2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Real-Time Automatic Music Transcription (AMT) with Zync FPGA

1st Vaca Kevin
Engineering Department
University of Houston Clear Lake
Houston, USA

2nd Archit Gajjar
Engineering Department
University of Houston Clear Lake
Houston, USA

3th Xiaokun Yang*
Engineering Department
University of Houston Clear Lake
Houston, USA
yangxia@uhcl.edu

***Abstract*—A real-time automatic music transcription (AMT) system has a great potential for applications and interactions between people and music, such as the popular devices Amazon Echo and Google Home. This paper thus presents a design on chord recognition with the Zync7000 Field-Programmable Gate Array (FPGA), capable of sampling analog frequency signals through a microphone and, in real time, showing sheet music on a smart phone app that corresponds to the user's playing. We demonstrate the design of audio sampling on programming logic and the implementation of frequency transform and vector building on programming system, which is an embedded ARM core on the Zync FPGA. Experimental results show that the logic design spends 574 slices of look-up-tables (LUTs) and 792 slices of flip-flops. Due to the dynamic power consumption on programming system (1399 mW) being significantly higher than the dynamic power dissipation on programming logic (7 mW), the future work of this platform is to design intelligent property (IP) for algorithms of frequency transform, pitch class profile (PCP), and pattern matching with hardware description language (HDL), making the entire system-on-chip (SoC) able to be taped out as an application-specific design for consumer electronics.**

***Index Terms*—automatic music transcription (AMT), field-programmable gate array (FPGA), pitch class profile (PCP), system-on-chip (SoC)**

I. Introduction

A remarkable pace at which automatic music transcription (AMT) is developing has put new demands on real-time audio processing systems. During the last decade AMT has been dominated by two algorithmic families: non-negative matrix factorization and deep neural network, which are accurate with note detection but very time- and energy-consuming on hardware platforms. To accelerate the data analytics in cloud centers, field-programmable gate arrays (FPGA) have been widely used by providers such as Amazon Web Services (AWS), Alibaba Cloud, and Microsoft [1]–[3].

With another trend of offloading data processing from cloud to edge, the use of FPGA has also evolved to edge computing due to the merits of programmability and computational parallelism [4], [5]. The Zync FPGA, a hardware-software co-design platform which integrates the software programmability of an ARM processor with the hardware programmability of an FPGA, enables the integration of key analytics and hardware acceleration with CPU, digital system design, and analog signal functionality onto a single device. Consisting of ARM cores, the Zynq FPGA is one of the best solutions to fully scalable, system-on-chip (SoC) platforms for the real-time application requirements.

Under this context, this paper proposes an application on music chord recognition with Zync7000 FPGA, enabling the input of raw analog frequency signal through a microphone, and, in real time, outputting sheet music on a smartphone app. Generally, chord recognition is a process of identifying specific harmonic sets of three or more musical notes. And a musical note refers to the pitch class set of C, $C\#$, D, $D\#$, E, F, $F\#$, G, $G\#$, A, $A\#$, and B. The entire system contains three phases: detection, processing, and user interface, which respectively sample, extract, and display the chords onto a smartphone app. The main phases – detection and processing – are computed on FPGA to handle the sampling, frequency spectrum generation, pitch class profile (PCP) building, and pattern matching. The user interface is being worked on using a separate Android device. The ideal end goal of this project is a demonstration on FPGA with hardware programming language (HDL) programmed intellectual property (IP), which is able to be taped out and integrated into SoCs for marketing devices like Amazon Echo and Google Home. The main contributions of this work are:

- We present a real-time music transcription system on Zync7000 FPGA, which is able to extract the frequencies from a live recording and then analyze the given frequencies to find specific chords played by the user. The goal of this system is to minimize the latency of recognizing one chord within 350 millionths of a second, which is the time constraint for processing music with a high tempo.
- We present the block-based design and the synthesis results in terms of slice count and power consumption on Zync7000 FPGA. Due to the significantly higher power consumption on programming system (the embedded ARM core) compared with the programming logic (the FPGA), the next step of this platform is to design the algorithms of frequency transform, PCP, and pattern matching with HDL, making the entire system as an IP for SoCs for consumer electronic devices.

