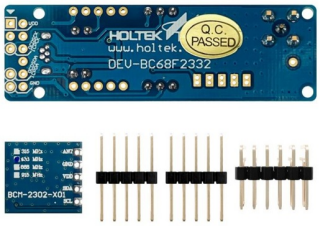

HOLTEK HT32 CMSIS-DSP Library User Guide

HT32 CMSIS-DSP Library
User Guide
D/N: AN0538EN

Contents [hide

1 Introduction
2 Functional Description
3 Environment Setup
4 File Structure
5 Direction for Use
6 Low-Pass Filter – FIR
7 Considerations
8 Disclaimer
9 Documents / Resources
9.1 References
10 Related Posts

Introduction

CMSIS is a software standard interface developed by ARM which has the full name of Cortex Microcontroller Software Interface Standard. With this standard interface, developers can use the same interface to control microcontrollers from different suppliers thus greatly shortening their development and learning time. For more information, refer to the CMSIS official website: http://www.keil.com/pack/doc/CMSIS/General/html/index.html. This text mainly describes the CMSIS-DSP application in the HT32 series of microcontrollers which includes environment setup, direction for use, etc.

Functional Description

CMSIS-DSP Features

CMSIS-DSP, which is one of the CMSIS components includes the following features.

1. Provides a set of generic signal processing functions dedicated to the Cortex-M.
2. The function library provided by ARM has over 60 functions.
3. Supports q7, q15, q31

   (Note) and floating-point (32-bit) data types
4. Implementations are optimized for the SIMD instruction set which is available for Cortex-M4/M7/M33/M35P.

Note: The naming q7, q15, and q31 in the function library respectively represent the 8, 16, and 32bit fixed-points.

CMSIS-DSP Function Library Items

The CMSIS-DSP function library is divided into the following categories:

1. Basic maths functions, fast maths functions, and complex maths functions
2. Signal filtering functions
3. Matrix functions
4. Transform functions
5. Motor control functions
6. Statistical functions
7. Support functions
8. Interpolation functions

Environment Setup

This section will introduce the hardware and software used in the application example.

Hardware

Although the CMSIS-DSP supports the full HT32 series, it is suggested to use an MCU with an SRAM capacity larger than 4KB as the CMSIS-DSP application example requires a larger SRAM size. This text takes the ESK32-30501 as an example which uses the HT32F52352.

Software

Before using the application example, first, ensure that the newest Holtek HT32 Firmware Library has been downloaded from the Holtek official website. The download location is shown in Figure

Decompress the file after downloading.

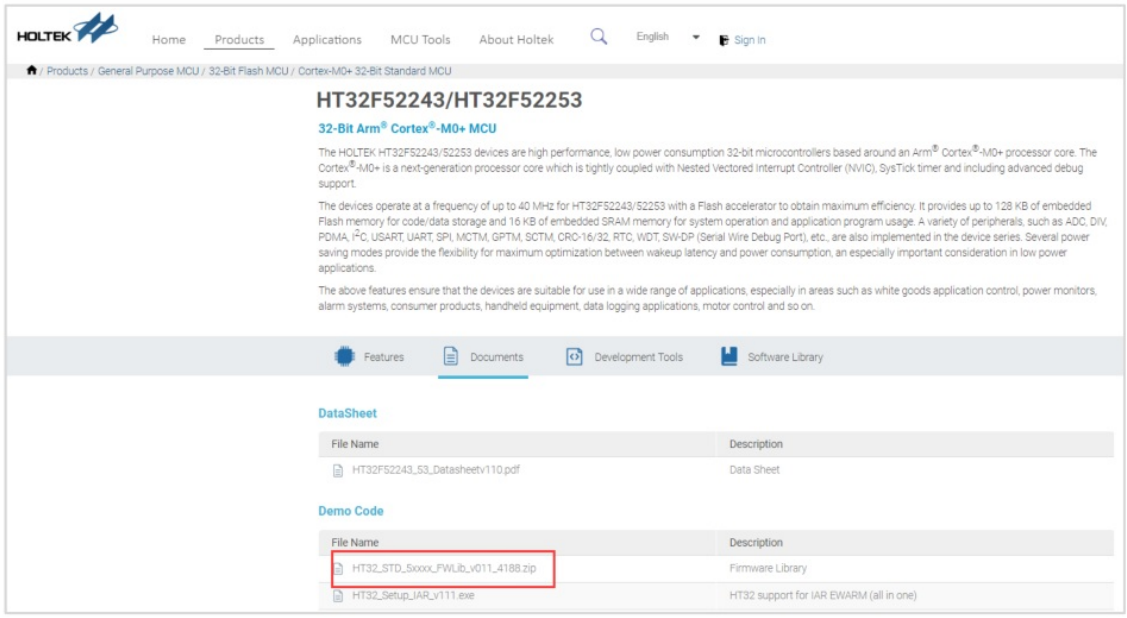

Figure 1. HT32 Firmware Library Download Website

Download the CMSIS-DSP application code through the link below. The application code is packed as a zip file with the name of HT32_APPFW_xxxxx_CMSIS_DSP_vn_m.zip.
Download path: **https://mcu.holtek.com.tw/ht32/app.fw/CMSIS_DSP/**
The file naming rule is shown in Figure 2.

