**Data Acquisition System Design and FFT analysis using ARM Cortex-M4 Microcontroller**

**1. Aim:**

To perform Fast Fourier Transform (FFT) on a current signal obtained from ADC.

**2. Introduction:**

A system that collects information is called a data acquisition system (DAS). In this experiment, we will use the two terms DAS and instrument interchangeably. Sometimes the acquisition of data is the fundamental purpose of the system, such as with a voltmeter, a tachometer, a multi-meter, an audio recorder, or a camera. At other times, the acquisition of data is an integral part of a larger system such as a control system or communication system.

The measurand is the physical quantity, property, or condition that the instrument measures. The measurand can be inherent to the object (like position, size, mass, or color), located on the surface of the object (like the human EKG or surface temperature), located within the object (e.g., fluid pressure or internal temperature), or separated from the object (like emitted radiation). A typical DAS data flow graph is given in the figure below.

A picture containing text, diagram, screenshot, font

Description automatically generated

**Data flow graph of a DAS**

The transducer converts the physical signal into an electric signal. The amplifier converts the weak transducer electric signal into the range of the ADC (e.g., 0 to 3 V). The analog filter removes unwanted frequency components within the signal. The analog filter is required to remove aliasing error caused by the ADC sampling. The analog multiplexer is used to select one signal from many sources. The S/H is an analog latch used to keep the ADC input voltage constant during the ADC conversion. The clock is used to control the sampling process. Inherent in digital signal processing is the requirement that the ADC be sampled on a fixed time basis. The computer is used to save and process the digital data. A digital filter may be used to amplify or reject certain frequency components of the digitized signal. Block diagram of a multichannel ADC is given in the figure below.

A picture containing diagram, text, line, plan

Description automatically generated

**Multichannel ADC**

**3. Determination of Sampling Rate:**

There are two errors introduced by the sampling process. Voltage quantizing is caused by the finite word size of the ADC. The precision is determined by the number of bits in the ADC. If the ADC has n bits, then the number of distinguishable alternatives is

Time quantizing is caused by the finite discrete sampling interval. Nyquist theory states that if the signal is sampled at Fs, then the digital samples contain frequency components from only 0 to 0.5 Fs. Conversely, if the analog signal does contain frequency components larger than 0.5 Fs, then there will be an aliasing error. Aliasing is when the digital signal appears to have a different frequency than the original analog signal. Simply put, if one samples a sine wave at a sampling rate of Fs,

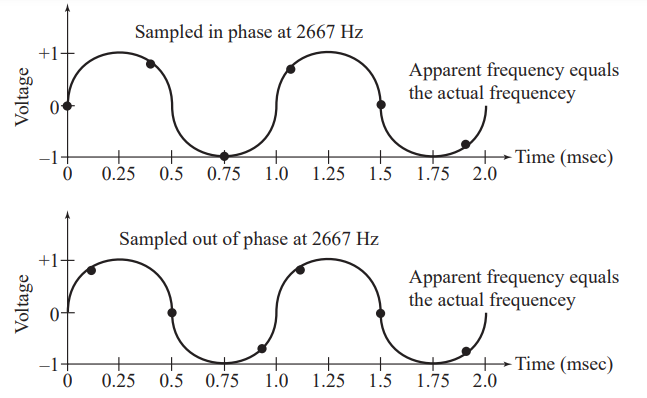
is it possible to determine A, f, and Φ from the digital samples? Nyquist theory says that if Fs is strictly greater than twice f, then one can determine A, f, and Φ from the digital samples. In other words, the entire analog signal can be reconstructed from the digital samples. But if Fs is less than or equal to twice f, then one cannot determine A, f, and Φ. In this case, the apparent frequency, as predicted by analyzing the digital samples, will be shifted to a frequency between 0 and 0.5 Fs.

