Demonstrate the Use of FFT

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	Demonstrate the method and the result of using Fast Fourier Transform to convert the ADC samples into frequency domain spectrum.
BSP Version	NUC121_125_Series_BSP_CMSIS_V3.01.003
Hardware	NuTiny-SDK-NUC121_V1.0

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

Fast Fourier Transform (FFT), a commonly used method for signal analysis, can be used to convert discrete-time signals into the frequency domain spectrum. User can benefit from FFT by observing the spectrum and find out the frequency composition of the original signal. This example will demonstrate how to use ADC and TIMER to sample a piece of discrete signal, which will be used for Fast Fourier Transform, and finally observe whether the spectral value distribution matches the characteristics of the original input signal. Three types of signal length are supported, which are 128 samples, 256 samples, and 512 samples.

1.2 Principle

This example is implemented using the functionality of TIMER trigger ADC. An additional function generator is used to create a sine wave with a fixed frequency of 60 Hz, simulating the source of the signal to be analyzed in a typical application scenario, and giving the microcontroller an input signal for ADC to sample. The user can adjust the sampling frequency according to the requirements of the application, note that the speed of the sampling frequency directly affects the bandwidth and resolution of the spectrum. Each time the TIMER timeout event occurs, the hardware will automatically trigger the ADC to perform a sampling operation. The software will store the data into the buffer in the order of bit-reversal rule after each ADC conversion is completed.

This example code provides multiple look-up tables of the necessary parameters, which simplifies the computational complexity of the microcontroller, including the twiddle factors, which are required for the Fourier Transform process, and the bit-reversal ordering required for the Decimate-in-time approach. The developer has it calculated beforehand and defined in the header file. Since the content of the look-up table varies by the length of the signal, the look-up tables cannot be shared between different signal length settings. Therefore, the user must select a specified length parameter provided by the example code in the header file to compile. The process can be divided into five parts:

1. Specify the signal length and sampling frequency
2. System and peripheral hardware initialization
3. ADC samples specified numbers of data
4. Perform Fourier transform
5. Post processing to display spectrum information

1.3 Demo Result

1.3.1 Initial Screenshot

Wait for the user to confirm that the ADC input pin PD.2 has been connected to the signal source to be measured. Press any key to start sampling.

```
+------------------------------------------------------------+
|            Fast Fourier Transform Example Code             |
+------------------------------------------------------------+

# Demonstrate the decimate-in-time(DIT) radix-2 FFT
# This example code supports 3 kinds of data length:
    * 128 samples
    * 256 samples
    * 512 samples
# Please confirm the selected length of fft sequence is [512].
    Otherwise, change the fft sequence length settings and
    re-compile again to acquire the correct look-up table.
# Connect PD.2 to the signal source.
# Press any key to start ADC capturing....
```

Figure 1-1 Initial screenshot in Terminal

1.3.2 Time-domain Data Display

After the sampling process is completed, the time domain signal information is displayed, as shown in Figure 1-2. Due to the excessive number of data rows, only part of the data is displayed. The display content is divided into three columns. The left field is the original sampling order of the data, the middle field is the index of the data written to the buffer according to the bit-reversal table, and the right field is the processed ADC data which are stored to the buffer.

```
+------------------------------------------------------------+
| Display Time-Domain Data                                   +
+------------------------------------------------------------+
#Index before reversal    #Index after reversal    #ADC Data
 0                         0                        108
 1                         256                      87
 2                         128                      63
 3                         384                      41
 4                         64                       22
 5                         320                      9
 6                         192                      4
 7                         448                      7
 8                         32                       19
 9                         288                      37
 10                        160                      59
 11                        416                      82
 12                        96                       104
 13                        352                      120
 14                        224                      127
 15                        480                      127
 16                        16                       123
 17                        272                      109
 18                        144                      88
 19                        400                      65
```

Figure 1-2 Display part of the time-domain data in Terminal

1.3.3 Frequency-domain Data Display

Figure 1-3 shows part of the spectrum obtained after performing Fast Fourier Transform. The display content is divided into four columns, from left to right. The first column is the data index, the second column is the frequency indicated by that data point, and the third column is the intensity of the specific frequency component, contained in the original signal source, the unit of this column can be ignored. The last column shows the percentage of intensity that this frequency component occupies in the spectrum. By observing the last column, we can see that most of the signal component locates at the frequency of 60 Hz, in accordance with the 60 Hz sine wave given by the function generator.

```
+------------------------------------------------------------+
Spectrum Parameters
# Sample rate: 1024 Hz
# Data length: 512 samples
# Resolution : 2.00 Hz
+------------------------------------------------------------+
```

