C-Coding int variables: [u]int#_t, #=Number of bits.
Uint8_t [0,255] **int8_t** [-128,127] **uint16_t** [0,65535] **int16_t** [-32768,32767]
uint32_t [0,4294967295] **int32_t** [-2147483648,2147483647]
float: 32-bit **±**[$1.4*10^{-45}$,$3.4*10^{38}$] Digit Accuracy:6
double: 64-bit **±**[$4.9x10^{-324}$,$1.8x10^{308}$] Digit Accuracy:15
Bitwise Operations: & | ^ ~
Logical Operations: ! && || == != > < >= <= (Result: true 1 / false 0)
Math Operators: + - * / % **bool:**in C the bool keyword only valid in lowercase
Value Operators: ! << >> ++ --
Functions: <return type> function_name(<arg_type> in1,<arg_type> in2,...)
void no arguments or return type **DON'T FORGET TO DECLARE!**
printf("format string",var1,var2,...)
printf formats: **%u** decimal unsigned integer, **%d** decimal signed integer,
%x or %X Hexadecimal integer (no 0x added), **%c** character
%lu Unsigned Decimal Number **%ld** Signed Decimal Number, **%f** float
\n Move down line, **\r** Move to beginning of line, **\t** tab, **\b** backspace
IO functions:
void **putchar**(uint8_t val) uint8_t **getchar**() uint8_t **getchar_nw**()
val to terminal get keypress (blocking) get keypress (non-blocking)
Arrays:<type**>** arrayname[maxsize] ={};

Bit Masking: &-set bits low, **|**-set bits high
^ (Exclusive OR) toggles the value of a bit
Set low: PxOUT &= ~0x26; **Set High:**PxOUT |= 0x49;
Toggle: PxOUT ^= 0x01 (eg:0101 ^= 1111 => 1010)

X X X X X X X X	**X X X X X X X X**
& 1 1 0 1 1 0 0 1 ~0x26	**| 0 1 0 0 1 0 0 1** 0x49
= X X 0 X X 0 0 X	**= X 1 X X 1 X X 1**

Base convert:
0001=0x1 0110=0x6 1011=0xB
0010=0x2 0111=0x7 1100=0xC
0011=0x3 1000=0x8 1101=0xD
0100=0x4 1001=0x9 1110=0xE
0101=0x5 1010=0xA 1111=0xF

false=0 true=any other

GPIO Registers x=1..11 (port#)
PxDIR: 0-Input,1-Output
PxOUT: Set state of outputs
PxIN: Read value of pins

Usually requires two bitmasking cmds.: &=,|=

Layout: Bit/Pin order: 76543210
Do not modify other bits if not necessary

GPIO DriverLib
uint8_t **GPIO_getInputPinValue**(uint8_t port,uint8_t pins)
Return GPIO_INPUT_PIN_LOW/GPIO_INPUT_PIN_HIGH
void **GPIO_setOutputLowOnPin**(uint8_t port,uint8_t pins)
void **GPIO_setOutputHighOnPin**(uint8_t port,uint8_t pins)
void **GPIO_toggleOutputOnPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsOutputPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsInputPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsInputPinWithPullUpResistor /**
void **GPIO_setAsInputPinWithPullDownResistor**
(uint8_t port,uint8_t pins)

Possible ports:
GPIO_PORT_Px
Possible pins:
GPIO_PINy

x=1..11, y=0..7

Multiple pins announcement:
GPIO_PIN0|GPIO_PIN1

GPIO Interrupt
void **GPIO_enableInterrupt**/**GPIO_disableInterrupt**(uint8_t port,uint8_t pins)
void **GPIO_interruptEdgeSelect**(uint8_t port,uint8_t pins,uint8_t edgeSelect)
edgeSelect=GPIO_LOW_TO_HIGH_TRANSITION or GPIO_HIGH_TO_LOW_TRANSITION
void **GPIO_registerInterrupt**(uint8_t port,<function_name>)
uint16_t **GPIO_getEnabledInterruptStatus**(uint8_t port)
returns bitwise OR of pins that triggered interrupt (eg. GPIO_PIN1|GPIO_PIN3)
void **GPIO_clearInterruptFlag**(uint8_t port,uint8_t pins)

Debouncing: __delay_cycles(#) to wait # of SMCLK cycles.
How to calc # for delay time:delay_time/(1/freq) 1 MHz = 1000000 Hz

Other Functions:
Absolute value: int32_t abs(int32_t number)
Round up: double ceil(double number), **Round down:** double floor(double number)
Random number:uint32_t rand(), Seed random number: void srand(uint32_t seed)

Timer:
struct Timer_A_UpModeConfig fields:
 .clockSource = TIMER_A_CLOCKSOURCE_x
  x=EXTERNAL,ACLK,SMCLK,INVERTED_EXTERNAL_TXCLK
 .clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_y
  y=1,2,3,4,5,6,7,8,10,12,14,16,20,24,28,32,40,48,56,64
 .timerPeriod = 0 to 65535 (Sets value of CCR0)
 .timerClear = TIMER_A_v_CLEAR,TIMER_A_v_CLEAR v=DO,SKIP
 .timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE or DISABLE

timer=TIMER_Ax_BASE, x=0..3
void **Timer_A_configureUpMode**(uint32_t timer,Timer_A_UpModeConfig *config)
void **Timer_A_startCounter**(uint32_t timer , uint16_t timerMode)
  timerMode=TIMER_A_UP_MODE,TIMER_A_UPDOWN_MODE,TIMER_A_CONTINUOUS_MODE
void **Timer_A_stopTimer**(uint32_t timer)
void **Timer_A_clearTimer**(uint32_t timer)
uint16_t **Timer_A_getCounterValue**(uint32_t timer)

Timer Interrupt
void **Timer_A_enable/disableInterrupt**(uint32_t timer) <- For Timer resets
void **Timer_A_registerInterrupt**(uint32_t timer ,
  TIMER_A_CCRX_AND_OVERFLOW_INTERRUPT , <function name>)
uint32_t **Timer_A_get**[**Enabled**]**InterruptStatus**(uint32_timer)
  Returns either TIMER_A_INTERRUPT_PENDING or TIMER_A_INTERRUPT_NOT_PENDING
void **Timer_A_clearInterruptFlag**(uint32_t timer)

$$f_{TCLK} = \frac{f_{SMCLK}}{N_{div}} \qquad T_{TCLK} = N_{div} T_{SMCLK}$$

$$T_{timer} = N_{timer} T_{TCLK} \qquad N_{timer} = 1 + N_{max} = 1 + CCR0$$