français

STM8 Tiny BASIC version 2.5 language refenrece

main index

---

main index

Data types

The only data type supported is 24 bits signed integers in range **-8388608...8388607**.

For PRINT and INPUT commands and only for these 2 commands quoted string are supported.

Also **\c**, i.e. a backslash followed by an ASCII character is also supported by **PRINT** and as parameter for some commands.

It is also possible to print a character from its ASCII code usint CHAR function. This function can also be used in expressions. When an expression using CHAR function is given as a parameter to command PRINT the CHAR function must not be the first factor of the expression.

```
 >? char(33),2*char(33)
!   66

>? char(33)*2
!
```

```
run time error, syntax error
    0 ? CHAR ( 33 ) *

>
```

---

main index

Variables

In Tiny BASIC there was only 26 variables given the names of alphabet letters {**A..Z**}. This hold in this BASIC but more variables can be created using keyword DIM. The maximum length of the dynamic variables is 15 characters.

Array

There is a predefine single dimension array named **'@'**. A minimum 10 elements are reserved for this array but its actual size depend on program size and the number of dynamic variables and constants defined in the program. The leftover free space is given to **@** array.

The first indice of the array is **1** and the last is known by invoking UBOUND function.

Labels

Labels are name used as first element of a BASIC line to identify a GOSUB or GOTO target.

Labels name have the same format as dynamic variables and constants names.

1. Maximum 15 characters
2. Begin with a letter
3. Can include digits,'**.**','**_**' and '**?**' characters

---

main index

Arithmetic expressions

There is 5 arithmetic operators, plus parenthesis. Order of priority is:

1. '**(**'expression'**)**'
2. '**-|+**' unary minus or plus.
3. '*****' multiplication, '**/**' division and '**%**' modulo
4. '**+**' addition and '**-**' substraction

Quotient is rounded toward zero.

Relational operators

Expressions can be compared for size with the following operators. Relations return -1 for true and 0 for false.

1. **'>'** True if left is greater than right expression.

2. **'<'** True if left is smaller than right expression.
3. **'>='** True if left is greater or equal to right expression.
4. **'<='** True if left is smaller or equal to right expression.
5. **'='** True if both expressions have the same value.
6. **'<>'** or **'><'** True if expressions have a different value.

Boolean operators

The **NOT**,**AND**,**OR** and **XOR** are binary operators like their machine level equivalent. But as relations return only **0|-1** they are effective as boolean expression operators used as condition in IF and UNTIL statement.

---

main index

Syntax

The code use for writing program is ASCII.

Each program line start with a line number in range {1..32767} followed optionally by a label then a list of commands separated by **':'** character. If no line number is given the line is compiled and interpreted immedialety.

```
>let t=ticks:for i=1 to 10000: let a=10:next i : ? ticks-t
400
```

A command is followed by its parameters list. The comma separate the parameters. Function parameters must be between parentheses but functions without parameters don't require parenthesis.

As names can comprise digits it is important to put a space after a command or function name if it is followed by un number.

```
?3*5  ' don't need space after '?'.
 15

> for i=1to 100 :? i;: next i  ' need a space between 'to' and '100'
```

Names can be entered in lower case and are converted to upper case. The language is not case sensitive.

The command PRINT can be replacec by ?.

The keyword REM can be replaced by a tick **'**.

End of line mark the end of command. There is no command continuation on next line.

---

main index

Numeric bases

Integers can be typed in 3 numeric bases.

1. Decimal: ['-'|'+']*digit+
    - -1343
    - +4677
    - 987
2. Hexadecimal: ['-'|'+']*hex_digit+
    - -$ffe
    - +$134a
    - $A5a5
3. binary: ['-'|'+']*['0'|'1']+
    - -%101
    - +%1011
    - %11111101

Integers are printable only in decimal or hexadecimal.

---

main index

Command line

At startup a beep is sounded and system information is displayed on terminal. followed by **>** which is the prompt.

```
Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.5R1

>
```

From there the user can enter direct commands or edit program lines. A line is limited to 79 characters and edition is terminated by **ENTER** key. When **ENTER** is pressed the input text is compiled to a tokens list. If there no line number this tokens list is executed immediately.

Otherwise the line is inserted in program space in RAM area.

- Line numbers are limited to range {1...32767}.
- if an existing line with the same number as the last edited one exist the new one replace it.
- If the new line as no text an a line with that number exist then it is erased.
- Lines are inserted sorted in increasing line number.

Some commands can only be used in direct mode others only inside programs. An error is displayed if a command is used in bad context.

The program in RAM is lost each time then MCU is resetted but it can be save in FLASH memory using SAVE commmand.

On Linux systems it is possible to write programs in a text editor on the PC the send it to the board using send.sh script.

Editing hot keys

The following hot keys can be used while entering a text line in terminal.

key	function
BS	Delete character left of cursor
ln CTRL+E	**ln** being a line number this display that line for editing.
CTLR+R	Redisplay last line entered.
CTRL+D	Delete currently edted line.
HOME	Move cursor to beginning of line.
END	Move cursor to end of line.
left arrow	Move cursor left one character.
right arrow	Move cursor right one character.
CTRL+O	Toggle between insert and overwirte mode. Cursor change shape.

main index

Commands and functions reference

{C,P} after command name indicate which context is valid for this command. **C** for command line and **P** for program.

main index

Vocabulary index

name	description
ABS	function that return absolute value.
ADCON	Power analog to digital converter.
ADCREAD	Read analog input pin.
ALLOC	Allocate space on data stack.
AND	Boolean operator.
ASC	Return ASCII value of a character.
AUTORUN	Enable or disable program auto run.
AWU	Put board in sleep mode for some msec.

name	description
BIT	Compute 2^bit.
BRES	Reset a bit in a peripheral register.
BSET	Set a bit in a peripheral register.
BTEST	Return the state of a bit in a peripheral register.
BTOGL	Toggle a bit in a peripheral register.
BUFFER	Allocate a buffer in RAM.
BYE	But board in sleep mode.
CHAIN	Chain program execution.
CHAR	Return the character corresponding to ASCII code.
CONST	Keyword to define symboli constants.
CR1	Return offset of GPIO CR1 register.
CR2	Return offset of GPIO CR2 register.
DATA	keyword that introduce a data line.
DDR	Return the offser of GPIO DDR register.
DEC	Define decimal base as output for PRINT command.
DIM	Keyword used to define dynamic variables.
DIR	Lis programs saved in FLASH memory.
DO	Keyword to introduce a DO..UNTIL control structure.
DREAD	Read a digital pin.
DROP	Drop top element of data stack.
DWRITE	Write a digital pin.
EDIT	Load in RAM a program saved in FLASH for edition.
EEFREE	Return EEPROM free address.
EEPROM	REturn EEPROM start address.
END	Terminate program execution.
ERASE	Erase a program saved in FLASH memory.
FCPU	Set MCU operating frequency.
FOR	Keyword that start a FOR..NEXT control structure
FREE	Return free RAM bytes.
GET	Read a character in variable, not wait.

name	description
GOSUB	Subroutine call.
GOTO	Unconditional jump.
HEX	Set hexadecimal base for PRINT command.
I2C.CLOSE	Close I2C peripheral.
I2C.OPEN	Open I2C peripheral.
I2C.READ	read data from I2C peripheral.
I2C.WRITE	Write data to I2C peripheral.
IDR	Return GPIO IDR register offset.
IF	Keyword for conditional execution.
INPUT	Input number in a variable.
IWDGEN	Enable Independant Watchdog Timer.
IWDGREF	Refresh IWDG before it expire.
KEY	wait a key from termnal.
KEY?	Check if there is a key waiting in terminal queue.
LET	Keyword to initialize variables.
LIST	List program in RAM.
LOG2	Return base 2 log of an integer.
LSHIFT	Shift left an integer.
NEW	Clear RAM memory from program.
NEXT	Close FOR..NEXT loop.
NOT	Boolean NOT operator.
ODR	Return GPIO ODR register offset.
ON	Keyword for selective GOTO or GOSUB.
OR	Boolean operator OR.
PAD	Return address of 128 bytes working buffer.
PAUSE	Suspend execution for some milliseconds.
PEEK	Return byte value at some address.
PICK	Return integer from data stack at selected position.
PINP	Read one of Arduino digital pin.
PMODE	Set OUT

name	description
POKE	Set byte value at some address.
POP	Function that remove and return top of data stack .
POUT	Change state of Arduino digital pin.
PRINT or ?	Print, string, charater or integer to terminal.
PORTA	Return base address GPIO A
PORTB	Return base address GPIO B
PORTC	Return base address GPIO C
PORTD	Return base address GPIO D
PORTE	Return base address GPIO E
PORTF	Return base address GPIO F
PORTG	Return base address GPIO G
PORTI	Return base address GPIO I
PUSH	Push integer on data stack.
PUT	Put an integer on data stack at selected position.
READ	Read in a variable data item from DATA line.
REBOOT	Reinitialize MCU.
REM ou '	Start a comment.
RESTORE	Reinitialize DATA pointer.
RETURN	Exit from subroutine.
RND	Return a random number.
RSHIFT	Shift right an integer.
RUN	Execute program.
SAVE	Save program in RAM to FLASH memory.
SIZE	Display address and size of active program.
SLEEP	Put MCU in low energy mode.
SPIEN	Enable SPI peripheral.
SPIRD	Read data from SPI peripheral.
SPISEL	Select SPI channel.
SPIWR	Write data to SPI peripheral.
STEP	Keyword used in FOR..NEXT loop to set increment.

