LAPIS SEMICONDUCTOR

FJXTMULDIVLIB_UM-04

MULDIVU8LIB
アクセラレータ対応 乗除算ライブラリ
ユーザーズマニュアル

発行日 2012 年 6 月 28 日

ご注意

本資料の一部または全部をラピスセミコンダクタの許可なく、転載・複写することを堅くお断りします。

本資料の記載内容は改良などのため予告なく変更することがあります。

本資料に記載されている内容は製品のご紹介資料です。ご使用にあたりましては、別途仕様書を必ずご請求のうえ、ご確認ください。

本資料に記載されております応用回路例やその定数などの情報につきましては、本製品の標準的な動作や使い方を説明するものです。したがいまして、量産設計をされる場合には、外部諸条件を考慮していただきますようお願いいたします。

本資料に記載されております情報は、正確を期すため慎重に作成したものですが、万が一、当該情報の誤り・誤植に起因する損害がお客様に生じた場合においても、ラピスセミコンダクタはその責任を負うものではありません。

本資料に記載されております技術情報は、製品の代表的動作および応用回路例などを示したものであり、ラピスセミコンダクタまたは他社の知的財産権その他のあらゆる権利について明示的にも黙示的にも、その実施または利用を許諾するものではありません。上記技術情報の使用に起因して紛争が発生した場合、ラピスセミコンダクタはその責任を負うものではありません。

本資料に掲載されております製品は、一般的な電子機器（AV 機器、OA 機器、通信機器、家電製品、アミューズメント機器など）への使用を意図しています。

本資料に掲載されております製品は、「耐放射線設計」はなされておりません。

ラピスセミコンダクタは常に品質・信頼性の向上に取り組んでおりますが、種々の要因で故障することもあり得ます。

ラピスセミコンダクタ製品が故障した際、その影響により人身事故、火災損害等が起こらないようご使用機器でのディレーティング、冗長設計、延焼防止、フェイルセーフ等の安全確保をお願いします。定格を超えたご使用や使用上の注意書が守られていない場合、いかなる責任もラピスセミコンダクタは負うものではありません。

極めて高度な信頼性が要求され、その製品の故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのある機器・装置・システム（医療機器、輸送機器、航空宇宙機、原子力制御、燃料制御、各種安全装置など）へのご使用を意図して設計・製造されたものではありません。上記特定用途に使用された場合、いかなる責任もラピスセミコンダクタは負うものではありません。上記特定用途への使用を検討される際は、事前にローム営業窓口までご相談願います。

本資料に記載されております製品および技術のうち「外国為替及び外国貿易法」に該当する製品または技術を輸出する場合、または国外に提供する場合には、同法に基づく許可が必要です。

ラピスセミコンダクタ株式会社

〒193-8550 東京都八王子市東浅川町 550-1
http://www.lapis-semi.com/jp/

目次

1. 概要

アクセラレータ対応乗除算ライブラリは、U8 または U16 をコアとするマイクロコントローラに搭載された乗除算アクセラレータを使用して演算を行うルーチン群をまとめたものです。（以降、本ライブラリを乗除算ライブラリと呼びます。）

C 言語にて 16 ビットまたは 32 ビットの乗除算を記述したプログラムに対し、リンク時に乗除算ライブラリを指定してリンクすることで、C ソースを変更することなく、乗除算アクセラレータを利用したコードに置き換えることができます。

2. 乗除算ライブラリの構成

乗除算ライブラリは、ヘッダファイルとライブラリファイルで構成されます。

表 2-1 乗除算ライブラリの構成

ファイル名	内容
muldivu8.h	C ライブラリ関数用ヘッダファイル
muldivu8.lib	乗除算ライブラリ本体

乗除算ライブラリには、C 言語で記述された乗除算の演算に対して呼び出される演算ルーチンと、乗除算アクセラレータの機能をサポートするための C ライブラリ関数が含まれています。

