Ann. For. Res. 65(1): 9368-9379, 2022
ISSN: 18448135, 20652445

ANNALS OF FOREST RESEARCH
https://www.e-afr.org/

FPGA-BASED SOUND FIELD MODULATION METHOD FOR MIDI MUSIC PRODUCTION

Tiancheng Yang
School of Music and Dance,Yunnan Normal University,Kunming,650500,China
510391565@qq.com

Abstract: Current sound field modulation methods mainly start from a single parameter of the music and achieve sound field modulation by changing the parameter values. This modulation method makes no obvious sound pressure difference within the sound field, which leads to poor musical directivity. With respect to the above problem, the FPGA-based sound field modulation method for MIDI music production is studied. After designing the FPGA-based sound field modulator, the MIDI music is pre-processed in the top module of the modulator, and the musical features and the main melodic contour of the music are extracted. According to FM modulation theory, the sound field modulation of MIDI music is implemented. The comparison experimental results show that the measured sound pressure of the modulated music by the studied method is consistent with the simulated value, and the frequency is above 2.5 KHz with good pointing effect.

Keywords: FPGA; MIDI music; music production; sound field modulation; main melody contour extraction; FM modulation;

0 Introduction

MIDI is a communication standard for digital music language, which allows music performance data information to be recorded in bytes on a computer. The sound field in MIDI music production is formed by artificial modulation in a virtual environment, and its sound space is generated by digital means in a computer, which has various characteristics such as adjustable, generative, simulable, and creative, and contains both the objective properties of sound and also the auditory subjective properties of evaluation [1-2]. In the post-mixing process of MIDI music composition, sound field modulation needs to be performed according to the basic criteria of objective operation and subjective auditory evaluation in order to make the purpose of sound field modulation clear. Sound field modulation can avoid the interference caused by the overlapping of sound fields corresponding to different tracks during the synthesis of MIDI music, which affects the audibility of MIDI music. Currently, sound field modulation in MIDI music production relies on the combination of MIDI controllers and other technologies. MIDI controllers are specifically divided into hardware MIDI controller devices and software MIDI controllers [3]. Their same role simply means that they can control the change of certain changing parameters of the music through these external controller devices or soft controllers. Hardware MIDI controllers can be connected to external MIDI controller devices that dynamically send signals to control certain MIDI messages and changes in some parameters to achieve variability in the processing of music production [4]. The software MIDI controller is set up by the host software [5]. In contrast, it is simpler and more efficient to use a hardware controller for sound field modulation in MIDI music production. The traditional sound field modulation method is mainly from adjusting the amplitude, frequency, and the strength and weight of the notes of the music signal, which not only has

subjectivity in modulation, but also leads to music modulation with no obvious directionality [6-9]. For this reason, based on the above analysis and discussion, this paper will study the FPGA-based sound field modulation method for MIDI music production.

1 FPGA-based sound field modulation method for MIDI music production

1.1 FPGA-based sound field modulator design

The FPGA-based sound field modulator architecture consists of a read/write circuit module (MIDI_rw), a file end detection circuit module (MIDI_check), a data comparison circuit module (MIDI_cmp), a data exchange circuit module (MIDI_exchange), an address accumulation circuit module (MIDI_addrp), and a top-level module (MIDI_top). The specific structure is shown in figure 1 below [10].