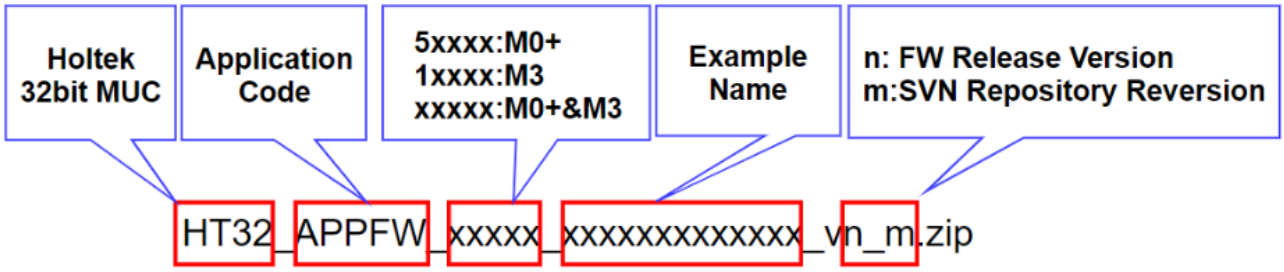

Figure 2. Application Code File Name Introduction

As the application code does not contain firmware library files, users need to place the unzipped application code and firmware library files into the correct path before starting compilation. The application code file contains two folders, which are the application and library whose location is shown in Figure 3. Place these two folders into the firmware library root directory to complete the file path configuration as shown in Figure 4. Users can also decompress the application code and firmware library compressed files into the same path to achieve the same effect. For this example, the directory for CMSIS_DSP will be seen under the application folder after decompression.

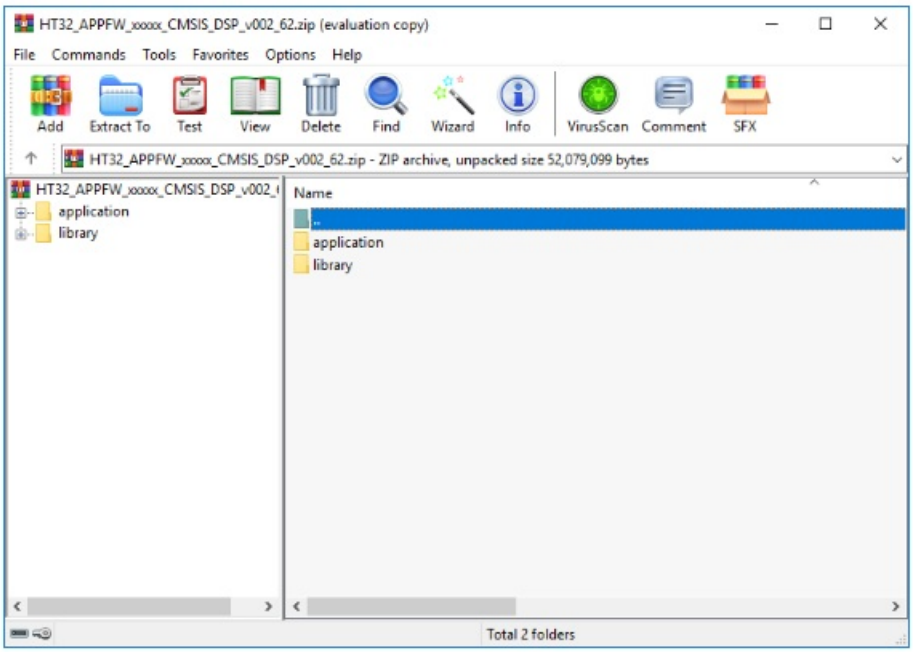

Figure 3. HT32_APPFW_xxxxx_CMSIS_DSP_vn_m.zip Content

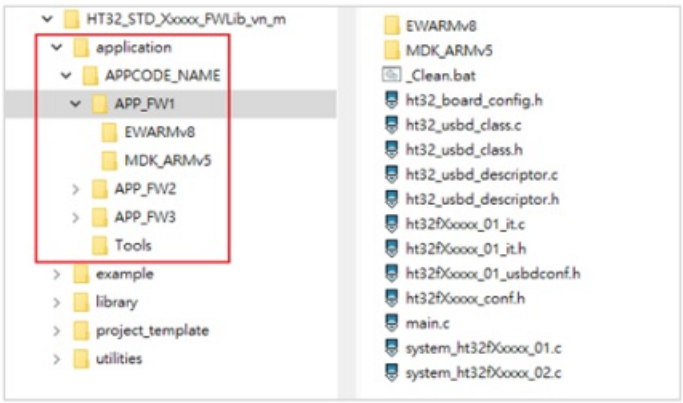

Figure 4. Decompression Path

File Structure

The two main folders included in the application code file, library\CMSIS, and application\CMSIS_DSP, are individually described below.
The contents of the library\CMSIS folder are as follows.

Folder Name	Description
DSP_Lib	Application FW source code
DSP_Lib\Examples	Contains multiple standard examples of the CMSIS-DSP function library wh ich are provided by ARM. The settings for these projects are executed in a simulated way without requiring an MCU. Users can quickly learn how to us e these examples by executing them.
DSP_Lib\Source	CMSIS-DSP function library source code
Include	Necessary header file when using the CMSIS-DSP function library
Include\arm_common_tables.h	Declaration of external array variables (extern)
Include\arm_const_structs.h	Declaration of external constants
Include\arm_math.h	This file is very important as the interface for using the CMSIS-DSP functio n library. Calls to any function library API are implemented through arm_ma th.h.
Lib\ARM	CMSIS-DSP function library for ARMCC l arm_cortexM3l_math.lib (Cortex-M3, Little ndian) l arm_cortexM0l_math.lib (Cortex-M0 / M0+, Little endian)
Lib\GCC	CMSIS-DSP function library for GCC l libarm_cortexM3l_math.a (Cortex-M 3, Little ndian) l libarm_cortexM0l_math.a (Cortex-M0 / M0+, Little endian)

The application\CMSIS_DSP folder contains multiple CMSIS_DSP examples, which use the HT32 series of MCUs and support the full HT32 series. The projects are developed using the Keil MDK_ARM.

