Audio spectrum analyzer
Project Number: AR1731

Abstract

Table of figures:
Fig.1 Block diagram
Fig.2 Schematic of ZL6ARM prototype board
Fig.3 Spectrum of square wave, 1kHz frequency, 0.75V amplitude and 50% duty (rectangular window)
Fig.4 Spectrum of square wave, 1kHz frequency, 0.75V amplitude and 50% duty (Hanning's window)
Fig.5 Project photograph

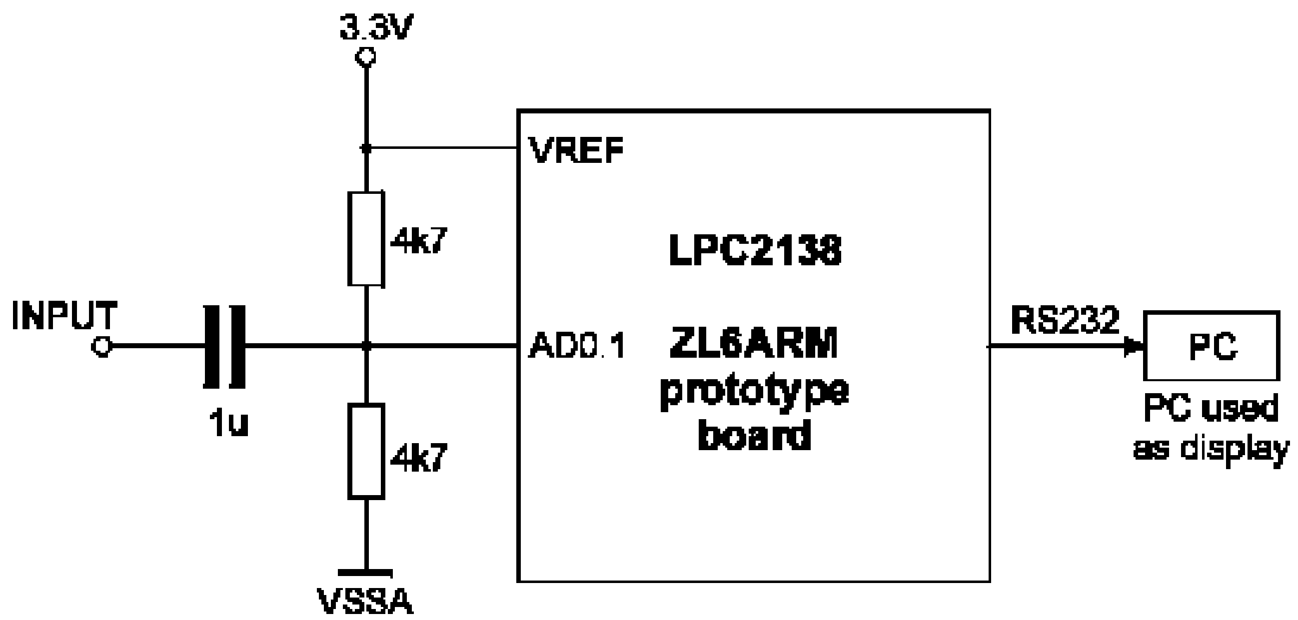
Table of listings:
List.1 Main loop of spectrum analyzer – file ARMSpectrumAnalyzer.c
List.2 Fixed point arithmetic – file fix.h
List.3 Complex numbers – file complex.h
List.4 FFT and FFT butterfly functions

