

## 实验三十八：DSP\_MATH 库测试

### 一、实验目的与意义

- 1、了解 STM32 DSP 结构
- 2、了解 STM32 DSP 特征
- 3、掌握 DSP 的使用方法
- 4、掌握 STM32 HAL 库中 DSP 属性的配置方法
- 5、掌握 KEIL MDK 集成开发环境使用方法

### 二、实验设备及平台

- 1、iCore4 双核心板
- 2、JLINK（或相同功能）仿真器
- 3、Micro USB 线缆
- 4、Keil MDK 开发平台
- 5、STM32CubeMX 开发平台
- 6、装有 WIN XP（及更高版本）系统的计算机

### 三、实验原理

#### 1、DSP 简介

STM32F7 采用 Cortex-M7 内核，相比 Cortex-M3 系列除了内置硬件 FPU 单元，在数字信号处理方面还增加了 DSP 指令集，支持诸如单周期乘加指令（MAC），优化的单指令多数据指令（SIMD），饱和算数等多种数字信号处理指令集。相比 Cortex-M3，Cortex-M4 在数字信号处理能力方面得到了大大的提升。Cortex-M7 执行所有的 DSP 指令集都可以在单周期内完成，而 Cortex-M3 需要多个指令和多个周期才能完成同样的功能。

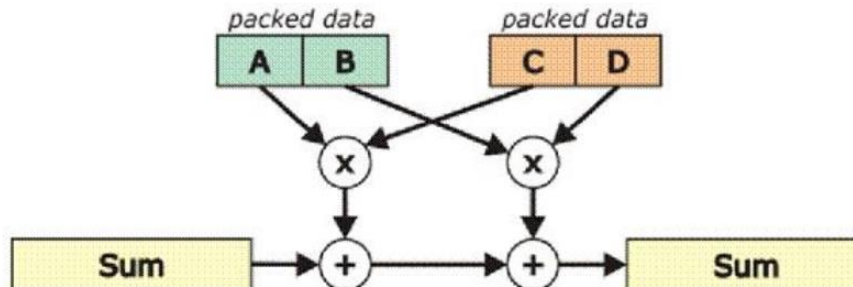
接下来我们来看看 Cortex-M7 的两个 DSP 指令：MAC 指令（32 位乘法累加）和 SIMD 指令。

32 位乘法累加（MAC）单元包括新的指令集，能够在单周期内完成一个  $32 \times 32 + 64 \rightarrow 64$  的操作或两个  $16 \times 16$  的操作，其计算能力，如下表所示：

计算	指令	周期
$16 \times 16 = 32$	SMULBB, SMULBT, SMULTB, SMULTT	1
$16 \times 16 + 32 = 32$	SMLABB, SMLABT, SMLATB, SMLATT	1
$16 \times 16 + 64 = 64$	SMLALBB, SMLALBT, SMLALTB, SMLALTT	1
$16 \times 32 = 32$	SMULWB, SMULWT	1
$(16 \times 32) + 32 = 32$	SMLAWB, SMLAWT	1
$(16 \times 16) \pm (16 \times 16) = 32$	SMUAD, SMUADX, SMUSD, SMUSDX	1
$(16 \times 16) \pm (16 \times 16) + 32 = 32$	SMLAD, SMLADX, SMLSD, SMLSDX	1
$(16 \times 16) \pm (16 \times 16) + 64 = 64$	SMLALD, SMLALDX, SMLSLD, SMLSLDX	1
$32 \times 32 = 32$	MUL	1
$32 \pm (32 \times 32) = 32$	MLA, MLS	1
$32 \times 32 = 64$	SMULL, UMULL	1
$(32 \times 32) + 64 = 64$	SMLAL, UMLAL	1
$(32 \times 32) + 32 + 32 = 64$	UMAAL	1
$2 \pm (32 \times 32) = 32$ (上)	SMMLA, SMMLAR, SMMLS, SMMLSR	1
$(32 \times 32) = 32$ (上)	SMMUL, SMMULR	1

Cortex-M7 支持 SIMD 指令集,这在 Cortex-M3/M0 系列是不可用的。上述表中的指令,有的属于 SIMD 指令。与硬件乘法器一起工作 (MAC),使所有这些指令都能在单个周期内执行。受益于 SIMD 指令的支持,Cortex-M4 处理器能在单周期内完成高达  $32 \times 32 + 64 \rightarrow 64$  的运算,为其他任务释放处理器的带宽,而不是被乘法和加法消耗运算资源。

比如一个比较复杂的运算:两个  $16 \times 16$  乘法加上一个 32 位加法,如图所示:



