NuMicro® M4系列 DSP 应用说明与程序范例

Application Note for M451 Series, NUC472/442 and NUC505

Document Information

Abstract	本文件介绍新唐NuMicro® M4内核系列中的DSP所具备的基本运算指令，DSP链接库使用方法以及DSP运用于各个常用运算的加速效能等，协助使用者能充分利用新唐NuMicro® M4内核系列的DSP优势，撰写出更高效能的运算代码。
Apply to	具有DSP的全系列M4内核微控制器，例：M451系列，NUC472/442及NUC505

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

目录

图目录

表目录

1 绪论

一般来说DSP分为两类，其一为数字信号处理－Digital signal processing，指的是将信号以数字方式表示并加以处理的理论和技术，并不限定需要专用的处理器，其目的为对真实世界中的连续模拟信号进行测量或滤波的一种方法。另外一类为数字信号处理器－Digital signal processor，是一种专用于数字信号处理的微处理器，具备专用的信号处理硬件，专责于信号的快速处理与转换。ARM®内核中的DSP属于Digital signal processor的一种，藉由几个扩充的多任务运算指令，加以组合出特定的数字信号处理功能，达到加速运算的目的。

Cortex®-M4为DSP扩充的多任务运算指令主要为：乘积累加(MAC, or Multiply-and-Accumulate)和单指令多数据流(SIMD, or Single Instruction Multiple Data)。DSP(Digital Signal Processing，数字信号处理)中会使用大量的数值运算，藉由DSP的多任务运算指令可以提高MCU处理与分析数字信号的能力。同时ARM®提供了CMSIS (Cortex Microcontroller Software Interface Standard) DSP函数库，内含优化后高效能的DSP算法，包含许多常用数学函数集合，用户无需自行组合多任务指令，只需调用这些函数即可。

1.1 Cortex®-M4 带浮点数与 DSP 的运算优势

下表1-1列举了部分的Cortex®-M系列微控制器的比较，可以看出Cortex®-M4在指令集上有硬件乘法、硬件除法、饱和运算、数字信号处理延伸指令以及浮点运算相较Cortex®-M3有明显的优势。

ARM® Cortex®-M	硬件乘法器	硬件除法器	DSP功能	浮点运算
Cortex®-M0+	单一频率	选项	无	无
Cortex®-M3	单一频率	有	无	无
Cortex®-M4	单一频率	有	有	选项

表 1-1 Cortex®-M 系列微控制器比较

一般熟知的DSP处理器，还可以再详细分为两类：

1.数字信号处理器(Digital Signal Processor)

2.数字信号控制器(Digital Signal Controller)

第一类即一般印象中的数字信号处理器，其特点就是架构完全针对数字信号处理而优化，是真正能够持续的以单一频率执行乘积累加运算，亦即乘积累加所需的操作数会在下一个频率就已经加载并执行运算，而不用等待操作数从内存加载的时间，发挥极致的运算能力，但是设计复杂度比较高，一般来说省电性不如微处理器或微控制器。大部分的DSP只支持定点运算，因为

同样的面积与功耗，定点运算能够提供更高的运算能力，而且以信号处理的应用而言，不需要浮点数这么高的动态范围。当然浮点DSP还是有其优势，尤其是在一些必须缩短开发时间的应用上，浮点数能力使得程序设计师不需要耗费太多精神与时间处理定点数运算。

第二类是将微控制器整合数字信号处理能力而成，以微控制器为主，特点为除了强化的信号处理能力外同时拥有丰富的周边支持与输出入脚位。ARM® Cortex®-M4属于这一个分类，并且因为ARM架构被广泛的支持，使得ARM® Cortex®-M4拥有丰富的工具可以应用。

1.2 NuMicro® M4 系列 DSP 功能介绍

1.2.1 乘积累加(Multiply-and-Accumulate)

乘积累加(MAC, or Multiply-and-Accumulate)，在信号处理中常用的运算，不管是矩阵相乘运算(1.1)，或是卷积运算(Convolution)和有限脉冲响应滤波器(FIR)(1.2)，还是快速傅立叶变换(FFT)中很重要的butterfly运算(1.3)及(1.4)都非常仰赖乘积累加。

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} \times \begin{bmatrix} b11 & b12 & b13 \\ b21 & b22 & b23 \\ b31 & b32 & b33 \end{bmatrix} = \begin{bmatrix} a11 \times b11 + a12 \times b21 + a13 \times b31 & \dots & \dots \\ a21 \times b11 + a22 \times b21 + a23 \times b31 & \dots & \dots \\ a31 \times b11 + a32 \times b21 + a33 \times b31 & \dots & \dots \end{bmatrix} \quad (1.1)$$

$$\mathrm{y[n]} = \sum_{k=0}^{N-1} h[k]x[n-k] \quad (1.2)$$

$$\mathrm{Y}[k_1] = \mathrm{X}[k_1] + \mathrm{X}[k_2] \quad (1.3)$$

$$\mathrm{Y}[k_2] = (\mathrm{X}[k_1] - \mathrm{X}[k_2])\mathrm{e}^{-\mathrm{j}\omega} \quad (1.4)$$

Cortex®-M4配置了一个32位的MAC，它能在1个周期里完成最高难度为32位乘32位再加64位的运算，或是两个16位乘16位的运算。ARM® Cortex®-M4与乘法或乘积累加有关的指令集如下表1-2，以下皆为定点运算且这些指令皆能在一个周期内完成。

运算	指令意义	指令列表
16×16=32	Signed Multiply (SMUL)	SMULBB, SMULBT, SMULTB, SMULTT
16×16+32=32	Signed Multiply and Accumulate (SMLA)	SMLABB, SMLABT, SMLATB, SMLATT
16×16+64=64	16-bit Signed Multiply with 64-bit Accumulate (SMLAL)	SMLALBB, SMLALBT, SMLALTB, SLMALTT

16×32=32	16-bit by 32-bit signed multiply returning 32 MSB, i.e. word (SMULW)	SMULWB, SMULWT
(16×32)+32=32	Q setting 16-bit by 32-bit signed multiply with 32-bit accumulate (SMLAW)	SMLAWB, SMLAWT
(16×16)+(16×16)=32	Q setting sum of dual 16-bit signed multiply (SMU-AD)	SMUAD, SMUADX
(16×16)-(16×16)=32	Dual 16-bit signed multiply returning difference (SMU-SD)	SMUSD, SMUSDX
(16×16)+(16×16)+32=32	Q setting dual 16-bit signed multiply with single 32-bit accumulator (SMLAD)	SMLAD, SMLADX
(16×16)-(16×16)+32=32	Q setting dual 16-bit signed multiply subtract with 32-bit accumulate (SMLSD)	SMLSD, SMLSDX
(16×16)+(16×16)+64=64	Dual 16-bit signed multiply with single 64-bit accumulator (SMLALD)	SMLALD, SMLALDX
(16×16)-(16×16)+64= 64	Q setting dual 16-bit signed multiply subtract with 64-bit accumulate (SMLSLD)	SMLSLD, SMLSLDX
32×32=32	Multiply	MUL
32+(32×32)=32	Multiply accumulate	MLA
32-(32×32)=32	Multiply subtract	MLS
32×32=64	Long signed/unsigned multiply	SMULL, UMULL
(32×32)+64=64	Long signed/unsigned accumulate	SMLAL, UMLAL
(32×32)+32+32=64	32-bit unsigned multiply with double 32-bit accumulation yielding 64-bit result	UMAAL
32+(32×32) = 32 MSB	32-bit multiply with 32-most-significant-bit accumulate	SMMLA, SMMLAR
32-(32×32) =32 MSB	32-bit multiply with 32-most-significant-bit subtract	SMMLS, SMMLSR
(32×32)=32 MSB	32-bit multiply returning 32-most-significant-bits	SMMUL, SMMULR

表 1-2 ARM® Cortex®-M4 与乘法或乘积累加有关的指令集

1.2.2 单指令多数据流(SIMD, or Single Instruction Multiple Data)

可以提高Cortex®-M4在执行DSP运算时的计算效率，这在Cortex®-M3中并不支持。使用Cortex®-M4的SIMD指令，其数学表示式为(1.5)，可以在一个周期内并行的完成4个8位的乘积累加，或是2个16位的乘积累加。其指令即包含在表1-2灰底处SMUAD, SMUADX , SMLSD, SMLSDX…等。

$$Sum = Sum + (a \times c) + (b \times d) \quad (1.5)$$

1.2.3 浮点运算器(FPU, or floating point unit)

ARM® Cortex®-M4也有符合IEEE-754规范的浮点运算单元，而单精度浮点数在信号处理算法上扮演很重要的角色，下表1-3列举了单精度浮点数的运算与频率数，可以看出浮点数的乘法比较花时间，ARM® Cortex®-M4有提供fused MAC指令以增加精确度。

