Fast Fourier transform using M4 DSP

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code uses FFT to get main frequency from Line in
BSP Version	NUC505 Series BSP CMSIS V3.02.000
Hardware	NuTiny-EVB-NUC505 V1.6

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

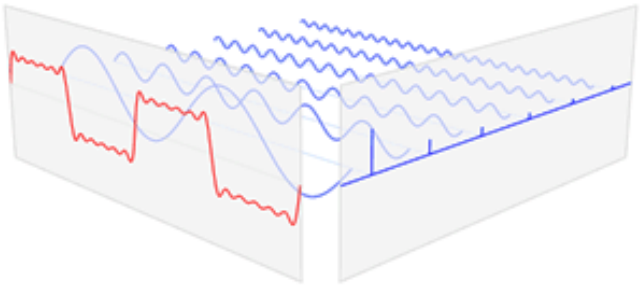
1.1 Introduction

This sample code uses NUC505 internal codec and gets audio data from LINE IN. Use CMSIS DSP library to do Fourier analysis. Users can hear the sound from LINE IN by connecting the headphones. In the sample code, only the right channel audio is analyzed, the sampling frequency is 8000 Hz, and the data length is 16 bits. The number of samples for FFT is 1024. Decreasing the number of samples reduces the computation time, but the resolution is worse. The user can set the volume threshold according to the VolTH parameter.

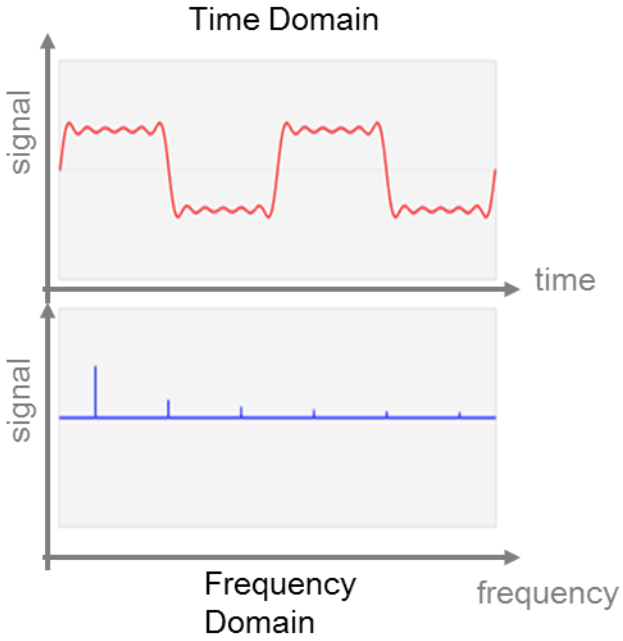
1.2 Principle

Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. As below picture 1-1 square wave *f*, it is composed by many sine waves. The composition of the sine wave is shown in Figure 1-2. According to Fourier analysis, this sine wave can be decomposed into a number of discrete frequencies or a spectrum of frequencies over a continuous range as picture 1-3.

Picture 1-1 square wave *f*



Picture 1-2 Square wave is composed by many sine waves



Picture 1-3 Square wave sine wave frequency and intensity

CMSIS DSP library provides FFT function. User can input data and get the maximum frequency. The equations used are defined as shown in Table 1-1~Table 1-3.

`arm_cfft_f32 (const arm_cfft_instance_f32 * S, float32_t * p1, uint8_t ifftFlag, uint8_t bitReverseFlag)`		
Parameters :	S	[in] points to an instance of the floating-point CFFT structure
	p1	[in, out] points to the complex data buffer of size 2*fftLen. Processing occurs in-place

	ifftFlag	[in] flag that selects transform direction value = 0: forward transform value = 1: inverse transform
	bitReverseFlag	[in] flag that enables / disables bit reversal of output value = 0: disables bit reversal of output value = 1: enables bit reversal of output
Returns :		

Table 1-1 Processing function for the floating-point complex FFT

arm_cmplx_mag_f32(float32_t *pSrc, float32_t *pDst, uint32_t numSamples)		
Parameters :	*pSrc	[in] points to input vector
	*pDst	[out] points to output vector
	numSamples	[in] number of samples in each vector
Returns :		none

Table 1-2 Computes the magnitude of the elements of a complex data vector

arm_max_f32(float32_t *pSrc, uint32_t blockSize, float32_t *pResult, uint32_t *pIndex)		
Parameters :	*pSrc	[in] points to the input vector
	blockSize	[in] number of samples in input vector
	*pResult	[out] maximum value returned here
	*pIndex	[out] index of maximum value returned here
Returns :		none

Table 1-3 Computes the maximum value of an array of data

After finding the maximum frequency position of the audio, the maximum frequency of the audio can be calculated according to the sampling rate and the number of FFT samples. The calculation formula is as follows:

$$Audio\ frequency = \frac{Sample\ rate}{number\ of\ FFT\ samples} \times Position$$