name	description
STOP	Stop program execution without resetting it.
TICKS	Return milliseconds coutn since power up.
TIMEOUT	Return true if TIMER as expired.
TIMER	Set TIMER.
TO	Keyword used in FOR..NEXT loop to set limit.
TONE	Tone generator.
UBOUND	Return last indice of @ array.
UFLASH	Return first free block address of FLASH memory.
UNTIL	Keyword that close DO..UNTIL control loop.
USR	Function to call machine code routine.
WAIT	Monitor some register state for expected change.
WORDS	List vocabulary with token index.
WRITE	Write data to FLASH or EEPROM.
XOR	Boolean operator exclusive OR.

main index

ABS(*expr*) {C,P}

This function return the absolute value of the expression.

```
>? abs(-45)
45
```

index

ADCON 0|1 [,divisor]

This command power on|off the analog/digital converter. **1** for **ON**, **0** for **OFF*
divisor parameter determine converter convertisseur et doit-être un entier dans l'intervalle {0..7}. This divisor is applied to Master clock Fosc. 11 clock cycles are required per convertion. If divisor is not given it is consired to be **0**.

paramer	divisor	Fconv
0	2	8Mhz
1	3	5,33Mhz

paramer	divisor	Fconv
2	4	4Mhz
3	6	2,66Mhz
4	8	2Mhz
5	10	1,6Mhz
6	12	1,33Mhz
7	18	0,89Mhz

```
>adcon 1,0 ' enable ADC maximum frequency

>?adcread(0) 'read channel 0
 757

>adcon 0 ' disable ADC
```

Disabling ADC reduce MCU power consumption.

index

ADCREAD(channer) {C,P}

Read one of 7 analog inputs on CN4. Pinout is different for each board.

MCU channel	NUCLEO-8S208RB CN4:pin	NUCLEO-8S207K8 CN4:pin
0	6	12
1	5	11
2	4	10
3	3	9
4	2	7
5	1	8
12	CN9:16	6

```
>adcon 1,0 ' active ADC fréquence maximale

>?adcread(0) 'Lecture canal 0
 655
```

index

ALLOC n {C,P}

Reserve *n* slots on data stack. These slots can be used as temporary or local variables in subroutines. See also PICK, PUT and DROP.

index

expr1|rel1|cond1 **AND** *expr2|rel2|cond2* {C,P}

Boolean operator to insert between two expressions or relations. This is a bit to bit **AND** operator.

When these allocated slots are no more used never forget to free them with DROP.

See also NOT,OR,XOR.

```
>a=2 ? a
   2

>b=4 ? b
   4

>if a>=2 and b<=4 ? "true"
   true

>
```

See also DROP,PICK,PUT,PUSH,POP

index

ASC(*string*|\c) {C,P}

This function return the ASCII value for first character of a string or of single character. See also CHAR function which is the opposite of this one.

```
>? asc("AB")
65

>? asc(\Z)
90

>
```

index

AUTORUN \C|name {C}

This command enable or disable program auto execution at board power up or rinitialisation.

- AUTORUN name search for a file with that name and if found set it as autorun program.
- AUTORUN \C cancel any autorun.
- CTRL-Z can also be used to cancel an autorun program if stuck in an infinite loop.

index

AWU *expr* {C,P}

This command put MCU in low power mode **(HALT)** for some amount of milliseconds defined by *expr*. After wakeup the program continue execution after this command. The command name come from the peripheral used **A**uto-**W**ake**U**p.

expr must be in range {1..32720}. The maximum delay is around 30.7 secondes.

```
>awu 1  ' 1 millisecond

>awu 30720 ' 30.7 seconds

>
```

index

BIT(*expr*) {C,P}

This function return 2^*expr*, i.e. 2 power of *expr* which must bit in the range {0..23}.

```
>for i=0 to 23: ? bit(i);:next i
1 2 4 8 16 32 64 128 1 2 4 8 16 32 64 128 65536 131072 262144 524288
1048576 2097152 4194304 -8388608

> bset PORTC,bit(5) ' Turn on user LED on board.
```

index](#index)

BRES addr,mask {C,P}

This command reset one or more bits at **addr**. Each bit of *mask* that are at **1** are reset at target address. The address can be RAM or register.

```
>bres PORTC,bit(5) ' turn off user LED on board.
```

index

BSET addr,mask {C,P}

This command set one or more bits at *addr*. Each bit of *mask* that is at **1** is set at target address. The address can be RAM or register.

```
>bset $500a,&100000 ' turn on user LED on board.
```

index

BTEST(addr,bit) {C,P}

This function return the state of a single bit at *addr*. *bit* is the position of bit to be tested in range {0..7}.

```
>? btest($50f3,5) ' BEEP_CSR enable bit
   0
```

index

BTOGL addr,mask {C,P}

This command toggle one or more bits at *addr*. bits of *mask* that are at **1** are inverted in target address. The address can be RAM or register.

```
>btogl PORTC,32 ' toggle user LED state.
```

index

BUFFER *name*, *size* {P}

This command reserve buffer space in RAM. This buffer can written to with POKE and read from with PEEK.

For usage examples look at i2c_eeprom.bas and i2c_oled.bas programs.

- *name* is the name of variable holding buffer address.
- *size* is in BYTES.

Size is limited by free RAM leftover by the program.

```
>list
   10  BUFFER BUF , 16: ' create buffer
   20  FOR I= BUF  TO I+ 15 POKE I, RND( 255) NEXT I : ' write to buffer
   30  FOR I= BUF  TO I+ 15 PRINT PEEK( I); NEXT I : ' read from buffer
program address:  $90, program size:  108 bytes in RAM memory

>run
```

```
 215 248 88 147 11 229 252 86 214 192 27 194 136 88 227 115
>
```

index

BYE {C,P}

This command place the MCU in HALT mode from which only a reset can reset it.

index

CHAIN name[,line#] {P}

This command is used to run a progrm stored in file system from the actual running program.

- *name* is the program file name.
- *line#' is optional and indicate at which line the execution should start.

When the chained program leave execution continue at the calling program after the **CHAIN** command.

A chained program can itself use **CHAIN** to execute another program file. The depth of chaining is limited by stack size.

index

CHAR(*expr*) {C,P}

This function return the ASCII character corresponding code *expr* which must be in the range {0..127}.

```
>for a=32 to 126:? char(a);:next a
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
>
```

index

CONST name=value [,name=value] {P}

This keyword is used to define symbolic constants. The list of constants to be defined are separated by comma **','**.