单精度浮点运算	指令	执行频率
加/减	VADD.F32, VSUB.F32	1
乘法	VMUL.F32	3
乘积累加	VMLA.F32, VMLS.F32, VNMLA.F32, VNMLS.F32	3
Fused MAC	VFMA.F32, VFMS.F32, VFNMA.F32, VFNMS.F32	3
平方根	VSQRT.F32	14
除法	VDIV.F32	14

表 1-3 单精度浮点数的运算与频率数

1.3 CMSIS DSP 函数库介绍

针对Cortex®-M4中的DSP功能，CMSIS-DSP部分提供了超过60种功能的DSP算法，主要分类如下表1-4：

函数运算分类	函数内容
Basic math functions	包含向量加减乘除、反向、绝对值和位移…等等。
Fast Math Functions	包含Cosine和Sine值以查表方式减少时间。

Complex math functions	包含复数乘积、共轭复数、复数绝对值…等等
Filtering Functions	包含卷积、相关系数、最小均方滤波、有限响应滤波和无线响应滤波…等等
Matrix functions	包含矩阵加减乘法运算、反矩阵和转置矩阵…等等
Transform Functions	包含实数和复数的快速傅立叶变换和离散余弦变换
Controller Functions	包含PID控制器、Clarke变换和Park变换
Statistics Functions	包含取最大值、最小值、指数运算、标准偏差、变异数和平方平均数…等等
Support Functions	包含复制向量数值、填满向量值和转换数值格式
Interpolation Functions	包含线性插值与双线性插值

表 1-4 DSP 函数库分类与内容

2 如何使用NuMicro® M4系列 DSP以及CMSIS函数库

使用NuMicro® M4系列 DSP指令功能分为三种方式如下，

1.使用CMSIS DSP的函数库。

2.依照固定格式撰写运算方程式，使系统认知使用DSP指令。

3.撰写汇编语言。

本文讲解重点为1.使用CMSIS DSP的函数库，其原因为函数库功能完善，用户可以直接使用开发容易，另外函数内运算算法皆已优化过，可以有效地降低运行的时间。

2.1 Cortex®-M4 DSP 原始码

新唐NuMicro® M4系列产品的BSP中皆有包含Cortex®-M4 DSP函数库各方程式的原始码，其档案位置\Library\CMSIS\DSP_Lib\Source如下图2-1。\Library\CMSIS\DSP_Lib\Examples部分为ARM®所提供使用DSP的范例程序，供用户参考。

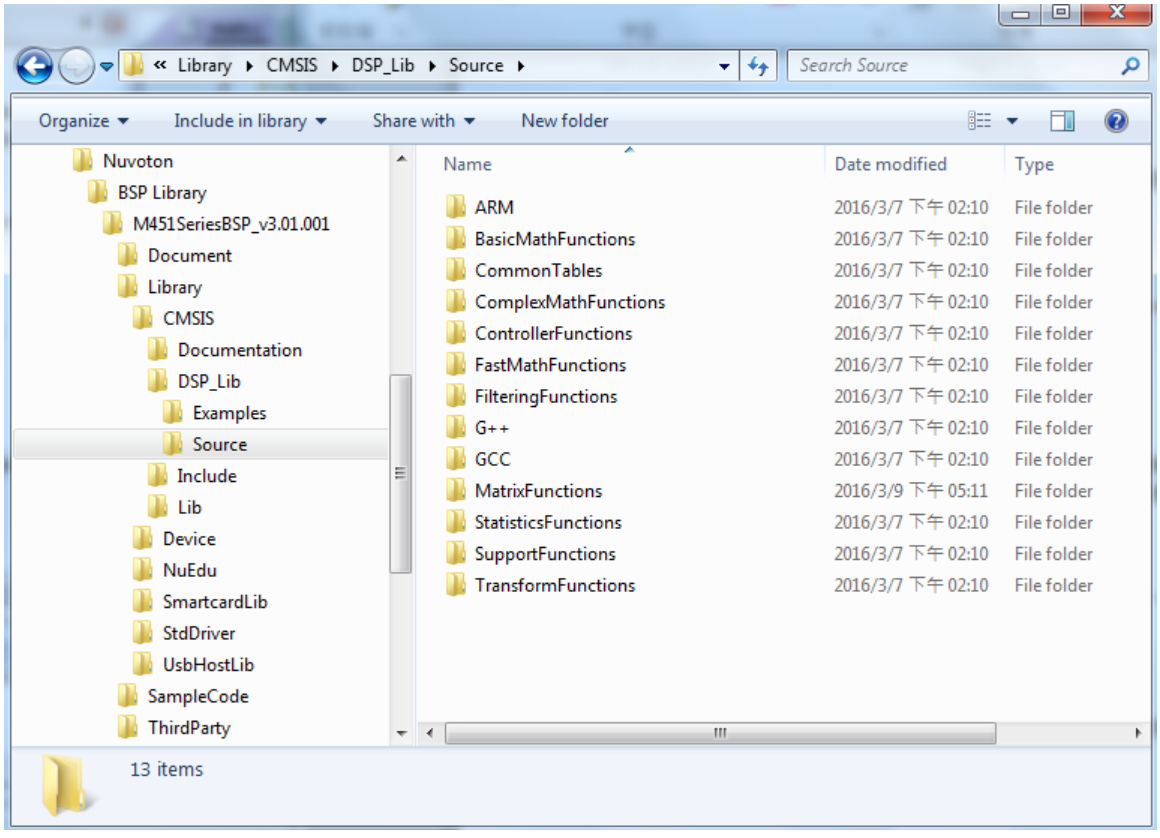
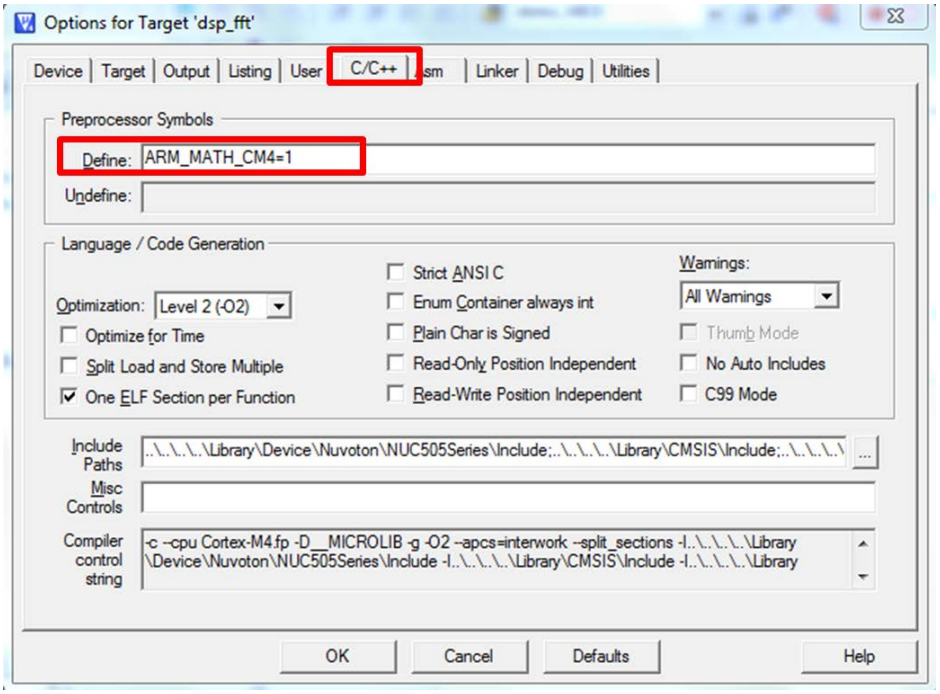


