Module	Function / Macro / Constant	#define in File “config.h“
main_general.h	setup()	none
	loop()	
	HIGH / LOW	
	true / false	
	boolean	
	string	
	char(d)	
	byte(d)	
	int()	
	word(a)	
	wordConcat(hb,lb)	
	long(c)	
	float(d)	
	min(a,b)	
	max(a,b)	
	abs(a)	
	constrain(x, low, high)	
	map(x,inMin,inMax,outMin,outMax)	
	pow(x,y)	
	sqrt(x)	
	sin(a)	
	cos(a)	
	tan(a)	
	isAlphaNumeric(a)	
	isAlpha(a)	
	isAscii(a)	
	isWhitespace(a)	

misc.h (auto loaded)	isControl(a)	none
	isDigit(a)	
	isGraph(a)	
	isLowerCase(a)	
	isPrintable(a)	
	isPunct(a)	
	isSpace(a)	
	isUpperCase(a)	
	isHexadecimalDigit(a)	
	randomSeed(d)	
	random()	
	lowByte(x)	
	highByte	
	bitRead(byte, bit)	
	bitWrite(byte, bit, value)	
	bitSet(byte, bit)	
	bitClear(byte, bit)	
	bitToggle(byte, bit)	
	bit(n)	
	interrupts()	
	noInterrupts()	
	round(x)	
	ceil(x)	
	floor(x)	
	toASCII(c)	
	toUpperCase(c)	
	toLowerCase(c)	
	log2(d)	
	floatToString(buf, value, digits)	USE_FTOA

binary literals	0b00000000 … 0b11111111	none
gpio (auto loaded)	pinMode(port, pin, mode)	none
	pinSet(port, pin)	
	pinRead(port, pin)	
	portSet(port)	
	portRead(port)	
	configExintEdge(&port, edge)	USE_PORT_ISR
sw_delay (auto loaded)	sw_delay(uint32_t N)	none
	sw_delayMicroseconds(uint16_t N)	
	sw_delayNOP(uint8_t N)	
stm8as (auto loaded)	ASM(mnem)	none
	NOP	
	WAIT_FOR_INTERRUPT	
	ENTER_HALT	
timer4 (auto loaded)	millis()	none
	micros()	
	flagMilli()	
	clearFlagMilli()	
	delay(ms)	
	delayMicroseconds(us)	
	setTimeout(N, ms)	
	checkTimeout(N)	
	resetTime()	
	attachInterruptMillis(fct)	USE_MILLI_ISR
	detachInterruptMillis()	

eeprom	read_1B(addr) / read_2B(addr) / read_4B(addr)	none
	write_1B(adr, val) / write_2B(adr, val) / write_4B(adr, val)	
	OPT_writeByte(adr, data)	
	OPT_setDefault()	
	OPT_setBootloader()	
	OPTx / NOPTx	
	flash_writeByte(adr, val)	
	EEPROM_writeByte(num, val)	
	EEPROM_readByte(num)	
	flash_eraseBlock(adr) (Cosmic only)	
	flash_writeBlock(adr, buf) (Cosmic only)	
	PFLASH_START / PFLASH_END / PFLASH_SIZE	
	EEPROM_START / EEPROM_END / EEPROM_SIZE	
uart1_blocking	UART1_begin(baudrate)	none
	UART1_end()	
	UART1_listen()	
	UART1_write(data)	
	UART1_writeBytes(num, buf);	
	UART1_available()	
	UART1_read()	
	UART1_readBlock()	
putchar	putcharAttach(fct)	none
	putcharDetach()	
getchar	getcharAttach(fct)	none
	getcharDetach()	
tone (requires option byte change)	tone(uint16_t Hz, uint16_t millis)	none
	noTone()	

