

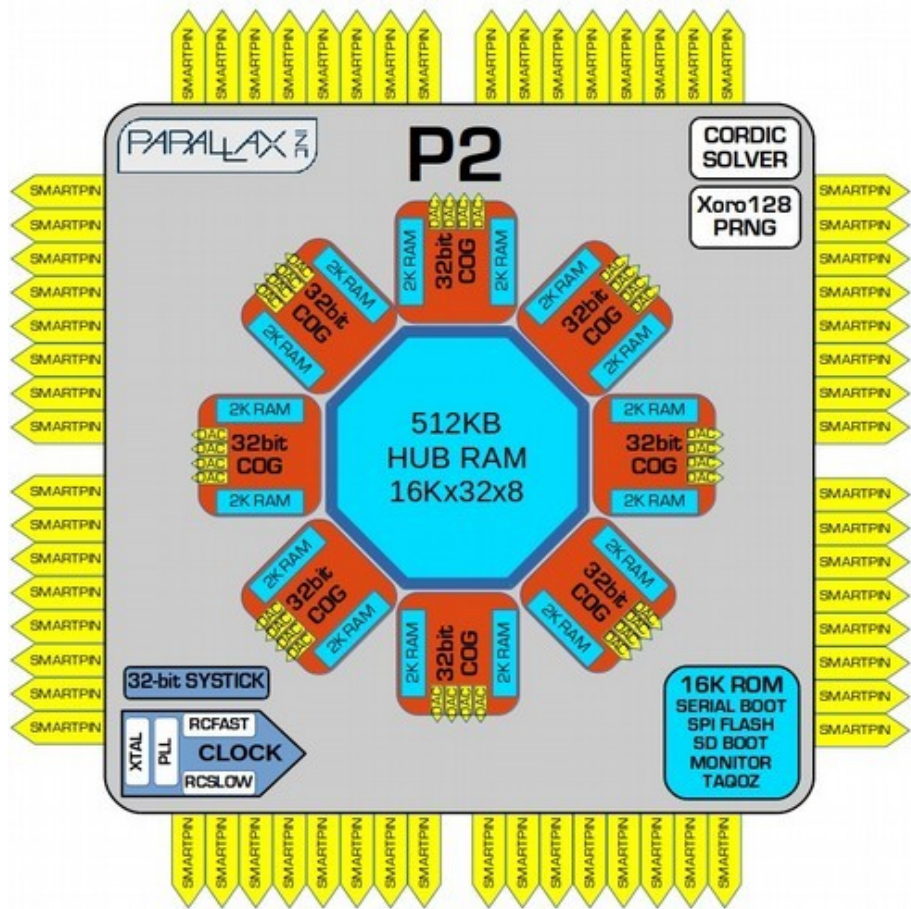
# TAQOZ Reloaded V2.8 GLOSSARY of WORDS

The Propeller P2

TAQOZ Reloaded is a really powerful extended version of TAQOZ ROM created by Peter Jakacki for the Propeller P2. The P2 is a feature packed 8 core 32 bit microcontroller from Parallax Inc. TAQOZ is a **Forth programmers' tool set** for the Parallax Propeller P2. [This article](#) explains how it came to be. Whereas TAQOZ ROM is permanently available in every P2, TAQOZ Reloaded resides on Flash memory or SD card.

This glossary is intended to be kept by you when programming. The headings and words are in alphabetic order to speed up searches. Acrobat Reader from Adobe will show a bookmark strip you can click on to jump to a particular section. Use **<ctrl> F** to search for a specific word. Please note that there are differences between this glossary for TAQOZ Reloaded and [TAQOZ ROM glossary](#) although many words work identically.

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



### About the format of the word tables

- The **WORDS** column shows TAQOZ words
- The **STACK EFFECT** column shows the effect each word has on the data stack - in effect the required input parameters and results for each word
- The **PRE?** column (if yes is present) shows the word is preemptive, that is, will execute immediately, even if within a code definition. Preemptive words can be overridden with the word [C] to compile within a definition
- The **DESCRIPTION** column briefly describes what the word does or provides a working example
- The **TIME** column shows the number of clock cycles the instruction takes

### Entering data

Data is referred to as bytes (ch, 8 bits), words (w, 16 bits), longs (n, 32 bits) and doubles (d, 64 bits). If the default number base is not the one required then the number base is best denoted by a prefix \$<number> = hexadecimal, #<number> = decimal and %<number> = binary.  
e.g.                                      \$AF                                      #1234                                      %1011101

If a double (64 bits) is required, this is indicated by a . suffix e.g. 1234. or \$4D2. puts 0 top of stack, with 1234 decimal next below.

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

If it is required to place a single printable character on the data stack, then enclose the char in single quotes e.g. 'A' places 41 hex on the stack

## LIST OF DEBUGGING CONTROL KEY SHORTCUTS

ESC ESC ESC ESC	Reset - preserving the unsaved new words in RAM * - very handy to get out of an endless loop
^Z ^Z ^Z ^Z	Reboot like switch on - losing the unsaved new words in RAM *
^T^T^T^T	Start word TRACE - each word executed is displayed with a snapshot of the data stack, execution is slowed *
^U^U^U^U	Stop TRACE, execution runs full speed *
ESC	Discard the current console line input
^C	Reset
^W	List 6 lines of most recently defined words in the dictionary (type WORDS to display all, or LWORDS <chars> to list selected)
^D or ^?	Runs word DEBUG which displays data stack, COG ram, the registers, CODE space, Dictionary space and I/O status
^Q	Print data stack
^S	Clear data stack, then places a single value \$DEADBEEF on the stack, to check stack balance when debugging new words
^B	Runs the command BU to back up the system to SD card
^F	Save the current TAQOZ image to flash memory
^X	Re-eXecute last line (very useful)

\* works even if TAQOZ is not checking the console. If TAQOZ is at the user prompt, then only one control character is required, not four.

# ANSI TERMINAL

TAQOZ, by default, expects a computer terminal connected to smartpin P62 for data transmitted by the P2 and smartpin P63 for data received by the P2.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
*ANSI*	( -- )	no	Module header marker – use with FORGET		extend.fth
%black	( – 0 )	no	constant colour		extend.fth
%BLINK	( -- )	no	Set blinking - sets character red in tera term and no blinking		extend.fth
%blue	( – 4 )	no	constant colour		extend.fth
%BOLD	( -- )	no	Character set to bold yellow		extend.fth
%CLS	( -- )	no	Clear screen – doesn’t appear to work on Tera Term in Windows. I use %ERSCN %HOME instead		extend.fth
%CURSOR	( on/off – )	no	Show the cursor – ON %CURSOR Hide the cursor – OFF %CURSOR		extend.fth
%cyan	( – 6 )	no	constant colour		extend.fth
%ERLINE	( -- )	no	Erase the current line		extend.fth
%ERSCN	( -- )	no	Erase the screen from the current location		extend.fth
%green	( – 2 )	no	constant colour		extend.fth
%HOME	( -- )	no	Home the cursor top left of the window		extend.fth
%magenta	( – 5 )	no	constant colour		extend.fth
%PAPER	( col – )	no	e.g. %green %PAPER		extend.fth
%PEN	( col – )	no	e.g. %red %PEN		extend.fth
%PLAIN	( -- )	no	White characters on a black background		extend.fth
%red	( – 1 )	no	constant colour		extend.fth
%REVERSE	( -- )	no	Swap character and background colour		extend.fth
%UL	( -- )	no	Set underline		extend.fth
%white	( – 7 )	no	constant colour		extend.fth
%WRAP	( on/off – )	no	Word wrap at end of line		extend.fth
%XY	( x y – )	no	Move cursor to x column, y row. 0 0 is top left		extend.fth
%yellow	( – 3 )	no	constant colour		extend.fth
ESC	( ch -- )	no	Send an ESC – ch sequence		extend.fth
ESCB	( ch -- )	no	Send an ESC - [ - ch sequence		extend.fth

# ASSEMBLY LANGUAGE

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
(ASM)	( -- )	no	Inline assembly language within a forth word e.g. <forth words> .. (ASM) <assembly code> (CODE) .. <more forth words		kernel
(CODE)	( -- )	no	Terminates in-line assembly code, see (ASM)		kernel

Note: This basic assembly language facility for entering machine code instructions in hexadecimal has been superceded by P2ASM, the Taqoz Interactive Assembler, described elsewhere

# BENCHMARK

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
fibonacci	( n – f )	no	Find the nth fibonacci element		extend.fth
fibonacci	( -- )	no	Print a number of fibonacci runs		extend.fth

# BRANCHING

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?EXIT	( flg -- )	no	Exit word if flg is true		kernel
<CASE>		yes	See SWITCH example below		kernel
0EXIT	( flg -- )	no	Exit word if flg is false (or zero) Used in place of IF.....THEN EXIT as false would just end up exiting		kernel
BREAK		yes	See SWITCH example below		kernel
CALL	( adr -- )	no	Call the code field address on the top of stack		kernel
CASE	( compare – )	yes	See SWITCH example below		kernel
CASE@	( – value )	no	Returns the value stored by the last SWITCH word		kernel
ELSE		yes	part of IF .. ELSE .. THEN structure		kernel
EXECUTE	( addr -- )	no	Enter code or threaded code at address		kernel
EXIT	( – )	no	Exit word now - pops return stack into IP		kernel
GOTO		yes	Used internally by IF and ELSE - compile a dummy NOP/GO to be replaced later with a goto (addr+ex)	internal use only	kernel
IF	( n1 – )	yes	If n1 true, do the words within IF ... THEN, or IF .. ELSE .. THEN		kernel

JUMP	( adr -- )	no	Jump to forth word code at adr		kernel
NOP	( – )	no	do nothing	72	kernel
REVECTOR	( <target> <new> -- )	yes	Revector any calls to <target> to jump to <new> by rewriting first wordcode. <target> and <new> are both words taken from the input stream e.g. REVECTOR mytargetname mynewname		
SWITCH	( val – )	no	See SWITCH example below		kernel
THEN		yes	part of IF .. ELSE .. THEN structure		kernel
VECTOR:		no	Create a vector table of fast jump addresses - see example below		extend.fth

**SWITCH example:-**

```

pub CASETEST ( val -- )
SWITCH
  1 CASE ." one" BREAK
  2 CASE ." two" BREAK
  3 CASE ." three" BREAK
  4 10 <CASE> ." between four and ten" BREAK
  ." default"
;

```

**VECTOR: example:-**

```

: BABA ." BABA" ;
: GAGA ." GAGA" ;
: DADA ." DADA" ;
: MYCHOICE VECTOR: BABA GAGA DADA NOP ;
Usage:
1 MYCHOICE prints GAGA on the terminal
0 MYCHOICE prints BABA on the terminal etc
It's down to you to guard against out of range values

```

CHARACTER I/O

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
@WORD	( – n )	no	Returns address of buffer where words from the input stream are assembled		kernel
?ERROR	( cond n -- )	no	Display error message and EXIT		kernel
."	( -- )	yes	Used to define a string for immediate print ** e.g. ." This prints immediately"		kernel
.CLK	( -- )	no	Prints the processor clock frequency e.g. 200MHz		kernel
.EMIT	( ch – )	no	Emit an alphanumeric character, else emit a dot for nonprintable	1248	kernel
.OK	( -- )	no	print OK!		extend.fth
.SUCCESS	( flag – )	no	if true, print SUCCESS!, else print ERROR!		extend.fth
.VER	( -- )	no	Prints 'Parallax P2 *TAQOZ RELOADED sIDE* V2.8 'CHIP' Prop_Ver G C8MHz 210401-1230' or similar on the output stream		kernel
+BUF	( -- )	no	this is code pointed to by uemit to send an output chr to the buffer set up by =BUF. The buffer wraps on overflow		extend.fth
+INFO	( -- )	no	Change prompt to include current code pointer		extend.fth
=BUF	( address size -- )	no	initialise a new chr o/p buffer at addr and set EMIT to jump to the +BUF code - all subsequent calls to EMIT will then fill this buffer instead of going to the terminal		extend.fth
BARE	( -- )	no	Plain non-interactive responses for remote operation - Remove prompt and accept strings		extend.fth
BUFFERS	( – n )	no	returns the address of a 2k buffer used when reading files from SD card	104	kernel
CLS	( – )	no	clear the screen using \$0C EMIT (doesn't work with tera term under Windows, but %ERSCN %HOME works well)	616	kernel
COM	( pin -- )	no	direct output to a smartpin (after init)		kernel
CON	( – )	no	select console output	248	kernel
CON?	( – flag )	no	Is console active?		extend.fth
CONEMIT	( chr – )	no	output chr to the console, regardless of any redirection	48	kernel
CONKEY	( – chr )	no	Read ch from the console, if zero then no key was read, regardless of any redirection	376	kernel
CONOUT	( -- )	no	Redirect the output stream via MBXEMIT, mailbox emit		extend.fth
CONSOLE		no	Main console line loop – get a new line (word by word)		kernel
CR	( – )	no	Emit carriage return	608	kernel
CRLF	( – )	no	Emit a carriage return line feed	6640	kernel
CTRL!	( fnc key -- )	no	Set a user function to be triggered by a control key press e.g. ' myfunction 'A' CTRL! will cause myfunction to run when CTRL-A is pressed		kernel
CTYPE	( str cnt -- )	no	Display cnt chars, starting at address str		kernel
DISCARD		no	Discard the current line	4002296	kernel
EMIT	( ch – )	no	Emit a char to the current output stream	520	kernel
EMIT;		no	Restore the setting previously save by EMIT:		extend.fth
EMIT:		no	save the current setting, and redirect (using uemit) all character output to the code that follows		extend.fth
EMIT!	( cfa -- )	no	Set the emit vector or 0 to fall through to CONEMIT	192	kernel
EMIT@	( – cfa )	no	Read the current emit vector	168	kernel

EMITS	( ch n – )	no	Emit n copies of chr to the current output stream	Around 257 per chr	kernel
EOL	( -- adr )	no	Called at end of line - default action is do nothing – deferred word executed as part of the TAQOZ input stream interpreter.		kernel
GET\$	( -- str )	no	Build a delimited word in wordbuf for wordcnt and return immediately upon a valid delimiter, like space, tab etc		kernel
KEY	( -- ch )	no	Read ch from the current input stream, if zero then no key was read	1280 if char found else 1032 if no char	kernel
KEY:		no	redirect (using ukey) all character input to the code that follows		extend.fth
KEY!	( ch -- )	no	Force a character as the next KEY read	224	kernel
KEY@	( – n )	no	read last 4 keys	152	kernel
LOCAL	( -- )	no	Reset prompt to local interactive		extend.fth
MBO		no	Disable serial in / out on smartpins 62 / 63		kernel
MBX		no	Use 'mailbox' as console for both transmit and receive		kernel
MBXEMIT	( chr -- )	no	Waits until the mailbox buffer has been read, then sends chr to to that mailbox buffer as the console output stream		extend.fth
MOUT	( addr -- )	no	Save output to hub memory address	176	kernel
NONE		no	discard all console output		kernel
NULL		no	discard all console output - later word has same effect as NONE?		extend.fth
PRINT"	( -- )	yes	Used to define a string for immediate print ** e.g. PRINT" This prints immediately"		kernel
PRINT\$	( str – )	no	Display string at address str ** e.g. 16 bytes mystring “ FRED” mystring \$! mystring PRINT\$		kernel
PROMPT	( -- )	no	execute user prompt code	14464 with default	kernel
PROMPT!	( cfa -- )	no	Set the prompt or use 0 to reset to default		extend.fth
SPACE	( -- )	no	emit a space char	584	kernel
SPACES	( n – )	no	emit n spaces	Around 566 per space	kernel
SPIN	( -- )	no	Emit the next character in a spinner sequence ( one of   / - \ ) using backspace to reposition		kernel
SPINNER	( -- )	no	Emit the next character in a spinner sequence ( one of   / - \ ) using backspace to reposition		extend.fth
TAB	( -- )	no	print the tab character		extend.fth
TABS	( n -- )	no	print n tab chars		extend.fth
WKEY	( -- ch )	no	wait for a key and return with character		kernel

\*\* To print a non-visual character, include the chars \ \$nn in the string e.g. PRINT" \ \$07" sounds the bell on the terminal.

#### GET\$ example

Quite often a PC <-> TAQOZ interface is possible using the TAQOZ console interpreter alone. The PC simply writes commands that you would write at the terminal. Where this won't do, the GET\$ word becomes useful for loading numbers (it has other applications of course) e.g.

```

--- read 'count' number of longs directly off the console input stream into an array starting at 'adr'
pub GETLONGS ( adr cnt -- )
FOR
    DUP                --- backup of the array address
    GET$ $># DROP      --- read and convert the next word in the console input stream to a long
    SWAP !             --- save the number in the array
    4+                 --- bump the address to the next array element
NEXT                  --- and repeat for all 'cnt' elements
DROP                  --- drop the address
;

```

How to add a PS-2 keyboard to the P2 is [explained in this article](#).