1.3 Demo Result

```
+------------------------------------------------------------+
|      Line-In get spectrum Sample Code (internal CODEC)     |
+------------------------------------------------------------+
Sample rate is 8 kHz
NOTE: Need 3.5 mm Male to Male Cable.

The audio frequency is as follows:
406.0 Hz
406.0 Hz
414.0 Hz
421.0 Hz
421.0 Hz
429.0 Hz
437.0 Hz
445.0 Hz
445.0 Hz
453.0 Hz
Silence...
Silence...
Silence...
```

2 Code Description

First, initialize the NUC505 in main.c and uses DMA to transmit/receive data. When a data is received, a FFT is calculated.

```
    /* This sample code only get audio data from right channel */
    while (1) {
        if (DataReady == 1) {
            /* Process the data through the CFFT/CIFFT module */
            arm_cfft_f32(&arm_cfft_sR_f32_len1024, RawData, ifftFlag, doBitReverse);
            /* Process the data through the Complex Magnitude Module for calculating the
magnitude at each bin */
            arm_cmplx_mag_f32(RawData, FFTOutput, BUFF_LEN / 2);
            /* Find maximum frequency */
            arm_max_f32(FFTOutput, BUFF_LEN / 2, &maxValue, &maxIndex);

            /* Check sound */
            if (maxValue < VolTH)
                printf("Silence... \n");
            else {
                /* Main frequency calculation, the resolution is sampleRate / fftSize */
                MainFreq = maxIndex * SampleRate / BUFF_LEN;
                printf("%0.1f Hz\n", MainFreq);
                DataReady = 0;
            }
        }
    }
```

When a data is received, it will enter I^{2}S IRQ handler. The audio input data(from LINE IN) will be stored in PcmRxBuff, the output data(to HP OUT) will be stored in PcmTxBuff, and FFT processing data will be stored in RawData.

```
void I2S_IRQHandler(void)
{
    uint32_t u32I2SIntFlag;
    uint16_t i;

    /* Check interrupt flag */
    u32I2SIntFlag = I2S_GET_INT_FLAG(I2S, (I2S_STATUS_RDMATIF_Msk |
I2S_STATUS_RDMAEIF_Msk));
```

```
    /* Copy RX data to TX buffer */
    if (u32I2SIntFlag & I2S_STATUS_RDMATIF_Msk) {

        I2S_CLR_INT_FLAG(I2S, I2S_STATUS_RDMATIF_Msk);

        /* Copy RX data to TX buffer */
        for (i = 0; i < BUFF_LEN; i++) {
            PcmTxBuff[0][i * 2] = PcmRxBuff[0][i * 2];
            PcmTxBuff[0][i * 2 + 1] = PcmRxBuff[0][i * 2 + 1];
            /* Copy data to working buffer */
            RawData[i * 2] = PcmRxBuff[0][i * 2]; //real part
            RawData[i * 2 + 1] = 0; //imaginary part
        }
        /* Working buffer data is ready */
        DataReady = 1;

    } else if (u32I2SIntFlag & I2S_STATUS_RDMAEIF_Msk) {

        I2S_CLR_INT_FLAG(I2S, I2S_STATUS_RDMAEIF_Msk);

        if (s_flag1 == 0) {
            s_flag1 = 1;
            /* Enable TX -> start to play */
            I2S_ENABLE_TXDMA(I2S);
            I2S_ENABLE_TX(I2S);
        }
        /* Copy RX data to TX buffer */
        for (i = 0; i < BUFF_LEN; i++) {
            PcmTxBuff[1][i * 2] = PcmRxBuff[1][i * 2];
            PcmTxBuff[1][i * 2 + 1] = PcmRxBuff[1][i * 2 + 1];
        }
    }
}
```

3 Software and Hardware Environment

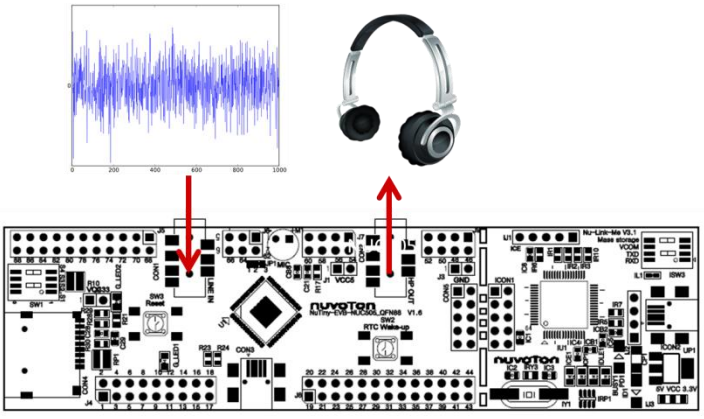
Software environment

- BSP version
  - NUC505 Series BSP CMSIS V3.02.000
- IDE version
  - Keil uVersion 5.26

Hardware environment

- Circuit components
  - NuTiny-EVB-NUC505 V1.6
  - 3.5 mm jack to jack cable
  - 3.5 mm headphones
- Diagram

Use 3.5 mm jack to jack cable to connect to onboard socket(LINE IN) and use 3.5 mm headphones to connect to onboard socket(HP OUT) and the user will hear the sound.



4 Directory Information

- EC_NUC505_LineInGetSpectrum_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 LineInGetSpectrum.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
Jul. 03, 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.