Example
setup()
loop()
LED = HIGH;
if (a==true)
boolean a;
string s[20];
c = char(d);
b = char(d);
d = int(c);
w = word(a);
w = wordConcat(hb, lb);
d = long(c);
f = float(d);
a = min(b,c);
a = max(b,c);
a = abs(a);
a = constrain(a, 10, 100);
b = map(a, 0,1024, 0,100);
y = pow(x, 0.3)
y = sqrt(x)
y = sin(x);
y = cos(x);
y = tan(x);
if (isAlphaNumeric(a))
if (isAlpha(a))
if (isAscii(a))
if (isWhitespace(a))

if (isControl(a))
if (isDigit(a))
if (isGraph(a))
if (isLowerCase(a))
if (isPrintable(a))
if (isPunct(a))
if (isSpace(a))
if (isUpperCase(a))
if (isHexadecimalDigit(a))
randomSeed(10);
a = random();
LB = lowByte(x);
HB = highByte(x);
a = bitRead(b, 4)
bitWrite(a, 3, 1);
bitSet(a, 3);
bitClear(a, 3);
bitToggle(a, 3);
a = bit(3);
interrupts();
noInterrupts()
a = round(a);
a = ceil(a);
a = floor(a);
c = toASCII(c);
c = toUpperCase(c);
c = toLowerCase(c);
n = log2(d)
printf("%s\n", floatToString(str,x,3));

A = 0b11001100
pinMode(PORT_H, pin3, OUTPUT);
pinSet(PORT_H, pin3) = state;
state = pinRead(PORT_D, pin7);
portSet(PORT_H) = portState;
portState = portRead(PORT_H);
configExintEdge(&PORT_E, CHANGE);
sw_delay(10);
delayMicroseconds(10);
sw_delayNOP(100);
ASM("trap");
NOP;
WAIT_FOR_INTERRUPT;
ENTER_HALT;
uint32_t time_ms = millis();
uint32_t time_us = micros();
if (flagMilli())
clearFlagMilli();
delay(500);
delayMicroseconds(500);
setTimeout(3, 500);
checkTimeout(3);
resetTime();
attachInterruptMillis(fct);
detachInterruptMillis();

A = read_1B(0x4000);
write_2B(0x0020, ‘a‘);
OPT_writeByte(OPT2, 0x00);
OPT_setDefault();
OPT_setBootloader(true);
see OPT_write
flash_writeByte(0xFFF0, val);
EEPROM_writeByte(10, val);
A = EEPROM_readByte(10);
flash_eraseBlock(0xFFF0);
flash_writeBlock(0xFFF0, buf);
UART1_begin(19200);
UART1_end();
UART1_listen();
UART1_write(c);
UART1_writeBytes(num, buf);
if (UART1_available())
Rx = UART1_read();
Rx = UART1_readBlock();
putcharAttach(UART1_write);
putcharDetach();
getcharAttach(UART1_readBlock);
getcharDetach();
beep(2000, 500);
noTone()

Short Description / Remark
user initialization routine. Called once after start of program
user loop routine. Called continuously
constants for 1 / 0, e.g. for pinSet()
constants for 1 / 0, e.g. for if
Boolean variable. Same as uint8_t
Character array. Same as char*
Converts a value to the char data type. Same as ((char) d)
Converts a value to the byte data type. Same as ((uint8_t) a)
Converts a value to the int data type.
Convert a value to the word data type.
Convert a word from two bytes.
Converts a value to the long data type.
Converts a value to the float data type.
minimum of 2 numbers; do not use as function argument
maximum of 2 numbers; do not use as function argument
absolute value of a number; do not use as function argument
clip value to range [low;high]; do not use as function argument
re-map a number from one range to another
Calculates the value of a number raised to a power.
Calculates the square root of a number.
Calculates the sine of an angle (in radians). The result is in [-1;1].
Calculates the cosine of an angle (in radians). The result is in [-1;1].
Calculates the tangent of an angle (in radians). The result is in [-inf;inf]
Analyse if a char is alphanumeric.
Analyse if a char is is alpha.
Analyse if a char is ASCII.
Analyse if a char is a white space.

Analyse if a char is a control character.	
Analyse if a char is a digit.	
Analyse if a char is a printable character.	
Analyse if a char is a lower case character.	
Analyse if a char is a printable character.	
Analyse if a char is punctuation character.	
Analyse if a char is a space character.	
Analyse if a char is a upper case character.	
Analyse if a char is a valid hexadecimal digit.	
seed the random number generator used by the random()	
generate a pseudo random number within [0;INT16_MAX]	
Extracts the low-order (rightmost) byte of a variable (e.g. a word)	Change for compatibility with
Extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type).	
read single bit position in byte	Change for compatibility with
set single bit value in byte to value	Change for compatibility with
set single bit in data to '1'	Change for compatibility with
clear single bit in data to '0'	Change for compatibility with
toggle single bit state in byte	Change for compatibility with
calculate bit value of bit n	Change for compatibility with
Globally enable interrupts	
Globally disable interrupts	
round x to the nearest integer	
round x upwards to the nearest integer	
round x downwards to the nearest integer	
return lower 7 bits of 1B argument (ASCII range)	
converts an alpha to upper case letter	
converts an alpha to lower case letter	
Integer calculation of (rough) log2(x), i.e. determine binary power to reach number	
convert float to string for printing floats. No scientific notation. Is rather large → only include if required	new