# COG / HUB

From switch on, TAQOZ runs in COG 0 with COG7 running a vga display on smartpins P0 to P4. COGS 1 to 6 are idle. All COGS are potentially available for the user's application.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
AUTO	( -- )	yes	e.g. AUTO MAIN followed by a backup, would make MAIN the autostart routine		kernel
COGATN	(mask -- )	no	Strobe the “attention” event flag for all cogs whose corresponding bits are high in mask ( bit 0 - 8 for P2 )	56	kernel
COGID	( – n )	no	Return current cog#	64	kernel
COGINIT	( adr cog – )	no	Start cog no. ‘cog’ with start up address ‘adr’		kernel
COGMOD	( -- )	no	Run the code previously loaded to COG Ram by LOADMOD - see example below		kernel
COGSTOP	( cog -- )	no	Stop a cog that is running a TAQOZ word	56	kernel
COLD	(– )	no	Cold start TAQOZ - Taqoz will restart, but only with the kernel and SPIFLASH module loaded. The user can then select which other modules are loaded for the job in hand		extend.fth
FILTER!	( tap0..31 len0..3 filter0..3 -- )	no	%0100_xxxx_xxxx_xxxx_xxxx_xxxR_RLLT_TTTT Set filter R to length L and tap T		extend.fth
HUBSET	( D -- )	no	Set hub configuration - see note below		kernel
LOADMOD	( src len -- )	no	Load COG memory from hub. With this version of TAQOZ, the COG memory starts at \$1CB. 40 instructions (longs) appears to be the maximum permitted size. * see comment below		kernel
NEWCOG	( n -- )	no	An idling instance of TAQOZ is loaded and run in cog#n	64	kernel
P2?	( – n )	no	Return the P2 revision number	72	kernel
POLLATN	( – flag )	no	The ATN flag is a means of one cog shouting oi! to another flag = TRUE the ATN flag was received, ATN event flag then cleared. flag = FALSE, no ATN flag received – see <a href="#">this example code</a>	72	kernel
REBOOT	( – )	no			kernel
REG	( index -- addr )	no	Find the address of the HUB REGISTER pointer specific to each COG. e.g. 38 REG C@ returns the current number base (variable pbase in the TAQOZ source code) From switch on, 0 REG returns \$200 for COG 0		kernel
RESET	( – )	no			kernel
RUN	( task cog -- )	no	e.g. ‘ MYWORD 6 RUN Run word MYWORD in COG 6	5584	extend.fth
RUN:	( cog -- )	no	Run the code that follows this word as a cog task		extend.fth
TASK	( cog -- addr )	no	Return with address of task control register in "tasks"		kernel
TERM	( -- )	no	Start main terminal console		kernel

\* N.B. Christof Eb warns that in some versions of TAQOZ e.g V2.8 'CHIP' Prop\_Ver G 200MHz 210401-1230. , word LOADMOD does not drop src and len from the stack like it should. If you find this too, use LOADMOD 2DROP as a bug fix.

## COG REGISTER LAYOUT

The cog's 512 x 32 dual-port COG RAM registers, are partially mapped as follows:-

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
IJMP3	( -- \$1F0 )	no	Constant, interrupt call address for INT3		extend.fth
IRET3	( -- \$1F1 )	no	Constant, interrupt return address for INT3		extend.fth
IJMP2	( -- \$1F2 )	no	Constant, interrupt call address for INT2		extend.fth
IRET2	( -- \$1F3 )	no	Constant, interrupt return address for INT2		extend.fth
IJMP1	( -- \$1F4 )	no	Constant, interrupt call address for INT1		extend.fth
IRET1	( -- \$1F5 )	no	Constant, interrupt return address for INT1		extend.fth
PA	( -- \$1F6 )	no	Constant, CALLD-imm return, CALLPA parameter, or LOC address		extend.fth
PB	( -- \$1F7 )	no	Constant, CALLD-imm return, CALLPB parameter, or LOC address		extend.fth
PTRA	( -- \$1F8 )	no	Constant, pointer A to hub RAM		extend.fth
PTRB	( -- \$1F9 )	no	Constant, pointer B to hub RAM		extend.fth
DIRA	( -- \$1FA )	no	Constant, output enables for P31..P0		extend.fth
DIRB	( -- \$1FB )	no	Constant, output enables for P63..P32		extend.fth
OUTA	( -- \$1FC )	no	Constant, output states for P31..P0		extend.fth
OUTB	( -- \$1FD )	no	Constant, output states for P63..P32		extend.fth
INA	( -- \$1FE )	no	Constant, input states for P31..P0 also debug interrupt call address		extend.fth
INB	( -- \$1FF )	no	Constant, input states for P63..P32 also debug interrupt return address		extend.fth

Access to the above cog ram registers is by using words COG@ and COG!

**HUBSET** - The hub contains several global circuits which are configured using the HUBSET instruction. HUBSET uses a single D operand to both select the circuit to be configured and to provide the configuration data:-  
D format:-

%0000_ <b>xxx</b> E_ DDDD_ DDMM_ MMMM_ MMMM_ PPPP_ CCSS	Set clock generator mode
%0001_ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b>	Hard reset, reboots chip
%0010_ <b>xxxx</b> _ <b>xxxx</b> _ <b>xx</b> LW_ DDDD_ DDDD_ DDDD_ DDDD	Set write-protect and debug enables
%0100_ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxxx</b> _ <b>xxx</b> R_ RLLT_ TTTT	Set filter R to length L and tap T
%1DDD_ DDDD_ DDDD_ DDDD_ DDDD_ DDDD_ DDDD_ DDDD	Seed Xoroshiro128** PRNG with D

**ATN examples**  
A cog may alert another cog by means of the ATN, the attention flag. Here's a demo - the SLAVE is mute until it receives an ATN from the MASTER:-

```
--- Just loop silently until an ATN flag is received
pub SLAVE      ( -- )
  BEGIN
    POLLATN
    IF
      ." Slave received ATN, thanks!" CRLF
    ELSE
      30 ms
    THEN
  AGAIN
;
```

```
--- Output a message to show the MASTER looping. Set cog 5's ATN flag on each pass
: MASTER      ( -- )
  BEGIN
    200 ms
    ." Hello from the Master, wake up cog# 5" CRLF
    %100000 COGATN      --- Send ATN to cog 5
    KEY
  UNTIL
  ;

--- set them both going - press any key to stop MASTER and note SLAVE falls silent
' SLAVE 5 RUN
MASTER
```

**COGMOD example**

```
--- create a test word - up to 40 instructions      --- copy the word to COG ram, starting at $1CB      --- execute the word from COG Ram and print
code test ( value n -- value+3*n 0 )                --- results

  add b,#3
  sub a,#1 wz
  if_nz jmp #\@COGMOD
  ret
end
```

```
AT test 2+ 4 LOADMOD

10 5 COGMOD . .
```

COMMENTS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
---	( -- )	yes	Ignores the rest of the line		kernel
(	( -- )	yes	Ignores characters until closing brace ) - use on one line only		kernel
{	( -- )	yes	Ignores characters until closing brace } - use across multiple lines		kernel
}	( -- )	yes	Close a multi line comment		kernel
\	( -- )	yes	Ignores the rest of the line, does not echo		kernel

[This article 'Comment all you like'](#) is worth checking.

COMPARISON

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
<	( n1 n2 -- flg )	no	true if n1 less than n2	80	kernel
<=	( n1 n2 -- flg )	no	true if n1 less than or equal n2	200	kernel
<>	( n1 n2 -- flg )	no	true if n1 is not equal to n2	56	kernel
=	( n1 n2 -- flg )	no	true if n1 is equal to n2	56	kernel
=>	( n1 n2 -- flg )	no	true if n1 greater than or equal n2	168	kernel
>	( n1 n2 -- flg )	no	true if n1 greater than n2	80	kernel
0<	( n1 n2 -- flg )	no	true if n less than 0	24	kernel
0<>	( n1 n2 -- flg )	no	true if n1 not equal 0	32	kernel
0=	( n1 n2 -- flg )	no	true if n1 equals 0	32	kernel
CLIP	( n limit -- res   0 )	no	If n>limit, returns 0, else returns n	328 max	extend.fth
U<	( u1 u2 -- flg )	no	true if u1 less than u2	72	kernel
U>	( u1 u2 -- flg )	no	true if u1 greater than u2	160	kernel
WITHIN	( n lo hi -- flg )	no	true if n is within range of low and high inclusive	416	kernel

CONSTANTS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-1	( – n1 )	no	n1 = -1	56	kernel
1M	( – n )	no	Constant 1000000	112	extend.fth
64KB	( – n )	no	Constant 64 KB	112	extend.fth
BMP	( – adr )	no	Returns constant adr BMPORG	112	kernel
CLKMHZ	( -- n )	no	Returns default P2 clock frequency in MHz	112	kernel
CELLS	( -- n )	no	Returns 4, the number of addressable bytes in a long		extend.fth
delim	( – n )	no	this stores the delimiter used in text input and a save location		kernel
FALSE	( – n1 )	no	n1 = 0	56	kernel
inbox	( – n )	no	pointer to the input stream buffer		kernel
names	( – n )	no	this stores address of the latest word defined in the dictionary (builds down)		kernel
OFF	( – n1 )	no	n1 = 0	56	kernel
ON	( – n1 )	no	n1 = -1	56	kernel
outbox	( – n )	no	pointer to the output stream buffer		kernel
TRUE	( – n1 )	no	n1 = -1	56	kernel
uaccept	( – n )	no	constant,this address stores addr of code to execute when Forth accepts a line to interpret (0=ok)		kernel
uemit	( – n )	no	constant, this address stores the address of the emit code		kernel
ufind	( -- n )	no	constant, returns relative address of extended dictionary flag		kernel
ukey	( – n )	no	constant, stores the address of the key code		kernel
unum	( – n )	no	constant, stores address of user number processing routine - executed if number conversion and word search in dictionary failed. unum routine should return true if successful, else false		kernel
uprompt	( – n )	no	constant, stores address of code to execute when Forth prompts for a new line		kernel

CONVERSION

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
B>L	( b1 b2 b3 b4 – n1 )	no	n1 = b4.b3.b2.b1 four bytes to long conversion	624	kernel
B>W	(b1 b2 – w1 )	no	w1 = b2*256 + b1	176	kernel
L>S	( n -- lsb9 h )	no	specialized operation for filesystem addresses ( splits off 9 lsbs )	280	kernel
L>W	( n1 – w1 w2 )	no	splits long n1 into to words	200	kernel
W>B	( w1 – b1 b2 )	no	splits word w1 into bytes	216	kernel
W>L	( w1 w2 – n1 )	no	n1 = w2*65536 + w1	200	kernel

DEBUG

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?PIN	( pin -- )	no	Print report of measuring pin capacitance as a float to high time		extend.fth
.CONS	( -- )	no	Print all constants		extend.fth
.CONVARS	( -- )	no	Print all constants and variables		extend.fth
.CTRLS	( -- )	no	Print all keyboard shortcuts		extend.fth
.DEVICES	( – )	no	Print report of loaded modules: on the P2-EVAL that is the SD card, I2C devices and clock frequency details		extend.fth
.io	( -- )	no	Print the state of all 64 i/o pins on two lines		extend.fth
.LAP	( -- )	no	print the time taken between LAP words - see below for a non-printing alternative		kernel
.LAPS	( n -- )	no	e.g. LAP 100 0 DO I DROP LOOP LAP 100 .LAPS Prints the time per pass through the loop		kernel
.MODULES	( – )	no	Print a list of loaded modules		extend.fth
.ms	( -- )	no	Another name for .LAP		kernel
.PCB	( – )	no	If this is executed on a P2D2 card, then this prints P2D2 else P2 for all others		extend.fth
.PU	( -- )	no	57 lines of some kind of speed test		extend.fth
.S	( -- )	no	Print out the data stack		kernel
.USAGE	( – )	no	Prints a memory use report e.g. CODE: 079BC 31,164 bytes WORDS: 1DA82 9,493 bytes DATA: 01B3D 419 bytes		extend.fth

			ROOM: 90,310 bytes ok		
.VARS	( -- )	no	Print all variables		extend.fth
COG	( -- )	no	prefix for DUMP to show cog ram		kernel
DEBUG	( -- )	no	Print the stack(s) and dump the registers - also called by hitting <ctrl>D during text input	7396208	kernel
DEVICES	( - )	no	Reports the devices connected to the P2. On the P2-EVAL that is: SD CARD 31 GB SANDISK SD SD32G REV\$85 #3200344782 DATE:2021/1		extend.fth
DUMP	( startadr cnt - )	no	e.g. \$0 \$100 DUMP, or a modifier can be used: \$0 \$100 RAM DUMP Data is shown in bytes		kernel
DUMP!	( 'C@ 'W@ '@ - )	no	Creating a new DUMP device: If at least one method is coded equivalent to C@ which takes an address and returns a byte, then it is possible to create a modifier for DUMP. In this example we will create one for the ROM which can normally be accessed at \$FC000 but we will make it appear as its own memory addressed from 0. pri ROMC@ ( addr -- byte ) \$FC000 + C@ ; pri ROMW@ ( addr -- byte ) \$FC000 + W@ ; pri ROM@ ( addr -- byte ) \$FC000 + @ ; pub ROM AT ROMC@ AT ROMW@ AT ROM@ DUMP! ; Use: \$1000 \$40 ROM DUMP		kernel
DUMPA	( startadr cnt - )	no	Like DUMP but data is shown as .ascii		kernel
DUMPAW	( startadr cnt - )	no	Like DUMP but data is shown wide format as .ascii		kernel
DUMPL	( startadr cnt - )	no	Like DUMP but data is shown as longs		kernel
DUMPW	( startadr cnt - )	no	Like DUMP but data is shown as words		kernel
DUMPZ	( src cnt -- )	no	dump memory as bytes but skip any 128 byte page that is all 0's or FF's		extend.fth
FL	( -- )	no	prefix for DUMP to show flash ram e.g. 0 \$20 FL DUMP		kernel
I2C	( -- )	no	prefix for DUMP to read bytes from i2c device		extend.fth
LAP	( -- )	no	Used to time words e.g. LAP <forth words> LAP .LAP will display how long the code took to execute		kernel
LAP@		no	used internally by .LAP		kernel
Isi2c	( - )	no	Print list i2c devices		extend.fth
Isio	( -- )	no	display all 64 pins states, high low etc	220808	kernel
LUT	( -- )	no	prefix with DUMP to show look up table (lut) ram		kernel
LWORDS	( -- )	yes	detailed dictionary word list with optional match characters defined by next word in input stream		extend.fth
MWORDS	( width - )	no	print dictionary in field 'width'		extend.fth
QD	( startadr - )	no	quick dump of 32 bytes from addr		kernel
QW	( startadr - )	no	Displays two lines of DUMP as words		kernel
REDUMP		no	Redump a byte dump listing back into memory - exit on empty line		extend.fth
RTC	( -- )	no	prefix with DUMP for listing real time clock registers		extend.fth
SD	( - )	no	select SD for DUMP method : use: 0 \$200 SD DUMP		file.fth
SYSINFO	( - )	no	Prints a report of: Loaded Modules, Memory usage, PCB type, P2 external clock frequency, SD card details, I2C device connected and P2 internal clock frequency		extend.fth
TRACE	( -- )	no	Activate TRACE module displaying instruction pointer, code and name of word about to be executed, and the state of the data stack. Use: TRACE <words> UNTRACE		kernel
TRACE!	( floor -- )	no	Set the TRACE module to only trace from this address or higher and activate TRACE - see example below		extend.fth
UNTRACE	( -- )	no	Turns off the TRACE module		kernel
W	( -- )	no	Print the 6 most recent lines of words		kernel
WORDS	( -- )	yes	Print dictionary full width. If -L next word in input stream, then the next words after that is a search string e.g. WORDS -L BL prints all words starting with BL		extend.fth

**New DUMP device example** - If at least one method is coded equivalent to C@ which takes an address and returns a byte, then it is possible to create a modifier for DUMP.  
In this example we will create one for the ROM which can normally be accessed at \$FC000 but we will make it appear as its own memory addressed from 0.

```
pri ROMC@ ( addr -- byte )    $FC000 + C@ ;
pri ROMW@ ( addr -- byte )    $FC000 + W@ ;
pri ROM@ ( addr -- byte )     $FC000 + @ ;
pub ROM                        AT ROMC@  AT ROMW@  AT ROM@  DUMP! ;
```



**TRACE! example** - most of the time we want to trace a recently defined word. Tracing through the rest of TAQOZ isn't useful. TRACE! allows us to filter the trace by address, so we trace only a few most recently defined words. Define a few test words ...

set detail level and switch trace on ...

set deeper trace level and switch trace on ...  
Thanks for this example Christof Eb

```
: wordA 1 - ;
: wordB wordA ;
: wordC
  2
  begin wordB dup 0 <= until
  drop
;
' wordB TRACE!
wordC UNTRACE
' wordA trace!
wordC UNTRACE
```

[This article 'Dumping permitted'](#) explains how to use DUMP and friends effectively. Some of the other powerful tools are [explained here](#).

The WORDS command has been made more useful by colour coding the lists it produces. Here's the key:-

Highlighting scheme used in dictionary listing	
pub	Public threaded words (normal : definitions)
pri	Private words - can be safely stripped & reclaimed from memory
pre	Preemptive immediate words (executes when typed - before enter)
DATA	DATA variable - points to variable in DATA memory
CON	Constant
COG	Cog kernel assembly code
HUB	Hubexec kernel assembly code

<b>Alternative to .LAP</b> If code timing needs to be part of a process, then displaying the result isn't so useful. Instead we can use code to leave the number of cycles taken on the data stack. Here's a word TEST to time:-  Here we time TEST and the number of cycles it took is placed on the data stack.	pub TEST 10 FOR I . NEXT ;  LAP TEST LAP LAP@ LAP LAP LAP@ - . --- 0 1 2 3 4 5 6 7 8 9 70208 ok
--	--

DECOMPILER

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
*DECOMPILER* ( -- )		no	Module header marker – use with FORGET		extend.fth
DECOMP ( -- )		yes	Prints a ‘source code’ decompilation of the words whose name is next in the input stream e.g. DECOMP WORDS		extend.fth
SEE ( -- )		yes	Prints a detailed listing of the word whose name is next in the input stream e.g. SEE MYWORD		extend.fth
SEECFA ( cfaptr -- )		no			extend.fth

This article explains about this powerful inspection tool, [the decompiler](#).