图 2-1 Cortex®-M4 DSP函数库档案位置

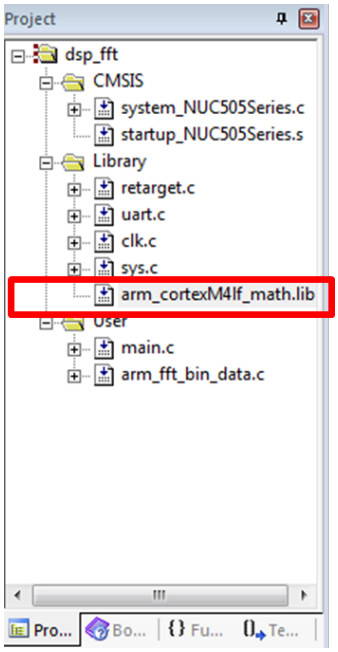
2.2 Keil 使用 NuMicro® M4 系列 DSP 功能

- **开启DSP功能**

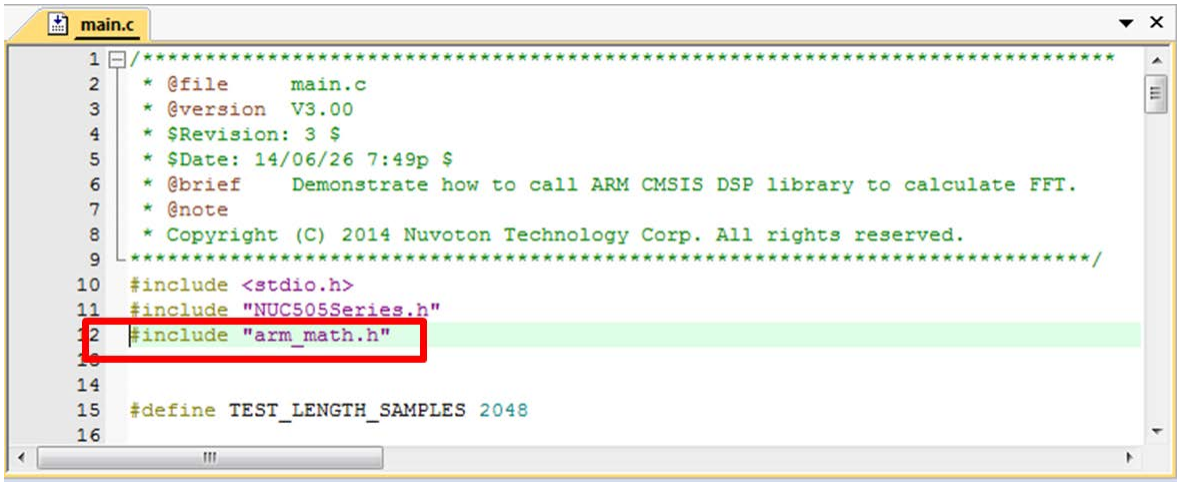
a) 选择 Target Options，切换至 C/C++窗口，Define 输入 ARM_MATH_CM4=1。



b) 在Library部分需要加入arm_cortexM4lf_math.lib，其位置为\Library\CMSIS\Lib\ARM。



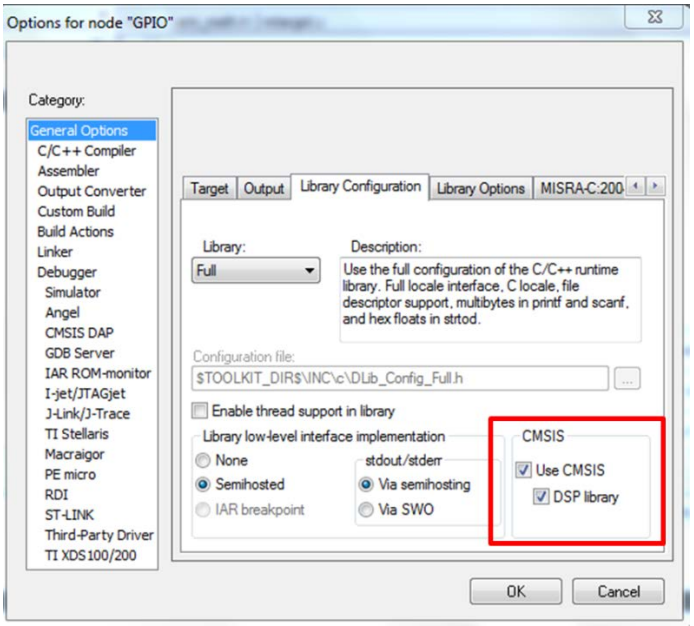
c) 在主程序里面include arm_math.h文件，完成后即可在主程序里面调用DSP函数库。



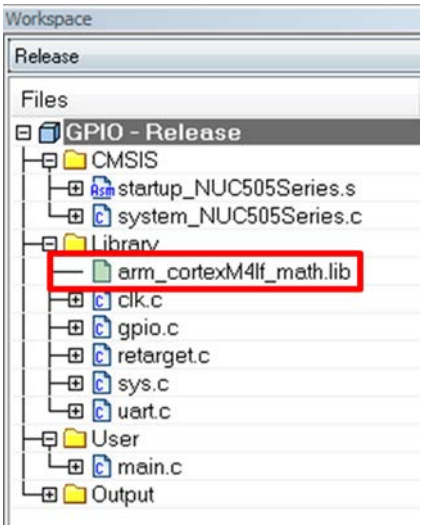
2.3 IAR 使用 NuMicro® M4 系列 DSP 功能

- **开启DSP功能**

a) 进入 IAR 后选择 Options，Category 切换至 General Options，并且在 Library Configuration 窗口勾选 Use CMSIS 和 DSP library。



b) 在Library部分需要加入arm_cortexM4lf_math.lib，其位置为\Library\CMSIS\Lib\ARM。



c) 在主程序里面include arm_math.h文件，完成后即可在主程序里面呼叫DSP函数库。

```
/*****************************************************************************//**
 * @file     main.c
 * @version  V1.00
 * $Date: 14/05/29 1:14p $
 * @brief    NUC505 General Purpose I/O Driver Sample Code
 *           Connect PB.10 and PB.11 to test IO In/Out
 *           Test PB.10 and PB.11 interrupts
 *
 * @note
 * Copyright (C) 2013 Nuvoton Technology Corp. All rights reserved.
 *
 *****************************************************************************/
#include <stdio.h>
#include "NUC505Series.h"
#include "gpio.h"
#include "arm_math.h"
```

NOTE：

arm_cortexM4lf_math.lib即包含Cortex-M4 DSP函数库。

3 NuMicro® M4系列DSP程序范例

本章所介绍新唐NuMicro® M4系列DSP程序范例皆可在新唐官方网站中下载，位置如下
http://www.nuvoton.com/hq/products/microcontrollers/arm-cortex-m4-mcus/Example-Code/?__locale=zh&resourcePage=Y

在Cortex®-M4所提供的DSP函数库中，大部分运算函数有四种数值格式：q31、q15、q7和f32，使用者可以根据需求应用，函数库中也有协助使用者转换不同数值格式的程序可以调用，如下表3-1~表3-4。本文章介绍以及范例程序皆以f32格式呈现。

`arm_float_to_q15 (float32_t *pSrc, q15_t *pDst, uint32_t blockSize)`
`arm_float_to_q31 (float32_t *pSrc, q31_t *pDst, uint32_t blockSize)`
`arm_float_to_q7 (float32_t *pSrc, q7_t *pDst, uint32_t blockSize)`		

参数：	`*pSrc`	[in] float格式数值
	`*pDst`	[out] q15, q31, q7格式数值
	`blockSize`	[in] 样本数
回传值：		无

表 3-1 float 转为 q15, q31, q7 程序设定

`arm_q15_to_float (q15_t *pSrc, float32_t *pDst, uint32_t blockSize)`
`arm_q15_to_q31 (q15_t *pSrc, q31_t *pDst, uint32_t blockSize)`
`arm_q15_to_q7 (q15_t *pSrc, q7_t *pDst, uint32_t blockSize)`		

参数：	`*pSrc`	[in] q15格式数值
	`*pDst`	[out] float, q31,q7格式数值
	`blockSize`	[in] 样本数
回传值：		无

表 3-2 q15 转为 float, q31, q7 程序设定

`arm_q31_to_float (q31_t *pSrc, float32_t *pDst, uint32_t blockSize)`

arm_q31_to_q15 (q31_t *pSrc, q15_t *pDst, uint32_t blockSize) arm_q31_to_q7 (q31_t *pSrc, q7_t *pDst, uint32_t blockSize)		
参数：	*pSrc	[in] q31格式数值
	*pDst	[out] float, q15,q7格式数值
	blockSize	[in] 样本数
回传值：		无

表 3-3 q31 转为 float, q15, q7 程序设定

arm_q7_to_float (q7_t *pSrc, float32_t *pDst, uint32_t blockSize) arm_q7_to_q15 (q7_t *pSrc, q15_t *pDst, uint32_t blockSize) arm_q7_to_q31 (q7_t *pSrc, q31_t *pDst, uint32_t blockSize)		
参数：	*pSrc	[in] q7格式数值
	*pDst	[out] float, q15,q31格式数值
	blockSize	[in] 样本数
回传值：		无