binary literals	new, copied from sduino
Set pin direction and optional features. Pin modes are INPUT, INPUT_INTERRUPT, INPUT_PULLUP, INPUT_PULLUP_INTERRUPT OUTPUT, OUTPUT_OPENDRAIN	
Set pin state	
Read pin state	
Set port state (8 pins)	
Read port state (8 pins)	
configure sensitivity for port interrupt. Use together with pinMode and ISR function	
Delay code for approximately N milliseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h	fix re-entrance bug & calibrat
Delay code for approximately N microseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h	calibrate timing for debug/opt
Delay code for Nx NOP() (inline blocking) For compiler / optimization dependent latency see sw_delay.h	
Inline STM8 assembler	
NOP operation (1 CPU cycle)	change from _NOP_ for read
Halt core with clock running. Resume execution, e.g. by timer interrupt	
Halt core and clock. Resume execution e.g. by auto-wakeup, see “awu“	
Milliseconds since start of program	
Microseconds since start of program with 4µs resolution	
Check if 1ms has passed. Reset by clearFlagMilli()	
Reset flagMilli() flag for 1ms	
Delay code for ms using timer 4. Timing is (rather) independent of interrupt load For compiler / optimization dependent latency see timer4.c	
Delay code for us using timer 4. Timing is (rather) independent of interrupt load. It has a resolution of 4µs For compiler / optimization dependent latency see timer4.c	
start single-shot timeout N (N=0..3) with ms period. Use together with checkTimeout()	
Check wither timeout N (N=0..3) has expired. Use together with setTimeout()	
Reset millis and micros to 0	
Attach user routine to 1ms interrupt (=TIM4UPD)	
Detach user routine from 1ms interrupt (=TIM4UPD)	

read byte, word or long word from flash, EEPROM, RAM	
write byte, word or long word to RAM. Flash and EEPROM see below	
write option byte and return if value has changed	
set all option bytes to default. On change trigger reset	
activate BSL via option byte. On change trigger reset	
name of option byte. See stm8as.h	
write 1B to P-flash using physical address	
write 1B to EEPROM using logical address	
read 1B from EEPROM using logical address	
erase 128B P-flash block. Requires RAM execution → Cosmic only	
write 128B P-flash block. Requires RAM execution → Cosmic only	
physical address start, end and size [B] of P-flash	
physical address start, end and size [B] of EEPROM	
initialize UART1 baudrate and enable sender & receiver	new
disable sender & receiver	
enable sender & receiver. Retain previous settings	
send 1 byte via UART1	new
send N bytes via UART1	new
check if byte received via UART1	new
read byte from UART1 receive buffer. Non-blocking	
read byte from UART1 receive buffer. Blocking	new
set send routine (1B) for stdio putchar / printf; For printing floats, use float2str() helper routine	new
detach send routine from stdio putchar / printf	new
set receive routine (1B) for stdio getchar / gets	new
detach receive routine from stdio getchar / gets	new
play tone via beeper module with given frequency in Hz (<500 off) and duration in millis (0=forever)	change from beep for compat
switch off tone started with tone() and duration=0 (see above)	new

Arduino

Arduino

Arduino

Arduino

Arduino

Arduino

Arduino

e timing for debug/optimize

imize

ability

tibility with Arduino. Added flexibility

Function / Macro / Constant	Module	#define in File “config.h“
setup()	main_general.h	
loop()	main_general.h	
HIGH / LOW	misc.h (auto loaded)	
true / false	misc.h (auto loaded)	
boolean	misc.h (auto loaded)	
string	misc.h (auto loaded)	
char(d)	misc.h (auto loaded)	
byte(d)	misc.h (auto loaded)	
int()	misc.h (auto loaded)	
word(a)	misc.h (auto loaded)	
wordConcat(hb,lb)	misc.h (auto loaded)	
long(c)	misc.h (auto loaded)	
float(d)	misc.h (auto loaded)	
min(a,b)	misc.h (auto loaded)	
max(a,b)	misc.h (auto loaded)	
abs(a)	misc.h (auto loaded)	
constrain(x, low, high)	misc.h (auto loaded)	