Folder Name	Description
arm_class_marks_example	Demonstrates how to obtain the maximum value, minimum value, expecte d value, standard deviation, variance and matrix functions.
arm_convolution_example	Demonstrates the convolution theorem via the complex FFT and support f unctions.
arm_dotproduct_example	Demonstrates how to obtain dot product via the multiplication and addition of vectors.
arm_fft_bin_example	Demonstrates how to calculate the maximum energy window (bin) in the fr equency domain of input signals using the complex FFT, complex magnitu de, and maximum module functions.
arm_fir_example	Demonstrates how to implement low-pass filtering using FIR.
arm_graphic_equalizer_example	Demonstrates how to change sound quality using the graphic equalizer.
arm_linear_interp_example	Demonstrates the usage of linear interpolation module and fast maths mo dule.
arm_matrix_example	Demonstrates matrix correlation calculation including matrix transform, ma trix multiplication, and matrix inverse.
arm_signal_converge_example	Demonstrates the self-adjustable FIR low-pass filter using NLMS (Normali sed Least Mean Square), FIR, and basic maths modules.
arm_sin_cos_example	Demonstrates trigonometric calculations.
arm_variance_example	Demonstrates how to calculate variance via basic maths and support functions.
filter_iir_high_pass_example	Demonstrates how to implement high-pass filtering using IIR.

Test

This text will use the application\CMSIS_DSP\arm_class_marks_example as the test example. Before starting testing, check whether the ESK32-30501 has been connected or not and ensure that the application code and firmware library have been placed in the right location. Open the application\CMSIS_DSP\arm_class_marks_example folder and execute the _CreateProject.bat file, as shown below. After this, open the MDK_ARMv5 (or MDK_ARM for Keilv4), to find that this example supports the full HT32 series. Open the Project_52352.uvprojx project because the ESK32-30501 is used.

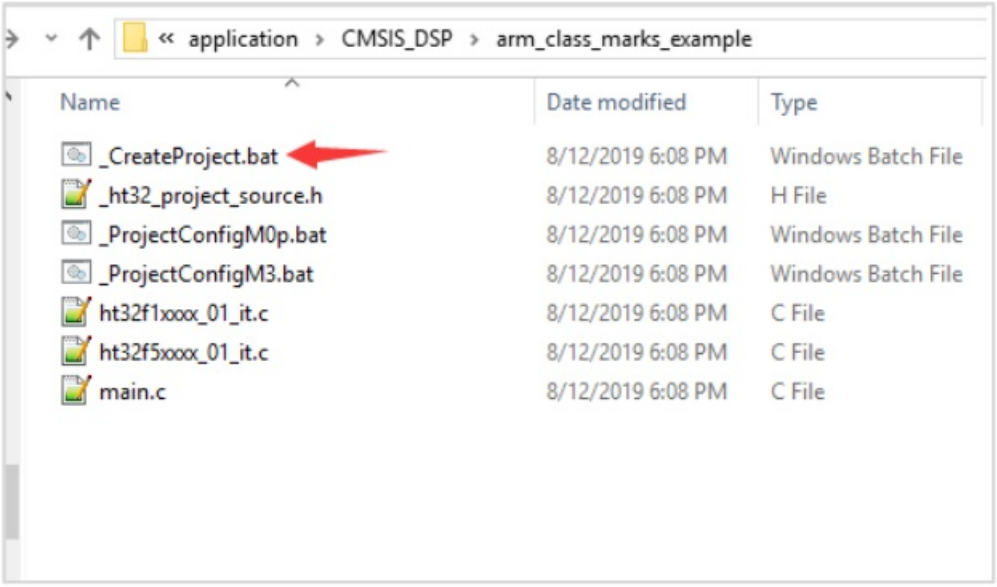

Figure 5. Project Generation

After opening the project, compile (shortcut key “F7”), download (shortcut key “F8”), debug (shortcut key “Ctrl+F5”) and then execute (shortcut key “F5”). The execution results can be observed using the variables listed below.

Variable Name	Data Direction	Description	Execution Result
testMarks_f32	Input	One 20×4 array	–
testUnity_f32	Input	One 4×1 array	–
test output	Output	The product of testMarks_f32 and testUnity_f32	{188 229 210…}
max_marks	Output	The maximum value of the elements in the test output array	364
min_marks	Output	The minimum value of the elements in the test output array	156
mean	Output	The expected value of the elements in the test output array	212.300003
std	Output	The standard deviation of the elements in the test output array	50.9128189
var	Output	The variance of the elements in the test output array	2592.11523

Direction for Use

Integration

This section will introduce how to integrate CMSIS-DSP into users’ projects.

Step 1

First, add a new Define symbol when setting the project, “ARM_MATH_CM0PLUS” for M0+ and “ARM_MATH_CM3” for M3. Setting procedure: (1) Options of Target shortcut key “Alt+F7”), (2) Select C/C++ page, (3) Add a new definition in the Define option, as shown below.

