TACHYON Forth version 5v7 GLOSSARY------ for the PARALLAX PROPELLER P1

The purpose of this glossary is to provide a sorted list of words, their stack effects, and a short description. It is recommended to go to the relevant source code to glean more information if necessary.

Stack notation shows input parameters with the rightmost being the top of the stack, input parameters on the left, output on the right, separated by --
Where a word has a compile time and a run time behaviour, the two stack effects are shown as <compile effect> ; <runtime effect>

e.g. (a b -- c) A word removes the top two stack items as input parameters, b is top of stack (tos). The word places c on the stack as the result
e.g. (n -- ; -- adr) During compilation this word removes n from the stack. During execution the word leaves an address on the stack

The Code Type column is coded as follows - C - Assembly language word, H - High level word, X - High level word defined in 'Extend.fth', EF - High level word defined in 'Easyfile.fth'. (there are other types unknown to the author)

The Word Type column shows whether a word is of type public (PUB), private (PRI), preemptive (PRE) or a module header (MOD).

Some words also have alias names, these are made available so that those with traditional Forth or Spin experience can be familiar with them although some are favoured for clarity and readability especially in various fonts.

The data stack is 4 levels deep in the cog and then implemented as a non-addressable LIFO stack in cog memory. Tachyon 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.

Data is referred to as bytes (8 bits), words (16 bits), longs (32 bits) and doubles (64 bits). The Tachyon stacks are 32 bit wide, so types other than longs are padded or split when placed on the stack

PUBLIC WORDS

This section lists all the public words in Tachyon. These are words that are always available in the dictionary to write programs with. If the programmer is not interested in the Private Words (Module low-level internal words), listed at the back of this document, then he can easily delete that section, or just print the Public Word section.

NAME	*STACK*	*CODE TYPE*	*WORD TYPE*	*DESCRIPTION*

DATA STACK

-ROT	(a b c -- c a b)	H	PUB	Reverse rotate (equiv. to ROT ROT)
!SP	(? --)	H	PUB	Init stack pointer, clear the stack
?DUP	(a -- a a)	C	PUB	dup if a <>0, else (a -- a)
2DROP	(a b --)	C	PUB	Drop top 2 items off the stack (pop)
2DUP	(a b -- a b a b)	H	PUB	Duplicate the top two stack items (equiv. to OVER OVER) (push)
2OVER	(n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)	H	PUB	Copy 3rd and 4th items to tos
2OVER	(a b c d -- a b c d a b)	H	PUB	Duplicate the next two stack items (as if it is a double number) (push)
2SWAP	(n1 n2 n3 n4 -- n3 n4 n1 n2)	H	PUB	Swap items 1,2 with items 3,4
3DROP	(a b c --)	C	PUB	Drop 3 top stack items
3DUP	(n1 n2 n3 -- n1 n2 n3 n1 n2 n3)	X	PUB	Copy first 3 items to tos
3RD	(a b c d -- a b c d b)	C	PUB	Copy third stack item (push)
4TH	(a b c d -- a b c d a)	C	PUB	Copy fourth stack item (push)
BOUNDS	(n1 n2 -- n1+n2 n1)	C	PUB	n1 becomes n1+n2, n2 unchanged
DEPTH	(-- depth)	H	PUB	Return with current depth of data stack (but does not include depth itself) (push)
DROP	(a --)	C	PUB	Drop top item off the stack (pop)
DROP;	(a --)	H	PUB	DROP ; - used by MOD
DUP	(a -- a a)	C	PUB	Duplicate top item on stack (push)
LP!	(a --)	H	PUB	Set loop stack memory - each cog that uses FOR NEXT needs room for 8 longs or more
NIP	(a b -- b)	C	PUB	Drop 2nd stack item (pop)
OVER	(a b -- a b a)	C	PUB	Copy 2nd stack location to first (push)
OVER+	(n1 n2 -- n1 n2+n1)	C	PUB	n2 becomes n1+n2, n1 unchanged
ROT	(a b c -- b c a)	C	PUB	Move 3rd item to 1st, 1st to 2nd, 2nd to 3rd
SWAP	(a b -- b a)	C	PUB	Swap top 2 stack items

RETURN STACK

!RP		C2	PUB	Init return stack pointer
>R	(a --)	C	PUB	Push a from data stack onto return stack
FORK	(--)	X	PUB	Duplicate top of return stack
R@	(-- a)	X	PUB	Copy top of return stack to tos
R>	(-- a)	C	PUB	Pop a from return stack onto data stack

LOGICAL

Some logical operations include a built-in parameter to avoid slow push/pop operations such as 8<< rather than 8 << (0.4us vs 2.4us)

<<	(n cnt -- n2)	C	PUB	Shift n left cnt places
>>	(n1 cnt -- n2)	C	PUB	Shift n right cnt places
>|	(mask -- bit)	X	PUB	Convert mask to bit position of first lsb that is set e.g. 512 >| . --- 9 ok
>b	(n1 -- byte)	C	PUB	Mask n to the l.s. bit
>B	(n -- byte)	C	PUB	Mask off a byte ($FF AND)
>N	(n -- nib)	C	PUB	Mask off a nibble ($0F AND)
|<	(bit -- mask)	X	PUB	Alias for MASK
|<=	(bit -- ; -- mask)	H	PRE	
2*	(a -- b)	C	PUB	Shift left one bit (multiply by two unsigned)
2/	(a -- b)	C	PUB	Shift right one bit (divide by two unsigned)
4*	(a -- b)	C	PUB	Shift left two bits (multiply by 4 unsigned)
8<<	(a -- b)	C	PUB	Fast 8-bit shift left - avoids slow push and pop of literal
8>>	(a -- b)	C	PUB	Fast 8-bit shift right - avoids slow push and pop of literal (i.e. $12345678 -> $00123456)

AND	(a b -- c)	C	PUB	c = a AND b
ANDN	(a b -- c)	C	PUB	c = a AND NOT b ($DEADBEEF $FF ANDN .LONG DEAD.BE00 ok)
BIT!	(addr flg --)	H	PUB	store 1 in long at addr if flg<>0, else store 0
BIT?	(mask addr -- mask flg)	C	PUB	flg = long at addr anded with mask
FALSE	(-- 0)	C	PUB	Constant
INVERT	(a -- b)	H	PUB	Bitwise inversion - all bits are flipped (i.e. $FFFFFFF5 -> $0A)
L>S	(n -- lsb9 h)	C	PUB	Specialized operation for filesystem addresses
MASK	(bit -- mask)	C	PUB	Convert 5-bit number to a mask over 32-bits - mask=0 if bit>31
NOOP	(--)	H	PUB	No operation
NOP	(--)	C	PUB	No operation - (0.4 us) - can be used as a placeholder and overwritten later e.g. pri trap nop nop ;
OFF	(-- 0)	C	PUB	Alias for FALSE
ON	(-- -1)	C	PUB	Alias for TRUE
OR	(a b -- c)	C	PUB	c = a OR b ($123400 $56 OR .LONG 0012.3456 ok)
REV	(n1 bits -- n2)	C	PUB	Reverse LSBs of n1 and zero-extend
ROL	(a cnt -- c)	C	PUB	Rotate a left with b31 rotating into b0 for cnt ($12345678 8 ROL .LONG 3456.7812 ok)
ROR	(a cnt -- c)	C	PUB	Rotate right bit b0 rotating into b31 for cnt ($DEADBEEF 8 ROR .LONG EFDE.ADBE ok)
SAR	(a n -- b)	C	PUB	b = a Shift Arithmetic Right n places
SHL	(a cnt -- c)	C	PUB	Shift left all bits by cnt
SHR	(a cnt -- c)	C	PUB	Shift right all bits by cnt
TOG	(mask caddr --)	H	PUB	toggle bits defined by mask in hub byte at caddr
TRUE	(-- -1)	C	PUB	Constant (although any non-zero number is treated as true as well
XOR	(a b -- c)	C	PUB	c = a XOR b ($123456 $FF XOR .LONG 0012.34A9 ok)

COMPARISON

<	(a b -- flg)	H	PUB	If a is less than b then return with true flag
<=	(a b -- flg)	X	PUB	if a < or = to b then flg = true
<>	(n1 n2 -- flg)	C2	PUB	Return with flag indicating if n1 <> n2 (equiv. to = NOT)
=	(a b -- flg)	C	PUB	Compare a with b
=>	(a b -- flg)	X	PUB	if a > or = to b then flg = true
>	(a b -- flg)	C	PUB	If a > b then flg = true
0<	(val -- flg)	C	PUB	If val is less than zero (negative) then return with true flag
0<>	(n -- flg)	C2	PUB	Return with flag indicating if n <> 0 (equiva. to 0= NOT)
0=	(val -- flg)	C	PUB	Compare n to zero and return with boolean flag
NOT	(val -- flg)	C	PUB	Alias for 0=
U<	(a b -- flg)	C	PUB	If a is unsigned less than b then return with true flag
U>	(a b -- flg)	X	PUB	If a is unsigned greater than b then return with true flag
WITHIN	(val min max -- flg)	H	PUB	Return with flag if val is within min and max (inclusive, not ANSI)

MEMORY

--	(adr --)	H	PUB	Decrement the long in hub memory
!	(long adr --)	C	PUB	Store the long in hub memory (2.2us) (pops)
@	(adr -- long)	C	PUB	Fetch a long from hub memory (0.4us)
@1	(-- adr)	X	PUB	1 long
@2	(-- adr)	X	PUB	1 long
@3	(-- adr)	X	PUB	1 long
+!	(long adr --)	C	PUB	Add long to long in hub memory
++	(adr --)	H	PUB	Increment the long in hub memory
<CMOVE	(src dst cnt --)	H	PUB	Reverse MOVE bytes starting from end of src to end of dst by cnt bytes
~	(adr --)	H	PUB	Set the long in hub memory to zeros
~~	(adr --)	H	PUB	Set the long in hub memory to all ones
1!	(n --)	X	PUB	store n in @1
1@	(-- n)	X	PUB	return value n stored at @1
1++	(--)	X	PUB	increment value stored at @1
2!	(n --)	X	PUB	store n in 2@
2@	(-- n)	X	PUB	return value n stored at @2
2++	(--)	X	PUB	increment value stored at @2
3!	(n --)	X	PUB	store n in 3@
3@	(-- n)	X	PUB	return value n stored at @3
3++	(--)	X	PUB	increment value stored in @3
ALIGN	(adr align -- adr1)	X	PUB	Align address upwards to match alignment boundary (i.e. $474A $40 ALIGN .WORD 4780 ok)
BIG!	(long addr --)	X	PUB	store long big-endian style
BIG@	(addr -- long)	X	PUB	read long big-endian style
C--	(adr --)	H	PUB	Decrement the byte in hub memory
C!	(byte adr --)	C	PUB	store byte to hub memory
C@	(adr -- byte)	C	PUB	Fetch a byte from hub memory
C@++	(adr -- adr+1 char)	C	PUB	Fetch a byte from hub memory and maintain and increment the address
C+!	(byte adr --)	C	PUB	add byte to hub memory
C++	(adr --)	H	PUB	Increment the byte in hub memory
C~	(adr --)	H	PUB	Clear the byte in hub memory to zeros
C~~	(adr --)	H	PUB	Set the byte in hub memory to all ones
CELL+	(addr -- addr+4)	H	PUB	Advance address by one cell - A cell is defined as 4 bytes in Tachyon
CLR	(mask addr --)	H	PUB	Clear the bits in the byte at addr
CMOVE	(src dst cnt --)	C2	PUB	CMOVE bytes from src to dst by cnt bytes (13.11ms for 32k)
D!	(n1 n2 addr --)	H	PUB	Store double n1 n2 at addr
D@	(addr -- n1 n2)	H	PUB	Fetch double n1, n2 from addr
ERASE	(adr cnt --)	H	PUB	ERASE memory (to 0) from adr for cnt bytes
FILL	(adr cnt ch --)	H	PUB	FILL memory from adr for cnt bytes with ch
SET	(mask addr --)	H	PUB	Set the bits in the byte at addr
SET?	(mask addr -- flg)	H	PUB	Test the bits in the byte at addr and return with state
U!	(n adr --)	X	PUB	Write unaligned long
U@	(adr -- n)	X	PUB	Read `unaligned long`
ulong	(-- adr)	X	PRI	long used by U!, U@
W--	(adr --)	H	PUB	Decrement the word in hub memory
W!	(word adr --)	C	PUB	Store word to word at adr
W@	(adr -- word)	C	PUB	Fetch a word from hub memory
W+!	(word adr --)	C	PUB	Add word to word at adr
W++	(adr --)	H	PUB	Increment the word in hub memory
W~	(adr --)	H	PUB	Clear the word in hub memory to zeros
W~~	(adr --)	H	PUB	Set the word in hub memory to all ones

