Using Keil M4 DSP Library

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	The CMSIS-DSP library is a suite of common signal processing and mathematical functions that have been optimized for the NuMicro® M4 microcontroller. This document describes how to use the basic math functions from the Keil M4 DSP library to help users get DSP high performance advantages.
BSP Version	M480 Series BSP CMSIS v3.03.000
Hardware	NuMaker-ETM-M487 v1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

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

www.nuvoton.com

1 Function Description

1.1 Introduction

Users use all the functions of the Keil M4 DSP library to implement basic math functions, such as vector dot product, vector multiply, vector addition.
There are two methods to implement vector operations in the example. One is using Keil M4 DSP library perform vector operations and the others is using C functions. Compare with C functions, DSP library has the faster execution speed.

1.2 Principle

The most common numerical format used for DSP operations are q31, q15, q7 and f32. Users can choose the appropriate numerical format according to the application.
The Basic math functions in the library, such as Vector Dot Product, Vector Absolute Value, Vector Addition, Vector Subtraction, Vector Multiplication, Vector Scale, Vector Offset and Vector Negate. Please refer to Table 1~Table 8 for each function setting.

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]		
Parameters	*pSrcA	[in] points to the first input vector
	*pSrcB	[in] points to the second input vector
	blockSize	[in] number of samples in each vector
	*result	[out] output result returned here
Returns		none

Table 1 Vector Dot Product

arm_abs_f32(float32_t *pSrc, float32_t *pDst, uint32_t blockSize)		
pDst[n] = abs(pSrc[n]), 0 <= n < blockSize		
Parameters	*pSrc	[in] points to the input buffer
	*pDst	[out] points to the output buffer
	blockSize	[in] number of samples in each vector
Returns		none

Table 2 Vector Absolute Value

arm_add_f32(float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrcA[n] + pSrcB[n], 0 <= n < blockSize		
Parameters	*pSrcA	[in] points to the first input vector
	*pSrcB	[in] points to the second input vector
	*pDst	[out] points to the output vector

	blockSize	[in] number of samples in each vector
Returns		none

Table 3 Vector Addition

arm_sub_f32 (float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrcA[n] - pSrcB[n], 0 <= n < blockSize		
Parameters	*pSrcA	[in] points to the first input vector
	*pSrcB	[in] points to the second input vector
	*pDst	[out] points to the output vector
	blockSize	[in] number of samples in each vector
Returns		none

Table 4 Vector Subtraction

arm_mult_f32 (float32_t *pSrcA, float32_t *pSrcB, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrcA[n] * pSrcB[n], 0 <= n < blockSize		
Parameters	*pSrcA	[in] points to the first input vector
	*pSrcB	[in] points to the second input vector
	*pDst	[out] points to the output vector
	blockSize	[in] number of samples in each vector
Returns		none

Table 5 Vector Multiplication

arm_scale_f32 (float32_t *pSrc, float32_t scale, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrc[n] * scale, 0 <= n < blockSize		
Parameters	*pSrc	[in] points to the input buffer
	scale	[in] scale factor to be applied
	*pDst	[out] points to the output vector
	blockSize	[in] number of samples in each vector
Returns		none

Table 6 Vector Scale

arm_offset_f32 (float32_t *pSrc, float32_t offset, float32_t *pDst, uint32_t blockSize)		
pDst[n] = pSrc[n] + offset, 0 <= n < blockSize		
Parameters	*pSrc	[in] points to the input buffer
	offset	[in] is the offset to be added
	*pDst	[out] points to the output vector

	blockSize	[in] number of samples in each vector
Returns		none

Table 7 Vector Offset

arm_negate_f32 (float32_t *pSrc, float32_t *pDst, uint32_t blockSize)		
pDst[n] = -pSrc[n], 0 <= n < blockSize		
Parameters	*pSrc	[in] points to the input buffer
	*pDst	[out] points to the output vector
	blockSize	[in] number of samples in each vector
Returns		none

Table 8 Vector Negate

1.3 Demo Result

After the program is compile completed, user can be downloaded it to the NuMaker-ETM-M487 development board for verification. User can see the speed difference between the two implementations from the printed results, as shown in Figure 1 and Figure 2.

