The Design and Implementation of a Digital Wireless-Controlled Multi-Effect Guitar Processor Based on Arduino DUE

Jingjie Zhang[1], Wenyuan Qian[2], Junlian Jia[1], Hui Feng[1]
[1]Department of Electronic Engineering, [2]School of Computer Science
Fudan University, Shanghai 200433, China
[jingjiezhang13, wyqian14, jljia10, hfeng]@fudan.edu.cn

***Abstract*—Effect processors are electronic devices that can adjust and enhance the sound of a musical instrument. Thanks to effect processors, electric guitars can produce more expressive tones. With the advent of wireless guitar connecting devices, guitarists' range of movements is no longer limited by cables. However, the location of processors still potentially limits guitarists' range of movements on big stages. Our research will change this situation by splitting the whole guitar effect system into three wireless-connected devices.**

***Index Terms*—Wireless Communication, Digital Audio Signal Processing, Audio Effects, Embedded System**

I. INTRODUCTION

In the seventies music industry, most of the bands that played rock/blues/jazz/fusion and/or electronic music, made use of sound effect pedals for their instruments. Generically called "stomp boxes", these effect pedals are widely used today for their pure analog sound. As the technology evolved, the big brother of the effect pedal appeared in the form of multi-effect digital processor. Examples of such devices are the Fractal Audio Systems Axe-Fx II XL+ [1], Line 6 Helix [2] and many more.

Although these effect processors have brought a monumental revolution to the expression of electric guitars, they still have some limitations in their structures and usage patterns, especially in some occasions of live performances on large stages. As shown in Fig. 1, the black arrows represent the transmission of guitar signals, which is previously realized by 3.5mm Jack cables with finite lengths that limits guitarists' range of movements on large stages.

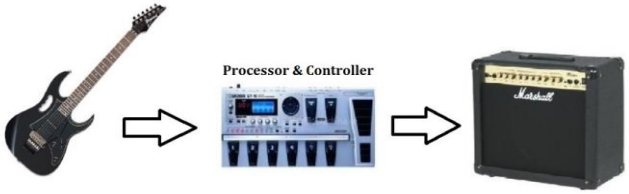

Fig. 1: The usage pattern of a common multi-effect guitar processor

With the advent of wireless guitar connecting devices [3], guitarists' range of movements is no longer limited by cables. However, there is still one potential limitation of movements on large stages – the location of effect processors. We have learned that currently the digital multi-effect processors available on the market all have a similar control form: the control interfaces and processing units of some products are integrated into one single device, while the others are designed to be controlled by an external MIDI foot controller [4]. Either way, guitarists have to stay beside the control interfaces (or foot controller) to maintain control of their guitar tones.

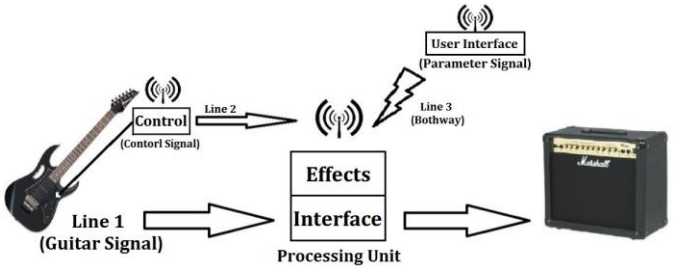

Fig. 2: The prospective structure of our final product

Our research will change this situation by splitting the whole system into three devices (Fig. 2): a processing unit that can be left backstage, a controlling device whose size is minimized to be placed on guitars' front panels and a user interface which is a touch screen for effect adjustments. A three-terminal wireless communication will be established among these devices.

The remainder of the paper is organized as follows: Section II is mainly about our design scheme, which is elaborated in three aspects: hardware, software and wireless communication. The experimental results of our implementation are presented in Section III, while conclusions and future directions are detailed in Section IV.

II. DESIGN SCHEME

In this section, we elaborate our design scheme in three aspects: hardware, software and our transport protocol for a three-terminal wireless communication.

A. Hardware Design

The overall hardware structure diagram of our system is shown in Fig. 3. The user interface consists of a ITDB02-4.3 TFT LCD Display Module [5] from ITEAD Intelligent Systems Co. Ltd., an XBee® ZB RF Module [6] from Digi International Inc. and an Arduino DUE [7] open-source hardware platform, while the controlling device is comprised of four buttons and

LEDs, an XBee® ZB RF Module and an IAP15F2K60S2 microcontroller [8] manufactured by STC MCU Limited. Users can use these buttons to switch among four presets, and the LEDs will indicate the current selection.

