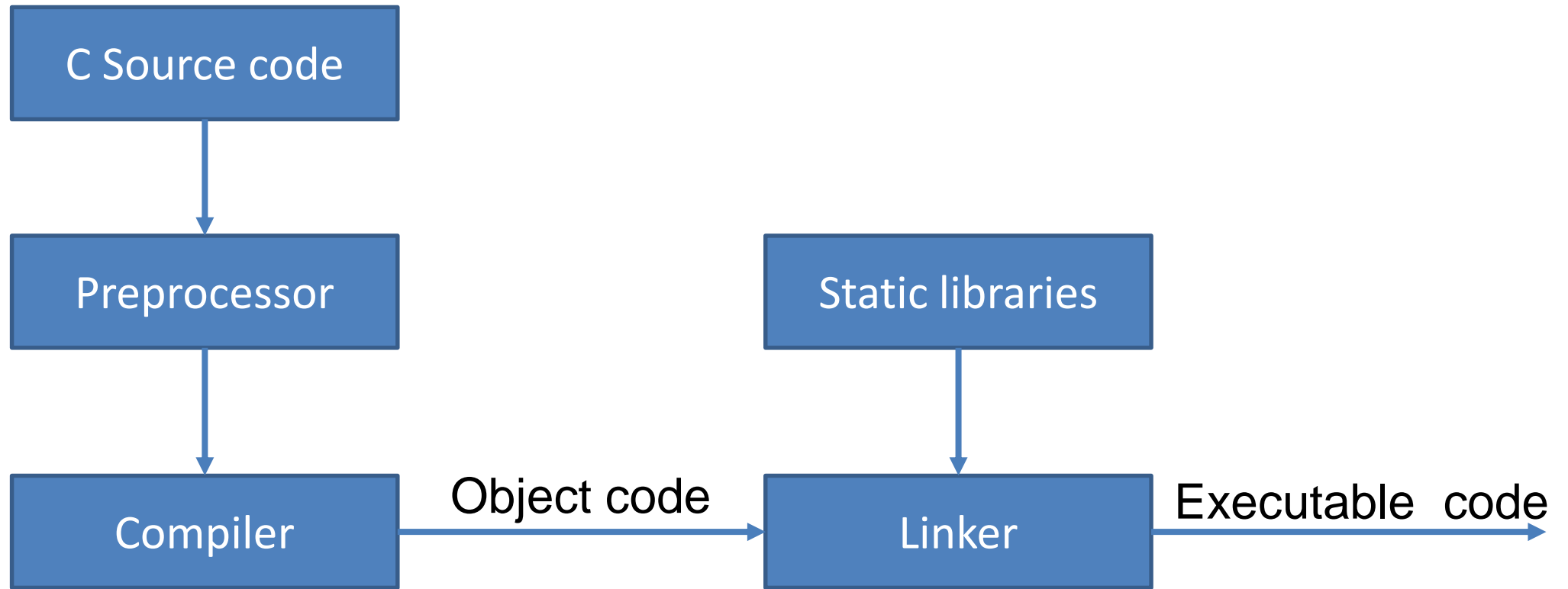


Macro and Bit Operations

- ❑ C Preprocessor Overview
- ❑ Macro
- ❑ C Preprocessor Directives
- ❑ Bit Operations

C Preprocessor Overview

C Preprocessor Overview



C Preprocessor Overview

- The C pre-processor is a macro processor that is used by the C compiler to transform our program before it is compiled. It allows to define macros.
- The preprocessing consists of preprocessor directives to be executed and macros to be expanded:
 - + Inclusion of header files.
 - + Macro expansion.
 - + Conditional compilation.
 - + Diagnostics directives.
- Preprocessing directives are lines that start with '#' symbol.

Macro

❑ What is macro

A *macro* is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro

Macro is defined using `#define` preprocessor directive in the C language.

❑ When to use

When creating constants that represent numbers, strings or expressions.