表 3-4 q7 转为 float, q15, q31 程序设定

3.1 基本数学运算

Cortex®-M4 DSP函数库中有许多基本的向量数学运算，包含：

1. 向量绝对值
2. 向量加法
3. 向量减法
4. 向量乘法
5. 向量内积
6. 向量反向
7. 向量缩放
8. 向量偏移

用户可以方便的使用这些函数，来实现自己的数学方程式。各函数程序设定如表3-5~表3-12：

`arm_abs_f32(float32_t *pSrc, float32_t *pDst, uint32_t blockSize )`		
pDst[n] = abs(pSrc[n]), 0 <= n < blockSize		
参数：	`*pSrc`	[in] 向量数据
	`*pDst`	[out] 向量绝对值
	`blockSize`	[in] 向量样本数
回传值：		无

表 3-5 向量绝对值运算程序设定

`arm_add_f32(float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)`		
pDst[n] = pSrcA[n] + pSrcB[n], 0 <= n < blockSize		
参数：	`*pSrcA`	[in] 第一向量数据
	`*pSrcB`	[in] 第二向量数据
	`*pDst`	[out] 计算结果
	`blockSize`	[in] 向量样本数
回传值：		无

表 3-6 向量加法运算程序设定

arm_dot_prod_f32 (float32_t *pSrcA, float32_t *pSrcB, uint32_t blockSize, float32_t *result)		
sum = pSrcA[0]*pSrcB[0] + pSrcA[1]*pSrcB[1] + ... + pSrcA[blockSize-1]*pSrcB[blockSize-1]		
参数：	*pSrcA	[in] 第一向量数据
	*pSrcB	[in] 第二向量数据
	blockSize	[in] 向量样本数
	*result	[out] 计算结果
回传值：		无

表 3-7 向量内积运算程序设定

arm_mult_f32 (float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrcA[n] * pSrcB[n], 0 <= n < blockSize		
参数：	*pSrcA	[in] 第一向量数据
	*pSrcB	[in] 第二向量数据
	*pDst	[out] 计算结果
	blockSize	[in] 向量样本数
回传值：		无

表 3-8 向量乘法运算程序设定

arm_negate_f32 (float32_t *pSrc, float32_t *pDst, uint32_t blockSize)		
pDst[n] = -pSrc[n], 0 <= n < blockSize		
参数：	*pSrc	[in] 欲计算向量数据

	*pDst	[out] 计算结果
	blockSize	[in] 向量样本数
回传值：		无

表 3-9 向量反向程序设定

arm_offset_f32 (float32_t *pSrc, float32_t offset, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrc[n] + offset, 0 <= n < blockSize		
参数：	*pSrc	[in] 欲计算向量数据
	offset	[in] 偏移值
	*pDst	[out] 计算结果
	blockSize	[in] 向量样本数
回传值：		无

表 3-10 向量偏移程序设定

arm_scale_f32 (float32_t *pSrc, float32_t scale, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrc[n] * scale, 0 <= n < blockSize		
参数：	*pSrc	[in] 欲计算向量数据
	scale	[in] 缩放值
	*pDst	[out] 计算结果
	blockSize	[in] 向量样本数
回传值：		无

表 3-11 向量缩放程序设定

arm_sub_f32 (float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrcA[n] - pSrcB[n], 0 <= n < blockSize		
参数：	*pSrcA	[in] 第一向量数据
	*pSrcB	[in] 第二向量数据
	*pDst	[out] 计算结果
	blockSize	[in] 向量样本数
回传值：		无

表 3-12 向量减法运算程序设定

在范例程序中，比较了各个函数使用DSP函数库与使用C函数库运算时间，计算样本数为32，下表3-13比较有无使用DSP函数库运行时间差异，可以看出使用DSP函数库可以大幅缩减计算时间。

```
/*  计算内积 (32个样本数) */
arm_dot_prod_f32(srcA_buf_f32, srcB_buf_f32, blockSize, &dotoutput_f32);

/* 计算绝对值 (32 个样本数) */
arm_abs_f32(srcA_buf_f32, absoutput, blockSize );

/* 计算加法 (32 个样本数) */
arm_add_f32(srcA_buf_f32, srcB_buf_f32, addoutput, blockSize);

/* 计算减法 (32 个样本数) */
arm_sub_f32 (srcA_buf_f32, srcB_buf_f32, suboutput, blockSize);

/* 计算乘法 (32 个样本数) */
arm_mult_f32 (srcA_buf_f32, srcB_buf_f32, multoutput, blockSize);

/* 计算缩放 (32 个样本数) */
arm_scale_f32 (srcA_buf_f32, 10, scaleoutput, blockSize);

/* 计算偏移 (32 个样本数) */
arm_offset_f32 (srcA_buf_f32, 10, offsetoutput, blockSize);
```

```
/* 计算反向 (32 个样本数) */
arm_negate_f32 (srcA_buf_f32, negoutput, blockSize);
```

Function(with 32 sample)	DSP(Time unit)	CPU(Time unit)	CPU/DSP
Dot Product	52	105	2.02
Absolute Value	34	78	2.29
Addition	46	100	2.17
Subtraction	47	100	2.13
Multiplication	47	100	2.13
Scale	36	78	2.167
Offset	37	79	2.14
Negate	37	79	2.14

表 3-13 基本运算有无使用 DSP 函数库运行时间比较表

3.2 三角函数运算

Cortex®-M4 DSP函数库，提供了计算Sine，Cosine的功能，以查表的方式减少计算时间，其方程式定义为下表3-14、表3-15和表3-16：

arm_cos_f32(float32_t x)		
参数：	x	[in]欲计算的弧度
回传值：		cos(x)

表 3-14 使用弧度计算Cosine程序设定

arm_sin_f32(float32_t x)		
参数：	x	[in]欲计算的弧度
回传值：		sin(x)

表 3-15 使用弧度计算Sine程序设定

arm_sin_cos_f32(float32_t theta, float32_t *pSinVal, float32_t *pCosVal)		
参数：	theta	[in] 欲计算的角度
	*pSinVal	[out] sine值输出
	*pCosVal	[out] cosine值输出
回传值：		无

表 3-16 使用角度计算Sine与Cosine程序设定

在范例程序中，比较了使用DSP函数库与使用C函数库运算时间，计算样本数为32，下表3-17比较有无使用DSP函数库运行时间差异，可以看出使用DSP函数库可以大幅缩减计算时间。

```
    /*计算sin, cos (32个样本数)*/
    for(i=0; i< blockSize; i++)
  {
    /* 输入为弧度 */
```

```
        cosOutput[i] = arm_cos_f32(testInput_f32[i]);
        sinOutput[i] = arm_sin_f32(testInput_f32[i]);

    /* 输入为角度 */
    arm_sin_cos_f32(testInput_f32[i], &sinOutput1[i], &cosOutput1[i]);
    }
```

Function(with 32 sample)	DSP(Time unit)	CPU(Time unit)	CPU/DSP
Sine	551	23251	42.2
Cosine	551	23720	43.05
Sine and Cosine	413	46561	112.74

表 3-17 三角函数运算有无使用DSP函数库运行时间比较表

3.3 线性插值

假设目前已知坐标*(x0,y0)*和*(x1,y1)*要得到在线区间内某一位置x在直线上的值。根据图3-1所示，我们得到

$$\frac{y-y0}{x-x0}=\frac{y1-y0}{x1-x0}$$

由于x值已知，所以我们可以从公式得到y的值。

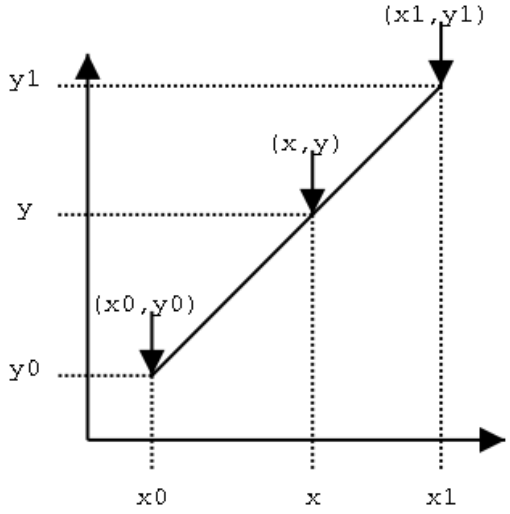


图 3-1 线性插值示意图

使用Cortex®-M4 DSP函数库线性插值功能首先定义x0的样本数、起始值x0和x值与值间隔，再来输入y值数据，程序设定如表3-18与表3-19，完成浮点数插值结构设定之后，再输入x值执行程序，即可得到线性插值所得y值。

`arm_linear_interp_instance_f32 S = {uint32_t nValues, float32_t x0, float32_t xSpacing, float32_t *pYData}`		
参数：	`nValues`	x值的样本数
	`x0`	x起始值
	`xSpacing`	x值与值之间隔
	`*pYData`	y值数据
回传值：		无