- **name** is constant name.
- **value** is a constant expression.

```
>list
   5 ' Test symbolic constant speed in comparison to literal constant.
  10 CONST TEST = 1024
  20 ? "assign a varaible."
```

```
   24 ? "literal constant: " ;
   30 LET T = TICKS : FOR I = 1 TO 10000 : LET A = 20490 : NEXT I
   32 ? TICKS - T ; "MSEC."
   34 ? "symbolic constant: " ;
   40 LET T = TICKS : FOR I = 1 TO 10000 : LET A = TEST : NEXT I
   44 ? TICKS - T ; "MSEC."
   50 CONST LED = 20490
   60 ? "Test toggling user LED on board."
   64 ? "Literal constant: " ;
   70 LET T = TICKS : FOR I = 1 TO 10000 : BTOGL 20490 , 32 : NEXT I
   72 ? TICKS - T ; "MSEC."
   74 ? "Symbolic constant: " ;
   80 LET T = TICKS : FOR I = 1 TO 10000 : BTOGL LED , 32 : NEXT I
   90 ? TICKS - T ; "MSEC."
program address: $91, program size: 496 bytes in RAM memory

>run
assign a varaible.
literal constant: 418 MSEC.
symbolic constant: 541 MSEC.
Test toggling user LED on board.
Literal constant: 587 MSEC.
Symbolic constant: 714 MSEC.

>
```

index

CR1 (C,P)

This constant is the offset of **CR1** register from GPIO base address. It must be added to **PORTx** constant to be accessed.

In input mode this register configure pull-up and in output mode it select between *push-pull* and *open-drain*.

See also ODR,IDR,DDR,CR2

index

CR2 {C,P}

This constant is the offset of **CR2** register from GPIO base address. It must be added to **PORTx** constant to be accessed.

In input mode it is used to enable or disable external interrupt on pin. In output mode it is used to limit port slew ratte (i.e. toggling speed).

See also ODR,IDR,DDR,CR1

index

DATA {P}

This keyword is used to declare a line containing only data. The interpreter skip over data lines. The data is accessed using READ function. Each the a data item is read the data pointer is moved to next item. Reading data after the last item is a fatal error. Note that contrary to Microsoft BASIC this is a function not a command. It doesn't accept any parameter.

See also RESTORE.

```
>list
    5 ' Play a tune from score in DATA lines
   10 RESTORE
   20 DATA 440,250,440,250,466,250,523,250,523,250,466,250,440,250
   30 DATA 392,250,349,250,349,250,392,250,440,250,440,375,392,125
   40 DATA 392,500
   50 FOR I =1TO 15:TONE READ ,READ :NEXT I
```

index

DDR {C,P}

This constant is the offset of DDR register from GPIO base address. It must be added to **PORTx** constant to be accessed.

This register is used to set GPIO pin as input or output.

See also ODR,IDR,CR1,CR2

```
>bset portc+ddr,bit(5) ' set user led pin as output

>
```

index

DEC {C,P}

This command is used to set the number printing format to decimal. It is the default format at startup.

See also HEX.

```
>HEX:?-10:DEC:?-10
$FFFFF6
-10
```

index

DIM var_name[=expr][,var_name[=expr]] {P}

This keyword is used to define symbolic variables in extra to the 26 Tiny BASIC variables {A..Z}.

- *var_name* is variable name and must be at least 2 characters beginning with a letter. The first letter can be followed by **'_','.','?'** and letters. The maximum length is 15 characters.
- *expr* is optional and used to initialize the variable. If not present the variable is initialized to **0**.
- The comma **','** is used as list separator.

index

DO {C,P}

Keyword used to introduce a **DO..UNTIL *condition* ** loop. The instructions inside the loop are executed until *condition* become true.

See also UNTIL.

```
>li
   10 A = 1
   20 DO
   30 PRINT A;
   40 A =A + 1
   50 UNTIL A > 10

>run
   1 2 3 4 5 6 7 8 9 10
```

index

DIR {C}

This command display the list of program saved in file system. Program saved with command SAVE are run in place.

See also SAVE,ERASE and AUTORUN.

```
>>DIR
$BB04 84 bytes,BLINK
$BB84 218 bytes,HYMNE
$BC84 127 bytes,FIBONACCI
```

index

DREAD *pin*

This function return the state of a digital pin which as been defined as input with PMODE. The value returned is either **0** or **1**. Tables below give pinout for each board.

NUCLEO-8S208RB

MCU PORT	Arduino Dx	board con
PD6	D0_RX	CN7:1
PD5	D1_TX	CN7:2
PE0	D2_IO	CN7:3
PC1	D3_TIM	CN7:4
PG0	D4_IO	CN7:5
PC2	D5_TIM	CN7:6
PC3	D6_TIM	CN7:7
PD1	D7_IO	CN7_8
PD3	D8_IO	CN8:1
PC4	D9_TIM	CN8:2
PE5	D10_TIM_SPI_CS	CN8:3
PC6	D11_TIM_MOSI	CN8:4
PC7	D12_MISO	CN8:5
PC5	D13_SPI_CK	CN8:6
PE2	D14_SDA	CN8:9
PE1	D15_SCL	CN8:10

NUCLEO-8S207K8

MCU PORT	Arduino Dx	board con
PD5	D0_TX	CN3:1
PD6	D1_RX	CN3:2
PD0	D2	CN3:5
PC1	D3	CN3:6
PD2	D4	CN3:7
PC2	D5	CN3:8
PC3	D6	CN3:9
PA1	D7	CN3:10
PA2	D8	CN3:11
PC4	D9	CN3:12

MCU PORT	Arduino Dx	board con
PD4	D10	CN3:13
PD3	D11	CN3:14
PC7	D12	CN3:15

```
10 PMODE 5,PINP
20 ? DREAD(5)
```

index

DROP *n* {C,P}

This command free *n* top slots from data stack.

See ALLOC,PICK,PUT,PUSH,POP.

index

DWRITE *pin*,*level*

This command change the state of a digital output pin defined as output by command PMODE.

- *pin* is one of available **Dx** on board.
- *level* is **1** or **0**.

See also PMODE, DREAD.

```
10 PMODE 10,POUT ' configure D10 as output
20 DWRITE 10, 0  ' set pin to 0 level
```

index

EDIT name {C}

Copy a program file from FLASH to RAM for modification.

```
>dir
$BB04 84 bytes,BLINK
$BB84 218 bytes,HYMNE

>edit blink

>list
    1 BLINK
```

```
     5 ' Blink LED2 on card
    10 DO BTOGL PORTC , BIT ( 5 ) PAUSE 500 UNTIL KEY?
    20 LET A = KEY
    30 BRES PORTC , BIT ( 5 )
    40 END
program address: $91, program size: 84 bytes in RAM memory

>
```

index

EEFREE {C,P}

This function return first free EEPROM address. The EEPROM is scanned from start address until 8 consecutives **0** values are found. The EEPROM is consedered free from that first zero to end.

See also AUTORUN,EEPROM.

```
>hex ? eeprom
$4000

>autorun blink

>? eeprom
$4000

>for i=EEPROM to i+15:? i;peek(i):next i
$4000 $41
$4001 $52
$4002 $BB
$4003 $0
$4004 $0
$4005 $0
$4006 $0
$4007 $0
$4008 $0
$4009 $0
$400A $0
$400B $0
$400C $0
$400D $0
$400E $0
$400F $0

>? eefree
$4003

>
```

index

EEPROM {C,P}

Return the base address of EEPROM.

See also AUTORUN,EEFREE.

```
>hex:? eeprom,peek(eeprom)
$4000   $41

>
```

index

END {C,P}

This keyword end program. It can be place anywhere in a program.

See also STOP

```
>list
   10 LET A = 0
   20 LET A = A + 1
   30 ? A ; : IF A > 100 : END
   40 GOTO 20
program address: $91, program size: 52 bytes in RAM memory

>run
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101
>
```

index

ERASE \E|\F|NAME {C}

This command is used to erase persistant memory, FLASH or EEPROM.

- *E* erase all EEPROM
- *F* erase all FLASH memory starting at **app_space**, i.e. after system firmware.
- *NAME* erase a specific program file.

See also SAVE,DIR

index

FCPU *integer*

This command set CPU clock frequency without affecting peripheral clock which stay at 16Mhz.

- *integer* in range **{0..7}**, Fcpu=16Mhz/2^7.

Reducing CPU clock frequency reduce energy consumption.

```
>fcpu 7 ' 125Khz

>let t=ticks: for i=1 to 10000:next i: ? ticks-t;" msec"
18140  msec

>fcpu 0 ' 16 Mhz

>let t=ticks: for i=1 to 10000:next i: ? ticks-t;" msec"
97  msec

>
```

index

FOR *var=expr1* TO *expr2* [STEP *expr3*] NEXT *var* {C,P}

Keyword **FOR** intialiaze a counted loop that exit when the control variable pass the limit.

