MDU(Multiplication Division Unit) 功能使用方法

1 适用产品：

1.1 SM59R16A2/ SM59R08A2

1.2 SM59R16A5/ SM59R09A5/ SM59R05A5/SM59R16A3/ SM59R09A3/SM59R05A3

1.3 SM59R04A2/ SM59R04A1/ SM59R03A1/ SM59R02A1

2 MDU 使用概述：

2.1. 使用 MCU 1T 且带有 MDU 之优势，在于大量运算系统中愈复杂则愈显而易见，执行乘除法及位移指令在数个微秒内即可完成，其相较于一般 1T 或 12T 8051 的效率高出数十倍至百倍。

2.2. 提供快速的硬件除法运算(32/16-bit & 16/16-bit division)、乘法运算(16*16-bit multiplication)、位移功能(32-bit shift)及归一化(normalize, 去除小数点运算)功能。

2.3. 乘除法功能，仅可使用无号整数形态运算。

2.4. ARCON[7:0]为 MDU 操作缓存器控制。MD5~MD0 为操作数和运算结果。

2.5. SM59Rxx 系列 MDU，***使用C 语言相当容易，只要在Keil C 中加一行指令“ #pragma mdu_r515 ”***于主程序前即可，请参考范例程序。***汇编(Assembly)可参考 MDU 操作步骤，依应用流程即可。***

2.6. 效率测试比较表，参考如下：

MCU Conditional / Program command	1T with MDU	1T without MDU	12T without MDU
0x0000FFFF / 0x00FF	**4.560**	33.88	280.0
0xFFFFFFFF / 0xFFFF	**4.601**	36.40	302.0
0xFFFF / 0x00FF	**2.320**	8.404	64.30
0xFFFF / 0xFFFF	**2.320**	9.200	76.84
0x00FF x 0xFFFF	**6.599**	7.922	62.01
0xFFFF x 0xFFFF	**6.639**	7.960	62.40
0x800FFFFF>>1	**3.678**	2.317	16.40
0x800FFFFF>>8	**3.841**	8.081	67.70
0x800FFFFF>>16	**3.840**	14.48	125.4
0x800FFFFF>>24	**3.841**	20.87	183.0
Normalizing_Write(0x00000001)	**3.079**	-	-
Normalizing_Write(0x81008000)	**2.600**	-	-
以上MUC皆使用crystal 25MHz为测试条件。 测试速度单位: us			

3 UV2 未支持 MDU 功能，建议用使者可更新为 UV3 和 UV4。针对 Keil C UV3 和 UV4 编译(Build)和软件模拟(Simulation)，如何才能完整的使用 MDU？

方法一.

Step1. 更新 UV4.cdb(或 UV3.cdb)