The organization of this paper is as follows. Section II briefly introduces the related works and III presents our work with design architecture. Section IV discusses the implementation of the proposed system. In Section V, the experimental results in terms of hardware cost, power consumption, and FPGA prototype are shown. Finally, Section VI presents the concluding remarks and future work in our target architecture.

II. Related work

Prior research on AMT mainly focused on the improvement of accuracy of algorithms [10]. The dominant algorithmic families for accuracy enhancement in the last decade are non-negative matrix factorization [13], [14] and deep neural network [12], [15], [16]. All this research highlights the achievement of high accuracy rather than the computation performance in terms of speed and energy efficiency on edge devices.

Our work concentrates on presenting a real-time system able to automate music transcription around 350 millionths of a second per chord recognition. This intended rate was established by taking into consideration that the tempo of a skilled pianist's performance is 120 beats per minute (bpm), and the tempo of jazz music can reach 340 bpm [18], with chords changing between one and two times a measure. Therefore, the number of chords that need to be recognized per second can be calculated as: $\frac{340\ beats}{1\ minute} \times \frac{1\ minute}{60\ seconds} \times \frac{1\ measure}{4\ beats} \times \frac{2\ chords}{1\ measure} \approx 2.83\ chords/second$. In other words, the time constraint for recognizing one chord should be less than or equal to $1/2.83 = 353\ thousandths\ of\ a\ second$.

All the existing computer programs for music transcription, such as Chordify [21] and AnthemScore [22], have an overhead of several minutes per song transcription, which take in audio files like mp3, WAV, etc., and use computer vision and convolutional neural networks to extract the notes from a visualized spectrogram. Considering offloading such a complex algorithm from cloud or PC to resource-limited edge devices, it near impossible to produce a real-time system.

A case study of chord detection on stand-alone device with Raspberry Pi has been proposed in [17]. The implementation of this system has the device record for one second and analyze the data for another one second, totaling 2 seconds compared to our intended range of 0.353 seconds. To overcome this challenge, we present a design with Zync7000 FPGA, enabling automating music transcription in real time by playing out a melody. Our final hope is to make the design available as an application-specific integrated circuit (ASIC) IP.

III. Proposed work

Fig. 1 shows the flow chart of the system. The different tasks are separated into four different parts which include: sampling, transform, vector building, and pattern matching. The audio sampling is processed by programming logic, or PL, on Zync FPGA, and the Discrete Fourier Transform (DFT), PCP building, and pattern matching are run on programming system, or PS, on Zync FPGA. The final goal of this project is to program the modules of DFT and PCP building on programming logic as well. Additionally a Raspberry PI is adopted to match the patterns and show the sheet music on an smartphone app. This paper focus on the first four tasks based on Zync FPGA.

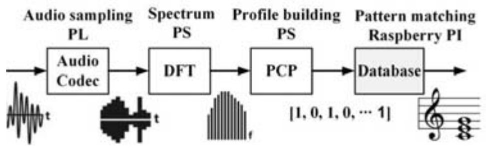


Fig. 1. Flow Chart

A. Sampling

Sampling is the conversion of a continuous time signal into a discrete time signal. The analog audio signal produced by music is picked up by the microphone and converted into binary data. The sampling process involves programming the Analog Devices SSM2603 Audio Codec present on the Zybo [7], in order to set the proper data path, data size, gain, and sampling rate. Each of the mentioned settings are programmed onto the chip's registers by using an I^2C interface. For the default sampling rate of 48 KHz, the master clock should be driven at 12.288 MHz by the SoC.