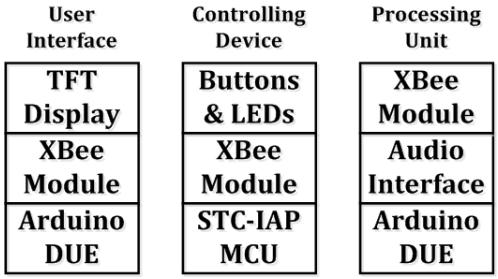

Fig. 3: Overall hardware structure of our system

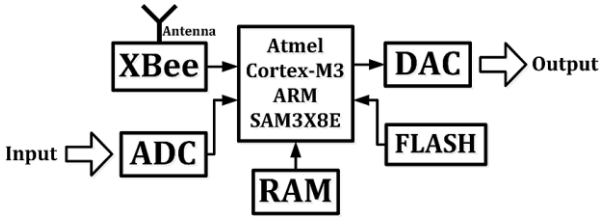

Fig. 4: Hardware diagram of our processing unit

Finally, our processing unit, with another XBee module as well as front-end audio filtering circuits, is also based on DUE because of its easy development and suitable performance with its Atmel SAM3X8E ARM Cortex-M3 CPU [9]. The hardware diagram of our processing unit based on Arduino DUE is shown in Fig. 4. When the analog guitar signal enters this system, it will be converted into digital signal by the embedded 12-bit ADC in DUE and processed in the Cortex-M3 CPU. The parameters used during processing are stored in the RAM, while the program codes are stored in the Flash Memory. All the control commands from the controlling device and parameter signals from the user interface are received by the XBee module. After processing, the digital guitar signal will be converted back to analog signal and sent out of our system.

B. Software Framework & Effect Algorithms

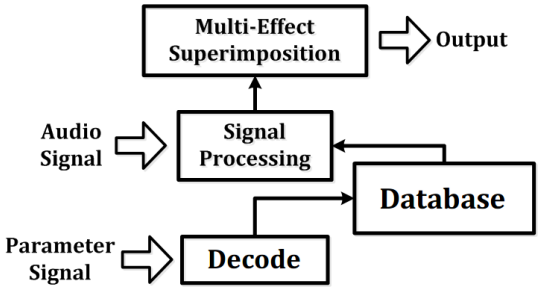

Fig. 5: Software framework diagram

As shown in Fig. 5, at the software level, our guitar audio signal will go through two steps: signal processing and effect superposition. The required parameters and effect sequence information is stored in a database in our processing unit. When the parameter signals from our user interface are received and decoded, our processing unit will make the corresponding modification to the database.

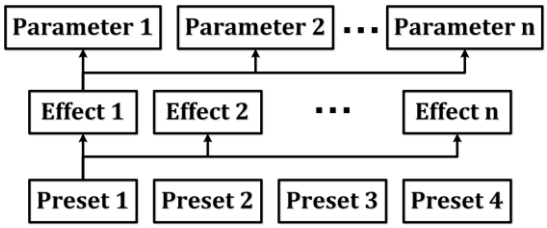

Fig. 6: Structure of our database & menu of user interface

We used a three-dimension array `Database[4][17][10]` to store and manage all the parameters. As shown in Fig. 6, the database has four pages to store four presets, and there can be at most sixteen effects in each page, while each effect can have at most eight parameters. The menu of our user interface is also based on this structure.

In each page of our database, the first column stores the sequence of the serial effect connection, and the second column stores the numbers of the parameters used in the effect of the current row. The actual values of these parameters are stored in the following columns. Once the signal processing begins, our program will start to execute the effect codes following the sequence stored in the first column of the current page per clock cycle. In this way, the superposition of effects is realized.

When one of the four buttons on the controlling device is pushed, a control signal that contains the preset number corresponding to the button will be sent to the processing unit. The page of database that our processing unit is currently working on will be switched to the one corresponding to the received and decoded preset number.

In the remainder of this subsection, the principle of some selected guitar effect algorithms will be elucidated:

1) NoiseGate

For every audio processing system, noise is the problem we must face. There are many ways to eliminate the noise interference, while the *NoiseGate* is one kind of software denoising method for some noises caused by some software effect algorithms, such as overload, distortion, especially when guitarists are not playing (i.e., there is no signal input). Since the input signal is zero, the only output is noise, so it is easy to notice the presence of noise. However, during the playing, noise is not so easy to be noticed because of the high SNR.