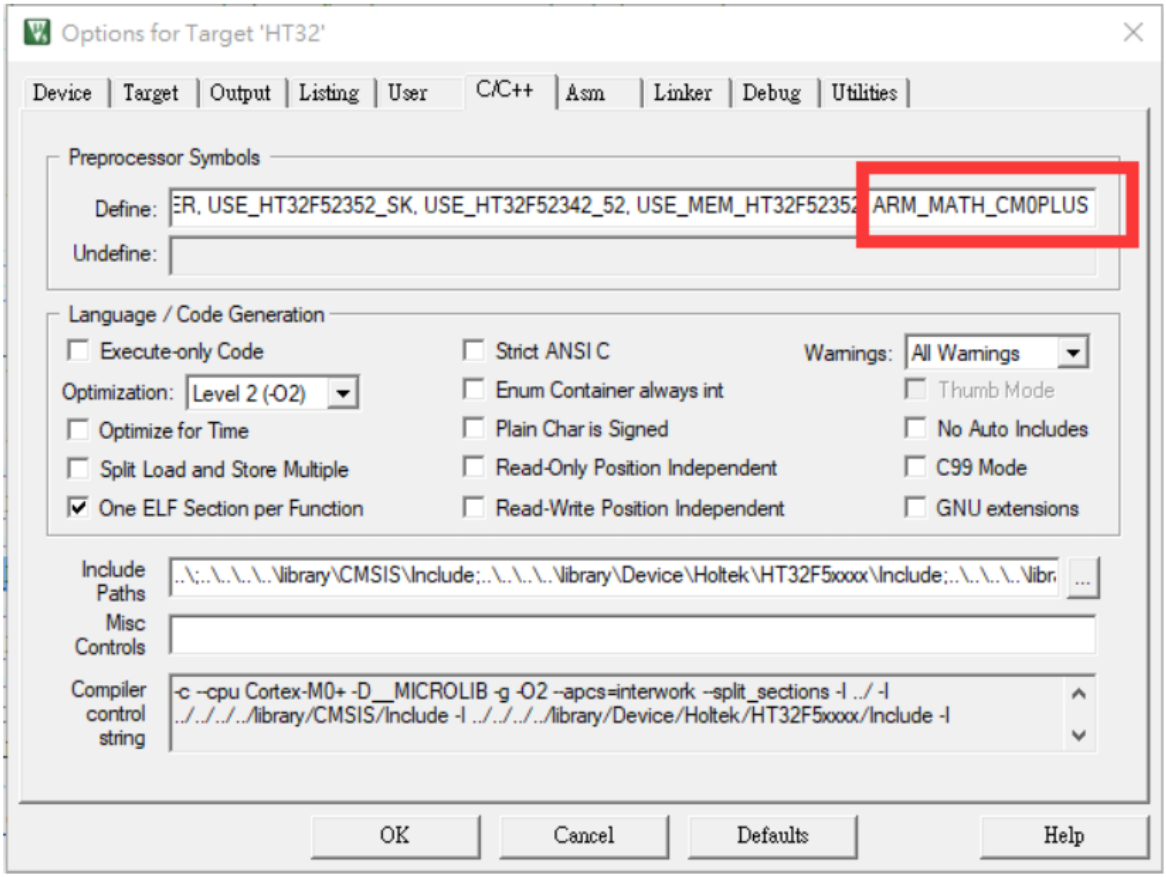

Figure 6. Add a New Define

Step 2

To add an Include path, click the button next to the “Include Paths” option on the C/C++ page. Then a Folder Setup window will pop up, where a new path ..\..\..\..\library\CMSIS\Include” can be added, as shown below.

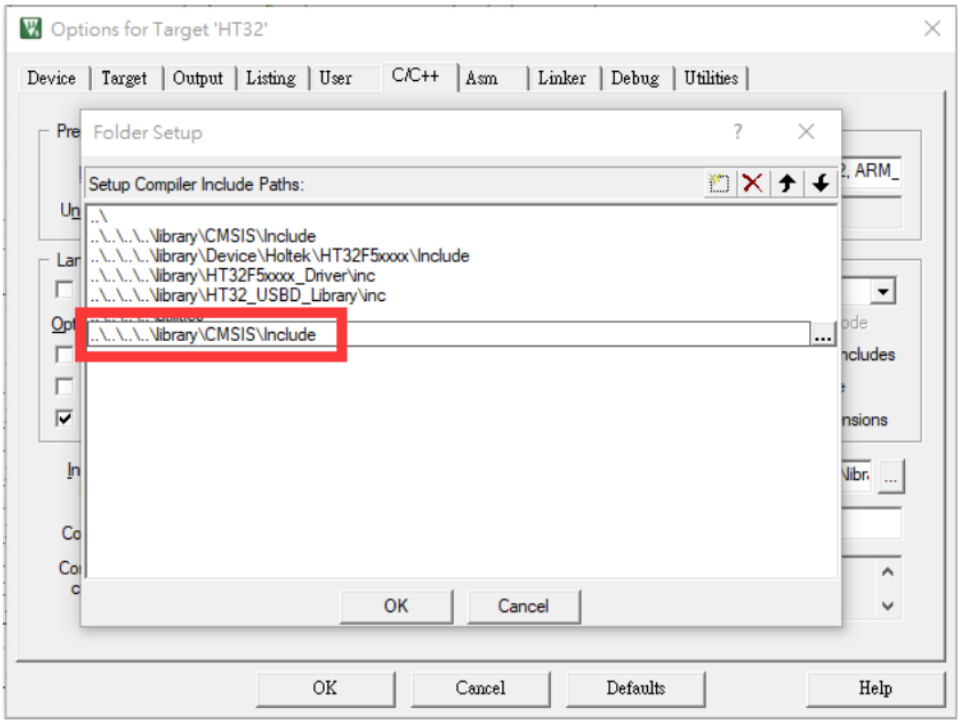

Figure 7. Add a New Include Path

Step 3 (Optional)

To add the function library, click the “Manage Project Items” button as shown below. If the button is not seen, click “Window → Reset View to Defaults → Reset”, so that the IDE window configuration will return to its default

settings. After this, the “Manage Project Items” button will be shown.

