Electronics The King of Hobbies!

Implementing Discrete Fourier Transform in Atmega32 to make an audio spectrum analyzer

"All waveforms, no matter what you scribble or observe in the universe, are actually just the sum of simple sinusoids of different frequencies."

Hi,
I am just refreshing the basics of fourier transform. I am not an expert. Now I did a small audio spectrum analyzer(0 - 10KHz) on a 16x2 character lcd using an atmega32 microcontroller. Since I am refreshing from the basics, so I started with simple DFT. Also, I believe I should learn to walk before running. So I am not straight away going towads the FFT, which is nothing but the fastest and a bit complicated algorithm to find DFT.(I will try it later, as soon as possible)

DFT is too slow compared to FFT. My lcd spectum analyzer doesn't need a great speed like that of an FFT, now if it is capable of providing a speed of around 30 frame/second, then it is more than enough for visualizing the audio spectrum on an LCD. But any way, in my case I can roughly achieve around 100 frames/second(any way it is too high refresh rate for a 16x2 lcd, not recommended also :-)). My audio sampling rate is 20KHz for 32 point DFT. Since the transform result is symmetric, I need to use only the first half, ie the first 16 results. So, it means, it can display upto 10KHz spectrum. So the resolution is 10KHz/16 = 625Hz.

I have tried to improve the speed of DFT computation. If it is an N point DFT, it needs to find (N^2)/2 sin and cos values. For a 32 point DFT, it needs to find 512 sine and cosine. Before finding the sine and cosine, we need to find the angle(degree) which takes some processor time, so I implemented a lookup table for that. Next two tables are for sine and cosine. I didn't used any float or double since it takes more processing time in 8 bit avr, instead I implemented the sine and cosine lookups as 16bit intiger, by multiplying the real sine and cosine values by 10000. Then after finding the transform, finally I need to divide each result by 10000. This eliminates the need of using float or double and makes it more faster. *Now I can calculate 120 32-point DFT operation/sec which is more than enough for my small spectrum analyzer.*

LCD

Now, looking towards the LCD side, I utilized the custom character feature of LCD to make 8 stacked horizontal bars which takes the entire 64bytes of the LCD RAM for custom character bitmap. *I ones seen a video is hackaday.com that a person used a 16x2 lcd in the similar manner for his spectrum analyser.* So I also adopted the same idea of using the custom character for my spectrometer.

AUDIO INPUT

Now one of the most important part of this stuff is the audio sampling via an eletret microphone. Special care must be given while designing the pre-amp for the mic. We need to set the zero-level of the ADC input to exactly half of the ADC reference voltage ie to 2.5v. Now it can have positive and negative swing on this 2.5v level according to the input audio signal but it should not cross the limit ie the amplifier gain should be properly adjusted to prevent clipping. I am using an LM324 op-amp for the mic pre-amp to meet the above conditions.

PHOTO

LABELS

MY HOBBY PROJECTS (26) AVR (14) PIC (10) PYTHON (9) msp430 launchpad (5) Assembly language (4) ARM (3) bit banging (3) Linux (2) multitasking (2) simple DSP experiments (2) 8051-Compatible Microcontrollers (1) hack (1) robotics (1)

Blog Archive

Blog Archive

Widgetized

Total Pageviews

About Me

Vinod.S

Hi, I am an electronics hobbyist from Kerala(India).

View my complete profile

EVERGREEN POSTS

Scrolling text in LED dotmatrix display

Introduction: Multiplexed displays are electronic displays where the entire display is not driven at one time. Instead, sub-units of the di...

4 bit interfacing of a 16X2 LCD display to PIC16F877A, Atmega16/32, MSP430 & Stellaris launchpad

16x2 LCDs are most commonly used display units in microcontroller based projects. I got much information about LCD, LCD commands, LCD initi...