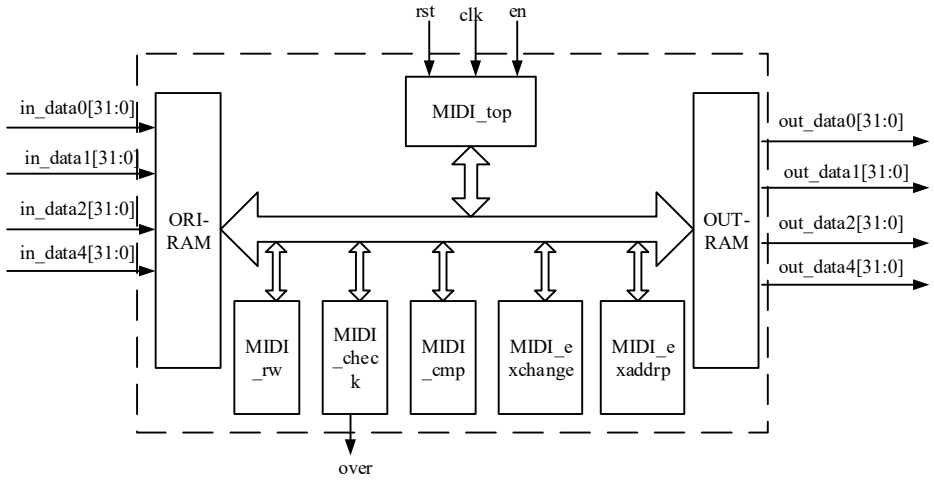

Figure 1 Structure of the sound field modulator

Among them, the read/write module is mainly responsible for controlling the input of the original note feature matrix and the output of the target note feature matrix at the end of the algorithm. When the read/write enable is high, it starts to enter the read/write state. First, the read/write mode is judged. When the in_sel_rd signal is high, it means that it is now read state, and in_sel_mem is selected as output RAM first, then the read address is accumulated, and the start time, duration, end time, pitch and speed are read in turn. When the in_sel_rd signal is low, it means that it is now write state, and in_sel_mem is selected as input RAM first, and then the data is judged, and writing can be done only when it is confirmed that the data is not empty. Next, the end time of the note should be calculated, and the duration and start time should be accumulated to finally get the five data that can be written to the RAM. Finally, the address of the input RAM is totalized, and the data is written in order.

The end-of-file detection module is responsible for detecting whether the end of the file has been read while ensuring that the data is valid, facilitating the determination of the completion of the algorithm. The CHECK_DATA4 state detects if the in_data4 signal is 0. The result is stored in register 4, and then the CHECK_DATA3 state detects if the in_data3 signal is 0, again using a per-bit or. When all five data are detected, the register set of the by-or result is then operated by the by-or operation, and the result is assigned to the output out_check_result, and the marker signal

out_check_end is pulled up to enter the CHECK_END state [11-12]. The top-level module extracts the main melodic contour of MIDI music mainly by enabling other modules and controlling the input and output to achieve the sound field modulation for MIDI music production.

1.2 MIDI music pre-processing

MIDI files contain up to sixteen channels, with channel numbers 0-15, and each channel is played independently of the other. There are 128 MIDI timbres available, of which only one timbre can be used per channel, and different timbres can be used between channels. The file structure of MIDI is shown in figure 2 [13].

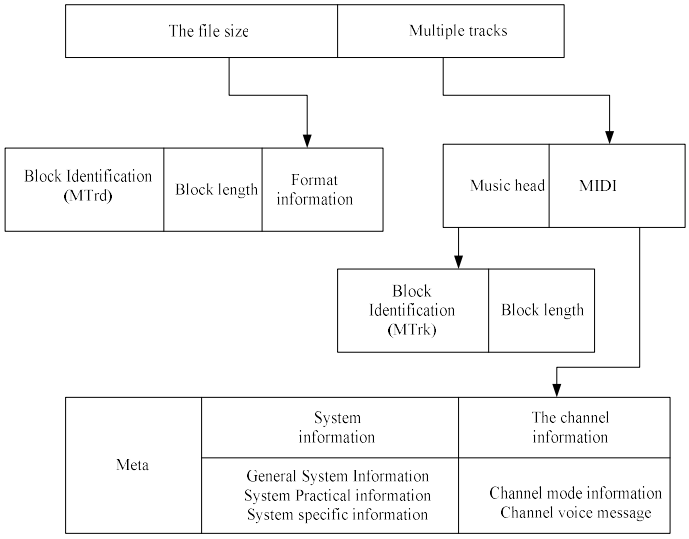

Figure 2 MIDI file structure

In figure 2, the file header has the form of the following hex code: 4d _54 68 64 |00 00 00 06 | xx xx |yy yy | zz zz [14]. Each field is described separately below.

"4d _54 68 64": these hexadecimal numbers are converted to ASCII C American Standard Code for Information Interchange (ASCII C) character code as "MThd", which represents the beginning of a MIDI file, which is the same in every MIDI file. It is a criterion to determine if the file is a MIDI file.

"00 00 00 06": this hexadecimal number represents the number of bytes occupied by the file header, excluding the preceding identifier and the hexadecimal number itself. This is also fixed for each MIDI file as far as the current MIDI standard is concerned.

"xx xx": this parameter defines the file format type of MIDI." 00 00" represents format 0, indicating that there is only one track." 00 01" represents format 1, indicating that multiple tracks exist and that they are played in sync." 00 02" represents format 2, indicating that multiple tracks exist and that these tracks are played asynchronously.

The parameter "yy yy" represents the number of track blocks. For a format 0 MIDI file, the parameter is 1, because only one track exists. For format 1 MIDI files, this parameter is equal to the actual number of tracks plus 1, because there is usually a global track, where the global track records basic information about the music, such as tempo, track name, etc.

"zz zz":Indicates the basic time format. When the highest bit is 0, it represents the number of ticks contained in 1/4 notes. When the highest bit is 1, it represents the number of ticks contained in each SMTPE frame. Where tick is the minimum time scale of a MIDI file.