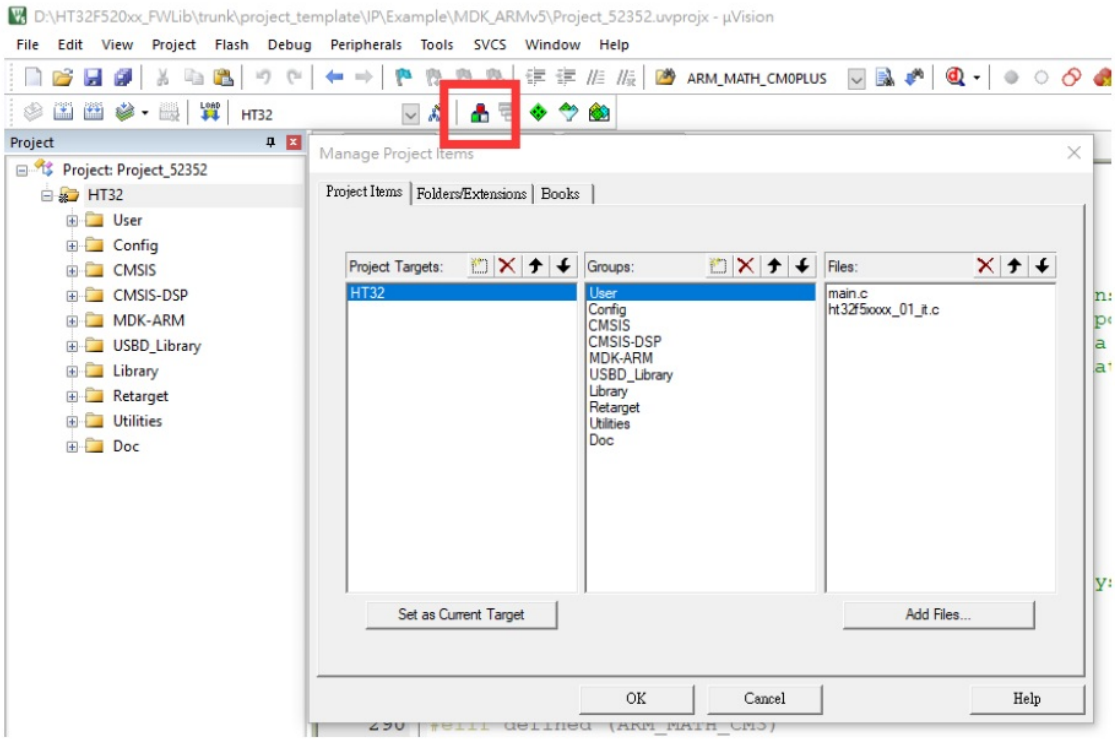

Figure 8. Add a New Folder – 1

Add the CMSIS-DSP folder using the buttons as shown in the red box below and move it under the CMSIS folder using the “Move Up” button. Close the Manage Project tems window when finished.

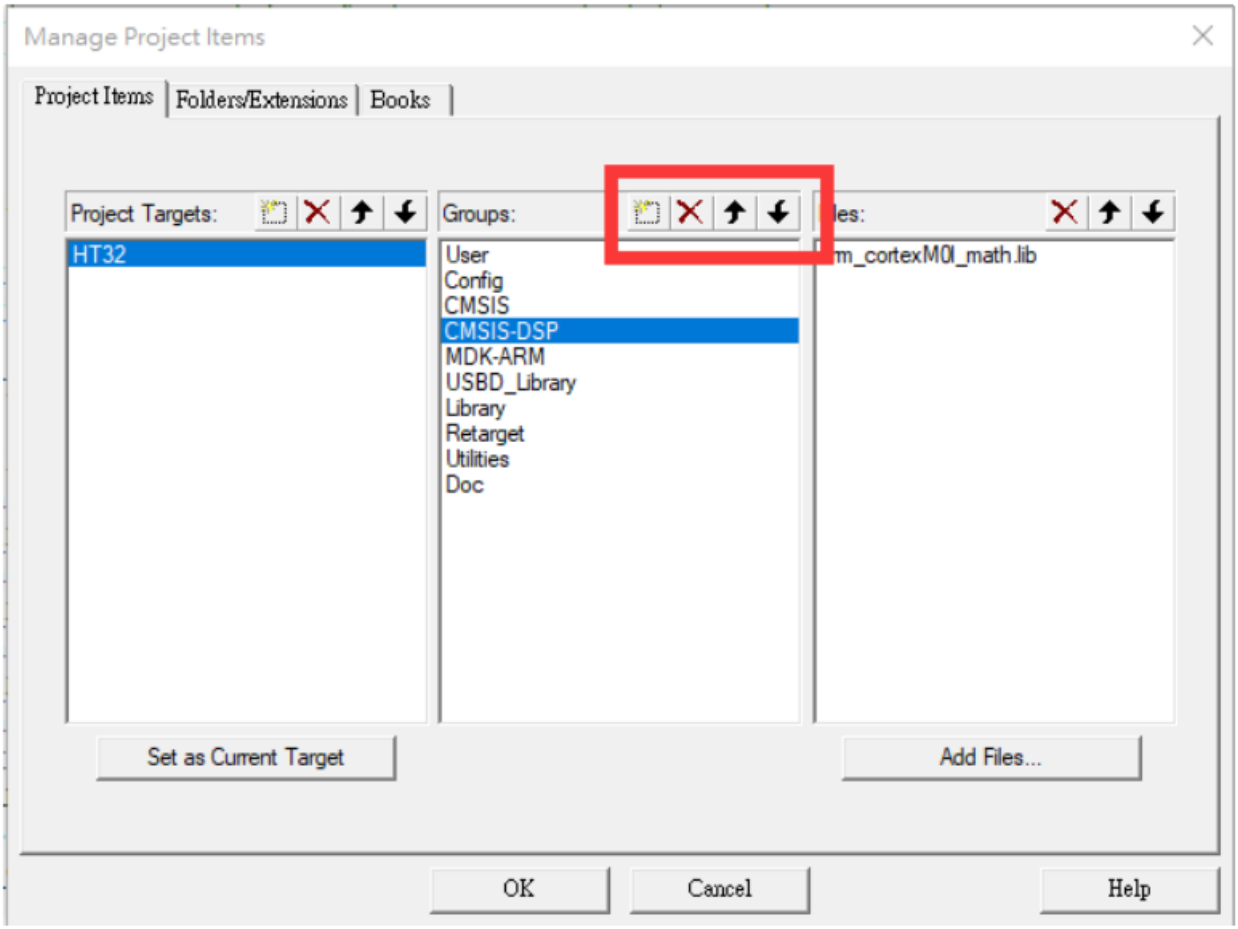

Figure 9. Add a New Folder – 2

Step 4

Double-click the CMSIS-DSP folder on the left (if Step 3 is skipped, select any folder such as User or CMSIS, etc.), then add the CMSIS-DSP function library into it. Choose \library\CMSIS\Lib\ARM\arm_cortexM0l_math.lib for M0+ or \library\CMSIS\Lib\ARM \arm_cortexM3l_math.lib for M3. Upon completion, the function library arm_cortexMxl_math.lib will be shown in the CMSIS-DSP folder, as shown below.