An attempt to access a memory card (MMC) using a PIC with limited RAM (PIC16F877A)

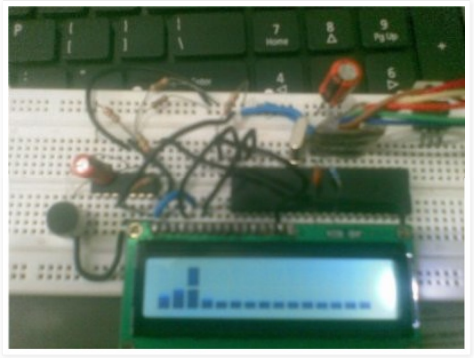
PIC16F877A BASED MMC VOICE RECORDER
Video of my PIC16F877A based MMC digital voice recorder: ...

Implementing Discrete Fourier Transform in Atmega32 to make an audio spectrum analyzer

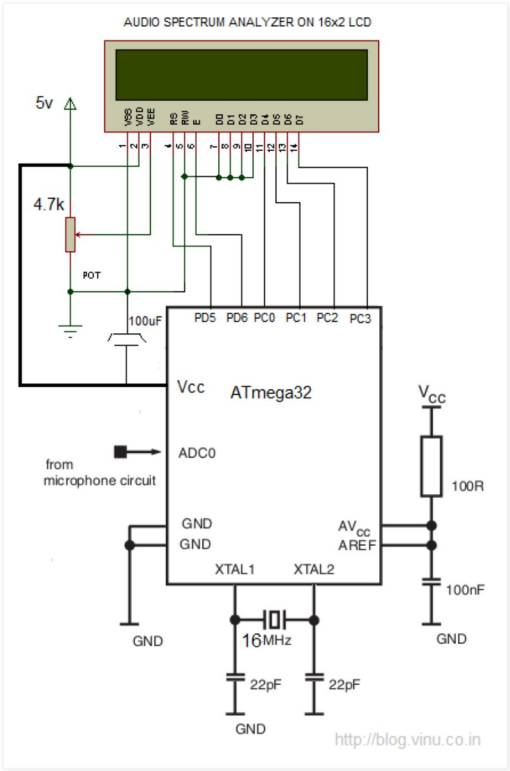
"All waveforms, no matter what you scribble or observe in the universe, are actually just the sum of simple sinusoids of differ...

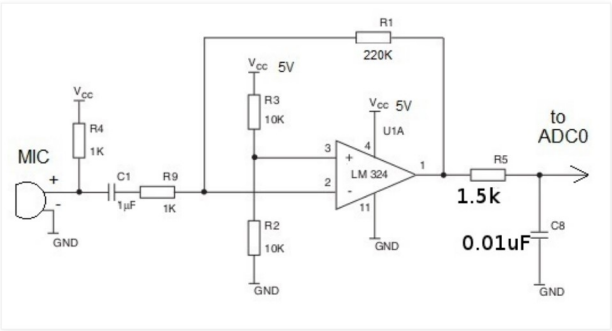
TV remote controller 160KHz High Quality Stereo MMC WAV player using ATMEGA32

(updated the complete source code + makefile + hex + asm + .out files on bitbucket repository) link is provided below the source co...



CIRCUIT DIAGRAM





SOURCE CODE: (main.c)

```
#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>
#define N 32
#include "lookup.h"
#define RS PD5
#define EN PD6
#define LCD_NIBBLE PORTC

void adc_init();
uint16_t adc_read();
void TRANSFORM();
void timer1_init();
void LCD_STROBE(void);
void lcd_data(unsigned char c);
void lcd_cmd(unsigned char c);
```

stm32f407 discovery board as a 100MHz FM transmitter using crystal feedback trick

Hi All, It's been a long time I haven't updated my blog coz it is hard to get some free time. Anyways, I have a small hack...

Resistive touch screen based wireless mouse

Hi, after a long time I am updating my blog again. I bought few nokia color LCDs and resistive touchpad last month, all are chinese clones ...

Happy Christmas and Happy New Year from Attiny13

While trying to open a chinese camera pen, unfortunately the PCB inside it got damaged. Few of the PCB traces got cut and it bec...