If only for the situation above, the implementation of *NoiseGate* is very easy, we can just cut the input signal when it is lower than a certain electrical level. But in fact there is another case, that is, the attenuation of the last note of a song. If we directly remove the signal, the last note will disappear immediately, which sounds unnatural. Therefore, some *NoiseGate* products on the market have developed a function by which we can adjust the speed of attenuation, and this function is the key point in our *NoiseGate* implementation.

After many failed attempts, our eventual scheme is: First, we converted all the input signals into positive amplitude values, (i.e., full-wave rectification). Therefore:

$$\overline{V} = 0.6366 V_{peak} \tag{1}$$

The lowest signal frequency is 100 Hz, so the maximum

signal cycle is 10 milliseconds, spanning 441 sampling points. Thus, we built a buffer with the length of 441 to real-timely record the values of 441 samples in history. Attenuation will be triggered once the inequality below is satisfied:

$$\sum_{i=1}^{441} x[n-i] < 441 \times 0.6366 \times Threshold \quad (2)$$

In the attenuation process, each input signal will be multiplied by an exponential decaying sequence as coefficients. The end of attenuation is determined by coefficients instead of input signals. Once the current coefficient is smaller than a certain value, the attenuation will be terminated. Then, the small signal left will be cut directly.

2) Delay

Delay is an effect which adds echoes to the original sound signal. Fig. 7 below shows a simple version of *Delay* algorithm: the input signal is stored in a buffer and will be mixed with a subsequent signal sometime later.

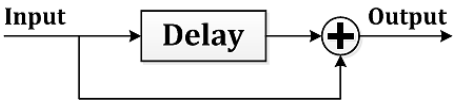

Fig. 7: A simple algorithm of *Delay*

However, the algorithm above can only implement one echo. Xin [10] state that for a complete *Delay* effect (shown in Fig. 8.), there must be a feedback control. The current output signal should be mixed with the current input signal and stored in the buffer again. Then a repetition of sound can be realized. The echo will gradually decays according to the feedback gain, until finally disappears. If the gain is l, the sound will be infinitely delayed, until the gain is adjusted.

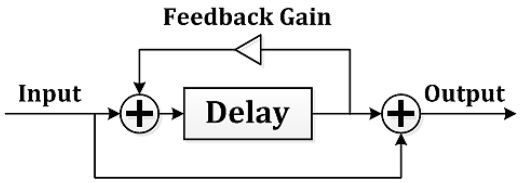

Fig. 8: The complete algorithm of *Delay*

Our *Delay* effect is based on the above algorithm with adjustable feedback gain. We established a buffer with length of 0.6 seconds (26460 sample points). The delayed signal is mixed with the current input signal and sent to output. in the meantime, it is also multiplied by a feedback gain (less than 1) to mix with the current input signal and then stored into the buffer, in order to form more than one echo.

3) Tremolo

According to Xin [10], *Tremolo* is an effect based on amplitude variation, which is relatively simple to realize. The typical *Tremolo* effect (Fig. 9) can periodically alter the output volume, which brings a feeling of vibration.

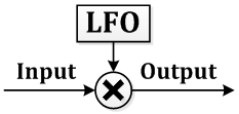

Fig. 9: The *Tremolo* algorithm

Again, we converted all the input signals into positive amplitude values, and multiplied them by the coefficient below:

$$\frac{y[n]}{x[n]} = A \sin\frac{2\pi n}{N} + 1 - A \quad (3)$$

In the equation above, A is the amplitude of modulation wave created by the low-frequency oscillator, while n is a counter which is continually increasing, and N is the number of samples corresponding to the period of modulation wave. The results of the sine function are acquired by a look-up table.

4) Chorus

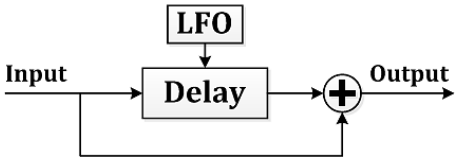

Fig. 10: A simple algorithm of *Chorus*

At first, *Chorus* effect is created to simulate a group of players playing the same instrument at the same time, in order to increase the thickness of the sound and make it sounds warmer. Although players are playing the same music at the same time, they certainly cannot do it exactly the same, and this slight difference here is simulated by periodically changing the length of the delay buffer according to the LFO [10].