B. Frequency Transform – DFT

After obtaining the required samples, the following process is the DFT. The DFT is used to convert the samples from the time domain to the frequency domain. It is in the frequency domain that we can differentiate between the different notes present in the sample. The equation for calculating the DFT is shown below

$$X_k = \sum_{n=0}^{N-1} x_n \times (\cos(\frac{2\pi kn}{N}) - i \times sin(\frac{2\pi kn}{N})) \tag{1}$$

where N is the number of samples, k is the index value for the frequency domain, and n is the index value for the time domain. X_k denotes the magnitude of the frequency bin at k, and x_n represents the time-domain sample data at n.

The real and imaginary parts of X_k can be further broken up into their own respective summation equations, as shown in Eq. 2 and Eq. 3.

$$X_{k_{real}} = \sum_{n=0}^{N-1} x_n \cos(\frac{2\pi kn}{N}) \tag{2}$$

$$X_{k_{imag}} = \sum_{n=0}^{N-1} x_n \sin(\frac{2\pi kn}{N}) \tag{3}$$

The final value of X_k is the absolute value of the real and imaginary parts combined, as shown in Eq. 4. Upon building the frequency spectrum of the collected samples, the next process would be finding the PCP.

$$|X_k| = \sqrt{(X_{k_{real}}^2) + (X_{k_{imag}}^2)} \tag{4}$$

C. Vector Building – PCP

In music, there are twelve notes that are denoted by the letters A through G, combined with accidentals that raise or lower a note by a half-step. For instance, C and D are a full-step apart, but C and $C\#$ (C sharp), or $C\#$ and D are only a half-step apart. Typically beginning with C, if one were to increment by half steps until reaching B, increasing it further would loop back to C, only an octave higher. A $C0$ (0th octave C) is twelve steps from a $C1$ (1st octave C). In terms of frequency, $C1$ is equal to twice the frequency of $C0$, and $C2$ would be twice $C1$. When looking for the PCP, one must find the summation of each of the twelve notes across all octaves present in a frequency spectrum.

The two equations used in calculating the PCP are shown in Eq. 5 and Eq. 6 [11]:

$$PCP(p) = \sum_{l s.t. M(l)=p}^{l} \|X(l)\|^2 \tag{5}$$

$$M(l) = round\left[12log_2\left(\frac{f_s l}{N f_{ref}}\right) mod\ 12\right]. \tag{6}$$

In Eq. 5, p corresponds to the note being built. A p of 0 would be a C, and a p of 1 would be a $C\#$, etc. $X(l)$ would be the frequency magnitude at index l and $M(l)$ would be the pitch class at index l.

Eq. 6 is for calculating the pitch class for index l. In this equation, f_s/N corresponds to the bin width in Hz, or what the change in frequency is between each bin/index l. The l is the current bin or index being calculated, and f_{ref} is the reference frequency for the base note of the vector.

Eq. 6 would be run for every bin l and every note p. When $M(l)$ is found to equal p, the magnitude of the frequency $X(l)$ would be added to the summation of the PCP at p. For example, say low E ($E2$, 82.41 Hz) on the guitar was plucked and recorded by the device. If the sampling rate was 8 kHz and the number of samples was 2048, then bin 21 ($8,000/2048 \times 21 = 82.03$) would correspond the closest to $E2$. When calculating the PCP for note E (p = 4, f_{ref} = $C0$ = 16.35 Hz), the equation would read out: $M(21) = 12\log_2(\frac{8000\times21}{2048\times16.35})\%12$, which turns out to be $M(21) = 3.92 \approx 4$. Thus, $M(21) = p = 4$, so the PCP summation for E (p = 4) would include the frequency magnitude at bin 21.

After finding the PCP summations for each of the twelve notes, a threshold is calculated and compared to each of the notes. Then the PCP logic vector (true/false, 1/0) is created based on the threshold which ultimately denotes which notes are present throughout the frequency spectrum. The logic vector will be further pattern matched in the final process as well.

