MODIC TOW	Module	Function / Macro / Constant	#define in File "config.h"	Example	Short Description / Remark
MODIC TOW		setup()		setup()	user initialization routine. Called once after start of program
Teach Section Teach Te	main_general.h	loop()	none	loop()	user loop routine. Called continuously
		HIGH / LOW		LED = HIGH;	constants for 1 / 0, e.g. for pinSet()
		true / false		if (a==true)	constants for 1 / 0, e.g. for if
Contraction and Section 10		boolean		boolean a;	Boolean variable. Same as uint8_t
b = card(s);		string		string s[20];	Character array. Same as char*
Comment of white the Desire disting types		char(d)		c = char(d);	Converts a value to the char data type. Same as ((char) d)
WestConcept(Dist)		byte(d)		b = char(d);	Converts a value to the byte data type. Same as ((uint8_t) a)
### 000-000-000-000-000-000-000-000-000-		int()		d = int(c);	Converts a value to the int data type.
Description		word(a)		w = word(a);	Convert a value to the word data type.
Part		wordConcat(hb,lb)		w = wordConcat(hb, lb);	Convert a word from two bytes.
max.l.a.b.		long(c)		d = long(c);	Converts a value to the long data type.
Racio(3) Racio(float(d)		f = float(d);	Converts a value to the float data type.
Bobic B		min(a,b)		a = min(b,c);	minimum of 2 numbers; do not use as function argument
A		max(a,b)		a = max(b,c);	maximum of 2 numbers; do not use as function argument
Description		abs(a)		a = abs(a);	absolute value of a number; do not use as function argument
Procedy Proc		constrain(x, low, high)		a = constrain(a, 10, 100);	clip value to range [low;high]; do not use as function argument
Sq. (C) Sq.		map(x,inMin,inMax,outMin,outMax)		b = map(a, 0,1024, 0,100);	re-map a number from one range to another
Sq. (C) Sq.					
Seption Sept					
Secolar				y = sin(x);	Calculates the sine of an angle (in radians). The result is in [-1;1].
skiphakumeric(a) skiphakue at char is a siphakumeric if (skiphakueric(a)) skiphakue fila char is a siphakueric skiphakueric if (skiphakueric(a)) skiphakue fila char is a siphakueric skiphakueric if (skiphakueric(a)) skiphakue fila char is a siphakueric skiphakueric if (skiphakueric(a)) skiphakueri fila char is a siphakueric skiphakueric if (skiphakueric(a)) skiphakueri fila char is a siphakueric skiphakueric if (skiphakueric(a)) skiphakueri fila char is a siphakueric if (skiphakueric(a) skiphakueric if (skiphakueric(a) skiphakueri		cos(a)		y = cos(x);	Calculates the cosine of an angle (in radians). The result is in [-1;1].
International processing		tan(a)		y = tan(x);	Calculates the tangent of an angle (in radians). The result is in [-inf;inf]
if (laAcel(a)		isAlphaNumeric(a)	•	if (isAlphaNumeric(a))	Analyse if a char is alphanumeric.
if (isWnitespace(a)) Scortol(a) scortol(isAlpha(a)		if (isAlpha(a))	Analyse if a char is is alpha.
scorrol(a)		isAscii(a)		if (isAscii(a))	Analyse if a char is ASCII.
misc.h no loaded scape(a) Scraph(a) Scraph		isWhitespace(a)		if (isWhitespace(a))	Analyse if a char is a white space.
if (sDgit(a) Analyse if a char is a digit. Score(Case(a) If (sDript(a)) Analyse if a char is a printable character.		isControl(a)		if (isControl(a))	Analyse if a char is a control character.
