参 考 資 料

JAJA034 was
SPRAA80– 02月 2005年

TEXAS INSTRUMENTS
www.tij.co.jp

C64xコア用C言語上での64bit整数演算の実現方法

服部基保　　DSP製品部アプリケーショングループ

アブストラクト

近年、あらゆる分野において、演算精度が向上しています。そのため、弊社C6000 DSP上でも、64bitでの整数演算の必要な場合が徐々に増えてきています。このドキュメントでは、C言語での64bit整数演算の実装方法について、long long型で記述する方法やdouble型を64bit整数型とみなして記載する方法など計３通り、説明します。また、実現するためのC64x用のプログラムの説明もおこないます。

目次

表

1　64bit 整数演算の実現方法

C6000のCコンパイラは、表 1のような型をサポートしています。これらの型を使って、C言語上での64bit整数演算を実現します。その方法は、以下の二つあります。

方法１：変数をlong long型で記述する方法
方法２：long long型でループ部分を最適に実行する方法
方法３：double型を64bit整数型とみなして記載する方法

次章以降で、上記の三つの方法についての実装方法、注意事項などを説明します。

表１　C6000 Cコンパイラの型

型	ビット長
Char	8bit 整数
Short	16bit 整数
Int	32bit 整数
Long	40bit 整数
long long	64bit 整数　（CCS3.0以降）
Float	32bit 浮動小数点数
double	64bit 倍精度浮動小数点数
long double	64bit 倍精度浮動小数点数　（CCS3.0以降）

2 変数を long long 型で記述する方法

この章では、64bit整数演算の実現方法の一つである「変数をlong long型で記述する方法について説明します。

2.1 実現方法

この方法は、以下のように、変数をlong long型で宣言して、記述するだけです。

```
long long x[1024],y[1024];
long long z=0;
int i;
for(i=0;i<1024;i++){
    z+=x[i]*y[i];
}
```

2.2 この方法の制限事項

この方法は、ANSI-C準拠で簡単に記載できます。しかし、これを使用するための制限事項が二つあります

CCS 2.x / CCS1.xのCコンパイラでは、long long型をサポートしておりません。CCS 3.0以上（Cコンパイラのバージョン5.00以上）をご使用ください。もし、CCS 2.x / CCS1.xを使用する場合は、方法３をご使用ください。

CCS 3.0 / 3.1（Cコンパイラ・バージョン5.00 / 5.10）では、long long型の乗算、シフトなどは、ライブラリの関数コールになります。
これらの演算をfor文内などループで使用すると、ソフトウェア・パイプライン化されないので、C6000 コアの性能が最大限に引き出されていないコードになることがあります。（ループ以外の部分では、最適なコードになります。）

2.3 サンプル・プログラム例

以下に三つのサンプル・プログラムを示します。
例１：32bitデータの積和演算（結果64bit）
例２：32bitデータ乗算の上位のみ使用する積和演算
例３：32bitデータの2乗の和

2.3.1 32bit データの積和演算（結果 64bit）

二つの32bitデータ列の積をおこない、64bitとして足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
long long rst_LL;

rst_LL= 0;
for(i=0;i<1024;i++){
   rst_LL = rst_LL
          + (   ((long long)src1_int[i])
              * ((long long)src2_int[i]) );
}
```

2.3.2 32bit データ乗算の上位のみ使用する積和演算

二つの32bitデータ列の積をおこない、その結果の上位32bit分を足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
int rst_int;

rst_int=0;
for(i=0;i<1024;i++){
   rst_int = rst_int
           + ( (int)
              (( ((long long)src1_int[i])
                * ((long long)src2_int[i]) )>>31)
             );
}
```