MATHS

-	(a b -- c)	C	PUB	c = a - b [6912 5678 - . 1234 ok]
?NEGATE	(a flg -- b)	H	PUB	Negate a if flg is true
*	(s1 s2 -- s3)	C2	PUB	Signed multiply [-1234 5678 * . -7006652 ok]
*/	(um1 um2 udiv -- rslt32)	H	PUB	Multiply um1 and um2 to produce a 64-bit intermediate result divided by udiv for 32-bit result
/	(div1 div2 -- rslt)	X	PUB	Signed divide
+	(a b -- c)	C	PUB	c = a + b [1234 5678 + . 6912 ok]
1-	(a -- a-1)	C	PUB	Decrement a unsigned
1+	(a -- a+1)	C	PUB	Increment a unsigned
2-	(a -- a-2)	C2	PUB	Decrement a by 2 unsigned (double bytecode instruction)
2+	(a -- a+2)	C2	PUB	Increment a by 2 unsigned (actually a double bytecode instruction 1+ 1+)
ABS	(a -- b)	C	PUB	Absolute value of a - if a is negative then negate it to a positive number
ADDABS	(n1 n2 -- n3)	C	PUB	n3 = n1 + abs(n3) e.g. -2 -3 ADDABS . --- 1 ok
AVG	(val var -- avg)	X	PUB	Accumulate the average using 25% of the difference between the current average and val
CMPSUB	(n1 n2 -- n3)	C	PUB	Compare unsigined, substitute n1 if lesser or equal e.g. 2 3 CMPSUB . --- 2 ok
LIMIT	(n min max --)	X	PUB	Return n limited to within range of min and max inclusive
MAX	(n1 n2 -- n3)	C	PUB	Return unsigned maximum of two items
MAXS	(n1 n2 -- n3)	C	PUB	Return signed maximum of two items
MIN	(n1 n2 -- n3)	C	PUB	Return unsigned minimum of two items
MINS	(n1 n2 -- n3)	C	PUB	Return signed minimum of two items
MOD	(a mod -- rem)	X	PUB	Extract the remainder after division
NEGATE	(a -- 0-a)	C	PUB	Negate a - that is subtract a from zero
RND	(-- rnd)	X	PUB	Generate a 32-bit pseudo-random number enhanced with the system counter
seed	(-- adr)	X	PRI	1 long used by RND
SUBABS	(n1 n2 -- n3)	C	PUB	n3 = n1 - abs(n2) e.g. 3 -2 SUBABS . --- 1 ok
U/	(u1 u2 -- u3)	H	PUB	Unsigned divide
U/MOD	(u1 u2 -- rem quot)	H	PUB	Unsigned modulo divide includes remainder [1024 10 U/MOD . SPACE . 102 4 ok]
UM*	(u1 u2 -- u1*u2L u1*u2H)	C	PUB	unsigned 32bit * 32bit multiply --> 64bit double result
UM*/	(um1 um2 udiv -- rsltL rsltH)	X	PUB	Multiply um1 by um2 with 64-bit intermediate divided by udiv for a 64-bit result
UM/DMOD	(dvnL dvdH dvsr -- rem qL qH)	C	PUB	Full 64-bit by 32-bit divide - used by U/ and U/MOD
UM/MOD	(dvndL dvndH dvsr -- rem quot)	C2	PUB	Same as UM/MOD64 but constructed with (bytecodes UM/MOD64 DROP)

FLOATING POINT MATHS

The following words require the F32 ROM be loaded - N.B. there are more functions in the ROM than there are forth words currently to drive them. The words below were written as proof of concept and would probably need expanding for any f.p. application. See the F32 ROM source for more details.

>F	(n1 -- result)	X	PUB	
F-	(n1 n2 -- result)	X	PUB	Subtraction
F*	(n1 n2 -- result)	X	PUB	Multiplication
F/	(n1 n2 -- result)	X	PUB	Division
F+	(n1 n2 -- result)	X	PUB	Addition
F>	(n1 -- result)	X	PUB	
FSIN	(n1 -- result)	X	PUB	Sine
FSQRT	(n1 -- result)	X	PUB	Square root

CONVERSION

>W	(n -- word)	X	PUB	Mask n to a 16-bit word (eq. $FFFF AND)
1M	(-- 1000000)	X	PUB	constant, 1000000
B>L	(a b c d -- dcba)	X	PUB	Merge four bytes into one long ($12 $34 $56 $78 B>L .LONG 7856.3412 ok)
B>W	(bytel byteh -- word)	X	PUB	Merge bytes into a word
KB	(n - n<<10)	X	PUB	kilobytes
L>W	(long -- wordl wordh)	X	PUB	Split a long into words
M	(n -- n*1000000)	X	PUB	million
MB	(n - n<<20)	X	PUB	megabytes
W>B	(word -- bytel byteh)	X	PUB	Split a word into bytes
W>L	(wordl wordh -- long)	X	PUB	Merge words into a long

LOOPING

DO and LOOP use a separate loop stack to hold the parameters and a branch stack to hold the looping address for very fast looping. So loop indices are available outside of the loop as when functions are called from inside the loop. The words associated with DO and FOR are actual instructions which do not need to calculate branch addresses immediately at compile time. These instructions push their current IP onto the branch stack and use this for fast and efficient looping. Take care with unstructured exiting from loops, use LEAVE or UNLOOP.

Each FOR will push four parameters onto the loop stack being:

FOR	The number of times FOR NEXT will loop (not affected by BY)
FROM	The value that the index starts FROM (0 if not set)
BY	The value to increment the FROM index I by (1 if not set)
BRANCH	The address after FOR which is used by NEXT

The loop stack is maintained in hub RAM at $180 for COG 0. Any other Tachyon cogs should allocate 12 to 16 bytes typically
for 3 to 4 levels recommended (rarely reaches 4)

ADO	(from for --)	C	PUB	Start a DO loop with slightly different parameters to DO
BY	(by --)	C	PUB	e.g. 100 FROM 2 BY 10 FOR I . NEXT --- 100 102 104 106 108 110 112 114 116 118 ok
BY!	(newby --)	C	PUB	set the BY value in a running FOR loop to newby
DO	(to from --)	C	PUB	Start a DO loop e.g. 5 0 DO I . LOOP --- 0 1 2 3 4 ok
FOR	(cnt --)	C	PUB	Push cnt onto loop stack and save next IP onto branch stack for NEXT e.g. 5 FOR I . NEXT --- 0 1 2 3 4 ok
FOR!	(n --)	C	PUB	FOR value on loop stack set to tos
FOR@	(-- n)	C	PUB	Present FOR value copied to tos
FROM	(start --)	C	PUB	e.g. 100 FROM 8 FOR I . NEXT -- --- 100 101 102 103 104 105 106 107 ok
I	(-- index)	C	PUB	Push DO index onto data stack
IC!	(byte -- byte)	C	PUB	Store byte in hub ram pointed at by I
IC@	(n -- byte)	C	PUB	Fetch byte from hub ram pointed at by I
J	(-- index3)	H	PUB	Push third level DO index

K	(-- index2)	H	PUB	Push second level DO index
LEAVE	(--)	H	PUB	Set the index to limit-1 so that it will LEAVE the loop when it encounters LOOP
LOOP	(--)	C	PUB	Loop back if the loop count has not finished, else leave the loop
LP!	(loopstk --)	H	PUB	
LP@	(offset -- (addr))	H	PUB	
NEXT	(--)	C	PUB	Decrement loop IX and exit loop if IX = 0 or else branch using saved FOR branch address

CONDITIONAL BRANCH & LOOPING

AGAIN		HI	PRE	Jump back AGAIN to first instruction after matching BEGIN (BEGIN.....AGAIN)
BEGIN		HI	PRE	BEGIN a conditional loop - marks the spot for a BEGIN UNTIL or BEGIN WHILE REPEAT. During compilation this leaves the address of the next instruction merged with $BE.0000
ELSE		HI	PRE	IF flg was not true then execute between ELSE THEN. During compilation this checks and resolves a preceding IF and sets up for a THEN
ENDIF		HI	PRE	Alias for THEN
IF	(flg --)	HI	PRE	IF flg is true (non-zero) then execute between IF THEN or IF ELSE. During compilation this leaves the address of the next instruction IF merged with $1F.0000
REPEAT		HI	PRE	REPEAT the conditional loop by jumping back to after matching BEGIN
THEN		HI	PRE	THEN continue on executing normally (terminates an IF). Check and resolve any IFs or ELSEs and set the forward branch offset
UNTIL	(flg --)	HI	PRE	UNTIL flg is true continue back to matching BEGIN (BEGIN.....UNTIL)
WHILE	(flg --)	HI	PRE	WHILE flg is true continue executing code up to REPEAT (BEGIN.....WHILE.....REPEAT)

Examples:

IF <more words> THEN

IF <more words> THEN

BEGIN <more words>WHILE <more words> REPEAT

BEGIN <more words> UNTIL

BEGIN <more words> AGAIN - useful within other COGS started within a Tachyon program

CALLS AND BRANCHING

?EXIT	(flg --)	H	PUB	Exit if flg is true.
?JUMP		C	PUB	VECTOR JUMP if set
0EXIT	(flg --)	C	PUB	Exit if flg is zero. This saves a IF … THEN ;
CALL	(adr --)	C	PUB	Call the adr - used to execute cfa vectors
EXIT		C	PUB	Exit from a called routine and pop the return stack into the IP
JUMP	(adr --)	C	PUB	Same as CALL but doesn't save the return address

CASE STATEMENTS

CASE statements are constructed in a manner similar to C using SWITCH, CASE, and BREAK.

SWITCH	(val --)	X	PUB	Store the switch value in a task variables so that is can be referenced by a CASE statement.
SWITCH@	(-- val)	X	PUB	Retrieve the switch value - useful if we want to perform more complex comparisons
SWITCHES	(val <val-word pairs> --)	X	PUB	Scan the following val and word pairs for a match or until a non-value is encountered
]		X	PRE	Alias for BREAK
=[		HI	PRE	Alias for CASE
BREAK		HI	PRE	Stop executing this CASE code and return immediately from routine
CASE	(val --)	HI	PRE	Execute the following code up to BREAK if val = SWITCH val i.e. ($0D CASE PRINT" CARRIAGE RETURN" BREAK)
CASES	(from to --)	H	PRE	Use as: from to CASES ..... BREAK

CASE and SWITCHES examples:-

```
pub RunLEDs
        LState C@ SWITCH
                0 CASE
                    10 HIGH 10 LOW 10 FLOAT 1 LState C! BREAK
                1 CASE
                    11 HIGH 11 LOW 11 FLOAT 2 LState C! BREAK
                2 CASE
                    12 HIGH 12 LOW 12 FLOAT 0 LState C! BREAK
                3 10 CASES
                    <more words> BREAK
        ;
```

SWITCH@ and SWITCH= can be used wherever the switch value needs checking for more complex behaviour within a CASE (say)

SWITCHES is followed by a list of <value> <word> pairs in the input stream. If the input = one of the <value>, then the corresponding word executes
e.g. SWITCHES $0D LCDCR $0A LCDLF $09 LCDTAB NOP This function automatically terminates when it encounters a word in the list that is not a 15-bit literal, so use any other function including NOP or ;

VECTORED EXECUTION

+VECTOR		X	PRE	Insert a call over first instruction
REDEFINE	(<target> <new> --)	X	PRE	Alias for REVECTOR
REVECTOR	(<target> <new> --)	X	PRE	Replaces first instruction of target with jump to new

TACHYON START-UP

!INITS	(--)	X	PUB	Initialise the user INIT list to do nothing on Tachyon start
+INIT	(nfa --)	X	PUB	Add the word whose nfa is tos to the user INIT list to execute on Tachyon starting
AUTORUN	(nfa --)	X	PUB	An alias of +INIT
boot	(-- adr)	X	PUB	1 long - maybe unused now
INIT		X	PUB	Executes up to 16 user INIT words at Tachyon start
uauto	(-- adr)	C	PUB	user autostart address if non-zero - called from within terminal