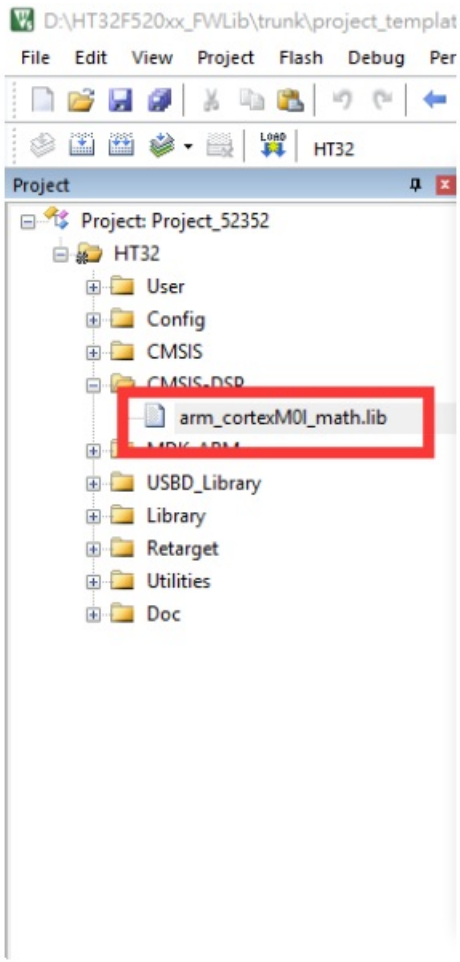

Figure 10. Add a New Folder – 3

Step 5

Add the head file “arm_math.h” into main.c, as shown below. Now all the integration settings have been complete

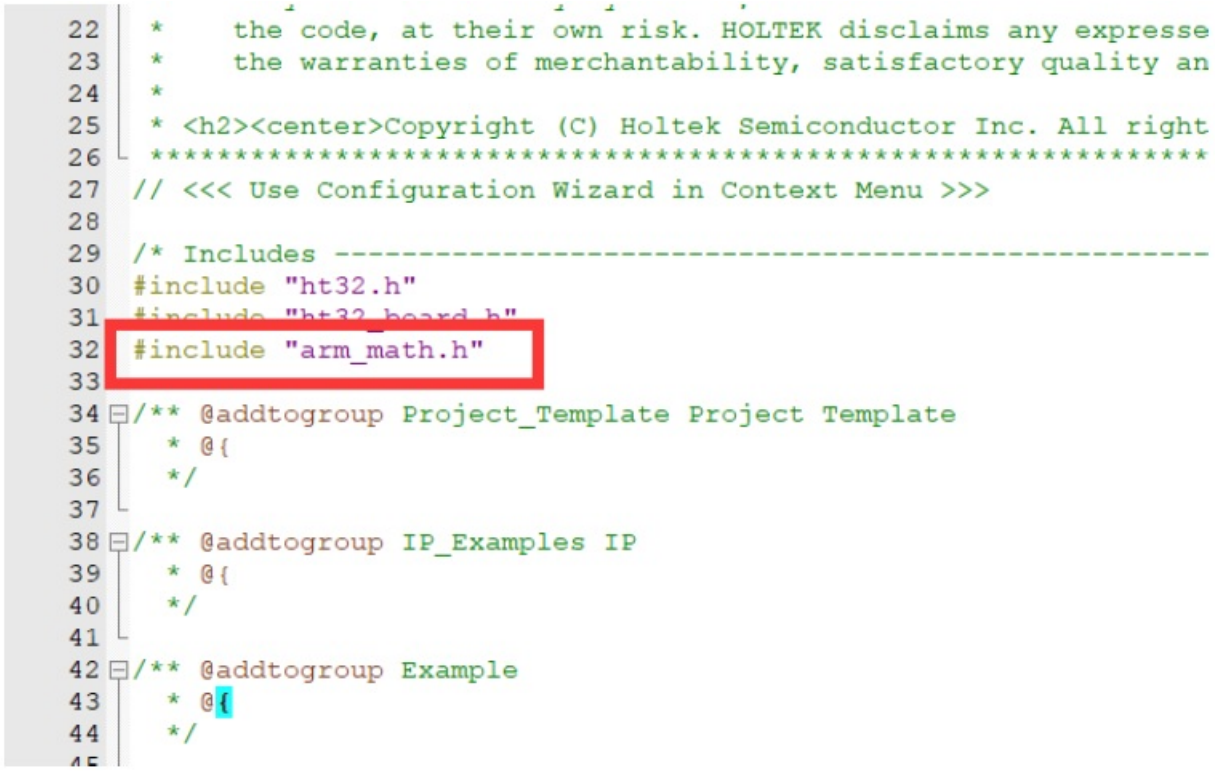

Figure 11. Add "arm_math.h"

Low-Pass Filter – FIR

This section, by introducing the application\CMSIS_DSP\arm_fir_example, will demonstrate how to set the FIR filter and remove high-frequency signals using the FIR. The input signal is composed of 1kHz and 15kHz sine waves. The signal sampling frequency is 48kHz. Signals above 6kHz are filtered by the FIR and 1kHz signals are output. The application code is divided into several parts.