**Example illustration:**

A picture containing text, diagram, line, plot

Description automatically generated

**Aliasing due to improper Fs**



**Sampling without aliasing**

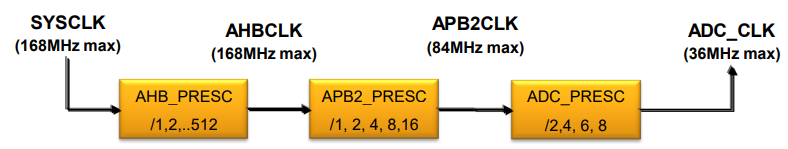
**4. STM32F407 ADC Features:**

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels allowing it to measure signals from 16 external sources, two internal sources, and the VBAT channel. The A/D conversion of the channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored into a left or right-aligned 16-bit data register.

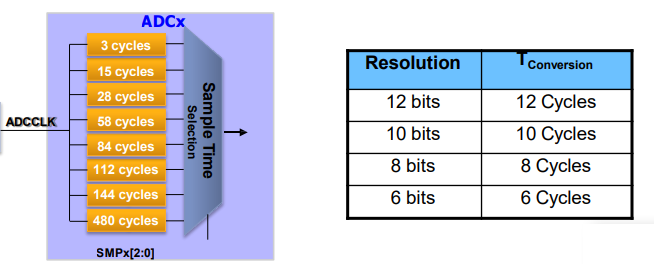
* 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
* Interrupt generation at the end of conversion
* Single and continuous conversion modes
* Channel-wise programmable sampling time
* ADC input range: VREF– ≤ VIN ≤ VREF+

**5. Procedure for 8000Hz Sampling Rate Configuration of ADC:**

Assume that we have configured our ADC as 8-bit. The ADC input clock is generated from the PCLK2 clock divided by a prescaler.



Total Conversion Time = Tsampling + Tconversion



For 8000 sampling rate,

Since we have configured it as 8-bit, Tconversion is 8 cycles. We can choose T sampling time as 112 cycles. Here total conversion time is 120cycles. Assume that ADC\_CLK is 1MHz with suitable prescaler values. Hence,

Sampling rate =1000000/120=8333Hz.

**5.1 Program for Current Signal Acquisition using ADC with 8000Hz Sampling Frequency:**

#include "stm32f4xx.h"

void ADC\_Init(void);

void ADC\_Enable(void);

void ADC\_Start(int);

float value[1024];

uint32\_t i,j;

uint8\_t ADC\_VAL[1024];

void ADC\_Init (void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* STEPS TO FOLLOW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Enable ADC and GPIO clock

2. Set the prescalar in the Common Control Register (CCR)

3. Set the Resolution in the Control Register 1 (CR1)

4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

5. Set the Sampling Time for the channels in ADC\_SMPRx

6. Set the Regular channel sequence length in ADC\_SQR1

7. Set the Respective GPIO PINs in the Analog Mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//1. Enable ADC and GPIO clock

RCC->APB2ENR |= (1<<8); // enable ADC1 clock

RCC->AHB1ENR |= (1<<0); // enable GPIOA clock

RCC->CFGR |= 6<<13; // set APB2 = 2 MHz

//2. Set the pre-scalar in the Common Control Register (CCR)

ADC->CCR |= 0<<16; // PCLK2 divide by 4, works at 4MHz

//3. Set the Resolution in the Control Register 1 (CR1)

//ADC1->CR1 &= ~(1<<8); // SCAN mode disabled, enable for multichannel use

ADC1->CR1 |= (2<<24); // 8 bit RES

//4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

//ADC1->CR2 |= (1<<1); // enable continuous conversion mode

ADC1->CR2 |= (1<<10); // EOC after each conversion

ADC1->CR2 &= ~(1<<11); // Data Alignment RIGHT

//5. Set the Sampling Time for the channels

//ADC1->SMPR2 &= ~(1<<0); // Sampling time of 3 cycles for channel 0

ADC1->SMPR2|=(7<<0);

//6. Set the Regular channel sequence length in ADC\_SQR1

//ADC1->SQR1 &= ~(1<<20); // SQR1\_L =0 for 1 conversion

//7. Set the Respective GPIO PIN in the Analog Mode

GPIOA->MODER |= (3<<0); // analog mode for PA 0 (channel 0)