I/O PORTS

!COUNT		X	PUB	
A		X	PUB	Select the desired target CTR A before use
APIN	(pin --)	X	PUB	Set the APIN of the current CTR
B		X	PUB	Select the desired target CTR B before use
BEEP	(--)	X	PUB	Output 2250Hz for 150 ms on the currently selected pin
BEEPS	(cnt --)	X	PUB	Output cnt BEEPs with a 50 ms break between on the currently selected pin
BIP	(--)	X	PUB	Output 3kHz for 80 ms on the currently selected pin
BLINK	(pin --)	X	PUB	Toggle pin on and off at 2Hz - also useful for setting up a pin quickly - then use HZ etc
BPIN	(pin --)	X	PUB	Set the BPIN of the current CTR
CLICK	(--)	X	PUB	Output a click on the currently selected pin
CLKIN	(iomask dat -- iomask dat2)	C	PUB	Shift bit in from pin, clock high, clock low
CLKOUT	(iomask dat -- iomask dat2)	C	PUB	Shift msb bit out to pin, clock high, clock low
CLOCK		C	PUB	.
COUNT@		X	PUB	
CTRMODE	(n --)	X	PUB	Writes to the CTRMODE field of the current CTR channel without disturbing the other bits of the counter
DAC!	(byte oin --)	X	PUB	Write an 8-bit value to the pin as a duty cycle - filter output for a voltage
DETECT	(pol edge fb --)	X	PUB	
DIFF		X	PUB	Change counter mode to differential
DUTY		X	PUB	
EDGE	(n -- n+2)	X	PUB	
F	(mask -- mask)	C	PUB	float (make high impedance) bits on the output - leave stack intact - fast operation
FB	(n -- n+1)	X	PUB	
FLOAT	(pin --)	X	PUB	Float the pin (make it an input)
FREQ@	(pin -- freq)	X	PUB	Measure frequency of pulses at pin, measured over 1/10s
FRQ	(n --)	X	PUB	Set the duty cycle frequency FRQ, either A or B
H	(iomask -- iomask)	C	PUB	Set mask bits on the output high - leave stack intact - fast operation
HIGH	(pin --)	X	PUB	Set pin high as an output
HIGH@	(pin -- clks)	X	PUB	Measure high pulse width in clock cycles at pin
HZ	(n --)	X	PUB	Output a 1:1 m/s tone, n Hz, on the currently selected pin
IN	(pinmask -- flg)	C	PUB	Test pins using mask
INPUTS	(mask --)	C	PUB	Float the pins to inputs.
ISERIAL	(pin --)	X	PUB	Redirect character output to select serial channel using SEROUT
ISEROUT	(data pin --)	X	PUB	
KHZ	(n --)	X	PUB	Output a 1:1 m/s tone, n kHz, on the currently selected pin
L	(mask -- mask)	C	PUB	Set mask bits on the output low - leave stack intact - fast operation
LOW	(pin --)	X	PUB	Clear pin low as an output
LOW@	(pin -- clks)	X	PUB	Measure low pulse width in clock cycles at pin
MHZ	(n --)	X	PUB	Output a 1:1 m/s tone, n MHz, on the currently selected pin
MUTE	(--)	X	PUB	Cancel all activity on the currently selected pin
NCO	(--)	X	PUB	Set counter mode to NCO
NEG	(-- 4)	X	PUB	
OUT	(data pinmask --)	X	PUB	Set pins in pinmask to outputs and write data to them
OUTCLR	(mask --)	C	PUB	Clear the pins to low outputs (also sets DIR bits)
OUTPUTS	(mask --)	C	PUB	Set the pins to outputs (normally redundant)
OUTSET	(mask --)	C	PUB	Set the pins as high outputs (also sets DIR bits)
P	(mask -- mask)	C	PUB	pulse high mask bits on the output - leave stack intact - fast operation
P!	(long --)	X	PUB	Write directly to OUTA
P@	(-- long)	X	PUB	Read directly from OUTA
PIN!	(state pin --)	X	PUB	Set pin to state (i.e. ON 6 PIN!)
PIN@	(pin -- flg)	C	PUB	Test state of pin
PINS@	(pin pins -- n)	X	PUB	Read from pin for pins and right justify result
PLL		X	PUB	
PLLDIV	(n --)	X	PUB	
POS	(-- 0)	X	PUB	
RING	(--)	X	PUB	WARBLE twice on the currently selected pin
RINGS	(rings --)	X	PUB	Ring 'rings' times on the currently selected pin
SERBAUD	(baud --)	X	PUB	Calc bit ticks and set as well as start bit compensation
SERIAL	(pin --)	X	PUB	redirect character output to select serial channel using SEROUT
SERIN	(pin -- data)	X	PUB	Receive 8 bit serial data from pin at rate set with SERBAUD, blocks until character received
SEROUT	(data pin --)	X	PUB	
SHRINP	(iomask dat -- iomask dat/2)	C	PUB	Shift in right into msb of dat using iomask to specify the pin.
SHROUT	(mask dat -- iomask dat/2)	C	PUB	Shift out right the lsb of dat over the pins in iomask and return with the shifted data
SIREN	(--)	X	PUB	Use WARBLE to make weewaa for 400 mS on currently selected pin
SPKR	(pin --)	X	PUB	Set the pin for audio output
T	(mask -- mask)	C	PUB	toggle bits on the output - leave stack intact - fast operation
TONE	(tone dur --)	X	PUB	Output tone Hz for dur milliseconds on the currently selected pin
VOLTS	(mV pin --)	X	PUB	Use duty cycle mode plus RC on a pin to generate a voltage from 0 to 3.3V (mV set 0-3300)
WAITHI	(--)	C	PUB	Wait until the currently selected pin goes high
WAITLO	(--)	C	PUB	Wait until the currently selected pin goes low
WARBLE	(hz1 hz2 ms --)	X	PUB	Flip between hz1 and hz2 frequency tone for ms milliseconds on the currently selected pin

SPI INSTRUCTIONS

These are fast optimized bytecode instructions for reading and writing an SPI bus whose parameters are held in COGREGS - use SPIPINS to set. Most parameters can be reused as in multibyte shifts plus this makes the transfer faster as pushing and popping the data stack slows things down.

@CE		X	PUB	Returns @SCK + 3
@CNT		X	PUB	Returns @SCK + 4
@MISO		X	PUB	Retuns @SCK + 2
@MOSI		X	PUB	Returns @SCK + 1
@SCK		X	PUB	Returns 1 default
@SCL		X	PUB	Constant, default 15
@SPISCK		X	PUB	Constant, default 10
SPICE	(--)	C	PUB	Release the SPI CE line (automatically enabled on any SPI operation)
SPIPINS	(&ce.miso.mosi.clk --)	X	PUB	Set pins to be used by SPI - parameter is encoded as byte fields - use & prefix to force decimal bytes
SPIRD	(long -- long1)	C	PUB	Read SPI data and left rotate into long with long1 as result ($12345678 -> $345678NN)
SPIWR	(long -- long1)	C	PUB	Send 8 MSBs of long over SPI and return with left rotated long1 ($12345678 -> $34567812)
SPIWR16	(long -- long1)	C	PUB	Send msb 16-bits (b31..b16) over SPI and return with long rotated left by 16 bits
SPIWR32	(long - long)	C	PUB	Send 32-bits over SPI, leaves tos unchanged
SPIWRB	(byte -- byte)	C	PUB	Send byte over SPI lines as defined in COGREGs and return with same byte

I/O MASKS

@CE	(-- 3)	X	PUB	
@CNT	(-- 4)	X	PUB	COGREG address for variable CNT used by some RUNMODs
@MISO	(-- 2)	X	PUB	
@MOSI	(-- 1)	X	PUB	
@SCK	(-- 0)	X	PUB	COGREG address for SCK mask
@SCL	(-- -6)	X	PUB	COGREG address for SCL (used for fast CLOCK instruction)
MODPINS	(pins --)	X	PUB	Set the pin masks for RUNMODs using (&27.26.25.23 MODPINS to set ce.miso.mosi.clk)
SETPINS	(pins adr --)	X	PUB	Set pins masks using adr for cog starting address of clk pin

COG INSTRUCTIONS

.TASKS	(--)	X	PUB	Display the status of cog 0,2-7 w.r.t what they are running
boot	(-- adr)	X	PUB	1 long
COG!	(long adr --)	C2	PUB	Store long to cog memory
COG@	(adr -- long)	C2	PUB	Fetch long from cog memory
COGID	(-- id)	C2	PUB	
COGINIT	(code pars cog -- ret)	H	PUB	Same as COGINIT in PASM - also saves information in cog TASK block (8 bytes/cog)
COGSTOP	(n --)	H	PUB	Stop cog n
LOADCOG	(name cog par --)	X	PUB	e.g. " VGA32x15 " 3 vgapars LOADCOG - Load VGA32x15 ROM into cog3 with vgapars parameter block
LOADCOGS	(name cog par step cogs --)	X	PUB	e.g. " HSUART " 3 par1 12 5 LOADCOGS- Load HSUART ROM into cog3, 12 byte pars entries for another 5 cogs
LOADMOD	(src dst cnt --)	C2	PUB	Load cog memory from hub memory - used internally by CODE MODULES
pCOGINIT		H	PUB	Part of COGINIT
REBOOT		H	PUB	Reboot the current cog
RESET		C	PUB	Reset this cog only
RUN	(pfa cog --)	X	PUB	e.g. ' MYTASK TASK? RUN - run MYTASK on the next available cog
RUN:	(pfa cog --)	X	PUB	Run following code as a task in cog n e.g. : pri SENSORS 3 RUN: BEGIN 12 13 DISTANCE mm W! 15 DHT 'c W! rh W! 60 ms AGAIN ;
RUNMOD		C	PUB	Run the currently loaded code module
TASK?	(-- task)	X	PUB	Find the next available cog that's free to run a task - ready and in IDLE
TASKREGS	(addr --)	X	PUB	Set starting address of a task's registers

CODE MODULES

These are small PASM modules that loaded into once the Tachyon cog and executed repeatedly with the separate RUNMOD word.

[SDRD] H SD card block read

RUNMOD	(dst char -- firstPos charcnt)	H	PUB

Read block from SD into memory while scanning for special char
dst is a 32 bit SD-card address 0..4GB, char is the character to scan for while, reading in the block.
NOTE: ensure MOSI is set as an output high from caller by 1 COGREG@ OUTSET
This is just the low-level block read once the SD card has been setup, so it just reads a sector into the dst
There is also a scan character that it will look for and return its first position and how many were found

[SDWR] H SD card block write

RUNMOD	(src cnt --)	H	PUB

Write a block to the SD card - normally 512 bytes

[PWM32] H PWM32 runtime (takes over cog)

RUNMOD	(table waitcnt --)	H	PUB

32 channel 8-bit PWM that runs up to 7.6kHz
Can be used as an arbitrary waveform generator too as it reads a long from table (32 channels) every waitcnt sample and writes to all the outputs that are enabled in the PWM cog. The normal wave table is 256 longs deep. The table must be aligned to a 256 long boundary

[PWM32!] H PWM32 table setup

RUNMOD	(duty8 mask table --)	H	PUB

Write 8-bit duty cycle to channels specified in mask at specified table

[WS2812] H WS2812 RGB LEDs (array cnt --)

RUNMOD	(array cnt --)	C	PUB	
pin mask is in COGREG4, line RET is done at HL, not here				
Will transmit a whole array of bytes each back to back in WS2812 timing format				
A zero is transmitted as 350ns high by 800ns low (+/-150ns)				
A one is transmitted as 700ns high by 600ns low				

[SDRDF] [SDRD] [SDWR] [SDIO] [SSD!] [PLOT] [CAP] [WAV] [MCP32] [RCTIME][LTC2754][SSD!]

[SSD]	H	PUB	TFT display
[ESPIO]	H	PUB	Enhanced Serial Peripheral I/O
[SPIO]	H	PUB	Serial Peripheral I/O
[MCP32]	H	PUB	SPI for MCP3208 style chips etc
[PLOT]	H	PUB	Fast plotting
[CAP]	H	PUB	Fast I/O Capture for SPLAT logic analyser (buf lcnt dly --)

ROMS

ROMS are binary images of assembly language code that are saved to upper EEPROM (or elsewhere) that can be loaded into cogs at runtime by name. Just send the relevant .hex file to Tachyon, like you would with any other .fth file. The new ROM will show up on boot or if 'lsrom' is executed

lsroms	PUB		List the ROMS present in the upper 32k of EEPROM

TIMING and FREQUENCY

~F	(--)	X	PUB	Display the P1 .clock frequency
.FREQ	(--)	X	PUB	Display the phrase 'FREQ = <cpu clock frequency>'
.LAP		X	PUB	Print results of LAP <tests> LAP
CLK		H	PUB	
CLKFREQ	(-- n)	X	PUB	constant, CPU frequency
CLKMHZ	(-- n)	X	PUB	constant, CPU frequency / 1000000
CLKSET		H	PUB	
LAP	(--)	C2	PUB	Latch the CNT value and before saving calculate the difference from previous LAP and save
LAP@	(-- n)	H	PUB	Used to zero LAP timing ?
ms	(n --)	X	PUB	Pause execution for n milliseconds
runtime	(-- addr)	X	PUB	1 long variable
s	(n --)	X	PUB	Pause execution for n seconds
time	(-- addr	X	PUB	4 bytes variable
TIMERJOB	(cfa --)	X	PUB	
timers	(--- addr)	X	PUB	1 cword variable
us	(n --)	X	PUB	Pause execution for n microseconds
us	(n --)	X	PUB	Delay for n microseconds (+10us overhead but values are compensated so 20 us = 20us)
WAITCNT		C2	PUB	Wait until CNT reaches the DELTA value - callo repeatedly after first setting DELTA
WAITX	(delta --)	C	PUB	Wait for x cycles and set WAITCNT delta

DEFINITIONS

pub		HI	PRE	Alias of :
;		HI	PRE	Compile an EXIT instruction before finishing off a definition
:	(<name> --)	HI	PRE	Create a Forth definition and compile all words into it until a ; is encountered
[C]	(<word> --)	H	PRE	Forces the compilation of a preemptive word
ALIAS	(<oldword> <newword> --)	H	PRE	Create an alias for an existing word
module	(--)	H	PRE	e.g. module EXTEND ." My Forth Module " ; The string is displayed under MODULES at Tachyon start
pre	(<name> --)	HI	PRE	Create a preemptive Forth header for a word which must execute at compile time, not be compiled
pri	(<name> --)	HI	PRE	Create a private Forth header exactly the same as : except set the private attribute in the header. If RECLAIM is executed later on it will find all headers with the private attribute and strip them out.

COMMENTS

---	XI	PRE	Prefered Tachyon comment as it separates sufficiently and does not look like any other operator
-->	HI	PRE	Result comment
"	HI	PRE	Similar to Spin comment operator
(	HI	PRE	Comment up to the matching) and echo - (what follows is a stack comment) (n1 n2 -- n3)
{	HI	PRE	Ignore all text up to the matching }. Used for multiline comments. Nesting to 255 levels
}	HI	PRE	Outside of a block comment this symbol will simply be ignored
\	HI	PRE	Comment the rest of the line and do not echo - \ this is a comment

CONDITIONAL COMPILE

IFDEF	(<name > --)	HI	PRE	IF <name> DEFINED then process all source between here and the matching }
IFNDEF	(<name> --)	HI	PRE	IF <name> NOT DEFINED then process all source between here and the matching }

COMPILE LITERALS

Bytes, words, and longs may be compiled directly into code memory usually for building fixed tables.
These cannot be used inside a definition as any preceding literal would have already been compiled as a literal.

,	(long --)	HI	PRE	Compile a long as used in building tables i.e. $1234.5678 , $DEAD.BEEF ,
|	(byte --)	HI	PRE	Alias for C, - less clutter when building tables i.e. 34 | 45 | 98 | 20 | etc
||	(word --)	HI	PRE	Compile the preceding word into code memory i.e. $1234 || $0FCA || $0082 ||
C,	(byte --)	HI	PUB	Compile the preceding byte into code memory
L,	(long --)	HI	PUB	Compile the preceding long into code memory
NFA,	(<word> --)	H	PRE	Compile the name field address into code memory
W,	(word --)	HI	PUB	Compile the preceding word into code memory

RADIX WORDS

Numbers entered and printed can be represented in any base (radix) by setting the cog variable "base" to that value.
The three most common bases are predefined.
NOTE: It is recommended that numbers other than decimal are always forced with a prefix or alternatively a suffix such as $0FAD or 0FADh etc.