2.1 演算ルーチン一覧

乗除算ライブラリに含まれる演算ルーチンの一覧を以下に示します。

表 2-2 演算ルーチン一覧

演算ルーチン	機能	演算結果ビット長	データ受け渡し	
			引数	戻り値
__imulu8	signed int 型の乗算 unsigned int 型の乗算	16 ビット	レジスタ	レジスタ
__idivu8	signed int 型の除算	16 ビット	レジスタ	レジスタ
__imodu8	signed int 型の剰余算	16 ビット	レジスタ	レジスタ
__uidivu8	unsigned int 型の除算	16 ビット	レジスタ	レジスタ
__uimodu8	unsigned int 型の剰余算	16 ビット	レジスタ	レジスタ
__lmulu8	signed long 型の乗算 unsigned long 型の乗算	32 ビット	スタック	スタック
__ldivu8	signed long 型の除算	32 ビット	スタック	スタック
__lmodu8	signed long 型の剰余算	32 ビット	スタック	スタック
__uldivu8	unsigned long 型の除算	32 ビット	スタック	スタック
__ulmodu8	unsigned long 型の剰余算	32 ビット	レジスタ	スタック

C 言語にて 16 ビットまたは 32 ビットの乗除算（*, /, %）を記述した場合に、コンパイラがデータの型および演算の種類に応じて、必要な演算ルーチンを呼び出すコードを生成します。

演算のオペランドの型が違う場合、演算はオペランドの型が大きい方にあわせられます。例えば、signed int * signed long の場合、signed long * signed long の演算として扱われます。

2.2 C ライブラリ関数一覧

乗除算ライブラリに含まれる C ライブラリ関数の一覧を以下に示します。

表 2-3 C ライブラリ関数一覧

C ライブラリ関数	機能	演算結果ビット長	データ受け渡し	
			引数	戻り値
__silmul	16 ビット乗算（符号付き）	32 ビット	レジスタ	レジスタ
__uilmul	16 ビット乗算（符号なし）	32 ビット	レジスタ	レジスタ
__silmac	16 ビット積和（符号付き）	32 ビット	レジスタ	レジスタ
__uilmac	16 ビット積和（符号なし）	32 ビット	レジスタ	レジスタ
__silmacs	16 ビット飽和型積和（符号付き）	32 ビット	レジスタ	レジスタ
__uilmacs	16 ビット飽和型積和（符号なし）	32 ビット	レジスタ	レジスタ
__sllmul	32 ビット乗算（符号付き）	32 ビット	レジスタ&スタック	レジスタ
__ullmul	32 ビット乗算（符号なし）	32 ビット	レジスタ&スタック	レジスタ

上記の C ライブラリ関数の使い方については、「3.1 C ライブラリ関数の使い方」を参照してください。

3. C ライブラリ関数

3.1 C ライブラリ関数の使い方

ここでは、「2.2 C ライブラリ関数一覧」で示した関数の使い方について説明します。

なお、C ライブラリ関数を使用する場合は、ヘッダファイル muldivu8.h をインクルードしてください。

3.1.1 積和演算を行う

積和演算を行う場合には、以下に示す関数を使用します。

符号	関数名	機能	備考
符号付き	__silmul	16 ビット乗算	積和演算を行うときに最初に呼び出す
	__silmac	16 ビット積和	
	__silmacs	16 ビット飽和型積和	
符号なし	__uilmul	16 ビット乗算	積和演算を行うときに最初に呼び出す
	__uilmac	16 ビット積和	
	__uilmacs	16 ビット飽和型積和	

符号付きの場合には__silmul 関数、__silmac 関数、__silmacs 関数を、符号なしの場合には、__uilmul 関数、__uilmac 関数、__uilmacs 関数を組み合わせて使用します。

積和演算を行う例を以下に示します。

積和演算の使用例

```
#include <muldivu8.h>          ← ヘッダファイル muldivu8.h をインクルード
long res;
void main(void)
{
    int i, j;
    int cnt;
    i = 5;
    j = 10;
    __silmul(10, 5);           ← 最初の 1 回だけ乗算用の関数をコール
    for (cnt = 0; cnt < 100; cnt++)
    {
        res = __silmac(i, j);  ← 以降は積和演算用の関数をコール
        i += 5;
        j += 10;
    }
}
```

