



MDU(Multiplication Division Unit) 功能使用方法

1 適用產品：

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 且帶有 MDU 之優勢，在於大量運算系統中愈複雜則愈顯而易見，執行乘除法及位移指令在數個微秒內即可完成，其相較於一般 1T 或 12T 8051 的效率高出數十倍至百倍。

2.2. 提供快速的硬體除法運算(32/16-bit & 16/16-bit division)、乘法運算(16*16-bit multiplication)、位移功能(32-bit shift)及歸一化(normalize, 去除小數點運算)功能。

2.3. 乘除法功能，僅可使用無號整數形態運算。

2.4. ARCON[7:0]為 MDU 操作暫存器控制。MD5~MD0 為運算元和運算結果。

2.5. SM59Rxx 系列 MDU，*使用 C 語言相當容易，只要在 Keil C 中加一行指令 “#pragma mdu_r515 ”* 於主程序前即可，請參考範例程式。*彙編(Assembly)可參考 MDU 操作步驟，依應用流程即可。*

2.6. 效率測試比較表，參考如下：

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
0x800FFFFFF>>1	3.678	2.317	16.40
0x800FFFFFF>>8	3.841	8.081	67.70
0x800FFFFFF>>16	3.840	14.48	125.4
0x800FFFFFF>>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為測試條件。 測試速度單位: us



- 3 UV2 未支援 MDU 功能，建議用使用者可更新為 UV3 和 UV4。針對 Keil C UV3 和 UV4 編譯(Build)和軟件模擬(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)檔案直接更新

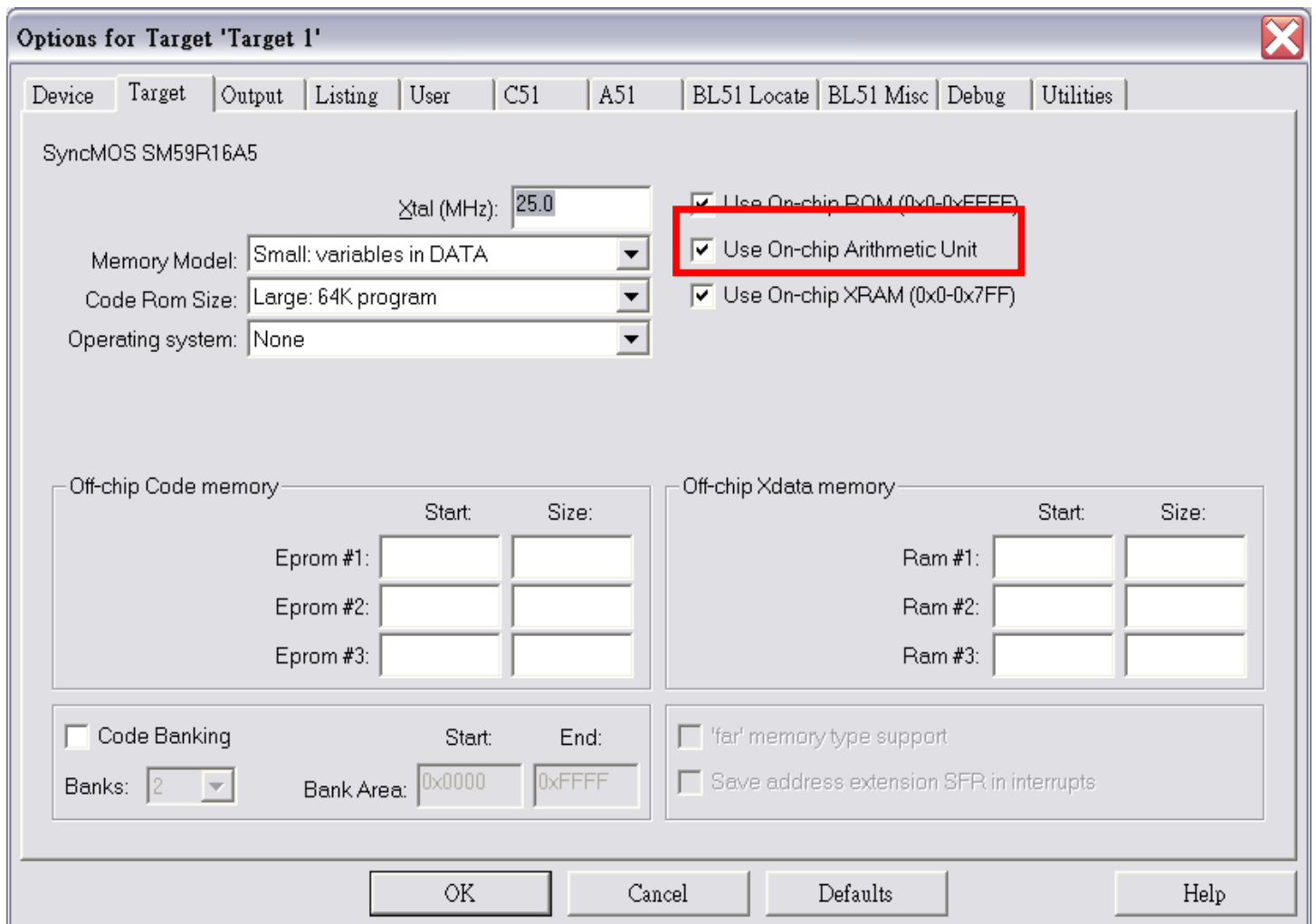
http://www.syncmos.com.tw/download_file/UV4.rar

