LAPIS SEMICONDUCTOR

FEXTMULDIVLIB_UM-04

MULDIVU8LIB
Multiplication and Division Library for U8/U16 Accelerator
User's Manual

ISSUE DATE : JUN. 2012

NOTICE

No copying or reproduction of this document, in part or in whole, is permitted without the consent of LAPIS Semiconductor Co., Ltd.

The content specified herein is subject to change for improvement without notice.

The content specified herein is for the purpose of introducing LAPIS Semiconductor's products (hereinafter "Products"). If you wish to use any such Product, please be sure to refer to the specifications, which can be obtained from LAPIS Semiconductor upon request.

Examples of application circuits, circuit constants and any other information contained herein illustrate the standard usage and operations of the Products. The peripheral conditions must be taken into account when designing circuits for mass production.

Great care was taken in ensuring the accuracy of the information specified in this document. However, should you incur any damage arising from any inaccuracy or misprint of such information, LAPIS Semiconductor shall bear no responsibility for such damage.

The technical information specified herein is intended only to show the typical functions of and examples of application circuits for the Products. LAPIS Semiconductor does not grant you, explicitly or implicitly, any license to use or exercise intellectual property or other rights held by LAPIS Semiconductor and other parties. LAPIS Semiconductor shall bear no responsibility whatsoever for any dispute arising from the use of such technical information.

The Products specified in this document are intended to be used with general-use electronic equipment or devices (such as audio visual equipment, office-automation equipment, communication devices, electronic appliances and amusement devices).

The Products specified in this document are not designed to be radiation tolerant.

While LAPIS Semiconductor always makes efforts to enhance the quality and reliability of its Products, a Product may fail or malfunction for a variety of reasons.

Please be sure to implement in your equipment using the Products safety measures to guard against the possibility of physical injury, fire or any other damage caused in the event of the failure of any Product, such as derating, redundancy, fire control and fail-safe designs. LAPIS Semiconductor shall bear no responsibility whatsoever for your use of any Product outside of the prescribed scope or not in accordance with the instruction manual.

The Products are not designed or manufactured to be used with any equipment, device or system which requires an extremely high level of reliability the failure or malfunction of which may result in a direct threat to human life or create a risk of human injury (such as a medical instrument, transportation equipment, aerospace machinery, nuclear-reactor controller, fuel-controller or other safety device). LAPIS Semiconductor shall bear no responsibility in any way for use of any of the Products for the above special purposes. If a Product is intended to be used for any such special purpose, please contact a ROHM sales representative before purchasing.

If you intend to export or ship overseas any Product or technology specified herein that may be controlled under the Foreign Exchange and the Foreign Trade Law, you will be required to obtain a license or permit under the Law.

Table of Contents

1. Overview

The multiplication and division library for U8 or U16 accelerator is a file which includes the routines which used the multiplication and division accelerator.

Henceforth, this library is called a muldivu8 library.

When the muldivu8 library is specified at the time of a link, the program which includes the multiplication or division operation for 16bits/32bits can transpose to the code which used an accelerator without modifying the C source code.

2. Construct of muldivu8 library

Muldivu8 library is constructed from the header file and the library file.

Table 2-1 Construct of muldivu8 library

File name	Description
muldivu8.h	The header file for C library
muldivu8.lib	The library file of muldivu8 library

The muldivu8 library contains the following.

- The operation routines which called to the operation for the multiplication or division described by the C language.
- The C library functions which support the multiplication and division accelerator.

2.1 Operation Routines List

The following list shows the operation routines which are contained in the muldivu8 library.

Table 2-2 Operation Routines List

Operation routine	Function	Bit width of operation result	Data passing	
			Argument	Return
__imulu8	signed int type multiplication unsigned int type multiplication	16 bits	Register	Register
__idivu8	signed int type division	16 bits	Register	Register
__imodu8	signed int type modulation	16 bits	Register	Register
__uidivu8	unsigned int type division	16 bits	Register	Register
__uimodu8	unsigned int type modulation	16 bits	Register	Register
__lmulu8	signed long type multiplication unsigned long type multiplication	32 bits	Stack	Stack
__ldivu8	signed long type division	32 bits	Stack	Stack
__lmodu8	signed long type modulation	32 bits	Stack	Stack
__uldivu8	unsigned long type division	32 bits	Stack	Stack
__ulmodu8	unsigned long type modulation	32 bits	Register	Stack

The compiler generates the code which calls a required operation routine according to the type of data and the kind of operation when multiplication or division is described by the C language.