1-1 使用新茂提供的 UV4.cdb(或 UV3.cdb)档案直接更新

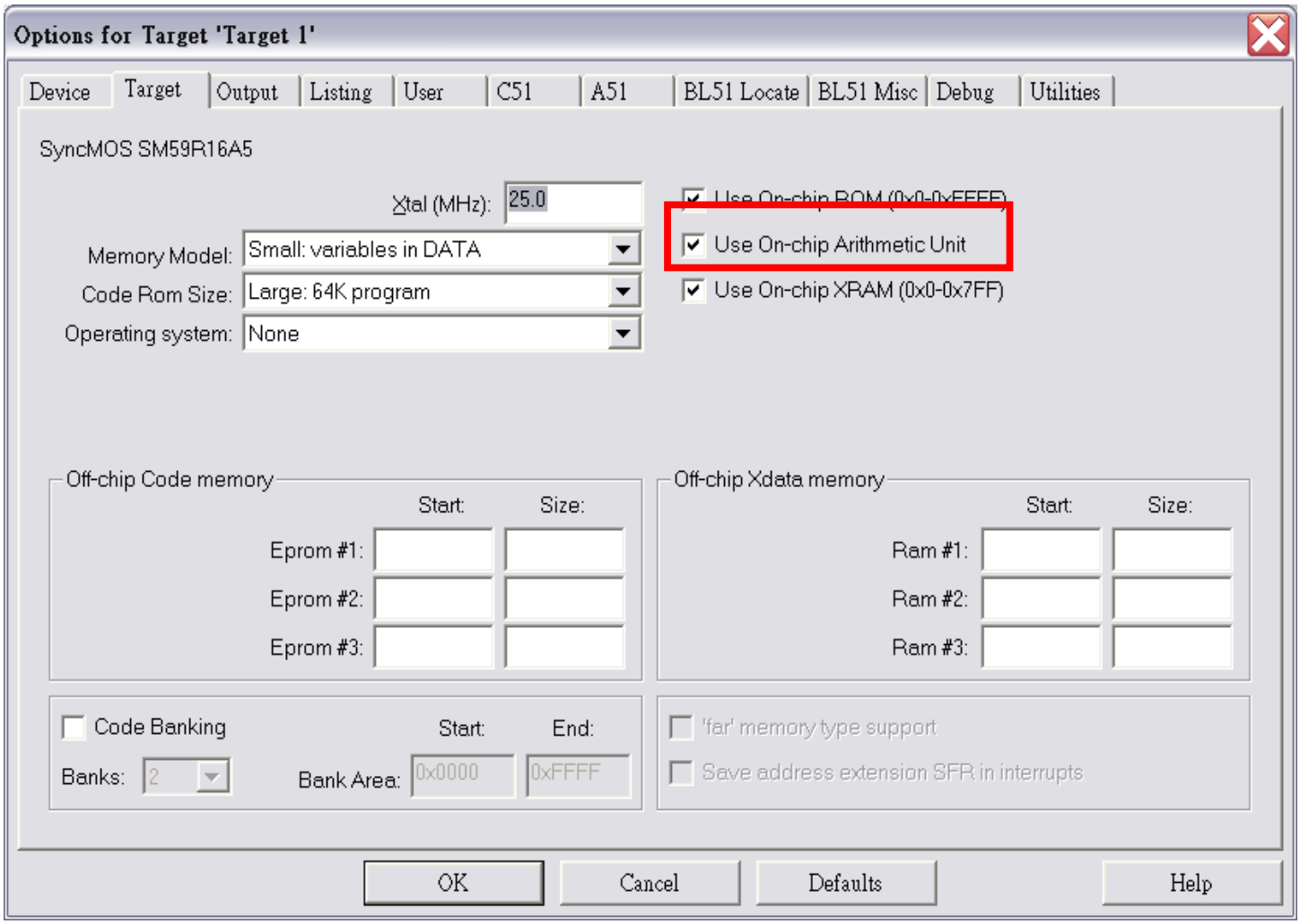
http://www.syncmos.com.tw/download_file/UV4.rar

http://www.syncmos.com.tw/download_file/UV3.rar

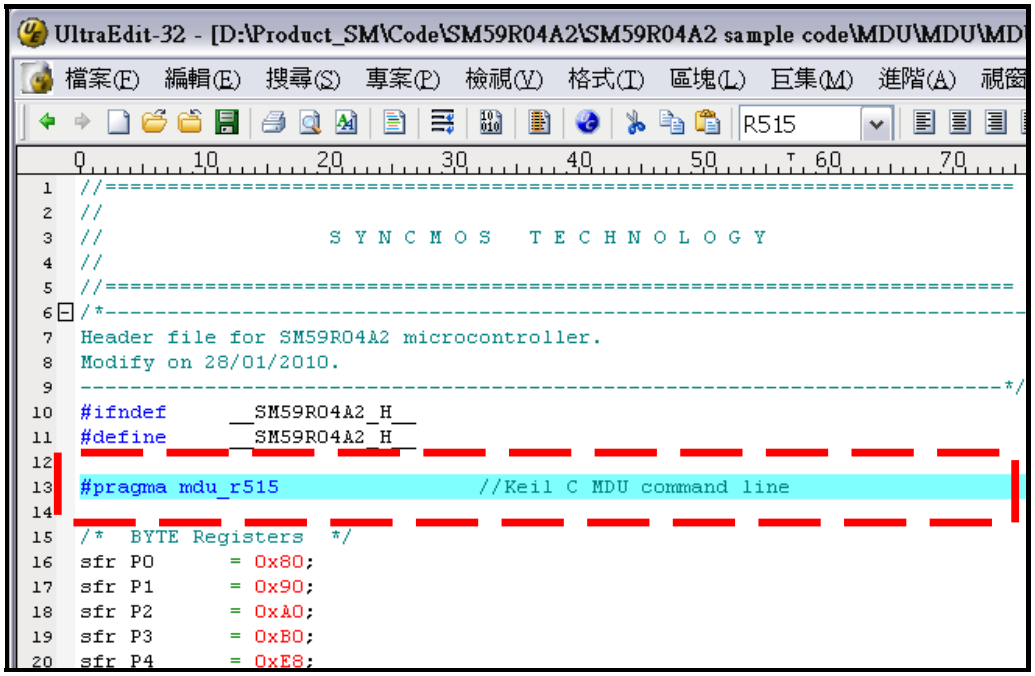
1-2 或由行用者自行修改现有的 UV4.cdb(或 UV3.cdb)，修改方式可参考文件，如下：