- *var* is the control variable {A..Z}.
- *expr1* is initial value of variable.
- TO keyword to introduce loop limit.
- *expr2* limit value.
- STEP keyword introduce the increment applied at variable at end of each loop.
- *expr3* increment value.
- NEXT keyword that close the loop. NEXT apply the increment and check for limit. if limit is crossed over the loop exit.
- *var* same as control variable.

A FOR..NEXT loop can span many lines of codes except for the initialization which must me on the same line.

FOR..NEXT loops can be nested.

```
>list
    5 'multipliation table 1..10
   10 FOR A = 1 TO 10
   20 FOR B = 1 TO 10
   30 ? A * B ,;
   40 NEXT B : ?
   50 NEXT A
program address: $91, program size: 91 bytes in RAM memory
```

```
>run
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100

>
```

index

FREE {C,P}

This function return size of free RAM in BYTES.

```
>new

>? free
5561

>10 ? "hello world!"

>? free
5542

>
```

index

GET *var*

This command read a character in a variable from terminal but contrary to KEY it doesn't wait for it. If no character is available when invoked it return **0**.

See also KEY,KEY?

```
10 PRINT "Press a key to end.\n" : PAUSE 400
20 DO ? "hello ";:  GET A: UNTIL A<>0
```

index

GOSUB *line#|label* {P}

Subroutine call.

- *line#* is the line number where the subroutine is located.
- *label* placed at beginning of line can be used as subroutine name instead of line number. label name obey same rules as variables and constants names.

```
>li
5 ' test GOSUB with line# and label
10 GOSUB 100
20 GOSUB LBL1
30 END
100 ? "GOSUB line# works!" return
200 LBL1 ? "GOSUB label works!" return

>run
GOSUB line# works!
GOSUB label works!

>
```

index

GOTO *line#|label* {P}

This keyword do an unconditionnal jump to some other program line.

- *line#* is target program line.
- *label* is a label at beginning of a line used as GOTO target.

```
>li
    5 ' test GOTO avec line# et label
   10 GOTO 100
   20  LBL1 PRINT "GOTO label works!"
   30 END
  100 PRINT "GOTO line# works!"GOTO LBL1
program address:  $80, program size:  119 bytes in RAM memory

>run
GOTO line# works!
GOTO label works!

>
```

index

HEX {C,P}

This command select integer output format for PRINT command.

See also DEC.

```
>HEX ?-10 DEC:?-10
$FFFFF6
  -10
```

index

I2C.CLOSE {C,P}

This command disable the I2C peripheral. The I2C peripheral is a 2 wires communication device. The alternate function 6 must be programmed in **OPT2** register and MCU rebooted before using this peripheral.

See also I2C.OPEN,I2C.READ,I2C.WRITE

index

I2C.OPEN *freq* (C,P)

This command enable I2C periphal. The I2C peripheral is a 2 wires communication device. Good examples of usage are i2c_eeprom.bas and i2c_oled.bas.

- *freq* in KiloHertz of communication speed. Usually 100 or 400 .

Pin out for each board.

SIGNAL	MCU PORT	NUCLEO-8S208RB CON	NUCLEO-8S207K8 CON
SCL	PB4	A1 (CN4:1)	A5 (CN4:8)
SDA	PB5	A0 (CN4:2)	A4 (CN4:7)

This peripheral is available as an alternate function. The **OPT2** bit 6 must be set and MCU rebooted before using it. It can be done on command line. It need to be done only once. It is persistant unless the device is reprogrammed.

```
>LET A=PEEK($4803) OR 64:WRITE $4803,A:REBOOT ' connect I2C to pins


Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.5R1

>
```

To disconnet **I2C** alternate function:

```
>LET A=NOT 64 AND PEEK($4803):WRITE $4803,A: REBOOT ' disconnect I2C from
pins.


Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.5R1

>
```

See also I2C.CLOSE,I2C.READ,I2C.WRITE

index

I2C.READ *dev_id*,*count*,*buf*,*stop* {C,P}

This command read data from I2C peripheral in a buffer.

- *dev_id* is the 7 bit address of device to read.
- *count* How many bytes to read.
- *buf* buffer address to receive bytes.
- *stop* Take **0** or **1**. **0** -> free bus after transaction. **1** keep hold on bus after transaction.

For usage example see i2c_eeprom.bas.

See also I2C.CLOSE,I2C.WRITE,I2C.OPEN

index

I2C.WRITE *dev_id*,*count*,*buf*,*stop* {C,P}

This command is used to write data to **I2C** device. See i2c_eeprom.bas and i2c_oled.bas usage examples.

- *dev_id* device 7 bits address.
- *count* number of bytes to be written.
- *buf* buffer address containing data to be transmitted.
- *stop* Take **0** or **1** value. **0** -> free bus after transaction. **1** hold bus after transaction.

See also I2C.CLOSE,I2C.READ,I2C.OPEN

index

IDR {C,P}

This constant is the offset of register **IDR** from **PORTx** address. To be accessed it is added to **PORTx** value.

GPIO port use 5 registers:

- ODR *Output Data Register*, offset 0

- IDR *Input Data Register*, offset 1
- DDR *Data Direction Register*, offset 2
- CR1 *Control Register 1, offset 3
- CR2 *Control Register 2, offset 4

```
>? "Nucleo board user LED ODR address: " PORTC+ODR
Nucleo board user LED ODR address:20490

>
```

index

IF *condition* : cmd [:cmd]* {C,P}

This keyword is used for conditionnal execution. The commands that follow the condition on the same line are executed only if *condition* is true.

- *condition* can be any integer expression, comparison or boolean expression.
- *cmd [:cmd]** is list of commands to be executed if *condition* is true.

```
>a=5%2:if a:?"vrai",a
vrai   1

>if a>2 : ? "vrai",a

>
```

index

INPUT [*string*]*var* [,[*string*]*var*]+ {C,P}

This command is used to prompt user to enter some integer value.

- *string* optional prompt string
- *var* variable to store inputted value.
- More than 1 value can be queried separated by a comma.

```
>list
    5 ' test INPUT command
   10 INPUT "age? " A , "sex(1=M,2=F)? " S
   14 IF A = 0 : END
   20 IF S = 1 ? "man " ; : GOTO 40
   30 ? "woman " ;
   40 IF A > 59 : ? "babyboomer" : GOTO 10
```

```
   50 ? "still young" : GOTO 10
program address: $91, program size: 162 bytes in RAM memory

>run
age? :60
sex(1=M,2=F)? :1
man babyboomer
age? :40
sex(1=M,2=F)? :2
woman still young
age? :0
sex(1=M,2=F)? :0

>
```

index

IWDGEN *expr* {C,P}

This command enable the *Independant WatchDog timer*.

- *expr* in the range {1.16383} is delay for watchdog to expired and trigger an MCU reboot. To avoid MCU reboot the IWDGREF command must be called before this delay.

16383 value is about 1 second.

```
>li
    5 ' indepencdent watchdog timer test
   10 IWDGEN 16383 ' enable **IWDG** with 1 second delay
   20 IF KEY?  GOTO 40
   30 PRINT \.,:PAUSE 100:IWDGREF ' refresh **IWDG** before it expire.
   34 GOTO 20
   40 PRINT "\nThe IWDG will reset MCU in 1 second ."
program address:  $80, program size:  225 bytes in RAM memory

>run
.................
The IWDG will reset MCU in 1 second .

> 

Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.0

>
```

index

IWDGREF {C,P}

This command is used to reset *IWDG* before its delay to avoid MCU reboot.

See also IWDGEN.

index

KEY {C,P}

This function wait for a character from terminal and return its integer value.

```
>do let a=a+1 until key? : ? a,char(key)
53266   q

>
```

See also KEY?,CHAR

index

KEY? {C,P}

This function return **TRUE (-1)** if a character is available in terminal reception queue. If none in queue return **FALSE (0)**.

```
>do LET A=A+1 until key? : ? a, char(key)
  -1 v

>
```

See also KEY,CHAR

index

LET *var*=*expr* [,var=expr] {C,P}

This keyword is used to initialize variables. More than one variable can be initialize in the same command provide they are separated by comma.

- *var* is variable to initialize, may be a single letter variable, a symbolic one or an array element.
- *expr* may be integer expr, relation or boolean condition.

```
>LET A=24*2+3:?a
51
>LET A=31416, b=2*A:?B
62832
>LET C=-4*(a<51):?C
0
```

```
>LET @(3)=24*3

>?@(3)
72

>
```

index

LIST [*line_start*][,*line_end*] {C}

This command print on terminal the listing of active program. This program can be in RAM or in FLASH depending which is active.