map(x,inMin,inMax,outMin,outMax)	misc.h (auto loaded)	
pow(x,y)	misc.h (auto loaded)	
sqrt(x)	misc.h (auto loaded)	
sin(a)	misc.h (auto loaded)	
cos(a)	misc.h (auto loaded)	
tan(a)	misc.h (auto loaded)	
isAlphaNumeric(a)	misc.h (auto loaded)	
isAlpha(a)	misc.h (auto loaded)	
isAscii(a)	misc.h (auto loaded)	
isWhitespace(a)	misc.h (auto loaded)	
isControl(a)	misc.h (auto loaded)	
isDigit(a)	misc.h (auto loaded)	
isGraph(a)	misc.h (auto loaded)	
isLowerCase(a)	misc.h (auto loaded)	
isPrintable(a)	misc.h (auto loaded)	
isPunct(a)	misc.h (auto loaded)	
isSpace(a)	misc.h (auto loaded)	
isUpperCase(a)	misc.h (auto loaded)	
isHexadecimalDigit(a)	misc.h (auto loaded)	

randomSeed(d)	misc.h (auto loaded)	
random()	misc.h (auto loaded)	
lowByte(x)	misc.h (auto loaded)	
highByte	misc.h (auto loaded)	
bitRead(byte, bit)	misc.h (auto loaded)	
bitWrite(byte, bit, value)	misc.h (auto loaded)	
bitSet(byte, bit)	misc.h (auto loaded)	
bitClear(byte, bit)	misc.h (auto loaded)	
bitToggle(byte, bit)	misc.h (auto loaded)	
bit(n)	misc.h (auto loaded)	
interrupts()	misc.h (auto loaded)	
noInterrupts()	misc.h (auto loaded)	
round(x)	misc.h (auto loaded)	
ceil(x)	misc.h (auto loaded)	
floor(x)	misc.h (auto loaded)	
toASCII(c)	misc.h (auto loaded)	
toUpperCase(c)	misc.h (auto loaded)	
toLowerCase(c)	misc.h (auto loaded)	
log2(d)	misc.h (auto loaded)	

floatToString(buf, value, digits)	misc.h (auto loaded)	USE_FTOA
B00000000 … B11111111	binary.h (auto loaded)	none
pinMode(port, pin, mode)	gpio (auto loaded)	
pinSet(port, pin)	gpio (auto loaded)	
pinRead(port, pin)	gpio (auto loaded)	
portSet(port)	gpio (auto loaded)	
portRead(port)	gpio (auto loaded)	
attachInterruptPort(portAddr, fctName, edge)	gpio (auto loaded)	USE_PORT_ISR
detachInterruptPort(portAddr)	gpio (auto loaded)	USE_PORT_ISR
attachInterruptPin(fctName, edge)	gpio (auto loaded)	USE_TLI_ISR
detachInterruptPin()	gpio (auto loaded)	USE_TLI_ISR
sw_delay(uint32_t N)	sw_delay (auto loaded)	
sw_delayMicroseconds(uint16_t N)	sw_delay (auto loaded)	
sw_delayNOP(uint8_t N)	sw_delay (auto loaded)	
ASM(mnem)	stm8as (auto loaded)	

NOP	stm8as (auto loaded)	
WAIT_FOR_INTERRUPT	stm8as (auto loaded)	
ENTER_HALT	stm8as (auto loaded)	
uint32_t millis()	timer4 (auto loaded)	
uint32_t micros()	timer4 (auto loaded)	
flagMilli()	timer4 (auto loaded)	
clearFlagMilli()	timer4 (auto loaded)	
resetTime()	timer4 (auto loaded)	
attachInterruptMillis(fct)	timer4 (auto loaded)	USE_MILLI_ISR
detachInterruptMillis()	timer4 (auto loaded)	USE_MILLI_ISR
UART1_begin(baudrate)	uart1_blocking	
UART1_end()	uart1_blocking	
UART1_listen()	uart1_blocking	
UART1_write(data)	uart1_blocking	
UART1_writeBytes(num, buf);	uart1_blocking	
UART1_available()	uart1_blocking	
UART1_read()	uart1_blocking	
putcharAttach(fct)	putchar	
putcharDetach()	putchar	