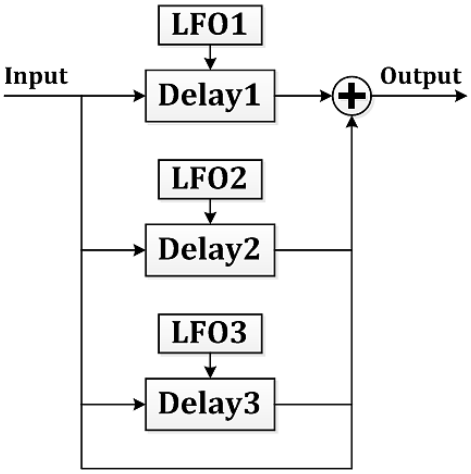

Fig. 11: The complete algorithm of Chorus

According to the algorithm in Fig. 10, we can implement that the delay length varies from 20 to 30 ms periodically. However, Gordon [11] pointed out that human ear is sensitive, so only using one buffer to create one modulation signal is not enough to fool our ears. In fact, in order to realize the simulation of the ensemble effect, all the *Chorus* effect pedals on market always use multiple delay lines and make the modulation signal in different phases. Therefore, according to Anghelescu, Angelescu and Ionita [12], Fig. 11 shows the final *Chorus*

algorithm we used.

5) Flanger

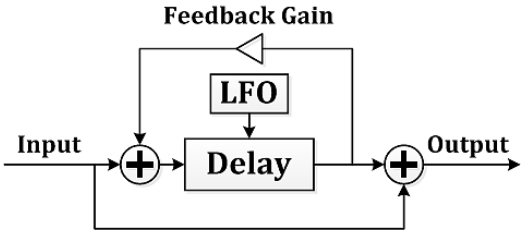

Fig. 12: The *Flanger* algorithm

Flanger is a distinctive effect. It can be understood as the feeling of a jet flying over the head. According to Xin [10], the principle of *Flanger* effect is similar to the *Chorus* effect (shown in Fig. 12.). The only difference is the delay time. *Chorus* generally has a longer delay than *Flanger*, between 20 to 30 ms, while *Flanger* is l to 10 ms.

6) Vibrato

Fig. 13: The *Vibrato* algorithm

Similar to *Chorus* and *Flanger*, *Vibrato* (Fig. 13) is also classed as delay line modulation effects, which are achieved through the continuous adjustment of delay line lengths. However, the biggest difference between *Vibrato* and *Chorus* is that *Vibrato* algorithm only output wet signals. In actual implementation, we use the triangle wave rather than the sine wave in the delay line modulation, because the triangle wave modulation appears to have lower noises.

7) Reverb

A reverberation occurs naturally when a sound or signal is reflected causing a large number of reflections to build up and then decay as the sound is absorbed by the surfaces of objects in the space. Digital reverberators use various signal processing algorithms in order to artificially create the *Reverb* effect.

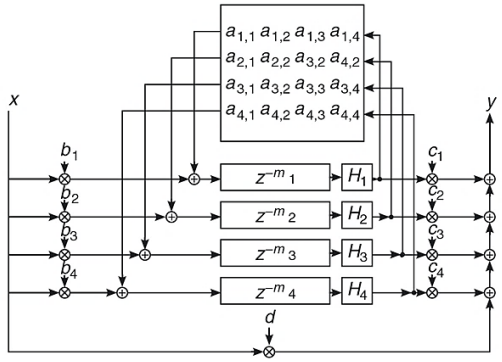

Fig. 14: Fourth-order feedback delay network [13]

Among them, we chose a relatively better *Reverb* algorithm: Feedback Delay Network (FDN). In FDN reverberators, only the important parts of the auditory perception is simulated, which reduces the computational complexity while producing high-quality reverberation. Fig. 14 shows the diagram of a fourth-order FDN reverberator [13], which is implemented in our product.

According to Smith [14], in reverberators based on FDN, the smoothness of the "perceptually white noise" generated by the impulse response of the lossless prototype is strongly affected by the choice of FDN feedback matrix. As one of the better known feedback-matrix choices, a fourth-order Hadamard matrix is chosen by us:

$$H_4 \triangleq \frac{1}{\sqrt{2}}\begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad (4)$$

Smith [14] stated that when n is a power of 4, the Hadamard matrix H_n of that order requires no multiplies in fixed-point arithmetic. A $n \times n$ Hadamard matrix has the maximum possible determinant of any $n \times n$ complex matrix containing elements which are bounded by 1 in magnitude. This can be seen as an optimal mixing and scattering property of the matrix.