2.3.3 32bit データの 2 乗の和

32bitデータの2乗を２つ求め、それを足して64bitとして書き込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
long lond dst2_LL[1024];

for(i=0;i<1024;i++){
   dst2_LL[i]
          =  ((long long)src1_int[i])*((long long)src1_int[i])
           + ((long long)src2_int[i])*((long long)src2_int[i]);
}
```

3 long long 型でループ部分を最適に実行する方法

この章では、64bit整数演算の実現方法の一つである「long long型でループ部分を最適に実行する方法」について説明します。

3.1 実現方法

この方法は、前章で説明した、変数をlong long型で宣言して記述する方法に、ループ部分を最適に実行する関数を併用して実装する方法です。

前章で説明したとおり、CCS 3.0 / 3.1（Cコンパイラ・バージョン5.00 / 5.10）では、long long型の乗算、シフトなどは、ライブラリの関数コールになります。
これらの演算をfor文内などループで使用すると、ソフトウェア・パイプライン化されないので、C6000 コアの性能が最大限に引き出されていないコードになることがあります。
そのため、この章では、ソフトウェア・パイプライン化されない関数を別関数として用意し、その関数をinlineすることにより、ループ部分を高速に実行する方法です。

この方法は、以下のように記述します。後の節で説明する関数を呼び出して記述してください。

```
long long x[1024],y[1024];
long long z=0;
int i;
for(i=0;i<1024;i++){
    z+= mpyLongL64( x[i], y[i] );
}
```

3.2 この方法の制限事項

この方法を使用するための制限事項があります。

CCS 2.x / CCS1.xのCコンパイラでは、long long型をサポートしておりません。前章と同様、CCS 3.0以上（Cコンパイラのバージョン5.00以上）をご使用ください。もし、CCS 2.x / CCS1.xを使用する場合は、方法３をご使用ください。

除算関数（divL64 / divuL64）は、その関数内でループを用いているため、inline展開する必要がありません。つまり、除算関数とともに繰り返す場合は、そのループの外では、ソフトウェア・パイプライン化されません。アルゴリズムとして可能であれば、除算関数部分とそうでない部分に分割してそれぞれ繰り返すことにより、全体が高速化できる場合があります。
除算以外の関数群は、ループの部分で使用する時に最適に実行できるように、必ずinline展開して、使用してください。
ループ以外の部分では、関数群を使用しても、高速化されない場合があります。

3.3 関数の使用方法の説明

この節では、long long型での64bit関数群について、Inlineする関数とそのまま使用する関数（divL64 / divuL64）とにわけて説明します。

3.3.1 64bit 整数 inline 関数群

- **mathLongL64.h**
  この節で説明するinline関数群を使用するときは、このヘッダ・ファイルをインクルードしてください。

- **inline long long absL64(long long src);**
  入力：long long（64bit符号付整数）
  出力：long long（64bit符号付整数）
  関数の内容：**入力データの絶対値を返す**

- **inline long long mpy32L64(int src1, int src2);**
  入力src1/src2：int（32bit符号付整数）
  出力：long long（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果を返す

- **inline unsigned long long mpyu32L64(unsigned int src1, unsigned int src2);**
  入力src1/src2：unsigned int（32bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：src1とsrc2の乗算結果を返す

- **inline long long mpyL64(long long src1, long long src2);**
  入力src1/src2：long long（64bit符号付整数）
  出力：long long（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果を返す

- **inline unsigned long long mpyuL64(unsigned long** long src1, unsigned long long src2);
  入力src1/src2：unsigned long long（64bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：src1とsrc2の乗算結果を返す

- **inline long long mpy32LL64(long long src1, int src2);**
  入力src1：long long（64bit符号付整数）
  入力src2：int（32bit符号付整数）

出力：long long（64bit符号付整数）
関数の内容：src1とsrc2の乗算結果を返す

- **inline unsigned long long mpyu32LL64(unsigned long long src1, unsigned int src2);**
  入力src1：unsigned long long（64bit符号なし整数）
  入力src2：unsigned int（32bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：src1とsrc2の乗算結果を返す
- **inline long long negL64(long long src);**
  入力：long long（64bit符号付整数）
  出力：long long（64bit符号付整数）
  関数の内容：入力データの符号を反転した値を返す
- **inline long long shlL64(long long src , unsigned int shift);**
  入力src：long long（64bit符号付整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：long long（64bit符号付整数）
  関数の内容：srcの値をshift分左シフトした値を返す
- **inline long long shrL64(long long src , unsigned int shift);**
  入力src：long long（64bit符号付整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：long long（64bit符号付整数）
  関数の内容：srcの値をshift分右シフトした値を返す
- **inline unsigned long long shruL64(unsigned long long src , unsigned int shift);**
  入力src：unsigned long long（64bit符号なし整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：long long（64bit符号なし整数）
  関数の内容：srcの値をshift分左シフトした値を返す
- **inline unsigned long long shlruL64(unsigned long long src , int shift);**
  入力src：unsigned long long（64bit符号なし整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：src値の下位（64-shift）分のみ切り出す
- **inline unsigned long long subcL64(unsigned long long src1, unsigned long long src2);**
  入力src1/src2：unsigned long long（64bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：SUBC命令の64bit版、割り算の時に使用

3.3.2 64bit 除算関数群

- **long long divL64(long long src1, long long src2);**
  入力src1/src2：long long（64bit符号付整数）
  出力：long long（64bit符号付整数）
  関数の内容：src1/src2の結果を返す
  条件：０で割ったときは、不定値を返す
- **unsigned long long divuL64(unsigned long long src1, unsigned long long src2);**
  入力src1/src2：unsigned long long（64bit符号なし整数）
  出力：unsigned long long（64bit符号なし整数）
  関数の内容：src1/src2の結果を返す
  条件：０で割ったときは、不定値を返す

3.4 サンプル・プログラム例

以下に三つのサンプル・プログラムを示します。
例１：32bitデータの積和演算（結果64bit）
例２：32bitデータ乗算の上位のみ使用する積和演算
例３：32bitデータの2乗の和

3.4.1 32bit データの積和演算（結果 64bit）

二つの32bitデータ列の積をおこない、64bitとして足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
long long rst_LL;

rst_LL=0;
for(i=0;i<1024;i++){
   rst_LL = rst_LL
            + mpy32LongL64(src1_int[i],src2_int[i]);
}
```