}

void msDelay(uint32\_t msTime)

{

/\* For loop takes 4 clock cycles to get executed. Clock frequency of stm32f407 by default is 16MHz

So, 16MHz/4=4MHz. If we want 1000ms delay, 4MHz/1000=4000, so we have to multiply by 4000 to get a delay of 1s

\*/

for(uint32\_t i=0;i<msTime\*3000;i++)

{

\_\_NOP();

}

}

int main ()

{

ADC\_Init ();

ADC1->CR2 |= 1<<0; // ADON =1 enable ADC1

uint32\_t delay = 10000;

while (delay--);//Wait sometime for ADC to start

//ADC1->CR2 |= (1<<30); // start the conversion

while (1)

{

for(i=0;i<1024;i++)

{

ADC1->CR2 |= (1<<30); // start the conversion

ADC1->SR = 0; // clear the status register

//ADC1->CR2 |= (1<<30); // start the conversion

while (!(ADC1->SR & (1<<1))); // wait for EOC flag to set

ADC\_VAL[i]=ADC1->DR;

}

for(j=0;j<1024;j++)

{

value[j]=ADC\_VAL[j];

}

}

}

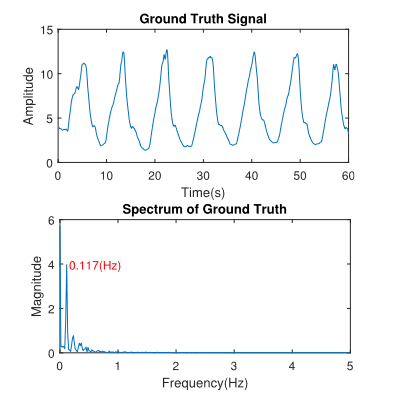
**5.2 Program for Current Signal Acquisition using ADC with 8000Hz Sampling Frequency and real-time FFT Computation:**

**Fast Fourier Transform:**

FFT equation is employed to extract the dominant frequency of the given signal:

FFT resolution is computed to obtain the analog frequency of corresponding dominant peak:

where Fs is the sampling rate of resampled signal, N is the size FFT, and ΔF is the FFT resolution. Real-time computation of FFT for a given breathe signal is illustrated below.



# To implement real-time FFT in ARM Cortex-M4 microcontroller, APIs from CMSIS-DSP library is used. This user manual describes the CMSIS DSP software library, a suite of common signal processing functions for use on Cortex-M and Cortex-A processor-based devices.

The library is divided into a number of functions each covering a specific category:

* Basic math functions
* Fast math functions
* Complex math functions
* Filtering functions
* Matrix functions
* Transform functions
* Motor control functions
* Statistical functions
* Support functions
* Interpolation functions
* Support Vector Machine functions (SVM)
* Bayes classifier functions
* Distance functions

The library has generally separate functions for operating on 8-bit integers, 16-bit integers, 32-bit integer and 32-bit floating-point values. The library functions are declared in the public file [arm\_math.h](https://www.keil.com/pack/doc/CMSIS/DSP/html/arm__math_8h.html) which is placed in the Include folder. Simply include this file. If you don't want to include everything, you can also rely on headers in Include/dsp folder and use only what you need.

A screenshot of a computer program

Description automatically generated with medium confidence

#include "stm32f4xx.h"

#define ARM\_MATH\_CM4

#include "arm\_math.h"

#include "arm\_const\_structs.h"

void ADC\_Init(void);

void ADC\_Enable(void);

void ADC\_Start(int);

float32\_t testOutput2[1024];

float32\_t testMag2[512],value[1024],value1[1024],Frequency,maxValue;

float32\_t delF=2051/1024;//Fs/N

uint32\_t fftSize = 1024,maxIndex;

uint32\_t ifftFlag = 0,i,j;

uint32\_t doBitReverse = 1;

uint8\_t ADC\_VAL[1024];

void ADC\_Init (void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* STEPS TO FOLLOW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Enable ADC and GPIO clock