HEX	H	PUB	Switch number base to HEX mode - all input and output will default to HEX unless overridden
DECIMAL	H	PUB	
BINARY	H	PUB	

RADIX OPERATORS

While not defined in the dictionary radix operators force a number to be recognized in a certain base. The operators may be prefixed or suffixed while the prefixed operators have the advantage that the compiler will compile these immediately as a number rather than search the dictionary first as it would with any other number. Tachyon convention is that all hex number are prefixed with $ with the number base set to decimal by default. All numbers 0 to 9 do not need a prefix as that is redundant and besides some of these single digits are predefined as fast constants making them a single bytecode.

$	HEX prefix - number may contain symbols but must end in a valid digit i.e. $Q00FA
#	Decimal prefix - " " i.e. #P26
%	Binary prefix
H	i.e. 00FAh - number must begin with a decimal digit or zero.
D	i.e. 1234d
B	i.e. 01101110b
^<ch>	Return with the control character literal for the next character i.e. ^Z returns $1A
'<ch>'	Return with the ASCII literal for the enclosed character i.e. 'Z' returns $5A

DEBUG

??	(--)	X	PUB	Called as part of REBOOT. Displays Modules loaded; Clock frequency; User initialisation words; ROMS; I2C devices sensed; Pi0-P31 loads sensed; Memory free; Data stack; Date and Time
.MODULES	(--)	X	PUB	Displays list of Tachyon forth modules installed
.S		H	PUB	Print the contents of the data stack
.STATS	(--)	H	PUB	Display code, name, data and free space sizes
.VARS	(--)	X	PUB	Display all variables and constants, with present values, in the dictionary
BOOT	(--)	X	PUB	Starts the Tachyon system
DEBUG		H	PUB	Will dump stacks, registers, and current compilation area. Can also be accessed by a single keystroke ^D (control D)
lsi2c	(--)	X	PUB	Displays list of i2c devices sensed - used by ?? word
lsini	(--)	X	PUB	Displays list of user initialisation words - used by ?? word
lsio	(--)	X	PUB	Displays list of loads sensed on P0 - P31 - used by ?? word
lsroms	(--)	X	PUB	Displays list of ROMS loaded - used by ?? word
REBOOT	(--)	H	PUB	Restarts the Tachyon system
TRAP	(<wordtotrap> <debugword> --)	X	PUB	Has the effect of inserting debugword at the start of wordtotrap e.g. to print stuff each time wordtotrap is called; wordtotrap will only revert to normal after Tachyon is rebooted
HELP	(<word> --)	X	PRE	Displays decompilation of the word if present in dictionary. Use CR to terminate

DEBUG CONTROL KEYS

To speed up interactive testing there are certain control keys that can perform operations.

^?	Executes .STATS , displaying code, name, data, free and data stack space
^B	Block dump all of hub memory (wait for it)
^C	Reboot Tachyon
^D	Dump stacks, registers, and current compilation area
^Q	Display two lines of memory data starting at address on tos. Consumes tos
^S	Reset data stack
^V	Display all variables and constants, with values, in memory
^W	Execute WORDS, display all the words in the dictionary
^X	Repeat previous line (re-executes the previous interactive code)
^Z^Z	Cold start - wipe all extensions bar the kernel although everything is still intact in EEPROM
<BREAK>	Will reboot the processor regardless of what code it is running (system in fact detects 100 "framing errors" in a row)
<ESC>	Discard the current interactive line
BKSP	Backspace up to the beginning of a word or else the line (preceding words are already compiled)
LF	Ignore (as part of a CRLF)
TAB	Tab as normal but handled like a single space

DUMP MEMORY OPERATIONS

Various words are available for general-purpose dumping of memory in hex format. Normally the memory that the DUMP words examine is hub memory but a modifier may be used before a DUMP is executed to use other types of memory. After any dump the default if set back to hub RAM.
Some modifiers are:
EE = EEPROM
COG = COG MEMORY (LONG)
SD = SD card raw
FS = File System (from the start of an open file)
WIZ = WIZnet chip

DUMP	(adr bytes --)	H	PUB	Dump as bytes from current dump device including an ASCII code column (revert back to RAM after)
DUMPW	(adr bytes --)	H	PUB	Dump as words (same as DUMP)
DUMPL	(adr bytes --)	H	PUB	Dump as longs (same as DUMP, formatted as 0000.0000)
DUMPA	(adr bytes --)	H	PUB	Dump ASCII printable characters - default width of 64 characters/line (uses . for non-printable)
DUMPAW	(adr bytes --)	H	PUB	Dump ASCII wide
DUMPC		H	PUB	Dump COG longs
QD	(adr --)	H	PUB	Quick Dump two lines of standard DUMP data
RAM		H	PUB	Change DUMP device to standard HUB RAM (DUMP always defaults here after every DUMP)

EE		H	PUB	Change DUMP device to EEPROM (>64k addresses next device etc)
DUMP:		H	PUB	For defining new DUMP type words e.g. pub SD DUMP: <more words> ;
DUMPX	(from cnt spaces:bytes:width+format 'method --)	H	PUB	

STREAMING I/O

Character based devices such as serial, VGA, LCD etc are treated as streaming I/O where the device code automatically detects and handles special characters. EMIT words will send a single character via the currently selected output device. Conversely KEY is the input from the device.

.VER		H	PUB	Print verbose Tachyon version number i.e. .VER Propeller .:.:--TACHYON--:.:. Forth V27150908.1000
(EMIT)	(ch --)	C2	PUB	The default emit code if uemit is zero
(EMITX)		C2	PUB	Part of (EMIT)
(KEY)	(-- ch)	H	PUB	Read the console input, this is the default execution vector when ukey = 0 (see task registers)
[CON		X	PUB	Switch to console but save current output device - use to print console messages without changes
<CR>		X	PUB	Emit a single CR (no LF)
CON	(--)	X	PUB	Reset character I/O (EMIT & KEY) back to default console
CON]		X	PUB	Restore previous output device before the [CON word was executed
CR	(--)	H	PUB	Emit a CR+LF sequence
CTYPE	(str cnt --)	X	PUB	Print the string for cnt characters, normally in Forth this is simple TYPE but that is used in FTP
doKEY			PUB	
DOT	(--)	H	PUB	emit one full stop character
ECHO		H	PUB	ECHO OFF - streaming input not echoed, ECHO ON - input is echoed
EMIT	(ch --)	H	PUB	emit the character via the vector at uemit
EMIT:	(--)	X	PUB	Used to create char output redirection words e.g pub NULLOUT EMIT: DROP ;
EMITS	(char cnt --)	X	PUB	Print the char repeatedly for cnt
ESC?	(-- flg)	X	PUB	Return true if the last console key pressed was an escape? (even if it's still buffered)
GRAB		H	PUB	Force execution of all preceding words on a streaming input line
KEY	(-- ch)	H	PUB	Read a character from the device, a null indicates that no character was available
KEY:	(--)	X	PUB	Used to create char input redirection words - like EMIT:
KEY!	(char --)	X	PUB	Force a character to be read as the next KEY
KEY$		H	PUB	
NULLOUT	(--)	X	PUB	Throw away all char output - do not display anywhere
QUIET	(on/off --)	X	PUB	Non-interactive mode - just accept "commands" - ON QUIET
SHORTCUT	(vec key --)	X	PUB	Assign a control key shortcut
SPACE	(--)	H	PUB	Emit a space
SPACES	(n --)	X	PUB	Emit n spaces
TAB	(--)	X	PUB	Emit a single TAB
TABS	(n --)	H	PUB	Emit n TAB characters
WKEY	(-- ch)	H	PUB	Always WAIT for a KEY so that even a null is a character
XTAB	(pos --)	H	PUB	

CONSTANTS and VARIABLES

:=!	(newcon 'oldcon --)	X	PUB	change a pseudo constant value e.g. : myconst 3 ; 5 ' myconst :=!. Does not work with := type constants
@org	(-- ptr)	H	PUB	Pointer for org
DS+	(bytes --)	X	PUB	Allocate bytes at org
org	(adr --)	X	PUB	Set the data origin for DS style data memory allocation
org@	(n --)	X	PUB	Save n as word at @org
vars			PUB	
VER	(-- adr)	H	PUB	Address of longs that holds current kernel version build i.e. VER @ .DEC 27150908 = V2.7 150908
:=	(val <name> --)	XI	PRE	Create a constant (preferred format reduces clutter around values and names)
byte	(<csv> --)	XI	PRE	Create byte variables from the CSV list (or just a single variable) (BYTE xy,myvar,net)
bytes	(n <name> --)	H	PRE	Create a block n bytes
CARRAY	(cnt <name> -- ; index -- addr)	X	PRE	create an array of bytes in code memory that can be indexed (saved in EEPROM on BACKUP)
cbytes	(val cnt <name> --)	X	PRE	allocate and fill bytes in code memory
clong	(<name> --)	X	PRE	create a long in code memory (saved in EEPROM on BACKUP)
cword	(<name> --)	X	PRE	create a word in code memory (saved in EEPROM on BACKUP)
DS	(bytes <name> --)	XI	PRE	Create a constant with the current value of the ORG then advance it by bytes for next DS
long	(<csv> --)	XI	PRE	Create long variables (long aligned)
longs	(cnt <name> --)	XI	PRE	Create a long array (long aligned)
res		XI	PRE	Alias for DS
TABLE	(<name> --)	XI	PRE	Create a table with zero entries (use , | || words to add entries)
word	(<csv> --)	XI	PRE	Create word variables (word aligned)
words	(n <name> --)	H	PRE	Create a block of n words (word aligned)
words	(cnt <name> --)	X		Indirectly called by WORDS which performs another action if there is no name

STRINGS

Strings are represented in Tachyon as an address to a null (or 8th bit) terminated string.
Strings may be defined in Tachyon as STRING mystringname Hello World! <cr>

$=	(str1 str2 -- flg)	X	PUB	Compare two strings for equality
$!	(str1 str2 --)		PUB	
."	(<str>” --)	HI	PRE	Print the literal string. Example: PRINT" HELLO WORLD" -- actually the compile-time part of it
"	(<str>" -- str)	HI	PRE	Process the following characters up to " as a string and leave the address on the stack. Outside of a definition the string buffer will be reused and not available after the line is processed.
(.")	(--)	HI	PUB	this is the helper which does the runtime printing
+CHAR	(ch str2 --)	X	PUB	Add a character to a string
APPEND$	(str1 str2 --)	X	PUB	Append str1 to the end of str2
LEFT$	(str len -- str)	X	PUB	Destructive LEFT$ - uses same string
LEN$	(str -- len)	H	PUB	Return with the length of the null terminated string
LOCATE$	(ch str -- str)	X	PUB	variableLocate the first ch in the string and return else null
MID$	(str offset len -- str)	X	PUB	Extract the substring of str starting at offset len chars long
NULL$	(-- str)	X	PUB	Just an empty string
PRINT"	(<str>" --)	HI	PRE	Alias for ."
PRINT$	(str --)	H	PUB	Print out the null terminated string at str onto the currently selected output device (via uemit)
RIGHT$	(str len -- str)	X	PUB	give a copy of the rightmost len chars of str
STRING	(<wordname> <string> --)		PRE	Define anew string called wordname, initialised to string
STRING	(str max --)	XI		Immediate word to build a string with a maximum size (use 0 to fit current length)

PRINT NUMBERS

.	(n --)	H	PUB	Print number unformatted in the current base
.AS	(num format --)	X	PUB	Display num with format defined in format string
.B	(n --)	X	PUB	Display n as two hex digits
.BIN16	(n --)	X	PUB	Display n as a 16 bit binary number
.BIN32	(n --)	X	PUB	Display n as a 32 bit binary number with an underscore in the middle and a % prefix
.BYTE	(n --)	X	PUB	Print the byte in n as two hex characters
.DEC2	(n --)	X	PUB	Display n as a decimal number in the range 00 - 99
.DEC2.	(n --)	X	PUB	Display n as a decimal number in the range 00 - 99 with a decimal point suffix
.DEC4	(n --)	X	PUB	Display n as a four digit decimal number
.DECL	(n --)	X	PUB	Display n with commas at the thousands, millions etc.
.DP	(dblnum decimals ---)	X	PUB	Print the double number with decimal places (scaled)
.HEX	(n --)	X	PUB	Print the nibble in n as a single hex character
.INDEX		X	PUB	Print the current DO index on a new line as (0000:)
.L	(n --)	X	PUB	Display n as 8 digits of hex with . in the centre
.LONG	(n --)	X	PUB	Print the number in hexadecimal as a long (i.e. 0 @ .LONG 05B8.D800 ok)
.NFA	(nfa --)	H	PUB	Display the corresponding word name
.W	(n --)	X	PUB	Display n as 4 digits in hex
.W:	(n --)	X	PUB	Display n as 4 digits of hex with : suffix
.WORD	(n --)	X	PUB	Print the word in n as four hex characters
.WORD$	(n)	H	PUB	Print the word in n as four hex characters with $ prefix
@.	(adr --)	X	PUB	Fetch long and print value in current base
@.	(addr --)	X	PUB	Display long value stored at addr
@PAD	(-- adr)	X	PUB	Pointer to current position in number pad
#	(a -- b)	X	PUB	Extract another digit from the a leaving b and prepend the digit to the number string buffer
#>	(a -- str)	X	PUB	Stop converting the number and discard what's left and return with a ptr to the string
#S	(a n --)	X	PUB	Extract n digits using # word
<#		X	PUB	Start converting a number to a string by resetting the number buffer
<D>	(--)	X	PUB	Signal that the current number to be printed should be processed as a double number
>CHAR	(val -- ch)	X	PUB	Convert a binary value to a character that represents that digit (0-9,A-Z,a-z)
ASCBIN	(char -- val flg)	H	PUB	If char is 0-9,A-F, converts to 4 bit binary and flg=true, else flg=false
D.	(n1 n2 --)	X	PUB	Display n1 / n2 as a double number (64 bit)
HOLD	(ch --)	X	PUB	Prepend the character to the number string buffer
NUM>STR	(num -- str)	X	PUB	Convert a number to a string and buffer it in NUM$ where it can be manipulated etc
PRINT	(n --)	H	PUB	Alias for .
PRINT&	(n --)	X	PUB	Display n in IP address format - &aa.bb.cc.dd
STR>NUM	(str -- val digits | false)	X	PUB	Convert a string to a number if possible
U.	(u1 --)	X	PUB	Print unsigned number
.AS"	(n --)	X	PRE	Define a number display word with a defined format e.g. pub .DEC4 .AS" ###`#" ;

Number Print Formatting --- .AS" and friends are very versatile ...

```
..  123456 .AS" $~###,###,##~#.##"    $1,234.56 ok
..  1234 .AS" 8|~"   00001234 ok
..  1234 .AS" 8|`"       1234 ok
~          pad leading zeros with spaces
`          skip over leading zeros
```

TASK VARIABLES

Each cog may have its own set of variables that are offset from the address in COGREG 7. This is so that any cog running Tachyon may have different I/O devices selected etc.Only a small number of these variables are named in the dictionary but they can be referenced from these with an offset by referring to the source.