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

1-2 或由行用者自行修改現有的 UV4.cdb(或 UV3.cdb)，修改方式可參考文件，如下：

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

Step2. 更新後在 Option for Target\ Target 會出現選項”Use On-Chip Arithmetic Unit”，將其致能即可，如下圖所示：





方法二. 將 MDU 指令”**#pragma mdu_r515**”加入標頭檔即可。該方法 Keil C 編譯沒問題，但注意無法使用軟件模擬(需軟件模擬建議使用方法一即可)。

```

1 //-----
2 //
3 //             S Y N C M O S   T E C H N O L O G Y
4 //
5 //-----
6 /*-----*/
7 Header file for SM59R04A2 microcontroller.
8 Modify on 28/01/2010.
9 -----*/
10 #ifndef      SM59R04A2_H
11 #define      SM59R04A2_H
12
13 #pragma mdu_r515 //Keil C MDU command line
14
15 /* BYTE Registers */
16 sfr P0      = 0x80;
17 sfr P1      = 0x90;
18 sfr P2      = 0xA0;
19 sfr P3      = 0xB0;
20 sfr P4      = 0xE8;

```

4 以下說明與 MDU 相關的特殊控制暫存器及特殊狀態暫存器

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				

Specifications subject to change without notice, contact your sales representatives for the most recent information.



MDU操作步驟，彙編(Assembly)請參考以下三個步驟（C語言的部分已在Keil C編譯時完成，使用方式可直接參考範例程式）：

■ 步驟一：寫入MDx (x = 0~5) 暫存器

使用 MDU 運算時，必須注意寫入的順序；所有的應用中暫存器“MD0”都必須是第一個被寫入 (First write)，其他暫存器寫入後，在最後一個指定的暫存器被寫入(Last write)時，MDU 即開始做運算處理：

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 Multiplier 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 Multiplier High	ARCON start conversion

■ 步驟二：執行計算

MDU是否完成計算，可由MDUF旗標判斷；當完成時由硬體設定為“1”，在下一執行計算時硬體會自動清除。

Mnemonic: PCON								Address: 87h	
7	6	5	4	3	2	1	0	Reset	
SMOD	MDUF	-	PMW	-	-	STOP	IDLE	40h	

MDUF: MDU 完成旗標(MDU finish flag)

1: 完成時由硬體設定

0: 下一次 MDU 執行時自動清除

下表說明MDU各個功能的執行週期：

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



■ 步驟三: 由MDx (x = 0~5) 讀取結果:

讀取暫存器時, 要注意 “Last read” 位元組必須是最後被讀出。

- Division應用: Last read為MD5
- Multiplication, shift和normalizing應用: Last read為MD3

(如下表的斜體字形所示)

Table 6-3: MDU registers read sequence

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

Mnemonic: ARCON

Address: EFh

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

MDEF: 乘除錯誤旗標 Multiplication Division Error Flag. (唯讀)

此旗標將指示錯誤發生在不正常操作狀況, 例如算運已經重新開始或被中斷。在第一步驟寫入 MD0 時, 錯誤旗標將自動致能; 在第三步驟完成讀取 MD3(乘法或位移)或 MD5(除法) 時被禁能。

1. 錯誤旗標被設定的情形如下:

在第二步驟, MDx 暫存器被運算和寫入資料時(重新開始或運算被中斷)
錯誤旗標被致能和 MDx 暫存器被讀取 (沒有中斷運算)

2. 錯誤旗標被清除的情形如下:

當第二步驟完成(運算成功), 且在第三步驟 MDx 暫存器被讀取

MDOV: 乘除溢位旗標 Multiplication Division Overflow flag.(唯讀)

當 MD0 暫存器被寫入資料(first write)時, 乘除溢位旗標即被清除為”0”

溢位將於以下狀況發生:

1. 除數為”0”
2. 乘法結果為 0000FFFFh
3. 使用歸一化(Normalizing)功能啓用之前, MD3 的 MSB 為”1”

SLR: 位移方向旗標 Shift direction bit.



SLR = 0 -左位移

SLR = 1 -右位移

SC [4:0]: 位移計數旗標 Shift counter.