- *line_start* start listing from this line or next above if doesn't exist.
- *line_end* end listing at this line or nearest below if doesn't exist.

```
>list
    5 ' test INPUT command
   10 INPUT "age? " A , "sex(1=M,2=F)? " S
   14 IF A = 0 : END
   20 IF S = 1 ? "man " ; : GOTO 40
   30 ? "woman " ;
   40 IF A > 59 : ? "babyboomer" : GOTO 10
   50 ? "still young" : GOTO 10
program address: $91, program size: 162 bytes in RAM memory

>list 10-30
   10 INPUT "age? " A , "sex(1=M,2=F)? " S
   14 IF A = 0 : END
   20 IF S = 1 ? "man " ; : GOTO 40
   30 ? "woman " ;
program address: $91, program size: 162 bytes in RAM memory

>list -30
    5 ' test INPUT command
   10 INPUT "age? " A , "sex(1=M,2=F)? " S
   14 IF A = 0 : END
   20 IF S = 1 ? "man " ; : GOTO 40
   30 ? "woman " ;
program address: $91, program size: 162 bytes in RAM memory

>list 10-
   10 INPUT "age? " A , "sex(1=M,2=F)? " S
   14 IF A = 0 : END
   20 IF S = 1 ? "man " ; : GOTO 40
   30 ? "woman " ;
   40 IF A > 59 : ? "babyboomer" : GOTO 10
   50 ? "still young" : GOTO 10
program address: $91, program size: 162 bytes in RAM memory
```

```
>
```

index

LOG2(*expr*) {C,P}

This function return the base 2 logarithm of *expr*. The logarithm is truncate toward zero.

```
>i=1 do ? log2(i),:i=i*2 until i=$400000
   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14
15   16   17   18   19   20   21
>
```

This function is the inverse of BIT.

```
>? log(bit(7))
   7
```

index

LSHIFT(*expr1*,*expr2*) {C,P}

This function shift left *expr1*, *expr2* bits the least be being replaced by **0**.

```
>? lshift(1,15)
 32768

>? lshift(3,2)
12

>
```

See also RSHIFT

index

NEW {C}

Clear program from RAM.

index

NEXT var {C,P}

This keyword is part of **FOR..NEXT** loop. It does the variable increment and check for limit crossover.

See also FOR, TO,STEP.

index

NOT *expr* {C,P}

Unary boolean operator. Take the value of *expr* and invert all bits. It as the highest priority of all boolean operators.

See also AND,OR,XOR.

```
>hex

>? not 0
$FFFFFF

>? not $ffffff
  $0

>? not 5
$FFFFFA

>? not $fffffa
  $5

>
```

index

ODR {C,P}

This constant is the offset of **ODR** register from **PORTx** constant. To access this register its value must be added to **PORTx** value.

```
>bset portc+odr,bit(5) ' turn on user LED

>bres portc+odr,bit(5) ' turn off user LED

>
```

See also IDR,DDR,CR1, CR2.

index

ON *expr* GOTO|GOSUB *target_list*

This keyword is used as a selector for GOSUB or GOTO.

- *expr* to be evaluate to select the target in *target_list*

- *target_list* comma separated list of line number or label.

expr must evaluate in range {1..length(list_target)} otherwise program execution continue on next line.

The selected target is the one at position corresponding to *expr* value counting from left to right, starting at count **1**.

```
>list
    5 ? "testing ON expr GOTO line#,line#,..."
    7 INPUT "select 1-5" A
   10 ON A GOTO 100 , LBL1 , 300 , 400 , EXIT
   14 ? "Woops! selector out of range." : END
   20 GOTO 500
  100 ? "selected GOTO 100" : GOTO 500
  200 LBL1 ? "selected GOTO LBL1" : GOTO 500
  300 ? "selected GOTO 300" : GOTO 500
  400 ? "selected GOTO 400"
  500 ? "testing ON expr GOSUB line#,line#..."
  505 INPUT "select 1-7" B
  510 LET A = 1 : ON A * B GOSUB 600 , 700 , 800 , 900 , 1000 , LBL2 , EXIT
  520 IF B < 1 OR B > 7 : GOTO 14
  524 GOTO 5
  600 ? "selected GOSUB 600" : RETURN
  700 ? "selected GOSUB 700" : RETURN
  800 ? "selected GOSUB 800" : RETURN
  900 ? "selected GOSUB 900" : RETURN
 1000 ? "selected GOSUB 1000" : RETURN
 1100 LBL2 ? "selected GOSUB LBL2" : RETURN
 2000 EXIT ? "selected EXIT"
 2010 END
program address: $91, program size: 618 bytes in RAM memory

>run
testing ON expr GOTO line#,line#,...
select 1-5:2
selected GOTO LBL1
testing ON expr GOSUB line#,line#...
select 1-7:4
selected GOSUB 900
testing ON expr GOTO line#,line#,...
select 1-5:6
Woops! selector out of range.

>
```

index

expr1 OR *expr2* {C,P}

This boolean operator combine bit to bit with an OR operator value of *expr1* with value of *expr2*.

```
>a=3:b=5 if a>3 or a<5 ? b
   5

>if a<3 or a>5 ? a

>
```

See also AND,NOT,XOR.

index

PAD {C,P}

This function return the address of a 128 bytes working buffer. This buffer is used from other usage to program FLASH memory block. Using it in program is safe provide there is no FLASH writing or number printing.

```
>? pad
5816

>
```

index

PAUSE *expr* {C,P}

This command suspend execution for the value of *expr* in milliseconds.

```
>list
   10 input"suspend for seconds? "s
   20 if s=0:end
   30 pause 1000*s
   40 goto 10

>run
suspend for seconds? 5
suspend for seconds? 10
suspend for seconds? 0

>
```

index

PEEK(*expr*) {C,P}

Return byte value at address resulting from evaluation of *expr*.

There is 32 interrupt vectors and they all begin with instruction **INT** which binary code is...

```
>hex: for i=$8000 to i+31*4 step 4: ? peek(i);:next i
$82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82
$82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82 $82
>
```

index

PICK(*n*) {C,P}

This function return the value of nth slot from data stack. The value stay on stack. The top slot as indice zero.

See also

- ALLOC n To reserve *n* slots on top of data stack.
- PUSH expr Push value of *expr* on top of data stack.
- POP extract top value of data stack.
- DROP n Discard *n* slots from top of data stack.
- PUT n,expr put at slot *n* value of *expr*.

```
>push 1:push 2: ? pick(0);pick(1)
2 1

>
```

index {C,P}

PINP pin

This is a constant used by command PMODE to set pin as digital input.

index

PMODE *pin,mode*

This command configure *Dx* pin as digital input PINP or digital output POUT. The power on default mode is digital input.

NUCLEO-8S208RB Dx pins.

PIN	connector
D0	CN7:1
D1	CN7:2
D2	CN7:3

PIN	connector
D3	CN7:4
D4	CN7:5
D5	CN7:6
D6	CN7:7
D7	CN7:8
D8	CN8:1
D9	CN8:2
D10	CN8:3
D11	CN8:4
D12	CN8:5
D13	CN8:6
D14	CN8:9
D15	CN8:10

NUCLEO-8S207K8 Dx pins.

PIN	connector
D0	CN3:2
D1	CN3:1
D2	CN3:5
D3	CN3:6
D4	CN3:7
D5	CN3:8
D6	CN3:9
D7	CN3:10
D8	CN3:11
D9	CN3:12
D10	CN3:13
D11	CN3:14
D12	CN3:15

```
10 PMODE 10,POUT
20 DWRITE 10, 1
```

See also PINP,POUT

index

POKE *expr1*,*expr2*

Put byte value of *expr2* at address of *expr1*

- *expr1* must result in a RAM address or register address.
- *expr2* result in an integer in range {0..255}.

```
>poke PORTC,32 ' turn on user LED.

>
```

See also PEEK

index

POP {C,P}

This function remove the top integer from data stack and return its value.

```
>push 1: push 2 :? pop; pop ' now data stack is empty
2 1

>
```

See also ALLOC,DROP,PICK,PUSH,PUT

index

POUT {C,P}

This constant is used by PMODE to configure **Dx** pin as digital output.

index

PRINT [*string*|*expr*|*char*][,*string*|*expr*|*char*][';'] {C,P}

This command type to terminal. It accept 3 types of information.

