Module	Function / Macro / Constant	#define in File “config.h“	Example	Short Description / Remark
main_general.h	setup()	none	setup()	user initialization routine. Called once after start of program
	loop()		loop()	user loop routine. Called continuously
misc.h (auto loaded)	HIGH / LOW	none	LED = HIGH;	constants for 1 / 0, e.g. for pinSet()
	true / false		if (a==true)	constants for 1 / 0, e.g. for if
	boolean		boolean a;	Boolean variable. Same as uint8_t
	string		string s[20];	Character array. Same as char*
	char(d)		c = char(d);	Converts a value to the char data type. Same as ((char) d)
	byte(d)		b = char(d);	Converts a value to the byte data type. Same as ((uint8_t) a)
	int()		d = int(c);	Converts a value to the int data type.
	word(a)		w = word(a);	Convert a value to the word data type.
	wordConcat(hb,lb)		w = wordConcat(hb, lb);	Convert a word from two bytes.
	long(c)		d = long(c);	Converts a value to the long data type.
	float(d)		f = float(d);	Converts a value to the float data type.
	min(a,b)		a = min(b,c);	minimum of 2 numbers; do not use as function argument
	max(a,b)		a = max(b,c);	maximum of 2 numbers; do not use as function argument
	abs(a)		a = abs(a);	absolute value of a number; do not use as function argument
	constrain(x, low, high)		a = constrain(a, 10, 100);	clip value to range [low;high]; do not use as function argument
	map(x,inMin,inMax,outMin,outMax)		b = map(a, 0,1024, 0,100);	re-map a number from one range to another
	pow(x,y)		y = pow(x, 0.3)	Calculates the value of a number raised to a power.
	sqrt(x)		y = sqrt(x)	Calculates the square root of a number.
	sin(a)		y = sin(x);	Calculates the sine of an angle (in radians). The result is in [-1;1].
	cos(a)		y = cos(x);	Calculates the cosine of an angle (in radians). The result is in [-1;1].
	tan(a)		y = tan(x);	Calculates the tangent of an angle (in radians). The result is in [-inf;inf]
	isAlphaNumeric(a)		if (isAlphaNumeric(a))	Analyse if a char is alphanumeric.
	isAlpha(a)		if (isAlpha(a))	Analyse if a char is is alpha.
	isAscii(a)		if (isAscii(a))	Analyse if a char is ASCII.
	isWhitespace(a)		if (isWhitespace(a))	Analyse if a char is a white space.
	isControl(a)		if (isControl(a))	Analyse if a char is a control character.
	isDigit(a)		if (isDigit(a))	Analyse if a char is a digit.
	isGraph(a)		if (isGraph(a))	Analyse if a char is a printable character.
	isLowerCase(a)		if (isLowerCase(a))	Analyse if a char is a lower case character.
	isPrintable(a)		if (isPrintable(a))	Analyse if a char is a printable character.
	isPunct(a)		if (isPunct(a))	Analyse if a char is punctuation character.
	isSpace(a)		if (isSpace(a))	Analyse if a char is a space character.
	isUpperCase(a)		if (isUpperCase(a))	Analyse if a char is a upper case character.
	isHexadecimalDigit(a)		if (isHexadecimalDigit(a))	Analyse if a char is a valid hexadecimal digit.
	randomSeed(d)		randomSeed(10);	seed the random number generator used by the random()
	random()		a = random();	generate a pseudo random number within [0;INT16_MAX]
	lowByte(x)		LB = lowByte(x);	Extracts the low-order (rightmost) byte of a variable (e.g. a word)
	highByte		HB = highByte(x);	Extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type).
