (Faster than I +)	TAQOZ# 4 FOR \$20 I+ . SPACE NEXT --- 36 35 34 33 ok
IC@	(-- c)	Fetch a byte using the current loop index I as the address	TAQOZ# " HELLO" 6 ADO IC@ .BYTE SPACE LOOP --- 48 45 4C 4C 4F 00 ok

TIMING

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
us ms s	(n --)	Wait for microseconds, milliseconds, or seconds	TAQOZ# CNT@ 100 ms CNT@ SWAP - . 8000552 ok
LAP .LAP	(--)	Timing code between each LAP and report results with .LAP. CNT@ reads the 32-bit CNT and LAP@ is used internally by .LAP	TAQOZ# LAP 1234 5678 * LAP .LAP 242 cycles = 3.25us ok
CNT@ LAP@	(-- val)		
ns	(ns -- cycles)	Convert nanosecond count to clock cycles	TAQOZ# CLKHZ . --- 20000000 ok TAQOZ# 5000 ns . --- 100 ok
WAIT	(n --)	Wait for n clock cycles to pass (uses P2 WAITX)	TAQOZ# LAP 1000 WAITX LAP .LAP --- 1145 cycles = 14.312us ok TAQOZ# LAP 10000 WAITX LAP .LAP --- 10146 cycles = 126.825us ok
CLKDIV	(n --)	Set clkdiv	
RCSLOW	(--)	Change P2 clock to RCSLOW (20kHz)	

MEMORY

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
ALIGN	(adr1 n -- adr2)	adr2 = adr1 realigned on an n'th byte boundary	TAQOZ# 0 4 ALIGN . 0 ok TAQOZ# 4 4 ALIGN . 4 ok TAQOZ# 1 4 ALIGN . 4 ok TAQOZ# 5 4 ALIGN . 8 ok
ERASE FILL	(adr cnt --) (adr cnt c --)	Erase hub memory starting at adr for cnt bytes, filling with byte 0 Fill hub memory started adr for cnt bytes, with c byte	TAQOZ# \$4.0000 \$10 \$FA FILL ok TAQOZ# \$4.0000 \$20 DUMP 04.0000: FA FA FA FA FA FA FA FA FA FA FA FA FA FA FA FA 04.0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ok
CMOVE <CMOVE	(src dst cnt --) (src dst cnt --)	Move hub memory bytes starting from source to destination by cnt bytes Same as CMOVE, in reverse address order (Use one or the other when the two blocks overlap each other)	TAQOZ# @NAMES \$4.0000 \$20 CMOVE ok TAQOZ# \$4.0000 \$20 DUMP 04.0000: 03 54 52 49 80 86 1F 03 53 41 57 80 80 1F 03 44 .TRI....SAW....D 04.0010: 55 50 80 6B 00 04 32 44 55 50 80 6D 00 04 4F 56 UP.k..2DUP.m..OV ok
C@ W@ @	(adr -- c) (adr -- w) (adr -- n)	Fetch a byte, word, or long from hub memory	TAQOZ# @NAMES \$10 DUMP 00.CFF2: 03 54 52 49 80 86 1F 03 53 41 57 80 80 1F 03 44 .TRI....SAW....D ok TAQOZ# @NAMES C@ .BYTE 03 ok