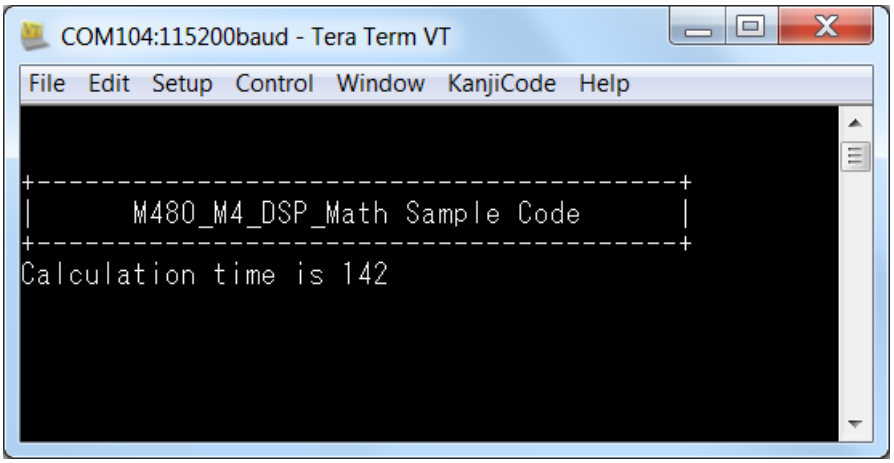


Figure 1 Use M4 DSP Math

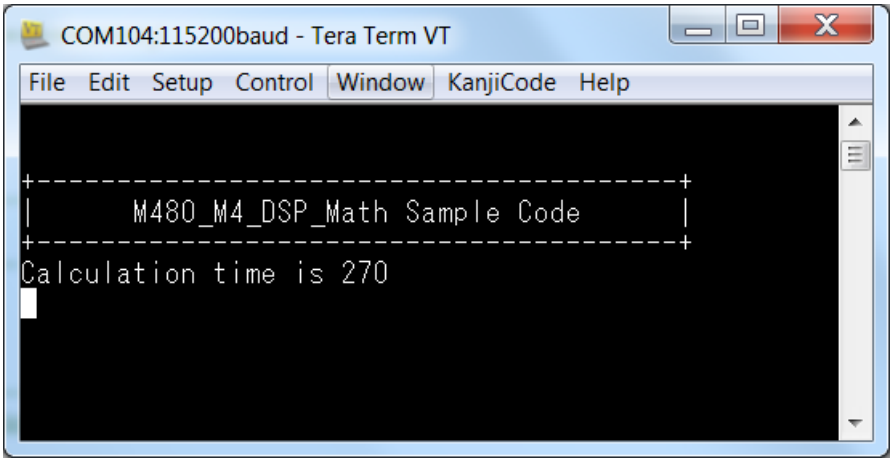


Figure 2 Not use M4 DSP Math

2 Code Description

The example tested with Keil uVersion 5.26. The following steps will introduce how to enable DSP with the Keil environment.

Step 1. After starting Keil, select Target Options, go to the C/C++ tab page, and input “ARM_MATH_CM4=1” and “__FPU_PRESENT=1UL” in the Define field of Preprocessor Symbols, as shown in Figure 3.

