My4TH light Forth Glossary

Dennis Kuschel
dennis_k@freenet.de
2023-12-08

My4TH light tries to adopt the Forth 2012 Standard as far as possible. Actually My4TH light implements all Core words, Core extension words, Block words and Double-Number words. In addition, many Floating-Point words and some other standard words from different categories are also implemented.

Because the standardized Forth words are publicly documented, this document is limited to the overview of all words. If you want to learn more about how the standardized words work, please follow this link to the Forth 2012 standard on the web: https://forth-standard.org/standard/words

Besides the standard words, My4TH light also implements some special words which are described in detail later in this document. For example, there are a couple of built-in words that support some of Adafruit's I^2C breakout boards for extended GP-I/O.

Contents

Forth Standard Words

Core Words

!	2DUP	C,	HOLD	S"
#	2OVER	C@	I	S>D
#>	2SWAP	CELL+	IF	SIGN
#S	:	CELLS	IMMEDIATE	SM/REM
'	;	CHAR	INVERT	SOURCE[1]
(	<	CHAR+	J	SPACE
*	<#	CHARS	KEY	SPACES
*/	=	CONSTANT	LEAVE	STATE
*/MOD	>	COUNT	LITERAL	SWAP
+	>BODY	CR	LOOP	THEN
+!	>IN[1]	CREATE	LSHIFT	TYPE
+LOOP	>NUMBER	DECIMAL	M*	U.
,	>R	DEPTH	MAX	U<
-	?DUP	DO	MIN	UM*
.	@	DOES>	MOD	UM/MOD
."	ABORT	DROP	MOVE	UNLOOP
/	ABORT"	DUP	NEGATE	UNTIL
/MOD	ABS	ELSE	OR	VARIABLE
0<	ACCEPT	EMIT	OVER	WHILE
0=	ALIGN	ENVIRONMENT?	POSTPONE	WORD
1+	ALIGNED	EVALUATE	QUIT	XOR
1-	ALLOT	EXECUTE	R>	[
2!	AND	EXIT	R@	[']
2*	BASE	FILL	RECURSE	[CHAR]
2/	BEGIN	FIND	REPEAT	]
2@	BL	FM/MOD	ROT	
2DROP	C!	HERE	RSHIFT	

1) My4TH light handles the input buffer differently than the standard. Since additions are slow, no "base address and offset" scheme is implemented. Instead, the offset to the input is an absolute memory address and the base address of the buffer is always zero.

Core Extension Words

.(	ACTION-OF	ENDOF	PARSE	TRUE
.R	AGAIN	ERASE	PARSE-NAME	TUCK
0<>	BUFFER:	FALSE	PICK	U.R
0>	C"	HEX	REFILL	U>
2>R	CASE	HOLDS	RESTORE-INPUT[2]	UNUSED
2R>	COMPILE,	IS	ROLL	VALUE
2R@	DEFER	MARKER	S\"	WITHIN
:NONAME	DEFER!	NIP	SAVE-INPUT[2]	[COMPILE]
<>	DEFER@	OF	SOURCE-ID	\
?DO	ENDCASE	PAD	TO[3]	

2) Not implemented on My4TH light. Due to the non-standard implementation of input buffers and pointers on My4TH light, it is almost impossible to implement this correctly. It is up to the user to provide dummy implementations of these words if needed.

3) The implementation of TO in My4TH light handles the assignment of single and double word values correctly.

Block Words

BLK	EMPTY-BUFFERS	LOAD	SCR
BLOCK[4]	FLUSH	REFILL	THRU
BUFFER[4]	LIST	SAVE-BUFFERS	UPDATE

4) Only one block buffer is implemented. If this word is called twice with different block numbers, the same buffer is returned.

Double-Number Words

2CONSTANT	D+	D0=	D>S	DU<
2LITERAL	D-	D2*	DABS	M*/[5]
2ROT	D.	D2/	DMAX	M+
2VALUE	D.R	D<	DMIN	
2VARIABLE	D0<	D=	DNEGATE	

5) On My4TH light, only a two-cell intermediate result is produced. This is to save processing time and reduce code size.

