Implementation of audio processing algorithm in ZYNQ platform

Raghunathan Mahesh Kumar, Gourav Modi
Nanyang Technological University, Singapore

I. Abstract

The main objective of this project is to display the frequency spectrum of input audio signal on an OLED by constructing various modules that utilizes minimal hardware resources under the given latency constraint. Hardware software co-design investigates the concurrent design of hardware and software components of complex electronic systems. We exploited the synergy of hardware and software with the goal to optimize the area in the hardware as in [5].

II. Introduction

The main objective of this project is to display the frequency spectrum of the input audio signal on OLED by constructing the software module algorithm with minimal latency. The algorithm consists of using fast fourier transform method to convert time domain audio signal into its corresponding frequency representation. Along with it, some other strategies such as sliding window averaging, dynamic noise cancellation and thresholding has been considered.

III. Hardware Software Co-Design Implementation

The entire implementation of the co-design part is demonstrated in the form of a flow chart in Figure 1. In the co-design part, we have pushed the noise module to hardware and implemented the remaining modules in software to achieve the maximum utilization.

IV. Processing System Implementation

1.Input Audio Module (Digitizer and I2S)
The audio module logic has been provided which obtains the real time audio signal from microphone and digitizes it using a sampler. Finally, using I2S bus a window of 128 audio samples is provided for further processing.

2.Fast Fourier Transform Module ([1],[2],[3],[4])
In order to analyze spectrum, the time domain signal needs to be converted into frequency domain using the technique known as Discrete Fourier Transform. In order to reduce computation complexity and enable faster computation, a widely known technique known as FFT (Fast Fourier Transform) is used. There are different variants of FFT available such as Danielson-Lanczos algorithm, Cooley-Tukey algorithm, to name a few.
Initially, we designed our own FFT module based on Cooley-Tukey algorithm and reduced computation time by exploiting symmetry in the output, creating a look-up-table (Precomputed values of trigonometric functions and bit reversal values) and specific case programming. The results were in order of 140 us. To evaluate our algorithm, we used a library which was using Danielson-Lanczos algorithm, and the performance that we achieved was better than our FFT code. Hence, we went for the library and further optimized the library code by including optimization techniques which we had incorporated in our earlier FFT design; the timings are in the order of 29us. As expected, there is a drastic improvement in the end results.
The algorithm breaks the entire input block recursively into smaller odd and even blocks (till block size 2). The input is swapped owing to the bit reversal format that needs to be given. In the library used, the twiddle factors are calculated manually by exploiting a trigonometric recurrence on real numbers and then constructing corresponding floating point numbers. Also, the library code has used trigonometric identities to minimize the number of circular functions needed to be computed. Hence, we see a huge performance improvement.
3.Magnitude Module
The output of the Fourier transform module provides the real and imaginary frequency components. The magnitude of this complex number is computed by taking the complex conjugate as shown in equation 1.

$$\|a + ib\| = a^2 + b^2 \tag{1}$$

Note: The magnitude is obtained by taking square

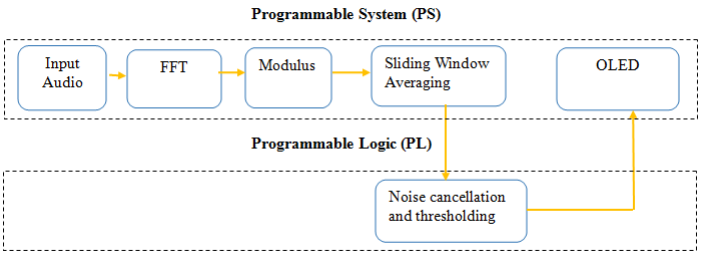

Fig. 1: Flow chart of the co-design implementation

root of $\|a+ib\|$. However, in this project, square root function is not implemented and this is taken to be approximation.

4.Averaging Module

The output of the magnitude module is then transferred to the next stage for performing sliding window averaging. As stated previously, this is done to mitigate the boundary errors of window while averaging.

Circular Buffer technique is used for sliding window averaging. In this method, we have to only use 8 windows (each of size 128) for the audio samples, and these windows are used in an efficient ordered fashion. The algorithm for Circular Buffer technique is such that after computing the average of 8th windows, the 9th window containing the new audio sample is put into the 1st window (contents are replaced since the samples in the 1st window will not be used).

Note: Without circular buffer technique, the number of windows that would be required for averaging would be much more, which is obviously costly in terms of memory allocation and memory access. This would in-turn increase the latency of the code.

