Applied mechatronics

Low-level programming

Sven Gestegård Robertz

sven.robertz@cs.lth.se

Department of Computer Science, Lund University

2020

Outline

Boolean and bitwise operators

- Boolean operators (evaluate to `true` or `false`)
  - `!` not
  - `&&` and
  - `||` or
- Bitwise operators (evaluate to a number)
  - `~` not
  - `&` and
  - `|` or
  - `^` exclusive or (xor)

Bitwise operators, examples

$$
\begin{array}{ll}
\sim & 1111\,0101 \\
\hline
= & 0000\,1010
\end{array}
$$

$$
\begin{array}{ll}
 & 1110\,0111 \\
\& & 0111\,1100 \\
\hline
= & 0110\,0100
\end{array}
$$

$$
\begin{array}{ll}
 & 0010\,1010 \\
| & 1100\,0001 \\
\hline
= & 1110\,1011
\end{array}
$$

- & is used for *bit masking* and for *clearing* bits
- | is used for *setting* bits
- ^ is used for *toggling* bits

Masking and setting bits

Example:

A register : `unsigned char ctrl_reg;`

A bitmask : `FLAGS` (e.g., `0111 0000 == 0x70`)

Named bits : `ENABLE_X` (e.g., `0001 0000 == 0x10`), `ENABLE_Y` (`0x20`), ...

Clear the bits specified by `FLAGS` and set individual bits.

```
// clear FLAGS using bitmask
ctrl_reg = ctrl_reg & ~FLAGS;
// set individual bits
ctrl_reg = ctrl_reg | ENABLE_X | ENABLE_Y;
```

Can also be written:

```
ctrl_reg &= ~FLAGS;
ctrl_reg |= ENABLE_X | ENABLE_Y;
```

Shift operators

- In addition to the bitwise logical operators, C and Java has operators for shifting the bits in a number
- x << n shifts x n steps to the left
- x >> n shifts x n steps to the right
- Example:
  - `2 << 2 == 8` `(0b0010 << 2 == 0b1000)`
- Shifting can be viewed as multiplication or division by powers of two.

Masking and shifting

- Low level drivers often access hardware *registers* where a single, or a few, bits control something
- Example: motor servo control word (16 bits)

```
- - - posref[8] velref[4] enable
```

  - Reading and writing posref:

```
#define POS_OFF 5
#define POS_MSK 0xff
unsigned short cr;//temporary for register value
unsigned char pr; //temporary for posRef value

cr= read_ctrl_reg();

pr = (cr >> POS_OFF) & POS_MSK; //shift and mask
cr &= ~(POS_MSK << POS_OFF);//clear posRef bits
pr = pr + 10;
cr |=  (pr << POS_OFF);   //new value for register

write_ctrl_reg(cr);
```

Byte order (Endianness)

- Data types larger than one byte are often represented (in memory) and sent (on communication channels) as a *sequence of bytes*
- For data types larger than 8 bits
  - is the most significant byte sent first (big endian)
  - or last (little endian)
- The byte order is different in different architectures/systems
- Don't confuse with bit order on serial lines

Byte order, example

- Example: the number 1000 as a 16 bit integer (== `0x03E8`)

  Big endian: `0x03`, `0xE8` ("network", IP, Java)

  Little endian: `0xE8`, `0x03` (CANopen, Intel)
- Conversion is done by mask & shift.

  Be careful with sign extension (use **`unsigned`** in C).
- The same applies to values in memory

Example: Little endian memory

- "First" byte means lowest address
- Example: `*a = 0x0A0B0C0D;`