本实验进行 DSP 浮点运算测试,分别测试出不使用 DSP MATH 库和使用 DSP MATH 库的运算时间,进行对比。

## 四、实验程序

### 1、主函数

```
int main(void)
{
    int i,j;
    int res;
    float time[2];
    static int error_flag = 0;

    /* MCU 配置*/
    /* 重置所有外设, 初始化 Flash 接口和 SysTick. */
    HAL_Init();
    /* 系统时钟配置 */
    SystemClock_Config();
    /* 初始化所有已配置的外设 */
    MX_GPIO_Init();
    MX_USART6_UART_Init();
    MX_TIM3_Init();

    usart6.initialize(115200);           //串口波特率设置
    usart6.printf("\x0c");               //清屏
    usart6.printf("\033[1;32;40m");      //设置终端字体为绿色
    usart6.printf("Hello, I am iCore4!\r\n\r\n");
    usart6.printf("DSP BasicMath TEST.....\r\n");

    while (1)
    {
        timeout = 0;
        __HAL_TIM_SET_COUNTER(&htim3,0); //重设 TIM3 定时器的计数器值
    }
}
```

```
for(j = 0;j < 10000;j++){
    for(i = 0;i < MAX_BLOCKSIZE;i ++){
        res = SinCos_Test(testInput_f32[i],0);
        if(res != 0)error_flag ++;
    }
}
time[0] = __HAL_TIM_GET_COUNTER(&htim3)+ timeout*5000;

timeout = 0;
__HAL_TIM_SET_COUNTER(&htim3,0);
for(j = 0;j < 10000;j++){
    for(i = 0;i < MAX_BLOCKSIZE;i ++){
        res = SinCos_Test(testInput_f32[i],1);
        if(res != 0)error_flag ++;
    }
}
time[1] = __HAL_TIM_GET_COUNTER(&htim3)+ timeout*5000;

if(error_flag == 0){
    usart6.printf("NO DSP MATHLIB runtime:%0.1fms *USE DSP MATHLIB runtime:%0.1fms\r",time[0] / 10, time[1] / 10);
    LED_GREEN_ON;
    LED_RED_OFF;
    LED_BLUE_OFF;
}
else{//测试失败
    usart6.printf("Error\r");
    LED_GREEN_OFF;
    LED_RED_ON;
    LED_BLUE_OFF;
}
}
```

## 2、Sin Cos 测试

```
int SinCos_Test(float testInput,unsigned char mode)
{
    float Sinx,Cosx;
    float Result;

    switch (mode){
        case 0://不使用 DSP MATH 库
            Sinx = sinf(testInput); //不使用 DSP 优化的 sin, cos 函数

            Cosx = cosf(testInput);
            Result = Sinx*Sinx + Cosx*Cosx; //计算结果应该等于 1
            Result = fabsf(Result-1.0f); //对比与 1 的差值
            if(Result > DELTA)return -1; //判断
            break;

        case 1://使用 DSP MATH 库
            Sinx = arm_sin_f32(testInput); //使用 DSP 优化的 sin, cos 函数
            Cosx = arm_cos_f32(testInput);
            Result = Sinx*Sinx + Cosx*Cosx; //计算结果应该等于 1
            Result = fabsf(Result-1.0f); //对比与 1 的差值
            if(Result > DELTA)return -1; //判断
            break;

        default:
            break;
    }

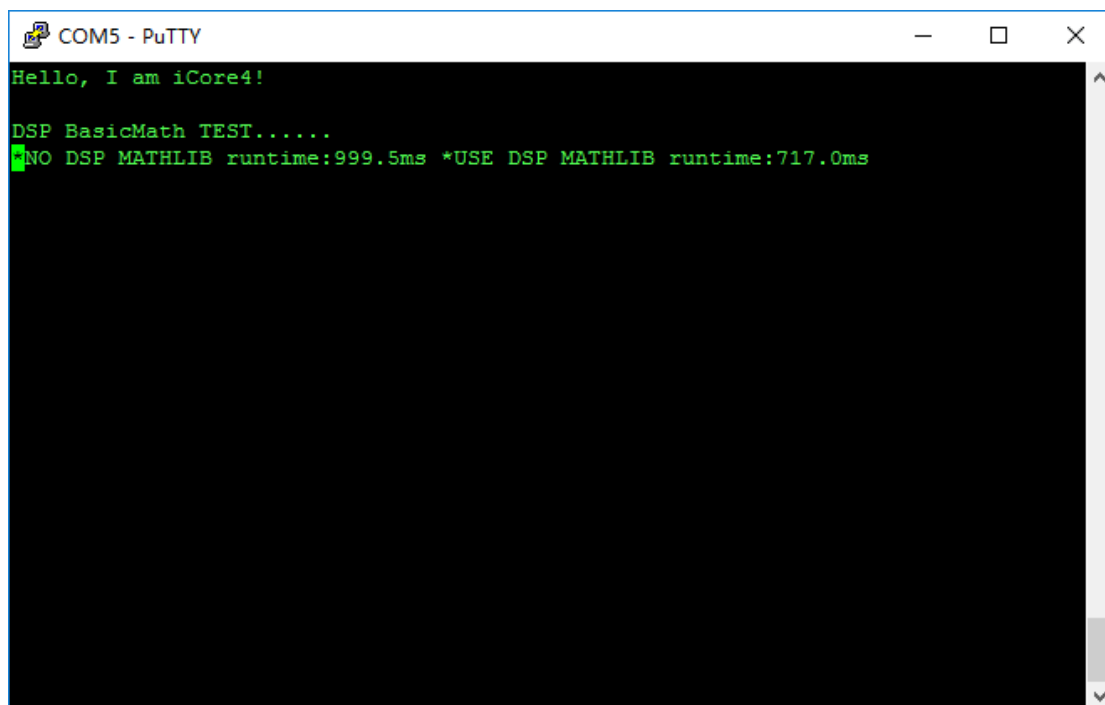
    return 0;
}
```

## 五、实验步骤

- 1、把仿真器与 iCore4 的 SWD 调试口相连（直接相连或者通过转接器相连）；
- 2、把 iCore4 通过 Micro USB 线与计算机相连，为 iCore4 供电；
- 3、打开 Keil MDK 开发环境，并打开本实验工程；
- 4、烧写程序到 iCore4 上；
- 5、也可以进入 Debug 模式，单步运行或设置断点验证程序逻辑。

## 六、实验现象

测试成功绿色 LED 点亮，并在终端上显示不使用 DSP MATH 和使用 DSP MATH 的运算时间；测试失败红色 LED 点亮，并在终端上显示“Error”。



```
COM5 - PuTTY
Hello, I am iCore4!
DSP BasicMath TEST.....
NO DSP MATHLIB runtime:999.5ms *USE DSP MATHLIB runtime:717.0ms
```