When the types of the operand of operation differ, operation with the larger type of an operand is called. For example, "signed int * signed long" is converted to "signed long * signed long".

2.2 C Library Functions List

The following list shows the C library functions which are contained in the muldivu8 library.

Table 2-3 C Library Functions List

C Library Function	Function	Bit width of operation result	Data passing	
			Argument	Argument
__silmul	signed 16 bits multiplication	32 bits	Register	Register
__uilmul	unsigned 16 bits multiplication	32 bits	Register	Register
__silmac	signed 16 bits multiplication and accumulation	32 bits	Register	Register
__uilmac	unsigned 16 bits multiplication and accumulation	32 bits	Register	Register
__silmacs	signed 16 bits multiplication and accumulation with saturation mode	32 bits	Register	Register
__uilmacs	unsigned 16 bits multiplication and accumulation with saturation mode	32 bits	Register	Register
__sllmul	signed 32 bits multiplication	32 bits	Register & Stack	Register
__ullmul	unsigned 32 bits multiplication	32 bits	Register & Stack	Register

About how to use the above functions, please see "3.1 How to use C library function".

3. C Library Functions

3.1 How to use C library function

Here, how to use the function shown by "2.2 C Library Function List" is explained.

Please include the header file "muldivu8.h" when you use the C library function of muldivu8 library.

3.1.1 When using multiplication and accumulation operation

When using multiplication and accumulation operation, please use the following functions.

signed / unsigned	Function Name	Function	Remark
signed	__silmul	16 bits multiplication	When performing multiplication and accumulation operation, please call this function first.
	__silmac	16 bits multiplication and accumulation	
	__silmacs	16 bits multiplication and accumulation with saturation mode	
unsigned	__uilmul	16 bits multiplication	When performing multiplication and accumulation operation, please call this function first.
	__uilmac	16 bits multiplication and accumulation	
	__uilmacs	16 bits multiplication and accumulation with saturation mode	

For the signed data type, it uses __silmul function, __silmac function and __silmacs function.

For the unsigned data type, it uses __uilmul function, __uilmac function, __uilmacs function.

The example which performs multiplication and accumulation is shown below.

Example of multiplication and accumulation

```
#include <muldivu8.h>        <- Include muldivu8.h
long res;
void main(void)
{
    int i, j;
    int cnt;
    i = 5;
    j = 10;
    __silmul(10, 5);         <- Call multiplication function first
    for (cnt = 0; cnt < 100; cnt++)
    {
        res = __silmac(i, j);  <- Call multiplication and accumulation
function
        i += 5;
        j += 10;
    }
}
```

3.1.2 Speed up of multiplication

If it is described long type multiplication in C source, C compiler generates code for calling the operation routine (__lmulu8) which performs long type multiplication. In __lmulu8, data passing is altogether performed via stack.

```
long x, y;
long res;
void main(void)
{
    res = x * y;      <- (1)
}
```

Data passing of __lmulu8	
x	Stack
y	Stack
return value	Stack
Execution cycle for (1) : 75cycles	

If __sllmul function or __ullmul function is used instead of __lmulu8, execution speed is faster than __lmulu8. Because the 1st argument's passing is carried out via register, the 2nd argument's passing is carried out via stack, and the return value's passing is carried out via register.

```
#include <muldivu8.h>
long x, y;
long res;
void main(void)
{
    res = __sllmul(x, y); <- (2)
}
```

Data passing of __sllmul	
x	Register
y	Stack
return value	Register
Execution cycle for (2) : 62cycles	

In the C language, the multiplication result of 16 bits is 16 bits. So in order to calculate 32 bits multiplication result, it has to carry out long type multiplication.