http://www.syncmos.com.tw/paper_file/IRFWX-A099_A_Keil%20C%20Data%20Base%20for%20new%20device%20(TC).pdf

Step2. 更新后在 Option for Target\ Target 会出现选项**"Use On-Chip Arithmetic Unit"**，将其致能即可，如下图所示：



方法二. 将 MDU 指令"***#pragma mdu_r515***"加入标头档即可。该方法 Keil C 编译没问题，但注意无法使用软件模拟(需软件模拟建议使用方法一即可)。



4 以下说明与 MDU 相关的特殊控制缓存器及特殊状态缓存器

Special Function Register (SFR)

Mnemonic	Description	Direct	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET
Multiplication Division Unit											
PCON	**Power control**	**87H**	SMOD	MDUF	-	-	-	-	STOP	IDLE	40H
ARCON	**Arithmetic Control register**	**EFh**	MDEF	MDOV	SLR	SC[4:0]					00H
MD0	**Multiplication/ Division Register 0**	**E9h**	MD0[7:0]								00H
MD1	**Multiplication/ Division Register 1**	**EAh**	MD1[7:0]								00H
MD2	**Multiplication/ Division Register 2**	**EBh**	**MD2[7:0]**								**00H**
MD3	**Multiplication/ Division Register 3**	**ECh**	**MD3[7:0]**								**00H**
MD4	**Multiplication/ Division Register 4**	**EDh**	**MD4[7:0]**								**00H**
MD5	**Multiplication/ Division Register 5**	**EEh**	**MD5[7:0]**								**00H**

MDU操作步骤，汇编(Assembly)请参考以下三个步骤（C语言的部分已在Keil C编译时完成，使用方式可直接参考范例程序）：

- **步骤一：写入MDx（x = 0~5）缓存器**

  使用 **MDU** 运算时，必须注意写入的顺序；所有的应用中缓存器**“MD0”**都必须是第一个被写入**(First write)**，其它缓存器写入后，在最后一个指定的缓存器被写入**(Last write)**时，**MDU** 即开始做运算处理：

Table 6-1: MDU registers write sequence

Operation	32bit/16bit	16bit/16bit	16bit x 16bit	shift/normalizing
First write	MD0 Dividend Low	MD0 Dividend Low	MD0 Multiplicand Low	MD0 LSB
	MD1 Dividend	MD1 Dividend High	MD4 Multiplicator Low	MD1
	MD2 Dividend		MD1 Multiplicand High	MD2
	MD3 Dividend High			MD3 MSB
	MD4 Divisor Low	MD4 Divisor Low		
Last write	MD5 Divisor High	MD5 Divisor High	MD5 Multiplicator High	ARCON start conversion

- **步骤二：执行计算**

MDU是否完成计算，可由MDUF旗标判断；当完成时由硬件设定为”1”，在下一次执行计算时硬件会自动清除。

Mnemonic: PCON **Address: 87h**

7	6	5	4	3	2	1	0	Reset
SMOD	MDUF	-	PMW	-	-	STOP	IDLE	40h

MDUF: MDU 完成旗标(MDU finish flag)

1: 完成时由硬件设定

0: 下一次 MDU 执行时自动清除

下表说明MDU各个功能的执行周期：

Table 6-2: MDU execution times

Operation	Number of Tclk
Division 32bit/16bit	17 clock cycles
Division 16bit/16bit	9 clock cycles
Multiplication	11 clock cycles
Shift	Min. 3 clock cycles, Max. 18 clock cycles
Normalize	Min. 4 clock cycles, Max. 19 clock cycles

■ **步骤三：由MDx（x = 0~5）读取结果：**

读取缓存器时，要注意 *“Last read”*字节必须是最后被读出.