if (isGraph(a) Analyse if a char is a printable character. sPrintable(a) siture(case(a)) Analyse if a char is a printable character. if (isPrintable(a)) Analyse if a char is a printable character. if (isPrintable(a)) Analyse if a char is a printable character. if (isPrintable(a)) Analyse if a char is a printable character. if (isPrintable(a)) Analyse if a char is a printable character. if (isPrintable(a)) Analyse if a char is a space character. if (isPrintable(a)) Analyse if a char is a space character. if (isPrintable(a)) Analyse if a char is a space character. if (isPrintable(a)) Analyse if a char is a space character. if (isPrintable(a)) Analyse if a char is a valid hexadecimal digit. inandomSeed(d) Analyse if a char is a valid hexadecimal digit. inandomSeed(d): seed the random number generator used by the random() a a random(); generate a pseudo random number within (io,NT16_MAX) bit (io) B	misc h	isDigit(a)	none	if (isDigit(a))	Analyse if a char is a digit.
sPrintable(a) sPrintable(a) sPunct(a) sPunct(a	auto loaded)	isGraph(a)		if (isGraph(a))	Analyse if a char is a printable character.
if (isPunct(a)) Analyse if a char is punctuation character. if (isSpace(a)) Analyse if a char is a space character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a valid hexadecimal digit. if (isUpperCase(a)) Analyse if a char is a valid hexadecimal digit. If (isUpperCase(a)) Analyse if a char is a valid hexadecimal digit. If (isUpperCase(a)) Extracts the high-order (eithrost) byte of a variable (e.g. a word) Character. Analyse if a char is a valid hexadecimal digit. Character the chardom number upper and order is a upper case in the generator upper character. Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word) Character (eithrost) byte of a variable (e.g. a word)		isLowerCase(a)		if (isLowerCase(a))	Analyse if a char is a lower case character.
if (isSpace(a)) Analyse if a char is a space character. if (isUpperCase(a)) Analyse if a char is a space character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(a)) Analyse if a char is a upper case character. if (isUpperCase(c)) calculate if uchar is a upper case character. if (isUpperCase(c)) calculate in upper case character. if (isUpperCase(c)) calculate in upper case character. if (isUpperCase(c)) calculate bit value of bit in data to '1' Characteristic in upper case integer a calculate bit value of bit in data to '0' Characteristic in upper case integer a calculate bit value of bit in data to '0' Characteristic interrupts Characteristic integer a calculate bit value of bit in data to '0' Characteristic interrupts Characteristic integer calculate bit value of bit in data to '0' Characteristic integer calculate bit value of bit in data to '0' Characteristic integer calculate bit value of bit in data to '0' Characteristic integer calculate bit value of bit in data to '0' Characteristic integer a calculate bit value of bit in data to '0' Characteristic integer a calculate bit value of		isPrintable(a)		if (isPrintable(a))	Analyse if a char is a printable character.
sUpperCase(a) sityperCase(b) sityperCase(a) sityperCase(a) sityperCase(a) sityperCase(a) sityperCase(a) sityperCase(b) sityperCase(a) sityperCase(b) sityperCase(b) sityperCase(c) seed the random number generator used by the random() a = random(); generate a pseudo random number within [0;NT16_MAX] lowByte(x) bigenerate a pseudo random number within [0;NT16_MAX] LB = lowByte(x); Extracts the low-order (rightmost) byte of a variable (e.g. a word) sityperCase(c) bitRead(byte, bit) bitRead(b, 4) read single bit position in byte chain bitRead(b, 4) read single bit in data to "1" chain bitRead(byte, bit) bitRead(byte, bit)		isPunct(a)		if (isPunct(a))	Analyse if a char is punctuation character.
if (isHexadecimalDigit(a) random/Seed(d) random/Seed(d) random/Seed(d) random/Seed(d) random/Seed(10); seed the random number generator used by the random() a = random(): generate a pseudo random number within [0:NT16_MAX] LB = lowByte(x): Extracts the low-order (rightmost) byte of a variable (e.g. a word) highByte biRead(byte, bit) biRead(byte, bit		isSpace(a)		if (isSpace(a))	Analyse if a char is a space character.