5.AXI Lite peripheral

The communication between PS and PL is established through this peripheral.

6.Display Module

The noise free output audio signal is then sent to OLED display. The driver for OLED display is provided.

V. Programmable Logic Implementation

Using AXI lite interface against AXI burst proved effective. We reduced the memory mapped RAMs and utilized only minimum number of slice registers and LUTs. For data computation of noise cancellation and thresholding, we use only three 32-bit registers.

1.Noise cancellation and thresholding

Noise cancellation is necessary to alleviate any interference with the spectrum. In this module, noise value is calculated for the first 1024 windows (sufficient enough to calculate the ambient noise in a room). After obtaining the noise value, the resultant value is averaged over 1024 windows and thereby threshold is obtained. This threshold is the Lower Threshold and it is subtracted from the audio signal (in frequency domain). Hence, the approach taken is to remove noise dynamically from the system.

The Upper Threshold of the audio signal is the OLED vertical display size which is found to be 63(by giving a ramp input). And any value of the audio signal beyond the size of the display must be truncated to the Upper Threshold. Upper thresholding is implemented in OLED module while sending the samples to the displays.

VI. Resource Optimization Technique

The following are the optimization techniques which are implemented:

1.AXI lite interface used to communicate between PS and PL. This minimizes many number block RAMs since LITE is not memory mapped unlike AXI burst.

2.For data input, noise cancellation & thresholding and data out to PS, the data computation is achieved using three 32-bit registers.

3.Using ISE design suite, area is optimized under timing constraint as in [6].

VII. Latency Reduction Technique

The following are the optimization implemented:

1.Circular Buffer Technique is used for sliding window averaging and noise calculations.

2.Optimized algorithm is used wherein only one half of the output spectrum is calculated and the other half is mirrored.

3.In FFT, in order to decrease computation speed, LUT (look up tables) are used.

4.In cases of, conditional statements such as if and else involves lots of jumps. Hence, to reduce such transitions, the most obvious case is accounted at the beginning of the loop.

5.Declaration of local variables is avoided since it involves lots of push and pop operations from the stack. Instead, some variables are globally declared which involves only registers.

6.Using Registers for computation and storing the commonly used variables.

7.Right shift operators are used in places of integer divisions and multiplications.

8.Data types are carefully selected to save on the memory space.

9.Hardware timer module is used to obtain the latency.

S.No	Latency	Timing
1	Worst case	44us
2	Average case	44us

TABLE I: Latency calculation

Fig. 2: Output spectrums on OLED display

10.Unrolling and merging of for loops are done.
Note: ARM gcc complier optimization technique is turned off under all circumstances.

VIII. Results

All the above modules are implemented and output spectrum on the OLED is captured for various audio input signals (by varying the frequency and magnitude). An implementation of the same is demonstrated in Figure 2.
Latency of the code is calculated from FFT module (where the audio sample is obtained) till noise cancellation and thresholding module (displaying it to OLED is not accounted).
The latency for individual module is tabulated in table 1.

IX. Observations

Comparison of various implementation methods are tabulated in Table 2. The values excludes audio and OLED latency and resources.
In the co-design part, all the modules are implemented in software except for the noise cancellation and thresholding which is implemented in hardware. This is how the area is brought down.
However, the latency value is brought down by computing FFT only for 64 values and extended to 128 by interleaving the adjacent value.
We captured in Table 3 the unique result we have observed during the co-design implementation :
All the latency measurements were carried out using no compiler optimization. However, using O3 optimization the latency of the co-design implementation can be brought down to 30us.

S.No	Method	Latency	Utilization
1	Software	71us	Reg:0, LUT:0
2	Hardware	42us	Reg:3572, LUT:6771
3	Co-Design	44us	Reg:70, LUT:70

TABLE II: Different implementation methods

S.No	Method	Latency	Utilization
1	Noise module	54us	Reg:129, LUT:129
2	Noise & Thresholding	44us	Reg:70, LUT:70

TABLE III: Different hardware implementation methods

X. References

[1] Numerical_Recipes.pdf
[2] www.nr.com
[3] www.katjaas.nl/FFT/FFT2.html
[4] beige.ucs.indiana.edu/B673/node14.html
[5] ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6172642
[6] Zynq Technical Reference Manual - http://www.xilinx.com/support/documentation/userguides/ug5 Zynq-7000-TRM.pdf.