REG	(index -- adr)	C	PUB	Find the address of the register for this cog
reg	(<name> --)	X	PRE	Create user register variable (normally up to 256 byte addresses)
rx	(-- adr)	H	PUB	Pointer to the rx buffer with the 2 words before the buffer as rxrd and rxwr index
flags	(-- adr)	H	PUB	Bit flags used by the kernel

- ❑ echo = 1 ' managed by pub ECHO \ ON ECHO \ OFF ECHO for console echo
- ❑ linenums = 2 ' prepend line number to each new line
- ❑ ipmode = 4 ' interpret this number in IP format where a "." separates bytes
- ❑ leadspaces = 4
- ❑ prset = 8 ' private headers set as default
- ❑ striplf = $10 ' strip linefeeds from output if set (not used - LEMIT replaces this !!!)
- ❑ sign = $20
- ❑ comp = $40 ' force compilation of the current word - resets each time
- ❑ defining = $80 ' set flag so we know we are inside a definition now

base	(-- adr)	H	Byte variable specifying the current base + backup byte used during overrides
digits	(-- adr)	H	Byte variable with count of digits from last number parsed
delim	(-- adr)	H	Word delimiter (normally space) plus backup byte with delimiter detected (SP,TAB,CR etc)
word	(-- adr)	H	Pointer to word buffer where a parsed word is stored
switch	(-- adr)	H	SWITCH value is stored as a long here (single level only)
autorun	(-- adr)	H	Pointer to cfa of user autostart routine normally used by EXTEND which implements a new user vect
keypoll	(-- adr)	H	User app may set this to the cfa of a routine that gets polled while KEY is idling.
tasks	(-- adr)	H	Holds task list of 8 bytes for each 8 cogs (IP[2],RUN[1] implemented in EXTEND
unum	(-- adr)	H	User number processor vector. 0 defaults to kernel method.
uemit	(-- adr)	H	Vector points to cfa of current EMIT routine (0=console=(EMIT))
ukey	(-- adr)	H	Vector points to cfa of current KEY routine (0=console=(KEY))
names	(-- adr)	H	Points to the start of the latest name field in the dictionary (builds down)
here	(-- adr)	H	Points to the end of the code space but normally referenced by HERE (here W@)
codes	(-- adr)	H	Temporary code space pointer while a line is compiled but not yet committed (interactive)
baudcnt	(-- adr)	H	EXTEND has SERIN and SEROUT routines which store their baudrate CNT value here for each cog
prompt	(-- adr)	H	User vector may point to code to change the prompt (normally blank)
ufind	(-- adr)	H	
create	(-- adr)	H	
Lines	(-- adr)	H	holds line count during block load
errors	(-- adr)	H	holds count of errors detected during block load of source via TACHYON word

lastkey (-- adr) H The last key that was pressed from the serial console is stored here, useful for lookaheads.

Special Purpose Registers

```
SPR     ( -- adr )      PUB  constant = $01F0, Special Purpose Register table
PAR     ( -- adr )      PUB  constant = $01F0, Boot Parameter
CNT     ( -- adr )      PUB  constant = $01F1, System Counter
INA     ( -- adr )      PUB  constant = $01F2, Input States for P31-P0
INB     ( -- adr )      PUB  constant = $01F3, Input States for P63-P32 (not used)
OUTA    ( -- adr )      PUB  constant = $01F4, Output States for P31-P0
OUTB    ( -- adr )      PUB  constant = $01F5 Output States for P63-P32 (not used)
DIRA    ( -- adr )      PUB  constant = $01F6 Direction States for P31-P0
DIRB    ( -- adr )      PUB  constant = $01F7, Direction States for P63-P32 (not used)
CTRA    ( -- adr )      PUB  constant = $01F8, Counter A Control
CTRB    ( -- adr )      PUB  constant = $01F9, Counter B Control
FRQA    ( -- adr )      PUB  constant = $01FA, Counter A Frequency
FRQB    ( -- adr )      PUB  constant = $01FB, Counter B Frequency
PHSA    ( -- adr )      PUB  constant = $01FC, Counter A Phase
PHSB    ( -- adr )      PUB  constant = $01FD, Counter B Phase
VCFG    ( -- adr )      PUB  constant = $01FE, Video Configuration
VSCL    ( -- adr )      PUB  constant = $01FF, Video Scale
```

Registers by index

```
' Minimum registers required for a new task - other registers after the ' ---- are not needed other than by the console
0           temp                res 12          ' general purpose
12          cntr                res 4           ' hold CNT or temp
' @16
16          uemit               res 2           ' emit vector – 0 = default
18          ukey                res 2           ' key vector
20          keypoll             res 2           ' poll user routines - low priority background task
22          base                res 2           ' current number base + backup location during overrides
24          baudcnt             res 4           ' SERIN SEROUT baud cnt value where baud = clkfreq/baudcnt – each cog can have it's own
28          uswitch             res 4           ' target parameter used in CASE structures
32          flags               res 2           ' echo,linenums,ipmode,leadspaces,prset,striplf,sign,comp,defining
34          keycol              res 1           ' maintains column position of key input
35          wordcnt             res 1           ' length of current word (which is still null terminated)
36          wordbuf             res wordsz      ' words from the input stream are assembled here
75          numpad              res numpadsz    ' Number print format routines assemble digit characters here – builds from end -
18,446,744,073,709,551,615
                                                ' numpad may continue to build backwards into wordbuf for special cases such as long binary numbers
101         padwr               res 1           ' write index (builds characters down from lsb to msb in MODULO style)

'
' ------ console only registers – not required for other tasks - so no need to allocate memory beyond here
'


102         unum                res 2           ' User number processing routine - executed if number failed and UNUM <> 0
104         anumber             res 4           ' Assembled number from input
108         bnumber             res 4
112         digits              res 1           ' number of digits in current number that has just been processed
113         dpl                 res 1           ' Position of the decimal point if encountered (else zero)

' WORD aligned registers

114         ufind               res 2           ' runs extended dictionary search if set after failing precompiled dictionary search
116         createvec           res 2           ' If set will execute user create routines rather than the kernel's

118         rxptr               res 2           ' Pointer to the terminal receive buffer - read & write index precedes
120         rxsz                res 2           ' normally set to 256 bytes but increased during block load
122         corenames           res 2           ' points to core kernel names for optimizing search sequence
124         oldnames            res 2           ' backup of names used at start of TACHYON load
126         names               res 2           ' start of dictionary (builds down)
128         prevname            res 2           ' temp location used by CREATE
130         fromhere            res 2           ' Used by TACHYON word to backup current “here” to determine code size at end of load
132         here                res 2           ' pointer to compilation area (overwrites VM image)
134         codes               res 2           ' current code compilation pointer (updates "here" or is reset by it)
136         cold                res 2           ' pattern to detect if this is a cold or warm start ($A55A )
138         autovec             res 2           ' user autostart address if non-zero - called from within terminal
140         errors              res 2
142         linenum             res 2

' Unaligned registers

144         delim               res 2           ' the delimiter used in text input and a save location

146         prompt              res 2           ' pointer to code to execute when Forth prompts for a new line
148         accept              res 2           ' pointer to code to execute when Forth accepts a line to interpret (0=ok)


150         prevch              res 2           ' used to detect LF only sequences vs CRLF to perform auto CR
152         lastkey             res 1           ' written to directly from serialrx
153         keychar             res 1           ' override for key character

154         spincnt             res 1           ' Used by spinner to rotate busy symbol

155         prefix              res 1           ' NUMBER input prefix
156         suffix              res 1           ' NUMBER input suffix
157                             res 3

160         tasks               res tasksz*8    ' (must be long aligned)
```

224 endreg res 0

DICTIONARY

The Tachyon names dictionary is a completely separate area from the code dictionary and is where all the names, name and code attributes, and code "pointer" is stored. Unlike traditional Forths the names are not stored inline with the code and this allows for more flexibility with memory and dictionary. For instance names may be removed without touching or impacting code and also aliases may be added that are simply clones of the original header so they do not add any overhead. Also names may be declared as private allowing them to be removed later on and the memory reclaimed. Because code no longer has inline headers and is always bytecode it also means that code can "fall through" into the next code forward definition thus simplifying code.

Here is an example of a code header which has the fields: count(1), name(V), ATR(1), pointer(2)

```
: HELLO PRINT" HELLO WORLD"  CR ;  ok
@NAMES 10 DUMP
0000_559D:   05 48 45 4C  4C 4F 82 BD   53
```

Dictionary ATR fields always have the msb set thus terminating the name if it is referenced as a string since both nulls and >$7F characters are valid string terminators in Tachyon. The pointer is more commonly not a pointer as such but the bytecodes to be compiled, either one or two bytes. The name count is used both for quick linking into the next word (nextadr = adr+cnt+4) and also to speed up searching as string compare is skipped if counts do not match. Name dictionaries build down from a high address toward the code dictionary which builds up (since it needs to execute in this manner) from a low address towards the name dictionary. So free space is the difference between the latest entry in the names dictionary @NAMES and the current HERE which points to the next free code location.

.CFA	(cfa --)		PUB	Displays word name and code field address
(FORGET)	(nfa --)		PUB	Forget the word with this nfa
[']		H	PUB	
[W,]	(wordcode --)		PUB	
@NAMES	(-- adr)	X	PUB	Return with point to start of latest dictionary header (builds down)
+NFA			PUB	
ALIGN			PUB	
ALIGNORG			PUB	
ALLOCATED		H	PUB	
ALLOT	(n --)	H	PUB	Allot n bytes of code memory - advances "here"
CFA	(-- adr)		PUB	
CONSOLE		H	PUB	
CPA			PUB	
CREATE	(<name> --)	H	PUB	create new dev with dummy cfa (save ptr to it
CREATE:			PUB	
CREATE$	(wordcode --)	X	PUB	Create a name in the dictionary from wordcnt+wordbuf
DEFAULTS	(--)		PUB	Reset dictionary pointers and stop COGS 3-7
DEFER	(<word> -- str)	H	PUB	Wait for a terminated word to be entered and return with the ptr to the wordbuf
DISCARD		H	PUB	Discard the current input line
DOES>	(--)		PUB	set new cfa to point back to DOES: code (skipped by DOES: itself)
END	(--)		PUB	Used to signal the end of a TACHYON source code section
GETWORD	(<word> -- str)	H	PUB	Wait for a terminated word to be entered and return with the ptr to the wordbuf (Deprecated use DEFER)
HERE	(-- adr)	H	PUB	
IDLE		H	PUB	Start-up used by idle cogs which checks for a run address while pausing to save power
names			PUB	
NFA			PUB	
NFA			PUB	
NFA$	(str -- nfaptr)	H	PUB	
PRIVATE	(--)		PUB	All : , var and const definitions that follow may be removed from the dictionary so that they can no longer be used in new definitions. 'pub' and 'pri' words override this
PUBLIC	(--)		PUB	All : , variable and constant definitions that follow can be used in all following definitions
QW	(--)	X	PUB	List the latest 128 words in quick compact format
SEARCH	(cstr -- nfaptr)	H	PUB	Search the dictionaries for cstr which points to the word string constructed as count+string+null
TACHYON	(--)		PUB	used to verify that source code is intended for Tachyon and also to reset load stats - terminate with END
TASK	(cog -- addr)	H	PUB	Return with address of task control register in "tasks"
uhere	(-- adr)		PUB	1 word, pointer to to compilation area
undef	(-- adr)	C	PUB	1 word, user word cfa can be stored here which will be called when a word in the input stream is not found in the dictionary. Reset 0 to do normal Tachyon behaviour e.g. ' FRUN undef W! causes the SD card to be searched for a file name and if found will load it to the input stream (Implements MSDOS 'batch file' behaviour) 0 undef W! disables the function
uthere	(-- adr)	C	PUB	1 word, current code compilation pointer (updates "here" or is reset by it)
V5			PUB	
W	(<char> --)	X	PUB	List all words with detail - like WWORDS <cr>
WORDS		XI	PUB	NOTE: special case performs one of two functions
WORDS	(--)	X	PUB	List words in dictionary
WORDS	(n <name> --)		PUB	Allocate n words of variable memory accessed by name
'	(<word> -- cfa)	H	PRE	Return with the code address of the following word. If cfa<$100 = bytecode address in cog memory.
[B]	(bytecode --)		PRE	append this bytecode to next free code location + append EXIT (without counting)
[W]	(wordcode --)		PRE	append this wordcode to next free code location + append EXIT (without counting)
FORGET	(<word> --)		PRE	Forget and remove code for 'word' and all words defined after it in the dictionary
NFA'	(<word> -- nfa)	H	PRE	Return with the name field address of the following word
RECLAIM	(<word> --)	X	PRE	Scan the dictionary for any private words and removed their headers and reclaim dictionary memory by compacting. Will RECLAIM from 'word' if that is present, else RECLAIMS all dictionary
STRIP	(<word> --)		PRE	Strip a single header from the dictionary - 'word' then cannot be used in future definitions
WWORDS	(<char> --)	X	PRE	List words in dictionary in wide 4 column format with detail and optional filtering *
EXTEND	(--)		MOD	Module start marker for high level extension words to the core Tachyon in file EXTEND.fth

* WWORDS has some useful features for inspecting the dictionary:-

The listing can be filtered or not: e.g. WWORD G lists only words starting with G. WWORD <cr> lists all words
Each entry is <Name Field Address> <Code Field Address> <Key Field> <Word Name>

1. Immediate words are shown in **BOLD**
2. Public words are coloured green
3. Private words are coloured red
4. Public variables and constants are coloured cyan
5.Private variables and constants are coloured yellow
6. Module Header words like EASYFILE and EXTEND which are defined with the 'module' word **are coloured bold red**

If RECLAIM is run to remove all private word entries from the dictionary, WWORDS <cr> will then list only the public words - all the plain red and yellow words disappear, as you'd expect.