Digital Signal Processing has became very important part of modern electronics. Although the best way to perform digital signal processing is to use a special DSP processor, it is possible to achieve good result when using fast conventional processor or microcontroller. This article describes how to build simple audio spectrum analyzer based on FFT algorithm using Philips LPC2138 microcontroller.
Fig.1 shows block diagram of spectrum analyzer, **fig.2** contains detailed schematic of ZL6ARM prototype board which has been used to build spectrum analyzer. ZL6ARM board contains LPC2138 microcontroller.
Input audio signal is sampled by LPC2138 microcontroller which uses built-in analog to digital converter. Sampling frequency equals 40kHz. After getting 256 samples of the signal the 256-point FFT (*Fast Fourier Transform*) algorithm is performed. The result of FFT consists of 256 complex numbers. Real and imaginary parts of those numbers are divided by 128 and absolute values of them are displayed. They are amplitude spectrum of the input signal.
Displaying the spectrum using simple graphic LCD displays is not a good idea due to relatively bad resolution of these displays. That's why I decided that spectrum will be send to PC computer via RS232 interface and there displayed by special created Windows application. It has to be stressed, that PC is used only for displaying the spectrum and all the job is made by LPC2138.
There has been designed special complex type (named *cplx*) to represent complex numbers. It's common knowledge that a complex number is a vector composed of two real numbers. Representing real (fractional) numbers was one of the main issues. Due to lack of FPU (*Floating Point Unit*) in ARM architecture the fixed point numbers with 32-bit width mantissa have been used to represent fractional numbers. Another issue was of course an implementation of FFT algorithm.
Spectrum analyzer allows to use one of three window functions: rectangular window, Hanning window and Hamming window. These three windows have bee chosen because they are the most popular windows used in DSP. Any other window function can be added by simple source code modification.
List.1 shows main loop of spectrum analyzer software. The microcontroller performs following actions periodically:
1. Getting 256 samples of input signal;
2. Multiplying samples by window function chosen by the user;
3. Computing 256-point FFT;

4. Computing squares of absolute values of FFT result (squares of amplitude spectrum);
5. Checking if any request came from PC;
6. Sending spectrum when request received.

List.2 shows implementation of fixed point arithmetic. The main advantage of fixed point numbers is that arithmetic operations on fractional numbers can be done using only operations on integer numbers. The main disadvantage is risk of overflow while multiplying or dividing. The discussion about this risk is one of article's issues. **List.3** shows implementation of complex number type *cplx* which is vector of two real numbers (*fix*). **List.4** shows a heart of the project's code – implementation of FFT algorithm using prior designed *cplx* and *fix* data types.
Fig.3 contains screenshot of PC application window made while displaying spectrum of square wave of 1kHz frequency, 0.75V amplitude and 50% duty factor. Rectangular window has been used and big DFT leakage is visible. Using Hanning window (**fig.4**) significantly decreases the leakage. Discussion about cause of leakage is also one of article's issues.

[1] Richard G. Lyons “Understanding Digital Signal Processing”
[2] “Fixed Point Arithmetic on the ARM” ARM DAI 0033A, Advanced RISC Machines Ltd (ARM) 1996 [pdf]
[3] “LPC2131/2132/2138 User Manual Preliminary Release”, Philips Semiconductors 2004 [pdf]


3.3V
VREF
4k7
LPC2138
INPUT
AD0.1
ZL6ARM
prototype
board
RS232
PC
PC used
as display
1u
4k7
VSSA

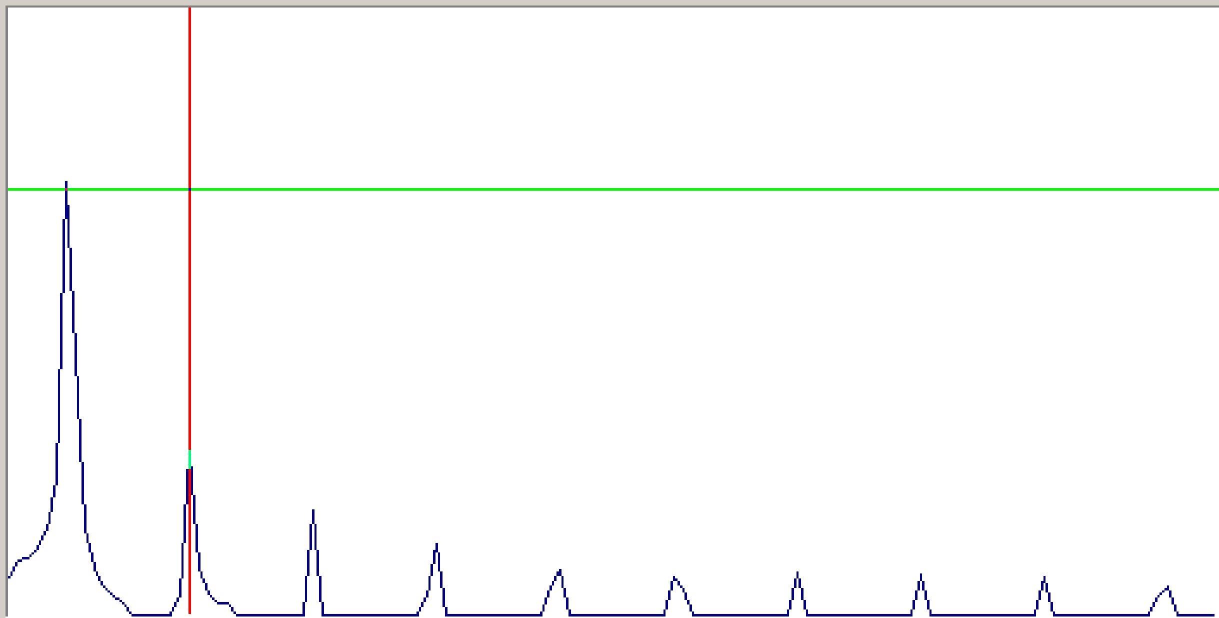
IC1
LPC2138
P0.0/TXD0/PWM1
P0.1/RXD0/PWM3/EINT0
P1.16/TRACEPKT0
P1.17/TRACEPKT1
P1.18/TRACEPKT2
P1.19/TRACEPKT3
P1.20/TRACESYNC
P1.24/TRACECLK
VBAT
VSS
VDD
VREF
VSSA
RTXC1
RTXC2
XTAL 1
XTAL 2
RESET
X1
12MHz
IC3
74LVC244
IC4
7805
Vin
Vout
GND
IC5
LM386
JP22
ZWS20
Z1
USB_A
GND
DP
DM
VCC
LPC2131
+3,3V
VCC
GND
LED
JP21
USB_RX
USB_TX
C17
100nF
R31
22k
C14
220µF
CE1
470µF
CE2
10µF