Playing video on nokia color LCD just using an 8 bit AVR! [A mad Project ;)]

A MaD PROjECt....;-) SD CARD + ATMEGA32 + NOKIA COLOR LCD = VIDEO PLAYER!!! Hi, I am introducing my new video player made using a...

Generating AUDIO ECHO using Atmega32 microcontroller

(video demo of echo generation using atmega32) Introduction: Hi, While I was studying at 10th standard, I used to play with small e...

Recent visitors

```
void lcd_clear(void);
void lcd_init();
void lcd_print(char *p, char l);
void lcd_fill_custom();

uint8_t lcd_buf1[16];
uint8_t lcd_buf2[16];
int32_t fx[N];
int32_t Fu[N/2][2];

void main()
{
    uint8_t mag;
    int i,j, temp_value;
    uint8_t temp_index;
    adc_init();
    lcd_init();
    lcd_fill_custom();
    lcd_print("DFT SPECTROMETER",1);
    lcd_print("0Hz - 10KHz(16)",2);
    _delay_ms(5000);
    lcd_clear();
    timer1_init();
    while(1) {
        TCNT1 = 0;
        TIFR |= 1<<OCF1A;
        for(i=0;i<N;i++) {
            while((TIFR & (1<<OCF1A)) == 0);
            fx[i] = ((int16_t)adc_read());
            TIFR |= 1<<OCF1A;
        }
        TRANSFORM();
        lcd_cmd(0xc0);
        for(i =1; i<N/2; i++) {
            if(Fu[i][0]<0)Fu[i][0]*=-1;
            if(Fu[i][1]<0)Fu[i][1]*=-1;
            mag = (uint8_t)(Fu[i][0] + Fu[i][1])/4;
            if((mag)>7) {
                lcd_buf1[i] = (mag) - 7 - 1;
                if(lcd_buf1[i] > 7)
                lcd_buf1[i] = 7;
                lcd_buf2[i] = 7;
            }
            else {
                lcd_buf1[i] = ' ';
                lcd_buf2[i] = mag;
            }
        }
        lcd_cmd(0x80);
        for(i=1;i<16;i++)
        lcd_data(lcd_buf1[i]);
        lcd_cmd(0xc0);
        for(i=1;i<16;i++)
        lcd_data(lcd_buf2[i]);


    }
}

void TRANSFORM()
{
    int16_t count,degree;
    uint8_t u,k;
    count = 0;
    for (u=0; u<N/2; u++) {
        for (k=0; k<N; k++) {
            degree = (uint16_t)pgm_read_byte_near(degree_lookup + count)*2;
            count++;
            Fu[u][0] +=  fx[k] * (int16_t)pgm_read_word_near(cos_lookup + degree);
            Fu[u][1] += -fx[k] * (int16_t)pgm_read_word_near(sin_lookup + degree);
        }
        Fu[u][0] /= N;
        Fu[u][0] /= 10000;
        Fu[u][1] /= N;
        Fu[u][1] /= 10000;
    }
}

void timer1_init()
{
    TCCR1B = (1<<WGM12)|(1<<CS10);
    OCR1A = 800;
}
```

```
void adc_init()
{
    ADMUX = 0b11000000;
    ADCSRA =0b10000010;
}

uint16_t adc_read()
{
    volatile uint16_t retl,reth;
    ADCSRA |= 1<<ADSC;
    while(!ADIF);
    ADCSRA |= 1<<ADIF;
    retl = ADCL;
    reth = ADCH;
    reth<<=8;
    reth|=retl;
    return reth;
}

void LCD_STROBE(void)
{
    PORTD |= (1 << EN);
    _delay_us(1);
    PORTD &= ~(1 << EN);
}

void lcd_data(unsigned char c)
{
    PORTD |= (1 << RS);
    _delay_us(50);
    LCD_NIBBLE = (c >> 4)|(LCD_NIBBLE&0xf0);
    LCD_STROBE();
    LCD_NIBBLE = (c)|(LCD_NIBBLE&0XF0);
    LCD_STROBE();
}

void lcd_cmd(unsigned char c)
{
    PORTD &= ~(1 << RS);
    _delay_us(50);
    LCD_NIBBLE = (c >> 4)|(LCD_NIBBLE&0xf0);
    LCD_STROBE();
    LCD_NIBBLE = (c)|(LCD_NIBBLE&0XF0);
    LCD_STROBE();
}

void lcd_clear(void)
{
    lcd_cmd(0x01);
    _delay_ms(5);
}

void lcd_init()
{
    DDRC = 0b00001111;
    DDRD |= (1 << RS)|(1 << EN);
    PORTC |= (1<<PC4);
    _delay_ms(15);
    lcd_cmd(0x30);
    _delay_ms(1);
    lcd_cmd(0x30);
    _delay_us(100);
    lcd_cmd(0x30);
    lcd_cmd(0x28);
    lcd_cmd(0x28);
    lcd_cmd(0x0c);
    lcd_clear();
    lcd_cmd(0x6);
}

void lcd_print(char *p, char l)
{
    if(l==1)lcd_cmd(0x80);
    else lcd_cmd(0xc0);
    while(*p)
    lcd_data(*p++);
}

void lcd_fill_custom()
{
    uint8_t i,j;
    i=0;j=0;
```

```
    lcd_cmd(64);
    for(i=1;i<=8;i++) {
        for(j=8;j>i;j--)
        lcd_data(0);
        for(j=i;j>0;j--)
        lcd_data(0xff);
    }
}
```