#Index	#Frequency(Hz)	#Amplitude	#Ratio
0	0.00	16900	0.00%
1	2.00	260	0.00%
2	4.00	3026	0.00%
3	6.00	2929	0.00%
4	8.00	340	0.00%
5	10.00	949	0.00%
6	12.00	2425	0.00%
7	14.00	712	0.00%
8	16.00	117	0.00%
9	18.00	457	0.00%
10	20.00	1060	0.00%
11	22.00	778	0.00%
12	24.00	949	0.00%
13	26.00	205	0.00%
14	28.00	648	0.00%
15	30.00	261	0.00%
16	32.00	17	0.00%
17	34.00	34	0.00%
18	36.00	221	0.00%
19	38.00	458	0.00%
20	40.00	232	0.00%
21	42.00	389	0.00%
22	44.00	1274	0.00%
23	46.00	292	0.00%
24	48.00	629	0.00%
25	50.00	449	0.00%
26	52.00	1481	0.00%
27	54.00	629	0.00%
28	56.00	2525	0.00%
29	58.00	2330	0.00%
30	60.00	255295080	49.95%
31	62.00	1732	0.00%
32	64.00	580	0.00%
33	66.00	45	0.00%
34	68.00	130	0.00%
35	70.00	229	0.00%
36	72.00	20	0.00%
37	74.00	50	0.00%
38	76.00	1	0.00%
39	78.00	106	0.00%
40	80.00	26	0.00%

Figure 1-3 Display part of the frequency-domain data in Terminal

1.3.4 Time-domain Data Plotting

Plot the entire time domain data to get the following figure 1-4, the horizontal axis is the index of sampling point, and the vertical axis is the ADC conversion value, showing a perfect sine wave.

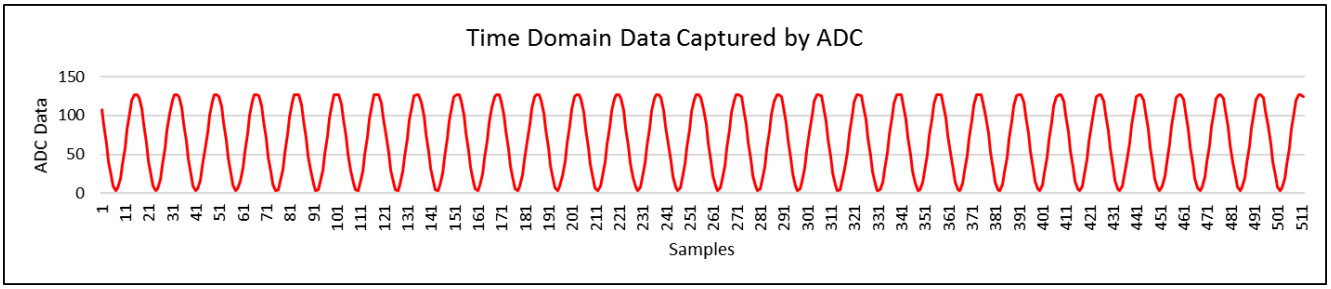


Figure1-4 Plot NUC121 ADC samples

1.3.5 Frequency-domain Data Plotting

Plot the entire frequency domain data to get the following figure 1-5. The left half and the right half of the spectrum will exhibit a conjugate symmetric distribution. Assuming that the sampling frequency configured by the user is F_s, the signal components distributed in the $F_s/2 \sim -F_s/2$ interval can be obtained, and the part of the negative frequency can be ignored. The horizontal axis represents the frequency and the vertical axis represents the intensity percentage. It can be clearly seen from the figure that the energy of the signal is concentrated on the coordinates of 60 Hz.

