ZedBoard Project – Task 1

Xu Shipeng

Suman Deb

Contents

[Project Introduction 2](#_Toc369429251)

[System Implementation 3](#_Toc369429252)

[Profiling and Timed Automata 4](#_Toc369429253)

[Profiling 4](#_Toc369429254)

[Profiling by Hardware Timer Counter 4](#_Toc369429255)

[Profiling by gprof 5](#_Toc369429256)

[Analyze 5](#_Toc369429257)

[Timed automata 6](#_Toc369429258)

[Appendix 7](#_Toc369429259)

[zynq\_audio\_fft.c 7](#_Toc369429260)

[fft.c 9](#_Toc369429261)

# Project Introduction

In this project, we read input from the MIC, calculate the FFT, get the absolute value from FFT output, average between windows and display them on OLED display. We also profile our project.

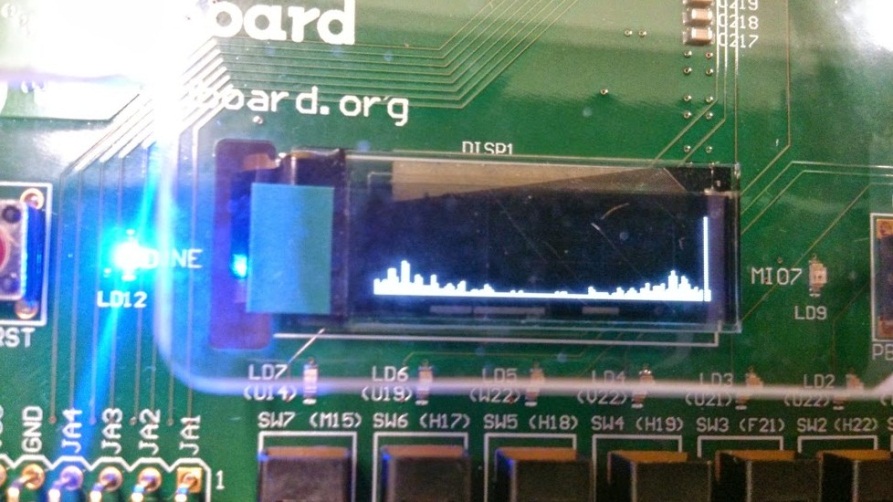


Figure 1 FFT result of human voice from audio input

Video Demo of our system:

<http://www.youtube.com/watch?v=vZl1M3n6_3s>

Project in GitHub:

<https://github.com/billhsu/FPGAFFT/>

# System Implementation

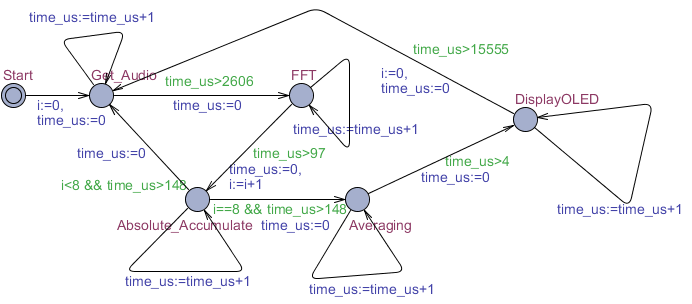


Figure 2 System Flow

Figure 2 is the system flow. It could be descripted as following pseudo-code:

while(1){

for(window = 0; window<WINDOW; ++window)

{

get\_audio();

FFT();

Get\_Abs\_and\_Accum();

}

Averaging();

Display();

}

We tried three FFT implementations to compare the performance between them.

The three FFT implementations are:

1. Kiss FFT(<http://sourceforge.net/projects/kissfft/>)
2. 16 Bit Fixed point FFT (<http://www.jjj.de/fft/fftpage.html>)
3. Floating point FFT (<http://www-ee.uta.edu/eeweb/ip/Courses/DSP_new/Programs/fft.cpp>)

# Profiling and Timed Automata

## Profiling

We profile our system in two ways: using hardware timer counter and using gprof. We use gprof to verify the profiling result by counting hardware timer.