- Division应用：Last read为**MD5**
- Multiplication, shift和normalizing应用：Last read为**MD3**

(如下表的斜体字形所示)

Table 6-3: MDU registers read sequence

Operation	32Bit/16Bit	16Bit/16Bit	16Bit x 16Bit	shift/normalizing
First read	***MD0 Quotient Low***	***MD0 Quotient Low***	***MD0 Product Low***	***MD0 LSB***
	MD1 Quotient	MD1 Quotient High	MD1 Product	MD1
	MD2 Quotient		MD2 Product	MD2
	MD3 Quotient High			
	MD4 Remainder L	MD4 Remainder Low		
Last read	***MD5 Remainder H***	***MD5 Remainder High***	***MD3 Product High***	***MD3 MSB***

Mnemonic: ARCON **Address: EFh**

7	6	5	4	3	2	1	0	Reset
MDEF	MDOV	SLR	SC [4:0]					00h

MDEF: 乘除错误旗标 Multiplication Division Error Flag. (只读)

此旗标将指示错误发生在不正常操作状况，例如算运已经重新开始或被中断．在第一步骤写入 MD0 时，错误旗标将自动致能；在第三步骤完成读取 MD3(乘法或位移)或 MD5(除法) 时被禁能．

1. 错误旗标被设定的情形如下：
   在第二步骤，MDx 缓存器被运算和写入数据时(重新开始或运算被中断)
   错误旗标被致能和 MDx 缓存器被读取 (没有中断运算)
2. 错误旗标被清除的情形如下：
   当第二步骤完成(运算成功)，且在第三步骤 MDx 缓存器被读取

MDOV: 乘除溢位旗标 Multiplication Division Overflow flag.(只读)

当 MD0 缓存器被写入数据(first write)时，乘除溢位旗标即被清除为”0”

溢位将于以下状况发生：

1. 除数为”0”
2. 乘法结果为 0000FFFFh
3. 使用归一化(Normalizing)功能启用之前，MD3 的 MSB 为”1”

SLR: 位移方向旗标 Shift direction bit.

SLR = 0 –左位移

SLR = 1 –右位移

SC [4:0]: 位移计数旗标 Shift counter.

当 SC[4:0] = 0，选择归一化(Normalizing)功能

当 SC[4:0]≠0，选择位移(Shift)功能，位移开始 (SC [4]：最高位；SC[0]：最低位)

5 MDU 应用程序请注意以下：

- 缓存器仍必须依顺序(First write, … , Last write)写入。
- 关于shift及normalizing的应用，ARCON中的SLR和SC[4:0]，**注意如下：**

当使用shift功能，**写入时必须同时对SLR和SC[4:0]做处理，如说明项目1&3，**参考以下范例说明：

1. 正确： **ARCON = 0x20 | value;** // set SLR & SC = Shift right & counter
2. 错误： ARCON |= 0x20; // set SLR = shift right
   ARCON |= value; // set SC = shift counter
3. 正确： **ARCON = 0x00 | value;** // set SLR & SC = Shift left & counter
4. 错误： ARCON &= 0x00; // set SLR = shift left
   ARCON |= value; // set SC = shift counter