lookup.h

```
//LOOKUP TABLE//
#include <avr/pgmspace.h>

PROGMEM const int16_t cos_lookup[]= {
    10000,9998,9993,9986,9975,9961,9945,9925,9902,
    9876,9848,9816,9781,9743,9702,9659,9612,9563,
    9510,9455,9396,9335,9271,9205,9135,9063,8987,
    8910,8829,8746,8660,8571,8480,8386,8290,8191,
    8090,7986,7880,7771,7660,7547,7431,7313,7193,
    7071,6946,6819,6691,6560,6427,6293,6156,6018,
    5877,5735,5591,5446,5299,5150,5000,4848,4694,
    4539,4383,4226,4067,3907,3746,3583,3420,3255,
    3090,2923,2756,2588,2419,2249,2079,1908,1736,
    1564,1391,1218,1045,871,697,523,348,174,
    0,-174,-348,-523,-697,-871,-1045,-1218,-1391,
    -1564,-1736,-1908,-2079,-2249,-2419,-2588,-2756,-2923,
    -3090,-3255,-3420,-3583,-3746,-3907,-4067,-4226,-4383,
    -4539,-4694,-4848,-4999,-5150,-5299,-5446,-5591,-5735,
    -5877,-6018,-6156,-6293,-6427,-6560,-6691,-6819,-6946,
    -7071,-7193,-7313,-7431,-7547,-7660,-7771,-7880,-7986,
    -8090,-8191,-8290,-8386,-8480,-8571,-8660,-8746,-8829,
    -8910,-8987,-9063,-9135,-9205,-9271,-9335,-9396,-9455,
    -9510,-9563,-9612,-9659,-9702,-9743,-9781,-9816,-9848,
    -9876,-9902,-9925,-9945,-9961,-9975,-9986,-9993,-9998,
    -10000,-9998,-9993,-9986,-9975,-9961,-9945,-9925,-9902,
    -9876,-9848,-9816,-9781,-9743,-9702,-9659,-9612,-9563,
    -9510,-9455,-9396,-9335,-9271,-9205,-9135,-9063,-8987,
    -8910,-8829,-8746,-8660,-8571,-8480,-8386,-8290,-8191,
    -8090,-7986,-7880,-7771,-7660,-7547,-7431,-7313,-7193,
    -7071,-6946,-6819,-6691,-6560,-6427,-6293,-6156,-6018,
    -5877,-5735,-5591,-5446,-5299,-5150,-5000,-4848,-4694,
    -4539,-4383,-4226,-4067,-3907,-3746,-3583,-3420,-3255,
    -3090,-2923,-2756,-2588,-2419,-2249,-2079,-1908,-1736,
    -1564,-1391,-1218,-1045,-871,-697,-523,-348,-174,
    0,174,348,523,697,871,1045,1218,1391,
    1564,1736,1908,2079,2249,2419,2588,2756,2923,
    3090,3255,3420,3583,3746,3907,4067,4226,4383,
    4539,4694,4848,5000,5150,5299,5446,5591,5735,
    5877,6018,6156,6293,6427,6560,6691,6819,6946,
    7071,7193,7313,7431,7547,7660,7771,7880,7986,
    8090,8191,8290,8386,8480,8571,8660,8746,8829,
    8910,8987,9063,9135,9205,9271,9335,9396,9455,
    9510,9563,9612,9659,9702,9743,9781,9816,9848,
    9876,9902,9925,9945,9961,9975,9986,9993,9998
};

PROGMEM const int16_t sin_lookup[]= {
    0,174,348,523,697,871,1045,1218,1391,