randomSeed(d) random() a = random(): generate a pseudo random number generator used by the random() a = random(): generate a pseudo random number within [0;INT16_MAX] LB = lowByte(x): Extracts the low-order (rightmost) byte of a variable (e.g. a word) The bit(byte, bit) bit(Read(byte, bit) bit(Read(byte, bit) bit(Seaf(byte, bit) bit(Clear(byte, bit) bit(Clear(byte, bit) bit(Clear(byte, bit) bit(Oper(byte, bit) calculate bit value of bit n chalculate bit value of bit n chalculate bit value of bit n calculate bit value of bit n chalculate bit value		isUpperCase(a)		if (isUpperCase(a))	Analyse if a char is a upper case character.
random() a = random(); generate a pseudo random number within [0;INT16_MAX] LB = lowByte(x); Extracts the low-order (rightmost) byte of a variable (e.g. a word) Chapter (rightmost) byte of a variable (e.g. a word) HB = highByte(x); of extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type). bitRead(byte, bit) bitRead(byte, bit) bitWrite(byte, bit, value) bitSet(byte, bit) bitSet(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(a, 3); oggle single bit in data to '0' Chapter (bit and byte of bit in data to '0' chapter (bit and byte of bit and byte of bit in data to '0' chapter (bit and byte of bit and byte of bit in data to '0' chapter (bit and byte of bit and byte o		isHexadecimalDigit(a)		if (isHexadecimalDigit(a))	Analyse if a char is a valid hexadecimal digit.
LB = lowByte(x); Extracts the low-order (rightmost) byte of a variable (e.g. a word) Chambighete		randomSeed(d)		randomSeed(10);	seed the random number generator used by the random()
highByte bitRead(byte, bit) bitRead(byte, bit) bitRead(byte, bit) bitRead(byte, bit) bitSet(byte, bit) bitSet(byte, bit) bitIclear(byte, bit) bitIclear(byte, bit) bitIclear(byte, bit) bitIclear(byte, bit) bitIclear(byte, bit) bitIclear(a, 3); bitIclear(a, 3); bitIclear(a, 3); bitIclear(byte, bit) bitIclear(a, 3); bitIclear(a, 4		random()		a = random();	generate a pseudo random number within [0;INT16_MAX]
bitRead(byte, bit) bitRead(byte, bit) bitRead(byte, bit) bitSet(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitClear(a, 3); bitClear(byte, bit) bitClear(a, 3); clear single bit in data to '1' Cha bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(a, 3); calculate bit value of bit n cha calculate bit value of bit n calculate bit value of bit n cha calculate		lowByte(x)		LB = lowByte(x);	Extracts the low-order (rightmost) byte of a variable (e.g. a word)
bitRead(byte, bit) a = bitRead(b, 4) read single bit position in byte ChabitWrite(byte, bit, value) bitSet(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitToggle(byte, bit) bitToggle(a, 3); boggle single bit in data to '0' Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); calculate bit value of bit n Chabit bitToggle(a, 3); boggle single bit state in byte Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit in data to '1' Chabit bit value of bit n Chabit bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bitToggle(a, 3); boggle single bit value of bit n Chabit bit bit bit bit byte a = bit(3); calculate bit value of bit n Chabit bit value of bit n Chabit bit byte bit bit bit bit byte or aliae bit value of bit n Chabit bit byte bit bit bit bit bit byte bit bit bit byte bit bit bit byte bit bit bit bit byte bit bit bit byte bit bit bit byte bit bit bit bit byte bit bit bit byte bit bit bit byte bit bit bit bit byte bit bit bit byte bit bit bit byte bit bit bit bit bit bit bit bit byte bit bit bit byte bit byte bit bi		highByte		HB = highByte(x);	
bitSet(byte, bit) bitClear(byte, bit) bitClear(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(a, 3); clear single bit in data to '0' Cha bitToggle(byte, bit) bitToggle(a, 3); toggle single bit state in byte Cha claudate bit value of bit n Cha interrupts() noInterrupts() noInterrupts() coll(x) a = round(a); round x to the nearest integer coll(x) a = cell(a); round x upwards 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 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)); set single bit in data to '1' Cha Cha Cha Cha Cha Cha Cha Ch		bitRead(byte, bit)		a = bitRead(b, 4)	