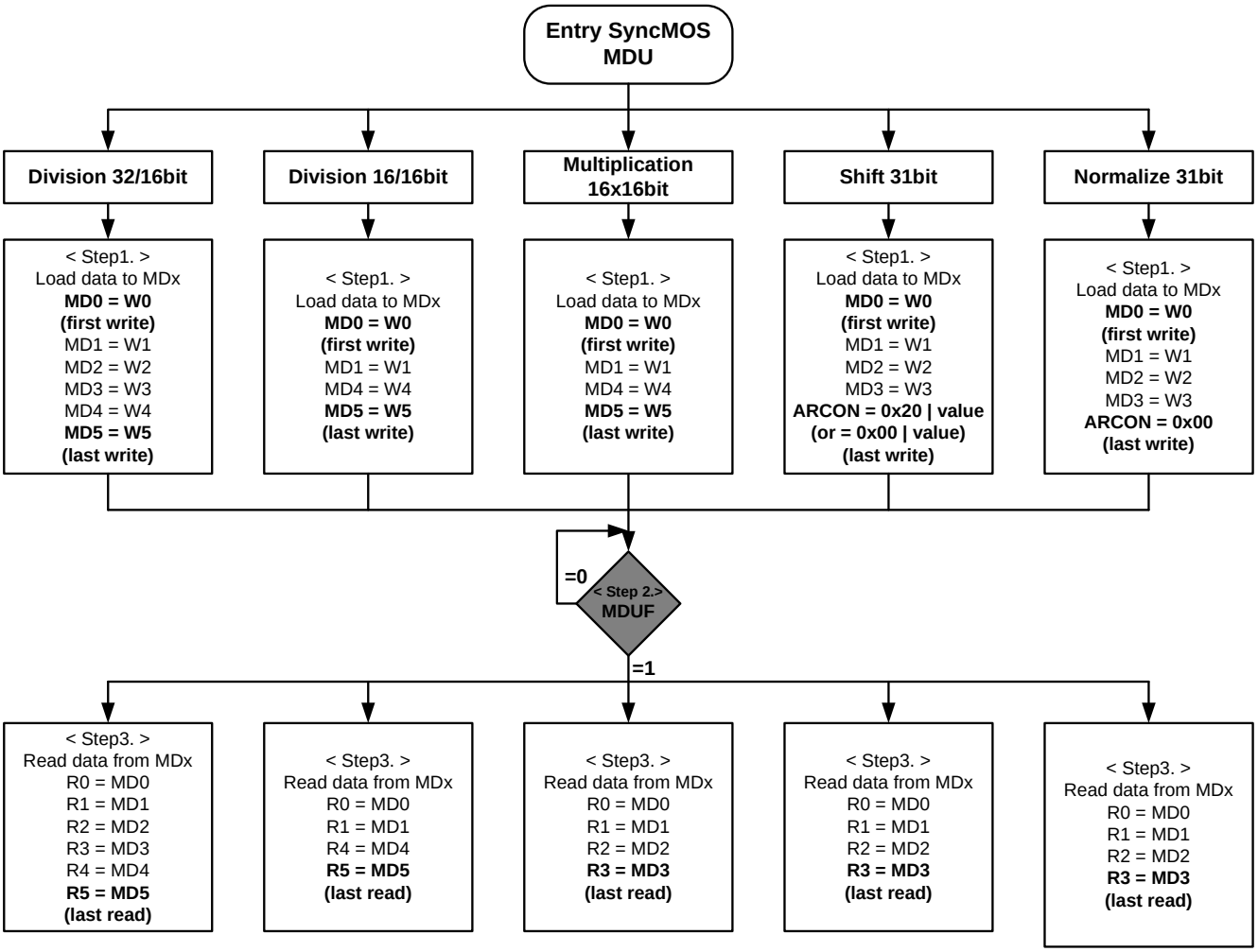
关于**shift**及**normalizing的last write的正确写法，**参考以下范例：

5. shift： **ARCON = 0x20 | value;** //shift last write
6. normalizing： **ARCON = 0x00;** // normalizing last write

当使用shift功能，**不必要也不可以对ARCON做清除动作，**参考以下范例：

7. 错误： ARCON = 0x20 | value; // set SLR & SC = Shift right & counter
   ……. // process… etc.
   ARCON = 0x00; **// unnecessary clear**

6 以下范例为 **MDU** 的应用流程图，**汇编(Assembly)请参考以下步骤（C 语言的部分在 Keil C 编译时已完成，使用方式可直接参考范例程序）**



以下数据型态定义参考自 **Keil C User's Guide：**

(http://www.keil.com/support/man/docs/c51/c51_mdu_r515.htm)

Operation	Implementation
signed int mul	Intrinsic
unsigned int mul	Intrinsic
signed int div	?C?SIDIVR515 routine
unsigned int div	?C?UIDIVR515 routine

signed long mul	?C?LMULR515 routine
unsigned long mul	?C?LMULR515 routine
signed long div	?C?SLDIVR515 routine
unsigned long div	?C?ULDIVR515 routine
signed long shift left	?C?LSHLR515 routine
unsigned long shift left	?C?LSHLR515 routine
unsigned long shift right	?C?ULSHRR515 routine

7 MDU 的范例程序

Description	**MDU** 支持方法二的范例,**#pragma mdu_r515** 为 **Keil C** 的 **command line**,加到 **file.c** 或 **headfile.h** 皆可，但建议加至 **headfile.h**，例如 **SM59R04A2.h** 中即可。

Main program	//==
	//
	// S Y N C M O S T E C H N O L O G Y
	//
	//==
	#include <SM59R04A2.h>
	