2. Set the prescalar in the Common Control Register (CCR)

3. Set the Resolution in the Control Register 1 (CR1)

4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

5. Set the Sampling Time for the channels in ADC\_SMPRx

6. Set the Regular channel sequence length in ADC\_SQR1

7. Set the Respective GPIO PINs in the Analog Mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//High speed internal clock is 16MHz

//1. Enable ADC and GPIO clock

RCC->APB2ENR |= (1<<8); // enable ADC1 clock

RCC->AHB1ENR |= (1<<0); // enable GPIOA clock

RCC->CFGR |= 6<<13; // AHB clock divided by 8; set APB2 = 2 MHz

//2. Set the pre-scalar in the Common Control Register (CCR)

ADC->CCR |= 0<<16; //00: PCLK2 divided by 2, works at 1MHz

//3. Set the Resolution in the Control Register 1 (CR1)

//ADC1->CR1 &= ~(1<<8); // SCAN mode disabled, enable for multichannel use

ADC1->CR1 |= (2<<24); // 8 bit RES

//4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

//ADC1->CR2 |= (1<<1); // enable continuous conversion mode

ADC1->CR2 |= (1<<10); // EOC after each conversion

ADC1->CR2 &= ~(1<<11); // Data Alignment RIGHT

//5. Set the Sampling Time for the channels

//ADC1->SMPR2 &= ~(1<<0); // Sampling time of 3 cycles for channel 0

ADC1->SMPR2|=(7<<0);// 480+8cycles (number of bits)=488 cycles

//6. Set the Regular channel sequence length in ADC\_SQR1

//ADC1->SQR1 &= ~(1<<20); // SQR1\_L =0 for 1 conversion

//7. Set the Respective GPIO PIN in the Analog Mode

GPIOA->MODER |= (3<<0); // analog mode for PA 0 (channel 0)

}

void msDelay(uint32\_t msTime)

{

/\* For loop takes 4 clock cycles to get executed. Clock frequency of stm32f407 by default is 16MHz

So, 16MHz/4=4MHz. If we want 1000ms delay, 4MHz/1000=4000, so we have to multiply by 4000 to get a delay of 1s

\*/

for(uint32\_t i=0;i<msTime\*3000;i++)

{

\_\_NOP();

}

}

int main ()

{

//FPU enable

SCB->CPACR|=(0xF<<20); //Co-processor (core peripheral) access control register

ADC\_Init ();

ADC1->CR2 |= 1<<0; // ADON =1 enable ADC1

uint32\_t delay = 10000;

while (delay--);//Wait sometime for ADC to start

arm\_rfft\_fast\_instance\_f32 S2;

arm\_rfft\_fast\_init\_f32(&S2,1024);

while (1)

{

for(i=0;i<1024;i++)

{

ADC1->CR2 |= (1<<30); // start the conversion

ADC1->SR = 0; // clear the status register

//ADC1->CR2 |= (1<<30); // start the conversion

while (!(ADC1->SR & (1<<1))); // wait for EOC flag to set

ADC\_VAL[i]=ADC1->DR;

}

for(j=0;j<1024;j++)

{

value1[j]=ADC\_VAL[j];

}

for(j=0;j<1024;j++)

{

value[j]=ADC\_VAL[j];

}

arm\_rfft\_fast\_f32 (&S2,value,testOutput2,ifftFlag);

arm\_cmplx\_mag\_f32(testOutput2, testMag2, fftSize);

arm\_max\_f32(testMag2+1, 511, &maxValue, &maxIndex);

Frequency=delF\*(maxIndex+1);

}

}

**6. Conclusion:**

The current signal obtained from the ADC has been transformed to Frequency Domain by performing FFT on it. Also, in addition to this, by analyzing the spectrum of the current signal, the most dominant signal frequency has been found.

**7. References:**

1. ARM CMSIS Library: <https://www.keil.com/pack/doc/CMSIS/General/html/index.html>