Bits and Shifts

C Language Programming

Selected Topics

Logical and Bitwise Operators

Logical Operators

- A logical operator is used to combine 2 or more conditions in an expression.
- Logical AND - &&
  - Operator **&&** returns true when both the conditions in consideration are true; else false
- Logical OR - ||
  - Operator **||** returns true when either or both the conditions in consideration are true; else false
- Logical NOT - !
  - Operator **!** returns true when either or both the conditions in consideration are true; else false
- Logical XOR
  - In the Boolean sense, this is just **!=** **(not equal)**

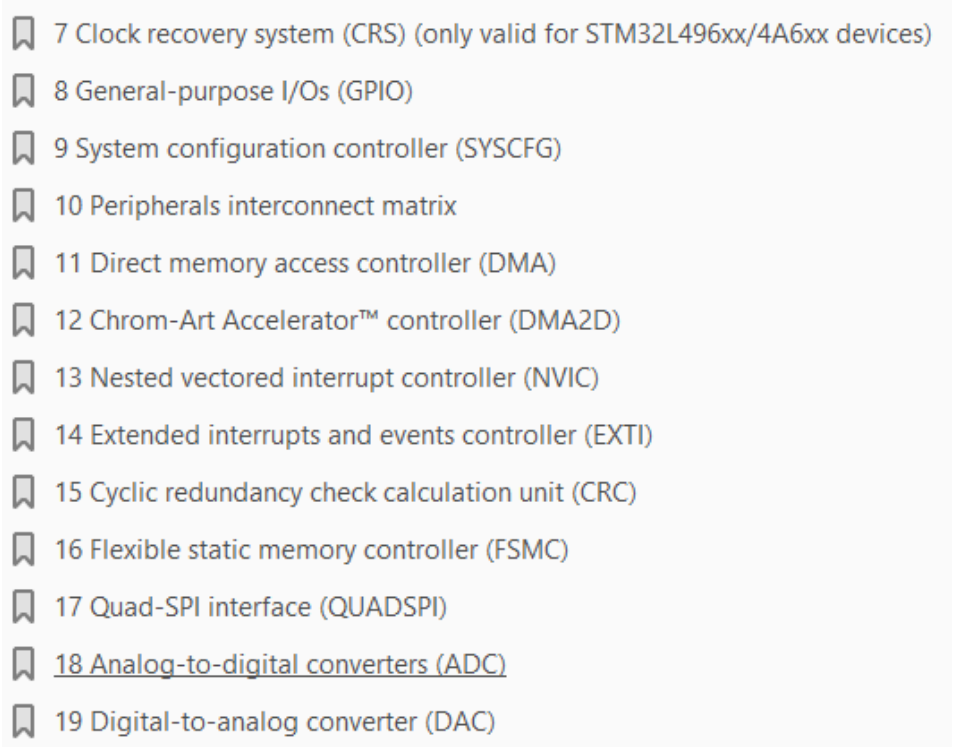
Logical example

```
int a = 10, b = 4, c = 10, d = 20;

// logical AND example
if (a > b && c == d)
    printf("a is greater than b AND c is equal to d\n");
    // doesn't print because c != d

// logical OR example
if (a > b || c == d)
    printf("a is greater than b OR c is equal to d\n");
    // NOTE: because a>b, the clause c==d is not evaluated

// logical NOT example
if (!a)
    printf("a is zero\n");    // doesn't print because a != 0
```

Software Engineering
Rochester Institute of Technology

Bitwise Operators

- A key feature of C essential to RT & ES programming is the set of bit manipulations
- Microcontrollers are filled with pages and pages of registers that control MCU peripheral hardware. These are all bit-based definitions.
- Some peripherals from STM32 Reference Manual…



23.5 OPAMP registers

23.5.1 OPAMP1 control/status register (OPAMP1_CSR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OPA_RANGE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAL OUT	USER TRIM	CAL SEL	CALON	Res.	VP_SEL	VM_SEL		Res.	Res.	PGA_GAIN		OPAMODE		OPA LPM	OPAEN
r	rw	rw	rw		rw	rw	rw			rw	rw	rw	w	rw	rw

Bit 31 **OPA_RANGE:** Operational amplifier power supply range for stability
All AOP must be in power down to allow AOP-RANGE bit write. It applies to all AOP embedded in the product.
0: Low range (VDDA < 2.4V)
1: High range (VDDA > 2.4V)

Bits 30:16 Reserved, must be kept at reset value.

Bit 15 **CALOUT:** Operational amplifier calibration output
During calibration mode offset is trimmed when this signal toggle.

Bit 14 **USERTRIM:** allows to switch from 'factory' AOP offset trimmed values to AOP offset 'user' trimmed values
This bit is active for both mode normal and low-power.
0: 'factory' trim code used
1: 'user' trim code used

Bit 13 **CALSEL:** Calibration selection
0: NMOS calibration (200mV applied on OPAMP inputs)
1: PMOS calibration (VDDA-200mV applied on OPAMP inputs)

Bit 12 **CALON:** Calibration mode enabled
0: Normal mode
1: Calibration mode (all switches opened by HW)

C Bitwise Operators

C has 6 operators for performing bitwise operations on integers

Operator	Meaning	
&	Bitwise AND	Result is 1 if both bits are 1
|	Bitwise OR	Result is 1 if either bit is 1
^	Bitwise XOR	Result is 1 if both bits are different
>>	Right shift	Result is divided by 2
<<	Left shift	Result is multiplied by 2
~	Ones complement	The logical invert, same as NOT

Bitwise Boolean examples

```
char j = 11;     // 0 0 0 0 1 0 1 1 = 11
char k = 14;     // 0 0 0 0 1 1 1 0 = 14
```

Bitwise Boolean Operators

```
char m = j & k; // 0 0 0 0 1 0 1 0 = 10
char n = j | k; // 0 0 0 0 1 1 1 1 = 15
char p = j ^ k; // 0 0 0 0 0 1 0 1 = 5
```

Shifting and Inversion

Shifting

Shifting

```
char j = 11;      // 0 0 0 0 1 0 1 1 = 11
char k = j<<1;    // 0 0 0 1 0 1 1 0 = 22 (j*2)
char m = j>>1;    // 0 0 0 0 0 1 0 1 = 5  (j/2)
```

Shifting

```
char s1, s2, s3, s4;
s1=-11;                 // 1  1  1  1  0  1  0  1 -11
s2=s1>>1;               // 1  1  1  1  1  0  1  0  -6

s3=117;                 // 0  1  1  1  0  1  0  1  117
s4=s3>>1;               // 0  0  1  0  0  0  0  0  58
                    // sign extension!

unsigned   char  u1,   u2;
u1=255;                 // 1  1  1  1  0  1  0  1   245
u2=u1>>1;               // 0  1  1  1  1  1  1  1   122
                        //  no sign  extension!
```

Inversion

Logical invert

```
char j = 11;      // j = 0 0 0 0 1 0 1 1 =  11
char k = ~j;      // k = 1 1 1 1 0 1 0 0 = 244
                  // Note:         j + k = 255
```