DEFER

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
->	( -- )	no	Defers a word executing until the end of a line. e.g. in simulating prefix operation in an assembler, where OPCODE <params> is preferred		extend.fth
CALL\$	( string -- )	no	Calls the word, if found in the dictionary, held in the string whose address is top of stack. Does not echo the word to the display e.g. " WORDS" CALL\$ will execute WORDS		extend.fth
EVALUATE	( string -- )	no	Calls all the words held in the string whose address is top of stack. Note: echoes those words to the display. Won't evaluate word definitions		extend.fth

-> example:-

```
: test1 -> ." test1 executed " ;
: test2 ." test2 executed " ;
: test3 test1 test2 ; when run, displays 'test2 executed test1 executed'
```

This is handy when sending a Taqoz based controller a command sentence like: 'CMDword1 parameter1 parameter2 parameter3'. If CMDword1, CMDword2 etc are made forth words, then the sentence can be parsed using the TAQOZ interpreter, rather than creating another special interpreter. But to do that cleanly, we would really want the command sentence to be translated to 'parameter1 parameter2 parameter3 CMDword1'. So the -> word defers the execution of CMDword1 to be last, so that all the parameters are on the stack ready for CMDword1 to process. The TAQOZ assembler makes use of -> so the user can write conventional assembly language rather than RPN style.

## DEFINING WORDS

Do not define words that start with \$ # or % and also contain valid numbers e.g. a word defined as \$GPRMC does not run properly, whereas a word defined as \$GPRMG does. The interpreter compiles \$GPRMC, but when run erroneously just puts \$C on the stack.  
Also do not define words that end in underscore \_ , they will compile but will not be interpreted or execute.

Words in TAQOZ can be recursive, i.e. they can call themselves without any special words to do that. However, remember the [depth of the return stack](#) is a limiting factor.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
'	( -- cfa )	yes	Using the next word in the input stream, look it up in the dictionary and return code field address		kernel
,	( n -- )	yes	Compile long n into code memory		kernel
;	( -- )	yes	Compile an EXIT word and ends word definition		kernel
:	( -- )	yes	an alias for pub		kernel
[	( -- )	yes	Flag that we have entered a definition - see below for an example		kernel
]	( -- )	yes	Used with [ - end definition and lock allocated bytes		kernel
[C]	( -- )	yes	Force compilation of the next word in the input stream even if it was defined as preemptive		kernel
[G]	( -- )	no	an alias for GRAB N.B. this alias is not preemptive, whereas as GRAB is preemptive, so they aren't an exact match		kernel
[W]	( wordcode -- )	yes	append this wordcode to next free code location + append EXIT (without counting)		kernel
	( ch -- )	yes	Compile byte ch into code memory		
	( w -- )	yes	Compile word w into code memory		kernel
+EXIT		yes	end definition and lock allocated bytes		kernel
ALIGN	( adr1 n -- adr2 )	no	adr2 = adr1 realigned on an n'th byte boundary		kernel
@CODES	( -- adr )	no	Point to code compilation memory		kernel
@HERE	( -- atradr )	no	Variable which holds the code pointer HERE		kernel
ALIAS	( -- )	yes	Create an alternative name for an existng word e.g. ALIAS <existingword> <newword>. <b>N.B. See the bug fix below</b>		extend.fth
ALLOT	( n -- )	no	Allot n bytes of code memory - increments HERE by n. n must be an even number		kernel
CREATE:	( -- )	yes	Create a name header in the dictionary using the next word in the input stream and compile a VARIABLE TYPE code entry		kernel
CREATE\$	( -- )	yes	Create a name header in the dictionary using the string that follows in the input stream		kernel
END		no	Marks the end of source code to be 'block loaded' - see also TAQOZ word. N.B. The word must be on it's own on a line		kernel
EXIT;		yes	compile an EXIT		kernel
GRAB	( -- )	yes	immediate word, executes preceding code to make it available for any immediate words following		kernel
HERE	( -- addr )	no	Address of next compilation location		kernel
IFDEF	( -- )	yes	If the next name in the input stream is already DEFINED then process all input stream that follows up to the terminating curly brace }		kernel
IFNDEF	( -- )	yes	If next name in the input stream is NOT DEFINED then process all source between here and the curly brace }		kernel
MKEY	( -- chr   false )	no	Part of the TAQOZ word: return chr from hub memory buffer. If chr=0, return false and switch input stream back to the terminal		kernel
MLOAD	( adr -- )	no	Part of the TAQOZ word: divert the input stream to a buffer in hub memory starting at address adr		kernel
MSAVE		no	Part of the TAQOZ word: save the input stream to a buffer in hub memory starting at address adr, until a line is detected which only contains END		kernel
pre	( -- )	yes	Create a new preemptive word that has the "immediate" attribute set so that it executes immediately at compile time (words like IF ELSE THEN etc)		kernel
pri	( -- )	yes	Create a new word using the next word in the input stream as the name; It is marked private so a RECLAIM can remove them from the dictionary		kernel

			when required. The private words will still function, but can no longer be used to make new words. Keeps the dictionary free of unnecessary detail		
private	( -- )	yes	All following variables and constants can be removed from the dictionary using RECLAIM. The private words will still function, but can no longer be used to make new words. Keeps the dictionary free of unnecessary detail		kernel
pub	( -- )	yes	Create new word as public using the next word in the input stream as the name – the name will remain always in the dictionary		kernel
public	( -- )	yes	All following variables and constants will remain always in the dictionary		kernel
SETORG	( adr -- )	no	Code compilation is set to continue from adr		kernel
TAQOZ	( -- )	no	TAQOZ marks the start of a block of source code to be input silently or 'block loaded', until the END word is encountered. All source between TAQOZ and END is transferred into a hub memory buffer (currently starting at hub address \$60000)). The terminal input is switched temporarily to read all of that buffer, which speeds up loading source code		kernel
uhere	( -- ptr )	no	pointer to compilation area (overwrites VM image)		kernel

### ALIAS bug fix

ALIAS and CPA! do not always work. This is a bug fix which can eliminate CPA! As it is not used elsewhere:-	<pre> pre ALIAS   [C] NFA' [G] ?DUP   IF ( nfa ) ---  read cfa then create new header &amp; store cfa in its cpa       DUP C@ \$E0 AND SWAP CFA ( atrs cfa )       [C] CREATE\$ ---  blend old atrs in with new count (also patch 230309)       OVER @WORDS C@ OR @WORDS C! ---  then store the CFA into the new CPA as 2 or 3 bytes (atr.5)       @WORDS CPA  ROT \$20 AND IF M! ELSE W! THEN       ELSE ERROR       THEN ; </pre>
---	---

### [ ] example

Some computation is needed during the compilation of a word. [ switches Taqoz to interpret the input stream. Notice the GRAB word to cause what's in the brackets to run before the ] bracket switches Taqoz back to compilation. Don't use the alias [G] as it isn't preemptive and doesn't work.	<pre> pub TEST CRLF CRLF CRLF ." Executes when TEST is run " [ CRLF CRLF CRLF ." Executes when TEST compiles " GRAB ] ; </pre>
--	--

### [W] example

Put the code field address of a word on the data stack ... Use [W] to compile that word into a new definition ... Now check the result ...  .S has been successfully included in the new word. This is a trivial example, but could be more useful when the code field address has been the result of a calculation.	<pre> '.S pub Q [W] CRLF ; SEE Q then returns on the terminal:- 1C6EF: pub Q 08F72: 2A74   .S 08F74: 20AF    CRLF ;       ( 4 bytes ) </pre>
--	--

[This article 'Code is contiguous'](#) discusses how memory is saved in TAQOZ in much the same way as in assembly language.  
[This article 'Interactive compilation'](#) explains why TAQOZ compiles words by word instead of the more usual line by line.

## DEFINING CONSTANTS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
:=	(n – )	yes	Create a long constant using the name next in the input stream		kernel
:=!	( n cfa – )	no	Change the value of a long constant created with := e.g. 45 := myconstant 34 ‘ myconstant :=!		kernel
DATCON	(n – )	yes	Same as := to compile a constant but is recognized by FORGET		kernel

## DEFINING VARIABLES

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
!org	( -- )	no	If 'org' has been used to relocate the start address for data storage, !org restores that start address to the saved value		kernel
byte	( -- )	yes	Define new byte variable, using next word in input stream as the name e.g. byte mybyte		kernel

bytes	( n -- )	yes	Define new byte array variable , using next word in input stream as the name e.g. 4 bytes buffer		kernel
long	( -- )	yes	Define new long variable, using next word in input stream as the name e.g. long mylong		kernel
longs	( n -- )	yes	Define new long array variable, using next word in input stream as the name e.g. 4 longs mylongbuffer		kernel
org	( adr -- )	no	save current org value and sets the start address for data storage to adr		kernel
org@	( -- adr )	no	read the current data storage address		kernel
orglen	( – n )	no	n is the total number of bytes of data storage used		kernel
res	( n -- )	yes	reserves n bytes of data memory		kernel
VAR	( -- )	yes	Create a long variable in code space using the next word in the input stream. Variables made in this way are saved with their current values when a system backup is performed		kernel
word	( -- )	yes	Define new word variable, using next word in input stream as the name e.g. word myword		kernel
words	( n -- )	yes	Define new word array variable , using next word in input stream as the name e.g. 4 words mywordbuffer		kernel

**Variable Space** The amount of room available for variables is shown at switch-on in the 'ROOM' section of the 'MEMORY MAP' report. [My system](#) just happens to report 79,801 bytes free, but your system may be different. When that runs low, more space is available above the 'TAQOZ system space' starting at \$20000. Before defining (maybe larger) variables from that address and upwards, use 'org' e.g.-->

**Defining a new type of variable** TAQOZ's word-by-word compilation takes a little getting used to. Here's a definition of a new variable type. Notice the use of [G] to make sure results are on the stack when needed by later preemptive words. [C] is used so that [G] is compiled into the 'double' word, not immediately executed - a bit confusing at first. Also notice the 'double' word is made preemptive so that it executes at compile time as do long, word, byte etc.

```

$20000 := userdata
userdata org
4096 longs myvector1
4096 longs myvector2 ...

--- create a double variable (64 bits )
pre double ( -- )
8 [C] [G] [C] bytes [C] [G] ;
double MYDOUBLE
0. MYDOUBLE D!
```

## DICTIONARY

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.NFA	( nfa -- )	no	Display the name of the word whose NFA is on the stack		extend.fth
@WORDS	( -- addr )	no	Point to start of the dictionary (grows down)		kernel
+DEVICE	( cfa -- )	no			extend.fth
AT	( -- cfa )	yes	More easily seen alias for '		kernel
CFA	( nfa -- cfa )	no	Convert name field address to code field address for a word in the dictionary		kernel
CFA>NFA	( cfa -- nfa )	no	returns the name field address of the word whose code field address is top of stack		extend.fth
CPA	( nfa -- cpa )	no	given the name field address (nfa) of a word, return it's cpa. The code pointer address (cpa) is used to store the cfa of the word (the cfa is where the word definition starts)		kernel
FIND\$	( str dict -- nfaptr   false )	no	Find str word in dictionary, returning name field address, else if not found return false		kernel
flags	( -- ptr )	no	Pointer to long that stores flags: echo,linenums,ipmode,leadspaces,prset,striplf,sign,comp, Defining		kernel
FORGET	( -- )	yes	Takes the next word in the input stream, and forgets all names in the current dictionary from this name onwards - from the most recent instance of the name. Also resets code pointers to the code pointer of the word.		kernel
NFA'	( – nfaptr )	yes	Takes next word in input stream as a name and returns the name field address		kernel
NFA+	( nfa -- nfa2 )	no	Traverse from NFA to next NFA but allow for extended CPA		kernel
RECLAIM		yes	Words that are only useful internally to a module clutter up the dictionary needlessly. It's useful to hide these. All words entries marked private in the dictionary are stripped out. The words can no longer be part of future word definitions, although they continue to function for past definitions		extend.fth
SEARCH	( str -- nfaptr )	no	Search dictionary for string str, returning name field address		kernel
STRIP	( – )	yes	Strip a single header from the dictionary, whose name is next in the input stream e.g. STRIP MYPRIVATEWORD. Used internally by RECLAIM		extend.fth
STRIPIT	( nameptr -- nameptr+ )	no	Strip a single header from the dictionary, whose name pointer is on the stack. Exits pointing to the next word in the dictionary. Used internally by RECLAIM		extend.fth
uwords	( -- ptr )	no	pointer to start of dictionary (builds down)		kernel

This article is a very useful description of the structure of the TAQOZ [Dictionary](#) and the way code is laid down. A [dictionary listing is show here](#).



ERRORS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
ERROR		no	count errors and force a new line to display error		kernel

FLASH MEMORY

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.SF	( -- )	no	Displays flash memory manufacturers' I.D. serial number etc. Useful for checking flash memory communication is working		kernel
*SPIFLASH*	( -- )	no	Serial Flash header - marker for use with FORGET		kernel
SF	( -- )	no	Select SPI Flash as the source for DUMP		kernel
SF?	( - n )	no	Read flash status		kernel
SF@	( adr - n )	no	Returns long from flash address adr		kernel
SFBU	( -- )	no	BACKUP the first 128K of memory into flash		kernel
SFBYTES		no	Reads top bit of flash Jedec ID		extend.fth
SFC@	( adr -- c )	no	Returns byte from flash address adr		kernel
SFCMD		no	Send flash command		kernel
SFER	( adr -- )	no			kernel
SFER32	( adr -- )	no			kernel
SFER4	( adr -- )	no			kernel
SFER64	( adr -- )	no			kernel
SFERP	( src dst -- )	no			kernel
SFJID	( - n )	no	Read serial Flash Jedec ID		kernel
SFPINS	( &cs.so.si.ck -- )	no	Set which pins are used by Serial Flash		kernel
SFRDS	( sfadr dst cnt -- )	no	read block from SF to RAM		kernel
SFRE		no	RESTORE TAQOZ from FLASH by copying to \$2.0000 first		kernel
SFSID	( -- n )	no	Read serial Flash serial number		kernel
SFW@	( adr - w )	no	Returns word from flash address adr		kernel
SFWD		no			kernel
SFWE		no	Write enable flash		kernel
SFWRP	( src dst -- )	no			kernel
SFWRS	( hubsrc sfdst cnt -- )	no	write block from RAM to SF		kernel

INPUT / OUTPUT & SMARTPIN BASICS

Smartpins can be read and written using the words below, or by directly reading and writing the [‘cog ram’ registers](#) using words COG@ and COG! .