If __silmul function or __uilmul function is used, it can get the result of 32bit for the multiplication of 16 bits. And the execution cycle is faster than __sllmul and __ullmul function.

```
#include <muldivu8.h>
int x, y;
long res;
void main(void)
{
    res = __silmul(x, y); <- (3)
}
```

Data passing of __silmul	
x	Register
y	Register
return value	Register
Execution cycle for (3) : 29cycles	

3.2 Details of C Library Functions

Below, the details of C library function of muldivu8 library are explained.

3.2.1 __silmul function

Function Prototype	signed long __silmul(signed short x, signed short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication (32 bits)
Description	__silmul function performs signed short type multiplication and returns signed long type result.

3.2.2 __uilmul function

Function Prototype	unsigned long __uilmul(unsigned short x, unsigned short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication (32 bits)
Description	__uilmul function performs unsigned short type multiplication and returns unsigned long type result.

3.2.3 __silmac function

Function Prototype	signed long __silmac(signed short x, signed short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication and accumulation (32 bits)
Description	__silmac function performs signed short type multiplication and accumulation, and returns signed long type result. The result of having added the multiplication result of x and y to the data stored in the coprocessor register is returned.

3.2.4 __uilmac function

Function Prototype	unsigned long __uilmac(unsigned short x, unsigned short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication and accumulation (32 bits)
Description	__ullmac function performs unsigned short type multiplication and accumulation, and it returns unsigned long type result. The result of having added the multiplication result of x and y to the data stored in the coprocessor register is returned.

3.2.5 __silmacs funciton

Function Prototype	signed long __silmacs(signed short x, signed short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication and accumulation (32 bits)
Description	__silmacs function performs signed type multiplication and accumulation with saturation mode, and it returns signed long type result. The result of having added the multiplication result of x and y to the data stored in the coprocessor register is returned. The operation result is saturated with the minimum (0x80000000) or the maximum (0x7fffffff).

3.2.6 __uilmacs function

Function Prototype	unsigned long __uilmacs(unsigned short x, unsigned short y)
Arguments	x : Multiplicand (16 bits) y : Multiplier (16 bits)
Return Value	Result of multiplication and accumulation (32 bits)
Description	__uilmacs function performs unsigned type multiplication and accumulation with saturation mode, and it returns unsigned long type result. The result of having added the multiplication result of x and y to the data stored in the coprocessor register is returned. The operation result is saturated with the maximum (0xffffffff).

3.2.7 __sllmul function

Function Prototype	signed long __sllmul(signed long x, signed long y)
Arguments	x : Multiplicand (32 bits) y : Multiplier (32 bits)
Return Value	Result of multiplication (32 bits)
Description	__sllmul function performs signed long type multiplication and returns signed long type result.

3.2.8 __ullmul function

Function Prototype	unsigned long __ullmul(unsigned long x, unsigned long y)
Arguments	x : Multiplicand (32 bits) y : Multiplier (32 bits)
Return Value	Result of multiplication (32 bits)
Description	__ullmul function performs signed long type multiplication and returns signed long type result.

4. How to link muldivu8 library

When you link the muldivu8 library, please specify muldiv.lib clearly at the time of a link.