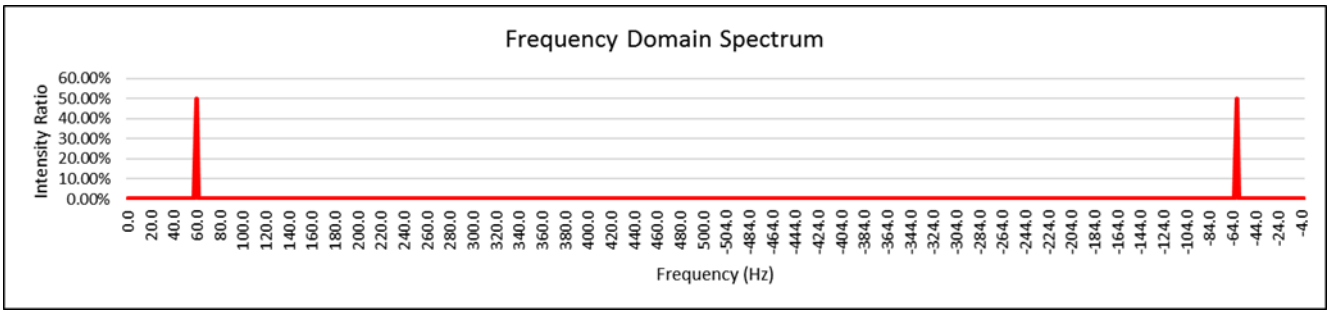


Figure 1-5 Plot Frequency-domain after FFT

2 Code Description

2.1 Specify the length of the signal

Define the length of the signal. This example provides 128, 256, and 512 samples. The longer the length, the higher the resolution of the spectrum, but the required resources and computation time will increase.

```
#ifndef __FFT_LUT_H__
#define __FFT_LUT_H__

/* This example provides 3 types selectable data length */
#define FFT_128_POINT               (0)
#define FFT_256_POINT               (1)
#define FFT_512_POINT               (2)

/* User can select one of the length setting here. */
#define FFT_N_POINT_SEL             (FFT_512_POINT)
```

2.2 Allocate the look-up table according to the length setting

A sine look-up table SIN_LUT, a cosine look-up table COS_LUT of the twiddle factors, and a bit-reversal table REV_LUT are provided. The twiddle factor, $exp(-j2\,\pi k/N)$, is the complex constant that the sampling point must multiply in the Fourier Transform, which can be disassembled into $\cos(2\pi k/N) - j\,\sin(2\pi k/N)$, representing the real part and the imaginary part respectively, and the constant values of their kth angle are given according to the previously defined signal length. The bit-reversal table is used to rearrange the original time domain data sequence. Based on the Decimate-in-time method, the time domain signal must be sorted by bit-reversal rule before the operation, so the data will be filled in the buffer at the specified address, which is based on the content of the table.

```
/* Assign the corresponding look-up table according to the length setting.
   Do not modify the contents here. */
#if (FFT_N_POINT_SEL == FFT_128_POINT)
#define SEQUENCE_LEN                (128)
#define RESCALE(x)                  (x>>7)
#define COS_LUT                     COS_SCALED_LUT_128
#define SIN_LUT                     SIN_SCALED_LUT_128
#define REV_LUT                     REV_LUT_128

#elif (FFT_N_POINT_SEL == FFT_256_POINT)
#define SEQUENCE_LEN                (256)
#define RESCALE(x)                  (x>>8)
```

```
#define COS_LUT                     COS_SCALED_LUT_256
#define SIN_LUT                     SIN_SCALED_LUT_256
#define REV_LUT                     REV_LUT_256

#elif (FFT_N_POINT_SEL == FFT_512_POINT)
#define SEQUENCE_LEN                (512)
#define RESCALE(x)                  (x>>9)
#define COS_LUT                     COS_SCALED_LUT_512
#define SIN_LUT                     SIN_SCALED_LUT_512
#define REV_LUT                     REV_LUT_512

#else
#define "configuration error"
#endif
```

2.3 Parameter Settings

The sampling frequency SAMPLE_RATE is configured by the user, and it is based on the application requirements. The SAMPLE_DATA_RSHIFT parameter is used to prevent the buffer overflow from the Fast Fourier Transform by right shifting the raw ADC data a specified numbers of bit before calculation. The value of this parameter is recommended to be reduced as much as possible without overflow, so that more features of the original data can be retained. This parameter can be configured by the user.

```
/* System clock frequency */
#define PLL_CLOCK                   (48000000)

/* The parameter SAMPLE_RATE is configurable. It is based on your application.
 * The unit of SAMPLE_RATE is hertz.
 */
#define SAMPLE_RATE                 (1024)

/* The parameter SAMPLE_DATA_RSHIFT is configurable, the purpose is to truncate
 * the data size by right shifting the raw ADC data, and it can prevent the buffer
 * from overflow when calculating a large amount N point of FFT. The longer data
 * length is selected, the higher SAMPLE_DATA_RSHIFT is required. If the spectrum
 * buffer is overflow, try to increase the SAMPLE_DATA_RSHIFT.
 */
#define SAMPLE_DATA_RSHIFT          (5)
#define SHRINK(x)                   ((x) >> SAMPLE_DATA_RSHIFT)
```