D. Pattern Matching

In music, a chord is made up of three or more notes in varying octaves. For example, a G Major chord on the guitar consists of $G2$, $B2$, $D3$, $G3$, $B3$, and $G4$. Without taking the octaves into consideration, the notes present in the G Major Chord Template would be G, B, and D. The PCP chord template for the G Major chord would be [0 0 1 0 0 0 0 1 0 0 0 1]. The G Minor chord contains G, D, and $A\#$, which would correspond to [0 0 1 0 0 0 0 1 0 0 1 0]. Every chord has its own pattern within the twelve note PCP vectors.

In the previous Section III-C, Eq. 5 and Eq. 6 are used to build the PCP summation and consequently the PCP chord template. In this section, the output from Eq. 5 and Eq. 6 is simply matched up to the predefined chord templates and the appropriate chord name is given out. In the cases where there are not enough notes to match a chord (only one or two notes are played), then whichever notes are present are output.

IV. Implementation

Fig. 2 depicts the hardware architecture for the implementation with Zync FPGA. The target device used in this work is Xilinx Zybo Zynq-7010, and the audio in is with a guitar playing as a test case.

The audio signal is introduced into the system by a microphone connected to the "J6 Microphone In" connection on the Zync SoC. The SSM2603 Audio Codec samples and sends the data to the ARM Processor which stores the data in an array. Both the DFT and PCP algorithms are implemented in C language to process the sampled data. The PCP output is then sent through the UART protocol to a separate system which handles the pattern matching. The process, from sampling to the PCP output on UART, is repeated until the user shuts off the system.

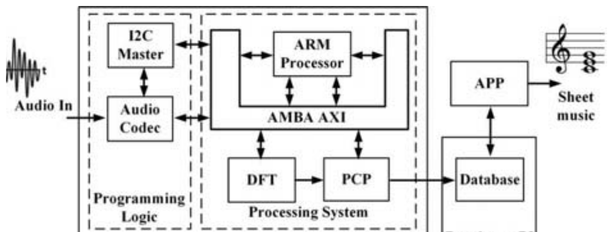


Fig. 2. Hardware Architecture

A. Audio Codec (SSM2603) Configuration

Before discussing the specific configuration of Audio Codec, the hardware-software co-design system is created in Fig. 3, including the ARM core, or specified as Cortex-A9 Processor, and the block-based design of an audio controller. The preliminary results shown in this paper are obtained by the integration of design logic and Cortex-A9 Processor, and the future work will focus on offloading the algorithms of DFT, PCP, and the database onto programming logic.

For the audio controller, registers "ANALOG_AUDIO_PATH" and "SAMPLING_RATE" are configured outside of their default settings [7]. Analog Audio Path has been configured to only mix the -15dB attenuated "Microphone In" input at the Analog to Digital