C@++	(adr1 -- n adr2	Fetch a byte from hub memory at adr1 and increment the address to adr2 leaving it tos	TAQOZ# \$1000 C@++ .S DATA STACK x2 1 \$0000.005C 92 2 \$0000.1001 4097 ok
C! W! !	(c adr --) (w adr --) (n adr --)	Store a byte, word, or long to hub memory	TAQOZ# \$DEADBEEF \$F000 ! ok TAQOZ# \$F000 \$10 DUMP 00.F000: EF BE AD DE 00 00 00 00 00 00 00 00 00 00 00 00 ok
C+! W+! +!	(c adr --) (w adr --) (n adr --)	Add parameter directly to hub memory location adr	TAQOZ# \$600 \$F000 +! ok TAQOZ# \$F000 \$10 DUMP 00.F000: EF C4 AD DE 00 00 00 00 00 00 00 00 00 00 00 00 ok
COG@ COG! LUT@ LUT!	(adr -- n) (n adr --) (adr -- n) (n adr --)	Read long at COG memory adr Write long to COG memory adr Read long at LUT memory adr Write long to LUT memory adr	TAQOZ# 0 COG@ .LONG FDA0.0990 ok TAQOZ# 0 LUT@ .LONG 0000.04D2 ok TAQOZ# \$CAFEBABE 2 COG! 2 COG@ .LONG CAFE.BABE ok
C++ C-- W++ W-- ++ --	(adr --) (adr --) (adr --) (adr --) (adr --) (adr --)	Increment byte at adr hub memory Decrement byte at adr hub memory Increment word at adr hub memory Decrement word at adr hub memory Increment long at adr hub memory Decrement long at adr hub memory	
C~ C~~ W~ W~~ ~ ~~	(adr --) (adr --) (adr --) (adr --) (adr --) (adr --)	Clear byte in hub memory (=0) Set all bits of byte in hub memory (=\$FF) Clear 16-bit word in hub memory Set all bits of 16-bit word in hub memory Clear long in hub memory Set all bits of long (= \$FFFFFFFF)	TAQOZ# \$10000 ~ \$10000 @ .L --- \$0000_0000 ok TAQOZ# \$10000 ~~ \$10000 @ .L --- \$FFFF_FFFF ok
DATA?	(addr cnt -- flag)	Test for non-zero data starting at memory addr for cnt longs	TAQOZ# \$10000 \$1000 ERASE --- ok TAQOZ# \$10000 \$1000 DATA? . --- 0 ok TAQOZ# \$10100 C~~ \$10000 \$1000 DATA? . --- 255 ok
BIT! SET CLR SET?	(mask adr --) (mask adr --) (mask adr -- flag)	Set 'mask' bits in the byte at adr in hub Clear 'mask' bits in the byte at adr in hub Test the 'mask' bits in the byte at addr and return with state	

SERIAL FLASH

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
BACKUP	(--)	BACKUP the TAQOZ system with any added user words into the last 64KB of 1MB of Flash	
RESTORE	(--)	Restore the backup after reset	
.SF	(--)	Displays flash memory manufacturers' I.D. serial number etc. Useful for checking flash memory communication is working	TAQOZ# .SF \$EF70_1800 \$0F0B_2826 \$E468_5CF4 ok
SF	(--)	Select SPI Flash as the source for DUMP	
SF?	(-- stat)	Read SPI Flash status register	
SF@	(adr -- n)	Returns long from flash address adr	
SFC@	(adr -- c)	Returns byte from flash address adr	
SFW@	(adr -- w)	Returns word from flash address adr	
SFER4 SFER32 SFER64	(adr --)		
SFERASE	(--)		
SFINS			
SFPINS	(-- n)	Returns constant n = \$3d3a3b3c , the four pin numbers of the Flash memory interface	
SFJID	(-- n)	Read serial Flash Jedec ID	
SFRDS			
SFSID	(-- n)	Read serial Flash serial number	
SFWD			
SFWE			
SFWRPG	(src dst --)		
SFWRS	(hubsrc sfdst cnt --)		

SD CARD (FAT32)

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
@BOOT	(sector str --)		
@FAT	(fat# -- sector)		
@ROOT			
!SD	(-- ocr false)	Initialise the SD card in SPI mode and return with the OCR, else false if an error occurs	
!SX			
ACMD	(data acmd -- res)		
cid			
CMD	(data cmd -- res)		
DIR	(--)	lists info about the card, FAT32 and directory entries	
fat	(-- n)	Constant n = \$FF00 the start location of the file allocation table	
FGET			
FLOAD		loads a TAQOZ source file	
FLUSH	(force --)		
FOPEN	(addr --)	Opens file by sector address visible in listing from DIR	TAQOZ# \$0001_518C FOPEN --- ok TAQOZ# 0 \$20 SD DUMP --- 00000: 54 68 65 20 50 72 6F 6A 65 63 74 20 47 75 74 65 'The Project Gute' 00010: 6E 62 65 72 67 20 45 42 6F 6F 6B 20 6F 66 20 57 'nberg EBook of W' ok
FREAD	(sdsrsrc hubdst bytes --)		
FWRITE	(hubsrc sddst bytes --)		
MOUNT	(--)	Mounts SD card and reports type, s/n, formatted capacity	TAQOZ# MOUNT .SDSC64G 6269_8201 P2 CARD 32k 60,890M ok
SD			
SDBUF			
sdpins	(-- n)	Returns constant n = \$3c3a3b3d , the four pin numbers of the SD card interface	
SD@ SD!	(xaddr -- long) (long xaddr --)		
SDC@	(xaddr -- byte)		
SDW@	(xaddr -- word)		
SD?	(-- flag)		
SDADR	(xaddr -- addr)		
SDRD	(sector dst --)		
SDRDS	(sector dst cnt -- crc false)		
SDWR	(src sect -- flag)	Write from src to xdst in the SD	
SDWRS	(hubsrc sdadr cnt --)		
SECTOR	(sect -- sdbuf)		