Following Schroeder's original insight [15], the delay line lengths are typically chosen to be co-prime. Thus, we first determine the maximum and minimum delay lengths by:

$$\begin{cases} Delay_{max} = \dfrac{SR \cdot Path_{max}}{c} \\ Delay_{min} = \dfrac{SR \cdot Path_{min}}{c} \end{cases}, \quad (5)$$

with the sample rate *SR*, and the maximum and minimum paths of sound propagation $Path_{max}$ and $Path_{min}$, while *c* is the velocity of sound, which is around $343\,m/s$ in the dry air at $20℃$. After $Delay_{max}$ and $Delay_{min}$ is determined, the *i*th of *N* desired delay line lengths is given by:

$$Delay_{i\,(desired)} = Delay_{min} \cdot \left(\frac{Delay_{max}}{Delay_{min}}\right)^{\frac{i}{N-1}} \quad (6)$$

Therefore, we choose each delay-line length $Delay_i$ as an integer power of a distinct prime number $p_i \in \{2, 3, 5, 7, \ldots\}$:

$$Delay_i \triangleq p_i^{m_i} \text{ with}$$
$$m_i \triangleq \left\lfloor 0.5 + \frac{log\left(Delay_{i\,(desired)}\right)}{log(p_i)} \right\rfloor \quad (7)$$

C. Wireless Communication Protocol

We designed a transmission protocol to coordinate the three-terminal wireless data transmission between our processing unit, controlling device and the user interface. Designed on top of ZigBee® [16] communication protocol, our communication packages are all embedded in the ZigBee® communication packages. Table 1 shows the basic command

frame of our communication packages.

Table 1: Basic command frame of our communication packages

	HEAD	TYPE	Data[n], n = PkgLen-40	TAIL
Length(Bit)	16	8	n	16
Description	0xFBFB	Types	Effect codes, parameters	0xFEFE

Each communication package starts with a 2-byte HEAD whose value is "0xFBFB" and ends with a 2-byte TAIL as "0xFEFE". The content of a package is comprised by two parts: a 1-byte TYPE of this package and the Data it carries whose length is varied according to the package's TYPE. In our wireless communication protocol, we have six TYPEs altogether, which involve six different operations such as switching, confirming, inquiring, saving, etc.

In our wireless network, the processing unit serves as a central coordinator, which sends its communication packages in broadcast mode, while our controlling device and user interface only respond to packages with specific TYPEs they recognize, and all the answering packages are sent only to the processing unit.

III. Implementation Results

A. *Hardware Demonstration*

Our processing unit (Fig. 15) is comprised of three layers that are fixed and connected by two-way hexagon copper pillar through screw holes. The top-layer PMMA sheet designed by CAD software and the middle-layer PCB designed by EDA tools both have matched shape and screw holes' positions with the Auduino DUE at the bottom.

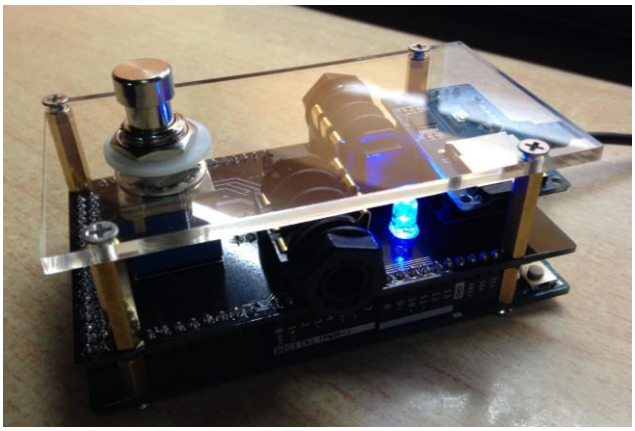

Fig. 15: Processing Unit

The controlling device (Fig. 16) is design to be portable and dismountable, which is achieved by compact PCB design as well as three suction cups. Components with smaller sizes (e.g. the buttons with built-in LEDs) are deliberately chosen to minimize the size of its PCB. The bottom-layer PMMA sheet is also designed to have matched screw holes' positions with top-layer PCB so that a 3.7V lithium battery can be installed in between the two layers.