2.4 TIMER Initialization

Set the TIMER0 working frequency to the sampling rate and enable the TIMER trigger ADC function.

```
void TIMER0_Init(void)
{
    uint32_t u32Freq;

    /* Set TIMER0 working frequency to SAMPLE_RATE */
    u32Freq = TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, SAMPLE_RATE);

    if (u32Freq != SAMPLE_RATE)
    {
        printf("# Warning. Sample rate does not equal to the pre-defined setting.\n\r");
    }

    /* Enable timer timeout interrupt trigger ADC */
    TIMER0->CTL |= TIMER_TRGSRC_TIMEOUT_EVENT | TIMER_TRG_TO_ADC | TIMER_CTL_INTEN_Msk;
}
```

2.5 ADC Initialization

```
void ADC_Init(void)
{
    /* Set the ADC operation mode as single, input mode as single-end and enable the
       analog input channel 2 */
    ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_SINGLE, 0x1<<g_u8AdcChannel);

    /* Enable TIMER Trigger ADC */
    ADC_EnableTimerTrigger(ADC, ADC_ADCR_TRGS_TIMER, 0);

    /* Power on ADC module */
    ADC_POWER_ON(ADC);

    /* Clear the A/D interrupt flag for safe */
    ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);

    /* Enable the ADC interrupt */
    ADC_EnableInt(ADC, ADC_ADF_INT);
    NVIC_EnableIRQ(ADC_IRQn);
}
```

2.6 ADC Interrupt Service Routine

The converted ADC raw data is first reduced and then written to the buffer in the order of bit-reversal.

```
void ADC_IRQHandler(void)
{
    /* clear the A/D conversion flag */
    ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);

    /* Toggle PB.0 to indicate an ADC data conversion is complete */
    PB0^= 1;

    /* Place the time-domain signal into the real part buffer */
    g_i32RealBuffer [REV(g_u32AdcConvNum)] = \
                          SHRINK(ADC_GET_CONVERSION_DATA(ADC, g_u8AdcChannel));

    /* The image part buffer is initialized to zeros */
    g_i32ImageBuffer[REV(g_u32AdcConvNum)] = 0;

    /* Increase the buffer index by 1 */
    g_u32AdcConvNum++;
}
```

2.7 Fast Fourier Transform

After all the data sequence of the specified length have been sampled, it first removes the DC component (average value) of the signal and performs the FFT algorithm.

```
    /* Remove the DC component from the spectrum by zero-mean the time-domain signal before
       performing FFT. Otherwise, it will remain a high DC component at the 0 Hz point. */
    Remove_DC_Component(g_i32RealBuffer, SEQUENCE_LEN);

    /* Transform the time domain signal into frequency domain */
    NuFFT(g_i32RealBuffer, g_i32ImageBuffer, SEQUENCE_LEN);
```