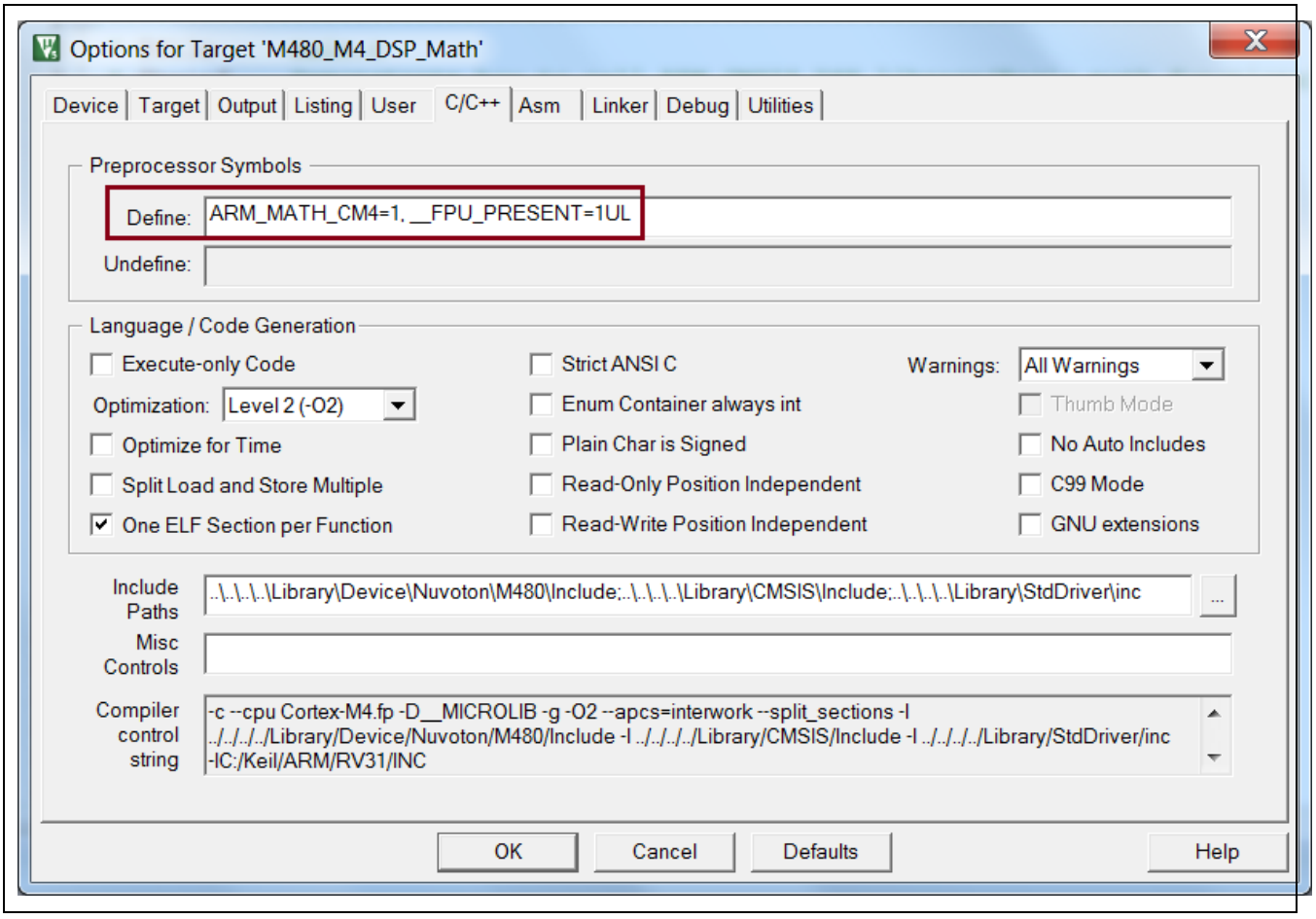


Figure 3 Add data to Define item

Step 2: Add “arm_cortexM4lf_math.lib” to the Library (\Library\CMSIS\Lib\ARM), as shown in Figure 4.

Step 3: Finally, include the arm_math.h file to the main.c(as shown in Figure 5). After above steps is ready, user can call the DSP library in the main.c.