3.4.2 32bit データ乗算の上位のみ使用する積和演算

二つの32bitデータ列の積をおこない、その結果の上位32bit分を足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
int rst_int;

rst_int=0;
for(i=0;i<1024;i++){
   rst_int = rst_int
           + shrLongL64(
              mpy32LongL64(src1_int[i],src2_int[i]),
              31
            );
}
```

3.4.3 32bit データの 2 乗の和

32bitデータの2乗を２つ求め、それを足して64bitとして書き込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
long lond dst2_LL[1024];

for(i=0;i<1024;i++){
   dst2_LL[i] =  mpy32LongL64(src1_int[i],src1_int[i])
           + mpy32LongL64(src2_int[i],src2_int[i]);
}
```

4 double 型を 64bit 整数型とみなして記載する方法

この章では、64bit整数演算の実現方法の一つである「double型を64bit整数型とみなして記載する方法」について説明します。

4.1 実現方法

この方法は、64bit倍精度浮動小数点型であるdouble型を64bit整数型としてみなし、ユーザーが考えて記述する方法です。そのため、以下のようにすべての演算を関数で実現します。後の節で説明する関数を使用して記述してください。

```
double x[1024],y[1024];
double z=0;
int i;
for(i=0;i<1024;i++){
    z= macL64( x[i], y[i], z );
}
```

4.2 この方法の注意事項

この方法は、CCSのすべてのバージョンで使用できます。しかし、これを使用するための注意事項があります

すべての演算は、64bit整数型の関数群にて記述しないといけません。算術論理演算を、関数群を使用せず、C言語そのまま記述すると、double型で定義していますので、64bit倍精度演算となってしまいます。ご注意ください。

除算関数は、その関数内でループを用いているため、inline展開する必要がありません。つまり、除算関数とともに繰り返す場合は、そのループの外では、ソフトウェア・パイプライン化されません。アルゴリズムとして可能であれば、乗算関数部分とそうでない部分に分割してそれぞれ繰り返すことにより、高速化できる場合があります。
除算以外の関数群は、ループの部分で使用する時に最適に実行できるように、必ずinline展開してください。

4.3 関数の使用方法の説明

この節では、double型での64bit関数群について、Inlineする関数と、そのまま使用する関数とにわけて説明します。

4.3.1 64bit 整数 inline 関数群

- **mathL64.h**
  この節で説明するinline関数群を使用するときは、このヘッダ・ファイルをインクルードしてください。

- **inline double absL64(double src);**
  入力：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：入力データの絶対値を返す

- **inline double addL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1とsrc2の加算結果を返す