bitClear(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bitToggle(byte, bit) bit(n) a = bit(3); calculate bit value of bit n characteristics interrupts(); noInterrupts(); noInterrupts() cell(x) a = round(a); round x to the nearest integer a = cell(a); round x upwards to the nearest integer toASCII(c) toUpperCase(c) toLowerCase(c) converts an alpha to upper case letter log2(d) n = log2(d) n = log2(d) n = log2(d) printf("%s\n", floatToString(str.x.3)); logary library l		bitWrite(byte, bit, value)		bitWrite(a, 3, 1);	set single bit value in byte to value
bitToggle(byte, bit) bit(n) a = bit(3); calculate bit value of bit n interrupts(); Globally enable interrupts noInterrupts(); Globally enable interrupts round(x) a = round(a); round x to the nearest integer ceil(x) a = round(a); round x upwards to the nearest integer floor(x) a = floor(a); round x downwards to the nearest integer toASCII(c) toUpperCase(c) toLowerCase(c) c = toLowerCase(c); converts an alpha to upper case letter log2(d) n = log2(d) floatToString(buf, value, digits) USE_FTOA printf("%s\n", floatToString(str,x,3)); same binary leave to require frequired binary.h poocooooo P11111111 poocoooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocooooooooooooooooooooooooo		bitSet(byte, bit)		bitSet(a, 3);	set single bit in data to '1'
bit(n) a = bit(3); calculate bit value of bit n Challed interrupts() noInterrupts() noInterrupts() noInterrupts() ceil(x) floor(x) toASCII(c) toUpperCase(c) toLowerCase(c) log2(d) floatToString(buf, value, digits) a = bit(3); calculate bit value of bit n Challed interrupts Globally enable interrupts noInterrupts() Globally disable interrupts a = round(a); round x to the nearest integer round x upwards to the nearest integer round x downwards to the nearest integer round x downwards to the nearest integer c = toASCII(c); return lower 7 bits of 1B argument (ASCII range) c = toUpperCase(c); converts an alpha to upper case letter 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		bitClear(byte, bit)		bitClear(a, 3);	clear single bit in data to '0'
interrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() a = round(a); round x to the nearest integer nound x upwards to the nearest integer nound x upwards to the nearest integer toASCII(c) toASCII(c) toUpperCase(c) toLowerCase(c) toLowerCase(c) log2(d) n = log2(d) floatToString(buf, value, digits) use_FTOA printf("%s\n", floatToString(str,x,3)): new A = R11001100 pinary.lh processors pinary interger pinary		bitToggle(byte, bit)		bitToggle(a, 3);	toggle single bit state in byte
interrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() noInterrupts() a = round(a); round x to the nearest integer nound x upwards to the nearest integer nound x upwards to the nearest integer toASCII(c) toASCII(c) toUpperCase(c) toLowerCase(c) toLowerCase(c) log2(d) n = log2(d) floatToString(buf, value, digits) use_FTOA printf("%s\n", floatToString(str,x,3)): new A = R11001100 pinary.lh processors pinary interger pinary					
noInterrupts() noInterrupts() round(x) a = round(a); round x to the nearest integer a = ceil(a); round x upwards to the nearest integer floor(x) toASCII(c) toUpperCase(c) toLowerCase(c) toLowerCase(c) log2(d) floatToString(buf, value, digits) round x interprets(ase(c); converts an alpha to upper case letter linteger calculation of (rough) log2(x), i.e. determine binary power to reach number new A = R11001100 A = R11001100 pinary.h poocooooo R11111111				* *	
round(x) a = round(a); round x to the nearest integer floor(x) a = floor(a); round x downwards to the nearest integer toASCII(c) toUpperCase(c) toLowerCase(c) toLowerC					
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) toUpperCase(c) toUpperCase(c) toLowerCase(c) toConverts an alpha to lower case letter integer calculation of (rough) log2(x), i.e. determine binary power to reach number tocovert float to string for printing floats. No scientific notation. is rather large — only include if required					