DEFAULT RADIX WORDS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
HEX DECIMAL BINARY	(--)	Switch the number base to hexadecimal Switch the number base to decimal Switch the number base to binary	TAQOZ# 23 DUP . 23 ok TAQOZ# HEX ok TAQOZ\$. AC ok

Note: The TAQOZ prompt suffix changes to show the default radix. \$=hex #=decimal %=binary

NUMBER I/O

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
. PRINT U.	(n --) (n --) (u --)	Display a signed number n Display a signed number n Display unsigned number u	
<# # #S HOLD #>	 (c --) (-- str)	Start a new formatted number string Insert the next digit of the number being displayed into the formatted number string Insert all remaining significant digits of the number Insert the character on the stack into the formatted string Terminate the formatted number leaving the address of the formatted number string at tos, ready for display	TAQOZ# pub TODOLLARS (n --) TAQOZ# <# # # 46 HOLD #S 36 HOLD #> ; ok TAQOZ# 1234 TODOLLARS PRINT\$ \$12.34 ok
.ADDR	(n --)	At column zero of the display, print n as 5 digit hex followed by : and <space>, as used in memory dumps	TAQOZ# HEX 1FF .ADDR TAQOZ# 001FF: ok
.AS			
.AS”	(n --)	Display n with format as per string spec: # Convert one digit (default is decimal) ~ Toggle leading zero suppression \ pad leading zeros with spaces \$ Hexadecimal * Convert all remaining digits 4 Convert 4 digits Terminated with “	TAQOZ# 32767 .AS” \$ #####“ \$EFFF ok
.B .BYTE	(n --)	Display n as two digit hex byte 00-FF	TAQOZ# 253 .B DF ok
.BIN	(n --)	Display n in binary with leading %	TAQOZ# 33 .BIN %100001 ok
.DEC	(n --)	Display n in decimal with at least a single digit	
.DECL	(n --)	Display n as decimal "###,###,###,####"	
.DEC4	(n --)	Display n as decimal "####" (n --)	
.H	(n --)	Display n as hex nibble 0-F	
.L .LONG	(n --)	Display n as eight digit hex 00000000 - FFFFFFFF	
.W .WORD	(n --)	Display n as four digit hex 0000-FFFF	

CHARACTER I/O

WORDS	Stack	DESCRIPTION	CONSOLE EXAMPLES
."	(--)	Print text in input stream until terminating “	TAQOZ# ." Hello” Hello ok
EMIT CONEMIT EMITS CLS CRLF CR SPACE SPACES	(c --) (c --) (c cnt --) (--) (--) (--) (--) (n --)	Display the character c from the input stream via vector ‘uemit’ Like EMIT, but always uses the default console port Display character c, cnt times Clear the screen (ANSI terminal) Send Carriage return line feed Send carriage return Display one space char Display n spaces	
KEY CONKEY WKEY	(-- c false) (-- c false) (-- c)	Return with the next character from the input stream via vector at ‘ukey’ or else a flag = false (0) Like KEY, but always uses the default console port Wait for a character, returned as c	
CTYPE	(str cnt --)	Display cnt chars, starting at address str	
ERROR		count errors and abort and display error message	
KEY!	(c --)	Force a character as the next key read in the input stream	
CON	(--)	Reset the uemit vector so that EMIT and friends output to the default console port again	
PRINT\$	(str --)	Display string at address str	TAQOZ# 16 BYTES mystring ok TAQOZ# “ FRED” mystring \$! ok TAQOZ# mystring PRINT\$ FRED ok
<D>	(d -- n)	Store m.s. long of double d for formatting, leaving the l.s. long n	
SPIN	(--)	Emit the next character in a spinner sequence (one of / - \) using backspace to reposition	TAQOZ# pub HANGON (--) --- Show user something’s in progress TAQOZ# 100 0 DO TAQOZ# SPIN 500 ms TAQOZ# LOOP ; Note: In practice, there would be other code within the loop, checking for some change in state
TERM	(--)	Warm start TAQOZ	

NOTE: **Diverting input and output stream** - CONEMIT and CONKEY will always use the default console port. All other character and number I/O words may be diverted to input or output from other hardware. User provided versions of EMIT or KEY may be written, and their code field addresses replace the values at uemit or ukey to immediately use the new hardware. To reset EMIT or KEY to use the default console again, set uemit or ukey back to zero.