表 3-18 线性插值结构设定

arm_linear_interp_f32(arm_linear_interp_instance_f32 *S, float32_t x)		
参数：	*S	[in, out]浮点数插值结构
	x	[in] 输入x值
回传值：		线性插值y结果

表 3-19 线性插值程序设定

程序范例流程如下，且在下表3-20比较有无使用DSP运算时间差异：

```
/*  设定结构变量S，定义x样本个数为2，其起始为0数值间隔为10，即x0=0，x1=10，且定义y值为
arm_linear_interep_table */
arm_linear_interp_instance_f32 S = {2, 0, 10, (float32_t *)&arm_linear_interep_table[0]};

/*  循环放入10个样本进行线性插值运算，并且输出结果到testLinIntOutput[]  */
for(i=0; i< TEST_LENGTH_SAMPLES; i++)
{
     testLinIntOutput[i] = arm_linear_interp_f32(&S, testInputSin_f32[i]);
}
```

Function(with 10 sample)	DSP(Time unit)	CPU(Time unit)	CPU/DSP
Linear Interpolation	138	155	1.12

表 3-20 线性插值有无使用DSP运算时间比较表

3.4 统计运算

Cortex®-M4 DSP函数库中也有关于统计方面的运算，包含：

1. 取最大值
2. 取最小值
3. 取平均值
4. 平方平均数
5. 标准偏差
6. 变异数

在此我们介绍两种较常用到的平方平均数以及标准偏差来做介绍。

平方平均数运算(Quadratic mean)，又称均方根(RMS, Root Mean Square)在统计学中很常使用到，其数学表示式为下，程序设定如表3-21：

$$M = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$$

标准偏差(SD, Standard Deviation)，在机率统计中最常使用作为测量一组数值的离散程度之用。其数学表示式为下，其中u为x的平均值，程序设定如表3-22：

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - u)^2}$$

`arm_rms_f32 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)`		
参数：	`*pSrc`	[in] 欲计算矩阵
	`blockSize`	[in] 矩阵样本数
	`*pResult`	[out] 计算结果
回传值：		无

表 3-21 均方根程序设定

`arm_std_f32 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)`		
参数：	`*pSrc`	[in] 欲计算数值

	blockSize	[in] 样本数
	*pResult	[out] 计算结果
回传值：		无

表 3-22 标准偏差程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-23，其程序如下：

```
/* 计算均方根 (32个样本数) */
arm_rms_f32 (testInput_f32, blockSize, &testoutput_f32);
```

```
/* 计算标准偏差 (32个样本数) */
arm_std_f32(testMarks_f32, blockSize, &std);
```

Function	DSP(Time unit)	CPU(Time unit)	CPU/DSP
RMS(with 32 sample)	45	413	9.18
Standard Deviation (with 32 sample)	116	1024	8.83

表 3-23 均方根与标准偏差有无使用DSP运算时间比较表

3.5 矩阵运算

Cortex®-M4 DSP函数库有许多关于矩阵的运算包含：

1. 矩阵加法运算
2. 矩阵减法运算
3. 矩阵乘法运算
4. 逆矩阵
5. 矩阵缩放
6. 转置矩阵

用户须先用arm_matrix_instance_f32宣告矩阵变量，再来对使用到的矩阵初始化设定，定义矩阵直行与横列个数以及矩阵数值到arm_mat_init_f32()。完成设定后即可对各矩阵进行运算，其程序设定如表3-24~表3-30。

arm_mat_init_f32(arm_matrix_instance_f32 *S, uint16_t nRows, uint16_t nColumns, float32_t *pData)		
参数：	*S	[in, out] 指向所宣告的浮点数矩阵结构
	nRows	[in] 矩阵直行个数
	nColumns	[in] 矩阵横列个数
	*pData	[in] 矩阵数据
回传值：		无

表 3-24 矩阵初始化设定

arm_mat_add_f32(const arm_matrix_instance_f32 *pSrcA, const arm_matrix_instance_f32 *pSrcB, arm_matrix_instance_f32 *pDst)		
参数：	*pSrcA	[in] 输入被加数矩阵
	*pSrcB	[in] 输入加数矩阵
	*pDst	[out] 输出结果矩阵
回传值：		成功显示ARM_MATH_SUCCESS, 矩阵不相合则显示 ARM_MATH_SIZE_MISMATCH

表 3-25 矩阵相加程序设定

arm_mat_sub_f32(const arm_matrix_instance_f32 *pSrcA, const arm_matrix_instance_f32 *pSrcB, arm_matrix_instance_f32 *pDst)		
参数：	*pSrcA	[in] 输入被减数矩阵
	*pSrcB	[in] 输入减数矩阵
	*pDst	[out] 输出结果矩阵
回传值：		成功显示ARM_MATH_SUCCESS, 矩阵不相合则显示 ARM_MATH_SIZE_MISMATCH

表 3-26 矩阵相减程序设定

arm_mat_mult_f32(const arm_matrix_instance_f32 *pSrcA, const arm_matrix_instance_f32 *pSrcB, arm_matrix_instance_f32 *pDst)		
参数：	*pSrcA	[in] 输入被乘数矩阵
	*pSrcB	[in] 输入乘数矩阵
	*pDst	[out] 输出结果矩阵
回传值：		成功显示ARM_MATH_SUCCESS, 矩阵不相合则显示 ARM_MATH_SIZE_MISMATCH

表 3-27 矩阵相乘程序设定

arm_mat_inverse_f32(const arm_matrix_instance_f32 *pSrc, arm_matrix_instance_f32 *pDst)		
参数：	*pSrc	[in] 输入欲计算矩阵
	*pDst	[out] 输出逆矩阵
回传值：		成功显示ARM_MATH_SUCCESS, 矩阵不为方阵则显示ARM_MATH_SIZE_MISMATCH, 若矩阵不可逆显示ARM_MATH_SINGULAR

表 3-28 反矩阵程序设定

arm_mat_scale_f32(const arm_matrix_instance_f32 *pSrc, float32_t scale, arm_matrix_instance_f32 *pDst)		
参数：	*pSrc	[in] 输入欲计算矩阵
	scale	[in] 缩放参数
	*pDst	[out] 输出结果矩阵
回传值：		成功显示ARM_MATH_SUCCESS

表 3-29 缩放矩阵程序设定

arm_mat_trans_f32(const arm_matrix_instance_f32 *pSrc, arm_matrix_instance_f32 *pDst)		
参数：	*pSrc	[in] 输入欲计算矩阵
	*pDst	[out] 输出转置后矩阵
回传值：		成功显示ARM_MATH_SUCCESS

表 3-30 转置矩阵程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-31，其程序如下：

```
    arm_matrix_instance_f32 A; /* 宣告矩阵A结构参数 */
    arm_matrix_instance_f32 AT;/* 宣告矩阵AT(A transpose)结构参数 */
    arm_matrix_instance_f32 ATMA;    /* 宣告矩阵ATMA(AT multiply with A)结构参数*/
    arm_matrix_instance_f32 ATMAI;   /* 宣告矩阵ATMAI(Inverse of ATMA)结构参数 */

    /* 初始化设定5*5矩阵A, 数据为A_f32 */
    arm_mat_init_f32(&A, 5, 5, A_f32);
    /* 初始化设定5*5矩阵AT, 数据为AT_f32 */
    arm_mat_init_f32(&AT, 5, 5, AT_f32);
    /* 矩阵A进行转置运算存入矩阵AT */
    arm_mat_trans_f32(&A, &AT);
    /* 初始化设定5*5矩阵ATMA, 数据为ATMA_f32 */
    arm_mat_init_f32(&ATMA, 5, 5, ATMA_f32);
    /* AT×A 计算结果存入矩阵ATMA */
    arm_mat_mult_f32(&AT, &A, &ATMA);
    /* 初始化设定5*5矩阵ATMAI, 数据为ATMAI_f32 */
```

```
    arm_mat_init_f32(&ATMAI, 5, 5, ATMAI_f32);
    /* 矩阵A进行反矩阵运算存入矩阵ATMAI */
    arm_mat_inverse_f32(&A, &ATMAI);
```

Function(with 5*5 sample)	DSP(Time unit)	CPU(Time unit)	CPU/DSP
Transpose	47	74	1.57
Multiply	288	483	1.67
inverse	725	7174	9.9