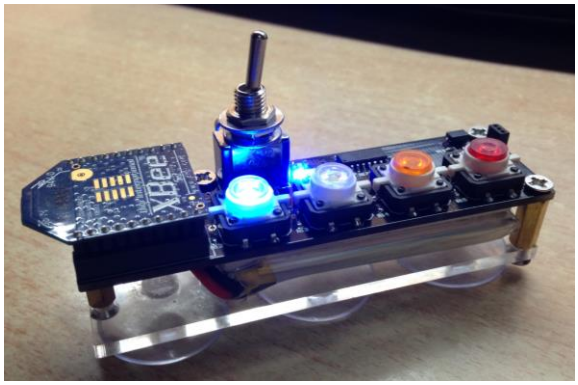

Fig. 16: Controlling device

On a PMMA sheet which is design to have a matched shape with the ITDB02-4.3 TFT LCD Display Module, the touch screen and another Arduino DUE are fixed by two-way hexagon copper pillar through matched screw holes and thus comprise our user interface (Fig. 17).

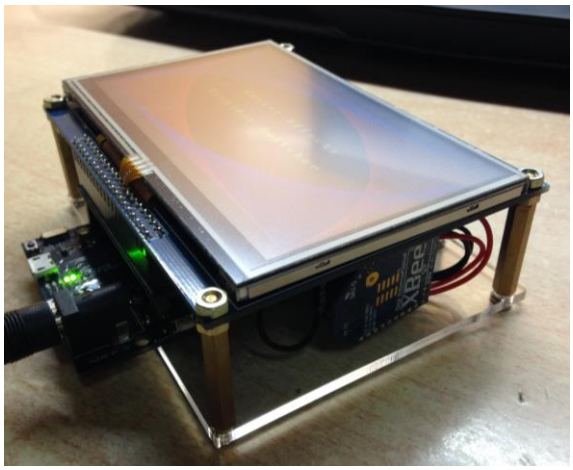

Fig. 17: User interface

B. *GUI Demonstration*

Fig. 18: Startup screen of user interface

After the startup screen (Fig. 18), our user interface will show a menu (Fig. 19) based on the menu structure in Fig. 6. Four specific colors are used to mark the four presets and their sub-effects to help the user to distinguish between the four different presets.

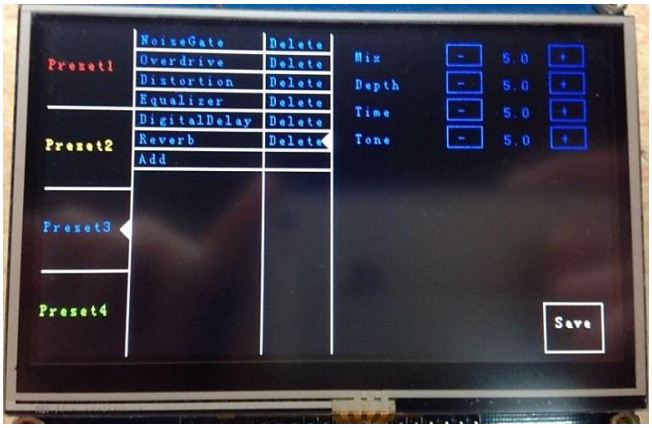

Fig. 18: GUI implementation result of the effect-editing interface

C. Guitar Effect Waveforms

1) NoiseGate

We can see the obvious turning point of the attenuation tendency at the position of the cursor in Fig. 19.

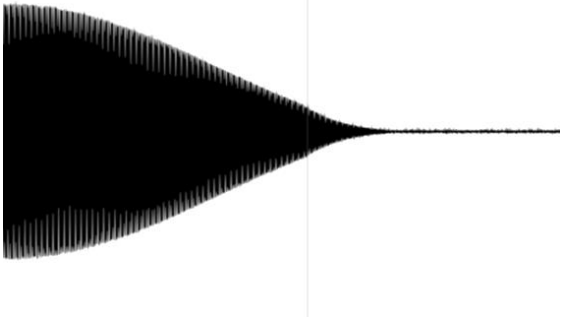

Fig. 19: Output waveform of *NoiseGate*

2) Delay

Fig. 20 shows the original signals and the echoes brought by the *Delay* effect.

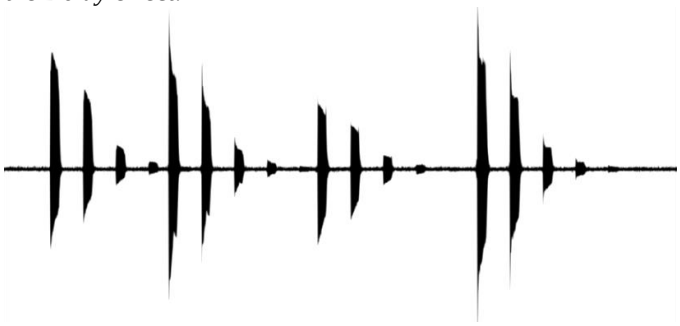

Fig. 20: Output waveform of *Delay*

3) Tremolo

The output waveform (Fig. 21) of the *Tremolo* effect demonstrates the typical characteristics of Amplitude Modulation (AM).