	#pragma mdu_r515 //Keil C MDU command line
	
	//typedef for normalizing ================================
	typedef union {
	  struct {
	  unsigned char Byte3; //MSB
	  unsigned char Byte2;
	  unsigned char Byte1;
	  unsigned char Byte0; //LSB
	  }ss1;
	  struct {
	  unsigned int Word1; //MSW
	  unsigned int Word0; //LSW
	  }ss2;

```
    unsigned long Ldata;
}Long_Data;
//=========================================================

void division_32_16(void)
{
    unsigned int    ui_data_002;            // ?C?UIDIVR515 routine
    unsigned long   ul_data_001;            // ?C?LMULR515 routine
    unsigned long   ul_data_002;            // ?C?ULDIVR515 routine

    // 32/16 ANS:0x00000101================================
    ul_data_002= 0x0000FFFF;
    ui_data_002= 0x00FF;
    P3_0 =0;
    ul_data_001 = ul_data_002/ui_data_002;  //4.560us   @25MHz  1T with MDU
                                            //33.88us   @25MHz  1T without MDU
                                            //280.0us       @25MHz 12T without
MDU
    P3_0 =1;
    P0 = ul_data_001;
    P1 = ul_data_001>>8;

    // 32/16 ANS:0x00010001================================
    ul_data_002= 0xFFFFFFFF;
    ui_data_002= 0xFFFF;
    P3_1 =0;
    ul_data_001 = ul_data_002/ui_data_002;  //4.601us   @25MHz  1T with MDU
                                            //36.40us   @25MHz  1T without MDU
                                            //302.0us       @25MHz 12T without
MDU
    P3_1 =1;
    P0 = ul_data_001;
    P1 = ul_data_001>>8;
    P2 = ul_data_001>>16;
}
```

```
void division_16_16(void)
{
    unsigned int    ui_data_001;
    unsigned int    ui_data_002;
    unsigned int    ui_data_003;

    // 16/16 ANS:0x0101================================
    ui_data_002= 0xFFFF;
    ui_data_003= 0x00FF;
    P3_2 =0;                                    //trig
    ui_data_001 = ui_data_002/ui_data_003;      //2.32us    @25MHz 1T with MDU
                                                //8.404us   @25MHz 1T without MDU
                                                //64.30us   @25MHz 12T without MDU
    P3_2 =1;                                    //trig
    P0 = ui_data_001;
    P1 = ui_data_001>>8;
    P2 = ARCON;

    // 16/16 ANS:0x0001================================
    ui_data_002= 0xFFFF;
    ui_data_003= 0xFFFF;
    P3_3 =0;                                    //trig
    ui_data_001 = ui_data_002/ui_data_003;      //2.320us   @25MHz 1T with MDU
                                                //9.200us   @25MHz 1T without MDU
                                                //76.84us   @25MHz 12T without MDU
    P3_3 =1;                                    //trig
    P0 = ui_data_001;
    P1 = ui_data_001>>8;
    P2 = ARCON;                                 //=0X00
}

void multiplication_16x16(void)
{
    unsigned int    ui_data_002;            // ?C?UIDIVR515 routine
```

```
    unsigned long   ul_data_001;            // ?C?LMULR515 routine
    unsigned long   ul_data_003;

    // 16x16 ANS:0x00FEFF01================================
    ui_data_002= 0x00FF;
    ul_data_001= 0xFFFF;
    P3_4 =0;
    ul_data_003 = ul_data_001*ui_data_002;  //6.599us   @25MHz 1T with MDU
                                            //7.922us   @25MHz 1T without MDU
                                            //62.01us   @25MHz 12T without MDU
    P3_4 =1;
    P0 = ul_data_003;
    P1 = ul_data_003>>8;
    P2 = ul_data_003>>16;

    // 16x16 ANS:0xFFFE0001================================
    ul_data_001= 0xFFFF;
    ui_data_002= 0xFFFF;
    P3_5 =0;
    ul_data_003 = ul_data_001*ui_data_002;  //6.639us   @25MHz 1T with MDU
                                            //7.960us   @25MHz 1T without MDU
                                            //62.4us    @25MHz 12T without MDU
    P3_5 =1;
    P0 = ul_data_003;
    P1 = ul_data_003>>8;
    P2 = ul_data_003>>16;
}

void Shift(void)