表 3-31 矩阵运算有无使用DSP运算时间比较表

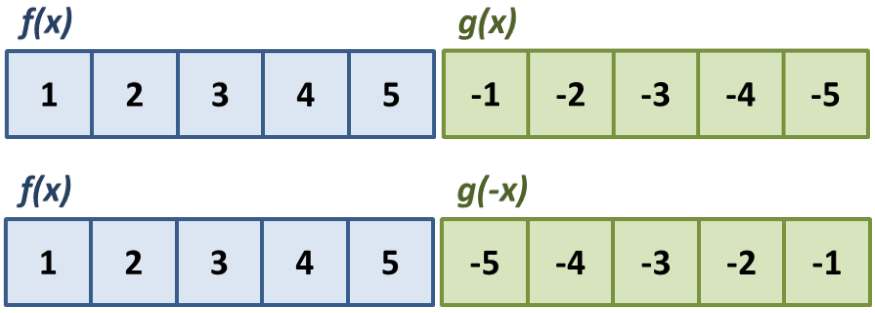
3.6 卷积(Convolution)

卷积又称折积、迭积或旋积，是通过两个函数*f(x)*和*g(x)*生成第三个函数*h(x)*的一种数学算法，记为*h(x)=(f*g)(x)*，它是其中一个函数翻转并平移后与另一个函数的乘积的积分，是一个对平移量的函数，也就是：

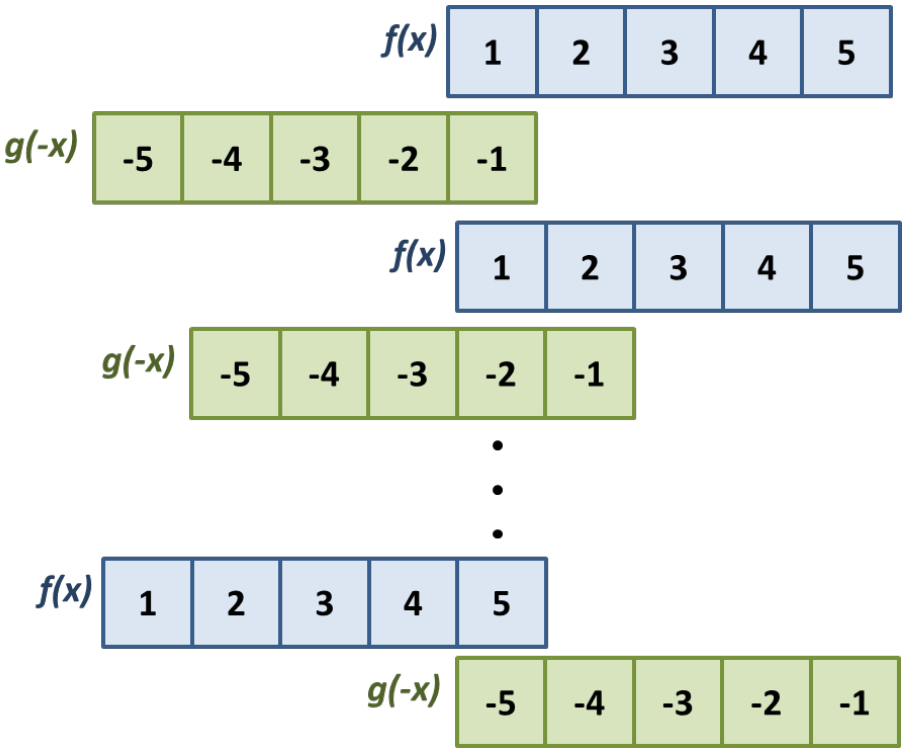
$$(f * g)(t) \stackrel{\text{def}}{=} \int f(\tau)g(t-\tau)d\tau$$

以下范例图解其计算方法与过程：

1.有两函数*f(x)={1,2,3,4,5}*与*g(x)={-1,-2,-3,-4,-5}*，并且*g(x)*翻转为*g(-x)*



2.平移两函数*f(x)*与*g(-x)*相乘生成第三个函数*h(x)*，其计算方式如下图：



h(1)=f(1)×g(1)= -1

h(2)=f(2)×g(1)+ f(1)×g(2)= -4

h(3)=f(3)×g(1)+ f(2)×g(2)+ f(1)×g(3)= -10

以此类推，因此得*h(x)={-1,-4,-10,-20,-35,-44,-46,-40,-25}*

在DSP函数库内可以调用`arm_conv_f32()`来进行卷积运算，其定义如下表3-32：

`arm_conv_f32(float32_t *pSrcA, uint32_t srcALen, float32_t *pSrcB, uint32_t srcBLen, float32_t *pDst)`		
参数：	`*pSrcA`	[in] 第一个函数f(x)
	`srcALen`	[in] f(x)函数样本数
	`*pSrcB`	[in] 第二个函数g(x)
	`srcBLen`	[in] g(x)函数样本数
	`*pDst`	计算输出结果h(x)，其样本数为 srcALen+ srcBLen-1
回传值：		无

表 3-32 卷积(convolution)程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-33，其程序如下：

```
/* 计算卷积 */
arm_conv_f32(testInput_f32_1kHz_15kHz, TEST_LENGTH_SAMPLES, firCoeffs32, NUM_TAPS,
conoutput);
```

Function	DSP(Time unit)	CPU(Time unit)	CPU/DSP
Convolution	7442	27703	3.72

表 3-33 卷积有无使用DSP运算时间比较表

3.7 有限脉冲响应(Finite impulse response)

有限脉冲响应（Finite impulse response，缩写 FIR）滤波器是数字滤波器的一种，简称FIR数字滤波器，其运算方法即是为卷积定理。有限脉冲响应滤波器是一线性系统，输入信号，*x(0)*、*x(1)⋯x(n)*，经过该系统后的输出信号，*y(n)*可表示为：

$$y(n) = h_0 x(n) + h_1 x(n-1) + \cdots + h_N x(n-N)$$

其中，h_0 、$h_1 \ldots h_N$是滤波器的脉冲响应，通常称为滤波器的系数。N是滤波器的阶数。上式也可表示为：

$$y(n) = \sum_{k=0}^{N} h_k \, x(n-k)$$

DSP函数库中首先使用`arm_fir_init_f32()`进行初始化设定，输入滤波器的脉冲响应系数h以及样本数，再把信号x输入`arm_fir_f32()`中执行，即可得到信号x经过FIR滤波器处理后的结果。以下表3-34与表3-35说明关于FIR运算相关程序设定：

`arm_fir_init_f32(arm_fir_instance_f32 *S, uint16_t numTaps, float32_t *pCoeffs, float32_t *pState, uint32_t blockSize)`		
参数：	`*S`	[in,out] 指向FIR滤波器结构
	`numTaps`	[in] 滤波器系数h样本数
	`*pCoeffs`	[in] 滤波器系数h矩阵
	`*pState`	[in] 状态矩阵
	`blockSize`	[in] 欲计算的样本数
回传值：		无

表 3-34 有限脉冲响应(FIR)初始化设定

`arm_fir_f32(const arm_fir_instance_f32 *S, float32_t *pSrc, float32_t *pDst, uint32_t blockSize)`		
参数：	`*S`	[in] 指向FIR滤波器结构
	`*pSrc`	[in] 输入欲计算的值
	`*pDst`	[out] 经过FIR计算后输出结果

	blockSize	[in] 欲计算的样本数
回传值：		无

表 3-35 有限脉冲响应(FIR)程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-36，其程序如下：

```
    /*宣告fir结构参数*/
    arm_fir_instance_f32 S;
    /*fir初始化设定*/
    arm_fir_init_f32(&S, NUM_TAPS, (float32_t *)&firCoeffs32[0], &firStateF32[0],
blockSize);
    /* 设定循环执行几次*/
    for(k=0; k < numBlocks; k++)
    {    /* 一次执行多少样本数fir运算 */
    arm_fir_f32(&S, inputF32 + (k * blockSize), outputF32 + (k * blockSize),blockSize);
    }
```

Function	DSP(Time unit)	CPU(Time unit)	CPU/DSP
FIR	7496	27703	3.7

表 3-36 有限脉冲响应有无使用 DSP 运算时间比较表

在范例程序中，有一信号为 1k 赫兹弦波带有 15k 赫兹噪声，如图 3-2，输入 FIR 滤波器做计算，此滤波器为低通滤波器，会把 6k 赫兹以上信号给滤除。因此信号经过 FIR 滤波器后，会输出 1k 赫兹信号，并把 15k 赫兹噪声给消除，如图 3-3。