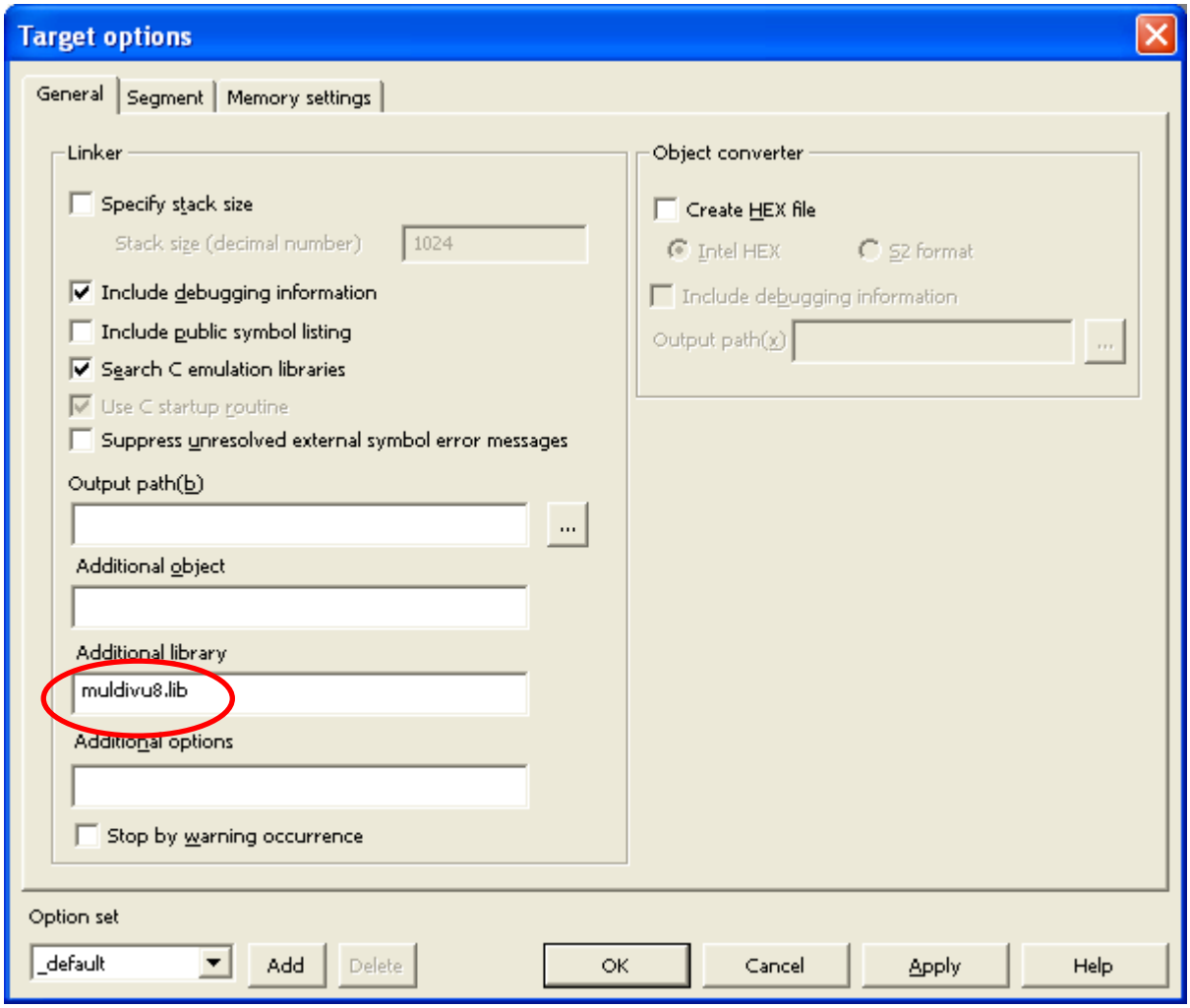
4.1 When specifying on a command line

When you invoke RLU8 linker, please specify muldivu8.lib to the library field.

```
RLU8 objfile1 objfile2, , , muldivu8.lib /CC
```

4.2 When specifying on IDEU8

When you link the muldivu8 library by IDEU8, please specify muldivu8.lib to "Additional library" field of a target option.



5. Saving/Restoring coprocessor registers in interrupt function

The CCU8 compiler does not generate the code which saves/restores coprocessor registers in the interrupt function.

In interrupt function, when multiplication or division are used, please insert the saving/restoring code as follows.

At the entrance of interrupt function, please save the contents of the coprocessor registers to the stack.

At the exit of interrupt function, please restore the contents of the coprocessor registers from the stack.

The example is shown below. In the following example, it makes two functions.

- The function (__pushCR) which saves the contents of the coprocessor registers to the stack.
- The function (__popCR) which restores the contents of the coprocessor registers from the stack.

At the entrance of interrupt function, calls __pushCR function, and at the exit of interrupt function, calls __popCR function.

```
void __pushCR(void)
{ // Save the contents of coprocessor registers to the stack.
#asm
    ADD SP, #-10
    MOV ER0,    SP
    LEA [ER0]
    MOV [EA+],  CQR0
    MOV [EA+],  CER8
#endasm
}

void __popCR(void)
{ // Restore the contents of coprocessor registers from the stack
#asm
    MOV ER0,    SP
    LEA [ER0]
    MOV R0, #0
    MOV CR8,    R0
    MOV CQR0,   [EA+]
    MOV CER8,   [EA]
    ADD SP, #10
#endasm
}

int i, j;
static void __swi00(void);
#pragma swi __swi00 0x80
```

Save the CR0-CR9 to the stack

1. Set up the mode register (CR8) to not start operation.
2. Restore the CR0-CR9 from the stack.

```
static void __swi00(void)
{
    __pushCR();     ← Saving the coprocessor registers
    i /= j;         ← int type division
    __popCR();      ← Restoring the coprocessor registers
}
```

Appendix A. Execution cycle and stack size list

Execution cycle and stack size of muldivu8 library routines are shown below.