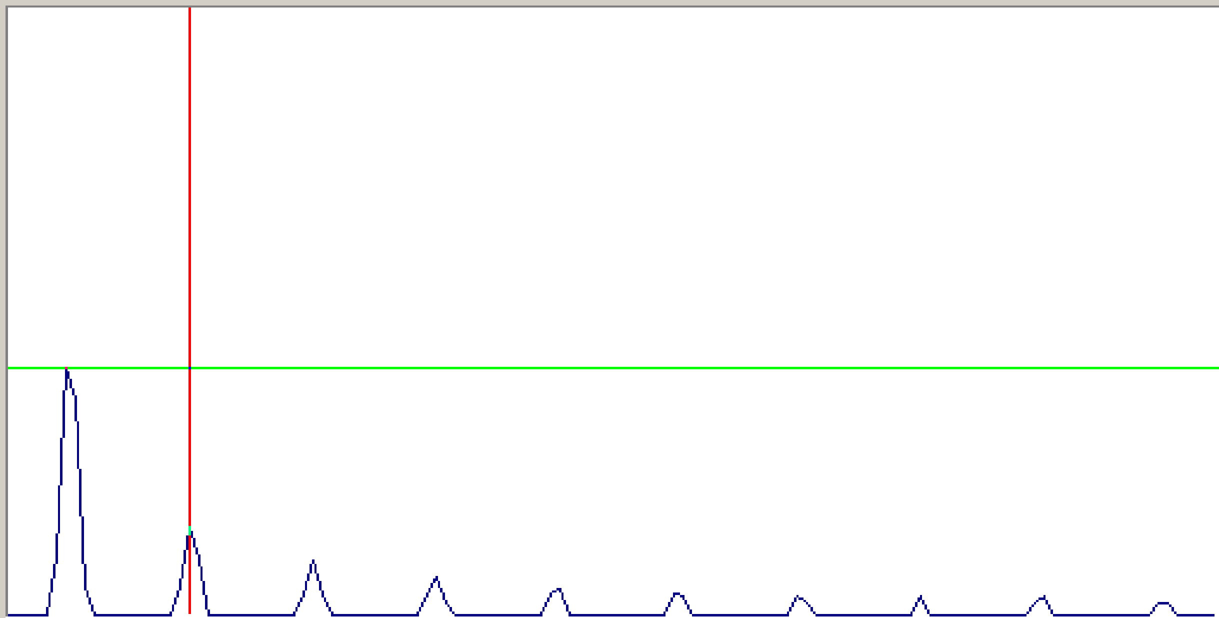
J1
COM1
J2
COM0

IC2
R2 IN
T2 OUT
V-
C2-
C2+
C1-
V+
C1+
R2 OUT
T2 IN
T1 IN
R1 OUT
R1 IN
T1 OUT
GND
VCC
9
10
11
12
13
14
15
16
R1
R2
R33
R44
VCC
MAX232