	bitRead(byte, bit)		a = bitRead(b, 4)	read single bit position in byte
	bitWrite(byte, bit, value)		bitWrite(a, 3, 1);	set single bit value in byte to value
	bitSet(byte, bit)		bitSet(a, 3);	set single bit in data to '1'
	bitClear(byte, bit)		bitClear(a, 3);	clear single bit in data to '0'
	bitToggle(byte, bit)		bitToggle(a, 3);	toggle single bit state in byte
	bit(n)		a = bit(3);	calculate bit value of bit n
	interrupts()		interrupts();	Globally enable interrupts
	noInterrupts()		noInterrupts()	Globally disable interrupts
	round(x)		a = round(a);	round x to the nearest integer
	ceil(x)		a = ceil(a);	round x upwards to the nearest integer
	floor(x)		a = floor(a);	round x downwards to the nearest integer
	toASCII(c)		c = toASCII(c);	return lower 7 bits of 1B argument (ASCII range)
	toUpperCase(c)		c = toUpperCase(c);	converts an alpha to upper case letter

	toLowerCase(c)		c = toLowerCase(c);	converts an alpha to lower case letter
	log2(d)		n = log2(d)	Integer calculation of (rough) log2(x), i.e. determine binary power to reach number
	floatToString(buf, value, digits)	USE_FTOA	printf("%s\n", floatToString(str,x,3));	convert float to string for printing floats. No scientific notation. Is rather large → only include if required
binary.h (auto loaded)	B00000000 … B11111111	none	A = B11001100	binary literals
gpio (auto loaded)	pinMode(port, pin, mode)	none	pinMode(PORT_H, pin3, OUTPUT);	Set pin direction and optional features. Pin modes are INPUT, INPUT_INTERRUPT, INPUT_PULLUP, INPUT_PULLUP_INTERRUPT OUTPUT, OUTPUT_OPENDRAIN
	pinSet(port, pin)		pinSet(PORT_H, pin3) = state;	Set pin state
	pinRead(port, pin)		state = pinRead(PORT_D, pin7);	Read pin state
	portSet(port)		portSet(PORT_H) = portState;	Set port state (8 pins)
	portRead(port)		portState = portRead(PORT_H);	Read port state (8 pins)
	attachInterruptPort(portAddr, fctName, edge)	USE_PORT_ISR	attachInterruptPort(&PORT_E, fct, FALLING);	Attach user routine to port interrupt (=EXINTx). Edges are LOW, CHANGE, RISING, FALLING, PREV_SETTING Enable pin interrupt via pinMode()
	detachInterruptPort(portAddr)		detachInterruptPort(&PORT_E);	Detach user routine from port interrupt (=EXINTx). Disable pin interrupt via pinMode()
	attachInterruptPin(fctName, edge)	USE_TLI_ISR	attachInterruptPin(fct, FALLING);	Attach user routine to pin D7 interrupt (=TLI). Edges are LOW, CHANGE, RISING, FALLING, PREV_SETTING Enable pin interrupt via pinMode()
	detachInterruptPin()		detachInterruptPin();	Detach user routine from pin D7 interrupt (=TLI). Disable pin interrupt via pinMode()
sw_delay (auto loaded)	sw_delay(uint32_t N)	none	sw_delay(10);	Delay code for approximately N milliseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h
	sw_delayMicroseconds(uint16_t N)		delayMicroseconds(10);	Delay code for approximately N microseconds without timer. Timing depends on interrupt load (inline blocking) For compiler / optimization dependent latency see sw_delay.h
	sw_delayNOP(uint8_t N)		sw_delayNOP(100);	Delay code for Nx NOP() (inline blocking) For compiler / optimization dependent latency see sw_delay.h
stm8as (auto loaded)	ASM(mnem)	none	ASM("trap");	Inline STM8 assembler
	NOP		NOP;	NOP operation (1 CPU cycle)
	WAIT_FOR_INTERRUPT		WAIT_FOR_INTERRUPT;	Halt core with clock running. Resume execution, e.g. by timer interrupt