### Profiling by Hardware Timer Counter

The hardware timer is defined in xtime\_l.h, so firstly we need to include the header file:

**#include "xtime\_l.h"**

Then we subtract the current counter value with the counter value before entering the function to get the time elapse of this function.

XTime\_GetTime(&tCur);

fft(datacpx,N,1);

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("FFT %d us\r\n",tUsed);

#### *1. Profiling result for 16 bit fixed point FFT implementation:*

\*Get audio input (one time): 2606 us

\*FFT (one time): 165 us

\*Get absolute value (one time): 37 us

Averaging: 4 us

OLED Display: 15555 us

#### *2. Profiling result for floating point FFT implementation:*

\*Get audio input (one time): 2606 us

\*FFT (one time): 97 us

\*Get absolute value (one time): 148 us

Averaging: 4 us

OLED Display: 15555 us

(\*: This is the time elapse for just one time. Totally 8 times are required for one complete loop.)

### Profiling by gprof

We also used gprof to verify the previous proofing result.

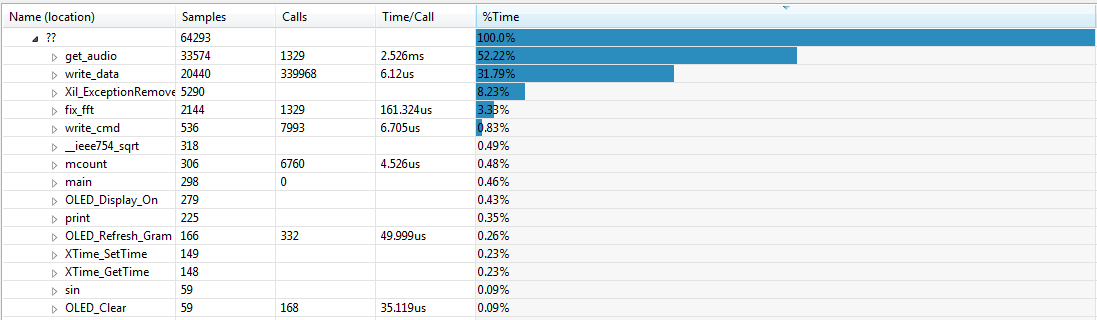


Figure 3 gprof result for fixed point implementation

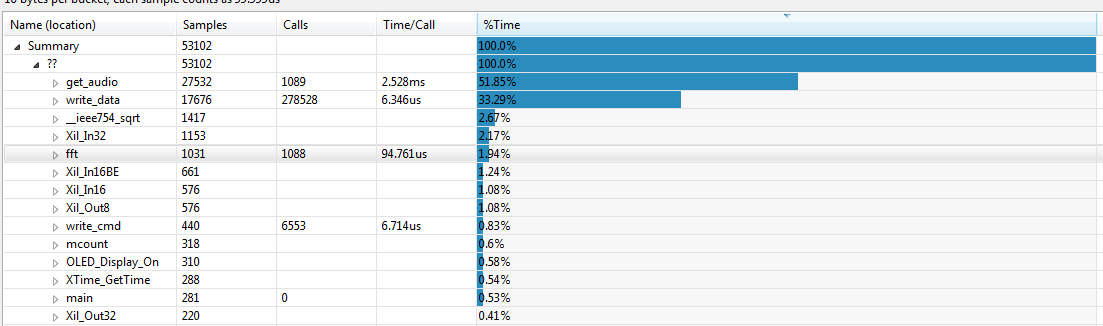


Figure 4 gprof results for floating point implementation

Form the gprof result, we can see that the fixed point fft is 161us, floating point fft is 94us. So our profiling result with timer is correct.

### Analyze

From the above results, what surprises us is that the fixed point version takes longer time in FFT than the floating point version. The algorithm in both implementations is the same. So our conclusion is that the ARM9 core in ZedBoard has a FPU to accelerate the floating point calculations. The fixed point version is slower because it used more instructions than the floating point version to do the computation.