a = floor(a); round x downwards to the nearest integer					
toASCII(c) c = toASCII(c); return lower 7 bits of 1B argument (ASCII range) c = toUpperCase(c); c converts an alpha to upper case letter toLowerCase(c) c = toLowerCase(c); c converts an alpha to lower case letter lnteger 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)); and A = R11001100 binary.h proposed A = R11001100 binary.literale					
toUpperCase(c) c = toUpperCase(c); converts an alpha to upper case letter toLowerCase(c) log2(d) floatToString(buf, value, digits) USE_FTOA printf("%s\n", floatToString(str,x,3)); converts an alpha to lower case letter linteger calculation of (rough) log2(x), i.e. determine binary power to reach number new binary.h poocooooo P11111111 poocoooo P11111111 poocooooo P11111111 poocooooo P111111111 poocooooo P111111111 poocooooo P111111111 poocooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocoooooo P111111111 poocooooooooooooooooooooooooo					
toLowerCase(c) c = toLowerCase(c); converts an alpha to lower case letter 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)); one of the proposition of the printing floats. No scientific notation. Is rather large — only include if required printing float to string for printing floats. No scientific notation. Is rather large — only include if required					
log2(d) n = log2(d) 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 A = R11001100 A = R11001100 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					
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 new binary.h				**	Integer calculation of (rough) log2(x), i.e. determine binary power to reach
binary.h p0000000 P11111111 pope A = P11001100 binary literals			USE ETOA		convert float to string for printing floats. No scientific notation.
B00000000 B11111111 none	hinon: h		USL_FIOA		is rather large - only include if required
	binary.h (auto loaded)	B00000000 B11111111	none	A = B11001100	binary literals new

	1				٦
	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	1
	portSet(port)		portSet(PORT_H) = portState;	Set port state (8 pins)	1
gpio (auto loaded)	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(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	fix re-entr
sw_delay (auto loaded)	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	calibrate t
	sw_delayNOP(uint8_t N)		sw_delayNOP(100);	Delay code for Nx NOP() (inline blocking) For compiler / optimization dependent latency see sw_delay.h	
	ASM(mnem)	none	ASM("trap");	Inline STM8 assembler	
stm8as	NOP		NOP;	NOP operation (1 CPU cycle)	change fr
(auto loaded)	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"	
	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	
timer4 (auto loaded)	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)	
	UART1_begin(baudrate)	none	UART1_begin(19200);	initialize UART1 baudrate and enable sender & receiver	new
uart1_blocking	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	new
	UART1_writeBytes(num, buf);		UART1_writeBytes(num, buf);	send N bytes via UART1	new
	UART1_available()		if (UART1_available())	check if byte received via UART1	new
	UART1_read()		Rx = UART1_read();	read byte from UART1 receive buffer. Non-blocking	new
putchar	putcharAttach(fct)	none	putcharAttach(UART1_write);	set send routine (1B) for stdio putchar / printf; For printing floats, use float2str() helper routine	new
	putcharDetach()		putcharDetach();	detach send routine from stdio putchar / printf	new
getchar	getcharAttach(fct)	none -	getcharAttach(UART1_readBlock);	set receive routine (1B) for stdio getchar / gets	new
	getcharDetach()		getcharDetach();	detach receive routine from stdio getchar / gets	new
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)	change fr
	noTone()		noTone()	switch off tone started with tone() and duration=0 (see above)	new
	1			1	