- *string* Quoted string.
- *expr* any integer, relation or boolean expression.

- *char* ASCII character preceded by **** or CHAR function.
- *','* comma send a tabulation character to terminal, i.e ASCII **9**. This move terminal cursor right to next column. Column width depend on terminal configuration.
- *';'* semi-colon at end of PRINT command cancel carriage-return. Between items it is a separator.

The PRINT command can be abbreviate by **'?'** character.

```
>? 3
 3

>?,3
     3

>? "hello";" world!"
hello world!

>? "hello","world!"
hello   world!

>? "hello" "world!"
helloworld!

>LET A=51: ? "A=",a
A=  51

>?"A="a
A=51

>
```

index

PORTx {C,P}

For each GPIO port there is a constant which value is the base address of the registers set of the port. Each use 5 registers for its configuration and data I/O.

- ODR Output data register
- IDR Input data register
- DDR Data direction register
- CR1 Configuration register 1
- CR2 Configuration register 2

For each of these register there is a defined constant when added to **PORTx** address give access to that register.

```
>? porta
 20480
```

```
>? portc+ddr
20492

>hex: ? portc+odr
 $500A

>bset portc+odr,bit(5) ' turn on user LED
```

index

PUSH *expr* {C,P}

This command push the value of *expr* on top of data stack. Its inverse is POP remove the top value from data stack and return it.

```
>push 1 push 2 ? pop pop
   2    1

>
```

See also ALLOC,DROP,PICK,PUT

index

PUT *n*,*expr* {C,P}

This command place the value of *expr* at nth position on stack. It is the inverse of PICK. Some slots must have been reserved with ALLOC prior to using these 2. Or may a serie of PUSH.

```
>LIST
    1 XSTACK
    2 ' tset xstack functions and commands
   10 ALLOC 3
   20 PUT 0 , - 1 : PUT 1 , - 2 : PUT 2 , - 3
   30 ? PICK ( 0 ) PICK ( 1 ) PICK ( 2 )
   40 PUT 2 , - 5
   50 ? PICK ( 2 )
program address: $91, program size: 128 bytes in RAM memory

>run
-1 -2 -3
-5

>
```

See also ALLOC,DROP,PICK,PUSH,POP

index

READ {P}

This function read next DATA item and move pointer to next item.

- Data items are separated by a comma.
- DATA lines must be grouped for all items to be read.
- Many DATA group may exist in the same program but at startup the DATA pointer is set to first group. To READ others group the command RESTORE must be used to set the DATA pointer to a specific group.

Reading over the last DATA item result in a fatal error.

```
>list
   10 RESTORE
   20 DATA 100,200
   30 DATA 300
   40 PRINT READ ,READ ,READ ,READ

>run
 100 200 300
No data found.
   40 PRINT READ ,READ ,READ ,READ
```

At any point in a program the command RESTORE can be used to reset the pointer or set it to some specific line number.

index

REBOOT {C,P}

This command reset the MCU.

```
>reboot


Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.0

>
```

index

REM|' *texte*

The keyword **REM** which can be replaced by the tick **'** character mark a comment. Comments end with the line but can be after one or more commands.

In listing only the tick is used to mark comments.

```
>10 rem This is a comment.

>20 ' Comment are skipped by the interpreter.

>list
   10 ' This is a comment.
   20 ' Comment are skipped by the interpreter.
program address:  $80, program size:   69 bytes in RAM memory

>
```

index

RESTORE [line#] {p}

This command is used to restore data pointer to first line of data if there is no parameter or to **line#** if one is given.

It is a fatal error to RESTORE to a line that doesn't exist or is not a data line.

```
>>LIST
    5 ? "test RESTORE command."
   10 RESTORE
   20 ? READ READ READ
   30 RESTORE 300
   40 ? READ READ READ
   50 END
  100 DATA 1 , 2 , 3
  200 DATA 4 , 5 , 6
  300 DATA 7 , 8 , 9
program address: $91, program size: 102 bytes in RAM memory

>RUN
test RESTORE command.
1 2 3
7 8 9

>
```

index

RETURN {P}

This keyword is used to exit from a subroutine and return after the GOSUB that called that subroutine.

```
>list
   10 GOSUB 100 : ? "back from subroutine"
   20 END
```

```
  100 ? "inside subroutine"
  110 RETURN
program address: $91, program size: 65 bytes in RAM memory

>run
inside subroutine
back from subroutine

>
```

index

RND(*expr*)

This function return a pseudo random integer in the interval {1..*expr*}

expr must be a positive integer otherwise it is a fatal error.

```
>list
   10 FOR I = 1 TO 100
   20 ? RND ( 1000 ) , ; : IF I % 10 = 0 : ?
   30 NEXT I
program address: $91, program size: 49 bytes in RAM memory

>run
767     286     763     974     413     313     955     592     228     979
784     5   372     393     765     989     354     794     254     938
598     456     318     233     945     228     803     608     831     126
638     981     459     505     247     616     859     799     753     893
163     439     844     637     964     195     637     876     2   859
766     983     5   78  525     929     193     108     936     187
437     778     261     367     676     266     561     774     473     318
420     132     179     379     708     921     356     5   759     637
140     279     580     713     930     657     294     709     177     179
827     520     117     541     214     197     58  799     330     581

>
```

index

RSHIFT(*expr1*,*expr2*) {C,P}

This function shift right the value of *expr1* by *expr2* bits. The most significant bit is replaced by **0**.

- *expr1* value to be right shifted
- *expr2* value must be in range {0..15} and is the number positions to be shifted.

See also LSHIFT

```
>? rshift($80,7)
   1

>?rshift($40,4)
   4

>
```

index

RUN [name] {C}

Without parameter execute the program residing in RAM. If *name* is given, a program file with that name is searched and if found executed from FLASH memory.

There is 3 hot keys to stop a running program.

1. **CTRL+C** end program and fall back to command line.
2. **CTRL+X** reboot the MCU.
3. **CTRL+Z** clear autorun data in EEPROM and reboot.

```
>dir
$B984   97 bytes,BLINK
$BA04  138 bytes,FIBONACCI

>run fibonacci
   0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
610  987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418
317811 514229 832040 1346269 2178309 3524578 5702887
>
```

index

SAVE {C}

This command is used to save the program in RAM to the file system in FLASH memory. To be saved the first line of the program must be labeled. A program saved can be run from giving its name to the command RUN. The command DIR list on the terminal the files saved.

index

SIZE {C}

This command display the address and size of the program in RAM or in FLASH if such a program was the last executed.

index

SLEEP {C,P}

This command is used to place the MCU in HALT mode. In this mode the internal oscillator is stopped an the MCU is in lowest energy mode. Only a reset or an external interrupt can wake it up. All peripherals are suspended in this mode except for the IWDG if this one is clocked by the LSI.

If the **SLEEP** command is called inside a program and the MCU is woke up by an external interrupt the program continue execution after the **SLEEP** command.

index

SPIEN *div,0|1* (NUCLEO-8S208RB only)

This command enable the SPI peripheral.

- *div* clock frequency divisor {0..7}, Fspi=16Mhz/(2^div)+1 {2..256}.
- *0|1* **0** disable, **1** enable.

NUCLEO-8S208RB pinout

SPI SIGNAL	**CONN.**
~CS	CN8:3
SCLK	CN8:6
MISO	CN8:5
MOSI	CN8:4

index

SPISEL *1|0* (NUCLEO-8S208RB only)

This command is used to select or deselect SPI device.

- *1|0* The **~CS** pin follow the inverse of this value.
  - **1** select, i.e.~CS is low.
  - **0** deselect, i.e. ~CS is high.