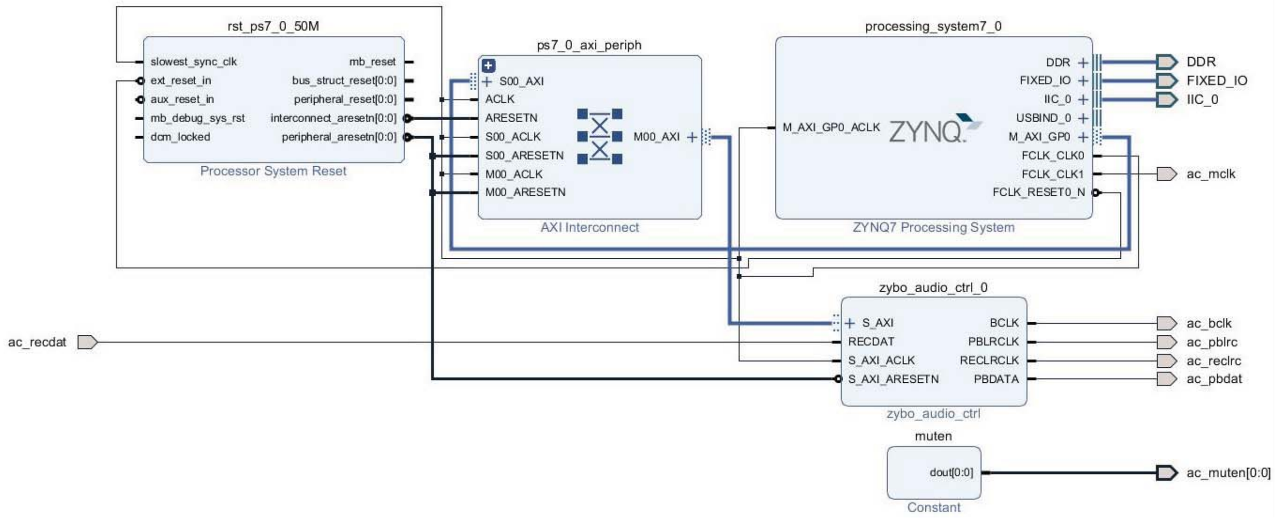


Fig. 3. The hardware-software co-design system.

Converter (ADC). The Digital Audio Interface (register “DIGITAL_AUDIO_I_F”) is set to default which has the data output of the ADC set to 24 bits transferred over the I^2S protocol. The Sampling Rate (register “SAMPLING_RATE”) has been adjusted so that, with a master clock driven at the default 12.288 MHz, the sampling rate for the system is 8 kHz.

B. DFT on Cortex-A9 Processor

The remainder of the implementation is done on the SoC’s processor. After configuring the Audio Codec, the processor collects samples from the Codec and stores them into memory. The following step is to transform the data from the time domain to the frequency domain. The theory is outlined in Section III-B.

To decrease processing time, only the first N/8 samples are processed to the frequency domain. Using a 8 kHz sampling rate and a 2048 sample size, the final sample processed (256) would correspond with the frequency near 1000 Hz. Outside of resonant frequencies created when playing the guitar, the range of a standard tuned guitar would not exceed 1000 Hz (though the resonant frequencies do inform the pitch class). The DFT equation has a big O of $O(N^2)$, but only processing an eighth of the samples, the big O is effectively divided by eight.

C. PCP on Cortex-A9 Processor

The PCP algorithm is also implemented on the processor as detailed in Section III-C. The previous reduction in processed sample size also reduces the big O for the PCP algorithm. Originally $O(N)$, the processing of only an eighth of the samples reduces the computation time to $O(N/8)$. Although a constant factor is not usually considered when discussing big O as N approaches infinity, with a relatively small and finite N, the difference in processing time is noticeable.

Alg. 1 depicts the pseudocode of PCP implementation. $Line\ 2-7$ is used to find the frequencies which contribute to the PCP element ($Line\ 3$) and then sum the corresponding DFT output to the PCP output ($Line\ 5$). In order to calculate the threshold, the PCP output is added to an average variable in $Line\ 8$. The final average PCP output should be the $average/12$ shown in $Line\ 10$.

$Line\ 11-17$ shows the collection of the PCP vector. The threshold calculation for the PCP output has been an area of experimentation. The current implementation has the threshold set to the average value of the PCP summations, as depicted in $Line\ 12$. The PCP vector over the threshold is specified as 1, otherwise being specified as 0. In such a way the PCP vector with 12 digits are created.

This approach tackles two issues concerning the process of extracting the PCP from the frequency domain. The first issue concerns the placement of the threshold. The two options consist of an absolute threshold and a relative threshold. With an absolute threshold, one runs the risk of detecting majority 1s when multiple notes overlap each other when playing. If the music is played in a way that allows for previous notes to be held over to new notes, the PCP algorithm will detect those old frequencies and you will be left with false positives.

Choosing to make the threshold relative to all the notes by using the average solves the issue of loud volume, but a second issue arises. In situations where the guitar is not being played, the frequency spectrum will not be completely empty, and the PCP algorithm will still create summations of different notes. It can introduce a lot of false positives in moments of low volume. Thus, the current solution is to scale the average up by a certain amount so the threshold will only pass notes that are present well above the average.