String Words

BLANK	CMOVE>	/STRING	SEARCH
CMOVE	COMPARE	-TRAILING	SLITERAL

Facility Extension Words

+FIELD	FIELD:	CFIELD:	BEGIN-STRUCTURE	END-STRUCTURE

Floating-Point Words

>FLOAT	F>D	FDEPTH[6]	FLOOR	FTAN
D>F	F>S	FDROP[6]	FMAX	FTRUNC
F.[9]	F@	FDUP[6]	FMIN	FVALUE
F!	FABS	FE.[9]	FNEGATE	FVARIABLE
F*[7]	FACOS[8]	FEXP[8]	FOVER[6]	PRECISION
F+[7]	FASIN[8]	FFIELD:	FROT[6]	REPRESENT[9]
F-[7]	FATAN[8]	FLITERAL	FROUND	S>F
F/[7]	FALIGN	FLN[8]	FS.[9]	SET-PRECISION[9]
F0<	FALIGNED	FLOAT+	FSIN	
F0=	FCONSTANT	FLOATS	FSQRT[8]	
F<	FCOS	FLOG[8]	FSWAP[6]	

6) There is no separate stack for floating point numbers, i.e. floating point numbers are stored on the normal data stack.
7) Floating point operations are very slow on My4TH light. To increase the speed, no rounding is implemented.
8) Some arithmetic functions are very imprecise, especially as the numbers get larger.
9) Up to seven significants (digits) can be entered and printed. The largest printable number is 8388607 (23 bit).

Other Standard Words

Word	Category	Function
K	Core	Makes a copy of the index of the next higher loop. See also I and J.
L	Core	Makes a copy of the index of the next higher loop. See also I and J.
?	Programming-Tools	Display the value stored at the supplied address.
.S	Programming-Tools	Display the values currently on the data stack.
DUMP	Programming-Tools	Display memory contents.
FORGET	Programming-Tools	Delete the named word and all following words from the dictionary.
WORDS	Programming-Tools	List the definition names in the search order.
AT-XY	Facility Word Set	Position the cursor on the output device.
KEY?	Facility Word Set	Test if a key was pressed. Note: KEY? always blocks until a keystroke is received.
PAGE	Facility Word Set	Clear the screen.
MS	Facility Word Set	Waits for the specified number of milliseconds.
>=	Gforth Numeric Comparison	Compare two numbers.
<=	Gforth Numeric Comparison	Compare two numbers.
-ROT	Gforth Data Stack	Rotate data stack in the opposite direction of ROT.
RDROP	Gforth Return Stack	Remove one value from the return stack.
BOUNDS	Gforth Memory Blocks	Calculates the parameters used for loops.
CELL	Gforth Address Arithmetic	Returns the size of one cell in bytes.
UP		Returns the user memory pointer. Points to 256 bytes of memory free to use.

Special My4TH light Words

Input and output words

`I2C-START ( addr -- ack )`

Sends an I²C start condition followed by the 7-bit I²C slave address and the R/W-bit. The addr parameter is the combination of the slave address and the R/W bit, i.e. the 7-bit device address shifted one to the left, with bit 0 always set to zero. After execution, I2C-START puts the result of the operation on the data stack. The operation was successful (the device accepted the request) when ack is TRUE. Otherwise, when ack is FALSE, no I²C slave has responded to the request.

`I2C-STOP ( -- )`

Sends an I2C stop condition to the currently active device.

`I2C-SEND ( byte -- ack )`

Sends a byte to the currently active I²C slave device. I2C-SEND puts the result of the operation on the data stack, i.e. the binary value of the ack bit of the transfer. If ack is zero, the transfer was successful.

`I2C-RECV ( flag -- byte )`

Receives a byte from the currently active I²C slave device. The flag value must be zero for all bytes to be received except the last byte. A non-zero value signals to the I²C slave device that this transfer is the last (in a series of transfers), and the device then releases the SDA line so that a stop condition can be sent. That is, if only a single byte is to be transmitted, the flag must be non-zero.

`I2C? ( -- )`

Prints the device addresses of all available I²C devices on the bus.