1 enable peripheral.

```
10 SPIEN 0,1 'enable SPI at 8Mhz.
20 SPISEL 1  ' Select the device.
30 SPIWR 5   ' write **5** to device.
40 ? SPIRD   ' read value from device.
50 SPISEL 0  ' deselect device.
```

index

SPIRD (NUCLEO-8S208RB only)

This function return a byte read from an SPI device.

index

SPIWR *byte* [, byte] (NUCLEO-8S208RB only)

This command write one or more bytes to SPI device. The following program show the use of an 25LC640 SPI EEPROM. Cette commande permet d'envoyer un ou plusieurs octets vers le périphérique SPI. Le programme suivant illustre l'utilisation de l'interface SPI avec une mémoire externe EEPROM 25LC640. Le programme active l'interface SPI à la fréquence de 2Mhz (16Mhz/2^(2+1)). Ensuite doit activé le bit **WEL** du **25LC640** pour authorizer l'écriture dans l'EEPROM. Cette EEPROM est configurée en page de 32 octets. On écris donc 32 octets au hazard à partir de l'adresse zéro. pour ensuite refaire la lecture de ces 32 octets et les affichés à l'écran.

```
>li
   10 SPIEN 2,1' spi clock 2Mhz
   20 SPISEL 1:SPIWR 6:SPISEL 0 'enable WEL bit in EEPROM
   22 SPISEL 1:SPIWR 5:IF NOT (AND (SPIRD ,2)):GOTO 200
   24 SPISEL 0
   30 SPISEL 1:SPIWR 2,0,0
   40 FOR I =0TO 31:SPIWR RND (256):NEXT I ' write 32 random values
   42 SPISEL 0
   43 GOSUB 100' wait for write completed
   44 SPISEL 1:SPIWR 3,0,0
   46 HEX :FOR I =0TO 31:PRINT SPIRD ,:NEXT I ' read back the written
values
   50 SPISEL 0
   60 SPIEN 0,0
   70 END
   90 ' wait for write completed
  100 SPISEL 1:SPIWR 5:S =SPIRD :SPISEL 0
  110 IF AND (S ,1):GOTO 100
  120 RETURN
  200 PRINT "failed to enable WEL bit in EEPROM"
  210 SPISEL 0  ' deselect EEPROM
  220 SPIEN 0,0 ' disable SPI

>run
 $3F $99 $19 $73 $4C $FE $B1 $66 $88 $7F $31 $FD $AD $BA $78 $1B $78 $2F
$23 $59 $7D $C6 $2E $D0 $80 $7A $19 $E8 $53 $BC  $5 $AC
>run
 $A0 $AE $DD $32 $C5 $D6 $DB $43 $90 $CA $CF $60 $37 $B9 $D8 $C0  $7 $3B
$AE $B2 $58 $5F $B5 $33 $8D $1D $7D $3F $94 $7D $FF $F3
>
```

index

STEP *expr* {C,P}

This keyword is part of **FOR..NEXT** loop initialization. It set the increment of control variable.

See FOR,TO,NEXT

index

STOP {P}

This command is a tool to help debugging a program. It is used to stop execution of a program at some point and go to command line. From command line variables content can be viewed or changed. When the RUN command is invoked after a stop the program continue after the **STOP** point.

If [END]](#end) command is invoked from command line while in STOP mode, the program is ended.

```
>10 FOR A=1 TO 10:PRINT A:STOP:NEXT A

>run
   1
break point, RUN to resume.

>run
   2
break point, RUN to resume.

>run
   3
break point, RUN to resume.

>run
   4
break point, RUN to resume.

>end

>run
   1
break point, RUN to resume.

>end

>
```

index

TICKS {C,P}

The system as an internal 24 bits counter incremented every millisecond. **TIMER4** is used for that purpose. **TICKS** function return the value of this counter. The counter rollover at 0x7fffff to stay in positive values. the give about 2.3 hours rollover. When the AWU or SLEEP is used the ticks counter is suspended during HALT period.

```
>let t=ticks: for a=1 to 1000:next a : ?ticks-t " msec"
  10 msec
```

index

TIMEOUT

This function check if the TIMER is expired. It return TRUE if so.

```
>TIMER 5:DO LET A=TIMEOUT:PRINT A;:UNTIL A
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
>
```

index

TIMER *expr* {C,P}

This command set a countdown timer. This count is decremented every millisecond. The function TIMEOUT is used to know when the count reach **0**.

```
>timer 1000: do until timeout ' time out after 1 second

>
```

index

TO *expr* {C,P}

This keyword is part of FOR..NEXT loop initialization. It is used to set the limit.

index

TONE *expr1*,*expr2* {C,P}

This command is used to generate a tone.

- *expr1* value is tone frequency.
- *expr2* value is tone duration in milliseconds.

The audio output for NUCLEO-8S208RB is on **CN9:6**

The audio outpout for NUCLEO-8S207K8 is on **CN3:3

This tone is generated using TIMER2 channel 1 configured in PWM mode with 50% duty cycle.

```
>list
    5 ' play scale
   10 LET @ ( 1 ) = 440 , @ ( 2 ) = 466 , @ ( 3 ) = 494 , @ ( 4 ) = 523 , @
( 5 ) = 554 , @ ( 6 ) = 587
   20 LET @ ( 7 ) = 622 , @ ( 8 ) = 659 , @ ( 9 ) = 698 , @ ( 10 ) = 740 ,
@ ( 11 ) = 784 , @ ( 12 ) = 831
   30 FOR I = 1 TO 12 : TONE @ ( I ) , 200 : NEXT I
program address: $91, program size: 187 bytes in RAM memory

>
```

index

UBOUND

This function return the last indice of **@** array. As this value depend on RAM left free when a program is loaded a runtime function is required to know this value. The **@** array is garanteed to have at least a size of 10.

The **@** indices are in range {1..ubound}.

index

UFLASH (C,P)

This function return the address of FLASH free for program use. This value varies as the numbers of files and size varies. So it should be called whenever a program want to write to FLASH memory.

This address is always aligned to FLASH block which are 128 bytes in size.

```
>list
    1 BLINK
    5 ' Blink LED2 on card
   10 DO BTOGL PORTC , BIT ( 5 ) PAUSE 500 UNTIL KEY?
   20 LET A = KEY
   30 BRES PORTC , BIT ( 5 )
   40 END
program address: $91, program size: 84 bytes in RAM memory

>? uflash
47872

>save

>dir
$BB04 84 bytes,BLINK

>? uflash
48000
```

```
>
```

index

UNTIL *expr* {C,P}

This keyword close a DO..UNTIL loop. When the *expr* evaluate to any not null integer the loop exit.

```
>LIST
   10 LET A = 1
   20 DO
   30 ? A ;
   40 LET A = A + 1
   50 UNTIL A > 10
program address: $91, program size: 51 bytes in RAM memory

>RUN
1 2 3 4 5 6 7 8 9 10
>
```

index

USR(*addr*,*expr*) {C,P}

This function is used to call a machine code routine. The machine code routine return a value on data stack.

- *addr* is address of machine code routine
- *expr* is a value used as input parameter to subroutine.
- An integer is pushed to data stack by the machine code routine.

In the following example the machine code stored in DATA line is written to FLASH memory then it is called to compute the square of a small integer.

machine code routine, file square.asm

```
   .area CODE

.nlist
.include "inc/stm8s207.inc"
.include "inc/nucleo_8s207.inc"
.include "tbi_macros.inc"
.list
 square:
   _at_top
   rrwa X
   ld xl,a
   mul x,a
  clr a
```

```
  _xpush
   ret
```

command to assemble source file.

```
picatout:~/github/stm8_tbi$ sdasstm8 -l  square.asm
```

part of listing containing the binary code, file square.lst

```
                                     7 .list
      000000                         8  square:
      000000                         9    _at_top
      000000 90 F6            [ 1]    1         ld a,(y)
      000002 93               [ 1]    2         ldw x,y
      000003 EE 01            [ 2]    3         ldw x,(1,x)
      000005 01               [ 1]   10    rrwa X
      000006 97               [ 1]   11    ld xl,a
      000007 42               [ 4]   12    mul x,a
      000008 4F               [ 1]   13   clr a
      000009                         14   _xpush
      000009 72 A2 00 03      [ 2]    1         subw y,#CELL_SIZE
      00000D 90 F7            [ 1]    2         ld (y),a
      00000F 90 EF 01         [ 2]    3         ldw (1,y),x
      000012 81               [ 4]   15    ret
```

BASIC program with DATA lines containing binary code, file usr_test.bas

```
>list
    1 ' write binary code in flash and execute it.
    2 ' square a number
   20 RESTORE
   22 ' machine code
   30 DATA 144 , 246 , 147 , 238 , 1 , 1 , 151 , 66 , 79 , 114 , 162
   40 DATA 0 , 3 , 144 , 247 , 144 , 239 , 1 , 129
   50 LET A = UFLASH : ? "routine at " ; A
   60 FOR I = 0 TO 18
   70 WRITE A + I , READ
   80 NEXT I
   90 INPUT "number {1..255, 0 quit} to square?" N
  100 ? USR ( A , N )
  110 IF N <> 0 : GOTO 90
program address: $91, program size: 382 bytes in RAM memory

>run
routine at 48000
number {1..255, 0 quit} to square?:255
```

```
65025
number {1..255, 0 quit} to square?:125
15625
number {1..255, 0 quit} to square?:40
1600
number {1..255, 0 quit} to square?:20
400
number {1..255, 0 quit} to square?:12
144
number {1..255, 0 quit} to square?:0
0

>
```

If there is space available in RAM a BUFFER for machine code can be created in RAM and data poked to that buffer instead of writting it to FLASH. That would reduce FLASH wear out.

index

WAIT *expr1*,*expr2*[,*expr3] {C,P}

This command wait for a change of state in a register or memory.