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
keypoll	( - n )	no	Relative address to the keypoll register	72	kernel
@PIN	( -- pin )	no	Read the current selected smart pin back		kernel
AKPIN	( -- )	no	Acknowledge pin S/#	24	kernel
F	( - )	no	Float the current pin	24	kernel
FLOAT	( pin - )	no	Set ‘pin’ output floating	48	kernel
H	( - )	no	Set current pin high e.g. 57 PIN H	24	kernel
HIGH	( pin - )	no	Set ‘pin’ output high e.g. : PINTEST BEGIN 10 HIGH 10 LOW AGAIN ; At 200MHz cpu clock, pin 10 goes high for 485nS and low for 675nS	48	kernel
IO?	( pin -- mode )	no	Return the i/o mode the pin is currently set to		extend.fth
L	( - )	no	Set current pin low e.g. 57 PIN H L H L	24	kernel
LOW	( pin - )	no	Set ‘pin’ output low	48	kernel
MUTE	( pin -- )	no	Mute smartpin functions on current pin (pins)		extend.fth
PIN	( pin - )	no	Define the current I/O pin (0-63)	296	extend.fth
PIN!	( flag pin -- )	no	output flag at pin	64	kernel
PIN@	( pin - flag )	no	read pin state	32	kernel
PINS	( from for -- )	no	Create a pin code range for Rev B pins instructions		extend.fth
POLLEDG	( -- flg )	no	Get SE1 event flag to stack, then clear it. flag = TRUE (-1) if edge occurred, else FALSE (0). See also SETEDG	72	kernel
PU?	( pin -- flg )	no	flg = true if there's a pullup attached to pin	5240	extend.fth
R	( - flag )	no	Read the state of the current pin as an input	56	kernel
RDPIN	( -- res )	no	Get smart pin S/# result Z into D, ack	48	kernel
RDPINC	( - res )	no	Get smart pin S/# result Z into D, flag into C, ack	56	kernel
RQPIN	( -- res )	no	Get smart pin S/# result Z into D, flag into C, don't ack	48	kernel
SETEDG	( edge pin -- )	no	Set SE1 event configuration, edge: 1=+ve edge 2=-ve edge, 3=+ve or -ve edge on 'pin' (0-63). See also POLLEDG	80	kernel
SETCT1	( delta -- )	no	Calculate and set the cnt delta and waitcnt		kernel
SETDACS	( n1 - )	no	DAC3 = n1[31:24], DAC2 = n2[23:16], DAC1 = n3[15:8], DAC0 = n4[7:0].	56	kernel



T	( – )	no	Toggle the current pin ( set to the opposite state )	24	kernel
WAITCT1	( -- )	no	Wait for CT1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.		kernel
WAITPIN	( -- )	no	wait for smartpin ack e.g. 48 PIN --- P48 is the target pin 12000 6000 HILO --- define the high and low periods 50 LOW --- P50 is driven low to signal code start 16 PULSES --- generate 16 pulses of set timing WAITPIN --- wait for smartpin acknowledgement 3 PULSES --- 3 more pulses, without any changes 50 HIGH --- P50 set high to indicate code complete		kernel
WAITX	( n -- )	no	Wait for 2n processor cycles (as measured by LAP / .LAP)		kernel
WRACK	( data -- )	no	Write current smartpin data and wait for empty then ack		kernel
WRPIN	( dst -- )	no	Set current smart pin mode to D/#, acknowledge pin	48	kernel
WSTX	( adr cnt -- )	no	output the data buffer at adr for cnt bytes to a WS2812 based Neopixel display		kernel
WXPIN	( dst -- )	no	Set current smart pin S/# parameter X to D/#, ack	48	kernel
WYPIN	( dst -- )	no	Set current smart pin S/# parameter Y to D/#, ack	48	kernel

## I2C

I2C programming hints are given in this [TAQOZ Bytes document](#)

The default i2c bus appears on pin P56 providing clock and pin P57 for data. These pin assignments can be changed, using the I2CPINS word.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
!I2C		no	Initialize the I2C bus pins with pullups and default to 400kHz		extend.fth
<I2C>	( -- )	no	alias for I2C.RESTART		extend.fth
I2C!	( data -- )	no	Write the data to the bus and count the ack in variable acks		extend.fth
I2C?	( -- flg )	no	test for I2C bus by checking for pullups		extend.fth
I2C.KHZ	( khz -- )	no	Set the bus frequency in kHz		extend.fth
I2C.RD	( ack -- data )	no	Generate a start and send the address as a read		kernel
I2C.RESTART		no	Restart without checking busy		kernel
I2C.START		no	Generate a start condition (or a restart)		kernel
I2C.START		no	Generate a start condition (or a restart) redefined		extend.fth
I2C.STOP		no	Generate a stop condition		kernel
I2C.WR	( data -- ack )	no	Generate a start and send the address as a write		kernel
I2C@		no	Read and ack the data from the bus		extend.fth
I2C>	( -- )	no	alias for I2C.STOP		extend.fth
I2CC@		no	i2c DUMP support word		extend.fth
i2cdev	( – n )	no	default i2c address - initialised with \$A4, the RV-3028 real time clock address		extend.fth
I2CPINS	( scl sda -- )	no	Select the I/O to be used as the I2C bus (and initialize to a stop)		extend.fth
I2CRD	( adr -- )	no	Same as I2CRD?, ack is discarded		extend.fth
I2CRD?	( adr -- ack )	no	Generate a start and send the address to prepare for a data read, return with ack		extend.fth
I2CRDREG	( dev reg -- )	no	Select register 'reg' on i2c device 'dev'		extend.fth
I2CWR	( adr -- )	no	Generate a start and send the address to prepare for data write		extend.fth
nakI2C@		no	Read and nak the data from the bus (last byte)		extend.fth

## INITIALISATION

**Up to 8 user words** can be added to the tasks that TAQOZ performs on start-up.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-CODE		no			extend.fth
-INIT	( 'code -- )	no	Remove the word whose cpa  is top of stack to the user INIT list to execute when TAQOZ starts		extend.fth
!INITS	( -- )	no	Init and clear all polling vectors		extend.fth
+CODE	( tbl entries -- )	no			extend.fth
+INIT	( 'code -- )	no	Add the word whose cpa is top of stack to the user INIT list to execute when TAQOZ starts		extend.fth
INITS	( -- )	no	A word initialized at startup with 8 NOPs terminated with EXIT. The NOPs can be overwritten by +INIT and -INIT		extend.fth

## LOGIC

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
AND	( n1 n2 – n3 )	no	n3 = n1 and n2	48	kernel
ANDN	( n1 n2 – n3 )	no	n3 = n1 nand n2	48	kernel
NOT	( n1 – n2 )	no	n2 = inverse of n1	32	kernel

OR	( n1 n2 – n3 )	no	n3 = n1 or n2	48	kernel
XOR	( n1 n2 – n3 )	no	n3 = n1 xor n2	48	kernel

## LOOPING

N.B. All loops in TAQOZ are made with **relative jumps**. If code within a loop is too large then your new word may not work properly and Taqoz may even crash. If you suspect this, use SEE to decompile the new word and check the jump distances. All jumps have to be less than 127 words in scope. Thanks for this tip, Christof Eb.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?LEAVE	( flag – )	no	If flag is true, set the index to limit-1 so that it will LEAVE the loop at the next LOOP or +LOOP executed		kernel
?NEXT	( flag – )	no	Exit FOR NEXT loop early if flag is true (non-zero) e.g. : TEST 1000000 FOR KEY ?NEXT ;		kernel
+LOOP	( incr – )	no	e.g.0 8 ADO I . 2 +LOOP produces 0246		kernel
ADO	( start count – )	no	e.g. 0 8 ADO I . LOOP produces 01234567		kernel
AGAIN		yes	part of BEGIN .. AGAIN structure		kernel
BEGIN		yes	part of BEGIN .. AGAIN, BEGIN .. <condition> UNTIL structures		kernel
DO	( limit start – )	no	e.g. 8 0 DO I . LOOP produces 01234567		kernel
FOR	( cnt -- )	no	e.g. 3 FOR ." N" I . SPACE NEXT --- N3 N2 N1		kernel
I	( – n1 )	no	I returns the current loop index		kernel
I+	( n -- n+I )	no	fast index offset i.e. table I+		kernel
IC@	( – c )	no	Fetch a byte using the current loop index I as the address		kernel
J	( – n1 )	no	J returns the index for the next outer loop		kernel
LEAVE	( – )	no	Set the index to limit-1 so that it will LEAVE the loop at the next LOOP or +LOOP executed		kernel
LOOP	( – )	no	Increment the index and loop if not at limit yet, continue to following code		kernel
NEXT		no	e.g. 3 FOR ." N" I . SPACE NEXT --- N3 N2 N1		kernel
REPEAT		yes	part of BEGIN .. <condition> WHILE .. REPEAT structure		kernel
UNLOOP		no	Discard current DO index and limit and branch address e.g. : TEST 1000000 0 DO KEY IF UNLOOP EXIT THEN LOOP ;		kernel
UNTIL		yes	part of BEGIN .. <condition> UNTIL structure		kernel
WHILE		yes	part of BEGIN .. <condition> WHILE .. REPEAT structure		kernel

[Here's an article 'Loops never return'](#) that discusses looping with TAQOZ.

## MASKING

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
>!		no	replace the inline 10-bit literal with a new value		extend.fth
>	( bitpos – bitmask )	no	Encode - bitpos = the position of the msb set in bitmask e.g. %0100 >  . would print 2	24	kernel
>9	( n – 9bits )	no	mask n to 9 least significant bits	32	kernel
>B	( n – byte )	no	mask n to a byte	32	kernel
>N	( n -- nibble )	no	mask n to a nibble ( 4 bits )	32	kernel
>W	( n – word )	no	mask n to 16 least significant bits	32	kernel
<	( bitpos – bitmask )	no	Decode - bitmask = long with one bit set at bitpos position e.g. 3  < .BIN would print %1000	24	kernel
1&	( n1 – n2 )	no	n2 = n1 and 1	24	kernel
BITS	( n1 bits -- n2 )	no	Masking off ‘bits’ number of least sig bits	56	kernel
BMASK	( s -- mask )	no	Get LSB-justified bit mask of size (S[4:0] + 1) into D. D = (\$0000_0002 << S[4:0]) - 1.	23	kernel
MOVBYTS	( D S – n2 )	no	n2 = Move bytes within D, per S. D = {D.BYTE[S[7:6]], D.BYTE[S[5:4]], D.BYTE[S[3:2]], D.BYTE[S[1:0]]}.	64	kernel
SIGN	( val bitpos -- sval )	no	Extend sign from bitpos (\$FC 7 sign .L --- \$FFFF_FFFC)	48	kernel

# MATHS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-	( n1 n2 – n3 )	no	n3 = n1 - n2	56	kernel
-NEGATE	( n1 sn -- n1   -n1 )	no	negate n1 if the sign of sn is negative (used in signed divide op)	56 if negated, else 48	kernel
?NEGATE	( n1 flg -- n2 )	no	negate n1 if flg is true	56 if negated, else 48	kernel
*	( n1 n2 – n3 )	no	n3 = n1 * n2	128	kernel
*/	( n1 n2 n3 -- res )	no	res = (n1 * n2 ) / n3 uses a 64 bit intermediate result for precision (cordic)	Around 216 data dependent	kernel
/	( n1 n2 – n3 )	no	n3 = n1 / n2	Around 470 data dependent	kernel
//	( n1 mod -- rem )	no	n1 / mod, leaving just the remainder	Around 120 data dependent	kernel
+	( n1 n2 – n3 )	no	n3 = n1 + n2	48	kernel
1-	( n1 – n2 )	no	n2 = n1 - 1	24	kernel
1+	( n1 – n2 )	no	n2 = n1 + 1	24	kernel
2-	( n1 – n2 )	no	n2 = n1 - 2	24	kernel
2+	( n1 – n2 )	no	n2 = n1 + 2	24	kernel
4+	( n1 – n2 )	no	n2 = n1 + 4	24	kernel
ABS	( n1 – n2 )	no	n2 = absolute value of n1	24	kernel
BIN	( -- )	no	Set the default number radix to binary		kernel
DEC	( -- )	no	Set the default number radix to decimal		kernel
DM*	( d. u -- d. )	no	Multiply signed double by unsigned long to give signed double result	Around 600 data dependent	kernel
EXP	( e1 – n2 )	no	n2 (long) = 2 <sup>e1</sup> (5 bits whole exponent, 27 bits fractional)	112	kernel
GETQY	( – n1 )	no	retrieve cordic solver Y result	64	kernel
GETRND	( – n1 )	no	return a random number – method 2	64	kernel
HEX	( -- )	no	Set the default number radix to hexadecimal		kernel
KB	( n1 – n2 )	no	n2 = n1 * 1024	168	kernel
LOG	( n1 – e2 )	no	e2 (5 bits whole exponent, 27 bits fractional) = binary log (n1) (cordic)	96	kernel
LW*	( mclong mpword -- result. )	no	Multiply a long by 16-bits and return 48-bits	Around 64 data dependent	kernel
M	( n1 – n2 )	no	n2 = n1 * 1000000	288	kernel
MAX	( u1 u2 – u3 )	no	u3 = unsigned maximum of u1, u2	48	kernel
MAXS	( n1 n2 – n3 )	no	n3 = signed maximum of n1, n2	48	kernel
MB	( n1 – n2 )	no	n2 = n1 * 1048576	328	kernel
MIN	( u1 u2 – u3 )	no	u3 = unsigned minimum of u1, u2	48	kernel
MINS	( n1 n2 – n3 )	no	n3 = signed minimum of n1, n2	48	kernel
n!	( n -- fac. )	no	Returns n factorial as a double		extend.fth
NEGATE	( n1 – n2 )	no	n2 = -n1	24	kernel
QFRAC	( – n1 )	no	retrieve cordic solver X result	120	kernel
QROTATE	( x y z -- x y )	no	rotate x and y by z (cordic)	120	kernel
QVECTOR	( x y -- len th )	no	convert x,y to amplitude len, angle th (cordic)	104	kernel
RND	( – n1 )	no	return a random number – method 1	72	kernel
SQRT	( d. -- sqrt )	no	sqrt = the square root of double d	120	kernel
U*/	( u1 u2 div1 -- res )	no	Multiply u1 by u2 and divide the double product by div1	200	kernel
U/	( u1 u2 – u3 )	no	u3 = u1 / u2	128	kernel
U//	( dvdn dvsr -- rem quot )	no	quot = dvdn / dvsr, with rem remainder	112	kernel
UM*	( u1 u2 -- ud. )	no	unsigned 32bit * 32bit multiply -- 64bit result	Around 112 data dependent	kernel
UM/	( d. dvsr -- res )	no	divide double d. by long dvsr. long res is the result	136	kernel
UM//	( d. dvsr -- rem quot )	no	divide double d. by long dvsr, double quot is result with long rem as remainder	Around 832 data dependent	kernel
W*	( w1 w2 – n1 )	no	n3 = w1 * w2	48	kernel

# MEMORY

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
--	( adr – )	no	long at hub address adr is decremented	256	kernel
!	( n addr -- )	no	store n to long at hub memory addr	64	kernel
@	( addr -- long )	no	Fetch a long from hub memory	32	kernel
+!	( n addr -- )	no	add n to long at hub memory addr	80	kernel
++	( adr – )	no	long at hub address adr is incremented	224	kernel
+~!	( adr -- )	no	negate long at adr		
<CMOVE	( src dst cnt -- )	no	Same as CMOVE, in reverse address order (Use one or the other when the two blocks overlap each other)		kernel
~	( adr – )	no	long at hub address adr is zeroed	72	kernel
~~	( adr – )	no	long at hub address adr is set all 1's	72	kernel
BIT!	( mask addr flag -- )	no	will take a flag and either SET or CLR bits in memory e.g. \$FF MYVAR 0 BIT! Would zero the bottom 8 bits of MYVAR \$FF MYVAR 1 BIT! Would set the bottom 8 bits of MYVAR		kernel
C--	( cadr – )	no	byte at hub address cadr is decremented	256	kernel
C!	( n caddr -- )	no	store n to byte at hub memory addr	64	kernel
C@	( caddr -- byte )	no	Fetch a byte from hub memory	32	kernel
C@++	( caddr -- caddr+1 byte )	no	fetch byte character from hub memory and increment address	56	kernel
C+!	( n caddr -- )	no	add n to byte at hub memory addr	80	kernel
C++	( cadr – )	no	byte at hub address cadr is incremented	216	kernel
C~	( adr – )	no	byte at hub address adr is zeroed	72	kernel
C~~	( adr – )	no	byte at hub address adr is set all 1's	72	kernel
CELL+	( n1 -- n2 )	no	n2 = n1 +4 advance n1 by the 4 addressable bytes in a long		extend.fth
CLR	( mask addr -- )	no	Clear bit(s) in hub memory long	104	kernel
CMOVE	( src dst cnt -- )	no	Move hub memory bytes starting from source to destination by cnt bytes		kernel
COG!	(n adr – )	no	Write long to COG memory adr	64	kernel
COG@	( adr – n )	no	Read long at COG memory adr	32	kernel
D!	( d. addr -- )	no	store n to double at hub memory addr	288	kernel
D@	( addr -- d. )	no	Fetch double from hub memory addr	248	kernel
DUPC@	( caddr – caddr data )	no	Fetch a byte at addr in hub memory, leave addr on stack	80	kernel
ERASE	( adr bytes -- )	no	erase bytes starting at adr		kernel
FILL	( addr cnt fillch -- )	no	fill cnt byteswith fillch starting from address addr		kernel
GO:	( index size -- )	no			extend.fth
HERE:		no			extend.fth
INDEX:	( index -- )	no	return with pointer to list of parameters etc following this word		extend.fth
LERASE	( src cnt -- )	no	erase cnt longs starting at src		kernel
LMOVE	( src dst longs -- )	no	Very fast block move uses SETQ2 and LUT to move chunks 64KB in 37,992 cycles= 151,968ns @250MHz		kernel
LUT!	( n adr – )	no	Write long to LUT memory adr	64	kernel
LUT@	( adr – n )	no	Read long at LUT memory adr	24	kernel
M!		no	works just like ! does except it skips leading zero bytes. So:- \$12345678 avar ! would store 4 bytes but \$C0 avar M! will only write to the first byte leaving \$123456C0		kernel
RAM	( -- )	no	modifier to dump from RAM e.g. 0 100 RAM DUMP		kernel
SET	( mask addr -- )	no	Set bit(s) in hub memory long	104	kernel
SET?	( mask addr -- flg )	no	Test single bit of a long in hub memory	88	kernel
TABLE	( bytes -- ) ( -- addr )	yes	create a table of preallocated bytes - must be an even number		extend.fth
W--	( wadr – )	no	word at hub address wadr is decremented	256	kernel
W!	( n waddr -- )	no	store n to word at hub memory addr	64	kernel
W@	( waddr -- word )	no	Fetch a word from hub memory	32	kernel
W+!	( n waddr -- )	no	add n to word at hub memory addr	80	kernel
W++	( wadr – )	no	word at hub address wadr is incremented	224	kernel
W~	( adr – )	no	word at hub address adr is zeroed	72	kernel
W~~	( adr – )	no	word at hub address adr is set all 1's	72	kernel