`LED ( flag -- )`

Switch the LED of My4TH light on or off. Set the flag to 0 to turn the LED off. Set it to 1 to switch the LED on. Note that the LED is connected to the serial TXD line. When My4TH light returns to the command prompt, the LED will be switched on again as TXD is high in the idle state.

`SX ( -- width )`

Returns the width of the currently used display on the data stack.

`SY ( -- height )`

Returns the height of the currently used display on the data stack.

Binary image words

`SAVE-IMAGE ( n -- )`

Save a binary image of the RAM contents to the EEPROM, starting at destination block number n. This word effectively stores a binary image of the FORTH program, including all words and variables, in the EEPROM. Unused areas of the RAM will not be saved, thus the image in the EEPROM is as small as possible. After successful execution this word outputs the range of occupied blocks in the EEPROM.

`LOAD-IMAGE ( n -- )`

Load a binary FORTH program image from EEPROM back to RAM. n is the first block of the image in the EEPROM.

`RUN-IMAGE ( n -- )`

Load a binary FORTH program image from EEPROM back to RAM and execute it. n is the first block of the image in the EEPROM. After the image is loaded, the custom word RUN is executed to start the program. This word is defined as a combination of LOAD-IMAGE and RUN. No error is issued if the RUN word is not present in the image.

`BLOAD ( n -- )`

Load a binary program from the EEPROM into RAM. n is the first block of the program in the EEPROM. The binary program can be a FORTH library, a device driver or any other machine code program. The BLOAD word automatically performs code relocation so that it is possible to load multiple binary programs into RAM.

Miscellaneous words

`EDIT ( n -- )`

Start the simple built-in text editor. n is the number of the screen to be edited.

`RANDOM ( u1 -- u2 )`

Generates a random number in the range 0 ... u1-1. It is not required to initialize the random number generator with a seed. Entropy is generated by the timing of user input from the keyboard or RS-232.

`SYSV ( n -- addr )`

Returns a pointer to a system vector. n is the index number of the vector, and addr is the address of the cell that contains the vector. The usefulness of vectors is limited for Forth programs because Forth programs cannot access the CPU registers that contain the vector parameters.

These vectors are currently defined:

Vector	Name	Function
0	OUTPUT	Default output vector for printing characters. When this vector is set to zero (this is the default setting), the system outputs characters via RS-232 or LCD (if installed). The user can set a vector that points to a custom print routine. Note that the custom routine must not modify any CPU registers. The character to be printed is provided in the accumulator.
1	INPUT	Default input vector for reading the keyboard. When this vector is set to zero (this is the default setting), the system reads characters via RS-232 or an attached keyboard. The user can specify a vector that points to a user-defined keyboard input routine. The user-defined routine must not wait for a key to be pressed. If no key is pressed, the routine must return zero in the accumulator. If a key was pressed, the routine must return the key code in the accumulator.
2	OK	This vector points to a function that prints out the “OK” text.
3	PROMPT	Pointer to the FORTH input prompt function. The prompt function reads a string from the keyboard, evaluates the string and then outputs “OK”, “COMPILED” or an error message.
4	INPSTR	Read a text string from the keyboard. This vector can be used to retrofit a more sophisticated keyboard input function that my also provide a history function of the last few inputs.
5	GET_NUMBER	Pointer to the function that reads numbers from the command line.

Support for Adafruit Boards

Words to control simple LCD text displays

My4TH light has built-in support for LCDs of various sizes that have a controller chip that is compatible with the HD44780, and a I^2C bridge chip that is compatible with the PCF8574. Displays from 8x1 to 20x4 characters are supported.

`LCD-INIT  ( type addr ofs0 [ofs1 [ofs2 ofs3]] width height --  )`

This word initializes the LCD display. It must be executed first before any text can be output on the display. The parameters have the following meaning:

type	:	LCD controller chip type: 0 = HD44780, 1 = Groove RGB LCD (unknown controller)
addr	:	The 8-bit I^2C address of the display (the least significant bit must be zero)
ofs1	:	Memory offset of the left most position of the first LCD line (usually 0x00)
ofs2	:	Only required if LCD has 2 or more lines. Memory offset of 2nd LCD line.
ofs3	:	Only required if LCD has 4 lines. Memory offset of 3nd LCD line.
ofs4	:	Only required if LCD has 4 lines. Memory offset of 4th LCD line.
width	:	The width of the display (characters per line). Usually 16 or 20.
height	:	The height of the display (number of lines). Can be 1, 2 or 4.