+3,3V
R7
33k
R8
22k
R9
33k
JP18
JP
+3,3V
R10
10k
D1
1N4148
RST
C6
100n
T1
BC850
T2
BC850
D2
1N4148
D3
1N4148
C7
220n
S6
RESET

VCC
C3
CAP
BUZZ
JP19
JP
G1
IC5
SPX1117
Vin
Vout
GND
+3,3V
R20
1k
D12
C4
100n
C5
100n
CE3
10uF


ARM Spectrum Analyzer - Entry: AR1731
X:
Y:
F: 2,97kHz
V: 0,70V
B
L
Run
Window: None


ARM Spectrum Analyzer - Entry: AR1731
X:
Y:
F: 2,97kHz
V: 0,41V
B
L
Run
Window: Hanning

Windowt Hanning

```
while(1)
{
	unsigned long i;

	//Sampling
	Sampling(Samples);

	//Multiplying by window's factors
	if(WinType==wtHanning)
		Hanning(Samples);
	else if(WinType==wtHamming)
		Hamming(Samples);

	//Computing Spectrum (FFT)
	FFT(Samples,FFTResult);

	//Computing squared modules of FFTResult (Spectrum)
	for(i=0;i<N;i++)
	{
		fix Re2,Im2;

		//Scaling by N/2=128
		FFTResult[i].Re>>=7;
		FFTResult[i].Im>>=7;

		//Squareing
		Re2=FMUL(FFTResult[i].Re,FFTResult[i].Re,q);
		Im2=FMUL(FFTResult[i].Im,FFTResult[i].Im,q);
		Spectrum[i]=Re2+Im2;
	}

	//Changing the window
	if(!SW1 | !SW2 | !SW3)
	{
		delay(0x800);
		if(!SW1)
			WinType=wtNone;
		else if(!SW2)
			WinType=wtHanning;
		else if(!SW3)
			WinType=wtHamming;

		DisplayWindowType(WinType);
	}

	//Checking request
	if((U0LSR & 0x01)==0)
		continue;

	//Sending spectrum to PC (if requested)
	SendSpectrumInfo(Spectrum,WinType);

	//Empty input buffer by dummy reading
	while(U0LSR & 0x01)
		i=U0RBR;
}
```

List.2

```
#ifndef FIX_H
#define FIX_H

extern const int q;      //exponent

//fixed point number type
typedef int fix;         //mantissa (interpreted as fixed point number with exponent
q)

//arithmetic operations
#define FADD(a,b)        ((a)+(b))
#define FSUB(a,b)        ((a)-(b))
#define FMUL(a,b,q)      (((a)*(b))>>(q))
#define FDIV(a,b,q)      (((a)<<(q))/(b))

#endif
```

```
typedef struct
{
        fix Re;
        fix Im;
} cplx;
```

```
//****************************************
//  Butterfly function
//  Input, output: x, y
//  x <- x + Wn[k]*y
//  y <- x - Wn[k]*y
//****************************************
void Butterfly(cplx *x, cplx *y, int k)
{
        cplx tx,ty,tWy;

        tWy=cplx_mul(Wn[k],*y);
        tx=cplx_add(*x,tWy);
        ty=cplx_sub(*x,tWy);

        *x=tx;
        *y=ty;
}

//****************************************
//  FFT function
//  x - input N complex values
//  X - output N complex values
//****************************************
void FFT(cplx *x, cplx *X)
{
        int stc;        //stages counter
        int blc;        //blocks of butterflys in stage counter
        int bfc;        //butterflys in block counter

        int NumOfStages;        //Number of stages = log2(N)
        int NumOfBlocks;        //in stage
        int NumOfButterflys;    //in block

        //Determining number of stages - counting log2(N)
        for(NumOfStages=0;NumOfStages<32;NumOfStages++)
                if(N==(1<<NumOfStages))
                        break;

        //Copying input data to output buffer
        //to perform in-place computing
        for(stc=0;stc<N;stc++)
                X[stc]=x[BitShift(stc,NumOfStages-1)];

        //Main FFT loop
        for(stc=0;stc<NumOfStages;stc++)
        {
                NumOfButterflys=(1<<stc);
                                NumOfBlocks=N>>(stc+1);

                for(blc=0;blc<NumOfBlocks;blc++)
                {
                        int base=(1<<(stc+1))*blc;
                        for(bfc=0;bfc<NumOfButterflys;bfc++)

Butterfly(X+base+bfc,X+base+bfc+NumOfButterflys,NumOfBlocks*bfc);
                }
        }
}
```