NUMBER I/O

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.	( n -- )	no	print n on the current output stream	up to 7528	kernel
.ADDR	( n -- )	no	Display n as an address e.g. 00017:		kernel
.ADRX	( n – )	no	Display n as an address e.g. 00017		kernel
.AS	( str -- )	no	Like AS" but expects a string ptr on the stack e.g. " #.###ms" .AS		kernel
.AS"	( n -- )	yes	Display n with format as per string spec: # Convert one digit (default is decimal) ~ Toggle leading zero suppression \ pad leading zeros with spaces \$  Hexadecimal *  Convert all remaining digits 4  Convert 4 digits Terminated with " e.g. .AS" #.###ms"		kernel
.ASBYTES	( n -- )	no	Print n as Display n as decimal "##,###,###,#### bytes"		extend.fth
.B	( n – )	no	Display n as two digit hex byte 00-FF		kernel
.BIN	( n – )	no	Display n in binary with leading %		kernel
.BITS	( n bits -- )	no	e.g. \$EC04 11 .BITS prints 0 0 1 0 0 0 0 0 0 1		extend.fth
.BYTE	( n – )	no	Display n as two digit hex byte 00-FF		kernel
.BYTES	( adr bytes -- )	no	e.g. 0 6 .BYTES prints 90 14 80 FD 50 32		extend.fth
.DEC	( n -- )	no	Display n in decimal with at least a single digit		kernel
.DEC2	( val – )	no	print unsigned val as two digits		extend.fth
.DECL	( n -- )	no	Display n as decimal "##,###,###,####"	up to 108424	kernel
.DECS	( val digits -- )	no	print val right justified in a field of digits chars, leading spaces		extend.fth
.DP	( num left-digits -- )	no			extend.fth
.H	( n -- )	no	Display n as hex nibble 0-F		kernel
.HERE	( -- )	no	Print code pointer prompt		extend.fth
.L	( n -- )	no	Display n in hex as long with leading \$ and _ in the middle		kernel
.LONG	( n – )	no	Display n in hex as long with _ in the middle		kernel
.M	( n -- )	no	print n as 5 hex digits		extend.fth
.MEM	( bytes -- )	no	print bytes followed by B, KB, MB or GB		extend.fth
.UBYTES	( n1 n2 – )	no	Prints n1 in hex n2 in decimal followed by 'bytes'		extend.fth
.W	( n -- )	no	Display n in hex as a word with leading \$		kernel
.WORD	( n – )	no	Display n in hex as a word		kernel
#		no	Insert the next digit of the number being displayed into the formatted number string		kernel
#>	( -- str )	no	Terminate the formatted number leaving the address of the formatted number string at tos, ready for display		kernel
#S	( -- )	no	Insert all remaining significant digits of the number		kernel
<#	( -- )	no	Start a new formatted number string		kernel
<D>	( d1 -- n1 )	no	Store high long of double for formatting		kernel
D.	( d. – )	no	print double number		extend.fth
D.DECS	( dval. digits -- )	no	print a double number right justified		extend.fth
D.R	( dval. digits -- )	no	print double dval in digits field, right justified, leading spaces		extend.fth
HOLD	( c -- )	no	Insert the character on the stack into the formatted string		kernel
PRINT	( n -- )	no	more easily seen alias for .	up to 7528	kernel
U.	( u1 – )	no	Print an unsigned number		kernel
U.N	( val digits -- )	no	print unsigned val in digits field, right justified, leading zeros		extend.fth
U.R	( val digits – )	no	print unsigned val in digits field, right justified, leading spaces		extend.fth

P2 CLOCK

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
!clkfreq	( -- )	no	Up to 8 words may be stored here to be performed as part of RCSLOW or RCFAST		extend.fth
.CLK	( -- )	no	An alias for .clkfreq		extend.fth
+PPPP	( 2..30 -- )	no	PPPP: Divide the PLL by 2 4 6 ... 30 for the system clock when SS = %11 --- 1 = 31 = no divide		extend.fth
+SS		no	clock source		extend.fth
clkfreq!	( Hz – )	no	Set the P2 clock to Hz, adjusting other P2 parameters to maintain the system e.g. maintaining the required terminal baud rate. Can be		extend.fth



			overclocked to around 300MHz it seems		
CLKSET		no	PLL should run between 100 – 200MHz		extend.fth
CLKSRC		no	An alias for +SS		extend.fth
PLLDIV	( 2..30 -- )	no	An alias for +PPPP		extend.fth
PLLEN	( -- )	no	Enable PLL		extend.fth
PLLOFF	( -- )	no	Disable PLL		extend.fth
RCFAST	( -- )	no	Set the P2 clock to around 25.1MHz 961600 terminal baud rate continues to work		extend.fth
RCSLOW	( -- )	no	Set the P2 clock to around about 20kHz – fast terminal baud rates are not maintained		extend.fth
USEPLL		no			extend.fth
USEXTAL		no			extend.fth
VCOMUL	( 1..1024 -- )	no			extend.fth
XIDIV	( 1..64 -- )	no			extend.fth

[This article 'Set TAQOZ clock frequency'](#) is useful.

## POLLING

Background polling of user tasks can be initiated 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 start-up routine as all polls are cleared on boot. To keep TAQOZ responsive, make all user POLLS as short as possible.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-POLL	( 'code -- )	no	Remove a task cpa from the polled list	968	extend.fth
!POLLS	( -- )	no	Init and clear all 8 polling vectors	680	extend.fth
+POLL	( 'code -- )	no	Add a task cpa to the polled list	1424	extend.fth
POLL:		no	Background Polling low priority background functions are checked when console input is idle Up to 8 vectors are polled as part of POLLS		extend.fth

## REAL TIME CLOCK

These words are used to communicate with an RV-3028 real time clock via i2c if one is connected to the P2.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
_RTC!	( byte reg -- )	no			extend.fth
!PW	( -- )	no	rtc password init		extend.fth
!RTC		no	Initialise real time clock		extend.fth
!TIME		no	boot time inits		extend.fth
.DATE		no			extend.fth
.DAY		no			extend.fth
.DT		no			extend.fth
.FDT		no			extend.fth
.TIME		no			extend.fth
@date		no	byte variable, rtc buffer		extend.fth
@day		no	byte variable, rtc buffer		extend.fth
@hour		no	byte variable, rtc buffer		extend.fth
@min		no	byte variable, rtc buffer		extend.fth
@month		no	byte variable, rtc buffer		extend.fth
@sec		no	byte variable, rtc buffer		extend.fth
@year		no	byte variable, rtc buffer		extend.fth
>HMS	( s m h -- hhmmss )	no			extend.fth
BCD>DEC	( bcds -- val )	no			extend.fth
BCDS	( bcds -- dec rem )	no			extend.fth
bdate		no	long variable, date at reset		extend.fth
bms		no	long variable, ms time at reset		extend.fth
btime		no	long variable, time at reset		extend.fth
DATE!	( yymmdd -- )	no			extend.fth
DATE@		no			extend.fth
DAY!		no			extend.fth
DAY@		no			extend.fth
HMS	( #xyyyzz -- zz yy xx )	no	split 6 digit decimal number into 3 two digit groups		extend.fth
I2CREG!	( dat reg dev -- )	no	store 8-bit to i2c device register		extend.fth
I2CREG@	( reg dev -- dat )	no	fetch reg from 8-bit reg		extend.fth
PW	( flag – )	no	turn pw protect on or off		extend.fth

QDATE\$	( --- yymmdd )	no	use bdate but check if qtime has gone into the next day (if so then refresh bdate )		extend.fth
QTIME@	( --- hhmmss )	no	Quick TIME@ (10us) using the boot time plus the curent GETMS		extend.fth
RDRTC		no	RV-3028 @\$A4 can latch 7 bytes for I2C read		extend.fth
RTC!		no			extend.fth
RTC?	( – flag )	no	returns whether a real time clock is present on address \$A4		extend.fth
RTC@	( reg -- byte )	no			extend.fth
RTCB!	( byte reg -- )	no	store byte as BCD in RTC register		extend.fth
SECS@	( -- seconds )	no	return with seconds of the day		extend.fth
TIME!	( hhmmss -- )	no			extend.fth
TIME@		no			extend.fth
UTIME!	( secs -- )	no			extend.fth
UTIME@	( -- secs )	no			extend.fth

## REVERSAL of BITS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
CREV	( b1 -- b2 )	no	Reverse byte 0..7 → 7..0	32	kernel
REV	( n1 -- n2 )	no	Reverse bits 0..31 → 31..0	32	kernel

Why these obscure instructions? Bit reversal of data buffer addresses is an essential element of the fast fourier transform - a fundamental function of digital signal processing. The assembly instruction these TAQOZ words are based on saves a great deal of time over other means.

## SD CARD

An SD card, if fitted to the P2, offers huge, removable data storage as compared to relatively small, fixed-in-place Flash memory. Note that the SD card driver:-

- assumes that all files stored on SD card are unfragmented, which (from experimentation with Windows and Linux) seems always to be the case
- File delete is currently not supported
- Files are created with a fixed size
- Only one file is opened at a time

With those limitations in mind, the driver benefits from being simple to use, small, fast and P.C. compatible.

The SD card default connections are:-

Pin P58        SI  
Pin P59        SO  
Pin P60        CK  
Pin P61        CS

These may be changed using the SDPINS word.

The following sections list the Basic through to High-level SD card words:-

### SD card - Data Buffers

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
DIRBUF	( – n )	no	const, dir sector buffer		file.fth
SDBUF	( – n )	no	const, file sector buffers initial value		file.fth

### SD card - Pin-out

These words relate to the Smart Pins used to interface to the SD card:-

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
SDCK	( – ck )	no	pin number (0-63) assigned for CK signal, default is pin 60		file.fth
SDCS	( – cs )	no	pin number (0-63) assigned for CS signal, default is pin 61		file.fth
SDDI	( – di )	no	pin number (0-63) assigned for DI signal, default is pin 59		file.fth
SDDO	( – do )	no	pin number (0-63) assigned for DO signal, default is pin 58		file.fth
SDPINS	( n – )	no	set the four pins used for sd card interface e.g. &60.58.59.61 SDPINS		file.fth

### SD card - SPI commands

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
!SD	( --- ocr   false )	no	Initialise the SD card in SPI mode and return with the OCR		file.fth
!SDIO		no	release SPI I/O to SD		file.fth
ACMD	( data acmd -- res )	no	send SD command		file.fth
CMD	( data cmd -- res )	no	send SD command	2777	file.fth
SD?	( – flg )	no	flg=TRUE if SD card present, else flg=FALSE		file.fth

SD card - Register status report

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.CID	( – )	no	Print the CID information in verbose report format e.g. CARD: SANDISK SD SD32G REV\$85 #BEC162CE DATE:7E5/1		file.fth
.CSD	( – )	no	Print the Card Specific Data information in verbose report format		file.fth
.OCR	( – )	no	Print card Operating Conditions Register in verbose report format		file.fth

SD card - File permissions

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
RO	( – )	no	Read Only – files permission		file.fth
RW	( – )	no	Read / Write – files permission		file.fth
RWC	( – )	no	Read / Write and Create – files permission		file.fth
RWS	( – )	no	Read / Write / System – files permission		file.fth

SD card - Sector

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?FLUSH	( – )	no	only flush sector if it has been written to		file.fth
@FILE	( -- sector )	no	return starting sector at current FILE		file.fth
@OPEN	( -- sector )	no			file.fth
FLUSH	( – )	no	flush sector		file.fth
OPEN-SECTOR	( sector -- )	no	Set the starting sector for file access		file.fth
SDRD	( sector dst -- )	no	read sector into memory and update sector number		file.fth
SDRDS	( sector dst bytes -- )	no	read multiple sectors in continuous multiblock mode -- update @sdrd pointer		file.fth
SDRDx	( sector dst -- )	no	read sector into memory silently		file.fth
SDWR	( src sect -- )	no	Write a sector		file.fth
SDWRS	( ram sectr bytes -- )	no	Write multiple sectors		file.fth
SECTOR	( sect -- sdbuf )	no	sector to hub ram		file.fth
SECTORF	( sect -- sdbuf )	no	Always read in a sector but flush first		file.fth

SD card - Virtual Memory operations

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
SD!	( long xaddr -- )	no	store long to SD virtual memory in current file		file.fth
SD@	( xaddr -- long )	no	fetch long from SD virtual memory in current file		file.fth
SDADR	( sdadr -- ramadr )	no	Convert SD file address to hub ram address where file is buffered	897	file.fth
SDC!	( byte xaddr -- )	no	store byte to SD virtual memory in current file		file.fth
SDC@	( xaddr – byte )	no	fetch byte from SD virtual memory in current file		file.fth
SDW@	( xaddr – word )	no	fetch word from SD virtual memory in current file		file.fth

SD card - FAT32

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
@BOOT	( -- sector )	no			file.fth
@CWD	( – n )	no			file.fth
@ROOT	( -- sector )	no			file.fth
CD#	( sect -- )	no	change current working directory		file.fth
CWD		no	open the current working dir as if it were a file		file.fth
CWD!	( n – )	no			file.fth
FAT1		no	Access FAT1 as a file		file.fth
FAT2		no	Access FAT2 as a file		file.fth
MBR		no	Use the MBR as a file		file.fth
ROOT		no	Open the root folder as a file		file.fth

SD card - Directory Record

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
dirrcd	( – adr )	no	points to the start of the 32 byte directory record as follows:-		file.fth
fname	( – adr )	no	8 byte file name in the directory record		file.fth
fname	( – adr )	no	3 byte file extension in the directory record		file.fth
fatr	( – adr )	no	1 byte file attribute 0:read-only, 1:hidden, 2:system, 3:volume label, 4:directory, 5:archive, 6-7: undefined		file.fth
res	( – adr )	no	reserved – set 0		file.fth
fcms	( – adr )	no	1 byte file creation time - milliseconds		file.fth
fctime	( – adr )	no	2 byte file creation time		file.fth
fcdate	( – adr )	no	2 byte file creation date		file.fth
fclsth	( – adr )	no	2 bytes		file.fth
ftime	( – adr )	no	2 bytes file modification time		file.fth
fdate	( – adr )	no	2 bytes file modification date		file.fth
fclstl	( – adr )	no	2 bytes		file.fth
fsize	( – adr )	no	4 bytes file size		file.fth
diridx	( – adr )	no	2 bytes		file.fth
file\$	( – str )	no	16 bytes filename		file.fth
FSIZE@	( – n )	no	fsize @		file.fth

SD card - Clusters

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
!MOUNT	( – )	no	mount the sd card		file.fth
?MOUNT	( – )	no	mount the sd card if not already mounted		file.fth
C>S	( clust# -- sector )	no	Convert Cluster to sector		file.fth
CLUSTERS?	( cluster# -- clusters )	no	count number of clusters allocated from start cluster		file.fth
FCLUSTER	( -- cluster#0 )	no	convert Directory address to first cluster		file.fth
FMAX	( -- bytes )	no	find total allocated cluster bytes for this byte		file.fth
FSECTOR	( -- sector )	no	read Directory cluster and convert to starting sector		file.fth
GETFAT		no	read fat32 as a byte array		file.fth
MOUNT	( – )	no	mount the sd card and display card identity		file.fth
MOUNTED?	( – flg )	no	mount the sd card if not already mounted, flg<>0 if successful		file.fth
SECT>CLST	( sector -- cluster )	no	convert a sector to a cluster ( result 0 = out of range ; 2 = 1st )		file.fth

SD card - Directory

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.FDATE		no	print the file date		file.fth
AS8.3	( str -- )	no	save file name in file\$ but convert to 8.3 format in dir		file.fth
CREATED	( -- )	no	Update the ‘created’ time and date of the current file		file.fth
f\$	( – adr )	no	pointer to 16 bytes, copy of 8.3 name in normal short format		file.fth
FDATE!	( #yymmdd field -- )	no	update file modification/create date in dir buf, Date (7/4/5 bits, for year-since-1980/month/day)		file.fth
FTIME!	( #hhmmss field -- )	no	update file modification/create time in dir buf, Time (5/6/5 bits, for hour/minutes/doubleseconds)		file.fth
GETDIR	( index -- )	no	read the nth directory entry into the dir buffer (index saved in diridx)		file.fth
IDX>DIR	( dirIndex -- diradr )	no	reads relevant dir sector in using index, returns with the address in the buffer		file.fth
MODIFIED	( -- )	no	Update the ‘modified’ time and date of the current file		file.fth
OPENDIR	( – )	no			file.fth
SAVEDIR	( – )	no	update directory on SD		file.fth

SD card - Directory Structure

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.LIST	( – )	no	Display files in current directory one per line, with attributes		file.fth
COPY	( -- )	yes	Using the next word in the input stream as filename, if this is found on disk, copy the file details to the ‘clipboard’ copybuf		file.fth
copybuf	( – adr )	no	32 byte ‘clipboard’ memory		file.fth
DIR	( – )	no	Display files in current directory one per line		file.fth
DIR+	( -- )	no	Display files in current directory one per line with more detail		

DIR++	( – )	no	Display files in current directory one per line with yet more detail		file.fth
DIRW	( – )	no	Display files in current directory in 6 columns		file.fth
DIRX	( n -- )	no	Display files in current directory in 'n' columns		file.fth
FREEDIR	( – )	no	Find the next free directory entry (just looks for a null but could do more) no range checking just to keep it simple for now		file.fth
ls	( – )	yes	list the directory in simple format i.e. ls <enter> or in wide format e.g. ls -l <enter>		file.fth
mk	( size -- )	yes	Create a new file by name ( using next word in input stream ) but if it already exists then delete the old one and reuse the dir entry, if size = 0 then max = 4GB		file.fth
mk\$	( size namestr -- )	no	Create a new file, named 'namestr' allocating 'size' bytes		file.fth
PASTE	( -- )	no	Make a duplicate entry in the Directory using the details saved in the COPY command (N.B. doesn't copy the file contents)		file.fth