`LCD-16X2  ( -- type addr 0 64 16 2 )`

Tries to detect the connected 16x2 character LCD and pushes the initialization values onto the stack. These values are the input for the LCD-INIT word. The display is searched at the I2C addresses 0x4E, 0x7C and 0x7E.

`LCD-20X4  ( -- type addr 0 64 20 84 20 4 )`

Tries to detect the connected 20x4 character LCD and pushes the initialization values onto the stack. These values are the input for the LCD-INIT word. The display is searched at the I2C addresses 0x4E, 0x7C and 0x7E.

`LCD-CLR  ( -- )`

Clears the display and moves the cursor to the home position.

`LCD-XY  ( x y -- )`

Moves the cursor to the new position specified by X and Y. The home position is 0 0.

`LCD-CH ( u -- )`

Outputs a character at the current position on the display. The cursor is moved to the right. When the rightmost position on the display is reached, the cursor jumps to the beginning of the next line (the cursor is not visible on the display).

`LCD-P ( s-addr n -- )`

Outputs a counted character string on the display. The cursor is advanced automatically.

`LCD-DC ( u0 u1 u2 u3 u4 u5 u6 u7 chr -- )`

Defines a custom character. u0 to u7 are the eight bytes that define the bitmap. u0 is the top line and u7 is the bottom line of the character. Within a byte, only bits 0 through 5 are valid. Bit 5 is the leftmost position and bit 0 is the rightmost position within the line. chr is the character number 0 - 7.

`LCD-BL ( u -- )`

Switches the display backlight on (u=1) or off (u=0).

Words to control a GPIO port expander

My4TH light has built-in support for some I^2C GPIO expander chips that are also available on Adafruit breakout boards. The supported chips are AW9523 (16 I/Os), MCP23017 (16 I/Os), PCF8574 (8 I/Os) and PCF8575 (16 I/Os).
Please note that the built-in words can only control one GPIO expander at a time, so if you want to connect two or more of these chips, you will need to write your own functions to control them.

`IOINIT  ( i2caddr chipid -- )`

Initializes the I/O board to be used. This word must be issued first before any other I/O word can be used. The parameters on the stack are the I^2C address (8-bit format with bit 0 set to zero) and the type of I/O board or chip. The following chip IDs are defined:

1 = AW9523, 2 = MCP23017, 3 = PCF8574, 4 = PCF8575

`IOCFG  ( value -- )`

Configures the direction (input or output) with a bit mask, where each bit represents one of the I/O pins. A bit set to 0 configures the corresponding pin for output, a bit set to 1 configures it for input mode.

`IOSET  ( value -- )`

Sets all digital output pins to the new state defined by the bits in the value parameter.

`IOGET  ( -- value )`

Reads all input pins at once and returns the bits as a single value on the data stack.

`IOHI  ( pin -- )`

Sets an output pin to high. The pin value ranges from 0 to 15, or 0 to 7 for the PCF8574.

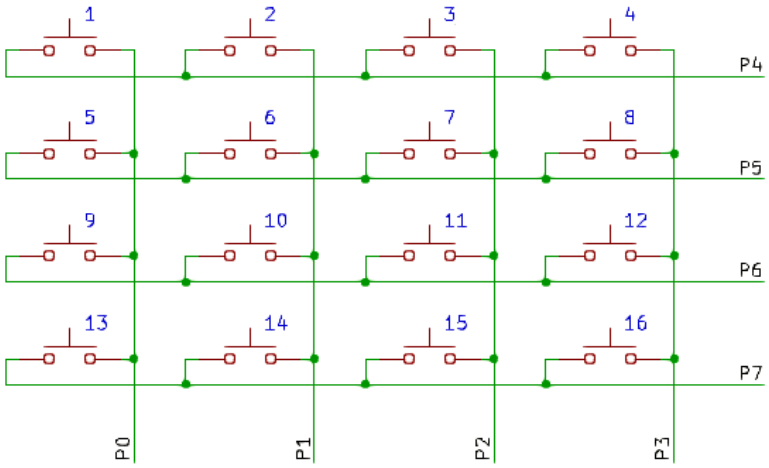
`IOLO  ( pin -- )`

Sets an output pin to low. The pin value ranges from 0 to 15, or 0 to 7 for the PCF8574.