2.7.1 Fast Fourier Transform Algorithm

```
void NuFFT(int * pi32RealBuffer, int * pi32ImageBuffer, uint32_t u32Len)
{
    const uint32_t N = u32Len;
    const uint32_t u32TotalStage = log(N) / log(2);

    uint32_t u32Stage, u32Span, u32Node, u32Twiddle, u32Angle;
    int32_t i32X1Real, i32X1Image, i32X2Real, i32X2Image;

    /* Iterations for log2(N) FFT butterfly stages */
    for (u32Stage=0; u32Stage<u32TotalStage; u32Stage++)
    {
        u32Span = pow(2, u32Stage);

        for (u32Twiddle=0; u32Twiddle<u32Span; u32Twiddle++)
        {
            u32Angle = (N>>1)/u32Span * u32Twiddle;

            for (u32Node=u32Twiddle; u32Node<N; u32Node+=2*u32Span)
            {
                /* Get the real and image part of the input variable X1 */
                i32X1Real  = pi32RealBuffer [u32Node];
                i32X1Image = pi32ImageBuffer[u32Node];

                /* Get the real and image part of the input variable X2 */
                i32X2Real  = pi32RealBuffer [u32Node+u32Span];
                i32X2Image = pi32ImageBuffer[u32Node+u32Span];

                /* Y1 = X1 + X2 * twiddle factor
                 * The real part is   real * cos() + image * sin()
                 * The image part is -real * sin() + image * cos()
                 */
                pi32RealBuffer [u32Node] = \
                          i32X1Real + RESCALE(i32X2Real * COS(u32Angle))
                                    + RESCALE(i32X2Image * SIN(u32Angle));
                pi32ImageBuffer[u32Node] = \
                          i32X1Image - RESCALE(i32X2Real * SIN(u32Angle))
                                     + RESCALE(i32X2Image * COS(u32Angle))
```

```
                /* Y2 = X1 - X2 * twiddle factor
                 * The real part is  -real * cos() - image * sin()
                 * The image part is  real * sin() - image * cos()
                 */
                pi32RealBuffer [u32Node+u32Span] = \
                          i32X1Real  - RESCALE(i32X2Real * COS(u32Angle))
                                     - RESCALE(i32X2Image * SIN(u32Angle));
                pi32ImageBuffer[u32Node+u32Span] = \
                          i32X1Image + RESCALE(i32X2Real * SIN(u32Angle))
                                     - RESCALE(i32X2Image * COS(u32Angle));
            }
        }
    }
}
```

2.8 Post Processing

Calculate and display the frequency domain information. Program termination.

```
 for (i=0; i<SEQUENCE_LEN ;i++)
    {
        /* The amplitude of the spectrum could be presented as the sum of square of the
           real part and the complex part. */
        g_i32Spectrum[i] = (((g_i32RealBuffer[i] * g_i32RealBuffer[i]) +
                            (g_i32ImageBuffer[i] * g_i32ImageBuffer[i])));
    }

    Display_Spectrum(g_i32Spectrum, SEQUENCE_LEN, SAMPLE_RATE);

    while(1);
```

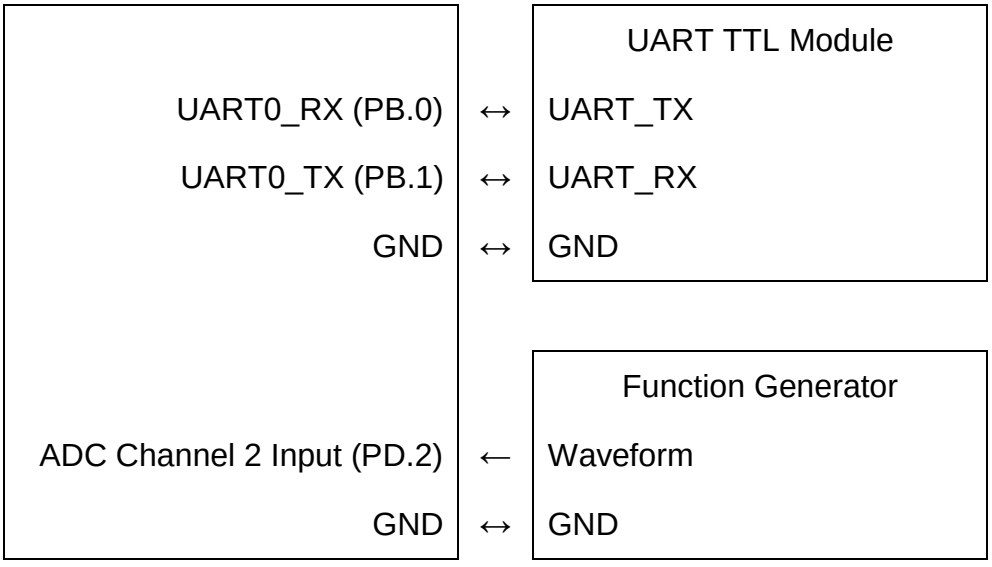
3 Software and Hardware Environment

Software Environment

- BSP version
  - NUC121_125_Series_BSP_CMSIS_V3.01.003
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuTiny-SDK-NUC121_V1.0 Developer Board
  - USB-UART-TTL Module (Optional)
  - Function generator (Optional)
- Diagram



4 Directory Information

- EC_NUC121_FFT_Demo_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 NUC121_FFT_Demo.uvprojx.
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
Sep. 30, 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.