	ENTER_HALT		ENTER_HALT;	Halt core and clock. Resume execution e.g. by auto-wakeup, see “awu“
timer4 (auto loaded)	uint32_t millis()	none	time_ms = millis();	Milliseconds since start of program
	uint32_t micros()		time_us = micros();	Microseconds since start of program with 4µs resolution
	flagMilli()		if (flagMilli())	Check if 1ms has passed. Reset by clearFlagMilli()
	clearFlagMilli()		clearFlagMilli();	Reset flagMilli() flag for 1ms
	resetTime()		resetTime();	Reset millis and micros to 0
	attachInterruptMillis(fct)	USE_MILLI_ISR	attachInterruptMillis(fct);	Attach user routine to 1ms interrupt (=TIM4UPD)
	detachInterruptMillis()		detachInterruptMillis();	Detach user routine from 1ms interrupt (=TIM4UPD)
eeprom	read_1B(addr) / read_2B(addr) / read_4B(addr)	none	A = read_1B(0x4000);	read byte, word or long word from flash, EEPROM, RAM
	write_1B(adr, val) / write_2B(adr, val) / write_4B(adr, val)		write_2B(0x0020, ‘a’);	write byte, word or long word to RAM. Flash and EEPROM see below
	OPT_writeByte(adr, data)		OPT_writeByte(OPT2, 0x00);	write option byte and return if value has changed
	OPT_setDefault()		OPT_setDefault();	set all option bytes to default. On change trigger reset
	OPT_setBootloader()		OPT_setBootloader(true);	activate BSL via option byte. On change trigger reset
	OPTx / NOPTx		see OPT_write	name of option byte. See stm8as.h
	flash_writeByte(adr, val)		flash_writeByte(0xFFF0, val);	write 1B to P-flash using physical address
	EEPROM_writeByte(num, val)		EEPROM_writeByte(10, val);	write 1B to EEPROM using logical address
	EEPROM_readByte(num)		A = EEPROM_readByte(10);	read 1B from EEPROM using logical address
	flash_eraseBlock(adr) (Cosmic only)		flash_eraseBlock(0xFFF0);	erase 128B P-flash block. Requires RAM execution → Cosmic only
	flash_writeBlock(adr, buf) (Cosmic only)		flash_writeBlock(0xFFF0, buf);	write 128B P-flash block. Requires RAM execution → Cosmic only
	PFLASH_START / PFLASH_END / PFLASH_SIZE			physical address start, end and size [B] of P-flash
	EEPROM_START / EEPROM_END / EEPROM_SIZE			physical address start, end and size [B] of EEPROM
uart1_blocking	UART1_begin(baudrate)	none	UART1_begin(19200);	initialize UART1 baudrate and enable sender & receiver
	UART1_end()		UART1_end();	disable sender & receiver
	UART1_listen()		UART1_listen();	enable sender & receiver. Retain previous settings
	UART1_write(data)		UART1_write(c);	send 1 byte via UART1
	UART1_writeBytes(num, buf);		UART1_writeBytes(num, buf);	send N bytes via UART1
	UART1_available()		if (UART1_available())	check if byte received via UART1
	UART1_read()		Rx = UART1_read();	read byte from UART1 receive buffer. Non-blocking
putchar	putcharAttach(fct)	none	putcharAttach(UART1_write);	set send routine (1B) for stdio putchar / printf; For printing floats, use float2str() helper routine
	putcharDetach()		putcharDetach();	detach send routine from stdio putchar / printf
getchar	getcharAttach(fct)	none	getcharAttach(UART1_readBlock);	set receive routine (1B) for stdio getchar / gets
	getcharDetach()		getcharDetach();	detach receive routine from stdio getchar / gets

tone (requires option byte change)	tone(uint16_t Hz, uint16_t millis)	none	beep(2000, 500);	play tone via beeper module with given frequency in Hz (<500 off) and duration in millis (0=forever)
	noTone()		noTone()	switch off tone started with tone() and duration=0 (see above)