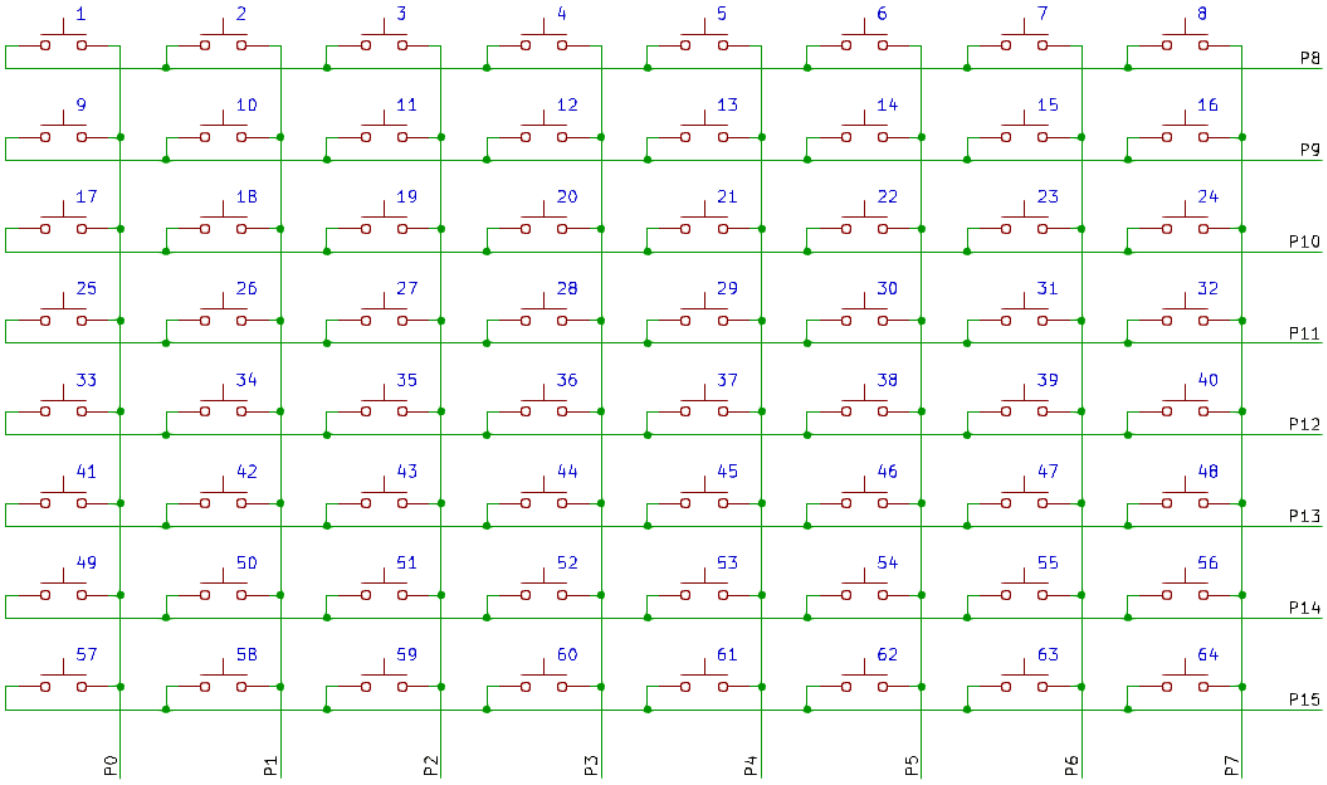
Words to read out a keyboard matrix

Since My4TH light has no GPIO to connect input keys directly, keys must be connected to a GPIO port expander. My4TH light supports the I/O port expander PCF8574 and PCF8575 for this purpose. You can connect up to 16 keys to a PCF8574 (4x4 matrix) or up to 64 keys to a PCF8575 (8x8 matrix). If you connect only 16 keys to a PCF8575, you can use the remaining 8 I/Os of the chip for general purpose I/O.