3.1.2 乗算の実行速度を速くする

C の記述にて long 型同士の乗算を記述すると、C コンパイラは long 型同士の乗算を行う演算ルーチン（__lmulu8）を呼び出します。__lmulu8 では、データの受け渡しはすべてスタックを介して行われます。

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

__lmulu8 のデータ受け渡し	
x	スタック
y	スタック
戻り値	スタック
左記(1)に対する実行サイクル数：75 サイクル	

__lmulu8 の代わりに、__sllmul 関数または__ullmul 関数を使用すると、データ受け渡しが、第 1 引数はレジスタ、第 2 引数はスタック、戻り値はレジスタを介して行われるため、__lmulu8 よりも実行速度を速くすることができます。

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

__sllmul のデータ受け渡し	
x	レジスタ
y	スタック
戻り値	レジスタ
左記(2)に対する実行サイクル数：62 サイクル	

C 言語では、16 ビット同士の乗算結果は 16 ビットであり、32 ビットの乗算結果を得たい場合には、long 型の乗算を行う必要があります。

__silmul 関数または__uilmul 関数を使用すると、16 ビット同士の乗算を行って、32 ビットの結果を得ることができ、実行速度を速くすることができます。

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

__silmul のデータ受け渡し	
x	レジスタ
y	レジスタ
戻り値	レジスタ
左記(3)に対する実行サイクル数：29 サイクル	

3.2 C ライブラリ関数説明

以下に、乗除算ライブラリの C ライブラリ関数の機能について説明します。

3.2.1 __silmul 関数

関数プロトタイプ	signed long __silmul(signed short x, signed short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	乗算結果（32 ビット）
機能	符号付き 16 ビットの乗算(x×y)を行い、32 ビットの結果を返します。

3.2.2 __uilmul 関数

関数プロトタイプ	unsigned long __uilmul(unsigned short x, unsigned short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	乗算結果（32 ビット）
機能	符号なし 16 ビットの乗算(x×y)を行い、32 ビットの結果を返します。

3.2.3 __silmac 関数

関数プロトタイプ	signed long __silmac(signed short x, signed short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	積和結果（32 ビット）
機能	符号付き 16 ビットの積和演算を行います。 コプロセッサレジスタに格納されているデータ（32 ビット）に x×y の乗算結果（32 ビット）を加算した結果（32 ビット）を返します。

3.2.4 __uilmac 関数

関数プロトタイプ	unsigned long __uilmac(unsigned short x, unsigned short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	積和結果（32 ビット）
機能	符号なし 16 ビットの積和演算を行います。 コプロセッサレジスタに格納されているデータ（32 ビット）に x×y の乗算結果（32 ビット）を加算した結果（32 ビット）を返します。

3.2.5 __silmacs 関数

関数プロトタイプ	signed long __silmacs(signed short x, signed short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	積和結果（32 ビット）
機能	符号付き 16 ビットの飽和型積和演算を行います。 コプロセッサレジスタに格納されているデータ（32 ビット）に x×y の乗算結果（32 ビット）を加算した結果（32 ビット）を返します。 この関数を実行すると、演算結果は最小値（0x80000000）、または最大値（0x7fffffff）で飽和します。

3.2.6 __uilmacs 関数

関数プロトタイプ	unsigned long __uilmacs(unsigned short x, unsigned short y)
引数	x：被乗数（16 ビット） y：乗数（16 ビット）
戻り値	積和結果（32 ビット）
機能	符号なし 16 ビットの飽和型積和演算を行います。 コプロセッサレジスタに格納されているデータ（32 ビット）に x×y の乗算結果（32 ビット）を加算した結果（32 ビット）を返します。 この関数を実行すると、演算結果は最大値（0xffffffff）で飽和します。