- *expr1* address of register or memory
- *expr2* AND mask to apply to value at address.
- *expr3* XOR mask to apply after the AND mask. If the parameter is missing value **0** is used.

This command can be used to wait for an input pin to switch from low to high and verse-versa. Or for some register state change. The following example poke a value to UART1_DR register and wait for the transmission to complete by polling bit 6 of UART1_SR register as this bit goes to **1** when transmission is completed.

```
>poke $5231,65:wait $5230,bit(6)
A
>
```

index

WORDS {C,P}

This command is used to print on terminal the list of commands, functions and operators used by Tiny BASIC with token number of each. This dictionary is used by compiler and decompiler.

```
>words
ABS     ADCON       ADCREAD     ALLOC       AND
ASC     AUTORUN     AWU     BIT     BRES
BSET        BTEST       BTOGL       BUFFER      BYE
CHAIN       CHAR        CONST       CR1     CR2
```

```
DATA        DDR     DEC     DIM     DIR
DO      DREAD       DROP        DWRITE      EDIT
EEFREE      EEPROM      END     ERASE       FCPU
FOR     FREE        GET     GOSUB       GOTO
HEX     I2C.CLOSE   I2C.OPEN    I2C.READ    I2C.WRITE
IDR     IF      INPUT       KEY     KEY?
LET     LIST        LOG2        LSHIFT      NEW
NEXT        NOT     ODR     ON      OR
PAD     PAUSE       PEEK        PICK        PINP
PMODE       POKE        POP     POUT        PRINT
PORTA       PORTB       PORTC       PORTD       PORTE
PORTF       PORTG       PORTI       PUSH        PUT
READ        REBOOT      REM     RESTORE     RETURN
RND     RSHIFT      RUN     SAVE        SIZE
SLEEP       STEP        STOP        TICKS       TIMEOUT
TIMER       TO      TONE        TRACE       UBOUND
UFLASH      UNTIL       USR     WAIT        WORDS
WRITE       XOR
107 words in dictionary

>
```

index

WRITE *expr1*,*expr2*[,*expr*]*

This command is used to write data to persistant memory, i.e. EEPROM and FLASH.

- *expr1* is the starting address.
- *data1*,*[,data2]** is a list of data elements to be written in consecutives adressess. these data elements can be of 3 types.
  - integer expression resulting in value {0..255}. If value is >255 only the least significant byte is used.
  - Quoted string, written as zero terminated string.
  - escaped character, i.e. **\c** where **c** is any ASCII character.

```
>write EEPROM,"Hello world!"

>for i=eeprom to i+11:? char(peek(i));:next i
Hello world!
>
```

index

term1 XOR *term2* {C,P}

This boolean operator apply **excluive or** bit to bit between left and right terms. Terms can be

- arithmetic expression.

- comparison between 2 arithmetic expressions.
- or boolean expressions themselve.

See arithmetic expressions for operators priorities.

```
>let a=5,b=10

>? a xor b
15

>? a>b xor b>9
-1

>? a>b xor b<9
0

>? a and b xor 7
7

>? a and 4 xor 7
3

>
```

index

main index

Files system

The are edited in RAM to minimize wear out of FLASH. Once a program is debugged it should be saved in FLASH memory. The file system is a very simple one. As the MCU block erase is organized in block 128 bytes the file system is orgnaize around this size. So a program file always take a multiple of 128 bytes in FLASH memory. The memory used for this file system is the one left after Tiny BASIC. When no files are saved the size can be known by the following command:

```
Tiny BASIC for STM8
Copyright, Jacques Deschenes 2019,2022
version 2.5R1

>? $1000-uflash
-44032

>? $10000-uflash
17408
```

As program files saved in this file system are executed in situ, the file system doesn't use extended memory, i.e. memory address over 65535.

Executing BASIC program in extended memory would require a modification to interpreter that would increase its size and slow it down. As it is not all STM8 MCU that have extended memory the choice was made to limit the system to {0...65535} memory range. But extended memory can be used for program data storage. It can be written with command WRITE and read with function PEEK.

main index

Firmware installation

The first step is to select options in config.inc file.

- **DEBUG** should be set to **0** for stable build where debugging is not required.
- **SEPARATE** should be set to **0** in most case. It is use only when debugging because sdasstm8 assembler message report is quite anoying. It doesn't report in which file there is an error when they are all assembled together.
- **NUCLEO_8S208RB** put this to **1** if this is the target board otherwise set it to **0**. The **NUCLEO_8S207K8** is automatically selected when this one is deselected.

In the root directory there is bash script build.sh to launch the build process. This script take only 2 parameters:

- first parameter **s207** or *s208* to select the target board.
- second and optional is **flash** if you want to flash the board after successfull build. If the build fail there will be no flash operation.

```
picatout:~/github/stm8_tbi$ ./build.sh s207 flash

***************
cleaning files
***************
rm -f build/*

************************************
compiling TinyBasic  for stm8s207K8
************************************
sdasstm8 -g -l -o build/TinyBasic.rel hardware_init.asm  arithm24.asm
debug_support.asm flash_prog.asm files.asm terminal.asm code_address.asm
compiler.asm i2c.asm decompiler.asm TinyBasic.asm app.asm
sdcc -mstm8 -lstm8 -L -I../inc -Wl-u -o build/TinyBasic.ihx
build/TinyBasic.rel
objcopy -Iihex -Obinary  build/TinyBasic.ihx build/TinyBasic.bin

-rw-rw-r-- 1 jacques jacques 15108 nov 27 11:12 build/TinyBasic.bin


******************
flashing stm8s207K8
```

```
******************
stm8flash -c stlinkv21 -p stm8s207K8 -s flash -w build/TinyBasic.ihx
Determine FLASH area
STLink: v2, JTAG: v25, SWIM: v7, VID: 8304, PID: 4b37
Due to its file extension (or lack thereof), "build/TinyBasic.ihx" is
considered as INTEL HEX format!
15108 bytes at 0x8000... OK
Bytes written: 15108
```

main index

Using NUCLEO-8S20x board

STM8 TinyBASIC is developped and built on Ubuntu system but the binary can be flashed on the board on Windows system as well. When the board is connected to PC USB port a new drive appears in file system. Dropping the file build/TinyBasic.bin on this drive will flash the firmware on the board.

The STLINK programmer on the board also emulate a serial port. The only requirement to communicate with TinyBASIC system is to have a terminal emulator configured to connect on that serial port at 115200 BAUD, 8 bit, 1 STOP, no parity. On Windows system TeraTerm or Putty can be used. On Ubuntu there is many options. I use GTKterm

main index

Sending a BASIC program to NUCLEO-8S20x board

Although STM8 TinyBASIC as a line editor in the 80'S 8 bits BASIC computer style. Writting your program on the PC using a full featured editor is nice and can be done.

I have written a small command line tool that can be used to send a BASIC program written on the PC to the board. This tool is only compiled for linux system though. There also bash script in root directory send.sh. This script expect the BASIC program to be in **BASIC** directory.

```
picatout:~/github/stm8_tbi$ ./send.sh blink.bas
port=/dev/ttyACM0, baud=115200,delay=100
Sending file BASIC/blink.bas

NEW
1  BLINK
5 ' Blink LED2 on card
10 DO BTOGL PORTC,BIT(5) PAUSE 500 UNTIL KEY?
20 LET A=KEY
30 BRES PORTC,BIT(5)
40 END

8 lines sent

picatout:~/github/stm8_tbi$
```

When the board receive the program the program lines scroll on terminal screen.

main index

Source code

The source code for this project is on https://github.com/Picatout/stm8_tbi

main index