1. Initialization. To initialize FIR, the following API is used.
   void arm_fir_init_f32 (arm_fir_instance_f32 *S, uint16_t numTaps, float32_t *pCoeffs, float32_t *pState, uint32_t blockSize);
   S: FIR filter structure
   numerals: The number of filter stages (the number of filter coefficients). In this example, numTaps=29.
   Coffs: Filter coefficient. There are 29 filter coefficients in this example which is calculated by MATLAB.
   state: Status indicator
   blockSize: Represents the number of samples processed at one time.
2. Low-pass filter. By calling the API of FIR, 32 samples are processed each time and there are 320 samples in total. The API used is shown below.
   void arm_fir_f32 (const arm_fir_instance_f32 *S, float32_t *pSrc, float32_t *pDst, uint32_t blockSize);
   S: FIR filter structure
   pSrc: Input signal. A mixed signal of 1kHz and 15kHz is input in this example. pDst: Output signal. The expected output signal is 1kHz. blockSize: Represents the number of samples processed at one time.
3. Data verification. The filtering result obtained by MATLAB is regarded as the reference and the filtering result obtained by CMSIS-DSP is the actual value. Compare the two results to verify whether the output result is correct or not. float arm_snr_f32(float *pRef, float *pTest, uint32_t buffSize)
   Pref: Reference value generated by MATLAB.
   post: Actual value generated by CMSIS-DSP.

blockSize: Represents the number of samples processed at one time.

As shown below, Input Data shows that the signal is not yet filtered and Output Data shows the filtered result. The Y-axis represents the amplitude of the signal and the sampling frequency is 48kHz, so the X-axis number plus one represents time plus 20.833μs. It can be found from Figure 12 and Figure 13 that the 15kHz signal is eliminated and only the 1kHz signal is left.

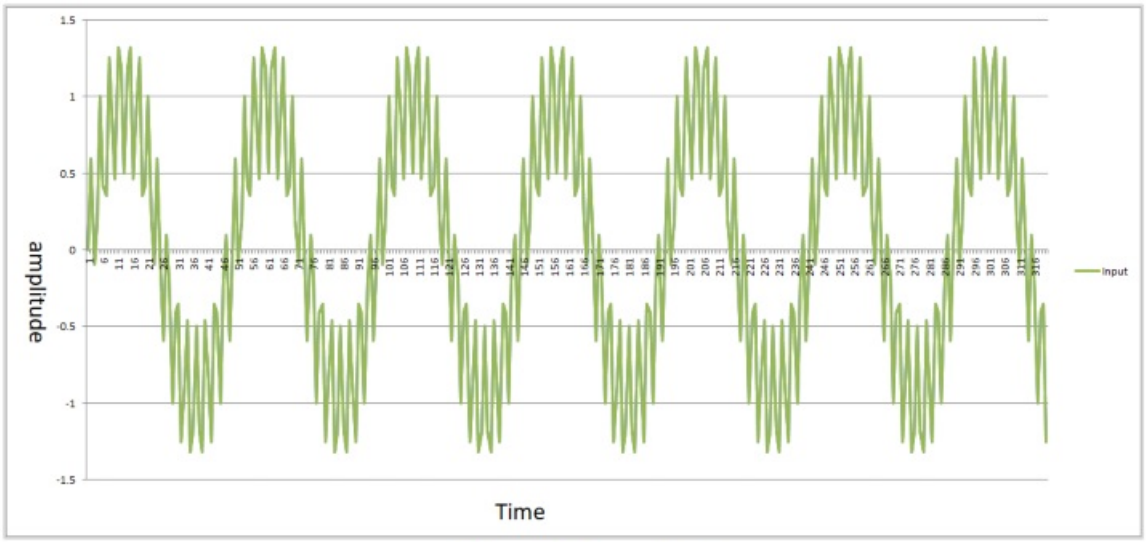

Figure 12. Input Data

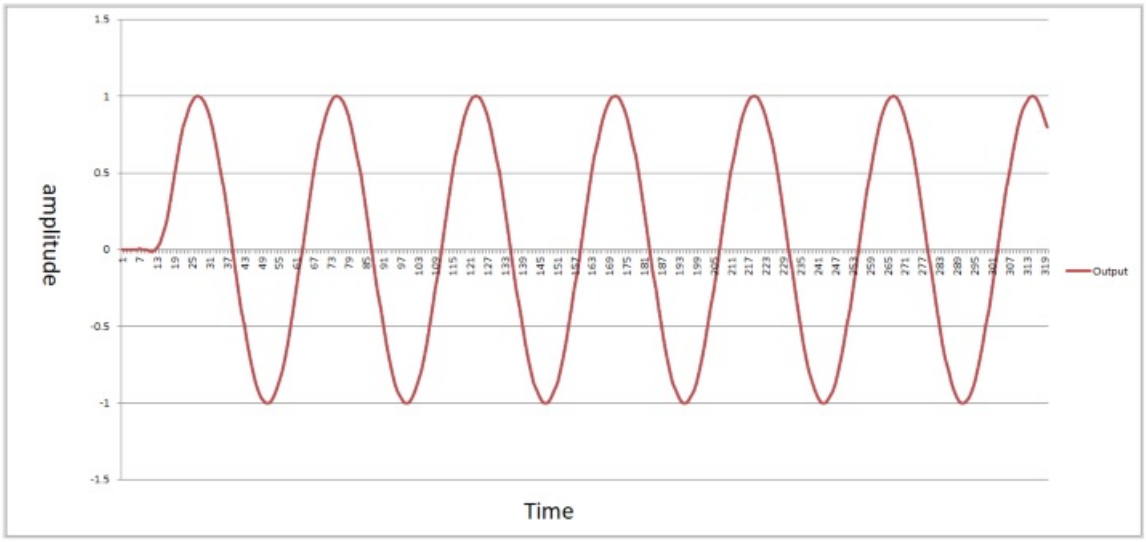

Figure 13. Output Data

High-Pass Filter– IIR

This section, by introducing the application\CMSIS_DSP\filter_iir_high_pass_example, will demonstrate how to set the IIR filter and remove low-frequency signals using the IIR. The input signal is composed of 1Hz and 30Hz sine waves. The signal sampling frequency is 100Hz and a total of 480 points are sampled. Signals below 7Hz are removed by the IIR.

The application code is divided into several parts.