Execution time won’t vary depending on the real numbers in the input buffer.

The FFT and get the absolute value calculation seems fast enough with this software only solution. Bottlenecks of this system are getting the audio data and displaying the result to OLED. We may improve the bottlenecks by introducing hardware acceleration technologies.

## Timed automata

We created this timed automata with uppaal. Figure 5 and Figure 6 are the screen capture of the timed automata for 16 bit fixed point implementation and floating point implementation. We simulated the model in uppaal and the simulation results verified that our system is working properly.

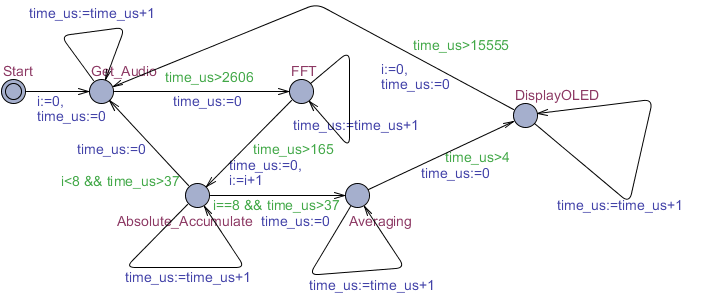


Figure 5 16 bit fixed point implementation

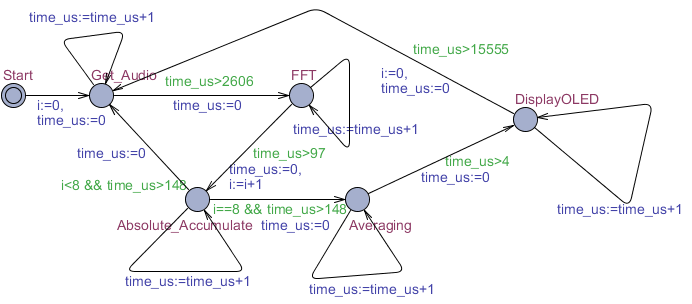


Figure 6 Floating point implementation

# Appendix

## zynq\_audio\_fft.c

**#include <stdint.h>**

**#include <stdio.h>**

**#include <stdio.h>**

**#include <sys/time.h>**

**#include <stdlib.h>**

**#include <math.h>**

**#include "fft.h"**

**#include "audio.h"**

**#include "oled.h"**

**#include "sleep.h"**

**#include "xtime\_l.h"**

**#include "xpm\_counter.h"**

**#include "xparameters.h"**

**#define N 128**

**#define POW\_N 7**

**#define WINDOW 8**

**#define FIX\_POINT**

**int** **main**(**void**)

{

**short** real[N], image[N];

**float** datacpx[N];

**int** outLED[N];

    Xint16 audio\_data[128];

**int** i,j,window;

    XTime tEnd, tCur;

    u32 tUsed;

**int** cycle**=**0;

    print("FFT start\n");

    u8 **\***oled\_equalizer\_buf**=**(u8 **\***)malloc(128**\*sizeof**(u8));

Xil\_Out32(OLED\_BASE\_ADDR,0xff);

OLED\_Init(); *//oled init*

IicConfig(XPAR\_XIICPS\_0\_DEVICE\_ID);

AudioPllConfig(); *//enable core clock for ADAU1761*

AudioConfigure();

xil\_printf("ADAU1761 configured\n\r");

*//srand(time(0));*

**while**(1){

OLED\_Clear();

printf("Cycle %d##############################\n",**++**cycle);

**for** (i **=** 0; i **<** N; i**++**) outLED[i] **=** 0;

**for**(window **=** 0; window**<**WINDOW; **++**window)

{

*//Generate input data*

XTime\_GetTime(**&**tCur);

get\_audio(audio\_data);

**for** (i **=** 0; i **<** N; i**++**)

{

**#ifdef FIX\_POINT**

*//real[i] = ((rand()%128-64)/128.0f)\*(1<<14);*

real[i] **=** audio\_data[i];

image[i] **=** 0;