    1564,1736,1908,2079,2249,2419,2588,2756,2923,
    3090,3255,3420,3583,3746,3907,4067,4226,4383,
    4539,4694,4848,4999,5150,5299,5446,5591,5735,
    5877,6018,6156,6293,6427,6560,6691,6819,6946,
    7071,7193,7313,7431,7547,7660,7771,7880,7986,
    8090,8191,8290,8386,8480,8571,8660,8746,8829,
    8910,8987,9063,9135,9205,9271,9335,9396,9455,
    9510,9563,9612,9659,9702,9743,9781,9816,9848,
    9876,9902,9925,9945,9961,9975,9986,9993,9998,
    10000,9998,9993,9986,9975,9961,9945,9925,9902,
    9876,9848,9816,9781,9743,9702,9659,9612,9563,
    9510,9455,9396,9335,9271,9205,9135,9063,8987,
    8910,8829,8746,8660,8571,8480,8386,8290,8191,
    8090,7986,7880,7771,7660,7547,7431,7313,7193,
    7071,6946,6819,6691,6560,6427,6293,6156,6018,
    5877,5735,5591,5446,5299,5150,4999,4848,4694,
    4539,4383,4226,4067,3907,3746,3583,3420,3255,
    3090,2923,2756,2588,2419,2249,2079,1908,1736,
    1564,1391,1218,1045,871,697,523,348,174,
    0,-174,-348,-523,-697,-871,-1045,-1218,-1391,
    -1564,-1736,-1908,-2079,-2249,-2419,-2588,-2756,-2923,
    -3090,-3255,-3420,-3583,-3746,-3907,-4067,-4226,-4383,
    -4539,-4694,-4848,-4999,-5150,-5299,-5446,-5591,-5735,
    -5877,-6018,-6156,-6293,-6427,-6560,-6691,-6819,-6946,
    -7071,-7193,-7313,-7431,-7547,-7660,-7771,-7880,-7986,
```

```
    -8090,-8191,-8290,-8386,-8480,-8571,-8660,-8746,-8829,
    -8910,-8987,-9063,-9135,-9205,-9271,-9335,-9396,-9455,
    -9510,-9563,-9612,-9659,-9702,-9743,-9781,-9816,-9848,
    -9876,-9902,-9925,-9945,-9961,-9975,-9986,-9993,-9998,
    -10000,-9998,-9993,-9986,-9975,-9961,-9945,-9925,-9902,
    -9876,-9848,-9816,-9781,-9743,-9702,-9659,-9612,-9563,
    -9510,-9455,-9396,-9335,-9271,-9205,-9135,-9063,-8987,
    -8910,-8829,-8746,-8660,-8571,-8480,-8386,-8290,-8191,
    -8090,-7986,-7880,-7771,-7660,-7547,-7431,-7313,-7193,
    -7071,-6946,-6819,-6691,-6560,-6427,-6293,-6156,-6018,
    -5877,-5735,-5591,-5446,-5299,-5150,-5000,-4848,-4694,
    -4539,-4383,-4226,-4067,-3907,-3746,-3583,-3420,-3255,
    -3090,-2923,-2756,-2588,-2419,-2249,-2079,-1908,-1736,
    -1564,-1391,-1218,-1045,-871,-697,-523,-348,-174
};

PROGMEM const uint8_t degree_lookup[]= {
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,5,11,16,22,28,33,39,45,50,56,61,67,
    73,78,84,90,95,101,106,112,118,123,129,135,140,146,151,
    157,163,168,174,0,11,22,33,45,56,67,78,90,101,112,