I/O PORTS

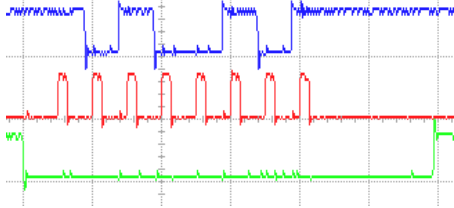
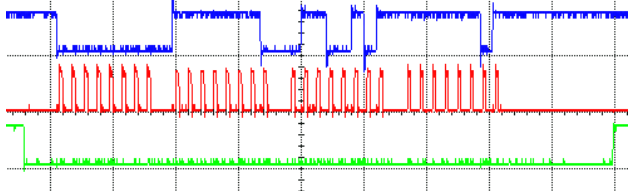
WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
HIGH LOW FLOAT	(pin --)	Set the direction and output of a pin	TAQOZ# 4 HIGH 4 PIN@ . -1 ok TAQOZ# 4 LOW 4 PIN@ . 0 ok
H L F R T	(--) (--) (--) (-- flag) (--)	Set a preselected pin to a high output Set a preselected pin to a low output Set a preselected pin to a floating state(high impedance) Set a preselected pin to input, flag = input state	TAQOZ# 4 PIN ok TAQOZ# L R . 0 ok TAQOZ# H R . -1 ok

SMARTPINS

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
WRPIN WXPIN WYPIN RDPIN RQPIN AKPIN	(dst --) (dst --) (dst --) (-- res) (-- res) (--)	Same as assembler - write dst to mode register of smart pin S[5:0], acknowledge smart pin. Write dst to parameter "X" of smart pin S[5:0], acknowledge smart pin. Set smart pin S/# parameter Y to dst res = smart pin S/#, ack Res = smart pin S/#, don't ack /#, acknowledge smart pin	
COM NONE	(--)	direct terminal output to a smartpin (after init)	
WRACK	(n --)	Write smartpin data, wait for empty then acknowledge	
WAITPIN TXDAT DUTY NCO	(buf cnt --) (val --) (n --)	Write buffer direct to WYPIN Set Numerically Controlled Oscillator duty cycle Start Numerically Controlled Oscillator, n sets relationship to system clock - see P2 manual	TAQOZ# 48 PIN \$8000_0000 NCO ok TAQOZ# --- pin 48 emits ½ system clock ok
HILO	(high low --)	Specify High and Low times for subsequent PULSE or PULSES words	
HZ KHZ MHZ BLINK	(freq --) (--)	Set PIN to NCO mode and output the frequency BLINK sets an output to 2 Hz - for LED flashing	TAQOZ# 4 PIN 10 MHZ ok TAQOZ# 5 PIN BLINK ok
MUTE	(--)	Cancel the presently selected smartpin activity	
PIN @PIN	(pin --) (-- pin)	Select a smartpin to use, this setting is remembered until another pin is selected Read smartpin setting back	TAQOZ# 5 PIN @PIN . 5 okTXD
PULSE	(--)	Generate one pulse according to specified high and low times	
PULSES	(cnt --)	Generate cnt pulses according to specified high and low times	
PW	(width --)		
PWM SAW	(duty frame div --) (duty frame div --)	Set pin to PWM triangle mode. Turn on for duty cycles, off for (frame - duty) cycles, 'div' is a prescaler (more=slower) Use SAW instead for sawtooth.	TAQOZ# 4 PIN 1000 4000 1 PWM ok
SEROUT			
SETDACS	(n --)	DAC3 set to bits 31-24, DAC2 set to bits 23-16, DAC1 set to bits 15-8, DAC0 set to bits 7-0 of n	
BAUD	(baud mode --)		
TXD	(baud --)	Setup a pin as an asynchronous transmitter using the supplied parameter as a baud rate	TAQOZ# 34 PIN 8 BIT 40 M TXD @NAMES \$10 TXDAT ok
RXD	(baud --)	Setup a pin as an asynchronous receiver using the supplied parameter as a baud rate	
BIT	(n --)	Specify the data bits (1..32) to be used whenever a pin is switched to RXD or TXD modes. If this is not specified it will default to 8	TAQOZ# 8 BIT 115200 TXD ok

SPI

Serial Peripheral Interface is implemented as cog instructions that handle half-duplex read or write in increments of 8-bits. Once the SPI pins have been set then any read or write will automatically enable the Chip Enable and only requires a SPICE to turn it back off if required. Sometimes all that is required here is to SPICE just before starting a new command to reset the slave.