{
    unsigned long ul_data_003;
//=========================================================
//Shift
    ul_data_003=0x800FFFFF;
```

```
P3_0=0;                  //trig
P0 = ul_data_003>>1;     //3.678us   @25MHz 1T with MDU
                         //2.317us   @25MHz 1T without MDU
                         //16.4us    @25MHz 12T without MDU
P3_0=1;                  //trig

P3_1=0;                  //trig
P1 = ul_data_003>>8;     //3.841us   @25MHz 1T with MDU
                         //8.081us   @25MHz 1T without MDU
                         //67.7us    @25MHz 12T without MDU
P3_1=1;                  //trig

P3_2=0;                  //trig
P0 = ul_data_003>>15;    //3.680us   @25MHz 1T with MDU
                         //13.52us   @25MHz 1T without MDU
                         //117.1us   @25MHz 12T without MDU
P3_2=1;                  //trig

P3_3=0;                  //trig
P1 = ul_data_003>>16;    //3.840us   @25MHz 1T with MDU
                         //14.48us   @25MHz 1T without MDU
                         //125.4us   @25MHz 12T without MDU
P3_3=1;                  //trig

P3_4=0;                  //trig
P2 = ul_data_003>>24;    //3.841us   @25MHz 1T with MDU
                         //20.87us   @25MHz 1T without MDU
                         //183.0us   @25MHz 12T without MDU
P3_4=1;                  //trig

P3_5=0;                  //trig
P0 = ul_data_003>>31;    //3.687us   @25MHz 1T with MDU
                         //26.32us   @25MHz 1T without MDU
                         //232.4us   @25MHz 12T without MDU
P3_5=1;                  //trig
```

```
    P3_6=0;                 //trig
    P0 = ul_data_003>>8;    //0xFF  //17.36us   @25MHz 1T with MDU
                                    //42.72us   @25MHz 1T without MDU
                                    //372.0us   @25MHz 12T without MDU
    P1 = ul_data_003>>16;   //0x0F
    P2 = ul_data_003>>24;   //0x80
    P3_6=1;                 //trig

    P3_7=0;                 //trig
    P0 = ul_data_003<<8;    //0x00  //10.56us   @25MHz 1T with MDU
                                    //42.72us   @25MHz 1T without MDU
                                    //372.0us   @25MHz 12T without MDU
    P1 = ul_data_003<<16;   //0x00
    P2 = ul_data_003<<24;   //0x00
    P3_7=1;                 //trig

}

void Normalizing_Write(unsigned long Data)
{
    Long_Data LD;
    LD.Ldata =Data;
    MD0 = LD.ss1.Byte0;
    MD1 = LD.ss1.Byte1;
    MD2 = LD.ss1.Byte2;
    MD3 = LD.ss1.Byte3;

    ARCON  = 0x00 ;         // Start Normalizing
    while(!(PCON & 0x40))   //check MDU finish flag
    {};
}

void Normalizing_Read(void)
{
```

```
    Long_Data LD;
    LD.ss1.Byte0 = MD0; //first read
    LD.ss1.Byte1 = MD1;
    LD.ss1.Byte2 = MD2;
    LD.ss1.Byte3 = MD3; //last read, MDEF(error flag) happen if not read
}

void Normalizing_test(void)
{
    P3_0=0;                                 //trig = 3.079us
    Normalizing_Write(0x00000001);          //ANS=0x1F  -->MDOV=0
    P3_0=1;                                 //trig
    Normalizing_Read();
    P0 = ARCON;

    P3_1=0;                                 //trig = 2.681us
    Normalizing_Write(0x00080000);          //ANS=0x03  -->MDOV=0
    P3_1=1;                                 //trig
    Normalizing_Read();
    P1 = ARCON;

    P3_2=0;                                 //trig = 2.603us
    Normalizing_Write(0x00080001);          //ANS=0x0C  -->MDOV=0
    P3_2=1;                                 //trig
    Normalizing_Read();
    P2 = ARCON;

    P3_3=0;                                 //trig = 2.600us
    Normalizing_Write(0x81008000);          //ANS=0x40  -->MDOV=1
    P3_3=1;                                 //trig
    P0 = ARCON;
    Normalizing_Read();
}

void main(void)
```

```
{
    IFCON |= 0x80;
    division_32_16();
    division_16_16();
    multiplication_16x16();
    Shift();
    Normalizing_test();

    while(1){};
}
```