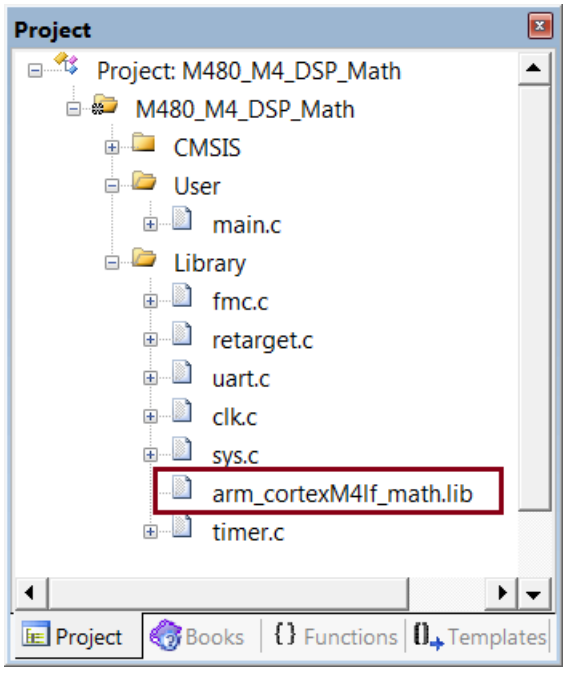


Figure 4 Add arm_cortexM4lf_math.lib

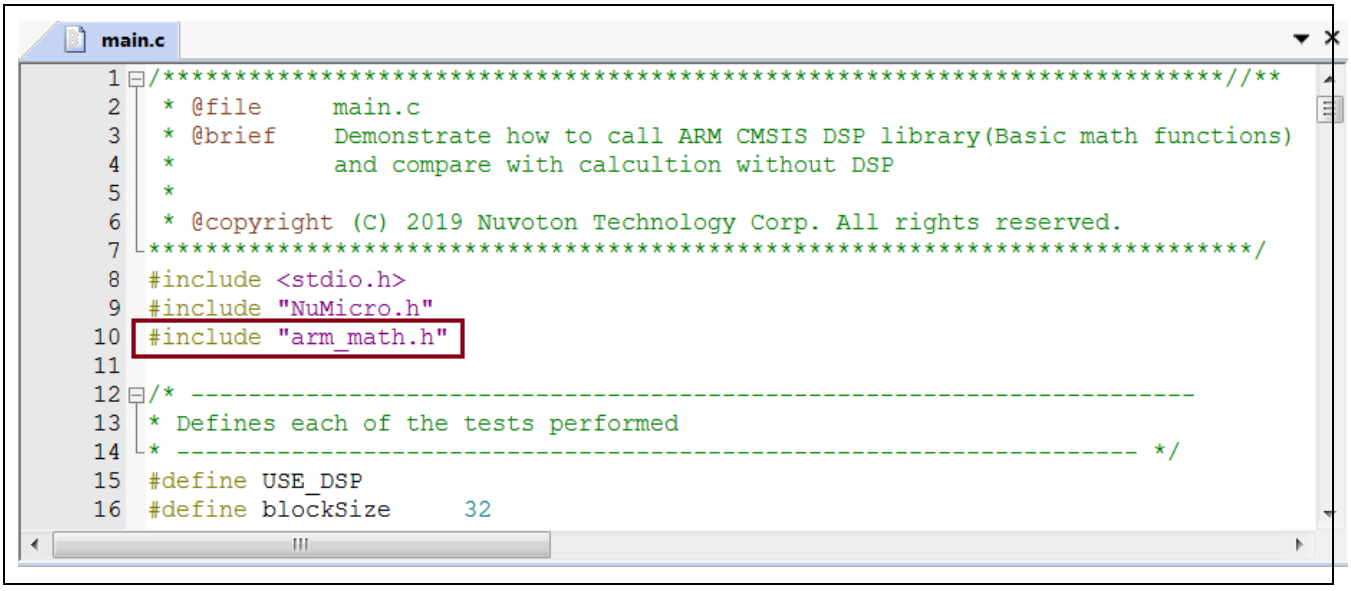


Figure 5 Include arm_math.h

In this example, users can use #define USE_DSP to decide whether to use the DSP library function to perform vector operations or use C functions. The timer used to calculate the execution time. The number of samples in each vector is 32 and data type is 32-bit floating-point values (f32).

```
/* ------------------------------------------------------------------
* Defines each of the tests performed
* ------------------------------------------------------------------ */
#define USE_DSP
```

```
#define blockSize     32
/* ----------------------------------------------------------------------
** Test input data of srcA for blockSize 32
** ------------------------------------------------------------------- */
float32_t srcA_buf_f32[blockSize] =
{
  -0.4325648115282207,  -1.6655843782380970,  0.1253323064748307,
   0.2876764203585489,  -1.1464713506814637,  1.1909154656429988,
   1.1891642016521031,  -0.0376332765933176,  0.3272923614086541,
   0.1746391428209245,  -0.1867085776814394,  0.7257905482933027,
  -0.5883165430141887,   2.1831858181971011, -0.1363958830865957,
   0.1139313135208096,   1.0667682113591888,  0.0592814605236053,
  -0.0956484054836690,  -0.8323494636500225,  0.2944108163926404,
  -1.3361818579378040,   0.7143245518189522,  1.6235620644462707,
  -0.6917757017022868,   0.8579966728282626,  1.2540014216025324,
  -1.5937295764474768,  -1.4409644319010200,  0.5711476236581780,
  -0.3998855777153632,   0.6899973754643451
};
/* ----------------------------------------------------------------------
** Test input data of srcB for blockSize 32
** ------------------------------------------------------------------- */
float32_t srcB_buf_f32[blockSize] =
{
   1.7491401329284098,  0.1325982188803279,   0.3252281811989881,
  -0.7938091410349637,  0.3149236145048914,  -0.5272704888029532,
   0.9322666565031119,  1.1646643544607362,  -2.0456694357357357,
  -0.6443728590041911,  1.7410657940825480,   0.4867684246821860,
   1.0488288293660140,  1.4885752747099299,   1.2705014969484090,
  -1.8561241921210170,  2.1343209047321410,  1.4358467535865909,
  -0.9173023332875400, -1.1060770780029008,   0.8105708062681296,
   0.6985430696369063, -0.4015827425012831,   1.2687512030669628,
  -0.7836083053674872,  0.2132664971465569,   0.7878984786088954,
   0.8966819356782295, -0.1869172943544062,   1.0131816724341454,
   0.2484350696132857,  0.0596083377937976
};

int main()
{
    …
    TIMER_Open(TIMER0, TIMER_CONTINUOUS_MODE, 1);
    TIMER_Start(TIMER0);
```

```
#ifdef USE_DSP
    /* Perform vector operations by DSP library function */
    …
#else
    /* Perform vector operations by C functions */
    …
#endif

    TIMER_Close(TIMER0);
    CalTime=TIMER_GetCounter(TIMER0);

    /* Display function calculation time */
    printf("Calculation time is %d \n", CalTime);

    while(1);
}
```