- **inline double adduL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src1とsrc2の加算結果を返す

- **inline int cmpeqL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：int（0 or 1）
  関数の内容：src1とsrc2が同じときに 1 を返す。違う場合は、 0 を返す。

- **inline int cmpgtL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：int（0 or 1）
  関数の内容：src1>src2の場合 1 を返す。違う場合は、 0 を返す。

- **inline int cmpgtuL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：int（0 or 1）
  関数の内容：src1>src2の場合 1 を返す。違う場合は、 0 を返す。

- **inline double macL3264(int src1, int src2, double acc);**
  入力src1/src2：int（32bit符号付整数）
  入力acc：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果にaccを足した値を返す

- **inline double mpy32L64(int src1, int src2);**
  入力src1/src2：int（32bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果を返す

- **inline double mpyu32L64(unsigned int src1, unsigned int src2);**

入力src1/src2：unsigned int（32bit符号なし整数）
出力：double（64bit符号なし整数）
関数の内容：src1とsrc2の乗算結果を返す

- **inline double mpyL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果を返す
- **inline double mpyuL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src1とsrc2の乗算結果を返す
- **inline double mpy32LL64(double src1, int src2);**
  入力src1：double（64bit符号付整数）
  入力src2：int（32bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1とsrc2の乗算結果を返す
- **inline double mpyu32LL64(double src1, unsigned int src2);**
  入力src1：double（64bit符号なし整数）
  入力src2：unsigned int（32bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src1とsrc2の乗算結果を返す
- **inline double negL64(double src);**
  入力：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：入力データの符号を反転した値を返す
- **inline double shlL64(double src , unsigned int shift);**
  入力src：double（64bit符号付整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：double（64bit符号付整数）
  関数の内容：srcの値をshift分左シフトした値を返す
- **inline double shrL64(double src , unsigned int shift);**
  入力src：double（64bit符号付整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：double（64bit符号付整数）
  関数の内容：srcの値をshift分右シフトした値を返す
- **inline double shruL64(double src , unsigned int shift);**
  入力src：double（64bit符号なし整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：srcの値をshift分左シフトした値を返す
- **inline double shlruL64(double src , int shift);**
  入力src：double（64bit符号なし整数）
  入力shift：unsigned int（32bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src値の下位（64-shift）分のみ切り出す
- **inline int signL64(double src);**
  入力src：double（64bit符号付整数）
  出力：int（0 or 1）
  関数の内容：srcが正であれば、０を返す。負であれば、１を返す。
- **inline double subL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1-src2の結果を返す
- **inline double subuL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src1-src2の結果を返す
- **inline double subcL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：SUBC命令の64bit版、割り算の時に使用

4.3.2 除算および 64bit 整数との変換ルーチン

- **double uL64toDbl(double src);**
  入力：double（64bit符号なし整数）
  出力：double（64bit倍精度浮動小数点数）
  関数の内容：倍精度浮動小数点への変換
- **double DbltouL64(double src);**
  入力：double（64bit倍精度浮動小数点数）
  出力：double（64bit符号なし整数）
  関数の内容：倍精度浮動小数点からの変換
- **double L64toDbl(double src);**
  入力：double（64bit符号付整数）
  出力：double（64bit倍精度浮動小数点数）
  関数の内容：倍精度浮動小数点への変換
- **double DbltoL64(double src);**
  入力：double（64bit倍精度浮動小数点数）
  出力：double（64bit符号付整数）
  関数の内容：倍精度浮動小数点からの変換
- **double divL64(double src1, double src2);**
  入力src1/src2：double（64bit符号付整数）
  出力：double（64bit符号付整数）
  関数の内容：src1/src2の結果を返す
  条件：０で割ったときは、不定値を返す

- **double divuL64(double src1, double src2);**
  入力src1/src2：double（64bit符号なし整数）
  出力：double（64bit符号なし整数）
  関数の内容：src1/src2の結果を返す
  条件：０で割ったときは、不定値を返す

4.4 サンプル・プログラム例

以下に三つのサンプル・プログラムを示します。
例１：32bitデータの積和演算（結果64bit）
例２：32bitデータ乗算の上位のみ使用する積和演算
例３：32bitデータの2乗の和