❑ Macro classification

- Predefined macro
- User-defined macro

□ Syntax

```
#define MACRO_NAME    macro's body  
#undef MACRO_NAME
```

- Give symbolic names to numeric constants
- The macro's body end at the end of the #define line
- Single line macro:

```
#define SIZE 10
```

- Multiple line macro:

```
#define NUMBERS    1, \  
                  2
```


□ Use case

- Un-define and re-define:

```
#ifdef TRUE
    #undef TRUE
    #define TRUE 1
#endif
```

□ Syntax

```
#define macro_name(list of parameters) macro's body
```

Lower
case

No white
space

- Each parameter should be guarded by parentheses.

```
#define mul(X, Y) ((X) * (Y))
```

```
x = mul(a, b); ==> x = ((a) * (b));
```

```
y = mul(1, 2); ==> y = ((1) * (2));
```

- Return wrong value if parentheses is not used for each parameter:

```
#define mul(X, Y) (X * Y)
```

```
y = mul(1+2, 2); ==> y = (1+2 * 2); /* return 5 */
```

❑ Nesting macro can be used in an macro like function:

```
#define UART0_BDH_SBR_MASK      (0x1FU)
#define UART0_BDH_SBR_SHIFT    (0U)
#define UART0_BDH_SBR(x)      (((uint8_t)(((uint8_t)(x)) << \
UART0_BDH_SBR_SHIFT)) & UART0_BDH_SBR_MASK)
```

❑ Compare macro and function:

Macro	Function
<ul style="list-style-type: none">- Macro is Pre-processed and is replaced by macro value.- Using Macro increases the code size.- Application execution is faster- Useful when it is used to replace small code many times.	<ul style="list-style-type: none">- Function is Compiled and use branch instruction when it is called.- Using Function will not increase the code size.- Application execution is lower.- Useful when large code is requested.

C Preprocessor Directives

❑ Syntax

`#include <file>`

`#include "file"`

- `<file>` is used for system header files such as `math.h`, `string.h`, `stdint.h`...
- `"file"` is used for header files of your own program.

□ Syntax

```
#if <constant-expression>  
#if defined MACRO_NAME  
#ifdef MACRO_NAME  
#if ! defined MACRO_NAME  
#ifndef MACRO_NAME  
#elif <constant-expression>  
#else  
#endif
```

□ Use case

- Avoid repeat inclusion header file:

In MKL46Z4.h:

```
#ifndef _MKL46Z4_H_
#define _MKL46Z4_H_

/* Content of the header file */
#endif /* _MKL46Z4_H_ */
```


□ Use case

- Disable and enable compile:

```
#if(UART_0_INIT_API == D_ON)
void UART0_Init()
{
    #ifdef UART_DMA_EXIST
        /* Enable DMA request */
        UART0->C5 |=  UART0_C5_RDMAE_MASK;
    #else
        /* Disable DMA request */
        UART0->C5 &=  UART0_C5_RDMAE_MASK;
    #endif
}
#endif
```

□ Syntax

```
#error "the error message"  
#warning "the warning message"
```

- For “#warning”: The pre-processor will continue pre-processing after the warning message appears.

Example:

```
#if !defined(DMA_EXIST) && defined(DMA_USED)  
    #error "The DMA resource does not exist to use."  
#endif
```

□ Syntax

<condition expression_1> ? <expression_2> : <expression_3>;

- The value of **expression_2** will be returned if **condition expression_1** is true.
- The value of **expression_3** will be returned if **condition expression_1** is false.

- Example:

```
#define min(X, Y) ((X) < (Y) ? (X) : (Y))  
x = min(a, b); ==> x = ((a) < (b) ? (a) : (b));  
y = min(1, 2); ==> y = ((1) < (2) ? (1) : (2));
```

- ❑ Stringizing operator (#)
 - Stringizing operator (#) converts macro parameters to string.
- ❑ Token pasting operator (##)
 - Used in a macro and the two tokens on either side of each '##' operator are combined into a single token.