This paper focuses on establishing the platform with Zync FPGA and the future work will concentrate on improving the accuracy of the system and design the PCP IP with HDL.

Algorithm 1 Pseudocode of the PCP algorithm.

```
Require: dft[f]: the output results of DFT in index f; N: the
    DFT Size; q: 12 PCP elements, cf: current frequency
 1: for $q = 0; q < 12; q++$ do
 2:     for $f = 0; f < N/8; f++$ do
 3:         $cf = round\left[12log_2\left(\frac{f_s l}{N f_{ref}}\right) mod\ 12\right];$
 4:         if $cf = q$ then
 5:             $pcp[q] += dft[f];$
 6:         end if
 7:     end for
 8:     $avg += pcp[q];$
 9: end for
10: $avg = avg/12;$
11: for $q = 0; q < 12; q++$ do
12:     if $pcpout[q] > avg$ then
13:         $pcpout[q] = 1;$
14:     else
15:         $pcpout[q] = 0;$
16:     end if
17: end for
```

V. Experimental results

In this section, the results of the transform output on FPGA are compared with the golden models on MATLAB. In what follows, the FPGA resource cost in terms of slice count and power consumption is evaluated [8], [9].

A. Accuracy comparison

Before discussing the system performance, Fig. 4 shows the platform for testing the audio-in channel, including a standard tuned guitar and a system with a microphone connected to the "J6 Microphone In" of a Zync SoC.

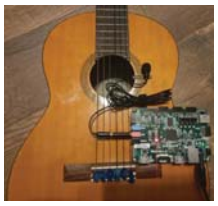
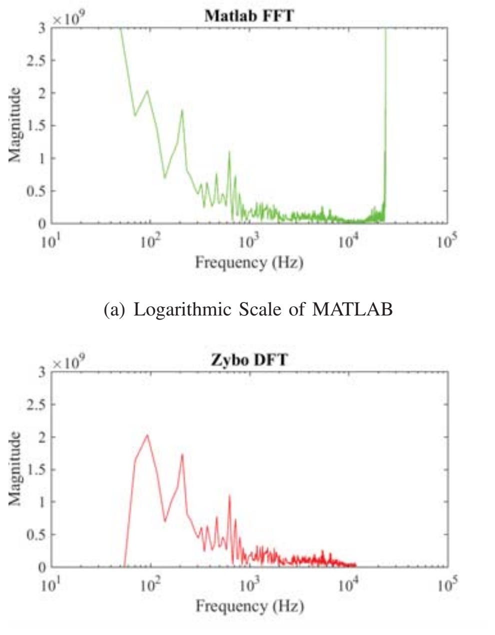


Fig. 4. Testing platform with guitar and FPGA

After establishing the platform, the transform output of the Zync programming system has been compared to the MATLAB output based on the same samples collected by the Zync programming logic. While testing the accuracy of our DFT results in the system, we output both the sample data collected by the Audio Codec and the results of DFT, through UART, back to the host computer. The sample data set was then processed through the MATLAB DFT function, which is seen as the golden models for testing the accuracy. Finally, the golden result shown in Fig. 5(a) and the FPGA result shown in Fig. 5(b) are compared.



(a) Logarithmic Scale of MATLAB

(b) Logarithmic Scale of FPGA DFT

Fig. 5. Fourier Transform Comparison

In order to clearly see the difference, we analyze both results overlaid on each other in Fig. 6. The first three results (0 Hz, 23.44 Hz, and 46.88 Hz) are discarded due to both the nature of low frequency waveform and the fact that a guitar's lowest note begins at 82 Hz. As shown in Fig. 6, both results begin to match up at the fourth result, and continue to match up very closely until the end of the Zybo results.