4.4.1 32bit データの積和演算（結果 64bit）

二つの32bitデータ列の積をおこない、64bitとして足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
double rst_L64;

rst_L64=_itod(0,0);
for(i=0;i<1024;i++){
   rst_L64 = mac32L64(src1_int[i],src2_int[i],rst_L64);
}
```

4.4.2 32bit データ乗算の上位のみ使用する積和演算

二つの32bitデータ列の積をおこない、その結果の上位32bit分を足し込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
double rst_L64;

rst_L64=_itod(0,0);
for(i=0;i<1024;i++){
   rst_L64
      = addL64(
         rst_L64,
         shrL64(
            mpy32L64(src1_int[i],src2_int[i]),
            31
         )
       );
}
```

4.4.3 32bit データの 2 乗の和

32bitデータの2乗を２つ求め、それを足して64bitとして書き込むルーチン

```
int i;
int src1_int[1024];
int src2_int[1024];
double dst1_L64[1024];

for(i=0;i<1024;i++){
   dst1_L64[i] = addL64(
                  mpy32L64(src1_int[i],src1_int[i]),
                  mpy32L64(src2_int[i],src2_int[i])
                 );
}
```

5 各方法の比較

この章では、第２～４章で説明した64bit整数演算の各実現方法についてパフォーマンス比較をおこないます。

5.1 比較条件

三つの方法について、以下の条件で、1024回演算を繰り返すプログラムにて、評価しています。

- **開発ツール**
  CCS 3.0（Cコンパイラ・バージョン5.00）
  C6416 Simulator
- **評価プログラム**
  - 例１：32bitデータの積和演算（結果64bit）
    各章の例１のサンプル・プログラム
  - 例２：32bitデータ乗算の上位のみ使用する積和演算
    各章の例２のサンプル・プログラム
  - 例３：32bitデータの2乗の和
    各章の例３のサンプル・プログラム
  - 64bitデータ加算
    64bitデータの加算を行い、その結果をストアする。これを1024回繰り返す。
  - 64bitデータ減算
    64bitデータの減算を行い、その結果をストアする。これを1024回繰り返す。
  - 32bitデータの乗算（結果64bit）
    32bitデータの乗算を行い、その64bitの結果をストアする。これを1024回繰り返す。
  - 64bitデータの乗算
    64bitデータの乗算を行い、その64bitの結果をストアする。これを1024回繰り返す。
  - 64bitデータの除算
    64bitデータの除算を行い、その64bitの結果をストアする。これを1024回繰り返す。

5.2 評価結果

評価結果を表 2に示します。
除算は、入力するデータによって、サイクル数がかわります。また、括弧の数字は、繰り返す数をｎとしたときのサイクル数を表しています。

表２ 各々の方法での評価結果

内容	1024 回繰り返した場合のサイクル数		
	方法１：long long型	方法２：long long型＋関数	方法３：Double型
例１：32bitデータの積和演算（結果64bit）	30667サイクル	3087サイクル (3n+15)	4111サイクル (4n+15)
例２：32bitデータ乗算の上位のみ使用する積和演算	31178サイクル	3614サイクル (3.5n+30)	4118サイクル (4n+22)
例３：32bitデータの2乗の和	57751サイクル	5139サイクル (5n+19)	5140サイクル (5n+20)
64bitデータ加算	1545サイクル (1.5n+9)	方法１と同じ	1549サイクル (1.5n+13)
64bitデータ減算	1545サイクル (1.5n+9)	方法１と同じ	1553サイクル (1.5n+17)
32bitデータの乗算（結果64bit）	30668サイクル	2578サイクル (2.5n+18)	2578サイクル (2.5n+18)
64bitデータの乗算	30668サイクル	6159サイクル (6n+15)	6159サイクル (6n+15)
64bitデータの除算	508263サイクル	242495サイクル	261089サイクル

参考文献

1. *TMS320C6000 Optimization Technique Application Note(日本語) (SCJA002)*
2. *TMS320C6000 Programmer's Guide(SPRU198)*
3. *TMS320C6000 Optimizing Compiler User's Guide (SPRU187)*

更新履歴

版	ページ	追加/変更/削除項目
初版 Feb/05		初版リリース