Preprocessor Operators

□ Example:

```
struct command {  
    char *name;  
    void (*function) (void);  
};  
#define COMMAND(NAME) { #NAME, NAME##_command }  
struct command commands[] = {  
    COMMAND (quit),  
    COMMAND (help),  
};
```

Token pasting
preprocessing
operator

The result:

```
struct command commands[] = {  
    { "quit", quit_command },  
    { "help", help_command },  
};
```

Stringizing
preprocessing
operator

Bit Operations

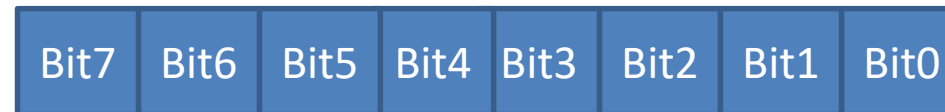
Bit Operations(1)

Symbol	Operation
&	AND
	OR
^	XOR
~	NOT
<<	Left shift
>>	Right shift

Bit A	Bit B	A & B	A B	A ^ B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

□ Shift operations

A



$A \ll 3$



$A \gg 3$



□ Examples:

A = 0xAA(0b10101010)

B = 0x0F(0b00001111)

A & B = 0x0A(0b00001010)

A | B = 0xAF(0b10101111)

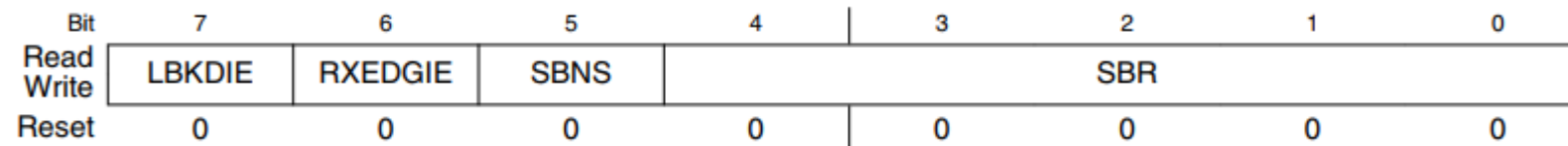
A ^ B = 0xA5(0b10100101)

~A = 0x55(0b01010101)

A << 3 = 0x50(0b01010000)

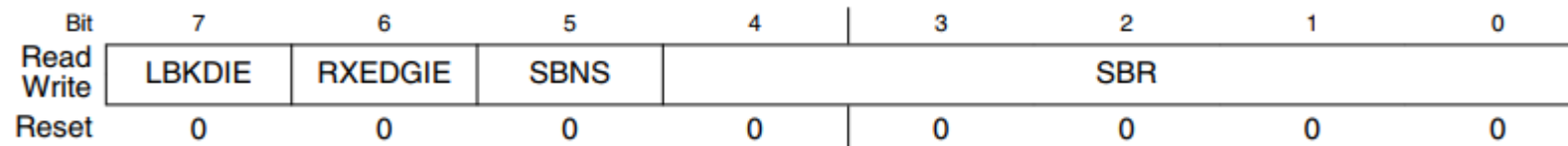
B >> 3 = 0x01(0b00000001)

□ Use case:



- Register mask for SBR bits field:
`#define UART0_BDH_SBR_MASK` (0x1FU)
`#define UART0_BDH_SBNS_MASK` (0x20U)
- Set SBNS bit:
`UART0->BDH = (UART0->BDH) | UART0_BDH_SBNS_MASK;`
- Clear SBNS bit:
`UART0->BDH = (UART0->BDH) & (~UART0_BDH_SBNS_MASK);`
- Read SBNS bit:
`Var_SBNS = (UART0->BDH) & UART0_BDH_SBNS_MASK;`

□ Use case:



- Modify SBR bits field:

Var_SBR = 0x0E;

UART0->BDH = ((UART0->BDH) & (~UART0_BDH_SBR_MASK)) |
(Var_SBR & UART0_BDH_SBR_MASK);

- Example about Peripheral Access Layer in MKL46Z4.h file

1. <https://gcc.gnu.org/>
2. <https://en.wikipedia.org/>
3. The C programming Language By Brian W. Kernighan and Dennis M. Ritchie.

Thanks for listening!