4) Chorus

Compared to the original signal on the left, the output signal of *Chorus* in Fig. 22 demonstrates notable variations involving some periodical features.

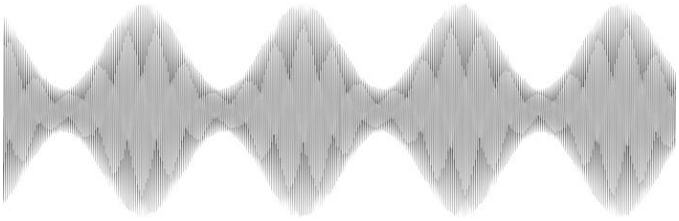

Fig. 21: Output waveform of *Tremolo*

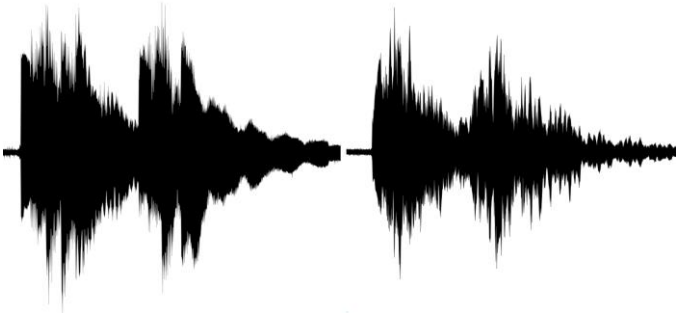

Fig. 22: Input and output waveform of *Chorus*

5) Flanger

Fig. 23 shows the comparison of the input and output signal of *Flanger*. The output waveforms of *Flanger* and *Chorus* appears to have similar features because of similar principles.

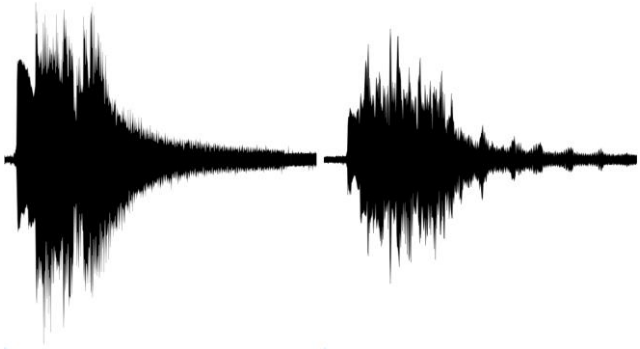

Fig. 23: Input and output waveform of *Flanger*

6) Vibrato

The auditory features of *Vibrato* involve the periodical continuous variation of guitar signal's pitch, however, which is hard to be distinctly observed in its waveforms (Fig. 24).

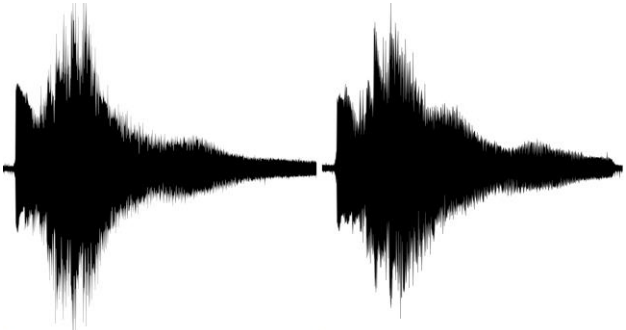

Fig. 24: Input and output waveform of *Vibrato*

7) Reverb

As shown in Fig. 25, our FDN reverberator succeeded in producing a desired reverberation for the input signal on the left artificially.

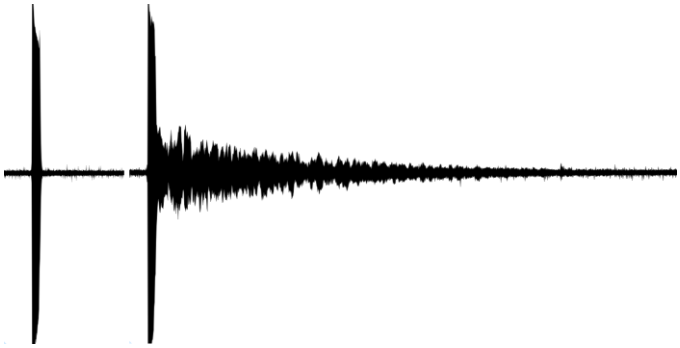

Fig. 25: Input and output waveform of *Reverb*

IV. Conclusion

Our work has successfully implemented a three-terminal wireless-controlled digital guitar sound processing system with a brand new control form – wireless control by buttons, which truly resolves the potential problem of guitarists' limited range of movements on big stages.