The following code lists DSP functions.

```
    /* Calculate dot product (32 sample) with DSP */
    arm_dot_prod_f32(srcA_buf_f32, srcB_buf_f32, blockSize, &dotoutput_f32);

    /* Calculate Absolute value (32 sample) with DSP */
    arm_abs_f32(srcA_buf_f32, absoutput, blockSize );

    /* Calculate Addition (32 sample) with DSP */
    arm_add_f32(srcA_buf_f32, srcB_buf_f32, addoutput, blockSize);

    /* Calculate Subtraction (32 sample) with DSP */
    arm_sub_f32 (srcA_buf_f32, srcB_buf_f32, suboutput, blockSize);

    /* Calculate Multiplication (32 sample) with DSP */
    arm_mult_f32 (srcA_buf_f32, srcB_buf_f32, multoutput, blockSize);

    /* Calculate Scale (32 sample) with DSP */
    arm_scale_f32 (srcA_buf_f32, 10, scaleoutput, blockSize);

    /* Calculate Offest (32 sample) with DSP */
    arm_offset_f32 (srcA_buf_f32, 10, offsetoutput, blockSize);

    /* Calculate Negate (32 sample) with DSP */
    arm_negate_f32 (srcA_buf_f32, negoutput, blockSize);
}
```

The following code lists perform vector operations by C functions.

```
    /* Calculate Dot Product (32 sample) without DSP */
    DOT(blockSize);

    /* Calculate Absolute value (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        absout[i]=fabs(srcA_buf_f32[i]);
    }

    /* Calculate Addition (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        addout[i]=srcA_buf_f32[i]+srcB_buf_f32[i];
    }

    /* Calculate Subtraction (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        subout[i]=srcA_buf_f32[i]-srcB_buf_f32[i];
    }

    /* Calculate Multiplication (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        multout[i]=srcA_buf_f32[i]*srcB_buf_f32[i];
    }

    /* Calculate Scale (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        scaleout[i]=10*srcA_buf_f32[i];
    }

    /* Calculate Offest (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        offsetout[i]=10+srcA_buf_f32[i];
    }

    /* Calculate Negate (32 sample) without DSP */
    for(i=0;i<blockSize;i++)
    {
        negout[i]=-srcA_buf_f32[i];
    }
```

3 Software and Hardware Environment

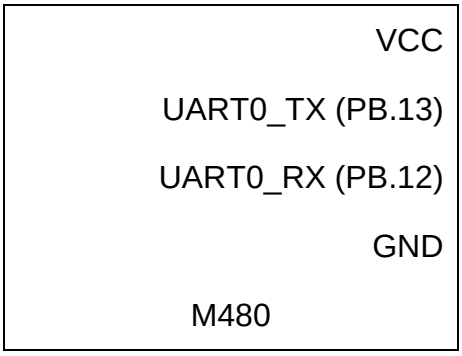
Software Environment

- BSP version
  - M480 Series BSP CMSIS v3.03.000
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuMaker-ETM-M487
- Diagram

Using printf function to output message and the UART0 TX RX pin are PB.13 and PB.12 respectively.



4 Directory Information

- EC_M480_Using_Keil_M4_DSP_Library_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click M480_M4_DSP_Math.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Jun. 14, 2019	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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.