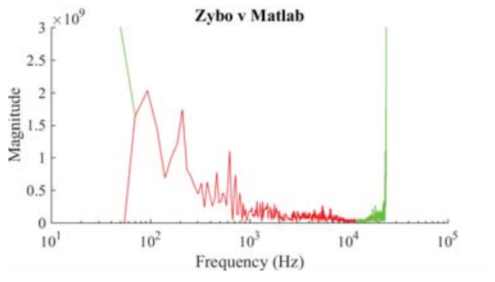


Fig. 6. Results Comparison between MATLAB and FPGA

In what follows, Fig. 7 shows the percent error calculation by using the standard percent error formula $\frac{MATLAB\_Result - FPGA\_Result}{MATLAB\_Result} \times 100\%$). The X-axis is the index number for our results up to 512 (our Zybo DFT was only calculated up to N/2). As seen in the figure, our largest error does not exceed 0.0004%.

The detection rate for the PCP algorithm is not as accurate. The large variation in our results for the PCP algorithm could be attributed to a number of issues including: resonant frequencies present in the environment, sensitivities of the

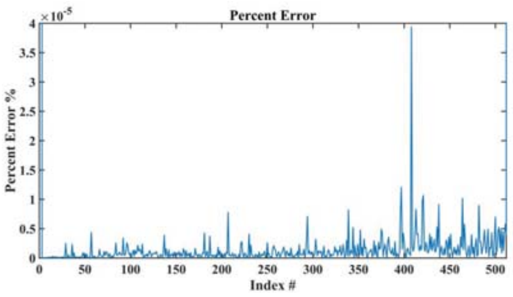


Fig. 7. Percentage Error

microphone picking up unwanted noise, and user error in playing the instrument. The introduction of a moving average for the PCP vector results should provide some stability in the vector building process. The accuracy improvement of the PCP implementation is one of our future works.

B. Resource cost on FPGA

The resource cost on FPGA is shown in Table I. The implementation of the system spends 547 slices of look-up-tables (LUTs) and 792 slices of flip-flops. The number of slices would significantly increase when integrating the IPs of DFT and PCP into the programming logic in the future.

TABLE I
FPGA RESOURCE COST

Resource	Utilization	Available
LUT	547	17600
LUTRAM	60	6000
FF	792	35200
IO	9	100
BUFG	2	32

The power consumption on Zync FPGA is shown in Table II, including 120 mW static power (SP) shown in the second column and the dynamic power (DP) shown as a sum between the third column to the seventh column. Notice that the dynamic power consumption on programming system (shown in the seventh column) is significantly higher than the dynamic power dissipation on the programming logic (shown as a sum between the third column to the sixth column). Thus, to design the IP of DFT and PCP with HDL has a great potential to improve the power efficiency of the entire system.

TABLE II
POWER CONSUMPTION ON FPGA

TP[a] (mW)	SP[b] (mW)	DP[c] (mW)				
		Clocks	Signals	Logic	I/O	PS7
1526	120	2	1	1	3	1399

[a]Total power consumption.
[b]Static power consumption.
[c]Dynamic power consumption.

VI. Conclusion and Future Work

This paper presents a real-time AMT system with Zync FPGAs, enabling one to find specific chords played by users through extracting and analyzing the frequencies from a live recording. Additionally, the hardware performance in terms of slice cost and power consumption has been evaluated. Considering the much higher power cost on programming system compared with the programming logic, our future work will focus on the IP design of frequency transform, PCP, and pattern matching algorithms with HDL. The ideal end goal of this project is a pure hardware implementation on a low-cost and low-power SoC architecture [19], [20], which is able to be taped out as an ASIC for edge devices. We believe that this platform is reusable and expandable to a diverse range of applications in audio processing and speech recognition.

VII. Acknowledgment

This research comes from one of the Undergraduate Senior Projects in Spring 2019 at University of Houston Clear Lake. We thank Mr. Mitchell Jefferies, Mr. Rigo De Leon, and Mr. Harold Mao for assistance with establishing the platform and testing the validity of the real-time system with Guitar music input and sheet music output on a smartphone app.