However, during the implementation, we have found that the central processing chip we used (Atmel SAM3X8E ARM Cortex-M3 CPU) does not have the capability to handle some complicated audio effects, as a good implementation of audio effects must be done with a dedicated hardware such as digital signal processors (DSPs) or FPGA devices. Thus, our future research direction will be embedding the universal MIDI protocol into the wireless control of some top-level guitar effect processors on the market.

As for the realization of our three-terminal wireless communication, our results still can be further improved. First, since our wireless protocol is based on the RS-232 serial communication, the smallest division of each command frame is a byte, which wastes part of the bits in our packages. Second, our basic strategy of anti-interference check is to resend, but actually there are many other ways to ensure accuracy.

What's more, while it is convenient to use a specialized TFT touch screen to develop our user interface, the GUI development on Arduino platform is still far from maturity and aesthetics. Mobile phones, on the other hand, have a mature GUI-developing system and a ready-made Bluetooth wireless communication support. Thus, the development direction of our user interface is the GUI development on mobile phones.

Acknowledgment

This work was supported by *Tengfei Program* of Tengfei College of Engineering, Fudan University and *Xiyuan Program* in Fudan's Undergraduate Research Opportunities Program.

References

[1] Fractal Audio Systems. (2014). Axe-Fx II: Owner's Manual (All Models) [Online]. Available: http://www.fractalaudio.com/downloads/manuals/axe-fx-2/Axe-Fx-II-Owners-Manual.pdf

[2] Line 6, Inc. (2016). Helix 2.0: Owner's Manual [Online]. Available: http://line6.com/support/manuals/helix

[3] Line 6, Inc. (2013). Relay G90: Receiver Pilot's Guide [Online]. Available: http://line6.com/support/manuals/relayg90

[4] Roland Corporation. (2007). FC-300: Owner's Manual [Online]. Available: https://www.roland.com/global/support/by_product/fc-300/owners_manuals/

[5] ITEAD Intelligent Systems Co. Ltd. (2015). ITDB02-4.3 - ITEAD Wiki [Online]. Available: https://www.itead.cc/wiki/ITDB02-4.3

[6] Digi International Inc. (2014). XBee®/XBee-PRO® ZigBee® RF Module User Guide [Online]. Available: http://www.digi.com/resources/documentation/DigiDocs/90002002/

[7] Arduino. (2012). Arduino DUE [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardDue

[8] STC MCU Limited. (2015). STC15 Series MCU Datasheet [Online]. Available: http://www.stcmcu.com/datasheet/stc/STC-AD-PDF/STC15-English.pdf

[9] Atmel Corporation. (2015). SAM3X Series Datasheet [Online]. Available: http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf

[10] W. Xin. "The Research of Guitar Effect Algorithm and Design of an Effect Processor," M.S. thesis, Department of Software Engineering, Fudan University, Shanghai, China, 2005.

[11] R. Gordon. (2004, Jun.). More Creative Synthesis with Delays. *Sound on Sound (SOS)* [Online]. 62nd in 63. Available: http://www.soundonsound.com/techniques/more-creative-synthesis-delays

[12] P. Anghelescu, S. Angelescu and S. Ionita, "Real-time audio effects with DSP algorithms and DirectSound," Proceedings of the 2014 6th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Bucharest, 2014, pp. 35-40.

[13] U. Zölzer. eds. DAFX: Digital Audio Effects, Second Edition. New York, NY: Wiley, 2011, pp. 169-175.

[14] J. O. Smith, Physical Audio Signal Processing for Virtual Musical Instruments and Audio Effects, 2010 edition. [Online]. Available: https://ccrma.stanford.edu/jos/pasp/

[15] M. R. Schroeder and B. F. Logan. (1961, Jul.). Colorless artificial reverberation. Journal of the Audio Engineering Society [Online]. vol. 9, pp. 192-197. Available: http://www.aes.org/e-lib/browse.cfm?elib=465

[16] ZigBee Alliance. (2016). Network Specifications [Online]. Available: http://www.zigbee.org/zigbee-for-developers/network-specifications/