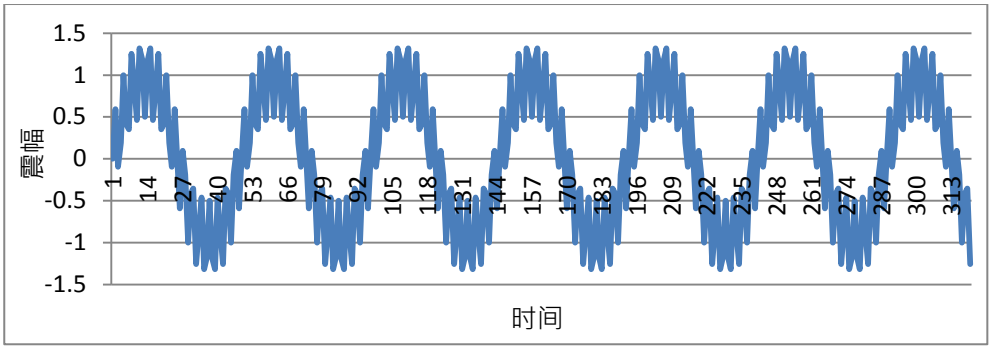


图 3-2 输入信号为 1k 赫兹弦波带有 15k 赫兹噪声

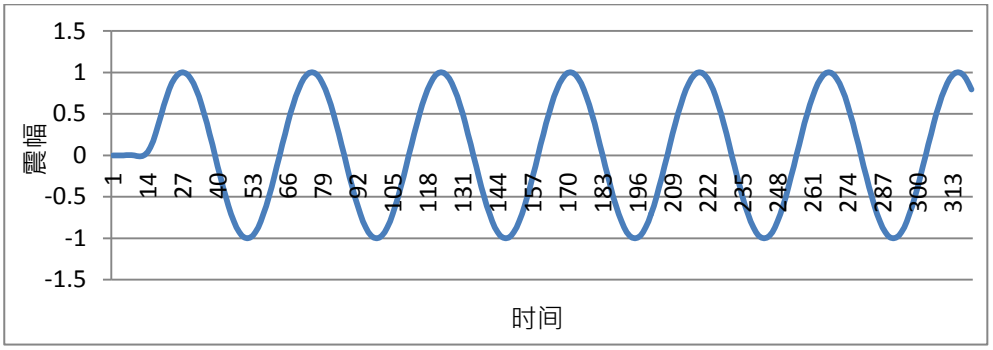


图 3-3 经过 FIR 滤波器后输出信号为 1k 赫兹弦波

3.8 PID 控制器(比例-积分-微分控制器)

PID控制器(比例-积分-微分控制器)，由比例单元P、积分单元I和微分单元D组成，分别对应目前误差、过去累计误差及未来误差。透过Kp，Ki和Kd三个参数以控制回路的回馈响应。PID控制器是一个在工业控制应用中常见的反馈回路部件，这个控制器把收集到的数据和一个目标值进行比较，然后把这个差值经过PID控制器运算后成为新的输入值，这个新的输入值的目的是可以让系统的数据达到或者保持在目标值。PID控制器可以根据历史数据和差别的出现率来调整输入值，使系统更加准确而稳定。

新唐在范例程序演示如何使用PID控制器运算。首先用arm_pid_instance_f32定义矩阵变量，再来对PID初始化设定。目标参考值为in输入arm_pid_f32()进行运算，得到进行PID运算后所输出值，再把第一次输出值与目标参考值相减为误差，将其误差输入PID进行第二次运算，以此进行下去输出值即可与目标参考值相近，其执行流程如图3-4，程序设定为表3-37与表3-38。

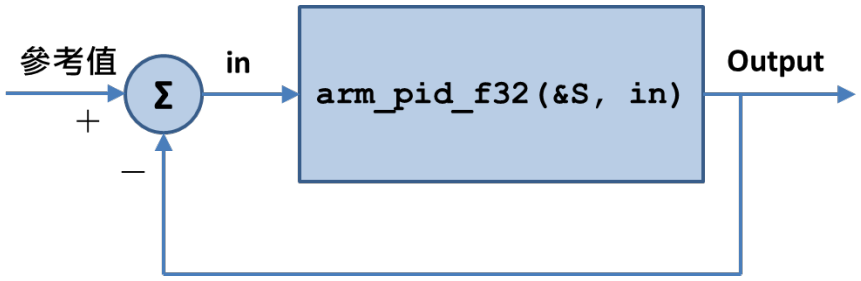


图 3-4 PID 控制执行流程图

arm_pid_init_f32(arm_pid_instance_f32 *S, int32_t resetStateFlag)		
参数：	*S	[in, out] 所宣告的PID结构参数
	resetStateFlag	[in] 0为不复位状态，1为复位状态
回传值：		无

表 3-37 PID 初始化设定

arm_pid_f32(arm_pid_instance_f32 *S, float32_t in)		
参数：	*S	[in] 所宣告的PID结构参数
	in	[out] 输入数值进行运算(参考值减去回传值)
回传值：		经过PID后输出数值

表 3-38 PID控制器程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-39，其程序如下：

```
/*宣告结构变量*/
 arm_pid_instance_f32 PIDS;
/*宣告Kp, Ki, Kd值*/
PIDS.Kp=0.4;
PIDS.Ki=0.4;
PIDS.Kd=0;
/*宣告目标值*/
target=500;
/*宣告起始值*/
ival=0;
/*宣告误差*/
ee=target-ival;
/*PID初始化设定*/
arm_pid_init_f32(&PIDS,0);
for(i=1;i<100;i++)
{
 /*输入前笔误差值(ee)进入运算得到现在值(output)*/
 output[i]=arm_pid_f32(&PIDS,ee);
 /*计算现在误差值*/
 ee=target-output[i-1];
 }
```

Function	DSP(Time unit)	CPU(Time unit)	CPU/DSP
PID	939	1702	1.81

表 3-39 PID有无使用DSP运算时间比较表

在范例程序中为马达转速从0开始经过PID控制后而达到500，而Kp、Ki、Kd值分别为0.4、0.4、0，经过100次运算后结果如图3-5。

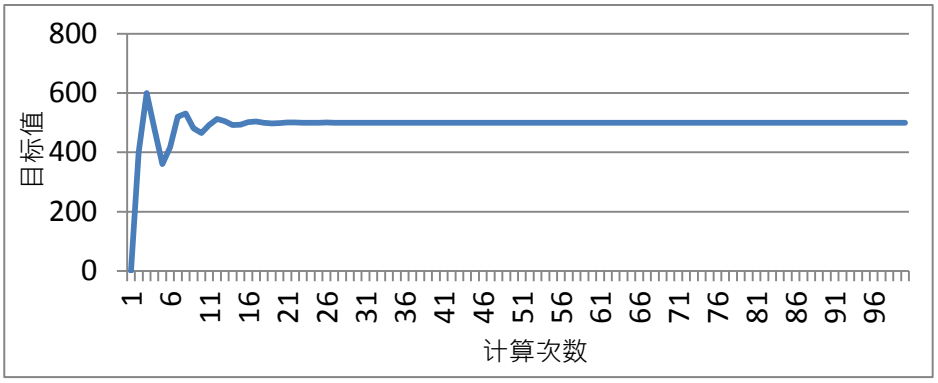


图 3-5 PID 控制运算范例程序计算结果

3.9 快速傅立叶变换(FFT, or Fast Fourier Transform)

人是活在时间的世界，所以一般我们在看事情的现象，多是在时间域(Time Domain)来观察。然而有许多现象在时间域是观察不到的，必须透过特定的转换方法，才得以观察出有意义的数据。数学家傅立叶推导出一个变换公式，将原本属于时间域的数据转换成频域(Frequency Domain)，此一变换对于信号的观察与改良提供了极大的帮助。信号在经过FFT转换后，我们可以明显看出此信号的主要频率位置，以及与其他频率强度的相对关系。

快速傅立叶变换可以将时域中的信号分解成频域，如下图3-6为一类方波*f*，其为许多弦波组合而成，弦波组成其成分为图3-7，再经傅立叶变换后可以得到组成这一类方波所有弦波的频率以及强度如图3-8，因此一个信号经过快速傅立叶变换后，可以很明显地了解其组成成分，并且根据结果来做信号处理，如滤波等。

图 3-6 类方波 f

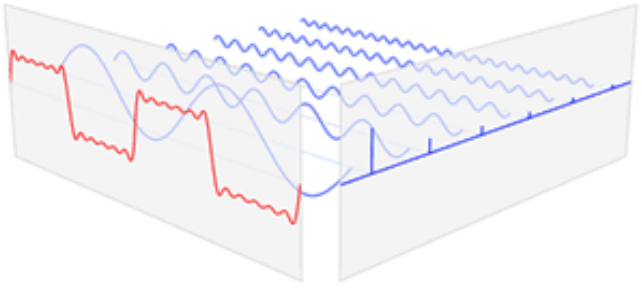


图 3-7 类方波为许多弦波组合而成

图 3-8 组合成类方波各弦波频率以及强度

Cortex®-M4 DSP函数库有提供计算FFT功能，用户输入数据即可得到最大主频强度以及位置。在此提供范例程序给用户参考，所使用到方程式定义方式如下表3-40~表3-43。