### SD card - File Operations

**N.B. All file operations are subject to 'File permissions' see the section above, so before attempting to write to a file, set RW, then afterwards optionally set RO to protect the contents from change. You won't get far otherwise!**

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
(cat)	( sector -- )	no	Display the contents of the file, whose sector is top of stack		file.fth
cat	( <src> – )	yes	Display the contents of the file, whose name follows in the input stream		file.fth
CD		yes	Alias for cwd		file.fth
CD\$	( str -- )	no	change directory to 'str'		file.fth
CLOSE-FILE		no	Close file by flushing, switching to read-only and setting 'readsect' to -1		file.fth
cwd	( <dir> -- )	yes	change directory, using next word in input stream as name		file.fth
DEL-FILE	( -- )	yes	currently does nothing		file.fth
FCLOSE		no	Alias for CLOSE-FILE		file.fth
FCOPY	( <src> <dst> -- )	yes	old file name and new file name both follow in the input stream		file.fth
FCOPY#	( srcsect dstsec bytes -- )	no	copy by by referring to the files' index		file.fth
FGET	( -- ch )	no	Read the next byte from the file		file.fth
FLOAD	( <name> -- )	yes	load file as console input, filename as next word in input stream, then restore on null to console input		file.fth
FLOADS	( sector -- )	no	load file as console input then restore on null to console input		file.fth
FOPEN	( <name> -- )	yes	grab parameters and permissions then open-file		file.fth
FOPEN\$	( str -- sector   0 )	no	Alias for OPEN-FILE\$, returns sector else 0 if it fails		file.fth
FOPEN#	( index -- )	no	open a file using the directory name index		file.fth
FREAD	( sdsrsrc hubdst bytes -- )	no	Read file offset to memory		file.fth
FSEND	( -- )	no	Send all output to the file if open - wraps using the variable fsize ( see section SD card - Directory Record)		file.fth
FWRITE	( hubsrc sddst bytes -- )	no	Write memory to file offset		file.fth
MAKE	( size -- )	no	force file open - create to size if it's not found		file.fth
OPEN-FILE	( <name> -- filesect )	no	Get the file name as the next word in the input stream and try to open it, return with sector		file.fth
OPEN-FILE\$	( str -- sector   0 )	no	check mount and find 8.3 file name, if found then convert directory entry to starting sector, else if create flag set then create a preallocated size file - else fail, open the sector although 0 = fail (mbr sector 0 is protected anyway)		file.fth
opensect	( -- adr )		long variable that stores starting sector of the open file		file.fth
pwd	( -- )	no	display the current directory name		file.fth
QV	( <src> – )	yes	Display the first 128 bytes as hex for the file whose name is next in the input stream		file.fth
readsect	( -- adr )	no	long variable stores the current buffered sector		file.fth
RENAME	( <old> <new> -- )	yes	old file name and new file name both follow in the input stream		file.fth
RENAME#	( dir# <new> -- )	yes	rename by referring to the files' index		file.fth
REOPEN-FILE	( – flg )	no	Like OPEN-FILE\$, but use filename stored in file\$		file.fth
SAVETEXT	( -- )	yes	Save all text that follows in the input stream to the filename, until the ESC char, then close the file. The file size is always 1 Mbyte e.g. RW SAVETEXT MYFILE.TXT Hello from myfile.txt <ESC> N.B If the file is more than one sector in size - the transfer speed of the terminal sending the file has to be limited -Christof Eb reports 1mS delay per character works OK. This delay is needed to give TAQOZ time to write to the SDcard.		file.fth



SD card - Disk Reporting / Formatting

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
*DISK*	( -- )	no	Module marker word – used by FORGET for module removal		file.fth
!FATS	( -- )	no			file.fth
.DISK	( -- )	no	display OCR, CSD, Speeds, MBR and FAT32 information		file.fth
.FAT	( -- )	no	display fat32 boot record		file.fth
.FREE	( -- )	no	display SD card free and used space ( takes a few seconds )		file.fth
.MB	( sectors -- )	no	Fetch sectors, print, and display in MB as well ( 15,515,648 = 7,944MB )		file.fth
.MBR	( -- )		Scan all 4 entries : check active partition type		file.fth
.SD	( -- )	no	Display OCR, CSD and Speeds		file.fth
.SPEEDS	( -- )	no	Display Latency, Sector and Blocks speeds		file.fth
FORMAT	( – )	no	Use xx CLSZ beforehand to set non-default cluster size Use RWS to set system permissions etc Usage: 32 KB CLSZ RWS FORMAT		file.fth
FORMAT.FAT	( -- )	no	Format SD card (takes a while )		file.fth
FORMAT.MBR		no	Format the Master Boot Record of the SD card		file.fth

SD card - Block File display

Originally, Forth programs accessed disk as contiguous memory, splitting the disk up into consecutive 'blocks'. Each block was dimensioned to store just one screen full of text. TAQOZ Reloaded supports the block file system, based on the current file, rather than the whole disk. Of course, the file must exist and be of appropriate size beforehand.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.BLOCK	( blk# -- )	no	List an old-school block within the current file		file.fth
LISTBLOCKS	( -- )	no	List all old-school blocks with the current file		file.fth

SD card - System

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?SD	( -- )	no	Display SD card summary – e.g. SD CARD 31 GB SANDISK SD SD32G REV\$85 #3200344782 DATE:2021/1		file.fth
BACKUP	( <src> – )	yes	Backup to filename ( following in input stream) or FLASH : options: DISABLE, FLASH, MBR, BIX usage examples – see section on Backup at rear of the document		file.fth
BACKUP\$	( str -- )	no	Backup to filename or FLASH : options: DISABLE, FLASH, MBR, BIX usage example: “ MBR” BACKUP\$		file.fth
BOOT	( -- )	no	Reboot Taqoz Reloaded, preserving all unsaved work		file.fth
BU	( -- )	no	Backup to SD MBR shortcut		file.fth

SERIAL RAM

These words enable communication with Serial RAM if one is connected to the P2

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
ISR	( init SPIRAM )	no			extend.fth
.SR	( – )	no	Print the SPIRAM ID		extend.fth
P2PAL		no	Use PSRAM on P2PAL pins 53,49,48 and 52		extend.fth
SPIRW	( -- word )	no			extend.fth
SR	( – )	no	prefix for DUMP to show serial ram e.g. 0 \$20 SR DUMP		extend.fth
SR!	( long adr -- )	no	write long from adr		extend.fth
SR@	( adr -- long )	no	read long from adr		extend.fth
SRC!	( byte adr -- )	no	write byte to adr		extend.fth
SRC@	( adr -- byte )	no	read byte from adr		extend.fth
SRCMD	( cmd -- )	no			extend.fth
SRID	( -- long long )	no			extend.fth
SRRD	( readadr -- )	no			extend.fth
SRW!	( word adr -- )	no	write word to adr		extend.fth
SRW@	( adr -- word )	no	read word from adr		extend.fth
SRWRP	( src dst -- )	no			extend.fth
SRWRS	( hub srdst bytes -- )	no			extend.fth

SHIFTING

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
<<	( n1 bits -- n2 )	no	Shift n1 left by count	48	kernel
>>	( n1 cnt -- n2 )	no	Shift n1 right by count	48	kernel
16<<	( n1 – n2 )	no	n2 = n1 shifted left 16 places	32	kernel
16>>	( n1 – n2 )	no	n2 = n1 shifted right 16 places	32	kernel
2*	( n1 -- n2 )	no	shift n1 left one bit (equiv to multiply by 2)	32	kernel
2/	( u1 -- u1/2 )	no	Shift u1 right by 1 (equiv to divide by 2) – N.B. unsigned arithmetic	32	kernel
4*	( n1 – n2 )	no	n2 = 4 x n1	32	kernel
4/	( n1 – n1/4 )	no	Shift n1 right by 2 (equiv to divide by 4)	32	kernel
8<<	( n1 – n2 )	no	n2 = n1 shifted left 8 places	32	kernel
8>>	( n1 – n2 )	no	n2 = n1 shifted right 8 places	32	kernel
9<<	( n1 – n2 )	no	n2 = n1 shifted left 9 places	32	kernel
9>>	( n1 – n2 )	no	n2 = n1 shifted right 9 places	32	kernel
ROL	( n1 bits -- n2 )	no	rotate n1 by bits leftwise	48	kernel
ROR	( n1 cnt -- n1>>cnt )	no	rotate n1 by bits rightwise	48	kernel
ROR?	( n1 cnt -- n1>>cnt carry )	no	rotate n1 by bits rightwise, carry=0 if no carry, else non-zero	32	kernel
SAR	( n1 cnt -- n1>>>cnt)	no	shift n1 right cnt arithmetic	40	kernel

SMARTPIN ANALOGUE

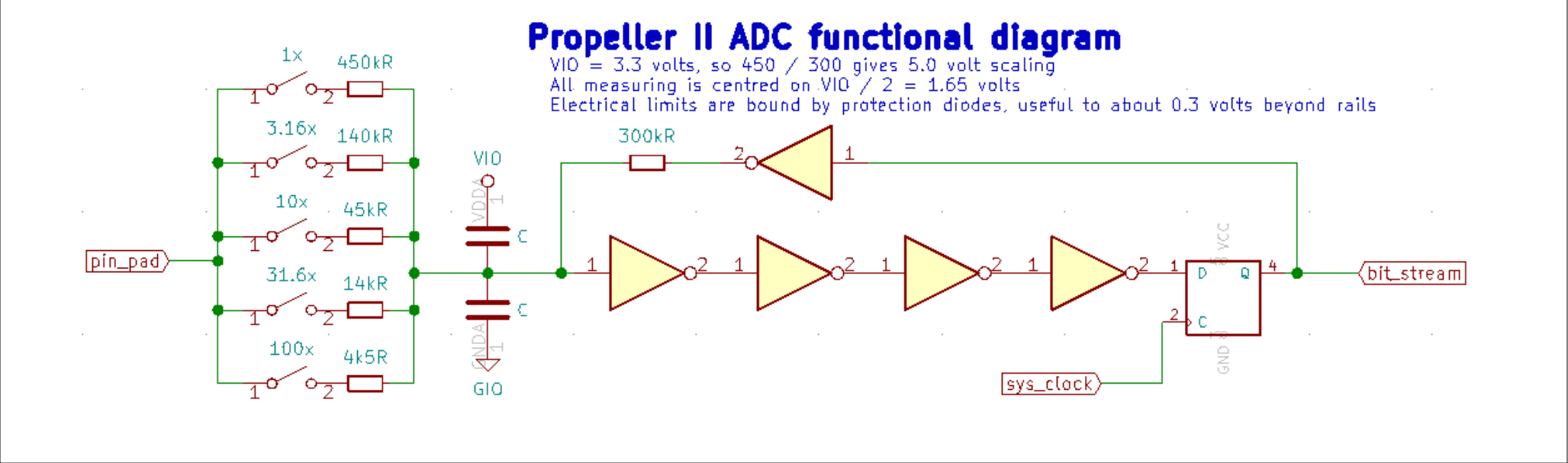
NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
ADC	( cycles -- )	no	Set the number of cycles for A/D conversion - see below		extend.fth
DAC	( bits -- )	no	16-bit dither DAC		extend.fth
mV	( millivolts --- )	no	Output a scaled voltage onto the DAC pin		extend.fth
SETADC	( min ref -- )	no	usage: 40 PIN 40000 ADC 7191 7675 SETADC - see below for further explanation		extend.fth
VOLTS@	( -- volts*1000 )	no	Read the DC input signal in millivolts from the current smartpin - N.B. this needs calibrating with SETADC prior to use		extend.fth

mV example

<p>The TEST word produces a 47 Hz 3.2V peak sawtooth waveform out of smartpin 17, until a key is pressed (assuming the P2 is running at 200 MHz)</p>	<pre>pub TEST   17 PIN   BEGIN     3200 0 DO I mV LOOP   KEY UNTIL ;</pre>
--	--

SETADC - further explanation:-

<p>SETADC stores calibration values in an array for the active smartpin before its use as an analogue input with VOLTS@. This kind of calibration is useful, because the sigma delta adc of P2 has a range different from 0...3.3V. The centre of this range is near to 1.65 Volts. The feedback resistor in the P2 smartpin circuit is 300 kohm, so for gain=1 the measurement resistor connected between signal source and P2 pin would need to be the 450 kohm option, giving <b>in theory</b> a range of 0 ... 5.0V. However, be aware of input voltage limitations of the P2 for the true input range - see the data sheet for more details.</p> <p>NB all parts in the circuit below are <b>internal</b> to the P2.</p>	<p>40 PIN --- Set the current pin to pin 40 40000 ADC --- Set the full scale value for the converter 7191 7675 SETADC --- Set offset and scaling for the current pin</p> <p>--- 7191 sets the input for 0V offset, 20000*(1-1.65V/2.5V) = 6800 --- 7675 sets the input scaling, 20000/2.5 = 8000</p>
---	--



[This article 'Output 0 - 3.3V in millivolts on any pin'](#) may also be useful for D/A conversion.

# SMARTPIN FREQUENCY COUNTER

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
.FREQ	( us -- )	no	demo to print the frequency and duty cycle		extend.fth
FREQCNT	( clocks mode0..2 -- )	no	Set up the pin in frequency counting mode		extend.fth
RAWFREQ?	( us -- clocks states periods )	no	Return with raw frequency measurements from current pin		extend.fth

[The article 'Measure Frequency and duty cycle with P2 Smartpins'](#) is a useful tutorial on frequency counting.

# SMARTPIN ELECTRICAL CHARACTERISTICS

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
IIN		no	set for inverted input		extend.fth
!OUT		no	set for inverted output		extend.fth
100ua	( – 5 )	no	drive mode operand for source/sink		extend.fth
10ua	( – 6 )	no	drive mode operand for source/sink		extend.fth
150K	( – 3 )	no	drive mode operand for source/sink		extend.fth
15K	( – 2 )	no	drive mode operand for source/sink		extend.fth
1K5	( – 1 )	no	drive mode operand for source/sink		extend.fth
1ma	( – 4 )	no	drive mode operand for source/sink		extend.fth
CLOCKED		no			extend.fth
CMOS	( – 0 )	no	drive mode operand for source/sink		extend.fth
OPEN	( – 7 )	no	drive mode operand for source/sink		extend.fth
OPEN-DRAIN	( -- )	no	open source cmos sink		extend.fth
PD	( -- )	no	alias for PULLDOWN		extend.fth
PU	( -- )	no	alias for PULLUP		extend.fth
PULLDOWN	( -- )	no	alias for SINK		extend.fth
PULLUP	( -- )	no	alias for SOURCE		extend.fth
SETPIN		no	executes 0 WRFNC		extend.fth
SINK	( val – )	no	drive mode operand for source/sink		extend.fth
SOURCE	( val – )	no	drive mode operand for source/sink		extend.fth
WRFNC	( MMMMM -- )	no	WRPIN in combo with modes - float, shift, combine, wrpin, low (DIR=1) enables smartpin		extend.fth

**To set up a pins' electrical characteristics**, use the following syntax in descriptive combination e.g.

4 PIN --- This defines pin 4 characteristics

1K5 SOURCE --- Pin 4 has a 1.5k ohm pullup to 3V3

CMOS SINK --- Pin 4 has a CMOS output transistor down to 0V

6 PIN

OPEN SOURCE

10ua SINK --- 10ua constant current sink - when pin 6 set low

# SMARTPIN NUMERICALLY CONTROLLED OSCILLATOR

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
BLINK	( pin -- )	no	flash pin on and off at 2Hz		extend.fth
DUTY	( val -- )	no			extend.fth
HZ	( Hz -- )	no	Set the current pin frequency in Hz		extend.fth
KHZ	( kHz -- )	no	Set the current pin frequency in kHz		extend.fth
MHZ	( MHz -- )	no	Set the current pin frequency in MHz		extend.fth
NCO	( count – )	no	set the NCO output to counter clock / count e.g. \$8000_0000 NCO would divide by 2		extend.fth
SETNCO	( count mode --)	no			extend.fth

[Here's an article 'Make Notes'](#) on producing a series of steady tones from a pin. Another article ['Smartpin NCO frequency'](#) is useful.