    123,135,146,157,168,0,11,22,33,44,56,67,78,90,101,
    112,123,134,146,157,168,0,16,33,50,67,84,101,118,135,
    151,168,5,22,39,56,73,90,106,123,140,157,174,11,28,
    45,61,78,95,112,129,146,163,0,22,45,67,90,112,135,
    157,0,22,44,67,90,112,134,157,0,22,45,67,89,112,
    134,157,0,22,45,67,89,112,135,157,0,28,56,84,112,
    140,168,16,44,73,101,129,157,5,33,61,89,118,146,174,
    22,50,78,106,135,163,11,39,67,95,123,151,0,33,67,
    101,135,168,22,56,90,123,157,11,45,78,112,146,0,33,
    67,101,135,168,22,56,90,123,157,11,44,78,112,146,0,
    39,78,118,157,16,56,95,134,174,33,73,112,151,11,50,
    89,129,168,28,67,106,146,5,44,84,123,163,22,61,101,
    140,0,45,90,135,0,44,90,134,0,45,89,134,0,45,
    89,135,0,44,90,135,179,44,89,134,0,45,90,134,179,
    44,90,135,0,50,101,151,22,73,123,174,45,95,146,16,
    67,118,168,39,90,140,11,61,112,163,33,84,134,5,56,
    106,157,28,78,129,0,56,112,168,44,101,157,33,89,146,
    22,78,135,11,67,123,179,56,112,168,45,101,157,33,90,
    146,22,78,134,11,67,123,0,61,123,5,67,129,11,73,
    134,16,78,140,22,84,146,28,89,151,33,95,157,39,101,
    163,44,106,168,50,112,174,56,118,0,67,135,22,90,157,
    45,112,0,67,135,22,90,157,44,112,0,67,134,22,90,
    157,44,112,0,67,134,22,89,157,45,112,0,73,146,39,
    112,5,78,151,45,118,11,84,157,50,123,16,90,163,56,
    129,22,95,168,61,134,28,101,174,67,140,33,106,0,78,
    157,56,134,33,112,11,89,168,67,146,44,123,22,101,179,
    78,157,56,134,33,112,11,89,168,67,146,44,123,22,101,
    0,84,168,73,157,61,146,50,135,39,123,28,112,16,101,
    5,90,174,78,163,67,151,56,140,45,129,33,118,22,106,
    11,95
};
```

To compile it using avr-gcc in linux, check my previous post (atmega32 based wav player), there I have explained it in detail.

By Vinod.S at Monday, May 28, 2012

Labels: AVR , MY HOBBY PROJECTS , simple DSP experiments

67 comments :

Jorge May 29, 2012 at 1:04 PM

nice project!
I would like to do almost the same thing for my Amp. :-)

Reply

REGISCRUZBR May 29, 2012 at 1:49 PM

Hi, nice project.

Can you tell how much percent of memory you used and how much still available?

Thanks

Reply

Laurens May 29, 2012 at 11:51 PM

Wouldn't it be faster if you multiply the sin and cos values by a power of 2, instead of 10000? That way multiplying and dividing is only a matter of shifting.