The Key field
Bit 7 is an immediate or preemptive attribute
Bit 6 is a private word
Bit 7&6 together represents this entry as a module header.
Bits 5 - 3 reserved
Valid values for bits 2 - 0 are:

0 public definition
1 reserved
2 private (can be removed by RECLAIM)
3 reserved
4 preemptive immediate word executes - normally to read in more words from the stream
5 reserved
6 module header
7 reserved

BUFFERS

BUFFERS	(-- adr)	H	PUB	2k bytes available for up to 4 open files or general use

REAL TIME CLOCK

Many I2C RTC chips are very similar in layout and at present there are two types, MCP79410 series and the DS3231 which is temperature compensated and includes temperature readings.. But these are mostly compatible with many other types. As the registers are in the same place it's just that sometimes some chips use the unused bits for various things.

!RTC	(--)	X	PUB	Read the RTC chip and set Tachyon date and time from it (if the chip is present)
.ASMONTH	(month --)	X	PUB	Print month index 1-12 as 3 characters (Jan..Dec)
.DATE		X	PUB	Print the current date as YY/MM/DD
.DAY		X	PUB	Print day as 3 characters (Mon..Sun)
.DT	(--)	X	PUB	Display day of week, date and time
.TIME		X	PUB	Print the current time as HH:MM:SS
BCD>DEC	(bcd -- dec)	X	PUB	BCD to decimal conversion
date	(-- adr)	X	PUB	4 byte buffer
DATE!	(date --)	X	PUB	Store date as international format decimal yymmdd
DATE@	(-- date)	X	PUB	Fetch data as yymmdd
DAY	(day --)	X	PUB	set DAY as in MON DAY etc
DAY@	(-- day)	X	PUB	Fetch day of week
DEC>BCD	(dec -- bcd)	X	PUB	Convert decimal 0..99 to BCD
DT!	(hh.mm.ss|yy.mm.dd d/t --)	X	PUB	Write time/date in decimal format where d/t = 4 for date
FRI	(-- 5)	X	PUB	Day constant
HMS	(#xxyyzz -- zz yy xx)	X	PUB	Split 6 digit decimal number into 3 two digit groups
MON	(-- 1)	X	PUB	Day constant
MS>TIME	(n --)	X	PUB	Convert N milliseconds to hhmmss
RTC		X	PUB	Select RTC as a DUMP device - byte method only
RTC!	(byte adr --)	X	PUB	Write a byte into the RTC register
RTC@	(adr -- byte)	X	PUB	Read a byte from the RTC register
runtime	(-- adr)	X	PUB	1 long
SAT	(-- 6)	X	PUB	Day constant
SDT!	(hh.mm.ss|yy.mm.dd d/t --)	X	PUB	Write soft time/date in decimal format where d/t = 4 for date
SETRTC	($opt.addr --)	X	PUB	Select and set the RTC device
STAMP@	(-- dhmsc)	X	PUB	Returns a 32-bit millisecond time + day of month in top 5-bits
SUN	(-- 7)	X	PUB	Day constant - use like this - SUN DAY!
THU	(-- 4)	X	PUB	Day constant
time	(-- adr)	X	PUB	4 byte buffered
TIME!	(time --)	X	PUB	Store time as decimal hhmmss
TIME@	(-- time)	X	PUB	Fetch time as decimal hhmmss
TUE	(-- 2)	X	PUB	Day constant
TZ	(str --)	X	PUB	Save time zone to EEPROM?
tz$		X	PUB	1 long, default to 0
WED	(-- 3)	X	PUB	Day constant

TIMERS

Tachyon maintains a background timer cog which counts every millisecond and scans a linked list of user counters that may be setup to simply countup or countdown to zero and optionally execute an ALARM condition. Also system runtime is maintained so this can be quite useful plus soft RTC functions are available too.

.TIMERS	(--)	X	PUB	Display the list of active timers
ALARM	(cfa timer --)	X	PUB	Set the alarm condition to be executed when this timer has timed out
ALARM:	(val tmr : code --)	X	PUB	use to setup timer code easily: e.g 50 mytimer ALARM: <inline alarm code follows> ;
COUNTUP	(timer --)	X	PUB	Set this timer as a simple up counter every ms
TIMEOUT	(ms addr --)	X	PUB	Set the timeout period in ms for this timer - link and init if not already set
TIMEOUT?	(timer -- flg)	X	PUB	Check if this timer (using TIMER name) has timed out. (also links this timer into the list)
TIMERJOB	(cfa --)	X	PUB	Wait until _job=0, then set _job to cfa
timers	(-- adr)	X	PUB	1 cword, link to timers - set to 1 as last
WATCHDOG	(ms --)	X	PUB	(Re)Trigger watchdog and timeout in milliseconds to reboot
TIMER		X	PRE	create a TIMER variable, ms(4), alarm/mode(2), link(2), tid(1), nu(1), nu(2)
TIMER	(<name> --)	X	PRE	Create a new timer structure (10 bytes)

SYSTEM COUNTER

=CNT	(addr --)		PUB	store the system counter value at addr
CNT?	(cycles var -- flg)		PUB	
CNT@	(-- cntr)		PUB	Fetch the current contents of the system counter

POLLING

Somewhat related to timers but completely different is the background polling which goes on when a cog is waiting for KEY input. In the main console task this allows a user routine to add POLLS that check for low priority tasks that are more able to be handled by the main console cog such as detecting for SD card inserted etc. The user may add up to 8 polls and this needs to be done in the user init routine as all polls are cleared on boot.

?POLL	(--)		PUB	read up to 8 words and call them if set
+POLL	(cfa --)	X	PUB	Add the code routine as a background POLL

I2C BUS

_I2C@			PUB	
?I2C			PUB	Wait but timeout while busy
*SCL			PUB	Pseudo constant, defaults to 28
*SDA			PUB	Pseudo constant, defaults to 29
<I2C			PUB	Generate an I2C START condition
<I2C>			PUB	Restart without checking busy
ackI2C@	(-- byte)	X	PUB	Same as I2C@ except the ack state is set active already (= 0 I2C@)
EEPROM		X	PUB	Setup P29 and P28 as the I2C bus pins
I2C!	(byte --)	X	PUB	Write a byte to the I2C bus
I2C!?	(byte -- flg)	X	PUB	Write a byte to the I2C bus and return with the ack state
I2C@	(ack -- byte)	X	PUB	Fetch a byte from the I2C bus and write the ack state (as is so that 0 = ack)
I2C>			PUB	Generate an I2C STOP condition
I2C100	(--)		PUB	Set the I2C bus to 100kHz rate
I2C400	(--)		PUB	Set the I2C bus to 400kHz rate
I2CFAST	(--)		PUB	Set the I2C bus to 1MHz rate
i2cflg			PUB	1 byte
I2CPINS	(sda scl --)	X	PUB	Setup the pins that will be used for the I2C bus
IO!	(data device --)		PUB	PCF8574 style I2C write
IO@	(device -- data)		PUB	PCF8574 style I2C read
nakI2C@			PUB	

PING-PONG NETWORKING

ID!	(str --)		PUB	save unit prompt ID (use NULL$ ID! to clear)
INTERCOM!	(&GP.ID.TR.TE baud --)		PUB	configure this Propeller as a Ping-Pong slave GP-group, ID-moduleID, TR -transmit/receive pin, TE-enable pin
.INTERCOM	(--)	X	PUB	Display the list of connected 'PING-PONG network' devices, if this system is installed

EEPROM

?BACKUP		X	PUB	Only backup if there were no errors in the TACHYON block load
@EE	(adr -- ack)	X	PUB	Select the appropriate device and issue an address, check ack
@EEWAIT	(adr --)		PUB	Revision 140602 - Added timeout loop counter to prevent hanging
AUTORUN <name>		XI	PUB	Set system to autorun name if found else clear autorun if invalid. Executed via EXTEND.boot If the name does not exist in the dictionary at boot-time it will no longer be valid (as in FORGET <name)
BACKUP		X	PUB	Backup all of 32k of hub RAM to $0000 of first EEPROM
COLD		X	PUB	Reset to a kernel only system in RAM without extensions although EEPROM is not affected
CONBAUD	(baud --)	X	PUB	Set the startup baudrate of the console into EEPROM, needs restart to activate. Recommend 300 to 2,000,000
E!	(long -- adr)	X	PUB	Write a long to EEPROM (non-aligned)
E@	(adr -- long)	X	PUB	Read a long from EEPROM (non-aligned)
EC!	(byte adr --)	X	PUB	Write a byte to the EEPROM at adr (spans multiple 64k devices)
EC@	(adr -- byte)	X	PUB	Read a byte from EEPROM at adr (spans multiple 64k devices)
ECOPY	(eesrc eedst cnt --)	X	PUB	Copy cnt bytes from eesrc to eedst
EE	(--)	X	PUB	Select EEPROM for memory DUMP using various DUMP methods (i.e. 0 $100 EE DUMP)
eedev	(-- adr)		PUB	1 byte
eeflg	(-- adr)		PUB	1 byte
EEPROM	(--)		PUB	Assign pins P28 and P29 for EEPROM interface
EERD	(--)	X	PUB	Switch EEPROM to read mode, check ack
EFILL	(src cnt ch --)	X	PUB	Fill EEPROM from src address for cnt times with byte ch
ELOAD	(eeprom ram cnt --)	X	PUB	Load a block of EEPROM to RAM. Will load 32K from EEPROM in 4.325sec
ep	(-- 128)		PUB	Constant
ESAVE	(ram eeprom cnt --)	X	PUB	Save a block of RAM to EEPROM using page write. Will backup 32K to EEPROM in 4.963 seconds)
ESAVEB	(ram eeprom cnt --)	X	PUB	Save a block of RAM to EEPROM using byte by byte method, slower and safer for non-page alignments
EW!	(word -- adr)	X	PUB	Write a word to EEPROM (non-aligned)
EW@	(adr -- word)	X	PUB	Read a word from EEPROM (non-aligned)
SAVEROM	(--)		PUB	Always called at the start of a ROM file to load the hex bytes of code and data that follow into cog memory

ANSI TERMINAL SUPPORT

?ANSI	(--)	X	PUB	Detects whether terminal supports ANSI commands and stores that at _ansi
.HEAD$	(str --)	X	PUB	Displays string at address str in bold with *** either side
BELL	(--)	X	PUB	emit one bell character
black	(-- 0)	X	PUB	
blue	(-- 4)	X	PUB	
BOLD		X	PUB	Enable ANSI bold type (if supported)
CLS		X	PUB	Hybrid $0C EMIT plus ANSI HOME + ERASE SCREEN
CLS	(--)	X	PUB	Clear the screen (ANSI)
CURSOR	(on/off --)	X	PUB	Set visible cursor on or off
cyan	(-- 6)	X	PUB	
ERLINE	(--)	X	PUB	Erase the current line
ERSCN	(--)	X	PUB	Erase the screen from the current location
ESC	(ch --)	X	PUB	
green	(-- 2)	X	PUB	
HOME	(--)	X	PUB	Cursor is set to the top left corner of the window
magenta	(-- 5)	X	PUB	
MARGINS	(top bottom --)	X	PUB	Sets the number of rows at top and bottom of the terminal window to be set as margins
NEON	(--)	X	PUB	Set pen color to a sequence per character
PAPER	(color --)	X	PUB	Set paper color (background 0..7) e.g. black PAPER
PEN	(color --)	X	PUB	Set pen color (foreground 0..7) e.g. white PEN
PLAIN		X	PUB	Reset all type to plain
red	(-- 1)	X	PUB	
REVERSE		X	PUB	Enable ANSI reverse type
SPINNER	(--)	X	PUB	Emit the next character in a spinner sequence (| / - \ ) using backspace to reposition
white	(-- 7)	X	PUB	Constant for ANSI pen or paper color (white PEN)
WRAP	(on/off --)	X	PUB	Set whether overflow at the end of line wraps round to the next line
XY	(x y --)	X	PUB	ANSI XY cursor positioning (1 1 XY = home)
yellow	(-- 3)	X	PUB	
_ansi		X	PRI	1 long, stores terminal ANSI support
.PAR	(--)	X	PRI	
AEMIT	(ch --)	X	PRI	Only emit ch if ANSI is supported
ANSI?	(-- flg)	X	PRI	Checks the value of _ansi and sets flag true if ANSI supported (faster than calling ?ANSI everytime)

EASYFILE FAT32

The FAT32 file layer is built on top of basic buffered sector layer and the virtual memory layer on top of that. Since the virtual memory is limited to 4GB using a 32-bit address a further step is taken to allow a file to be addressed as virtual memory of up to 4GB. There are sector and directory buffers for up to four files which are opened in the sense that the virtual memory address to the start of the file is located. File sectors are assumed to be contiguous without fragmentation and this is normally the case as I have never found a fragmented SD card before. Not having to follow clusters simplifies and speeds the virtual memory layer.