SMARTPIN PULSES

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
HILO	( high low -- )	no	e.g. If the P2 is running at 300MHz, the command 12000 6000 HILO produces pulses 40uS high, 20uS low		extend.fth
PULSE	( -- )	no	Output one pulse on the current pin		extend.fth
PULSES	( n -- )	no	Output n pulses on the current pin		
PW	( width -- )	no			extend.fth

Pulse stream example (assuming the P2 is clocking at 300MHz):-

```
48 PIN          --- Select pin 48 as the target smartpin
12000 6000 HILO --- HILO takes high and low count and sets up the Smartpin, so 40us high, 20us low
50 LOW          --- set pin 50 low
16 PULSES       --- writes wypin with that value (just an alias) to generate 16 pulses of set timing
WAITPIN         --- waits for smartpin ack (that the smartpin is ready for more)
3 PULSES        --- 3 more pulses without any other changes
50 HIGH         --- HIGH drives P50 high to indicate that the code has completed (although the smartpin is still busy)
```

This is explained further in the [article 'Smartpin variable pulse stream'](#)

SMARTPIN PWM

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
PWM	( duty frame div -- )	no			extend.fth
SMPS	( duty fram div -- )	no			extend.fth
TRI	( duty frame div -- )	no			extend.fth

PWM example:-

[This article 'Smartpin PWM'](#) is useful

Here we setup 4 pins P0..P3 to output 4 different rates from 20 to 80% duty cycle:-

```
4 FOR I PIN I 1+ 20 * 100 10 PWM NEXT
```

SMARTPIN UART

TAQOZ, by default, expects a computer terminal connected to smartpin P62 for data transmitted by the P2 and smartpin P63 for data received by the P2. These pin assignments can be changed permanently or temporarily - use SEE CONBAUD to see how that's done.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-bit		no	Set up the number of data bits		extend.fth
COMPORT	( pin -- )	no	All further console output is diverted to this pin as a serial out port		extend.fth
CONBAUD	( baud -- )	no	Set pin 62 for o/p, 63 for i/p as the console i/o, set at 'baud' rate		extend.fth
RXD	( baud -- )	no	Set up the receiver baud rate		extend.fth
SEROUT	( data pin -- )	no	Transmit a character through the smartpin that is normally configured for asynch		extend.fth
TXD	( baud -- )	no	Set up transmitter baud rate		extend.fth
TXDAT	( buf cnt -- )	no	write byte buffer direct to WYPIN		kernel

Transmit examples:-

```
48 PIN 1 M TXD          --- Set up pin 48 for 1Mbaud rate serial output
48 COMPORT PRINT" Hello World! " CON --- Output a serial string on pin 48, then revert to the default console pin
```

8 PIN 115200 RXD -- Set up PIN 8 as serial input at 115200 baud (default 8-bit)

9 -bit 1 M TXD --- Set up serial output for 9-bit 1 megabaud

[This article 'Smartpin asynch serial transmit'](#) gives some further insight into serial comms, with 'scope traces.

Receive example (thanks to Peter\_F on the Parallax forum):-

--- This example repeats the incoming 9600 baud NMEA sentences from a GY-GPS6MV2 module on smartpin 55 to the terminal screen

```
9600 := BRATE          --- Smartpin asynch operation at this baudrate only works at 20 MHz clock rate e.g. P2-EVAL board
55  := RXPIN           --- Smartpin P55 chosen for serial data input - N.B. P2 IO input voltage must be no greater 3.3 volts
```

```
pub SERIN ( -- char )  --- Wait until char received from smart pin serial input
  WAITPIN              --- wait for smartpin acknowledgement
  RDPIN                --- receive smartpin char on RXPIN
  #24 >>               --- shift data from bits 31-24 to 7-0
;
```

```
pub RXINIT ( -- )                --- initialise smartpin for ASYNC serial RX at BRATE baud rate
  RXPIN PIN MUTE                 --- stop any previous smartpin activity on P55
  BRATE RXD H                    --- H - set pin high needed to enable smartpin operation?
;

pub GPS. ( -- )                  --- Repeat incoming serial stream to the terminal until key pressed
  RXINIT                         --- set up the serial input port
  CRLF
  BEGIN
  SERIN EMIT                      --- stream incoming chars to the terminal screen
  KEY UNTIL                      --- until the user presses a key
;
```

If two way communication is required, then the transmit function must be run in a separate cog from the receive, since both operate asynchronously from each other. For an example of this, see [Peter-F's full article](#) on the Parallax forum. It demonstrates a loopback test of the Serial Tx to the Rx.

SPI

The default smartpins for the SPI bus are:-  
pin P27        slave select  
pin P26        master in slave out  
pin P25        master out slave in  
pin P24        serial clock  
These pin assignments may be changed using the SPIPINS word

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
*SPIRAM*		PRE?	Module header marker – use with FORGET. This module supports LY68L6400 8MB SPI RAM		
CLKS	( cnt -- )	no	idle is set low, mosi set high, then emit cnt clocks		kernel
SPICE		no	Set SPI slave select and clock high		kernel
SPIPINS	( &cs.mi.mo.ck -- )	no	Setup I/O pins to be used for SPI instructions		kernel
SPIRD	( n1 -- n2 )	no	Read 8-bits left into n1 so that n2 = n1<<8+new. Four successive SPIRDs will receive 32-bits		kernel
SPIRL	( n1 -- n2 )	no	Read 32-bits from SPI		kernel
SPIRX	( dst cnt -- )	no	Read bytes in from SPI to memory		kernel
SPITX	( src cnt -- )	no	Write bytes from memory to SPI		kernel
SPITXE	( src bytes -- )	no	Write bytes from memory to SPI		kernel
SPIWB	( byte -- )	no	Shift 8 bits from data[0..7] out and leave data on stack (restored with other bytes zeroed)		kernel
SPIWC	( com – )	no	Write SD Command		kernel
SPIWL	( long -- )	no	write 32 bits		kernel
SPIWM	( long -- )	no	write 24 bits		kernel
SPIWR		no			kernel
SPIWW	( word -- )	no	write 16 bits		kernel

STACK

- Taqoz stacks are all located in LUT memory with a bit of COG memory too. All stacks extend upwards in address as data is stored in them. LUT memory for each COG is 32 bits wide, 512 long. It is only addressable as longs.
1. The data stack is 32 longs deep, starting at LUT address \$0. The top four entries of this stack are stored in COG memory, however.
  2. The loop stack used by DO, FOR etc. is 16 longs deep starting at LUT address \$20. The top three entries in this stack are in COG memory. Each loop will consume 3 longs in the stack, so loops can be a maximum of 6 deep at any one place in an application.
  3. The branch stack used by IF THEN etc. is 16 longs deep starting at LUT address \$30
  4. The return stack used for subroutining, R> , >R etc. is 64 longs deep starting at LUT address \$40
  5. The auxiliary stack used by >L and L> starts from LUT address \$80 to the end of LUT at \$1FF if the user so wishes. More likely the user will choose to limit the depth of the L stack in his application. The rest of LUT memory is then available for user application storage. User variables should be located near the \$1FF end of LUT, to maximise the L stack size.

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
-ROT	( c b a -- a c b )	no	rotate the top four data stack entries	56	kernel
-ROT4	( a b c d -- d a b c )	no	reverse rotate the top four data stack entries	56	kernel
!SP	( – )	no	Empty the data stack	40	kernel
?DUP	( n1 -- n1 n1   0	no	DUP n1 if non-zero	56 if non-zero, else 40	kernel
>L	( n – )	no	Push n onto the general purpose L stack	40	kernel
>R	( n -- )	no	Push n onto the return stack from the data stack	40	kernel
2DROP	( n1 n2 -- )	no	Drop the top two data stack entries	56	kernel
2DUP	( n1 n2 -- n1 n2 n1 n2 )	no	Duplicate the top two data stack entries	80	kernel
2SWAP	( n1 n2 n3 n4 -- n3 n4 n1 n2 )	no	Swap the top two data stack entries with entries 3 and 4	56	kernel



3DROP	( n1 n2 n3 -- )	no	Drop the top three data stack entries	64	kernel
3RD	( n1 n2 n3 -- n1 n2 n3 n1 )	no	Copy the 3rd item on the stack	48	kernel
4TH	( n1 n2 n3 n4 -- n1 n2 n3 n4 n1 )	no	Copy the 4th item on the stack	48	kernel
b++	( n1 n2 – n1+1 n2 )	no	Increment 2nd on the stack	56	kernel
BOUNDS	( n1 n2 – n3 n2 )	no	n3 = n1 + n2		kernel
DEPTH	( – n )	no	Return the number of entries on the data stack	64	kernel
DROP	( n -- )	no	Pop the data stack using fixed size register stack in COG memory (allows fast direct access for operations)	48	kernel
DUP	( n1 - n1 n1 )	no	Duplicate the top item on the stack	48	kernel
L>	( – n )	no	Pop n from the general purpose L stack	56	kernel
NIP	( n1 n2 -- n2 )	no	drop the 2nd stack entry only	48	kernel
OVER	( n1 n2 -- n1 n2 n1 )	no	read second data item and place copy top of data stack	48	kernel
OVER+	( n1 n2 -- n1 n2+n1 )	no	read second data item, add to top of data stack	32	kernel
R>	( -- n )	no	Pop n from the return stack to the data stack	56	kernel
ROT	( a b c -- b c a )	no	rotate the top three data stack entries	40	kernel
ROT4	( a b c d -- b c d a )	no	rotate the top four data stack entries	64	kernel
SWAP	( n1 n2 -- n2 n1 )	no	Swap the top two items	40	kernel

## STRINGS

The end of TAQOZ strings are marked with byte 00 (zero delimited strings)  
If it is required to place a single printable character on the data stack, then enclose the char in single quotes e.g. 'A' places 41 hex on the stack  
To print a non-visual character, include the chars \ \$nn in the string e.g. " \\$07" sounds the bell on the terminal, when printed.  
Here's the [ASCII character set](#).

NAME	STACK EFFECT		DESCRIPTION	TIME (CYCLES)	SOURCE
[“]	( -- )	yes	Compile string inline e.g. [“] my string” within a definition		kernel
“	( -- str )	yes	Interpret a string in the input stream up to the trailing " and leave the address of the null terminated string on the stack e.g. " Hi Bob"		kernel
>STR	( num -- str )	no	convert num to string at addr str		extend.fth
>UPPER	( str1 -- )	no	Convert string at address str to upper-case		kernel
\$!	( str1 str2 -- )	no	Store string at hub memory address str1, at address str2 e.g. 16 bytes mystring " Hello" mystring \$!		kernel
\$+	( str1 str2 -- strx )	no	strx = str1 + str2 ..... strx being a temporary 32 char store (returned by X\$)		extend.fth
\$+!	( str1 str2 -- )	no	str2 = str1 + str2		extend.fth
\$=	( str1 str2 -- flag )	no	flag = true if string at hub memory address str1 is identical tostring at str2, else flag = false		kernel
\$=	( str1 str2 – flag )	no	flag = true if string at hub memory address str1 is identical to string at str2, else flag = false		extend.fth
\$>#	( str -- value digits   false )	no	attempts to convert str to number		kernel
a>A	( ch1 – ch2 )	no	Converts ch to uppercase if a-z, else ch2 = space		kernel
LEN\$	( str - n )	no	n = length of string at addr str e.g. mystring LEN\$		kernel
NULL\$	( – str )	no	An empty string at address str		kernel
NUMBER	( str -- value digits   false )	no	an alias for \$>#		kernel
SPACE?	( – flag )	no	returns true if delimiter is the space char		extend.fth
UPPER\$	( str – )	no	Convert lower-case letters to upper-case		kernel
WORD\$	( str1 -- str2 )	no	terminate a string at the end of the first word		extend.fth
X\$	( -- n )	no	Returns the address n of a 32 char temporary store for strings - used in \$+		extend.fth

## TIMING

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
?UNTIL	( daddr -- )	no	a double variable at daddr stores a TIMEOUT. ?UNTIL will wait until that timeout occurs - see below		extend.fth
CLKDIV		no			kernel
CLKHZ	( – n )	no	Returns the p2 clock frequency in Hz		kernel
CNT@	( – n )	no	read the 32 bit CNT register		kernel
CYCLES	( cycles -- )	no	define a delay unit in processor cycles		kernel
DELAY	( -- )	no	Wait for one delay		kernel
DELAYS	( n -- )	no	Wait for n delays		kernel
GETMS	( – m )	no	read the number of milliseconds since reset - rolls over in about 49 days		kernel
INS	( cnt -- )	no	interruptable count loop used for timing instructions		kernel
ms	( n -- )	no	wait n milliseconds		kernel

s	( n -- )	no	wait n seconds		kernel
TIMEOUT	( ms daddr -- )	no	a simple timer, value 'ms' is stored at the double variable daddr, along with the current time. ms can be in the range 0 - 65535 - see below		extend.fth
TIMEOUT?	( daddr -- flg )	no	flg = true indicates the TIMEOUT stored in double variable daddr has passed - see below		extend.fth
us	( n -- )	no	wait for n microseconds		kernel
WAIT	( n -- )	no	Wait for n clock cycles to pass (uses P2 WAITX)		kernel

**TIMEOUT examples**

8 bytes TIME1
 --- storage for a timeout - (a 'double' variable)

1000 TIME1 TIMEOUT
 --- start a timeout, stored at TIME1

TIME1 TIMEOUT? .
 --- we check whether TIME1 has timed out yet - would return 0 (false)

2000 ms TIME1 TIMEOUT? .
 --- we check again, after a delay of 2s - TIMEOUT? returns -1 (true)

3000 TIME1 TIMEOUT
 --- somewhere in a program we start a timer and elsewhere we can program ...

TIME1 ?UNTIL
 --- ?UNTIL will only finish executing once timeout has occurred

**VIDEO**

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
@VGA	( n -- addr )	no	return with VGA parameter table address		kernel
DWIDTH	( src scr -- src+640 scr )	no	3ms per frame QVGA-->VGA FULL-SCREEN upscaler. Upscales 320x240 bmp frames to 640x480		kernel

**VOCABULARY**

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
@VOCAB	( index -- addr )	no			extend.fth
+VOCAB	( nfa --- )	no	push this vocab to the top of the search list		extend.fth
FORTH	( -- )	no	Make Forth the current vocabulary		extend.fth
SEARCH-VOCABS	( str -- nfa )	no			extend.fth
VOCAB	( <name> <including> -- )	yes	First create a dummy link word with no skip (04) Then create a definition that calls the vocab linking code Then calculate the link displacement and update it Use: VOCAB ASSEMBLER *TIA*		extend.fth

**Vocabulary example**

pub \*MYVOCAB\* ;
 --- mark the start of a new vocabulary

pub TEST1 10 0 DO I . LOOP ;
 --- the 1st word in the new vocabulary, display 0 to 9

pub TEST2 20 10 DO I . LOOP ;
 --- another word in the new vocabulary

VOCAB MYVOCAB \*MYVOCAB\*
 --- create the vocabulary, encompassing the words between here and the word \*MYVOCAB\*

FORTH

TEST1
 --- TAQOZ will respond with "???" as TEST1 will not be found in the FORTH vocabulary

MYVOCAB

TEST1
 --- TAQOZ reponds with "0 1 2 3 4 5 6 7 8 9 ok", because TEST1 is now in the search path

See PASM2.FTH, the TAQOZ Assembler source file, for another example of the use of VOCAB.

**ORPHAN WORDS**

NAME	STACK EFFECT	PRE?	DESCRIPTION	TIME (CYCLES)	SOURCE
HUBEXEC	( – )	no	marker for end of HUBEXEC code itself (NOP)	Obsolete – do not use	kernel
RCLIO		no			kernel
WE		no			kernel
WP		no			kernel

# SYSTEM INSTALLATION and BOOTING UP

## Dip your toe in with no effort - try TAQOZ ROM

[This article 'Getting Started'](#) is useful advice for trying out TAQOZ ROM, built right into the P2. This is a useful first step in testing any new P2 board, as it doesn't depend on any external memory or SD card. Not much more than the serial terminal port has to work. The ['BitBashers Guide'](#) is also a good intro. The [glossary for TAQOZ ROM](#) is a useful programmer's reference.

### Fake SD cards

**There are many fake SD cards about.** If the bargain on ebay appears to good too be true – it probably is. Fake cards will be packaged convincingly and also report the advertised size when plugged in. They may appear to work on the PC, but not support the SPI interface required by Taqoz. When tested on a PC with a 'capacity test' program they may fail, showing the part to be smaller capacity than it reports.

To minimise the risk of buying a fake SD card, I buy from a large computer store just outside my home town. That way, if there is a problem I can take it back, no questions asked.

## Installing TAQOZ RELOADED - Method 1

The **simplest way** of installing TAQOZ Reloaded is by means of a .BIX file saved on an SD Card:-

1. Download the [TAQOZ forth system](#)
2. Unzip it to a folder on your Windows PC
3. Format a **Sandisk SD card** on Windows using [SD Card Formatter](#) with **Overwrite format** set. ( Other makes have been found to work, but *don't* expect much from unmarked SD cards you've found in your junk box or cheaper fakes purchased from ebay )
4. Copy the latest \_BOOT\_P2.BIX file from drop box folder P2\TAQOZ\binaries
5. Rename this file if need be to exactly \_BOOT\_P2.BIX
6. Install the SD card in the P2 board and set your P2 to boot from SD card (This may be a dip switch or a link)
7. Start tera term and set **Serial port** to the port your P2 is connected to, 921600 baud, 8 bits, no parity, 1 stop bit, no flow control. Set **Terminal** to CR for New-line on Receive and Transmit, no local echo, terminal VT-100
8. Reset the P2 and a **TAQOZ Reloaded** start up message should appear in the tera term window.

### Some background information on what is going on:

The root directory is searched for file "\_BOOT\_P2.BIX", and if not found then the root directory is searched for file "\_BOOT\_P2.BIY" (upper case is required). If the file is found, then all files' data is loaded into HUB \$0. Maximum file size is 512KB-16KB = 496KB. Once loaded into HUB, the first 496 longs will be copied to COG \$0 and then a JMP #\$0 will be executed. Note: The files' data must be in contiguous sectors.

If the above doesn't work for you, then try Method 2 below.