3.2.7 __sllmul 関数

関数プロトタイプ	signed long __sllmul(signed long x, signed long y)
引数	x：被乗数（32 ビット） y：乗数（32 ビット）
戻り値	乗算結果（32 ビット）
機能	符号付き 32 ビットの乗算(x×y)を行い、32 ビットの結果を返します。

3.2.8 __ullmul 関数

関数プロトタイプ	unsigned long __ullmul(unsigned long x, unsigned long y)
引数	x：被乗数（32 ビット） y：乗数（32 ビット）
戻り値	乗算結果（32 ビット）
機能	符号なし 32 ビットの乗算(x×y)を行い、32 ビットの結果を返します。

4. リンク方法

乗除算ライブラリをリンクする場合は、明示的に muldivu8.lib をリンク時に指定してください。

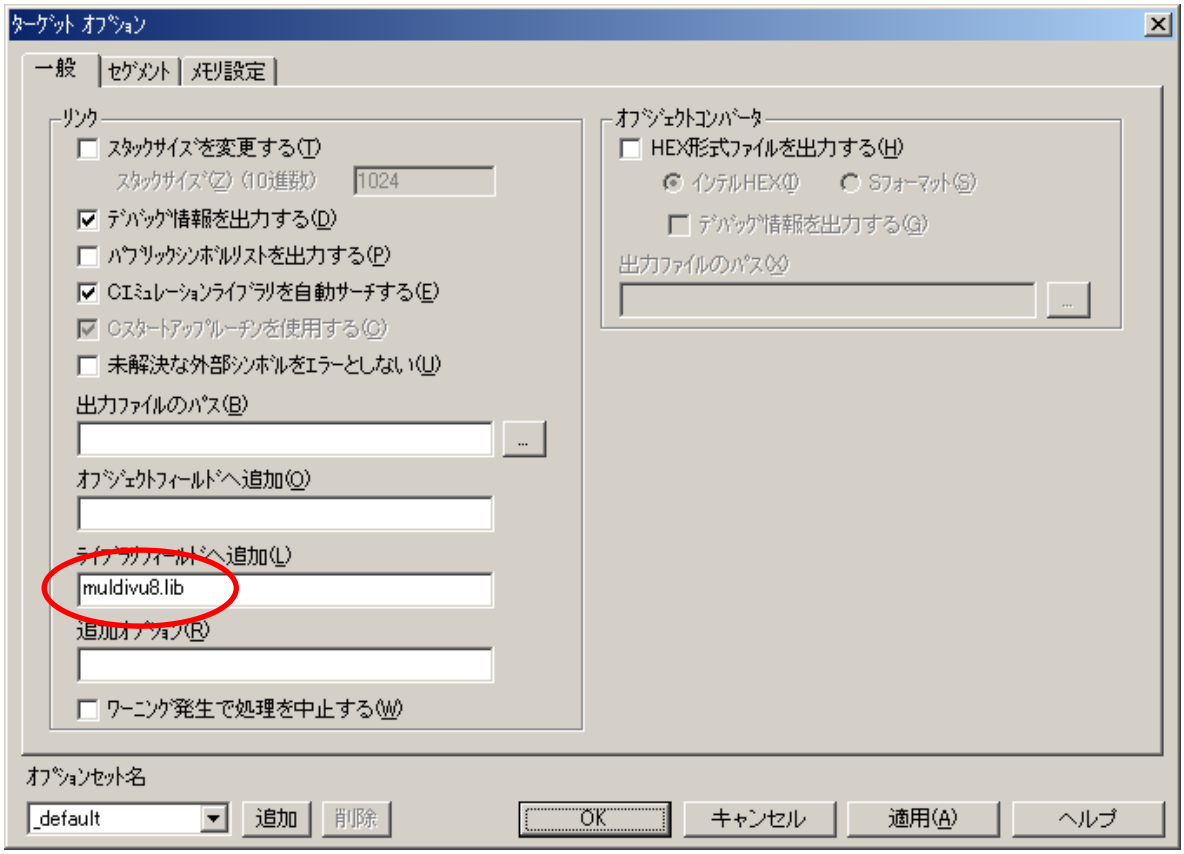
4.1 コマンドライン上で指定する場合

RLU8 リンカを起動するときに、ライブラリフィールドへ muldivu8.lib を指定してください。

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

4.2 IDEU8 で指定する場合

IDEU8 で乗除算ライブラリをリンクする場合は、ターゲットオプションの[ライブラリフィールドへ追加]フィールドに muldivu8.lib を指定してください。