N.B. The SD card interface pins may need setting up with SDCARD, to suit your circuit board SanDisk cards are known to work, but other makes are not guaranteed

_sdpins	(-- adr)	EF	PUB	1 long in code memory, SD pins setting which is saved every time BACKUP is run
-FERASE		EF	PUB	Erase the current file by overwriting with nulls
!SD		EF	PUB	Initialise the SD card (with timeout)
?MOUNT	(--)	EF	PUB	If not already, mount sdcard
.FILE	(--)	EF	PUB	Display the status of the file stream FILE#
.FILES		EF	PUB	Display the present status of the four four file streams
.FNAME	(--)	EF	PUB	Print the file name at the current loop index I
.FX	(index --)	EF	PUB	Display the status of the file stream 'index'
.LIST	(--)	EF	PUB	Display verbose listing of files
(cat)		EF	PUB	Display the currently open file else ignore
(SLIST)	('method --)	EF	PUB	
@BOOT	(-- bootsect)	EF	PUB	
@FAT	(fat# -- sector)	EF	PUB	return with the starting address of the selected FAT (normally 0 or 1)
@FILE	(addr -- addr+off)	EF	PUB	returns the address offset into sector & sectcrc tables for active file channel as set with index FILE e.g. val sector @FILE ! otherval sectcrc @FILE ! sector @FILE @
@ROOT	(-- rootsect)	EF	PUB	start sector of the root directory
#files	(-- byte)	EF	PUB	4, the number of file channels as standard
>FILE		EF	PUB	Redirect character output via uemit to the open file using "fptr" which is set to the start of the file when opened. If the file is not opened and a valid write pointer set then output will be discarded
ACMD	(data acmd -- res)	EF	PUB	Send an ACMD to the card and return with response
APPEND	(eof -- fsptr)	EF	PUB	Find the EOF marker (normally a null) and set the write pointer and result to this ready to append return with null if failed.
BLKSIZ	(-- word)	EF	PUB	512, the block size
CARD?	(-- flg)	EF	PUB	Detect SD card presence - the CS line must not have a pullup on it (redundant and undesirable). Action is - pulse low, float, check, return high
cd$	(dirstr --)	EF	PUB	change directory to that defined by dirstr
cid	(-- adr)	EF	PUB	16 bytes, Card I.D.
CLUST>SECT	(clust# -- sector)	EF	PUB	convert a cluster number to a physical start sector (normally 64 sectors/cluster)
CMD	(data cmd -- res)	EF	PUB	Send the command to the SD card and read result
csd	(-- adr)	EF	PUB	16 bytes, Card Specific Data
DIR	(--)	EF	PUB	Display a listing of the current directory
DIR?	(str -- diradr | false)	EF	PUB	Find the name in the current directory and return with (virtual memory address) XXXX the dir buffer address
dirbuf		EF	PUB	An array of 32 byte buffers one per file (4 as standard)
dirfsa		EF	PUB	
DIRW	(--)	EF	PUB	Display shortform list of files
FCLOSE	(--)	EF	PUB	Close the current file
FCOPY$	(src$ dst$ --)	EF	PUB	Copy file src$ to file dst$
FCREATE$	(size namestr -- flg)	EF	PUB	Create a new file by name but if it already exists then delete the old one and reuse the dir entry
FGET	(-- ch)	EF	PUB	Read in the next character in the virtual buffer as part of the console input stream
FILE	(index --)	EF	PUB	index in range 0-3, set the active file channel
FILE#	(-- file#)	EF	PUB	Return the current file channel
FILE>		EF	PUB	Set current file as an input device (instead of from console etc)
FILE$	(-- addrofFilenameString)	EF	PUB	
FLUSH	(--)	EF	PUB	Write the sector buffer if it has been modified
FMAKE$	(name$ -- flg)	EF	PUB	Open or create a file

FMAX@		EF	PUB	
FOPEN$	(namestr -- sector)	EF	PUB	Open the file with the 8.3 name and return with its sector or 0 if failed. The variable fstat can be checked for more information if there was an error
FPRINT$	(file$ --)	EF	PUB	Display the contents of the file specified by the string if it exists
FPUT	(ch --)	EF	PUB	Write a character into the logical end of the file and update the write pointer
FPUTB	(byte --)	EF	PUB	Write a character into the logical end of the file and update the write pointer, even if it is a null char
fread	(-- readptr)	EF	PUB	
FREM	(-- rem)	EF	PUB	returns the number of remaining bytes in a file
FRUN		EF	PUB	exception handler - if word not found then run from file - point unum to this code
FS		EF	PUB	FS DUMP method - dump contents of open file
FS!	(long faddr --)	EF	PUB	
FS@		EF	PUB	
FSADR	(faddr -- addr)	EF	PUB	
FSC!	(byte faddr --)	EF	PUB	
FSC@	(faddr -- byte)	EF	PUB	
FSECT@	(-- sect)	EF	PUB	
FSIZE		EF	PUB	
FSIZE!		EF	PUB	
FSIZE@		EF	PUB	
FSTAMP	(--)	EF	PUB	Update the modified time and date of the current file
FSW@		EF	PUB	
fwrite	(-- writeptr)	EF	PUB	
I+	(n -- I+n)	EF	PUB	Adds n to the loop index I
lss	(sector --)	EF	PUB	Display dump 200 hex bytes from 'sector'
MAKE	(size --)	EF	PUB	Force file open, create to size of it not found
MOUNT	(--)	EF	PUB	Mount the currently selected storage device and init all 4 file handles. Read the FAT32, set variables accordingly
ocr	(-- adr)	EF	PUB	4 bytes, Operating conditions register
OpenDir		EF	PUB	Make the current working directory accessible as a file itself
pwd	(--)	EF	PUB	Display current directory name
RENAME$	(from$ to$ --)	EF	PUB	Rename the file using two string parameters
RO	(--)	EF	PUB	Make current file read only
ROOT		EF	PUB	
RW	(--)	EF	PUB	Mark the currently open file read write
SCAN!	(ch/flg --)	EF	PUB	set scan char or disable with -1
SD	(--)	EF	PUB	Dump memory modifier e.g. 0 $200 SD DUMP
sdbuf		EF	PUB	Initial value for SDBUF which depends upon which file handle is selected
SDBUSY	(state --)	EF	PUB	Do what has to be done if the SD card is busy - does nothing as a default
SDERR	(state --)	EF	PUB	Do what has to be done if the SD card errors - does nothing as a default
SDIO32		EF	PUB	Made SDIO RUNMOD handle 32-bit with normal entry and 8-bit with entry+2 including ROL
SDIO8		EF	PUB	Made SDIO RUNMOD handle 32-bit with normal entry and 8-bit with entry+2 including ROL
SDPINS	(ce-miso-mosi-clk --)	EF	PUB	Remember which pins are to be used for BACKUP
SDRD	(sector dst -- buffer)	EF	PUB	Read sector from SD into dst
SDWR	(src sect -- flg)	EF	PUB	Write from src to xdst in the SD
SECTOR	(sect -- buffer)	EF	PUB	read sector into buffer, if not already there
UpdateDir		EF	PUB	Update current directory entry from buffer
VOLNAME!	(str --)	EF	PUB	change the FAT VOL name
WRSECT	(--)	EF	PUB	Write the current sector back to the storage media
X!	(long xaddr --)	EF	PUB	Write a long to virtual memory address xaddr
X@	(xaddr -- long)	EF	PUB	Read a long from memory address xaddr
XADR	(xaddr -- addr)	EF	PUB	Translate absolute 4GB SD memory address to a buffered address in hub RAM
XC!	(byte xaddr --)	EF	PUB	Write a byte to virtual memory address xaddr
XC@	(xaddr -- byte)	EF	PUB	Read a byte from virtual memory address xaddr
XW@	(xaddr -- word)	EF	PUB	Read a word from virtual memory address xaddr
FOPEN#	(dirfsa -- sector)	EF	PUB	Open the file pointed to by the virtual directory entry address fsa = file system address (offset into directory file)
cat	(<file> --)	EF	PRE	Command line "cat" command to list the contents of a file
cd		EF	PRE	deferred word - cd$
FCOPY	(<from> <to> --)	EF	PRE	Make a copy of a file with a new name
FL		EF	PRE	Load text input into specified file or TEMP.TXT and then process
FLOAD	(<file> --)	EF	PRE	Command line File load - loads source file which is executed and/or compiled
FOPEN	(<filename> --)	EF	PRE	Open a file interactively with the name specified in the input stream, report back to the console
ls		EF	PRE	List the directory in wide and simple format (alias for DIRW?)
mk	(size <name> --)	EF	PRE	Make file
QV	(<filename> --)	EF	PRE	Quick view of file header in ASCII dump format
RENAME	(<from> <to> --)	EF	PRE	Command line file rename
SAVETEXT	(<filename> --)	EF	PRE	File write text input to file
EASYFILE		EF	MOD	Module name in the dictionary for Easyfile.fth

BUILT-IN APPLICATIONS AND DEMOS

TOOLS	(--)	X	MOD	Module marker for words defined in EXTEND.fth

Fibonacci series

fibos	(--)	X	PUB	Benchmark - Compute a series of fibonacci numbers and display them with execution times
fibo	(n -- f)	X	PUB	Subroutine used by fibos

Servos

DEGREES	(degrees pin --) marker	X	PUB	
RUNSERVOS	(cog --)	X	PUB	
SERVO!	(word pin)	X	PUB	Initialise the servo system
servos		X	PRI	variable, 64 bytes

Pulse Width Modulation

PWM	(duty8 mask --)	X	PUB	
PWM!	(duty8 mask --)	X	PUB	Initialise the PWM system

PWM.START	(mask table Hz --)	X	PUB	Main multichannel PWM start method in next available cog
PWM%	(%duty pin --)	X	PUB	set the PWM duty cycle as a percentage
PWM.TASK		X	PRI	Setup pins as outputs and load, setup and run PWM32 module
pwmfreq		X	PRI	1 word
pwmpins		X	PRI	1 long
pwmtbl		X	PRI	1 word

WS2812 RGB LED

ALED		X	PUB	
RGB	(rrggbb --)	X	PUB	
RGBS	(array leds --)	X	PUB	Output an rgb array for the number of leds (3 bytes/led) on the selected pin
RGNPIN	(pin# --)	X	PUB	
ansi		X	PRI	Table of 8 longs
brg		X	PRI	1 long

INFRARED CONTROL

IRRX	(pin -- code)	X	PUB	NEC IR Receive
IRTX	(code pin --)	X	PUB	NEC IR Transmit

DHT22 humidity and Temperature

DHT	(pin -- rhum temp)	X	PUB	Read values
.DHT	(pin --)	X	PUB	Read and display values e.g. 15 .DHT displays 34.6'C 42.5%RH
DHTBYTE	(-- byte)	X	PRI	used internally in DHT
DHTBIT	(-- cnt)	X	PRI	used internally in DHT
dt		X	PRI	Timer used in DHT
htsav	(-- adr)	X	PRI	1 long
htref	(-- adr)	X	PRI	1 word
htck	(-- adr)	X	PRI	1 word

ULTRASOUND DISTANCE MEASUREMENT

DISTANCE	(trig echo -- distance.mm)	X	PUB	PING sensor and return with reading in millimetres
PING	(trig echo -- us)	X	PUB	Trigger PING sensor and listen on echo pin and return with microseconds value

DS3231 THERMOMETER

.TEMP	(--)	X	PUB	Read and Display the temperature from a DS3231 chip in 'C
'C	(-- 'C*100)	X	PUB	Read the temp and return value in 'C x 100
'F	(-- 'F*100)	X	PUB	Read the temp and return value in 'F x 100

UNSORTED WORDS

(")			PUB	
@WORD			PUB	
id			PUB	
keytable			PUB	
num			PUB	
rxpars			PUB	
TX!	(dat buf --)	C	PUB	write single buffer - but this word isn't used in kernel, EXTEND or EASYFILE, so obsolete?
WAITVID			PUB	
["]		H	PRE	
...				

PRIVATE WORDS

This section lists all the private words in Tachyon. These are low-level support words useful only within a Module. They are not that useful to Tachyon programmers, although they are left visible in the dictionary. They can be removed from the dictionary, so they are no longer available for new definitions (but still continue to function) by the use of the RECLAIM word. This makes a further 1900 bytes available to the user.