**#else**

*//datacpx[2\*i]=(rand()%128-64)/128.0f;*

datacpx[2**\***i]**=**((**float**)(audio\_data[i])**/**200.0f);

datacpx[2**\***i**+**1] **=** 0;

**#endif**

}

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("get input %d us\r\n",tUsed);

*//FFT*

XTime\_GetTime(**&**tCur);

**#ifdef FIX\_POINT**

fix\_fft(real, image, POW\_N);

**#else**

fft(datacpx,N,1);

**#endif**

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("FFT %d us\r\n",tUsed);

*//Conj*

XTime\_GetTime(**&**tCur);

**for** (i **=** 0; i **<** N; i**++**)

{

**#ifdef FIX\_POINT**

**int** conj\_pdt\_out **=**sqrt((real[i]**\***real[i]) **+** (image[i]**\***image[i]));

**#else**

**int** conj\_pdt\_out **=**sqrt((datacpx[2**\***i]**\***datacpx[2**\***i]) **+** (datacpx[2**\***i**+**1]**\***datacpx[2**\***i**+**1]));

**#endif**

*//conj\_pdt\_out=conj\_pdt\_out/2;*

outLED[i]**+=**conj\_pdt\_out;

}

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("Conj %d us\r\n",tUsed);

}

*//Averaging*

XTime\_GetTime(**&**tCur);

**for**(i**=**0;i**<**N;**++**i)

{

**#ifdef FIX\_POINT**

oled\_equalizer\_buf[i]**=**outLED[i]**>>**2;

**#else**

oled\_equalizer\_buf[i]**=**outLED[i]**/**8;

**#endif**

}

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("Averaging %d us\r\n",tUsed);

*//Display*

XTime\_GetTime(**&**tCur);

OLED\_Equalizer\_128(oled\_equalizer\_buf);

OLED\_Refresh\_Gram();

XTime\_GetTime(**&**tEnd);

tUsed **=** ((tEnd**-**tCur)**\***1000000)**/**(COUNTS\_PER\_SECOND);

printf("Display %d us\r\n",tUsed);

}

**return** 0;

}

## fft.c

**#include "fft.h"**

**#define N\_WAVE 1024** *// full length of Sinewave[]*

**#define LOG2\_N\_WAVE 10** *// log2(N\_WAVE)*

*// Since we only use 3/4 of N\_WAVE, we define only*

*// this many samples, in order to conserve data space.*

**const** **short** Sinewave\_FIX[N\_WAVE**-**N\_WAVE**/**4] **=** {

      0, 201, 402, 603, 804, 1005, 1206, 1406,

   1607, 1808, 2009, 2209, 2410, 2610, 2811, 3011,

...

...

**-**32609, **-**32628, **-**32646, **-**32662, **-**32678, **-**32692, **-**32705, **-**32717,

**-**32727, **-**32736, **-**32744, **-**32751, **-**32757, **-**32761, **-**32764, **-**32766,

};

//16 bit Fixed point FFT

**void** **fix\_fft**(**short** fr[], **short** fi[], **short** m)

{

**long** **int** mr **=** 0, nn, i, j, l, k, istep, n, shift;

**short** qr, qi, tr, ti, wr, wi;

n **=** 1 **<<** m;

nn **=** n **-** 1;

*/\* max FFT size = N\_WAVE \*/*

*//if (n > N\_WAVE) return -1;*

*/\* decimation in time - re-order data \*/*

**for** (m**=**1; m**<=**nn; **++**m)

{

l **=** n;

**do**

{

l **>>=** 1;

} **while** (mr**+**l **>** nn);

mr **=** (mr **&** (l**-**1)) **+** l;

**if** (mr **<=** m) **continue**;

tr **=** fr[m];

fr[m] **=** fr[mr];

fr[mr] **=** tr;

ti **=** fi[m];

fi[m] **=** fi[mr];

fi[mr] **=** ti;

}

l **=** 1;

k **=** LOG2\_N\_WAVE**-**1;

**while** (l **<** n)

{

**long** **int** c;

**short** b;

istep **=** l **<<** 1;