當 SC[4:0] = 0，選擇歸一化(Normalizing)功能

當 SC[4:0] ≠ 0，選擇位移(Shift)功能，位移開始 (SC [4]：最高位；SC[0]：最低位)

5 MDU 應用程序請注意以下：

- 暫存器仍必須依順序(First write, ... , Last write)寫入。
- 關於shift及normalizing的應用，ARCON中的SLR和SC[4:0]，注意如下：
當使用shift功能，寫入時必須同時對SLR和SC[4:0]做處理，如說明項目1&3，參考以下範例說明：

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

關於shift及normalizing的last write的正確寫法，參考以下範例：

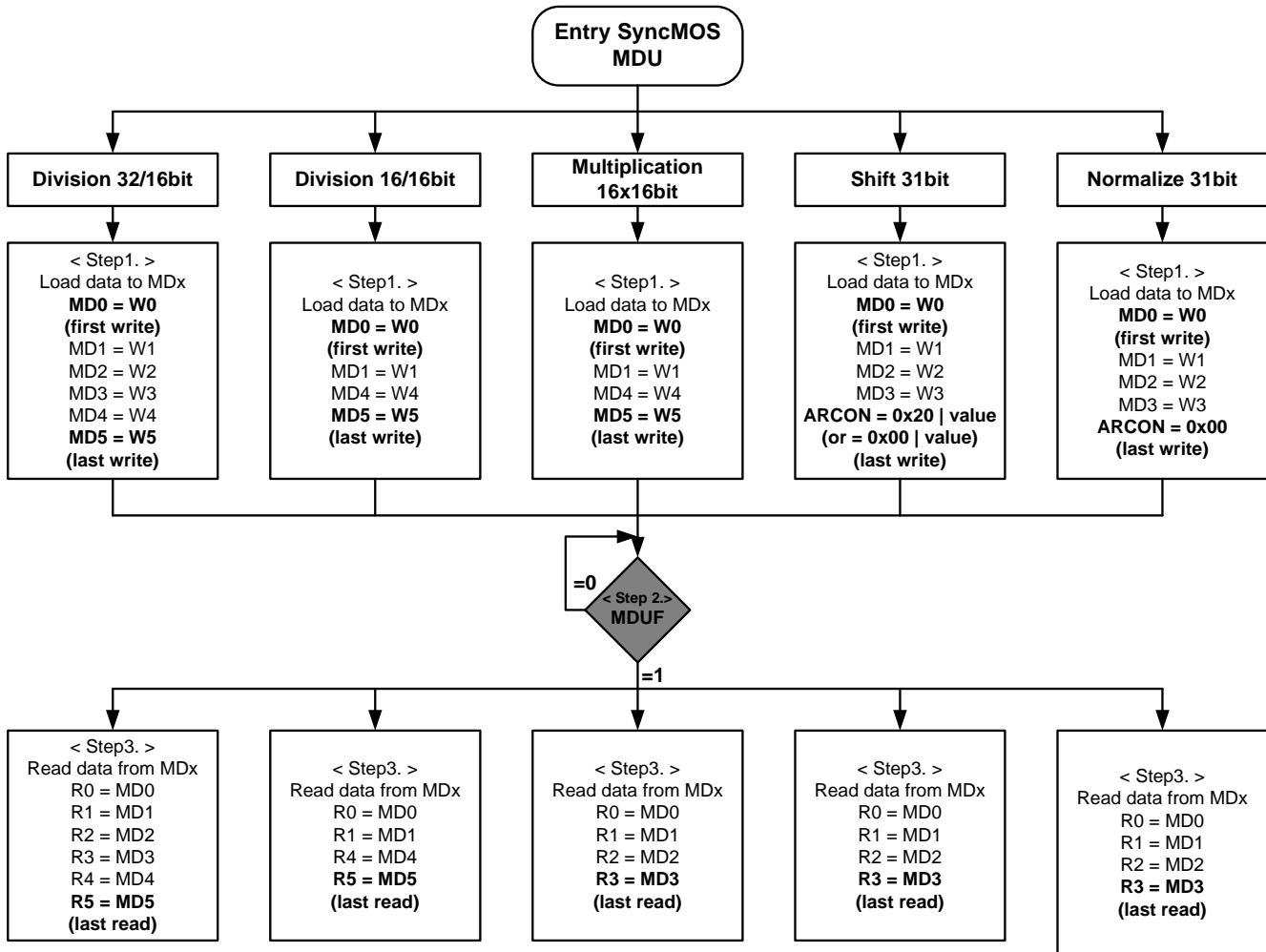
5. shift：**ARCON = 0x20 | value;** //shift last write
6. normalizing：**ARCON = 0x00;** // normalizing last write

當使用shift功能，不必要也不可以對ARCON做清除動作，參考以下範例：

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



6 以下範例為 MDU 的應用流程圖，彙編(Assembly)請參考以下步驟（C 語言的部分在 Keil C 編譯時已完成，使用方式可直接參考範例程式）



以下資料型態定義參考自 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 的範例程式

Description	MDU 支援方法二的範例，#pragma mdu_r515 為 Keil C 的 command line，加到 file.c 或 headfile.h 皆可，但建議加至 headfile.h，例如 SM59R04A2.h 中即可。
Main program	<pre> //===== // // 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; </pre>



```
    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:0xFFFFE0001=====
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=0x800FFFFFF;
```



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.ssl.Byte0 = MD0; //first read

LD.ssl.Byte1 = MD1;

LD.ssl.Byte2 = MD2;

LD.ssl.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){};  
}
```