NAME	*STACK*	*CODE TYPE*	*WORD TYPE*	*DESCRIPTION*

FLOATING POINT MATHS - Private words

f32cmd	(-- adr)	X	PRI	1 long (F32 parameter block header)
FCMD	(n1 n2 cmd -- result)	X	PRI	
fnumA	(-- adr)	X	PRI	1 long
fnumB	(-- adr)	X	PRI	1 long
result	(-- adr)	X	PRI	1 long

CASE STATEMENTS - Private words

>SWITCH<	(min max --)	X	PRI	Part of CASES
SWITCH=	(val -- flg)	X	PRI	Return true if val = SWITCH

VECTORED EXECUTION - Private words

~v		X	PRI	Part of +VECTOR

I/O PORTS - Private words

_ctr		X	PRI	1 byte, storing the selected CTR, A or B
*spkr	(-- adr)	X	PRI	1 long, defaults to 0, used to store pin for audio output
+CTR		X	PRI	
~1		X	PRI	Part of CTR!
~S		X	PRI	Part of ISERIAL
baudcnt		X	PRI	user variable
CTR	(-- addr)	X	PRI	
CTR!		X	PRI	clear the whole counter - except pin numbers
CTR@	(-- val)	X	PRI	Read this cogs CTR
TXSER		X	PRI	
WAIT!	(n --)	X	PRI	Selects the pin n to be measured for pulse width etc. - used in LOW@ and friends

ROMS - Private words

FINDROM	(name -- addr)		PRI	

TIMING and FREQUENCY - Private words

+TIMER	(addr --)	X	PRI	
CountDown		X	PRI	
HZCON	(-- n)	X	PRI	constant, used to convert various register values to the equivalent in Hz. Cpu clock frequency dependent

STREAMING I/O - Private words

~c	(-- addr)	X	PRI	1 long, used by [CON, CON] etc

PRINT NUMBERS - Private words

.AST		X	PRI	Part of .AS"
.SIZE	(n --)	H	PRI	9999 or 26k or 9.9M or 3.4G
#NUM			PRI	
~P		X	PRI	Part of PRINT&
AS#		X	PRI	Part of .AS - usable command characters
D0=		X	PRI	
pbase	(-- adr)	X	PRI	1 byte variable
pflg	(-- adr)	X	PRI	1 byte variable

DICTIONARY - Private Words

!DEFER			PRI	Part of DEFER
(STRIP)	(nfa -- nfa+)		PRI	Strip name by moving newer words over
+UNDEF			PRI	Part of DEFER
lsword	(nfa -- nfa)		PRI	Display numeric params for a word in the dictionary, used in WWORDS
undef$	(-- adr)		PRI	16 words
UNDEFER			PRI	Part of DEFER

REAL TIME CLOCK - Private words

.ASMONTH			PRI	
.DATEF	(n --)	X	PRI	Print n as date as YY/MM/DD
.DTS	(time date day --)	X	PRI	Display the day of week, date and time
.TIMEF	(n --)	X	PRI	Display n as time as HH:MM:SS
@rtc	(n --)	X	PRI	Address nth byte of rtc buffer
@rtc	(-- id)	X	PRI	I2C RTC address 8-bit constant

<RTC		X	PRI	Start and address RTC device
<RTC>		X	PRI	Restart and read RTC device
~S		X	PRI	Part of WRRTC
day	(-- adr)	X	PRI	2 byte buffer
RDRTC		X	PRI	Read first 8 timekeeping bytes of RTC into rtc
rtc	(-- adr)	X	PRI	10 byte buffer
rtc!	(byte reg --)	X	PRI	
rtc@	(reg -- byte)	X	PRI	
rtc#	(-- 0)	X	PRI	1 15-bit pseudo constant (value can be changed with :=!)
sec#	(-- 0)	X	PRI	1 15-bit pseudo constant (value can be changed with :=!)
WRRTC		X	PRI	Write rtcbuf to RTC

TIMERS - Private Words

_job		X	PRI	1 word
+TIMER	(addr --)	X	PRI	Part of TIMEOUT
CountDown		X	PRI	
CountIt		X	PRI	
tid	(-- adr)	X	PRI	1 byte, timer ID signature (match to this indicates timers linked)
TIMERS	(--)	X	PRI	Provide background timing functions including alarm actions on timeouts
ttint	(-- n)	X	PRI	timing constant, CLKFREQ #1000 /
wdt		X	PRI	Watchdog timer

I2C BUS - Private words

~D	(long --)		PRI	Store long to cog memory 16, used to store the current I2C bus speed

EEPROM - Private words

?E@			PRI	
?EC@			PRI	
?EE	(addr --)		PRI	
?EW@			PRI	
.ES			PRI	Print counted string from EEPROM
.rom			PRI	
@EEX			PRI	Part of @EE
EERDW			PRI	Wait and switch EEPROM to read mode, check ack
EESPEED			PRI	Set the I2C Bus to 'fast'
ENDRD			PRI	Read last byte [no ack] and stop
FINDROM	(name -- nameaddr | false)		PRI	Find the named ROM in the upper 32kb of the EEPROM, else return false to show not found
GET8	(-- byte)		PRI	Get the next byte from the ROM file, stored there as a two digit hex value. Used in SAVEROM to load a ROM into a cog
NEXTROM	(addr1 -- addr2 | false)		PRI	Find the next ROM, else return false if no more to be found
R$			PRI	String variable, initialised to ROMS
RDI2C	(cnt --)		PRI	Fast sequential read bytes from selected I2C device into indexed memory
roms			PRI	Constant, $C000, start address for ROM storage in EEPROM
romsz			PRI	Constant, $3F00

ANSI TERMINAL SUPPORT - Private words

asw		X	PRI	
ATR	(ch --)	X	PRI	
COL	(col fg/bg --)	X	PRI	
CUR	(cmd n --)	X	PRI	
ESCB	(ch --)	X	PRI	
ncol		X	PRI	1 byte, stores current color for NEON etc
WRAP	(size -- ; val --)	X	PRE	define a wrap constant := val & size-1

EASYFILE FAT32 - Private words

_!SD	(-- ocr | false)	EF	PRI	Initialise the SD card (with timeout) - internals
_card	(-- adr)	EF	PRI	1 byte, card detect transition memory
_FCOPY		EF	PRI	
_file		EF	PRI	4 longs, table entry for the 4 file channels - holds sector address
_file$			PRI	
,ASMONTH	(index --)	EF	PRI	index in range 1-12, display shortform month
,CARD	(--)	EF	PRI	Display 1 line of sdcard properties
,DIR$		EF	PRI	Format a directory name
!SDIO		EF	PRI	Initialize the SD I/O and basic card
!sect	(--)	EF	PRI	
?SDTO		EF	PRI	In SPI Mode, only the OCR, CSD and CID registers are accessible
.ATR	(atr --)	EF	PRI	Display the symbol(s) for each active directory name attribute
.CARD			PRI	
.DIR	(addr --)	EF	PRI	
.DIR$		EF	PRI	
.FAT	(--)	EF	PRI	Displays two lines of sdcard properties
.FDATE	(fdate --)	EF	PRI	Display date in Unix format
.FDATES	(diradr field ch -- diradr)	EF	PRI	Display date
.FTIME	(ftime --)	EF	PRI	Display file time
.FTIMES		EF	PRI	Display file time
.UTIME	(--)	EF	PRI	print the unix file mod time or year if the file is older than 6 months
(.DIR)		EF	PRI	
(.LIST)	(<index> --)	EF	PRI	List a single directory entry in FTP compatible format
(DIR)	(code --)	EF	PRI	
(ls)		EF	PRI	directory list method for ls
@ATR		EF	PRI	Directory structure ptr, Attribute
@CDATE		EF	PRI	Directory structure ptr, Creation date
@CLUSTER	(index - xadr)	EF	PRI	
@CTIME		EF	PRI	Directory structure ptr, Creation time

@DIRBUF!	(word field --)	EF	PRI	Write to directory entry as new date
@FCLST		EF	PRI	Directory structure ptr, First cluster of file (low file)
@FCLSTH		EF	PRI	Directory structure ptr, First cluster (high word)
@FDATE		EF	PRI	Directory structure ptr, Modification date
@FSIZE		EF	PRI	Directory structure ptr, Size of filename
@FTIME		EF	PRI	Directory structure ptr, Modification time
@sdrd	(-- adr)	EF	PRI	1 long
@sdwr	(-- adr)	EF	PRI	1 long
@sector	(-- adr)	EF	PRI	Points to 'sectors'
@sectors	(-- adr)	EF	PRI	4 longs, Current sector loaded in SDBUFs for the 4 files possible
*SDCS	! -- byte)	EF	PRI	cspin C@
=dtk	(-- byte)	EF	PRI	$FE, data token for single block read/write
>F83	(str1 -- str2)	EF	PRI	Format friendly file name into directory format
~!		EF	PRI	
APPEND.BLK	(-- reblk)	EF	PRI	find the active block to use
byte/sect		EF	PRI	1 word
cd!	(sect str --)	EF	PRI	
cid+		EF	PRI	
ClaimClusters	(size startcluster --)	EF	PRI	link clusters and mark end cluster
clshift		EF	PRI	1 byte, cluster shift (fast multiplier)
CLUSTER@	(index -- cluster)	EF	PRI	
crc	(-- adr)	EF	PRI	1 byte, crc
cspin	(-- adr)	EF	PRI	Pointer to cs pin value
cwd$		EF	PRI	16 bytes storage
cwdsect		EF	PRI	1 long
diradrs		EF	PRI	4 longs, virtual memory address of file's directory entry
dirbufs		EF	PRI	A 32 byte directory buffer for each of the 4 file channels
endcl		EF	PRI	constant $0FFFFFFF
FABORT	(code --)	EF	PRI	
fat1	(-- adr)	EF	PRI	1 long
fat2	(-- adr)	EF	PRI	1 long
fat32		EF	PRI	data block for easyfile, see source
fatname	(-- adr)	EF	PRI	8 bytes, always FAT32 - (don't trust)
fatptr	(-- adr)	EF	PRI	points to 'parts'
fats	(-- adr)	EF	PRI	1 byte, Copies of FAT
fboot		EF	PRI	1 long, boot signature - determines whether it needs to remount
FDATE!	(#yymmdd field --)	EF	PRI	Arrange as decimal YYMMDD from 1980 (2000.0000 + 1980.0000 -)
FirstCluster	(diradr - cluster)	EF	PRI	find first cluster of this directory entry
fkey		EF	PRI	word stores backup for input device when input is switched to file
fname$		EF	PRI	file$ stores 4 8.3 filenames at 16 byte boundaries
freads		EF	PRI	4 bytes storing read pointers for 4 files
FreeClusters?	(size -- size startcluster)	EF	PRI	Find free clusters for the file size in bytes - 0 = all, return with address of first free cluster
FreeDir?	(-- fsadr)	EF	PRI	Find the next free directory entry and also set dirfsa
FSADR!	(faddr - addr)	EF	PRI	
fsel	(-- adr)	EF	PRI	1 bytes, currently selected file channel
fstat		EF	PRI	byte stores current status of file system
FTIME!	(#hhmmss field --)	EF	PRI	Update file modification/create time in dir buf, time (5/6/5 bits, for hour/minutes/doubleseconds
fwrites	(-- adr)	EF	PRI	4 bytes storing write pointers for 4 files
lscount		EF	PRI	variable stores the number of files in the current directory
lsdirs	(buffer --)	EF	PRI	scan the buffer for dir entries -- 32 bytes/entry
MARKER?	(marker -- flg)	EF	PRI	Find SD marker and return true before timeout
mksiz	(-- adr)	EF	PRI	long variable, used to create a file if file not found
mounted		EF	PRI	1 byte, true flag if mounted (but also depends upon other checks)
oemname	(-- adr)	EF	PRI	8 chars
parts	(-- adr)	EF	PRI	64 bytes, Room for 4 entries of 16 bytes
RDOCR	(-- ocr)	EF	PRI	
RDSECT	(dst -- crcflg)	EF	PRI	card has been prep'd for read - proceed and read a block of data
READFAT32		EF	PRI	Read and buffer the FAT32 boot record
RES@	(-- res)	EF	PRI	
rootcl		EF	PRI	1 long, Cluster Number of the Start of the Root Directory
rootdir		EF	PRI	1 long, sector address of root directory
rsvd		EF	PRI	1 word
scanch	(-- adr)	EF	PRI	1 byte
scancnt	(-- adr)	EF	PRI	1 word
scanpos	(-- adr)	EF	PRI	1 word
scrc		EF	PRI	Pseudo constant - address of sd crc
scrcs	(-- adr)	EF	PRI	4 longs, sector CRCs
SD@	(-- byte)	EF	PRI	Fetch a byte from the SD card (clock in 1's)
SDCLK	(cnt --)	EF	PRI	Faster byte wide clocks (8/count)
SDDAT!	(adr --)	EF	PRI	Wait for read token and read SD data into buffer
sdsize		EF	PRI	1 long, Number of sectors * byte/sect (512) = capacity
sdtimer		EF	PRI	Timer for use by SD card interface
sect/clust		EF	PRI	1 byte
sect/fat		EF	PRI	1 long, Number of sectors per FAT table
serial		EF	PRI	4 bytes, #67 serial number of partition
STAT@	(-- stat)	EF	PRI	
udir		EF	PRI	calls (.DIR)
volname		EF	PRI	11 bytes, #71 volume name
wrflg	(-- adr)	EF	PRI	
wrflgs	(-- adr)	EF	PRI	4 bytes, one per file channel, indicates current sector buffer has been written to
XADR!	(xaddr -- addr)	EF	PRI	Same as XADR but indicate a write operation for later flushing

UNSORTED PRIVATE WORDS

htck	PRI
htref	PRI
htsav	PRI
tid	PRI
ttint	PRI

Document version

Version 1.4 - A new column for public, private, preemptive and module added, public and private modules are separated
Version 1.3 - Conditional and FOR examples added, removed some duplicated entries, numerous typos, Added navigation to sections in pdf
Version 1.2 - CASE and SWITCH examples added, WWORDS description - added Peter's detail
Version 1.1 - Numerous typos fixed
Version 1.0 - Adapted from this glossary, by Bob Edwards, in September 2020 using Tachyon V5r7 NEON 570190926.2300 fitted with EXTEND and EASYFILE modules