Here is an example of what a key matrix would look like. This 4x4 matrix can be connected to I/O pins P0 through P7 on a PCF8574 or PCF8575. Note that no additional pull-up resistors are required because the PCF chip has weak pull-ups built in. The blue numbers are the key codes returned by the words `KSCAN` and `KREAD` when the key is pressed:



If you need more keys, you can connect a big 8x8 matrix to the pins P0 through P15 of a PCF8575:



`KINIT  ( i2caddr matrixsize -- )`

Initializes the PCF8574/5 and the software driver for the keyboard matrix. This word must be issued before the keyboard matrix can be read. The parameters on the stack are the I^2C address (8-bit format with bit 0 set to zero) and the matrix size. The matrix size parameter can be 4 (for a 4x4 matrix) or 8 (for an 8x8 matrix).
Important: If you want to use a PCF8575 to read out a 4x4 matrix connected to P0 through P7, you must call the word `KOUT` once right after `KINIT`, otherwise the words `KSCAN` and `KREAD` won't work as expected.

`KOUT  ( value -- )`

This word is only applicable when using a PCF8575 with a 4x4 key matrix. It outputs the 8-bit value on the data stack to port pins P8 to P15 of the PCF8575.

`KSCAN  ( -- false | keycode true )`

Scans the key matrix for a pressed key. If no key is pressed, the word returns *false* on the data stack. If a key is pressed, the key code and *true* is returned. Important: As long as the key is pressed, this word returns the key code and *true*. This word does not implement key debouncing.

`KREAD  ( -- false | keycode true )`

This word uses `KSCAN` to read out the key matrix. The word returns *false* as long as no key is pressed. When a key is pressed, it returns the key code and *true* only once. To re-arm this word, the key must first be released. This word implements key debouncing.

Words to send and receive data over a SC16IS750 UART

The SC16IS750 chip is a serial UART for RS-232 communication that can be connected to the host system via I^2C or SPI. There are many sources on the Internet that offer breakout boards with this chip. Although this chip has 3.3V logic, its I^2C bus can operate at 5V, making it is easy to connect it to the My4TH light computer board.

The advantage of the SC16IS750 UART is that it has a FIFO for received data and an automatic hardware handshake control. Thus, it is possible to send and receive data at much higher baud rates than 4800 baud without the risk of losing bytes.

```
UART-INIT  ( i2caddr crystalfreq. baudrate.
             databits stopbits parity handshake -- )
```

Initializes the SC16IS750 UART chip. The following parameters must be provided on the data stack:

i2caddr	:	8-bit I^2C-address of the UART (bit 0 of the address must be zero).
crystalfreq.	:	Frequency of the on-board crystal. This value is a double integer. For example, a common value is 14.7456 MHz. In this case, put the number 14745600. on the stack.
baudrate.	:	The desired baud rate (double integer). For example, put 115200. on the stack.
databits	:	The number of databits to transmit per byte. Values from 5 to 8 are possible.
stopbits	:	1 or 2 stop bits are supported.
parity	:	0 = no parity, 1 = odd parity, 2 = even parity
handshake	:	0 = no RTS/CTS handshake, 1 = automatic RTS/CTS handshake

```
UART!  ( data -- )
```

Sends one byte over the UART. If hardware handshake is enabled, this word waits until the receiver is ready again before sending the data.

```
UART@  ( -- data )
```

Receives a byte from the UART. If no byte is available in the input FIFO, this word waits until a byte is received. See also the word `UART?`.

```
UART?  ( -- true|false )
```

Tests if data has been received from the UART. The word puts true on the stack if at least one byte is available in the RX FIFO. Use the word `UART@` to get the byte from the FIFO.