Short Description / Remark
user initialization routine. Called once after start of program
user loop routine. Called continuously
constants for 1 / 0, e.g. for pinSet()
constants for 1 / 0, e.g. for if
Boolean variable. Same as uint8_t
Character array. Same as char*
Converts a value to the char data type. Same as ((char) d)
Converts a value to the byte data type. Same as ((uint8_t) a)
Converts a value to the int data type.
Convert a value to the word data type.
Convert a word from two bytes.
Converts a value to the long data type.
Converts a value to the float data type.
minimum of 2 numbers; do not use as function argument
maximum of 2 numbers; do not use as function argument
absolute value of a number; do not use as function argument
clip value to range [low;high]; do not use as function argument

re-map a number from one range to another
Calculates the value of a number raised to a power.
Calculates the square root of a number.
Calculates the sine of an angle (in radians). The result is in [-1;1].
Calculates the cosine of an angle (in radians). The result is in [-1;1].
Calculates the tangent of an angle (in radians). The result is in [-inf;inf]
Analyse if a char is alphanumeric.
Analyse if a char is is alpha.
Analyse if a char is ASCII.
Analyse if a char is a white space.
Analyse if a char is a control character.
Analyse if a char is a digit.
Analyse if a char is a printable character.
Analyse if a char is a lower case character.
Analyse if a char is a printable character.
Analyse if a char is punctuation character.
Analyse if a char is a space character.
Analyse if a char is a upper case character.
Analyse if a char is a valid hexadecimal digit.

seed the random number generator used by the random()
generate a pseudo random number within [0;INT16_MAX]
Extracts the low-order (rightmost) byte of a variable (e.g. a word)
Extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type).
read single bit position in byte
set single bit value in byte to value
set single bit in data to '1'
clear single bit in data to '0'
toggle single bit state in byte
calculate bit value of bit n
Globally enable interrupts
Globally disable interrupts
round x to the nearest integer
round x upwards to the nearest integer
round x downwards to the nearest integer
return lower 7 bits of 1B argument (ASCII range)
converts an alpha to upper case letter
converts an alpha to lower case letter
Integer calculation of (rough) log2(x), i.e. determine binary power to reach number

convert float to string for printing floats. No scientific notation. Is rather large → only include if required
binary literals
Set pin direction and optional features. Pin modes are INPUT, INPUT_INTERRUPT, INPUT_PULLUP, INPUT_PULLUP_INTERRUPT OUTPUT, OUTPUT_OPENDRAIN
Set pin state
Read pin state
Set port state (8 pins)
Read port state (8 pins)
Attach user routine to port interrupt (=EXINTx). Edges are LOW, CHANGE, RISING, FALLING, PREV_SETTING Enable pin interrupt via pinMode()
Detach user routine from port interrupt (=EXINTx). Disable pin interrupt via pinMode()
Attach user routine to pin D7 interrupt (=TLI). Edges are LOW, CHANGE, RISING, FALLING, PREV_SETTING Enable pin interrupt via pinMode()
Detach user routine from pin D7 interrupt (=TLI). Disable pin interrupt via pinMode()
Delay code for approximately N milliseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h
Delay code for approximately N microseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h
Delay code for Nx NOP() (inline blocking) For compiler / optimization dependent latency see sw_delay.h
Inline STM8 assembler

NOP operation (1 CPU cycle)
Halt core with clock running. Resume execution, e.g. by timer interrupt
Halt core and clock. Resume execution e.g. by auto-wakeup, see “awu“
Milliseconds since start of program
Microseconds since start of program with 4µs resolution
Check if 1ms has passed. Reset by clearFlagMilli()
Reset flagMilli() flag for 1ms
Reset millis and micros to 0
Attach user routine to 1ms interrupt (=TIM4UPD)
Detach user routine from 1ms interrupt (=TIM4UPD)
initialize UART1 baudrate and enable sender & receiver
disable sender & receiver
enable sender & receiver. Retain previous settings
send 1 byte via UART1
send N bytes via UART1
check if byte received via UART1
read byte from UART1 receive buffer. Non-blocking
set send routine (1B) for stdio putchar / printf; For printing floats, use below float2str() helper routine
detach send routine from stdio putchar / printf