1. There are 480 samples. Sample 0~159 are 30Hz sine waves, sample 160~319 are 1Hz sine waves and sample 320~479 are 30Hz sine waves.
2. Initialization. To initialize the IIR, the following API is used. void arm_biquad_cascade_df1_init_f32 (arm_biquad_casd_df1_inst_f32 *S, uint8_t numStages, float32_t *pCoeffs, float32_t *state));
   S: IIR filter structure

sum stages: The number of second-order stages in the filter. In this example, numStages=1.

Coffs: Filter coefficient. There are 5 filter coefficients in this example.

state: Status indicator

3. High-pass filter. By calling the API of the IIR, 1 sample is processed each time and there are 480 samples in total. The API used is shown below. void arm_biquad_cascade_df1_f32 (const arm_biquad_casd_df1_inst_f32 *S, float32_t *pSrc, float32_t *pDst, uint32_t blockSize);

   S: IIR filter structure

   pSrc: Input signal. A mixed signal of 1Hz and 30Hz is input in this example.

   pDst: Output signal. The expected output signal is 30Hz.

   blockSize: Represents the number of samples processed at one time.

4. Result output. The input and output signals are output to the PC through print. As shown below, Input Data shows that the signal is not yet filtered and Output Data shows the filtered result. The Y-axis represents the amplitude of the signal and the sampling frequency is 100Hz, so the X-axis number plus one represents time plus 10ms. It can be found from Figure 14 and Figure 15 that the 1Hz signal is eliminated and only the 30Hz signal is left.

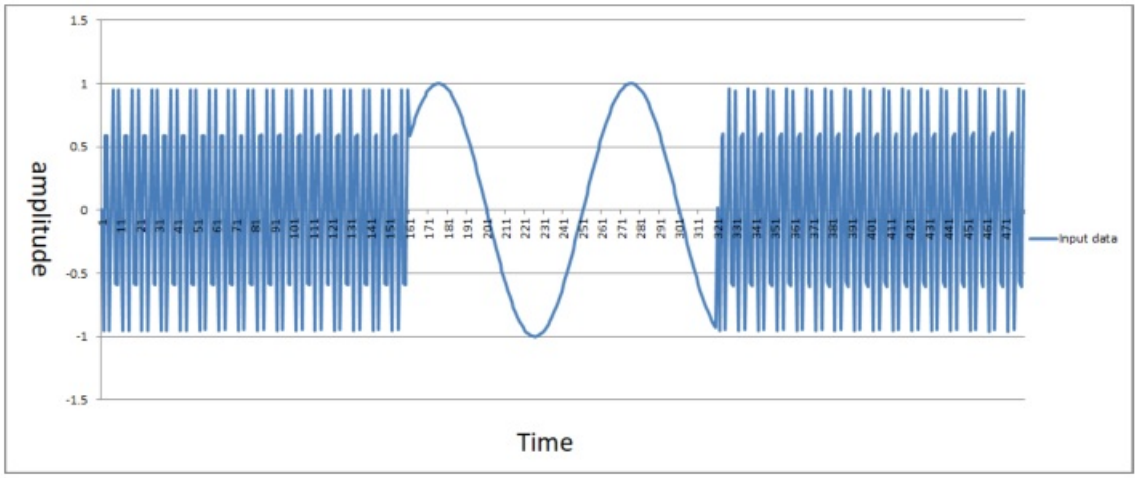

Figure 14. Input Data

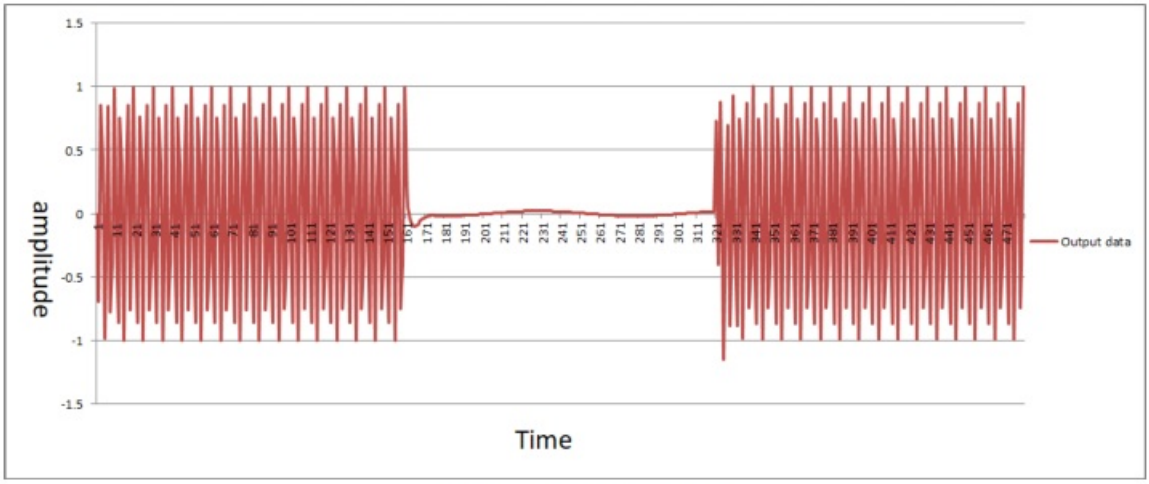

Figure 15. Output Data

Considerations

Users should pay special attention to the memory size after compiling when using the CMSIS-DSP function library. Ensure that no memory overflow occurs before testing.

Conclusion

The CMSIS-DSP has great abilities in signal processing and mathematical calculation and is worthy of serious consideration by users.
Reference Material
Reference website: http://www.keil.com/pack/doc/CMSIS/General/html/index.html
Versions and Modification Information

Date	Author	Issue	Modification Information
2022.06.02	Writing, Liu	V1.10	Modify the download path
2019.09.03	Allen, Wang	V1.00	First Version

Disclaimer

Documents / Resources

	HOLTEK HT32 CMSIS-DSP Library [pdf] User Guide HT32, CMSIS-DSP Library, HT32 CMSIS-DSP Library, Library

References

- Introduction
- HT32 Application Firmware - CMSIS_DSP