## Installing TAQOZ RELOADED - Method 2

This method takes more steps, but allows the user to select which modules are installed. Method 1 installs the lot. Here is what worked for me on a Windows computer:-

1. Download the [Taqoz forth system](#)
2. Unzip it to a folder
3. Set the '**Flash**' dip switch on the P2-EVAL, reset **P59^** and **P59v**. **USB RES** stays set.
4. Download the [flexprop tool](#) and unzip it to a folder
5. Run flexprop and in **Ports** set 921600 baud and the com port your P2 is connected on
6. Using flexprop, send file **TAQOZ.bin** from folder **Tachyon dropbox April 2021\P2\TAQOZ** to the P2 with **Flash binary file** found in the **Commands** menu - it only takes a second or so as shown by just three lines on the **Propeller Output** window.
7. Shut down flexprop
8. Start tera term and set **Serial port** to the port your P2 is connected to, 921600 baud, 8 bits, no parity, 1 stop bit, no flow control. Set **Terminal** to CR for New-line on Receive and Transmit, no local echo, terminal VT-100
9. Press the reset on the P2-EVAL, you should get this start up message from TAQOZ v2.8:-

```
pxQOZ#
*Cold start*
-----
Parallax P2 *TAQOZ RELOADED sIDE* V2.8 'CHIP' Prop_Ver G 200MHz 210401-1230
-----
TAQOZ#
```

10. Using tera term, upload to TAQOZ two TAQOZ modules **EXTEND.FTH** and then **FILE.FTH** from folder **Tachyon dropbox April 2021\P2\TAQOZ\Forth** Check they compile with no errors. You might decide to upload others. I loaded **P2ASM.FTH** to allow writing new TAQOZ words in assembly code as I am interested in DSP for software defined radio.

11. Format a **Sandisk SD card** on Windows using [SD Card Formatter](#) with **Overwrite format** set. (Other makes have been found to work, but don't expect much from unmarked cards found in the bottom of your junk box or cheaper fakes purchased from ebay ) Insert newly formatted SD card into the P2-EVAL
12. At the TAQOZ prompt, type **BU** *enter* to backup the tool set to the SD card - this takes around 2 s
13. Reset the **FLASH** dip switch, so that the P2 will boot from SD card next time
14. Reset the P2, the following start up message appeared (for my choice of extensions) in tera term. Notice the results of an improved [I2C bus scanner](#) at the bottom of the start-up message :-

```
BOOTING.... CARD: SANDISK SD SD32G REV$85 #3200344782 DATE:2021/1
KERNEL      Parallax P2 *TAQOZ RELOADED sIDE* V2.8 'CHIP' Prop_Ver G 200MHz 210401-1230
MODULES:
4890 *P2ASM*      TAQOZ INTERACTIVE ASSEMBLER for the PARALLAX P2 - 210124-1200
2726 *DISK*       SD DISK REPORTING & FORMATTING TOOLS 190800-0000
5042 *FILE*       TAQOZ FAT32 FILE SYSTEM for SD CARD plus VIRTUAL MEMORY 210128-0830
1996 *SPIRAM*     LY68L6400 8MB SPI RAM ACCESS 191020-0000
1760 *DECOMPILER* A decompiler for TAQOZ 190825-0000
822 *UBI2C*       EFM8UB3 I2C COMMANDS 210118-0000
614 *RTC*         RV-3028 RTC DATE and TIME 190800-0000
882 *SMARTPINS*   SMARTPIN FUNCTIONS and drive modes 190800-0000
382 *P2CLOCK*     P2 CLOCK CONTROL 190800-0000
2780 *ANSI*       ANSI TERMINAL SUPPORT 200410-0000
494 *EXTEND*      Primary kernel extensions for TAQOZ 210421-2100
5990 *SPIFLASH*

MEMORY MAP
CODE:      08F3C 36,668 bytes
WORDS:     1C6F5 14,498 bytes
DATA:      01EF5 1,368 bytes
ROOM:      79,801 bytes
HARDWARE
PCB        P2
CLOCK IN   20.000000MHZ
DEVICES
SD CARD    31 GB SANDISK SD SD32G REV$85 #3200344782 DATE:2021/1

I2C DEVICES
Address Max clock Identity
$1A 2600kHz QMC5883 Magnetic Field Sensor
$C0 1700kHz Si5351A Clock Generator
$EC 2100kHz BMP280 Pressure Sensor or MS5611 Pressure Sensor with pin CSB=1
$EE 2600kHz MS5611 Pressure Sensor with pin CSB=0

STATS      cpufreq: 200MHz systicks: 1,497
-----
TAQOZ#
```

15. TAQOZ is now installed on SD card, although it won't appear in the directory as a filename
16. Other files can be loaded onto the SD card from Windows - and they *will* be listed with the TAQOZ **DIR** or **DIRW** command

**Some background information on what is going on:**

The SD boot will check for valid P2 Boot Data on the MBR sector (sector 0), and if not found it will try the VOL sector (provided the SD Card is in FAT32 format).

- Valid Boot Data is "Prop" at offset \$17C on the MBR/VOL sector...The sector (512 bytes) will be read into Hub \$0, followed by a JMP \$080. Only code in sector byte offset \$080-\$17F is considered valid but this program may know that other areas are also valid.
- Valid Boot Data is "ProP" at offset \$17C on the MBR/VOL sector...The long at offset \$174 is used as the sector start address, and the long at \$178 is used as the maximum filesize (in bytes) is 512B-16KB = 496KB for the data to be loaded into Hub, followed by a `JMP $000`.
- These two methods also cover the case where the SD Card is not formatted as FAT32 (eg exFAT). The boot sector(s) MBR and/or VOL must be specifically written with a program such as HxD on Windows 10.

**SD card formatting advice**

Peter Jakacki advises "If you want to format your SD card then I'd recommend using the maximum cluster size that FAT32 supports of 64KB which is much better suited for embedded systems. Even if you had one thousand small files (really!?), that's still only 64MB which is barely noticed even in the smallest SD cards. I've made that the default now although you can specify the cluster size and also set permissions (for safety) before a FORMAT".

The SD Associations's format utility on Windows doesn't allow you to choose cluster size, but it is selectable when reformatting the SD card from the P2 using Taqoz:-

64 KB CLSZ RWS FORMAT

...this would format the SD card with a 64 kilobyte cluster and runs quite quickly. Notice that you will need to reload any Taqoz system and other files that were on the card – it will be wiped clean.

**SYSTEM BACKUP**

The word **BACKUP**, defined in **FILE.FTH** from folder **Tachyon dropbox April 2021\P2\TAQOZ\Forth** is used to permanently store the current TAQOZ system to the Flash Memory or SD card. You can check FILE.FTH is compiled in with your TAQOZ system as part of the start-up message:-

```
5042 *FILE*      TAQOZ FAT32 FILE SYSTEM for SD CARD plus VIRTUAL MEMORY 210128-0830
```

**Saving to SD card**

The command options are:-

- **BACKUP <filename>** Remember this is limited to 8chrs with 3chr extension
- **BACKUP BIX**, save the system to file \_BOOT\_P2.BIX in the root folder
- **BACKUP MBR** backs the system up to sector 1 and sets the boot signature in the MBR. (Assumes you have a properly formatted SD card with unassigned storage before the first partition) As a shortcut, you can also just type **BU <enter>** or press **<ctrl>B**

**Saving to Flash memory**

The command is:-

- **BACKUP FLASH** will write the bootloader to flash page 0 and then writes the 128kbyte TAQOZ image.  
As a shortcut, you can also press **<ctrl>F**

**Backup disable**

- **BACKUP DISABLE** will disable Flash and MBR booting only - if you also want to disable the BIX file then use something like  
RENAME \_BOOT\_P2.BIX \_BOOT\_P2.BIN

**AUTO-START A USER PROGRAM FROM SOURCE CODE**

Thanks to Christof Eb on the Parallax forum for this one:-

You want an application, stored as source code on SD Card, to load when the P2 is switched on. The following code would load file 'auto.fth' from the SD card:-

```
: aload
  " auto.fth" FOPEN$      --- attempt to open auto.fs
  IF                      --- if it opens
    @FILE FLOADS          --- starting from the 1st sector, load file as console input
  THEN
  ;

!INITS                    --- Initialise and clear all polling vectors
' aload +INIT             --- add the word aload to the list that executes when TAQOZ starts

BACKUP BIX                --- save the TAQOZ system to SD Card as _BOOT_P2.BIX
```

This approach maximises available memory: The user program can load and unload parts of itself to suit the task in hand, rather than it all be loaded at start up and taking up unnecessary space that would otherwise be useful.

**AUTO-START A COMPILED USER PROGRAM**

You have your program compiled in hub memory as part of the Taqoz system: You want that program to run when the P2 is switched on. Here's a small example 'user program' - it counts until the user presses a key:-

```
pub TEST 0 BEGIN 1+ DUP . SPACE KEY UNTIL DROP ;
```

To get this to run on switch on:-

```
' TEST +INIT      --- Add TEST to the list of words that run at start-up
BU                --- Save TAQOZ to SD card
```

When TAQOZ is reset, TEST will run and pressing a key will return you to the system prompt. The system can be returned to normal for more coding by:-

```
FORGET TEST      --- Forget the user application from the dictionary
!INITS           --- Remove all user words form the list that runs at start up
BU               --- Save TAQOZ to SD card
```

**TERMINAL BAUD RATE**

TAQOZ v2.8 terminal runs at a default of 921600 baud. This baud rate can be inspected and changed:-

28 @ . will print the present baud rate of 921600

115200 28 ! changes the baud rate to that lower speed, which will take effect the next time the system is restarted if the system is backed up. Also don't forget to reset the terminal to the same rate, ready for TAQOZ restart.

By default, the terminal serial input to the P2 is on smartpin P63, the serial output is smartpin P62.



# DICTIONARY STRUCTURE

The dictionary is a contiguous block of word headers. New word headers extend downwards in memory, usually located within the 2nd 64k of RAM, although the dictionary can be relocated. The address of the most recently defined word header in the dictionary is returned by @WORDS. This points to the char count byte in the word header. The same is true of word NFA’ as can be seen when this Taqoz phrase is run to display the dictionary header of the word DUP:-

NFA’ DUP 10 DUMP

Each word header within the Taqoz dictionary, **in increasing address order**, comprises:-

### The char count byte

b7	b6	b5	b4	b3	b2	b1	b0
WORD ATTRIBUTES		CPA TYPE	WORD NAME STRING COUNT N				

Word attributes can take the following values:-

00 = public or code word

01 = private word – header can be reclaimed

10 = pre-emptive word – executes immediately

11 = command line interactive but compiles in definition

CPA type:-

0 = Word has a 16-bit CPA (located in the dictionary when less than 64k in size – typical of a small to medium size progam)

1 = Word has a 24-bit extended CPA (header in the top portion of the dictionary, when it has exceeded 64k in size – due to a mega size program)

### Word name string

The word name occupies N bytes as defined by the string count in the char count byte:-

b7	b6	b5	b4	b3	b2	b1	b0
char1 (first)							
char2							
...							
char N (last)							

### The CPA

The CPA contains the address of the start of code to execute.

The CPA contains a 16 bit code field address if CPA TYPE = 0 in the char count byte

OR contains a 24 bit extended code field address if CPA TYPE = 1 in the char count byte

i.e. If CPA TYPE = 0

b7	b6	b5	b4	b3	b2	b1	b0
ls byte							
msbyte							

If CPA TYPE = 1

b7	b6	b5	b4	b3	b2	b1	b0
ls byte							
middle byte							
msbyte							

The next word in the dictionary follows immediately.

### Dictionary Bottom

The end of the dictionary is indicated by a dummy word header whose char count byte = 00

# Wordcode structure

Forth words are structured in various ways, to suit the function required. The following sections describe the word family:-

## Coded words in Hub or Cog RAM

<17ff: Address of assembler code, will be just called in Hub or Cog Ram. Ends with RET.

## Threaded words

>\$2000: Normal Wordcode is the address of the wordcodes of this word.

## Short Literals

\$1800..\$1BFF: \$1800 + 10bit literal, positive numbers

## Relative Conditional Branches

Relative conditional Branches: \$1Cxx: xx ist the relative jump distance +/- 128 words

## Relative Unconditional Branches

\$1Dxx

## (Code)

\$0065 is followed by PASM code, which ends with RET

## Paging words

PAGE1...PAGE7 \$004e...\$0054: This wordcode is followed by a word, that contains the lower word \$xxxx of the address of code, which is located at \$000p\_xxxx.!

# USEFUL LINKS

## Documentation:-

Parallax's [Propeller 2 Documentation](#) page

The [Parallax Forum](#) is a great resource for all things Propeller P2. Here's the [TAQOZ thread](#).

The latest version of this document [can be found here](#).

The [TAQOZ P2 Boot Firmware](#) web site is a useful intro to TAQOZ

This [Tachyon Forth Model paper](#) explains quite a lot of the differences form ANSI forth

These [TAQOZ and Tachyon links](#) are useful

The [Propeller P2 Get Started Here](#) page

A thread on the [TAQOZ Interactive Assembler](#)

Notes on getting started [writing inline Assembly code](#)

The P2 Assembly Language [Instruction set](#) table

The P2 [Assembly Language Manual](#)

120 back issues of [the magazine Forth Dimensions](#) - full of code snippets to adapt to Taqoz or Tachyon

An [ASCII Table](#) is always useful

Tera Term for Windows [help index](#)

## Code:-

All source code and binaries for TAQOZ [can be found here](#)

Tera Term for Windows [can be found here](#)

[CCS811 Air Quality sensor](#) via i2c

[SI5351 dual clock driver via i2c](#) which can provide two clocks 8kHz to 120MHz with around 2Hz resolution

[QMC5883 magnetic field sensor](#) via i2c

[MS5611 barometric pressure sensor](#) via i2c

[64 bit integer maths](#) library - not often needed, but when you do - you do!

[Mini-OOF](#) - a tiny object oriented forth module originally written by Bernd Paysan and now adapted for TAQOZ

[Timers](#) - running multiple words periodically on one COG or more, each with it's own period - written in Mini-OOF

[State Machines](#) - running multiple state machines on one COG or more - written in Mini-OOF

[Queues / Stacks](#) - written in Mini-OOF, so as many of them as needed can be made

A few [useful string words](#) borrowed from Tachyon Forth

Running [commands from SD card](#)

Improved [I2C bus scanner](#) shows max clock speed and identifies a few extra devices

The P2 has [16 lock bits](#) used to prevent race conditions in executing code or accessing data

Reading a [quadrature encoder](#)

Synchronising [COG execution using ATN](#)

Hardware:-

There are now quite a few modules available, for p2 evaluation and development or encorporation within your own product

A [reference design, the P2D2](#) by Peter Jakacki



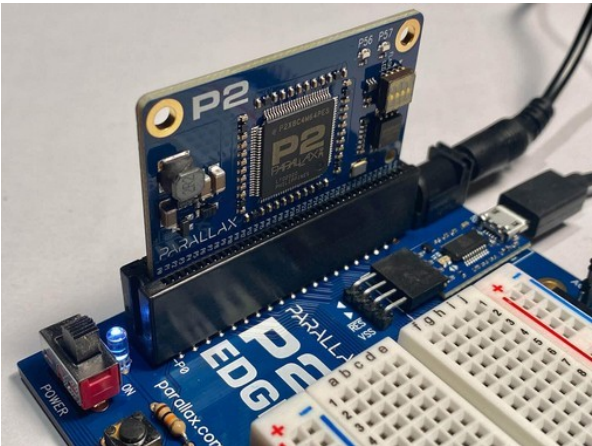
The [P2-EVAL](#) evaluation board



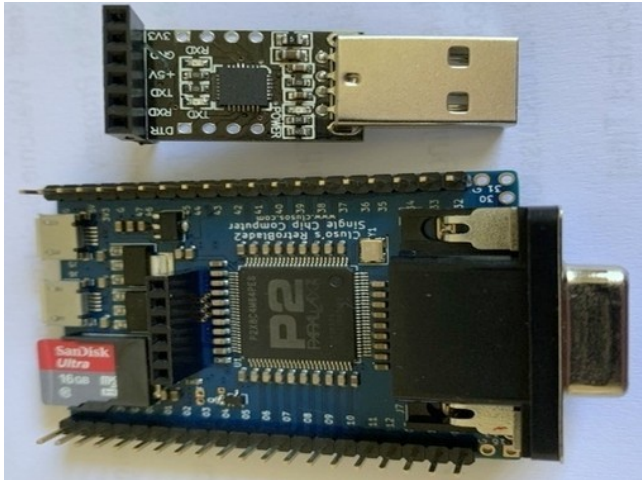
The [P2 Edge](#) module



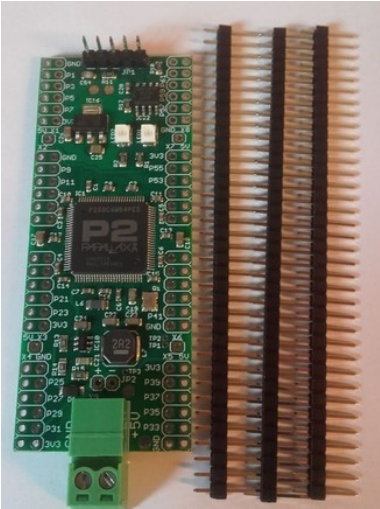
Here's the Edge module mounted in the P2 Edge Module breadboard



The [RetroBlade2](#) board



The [KISS Eval](#) board



CONCLUSION

If you'd like a change or make a contribution to this glossary, you're very welcome to get in touch via the Parallax Forum. It's likely the document still contains a few errors and omissions. The document is up-issued quite often, as and when I've discovered something new. Quite a few words are not fully described - there wasn't enough information in the source listings or forum. If you know how to use these words, then please tell.

This glossary was compiled by Bob Edwards, retired EMC engineer in SW U.K. Radio Amateur callsign G4BBY, 2021-2023. He uses it all the time whilst programming with TAQOZ Reloaded, and hopes you do too.