`arm_cfft_radix4_init_f32(arm_cfft_radix4_instance_f32 *S, uint16_t fftLen, uint8_t ifftFlag, uint8_t bitReverseFlag)`		
参数:	`*S`	[in, out] 复数FFT结构
	`fftLen`	[in] FFT计算样本数(须为2的次方)
	`ifftFlag`	[in] 执行正或反FFT运算(0为正，1为反)
	`bitReverseFlag`	[in] 是否位反转(1为正，0为反)。
回传值:		若初始化成功则会回传 ARM_MATH_SUCCESS 若样本数不对则会显示 ARM_MATH_ARGUMENT_ERROR

表 3-40 复数快速傅立叶变换初始设定

`arm_cfft_radix4_f32(const arm_cfft_radix4_instance_f32 *S, float32_t * pSrc )`		
参数:	`*S`	[in] 复数FFT结构

	*pSrc	[in, out] 欲计算的数值数组，并且回传计算结果到原数组，其数组格式为(实数，复数，实数…)
回传值：		无

表 3-41 复数快速傅立叶变换运算程序设定

arm_cmplx_mag_f32(float32_t *pSrc, float32_t *pDst, uint32_t numSamples)		
参数：	*pSrc	[in] 欲计算的复数数组
	*pDst	[out] 计算后实数数组
	numSamples	[in] 输入复数数组的样本数
回传值：		无

表 3-42 取复数绝对值程序设定

arm_max_f32(float32_t *pSrc, uint32_t blockSize, float32_t *pResult, uint32_t *pIndex)		
参数：	*pSrc	[in] 欲计算的数组
	blockSize	[in] 需计算数值的样本数
	*pResult	[out] 所得最大值
	*pIndex	[out] 所得最大值的位置
回传值：		无

表 3-43 取最大值程序设定

新唐提供范例程序供用户参考，并且比较有无使用DSP其运算时间如表3-44，其程序如下：

```
    /* 宣告FFT结构变量 */
    arm_cfft_radix4_instance_f32 S;
    float32_t maxValue;
```

```
/* 初始化设定FFT/IFFT */
arm_cfft_radix4_init_f32(&S, fftSize, ifftFlag, doBitReverse);
/* 执行FFT运算，将结果回传到原输入数组中 */
arm_cfft_radix4_f32(&S, testInput_f32_10khz);
/* 取各计算结果数值的绝对值来得其各个频率强度 */
arm_cmplx_mag_f32(testInput_f32_10khz, testOutput, fftSize);
/* 取最大值为主频位置及数值 */
arm_max_f32(testOutput, fftSize, &maxValue, &testIndex);
```

Function(1024 sample)	DSP(Time unit)	CPU(Time unit)	CPU/DSP
FFT	18027	233893	12.98

表 3-44 FFT有无使用DSP运算时间比较表

范例程序中，有一信号如图3-9，经过快速傅立叶传换后，取得各频率的复数值，再对复数值取绝对值，即可得到各频率强度分布如图3-10。因此杂乱信号经过傅立叶变换后即可看出其频率强度分布，并找出此输入信号为10k赫兹带有噪声的信号。

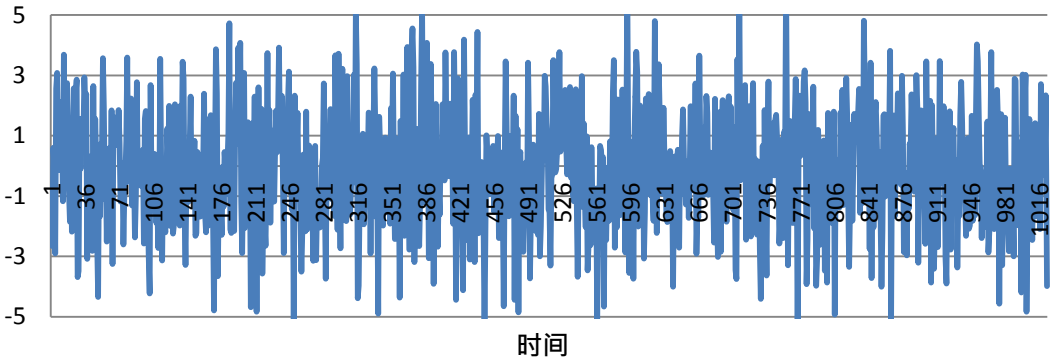


图 3-9 范例程序输入信号时域图

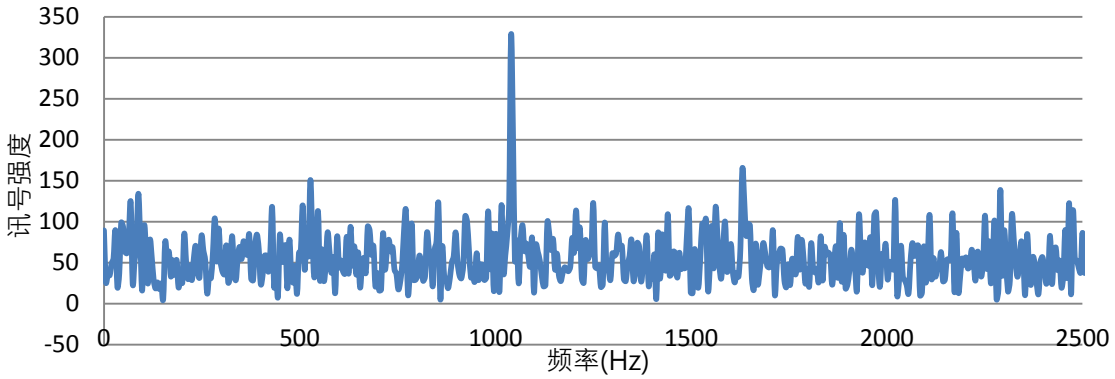


图 3-10 信号经 FFT 转换后的频域图

4 结论

本文介绍NuMicro® M4系列的DSP功能以微控制器为主，加入了数字信号处理中常用的乘积累加功能以及单指令多数据流功能，同时拥有丰富的周边支持与输出入脚位，并且支持浮点数运算，让使用者在应用上增快程序的运行，而且开发更加方便。而CMSIS DSP函数库提供了超过60种数学公式，其功能完善且函数内运算算法皆已优化过，可以有效地降低运行的时间，使用者可以很方便的直接调用来开发程序，提升自己的算法指令周期来达成应用需求。

本文取20几种常用的数学方程式做运算时间比较如表4-1，未注明的方程式皆以32个样本数来做运算。总共分为三种运算方式，第一为使用Cortex®-M4并且开启DSP功能以及调用CMSIS函数库做运算，第二为使用Cortex®-M4但未使用DSP功能以及函数库，第三种为使用Cortex®-M3架构无DSP功能以及不支持浮点运算器。后面两行比较了使用Cortex®-M4 DSP功能对比不使用DSP功能以及Cortex®-M3可以缩短多少倍的运算时间，从表中可以看出最大缩短时间的倍数可以达到100倍之多，以此证明使用NuMicro® M4系列DSP功能可以带给用户程序更快的执行速度。

Function (with 32 sample)	DSP (Time unit)	CPU (Time unit)	M3 without FPU	CPU/DSP	M3/DSP
Dot Product	52	105	614	2.02	11.8
Absolute Value	34	78	67	2.29	1.97
Addition	46	100	382	2.17	8.3
Subtraction	47	100	406	2.13	8.64
Multiplication	47	100	327	2.13	6.96
Scale	36	78	330	2.167	9.167
Offset	37	79	327	2.14	8.84
Negate	37	79	67	2.14	1.81
Sine	551	23251	22165	42.2	40.23
Cosine	551	23720	22622	43.05	41.06
Sine and Cosine	413	46561	44742	112.74	108.33
Linear Interpolation (with 10 sample)	138	155	1479	1.12	10.72

RMS (with 32 sample)	45	413	951	9.18	21.13
Standard Deviation (with 80 sample)	116	1024	6676	8.83	57.55
Matrix Transpose(5*5)	47	74	65	1.57	1.38
Matrix Multiply(5*5)	288	483	1335	1.67	4.64
Matrix inverse(5*5)	725	7174	7264	9.9	10.02
Convolution (320*29 sample)	7442	27703	171283	3.72	23.02
FIR(320*29 sample)	7496	27703	171283	3.7	22.85
PID (Calculate 100 times)	556	1014	7139	1.82	12.84
FFT (with 1024 sample)	18027	233893	237829	12.98	13.2

表 4-1 Cortex®-M4有无使用DSP以及与Cortex®-M3运算时间比较表

Revision History

Date	Revision	Description
2016.08.16	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.