5. 割り込み処理でのコプロセッサレジスタの退避・復帰

コンパイラは、割り込み処理に対してコプロセッサレジスタを退避・復帰するコードを生成しません。

割り込み処理内で、乗除算ライブラリを使用する記述をしている場合には、割り込み処理の入り口にてコプロセッサレジスタの内容をスタックに保存し、割り込み処理の出口にてコプロセッサレジスタの内容を復帰してください。

以下に例を示します。この例では、コプロセッサレジスタの内容をスタックに保存する関数（__pushCR）と、コプロセッサレジスタの内容を復帰する関数（__popCR）を作成し、割り込みの入り口で__pushCR 関数を、割り込みの出口で__popCR 関数を呼び出しています。

```
void __pushCR(void)
{ // コプロセッサレジスタの内容をスタックに保存
#asm
    ADD SP, #-10
    MOV ER0,    SP
    LEA [ER0]
    MOV [EA+],  CQR0
    MOV [EA+],  CER8
#endasm
}

void __popCR(void)
{ // コプロセッサレジスタの内容をスタックから復帰
#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

static void __swi00(void)
{
    __pushCR();     ← コプロセッサレジスタの退避
    i /= j;         ← 除算処理
    __popCR();      ← コプロセッサレジスタの復帰
}
```

CR0～CR9 をスタックに退避

演算を行わないように演算モードを設定してから、CR0～CR9 をスタックから復帰

付録 1. サイクル数およびスタック消費量一覧

乗除算ライブラリの演算ルーチンおよび C ライブラリ関数のサイクル数およびスタック消費量を以下に示します。サイクル数は、U8 と U16 でのサイクル数を示しています。

表 付-1 サイクル数およびスタック消費量

演算ルーチン	スモールモデル			ラージモデル		
	サイクル数		スタック	サイクル数		スタック
	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：除数が 0 の場合のサイクル数

※2：除数の上位 16 ビットが 0 の場合のサイクル数

※3：除数の上位 16 ビットが 0 以外の場合のサイクル数（被除数と除数の値によりサイクル数が変わる）

※4：除数の上位 16 ビットが 0 以外の場合のサイクル数

付録 2. コードサイズ一覧

乗除算ライブラリの演算ルーチンおよびCライブラリ関数のコードサイズを以下に示します。

表 付-2 コードサイズ

ルーチン名・関数名	コードサイズ	備考
__imulu8	26	左記のルーチンをすべて使用する場合、最大 34 バイトとなります。
__uilmul	26	
__silmul	34	
__idivu8	54	左記のルーチンをすべて使用する場合、最大 86 バイトとなります。
__imodu8	86	
__uidivu8	46	左記のルーチンをすべて使用する場合、最大 62 バイトとなります。
__uimodu8	62	
__lmulu8	70	－
__ldivu8	390	左記のルーチンをすべて使用する場合、最大 472 バイトとなります。
__lmodu8	390	
__uldivu8	296	
__ulmodu8	296	
__sllmul	62	左記のルーチンをすべて使用する場合、最大 62 バイトとなります。
__ullmul	62	
__silmac	34	左記のルーチンをすべて使用する場合、最大 50 バイトとなります。
__uilmac	26	
__silmacs	34	
__uilmacs	34	

MULDIVU8LIB
アクセラレータ対応乗除算ライブラリ
ユーザーズマニュアル
SQ003125E001

第 4 版
2012 年 6 月 28 日　　発行