**for** (m**=**0; m**<**l; **++**m)

{

j **=** m **<<** k;

*/\* 0 <= j < N\_WAVE/2 \*/*

wr **=** Sinewave\_FIX[j**+**N\_WAVE**/**4];

wi **=** **-**Sinewave\_FIX[j];

wr **>>=** 1;

wi **>>=** 1;

**for** (i**=**m; i**<**n; i**+=**istep)

{

j **=** i **+** l;

*// Here I unrolled the multiplications to prevent overhead*

*// for procedural calls (we don't need to be clever about*

*// the actual multiplications since the pic has an onboard*

*// 8x8 multiplier in the ALU):*

*// tr = FIX\_MPY(wr,fr[j]) - FIX\_MPY(wi,fi[j]);*

c **=** ((**long** **int**)wr **\*** (**long** **int**)fr[j]);

c **=** c **>>** 14;

b **=** c **&** 0x01;

tr **=** (c **>>** 1) **+** b;

c **=** ((**long** **int**)wi **\*** (**long** **int**)fi[j]);

c **=** c **>>** 14;

b **=** c **&** 0x01;

tr **=** tr **-** ((c **>>** 1) **+** b);

*// ti = FIX\_MPY(wr,fi[j]) + FIX\_MPY(wi,fr[j]);*

c **=** ((**long** **int**)wr **\*** (**long** **int**)fi[j]);

c **=** c **>>** 14;

b **=** c **&** 0x01;

ti **=** (c **>>** 1) **+** b;

c **=** ((**long** **int**)wi **\*** (**long** **int**)fr[j]);

c **=** c **>>** 14;

b **=** c **&** 0x01;

ti **=** ti **+** ((c **>>** 1) **+** b);

qr **=** fr[i];

qi **=** fi[i];

qr **>>=** 1;

qi **>>=** 1;

fr[j] **=** qr **-** tr;

fi[j] **=** qi **-** ti;

fr[i] **=** qr **+** tr;

fi[i] **=** qi **+** ti;

}

}

**--**k;

l **=** istep;

}

}

//floating point FFT

**void** **fft**(**float** data[], **int** nn, **int** isign)

{

**int** n, mmax, m, j, istep, i;

**float** wtemp, wr, wpr, wpi, wi, theta;

**float** tempr, tempi;

    n **=** nn **<<** 1;

    j **=** 1;

**for** (i **=** 1; i **<** n; i **+=** 2) {

**if** (j **>** i) {

tempr **=** data[j]; data[j] **=** data[i]; data[i] **=** tempr;

tempr **=** data[j**+**1]; data[j**+**1] **=** data[i**+**1]; data[i**+**1] **=** tempr;

}

m **=** n **>>** 1;

**while** (m **>=** 2 **&&** j **>** m) {

j **-=** m;

m **>>=** 1;

}

j **+=** m;

    }

    mmax **=** 2;

**while** (n **>** mmax) {

istep **=** 2**\***mmax;

theta **=** TWOPI**/**(isign**\***mmax);

wtemp **=** sin(0.5**\***theta);

wpr **=** **-**2.0**\***wtemp**\***wtemp;

wpi **=** sin(theta);

wr **=** 1.0;

wi **=** 0.0;

**for** (m **=** 1; m **<** mmax; m **+=** 2) {

**for** (i **=** m; i **<=** n; i **+=** istep) {

j **=**i **+** mmax;

tempr **=** wr**\***data[j] **-** wi**\***data[j**+**1];

tempi **=** wr**\***data[j**+**1] **+** wi**\***data[j];

data[j] **=** data[i] **-** tempr;

data[j**+**1] **=** data[i**+**1] **-** tempi;

data[i] **+=** tempr;

data[i**+**1] **+=** tempi;

}

wr **=** (wtemp **=** wr)**\***wpr **-** wi**\***wpi **+** wr;

wi **=** wi**\***wpr **+** wtemp**\***wpi **+** wi;

}

mmax **=** istep;

    }

}