Table A-1 Execution cycle and stack size

Operation Routine	Small Model			Large Model		
	Cycle		Stack size	Cycle		Stack size
	U8	U16		U8	U16	
__imulu8	16	16	0	16	16	0
__idivu8	28 13 (*1)	27 11 (*1)	0	28 13 (*1)	27 11 (*1)	0
__imodu8	32 16 (*1)	31 15 (*1)	0	32 16 (*1)	31 15 (*1)	0
__uidivu8	28 9 (*1)	27 7 (*1)	0	28 9 (*1)	27 7 (*1)	0
__uimodu8	32 8 (*1)	31 7 (*1)	0	32 8 (*1)	31 7 (*1)	0
__lmulu8	62	42	8	62	42	8
__ldivu8	128 (*1) 125 (*2) 187-253 (*2)	93 (*1) 92 (*2) 96-114 (*3)	24	134 (*1) 131 (*2) 193-259 (*3)	97 (*1) 96 (*2) 100-118 (*3)	28
__lmodu8	127 (*1) 124 (*2) 187-262 (*3)	92 (*1) 91 (*2) 96-120 (*3)	24	133 (*1) 130 (*2) 193-268 (*3)	96 (*1) 95 (*2) 100-124 (*3)	28
__uldivu8	86 (*1) 83 (*2) 163-211 (*3)	57 (*1) 56 (*2) 76 (*4)	20	89 (*1) 86 (*2) 166-214 (*3)	59 (*1) 58 (*2) 78 (*4)	22
__ulmodu8	86 (*1) 83 (*2) 163-211 (*3)	57 (*1) 56 (*2) 76 (*4)	20	89 (*1) 96 (*2) 166-214 (*3)	59 (*1) 58 (*2) 78 (*4)	22
__silmul	18	18	0	18	18	0
__uilmul	16	16	0	16	16	0
__sllmul	43	35	4	43	35	4
__ullmul	43	35	4	43	35	4
__silmac	18	18	0	18	18	0
__uilmac	16	16	0	16	16	0
__silmacs	18	18	0	18	18	0
__uilmacs	18	18	0	18	18	0

*1：Cycles in case a denominator is 0.

*2：Cycles in case the high word (16 bits) of denominator is 0.

*3：Cycles in case the high word (16 bits) of denominator is not 0. (The cycle depends on the value of the numerator and denominator)

*4：Cycles in case the high word (16 bits) of denominator is not 0.

Appendix B. Code size list

Code size of muldivu8 library routines are shown below.

Table B-1 Code size

Operation Routine	Code size (bytes)	Remarks
__imulu8	26	When the left 3 routines are all used, the maximum code size becomes 34 bytes.
__uilmul	26	
__silmul	34	
__idivu8	54	When the left 2 routines are all used, the maximum code size becomes 86 bytes.
__imodu8	86	
__uidivu8	46	When the left 2 routines are all used, the maximum code size becomes 62 bytes.
__uimodu8	62	
__lmulu8	70	—
__ldivu8	390	When the left 4 routines are all used, the maximum code size becomes 472 bytes.
__lmodu8	390	
__uldivu8	296	
__ulmodu8	296	
__sllmul	62	When the left 2 routines are all used, the maximum code size becomes 62 bytes.
__ullmul	62	
__silmac	34	When the left 4 routines are all used, the maximum code size becomes 50 bytes.
__uilmac	26	
__silmacs	34	
__uilmacs	34	

MULDIVU8LIB
Multiplication and Division Library for U8/U16 Accelerator

User's Manual
SQ003125E001

Fourth Edition
ISSUE DATE Jun. 28, 2012