References

[1] S. Che, J. Li, J. W. Sheaffer, et al., "Accelerating Compute-Intensive Applications with GPUs and FPGAs," 2008 Symposium on Application Specific Processors, Anaheim, CA, pp. 101-107, 2008.
[2] A. M. Caulfield et al., "A cloud-scale acceleration architecture," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, pp. 1-13, 2016.
[3] D. Firestone, A. Putnam, S. Mundkur, et al., "Azure Accelerated Networking: SmartNICs in the Public Cloud," Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2018), pp. 51-64, 2018.
[4] W. Shi, et al. "Edge Computing: Vision and Challenges," IEEE Internet of Things, vol 3, no. 5, pp. 637–646, Oct. 2016.
[5] X. Yang, et al., "A Vision of Fog Systems with Integrating FPGAs and BLE Mesh Network," Journal of Communications (JC) , Vol. 14, No. 3, PP. 210-215, March 2019.
[6] "Zynq-7000 SoC Technical Reference Manual," V1.12.2, Xilinx, July 2018.
[7] "Low Power Audio Codec – SSM2603 Data Sheet," Analog Devices, pp. 19-29, 2018.
[8] X. Yang , N. Wu, and J. Andrian, "A Novel Bus Transfer Mode: Block Transfer and A Performance Evaluation Methodology," Elsevier, Integration, the VLSI Journal, Vol. 52, PP. 23-33, Jan. 2016.
[9] X. Yang and J. Andrian, "A Low-Cost and High-Performance Embedded System Architecture and An Evaluation Methodology," IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2014), PP. 240-243, Tampa, FL, USA, Sept. 2014.
[10] E. Benetos, S. Dixon, Z. Duan, and S. Ewert, "Automatic Music Transcription," IEEE Signal Processing Magazine, pp.21-30, 2019.
[11] T. Fujishima, "Real time chord recognition of musical sound: a system using common lisp music," in Proceedings of the International Computer Music Conference (ICMC1999), pp. 464–467, 1999.
[12] S. Sigtia, E. Benetos and S. Dixon, "An End-to-End Neural Network for Polyphonic Piano Music Transcription," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, no. 5, pp. 927-939, May 2016.
[13] E. Benetos and S. Dixon, "Multiple-instrument polyphonic music transcription using a temporally-constrained shift-invariant model," J. Acoust. Soc. Amer., vol.133, no. 3, pp. 1727–1741, 2013.

[14] B. Fuentes, R. Badeau, and G. Richard, “Harmonic adaptive latent component analysis of audio and application to music transcription,” IEEE Trans. Audio, Speech, Language Process. (2006–2013), vol. 21, no. 9, pp. 1854–1866, 2013.

[15] R. Kelz, M. Dorfer, F. Korzeniowski, S. Bock, A. Arzt, and G. Widmer, “On the potential of simple framewise approaches to piano transcription,” in Proc. Intl. Society Music Information Retrieval Conf., 2016, pp. 475–481.

[16] H. He, et al., “Dual Long Short-Term Memory Networks for Sub-Character Representation Learning,” The 15th Intl. Conference on Information Technology (ITNG-2018), 2018.

[17] T. Palace, “Stand-Alone Device for Chord Detection,” Capstone Design Project, Muse Union, 2015.

[18] G. Husain, W. Thompson, and E. Schellenberg, “Effects of Musical Tempo and Mode on Arousal, Mood, and Spatial Abilities,” An Interdisciplinary Journal, vol. 20, no. 2, pp. 151-171, 2002.

[19] X. Yang and J. Andrian, “A High Performance On-Chip Bus (MSBUS) Design and Verification,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Vol. 23, Issue: 7, PP. 1350-1354, Sept. 2015.

[20] X. Yang and J. Andrian, “An Advanced Bus Architecture for AES-Encrypted High-Performance Embedded Systems,” Patent, US20170302438A1, Oct. 19, 2017.

[21] “Chordify”, 2019. [Online]. Available: https://chordify.net/.

[22] Lunaverus, “Automatic Music Transcription Software,” 2019. [Online]. Available: https://www.lunaverus.com/