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
SPIWB SPIWW SPIWC	(n1 -- n1)	Write Byte or Word or SD Command to SPI pins. 200ns/div example sending \$A5 then SPICE 	TAQOZ# \$12345678 SPIWB .S DATA STACK x1 1 \$0000.0078 120 ok
SPIWL	(n1 -- n1)	Write Long to SPI pins	
SPIWM	(n1 -- n1)	Write 24 bits to SPI pins	
SPIRD	(n1 -- n2)	Read 8-bits left into n1 so that n2 = n1<<8+new. Four successive SPIRDs will receive 32-bits	
SPIRDL	(n1 -- n2)	Read a long from SPI, overwriting n1??	
SPICE	(--)	Release SPI chip enable line (automatically enabled on any SPI operation)	
SPIPINS	(&cs.mi.mo.ck --)	Setuppins to be used by SPI; parameter encoded by byte fields - use & prefix to force decimal bytes	TAQOZ# &32.33.34.35 SPIPINS ok
SPIRX SPITX SPITXE	(adr cnt --)	Send or receive between memory and SPI 500ns/div example writing 4 bytes from memory 	TAQOZ# &32.33.34.35 SPIPINS ok TAQOZ# 0 4 SPITXE SPICE ok

COG/HUB

WORDS	STACK	DESCRIPTION	CONSOLE EXAMPLES
COGID COGINIT	(-- cog#) (adr cog# --)	Return current cog# Same as COGINIT in PASM - also saves information in cog TASK block at adr	
COGSTOP COGATN	(cog# --) (maskw --)	Stop cog# Strobe the “attention” event flag for all cogs whose corresponding bits are high in maskw (bit 0 - 8 for P2)	
NEWCOG	(n --)	An idling instance of TAQOZ is loaded and run in cog#n	
POLLATN SETEDG POLLEDG	(-- flag) (edge pin --) (-- flag)	flag = ATN attention event flag, ATN event flag then cleared	
HUBSET WP WE REBOOT	(n --)	HUBSET - configure various global circuits Write Protect ROM Write Enable ROM	
TASK	(cog# -- adr)	Index the cog’s task variable and return with its address	
WAITCNT	(--)	Continue from last count (must be called before target is reached)	
WAITX	(delta --)	Calculate and set the cnt delta and waitcnt	

COMMENTS

WORDS	DESCRIPTION	CONSOLE EXAMPLES
{ <multiline comment the compiler ignores> }	Ignores all text up to the matching curly brace }	
--- <rest of line ignored> \ <rest of line ignored>	Ignores the rest of the line Ignores the rest of the line, does not echo	
(<words to ignore>)	Ignores characters until closing brace) Use on one line only, else use { } multiline	

Booting

- [GETTING THE TAQOZ ROM PROMPT UP](#)
- **SD-Boot FAT32:** The root directory is searched for file "_BOOT_P2.BIX", and if not found then the root directory is searched for file "_BOOT_P2.BIY" (upper case is required). If the file is found, then all files' data is loaded into HUB \$0. Maximum file size is 512KB-16KB = 496KB. Once loaded into HUB, the first 496 longs will be copied to COG \$0 and then a JMP #\$0 will be executed. Note: The files' data must be in contiguous sectors.
- **SD-Boot:** The SD boot will check for valid P2 Boot Data on the MBR sector (sector 0), and if not found it will try the VOL sector (provided the SD Card is in FAT32 format).
 - 1. Valid Boot Data is "Prop" at offset \$17C on the MBR/VOL sector...The sector (512 bytes) will be read into Hub \$0, followed by a JMP \$080. Only code in sector byte offset \$080-\$17F is considered valid but this program may know that other areas are also valid.
 - 2. Valid Boot Data is "ProP" at offset \$17C on the MBR/VOL sector...The long at offset \$174 is used as the sector start address, and the long at \$178 is used as the maximum filesize (in bytes) is 512B-16KB = 496KB for the data to be loaded into Hub, followed by a `JMP $000`.
 - These two methods also cover the case where the SD Card is not formatted as FAT32 (eg exFAT). The boot sector(s) MBR and/or VOL must be specifically written with a program such as HxD on Windows 10.