The second part of the MIDI file consists of track blocks, each of which is composed of a track header and a number of note data. The track header is in hexadecimal form: 4d _54 72 6b | xx xx xx xx, where "4d _54 72 6b" is converted to ASCII as "MTrk", and whenever this is detected, it marks the start of a new track block. The following "xx xx xx xx" indicates the number of bytes occupied by the current track, but it should be noted that the track headers are not included in the calculation of the number of bytes occupied.

The note data is composed of <delta-time> + <event>. Where the delta-time unit is the tick, which is the smallest unit of time specified in the MIDI header block, and represents the time difference between two adjacent events of the note data as that event minus the time of the previous event. Event consists of MIDI events, which consist of a status byte and a number of data bytes. The main MIDI events are shown in table 1, where the last four bits of the status byte represent the channel number [15].

Table 1 Table of major MIDI events (partial)

Midi events	Status byte	Data byte	Data byte description
Note off	1000 xxxx	0xxx xxxx	Note number
		0xxx xxxx	Release key strength
Note on	1001 xxxx	0xxx xxxx	Note number
		0xxx xxxx	Key strength
After the polyphony button	1010 xxxx	0xxx xxxx	Note number
		0xxx xxxx	Continuous pressure after key press
Control information	1011 xxxx	0xxx xxxx	Control number
		0xxx xxxx	Type of control
Channel method	1011 xxxx	0xxx xxxx	122~127 channel method
		0xxx xxxx	Type of control
Tone adjustment	1100 xxxx	0xxx xxxx	Tone number
Channel after-touch pressure	1101 xxxx	0xxx xxxx	Pressing the key back channel pressure
		0xxx xxxx	Low 7-bit conversion value
Bend transformation	1110 xxxx	0xxx xxxx	High 7-bit conversion value

Convert the MIDI format file to hexadecimal text as described above. After the initial processing of MIDI music, extract the feature matrix from the MIDI music.

1.4 Extracting the MIDI music feature matrix

In the track block, the note data is composed of <delta-time> + <event>. In the calculation of the note feature matrix, as soon as delta-time can be calculated, the start and end times of each note can be calculated, and then the corresponding MIDI events can be analyzed to get information about the pitch and speed of each note.

In actual MIDI files, the duration of some notes can be particularly long due to the presence of background tones and trailing notes. In order to meet the requirements of delta-time, which can be represented by up to four bytes, a fixed-length encoding would make the entire MIDI file redundant, so a variable-length encoding is used for delta-time, with bit7=1 for the high byte and bit7=0 for the low byte. This makes the calculation of delta-time complicated [16].

For a normal byte, there are 8 bits to represent the data, and MIDI specifies the highest bit of delta-time as the flag bit, so only 7 bits can actually be used to represent the size of the data, and can represent a number in the range 0 to 127, beyond which multiple bytes are needed to represent it. One of the conversion tables for variable length encoded numbers to actual data is shown in Table 2 [17].

Table 2 Conversion table of variable length coded data to actual data

Actual data		Variable length encoded data	
Hexadecimal	Binary system	Binary system	Hexadecimal
00	0000_0000	0000_0000	00
...	...	...	...
...	...	...	...
7f	0111_1111	0111_1111	7f
80	00000000_10000000	10000001_00000000	80 00
...	...	...	...
...	...	...	...
3fff	00111111_11111111	111111111_01101000	Ff 7f
02 e4	00000010_11100100	10000101_01100100	85 64

Pointer traversal is used to identify MIDI commands and MIDI data separately, then the MIDI commands are parsed, and then the MIDI data is analyzed according to the parsed results to obtain the note information of the entire MIDI file.

First, find the first data of the first MIDI event, determine if its value is less than 128, if it is greater than that, then prove that it is not the last bit of delta-time, save it and then compare the next bit until the length of the bytes of delta-time is determined, then reduce it to the actual delta-time value according to the rules of variable length encoded data. Next, analyze the MIDI commands as mentioned earlier. If the commands of two adjacent MIDI events are the same, then the commands can be omitted and the MIDI data directly followed by the delta-time, so if *p is less than 128, then it proves to be the same as the previous command and sends the last two bits of data directly. If *P is greater than 128, determine if it is a program change command

(191<*p<208) and channel aftertouch pressure (207<*p<224), if so send two bits of data, if not send three bits of data, so you get the delta-time information as well as the note information. Calculate the note lengths of all notes according to the following equation.

$$T(i) = Ts(i+1) - Ts(i), i = 1, 2, \ldots, N \quad (1)$$

Where, $T(i)$ is the length of note i; $Ts(i)$ is the start time of the note i. In addition, MIDI generally specifies its own minimum time unit tick in Meta events, so when calculating the length of a tone, the time unit calculated directly through MIDI data stream is tick. By calculating the time occupied by a tick is 1391 μs. After extracting the length, pitch, speed of sound, channel and other information of each note, a note feature matrix of MIDI file can be established through them. The process of extracting note feature matrix is shown in figure 3 [18].

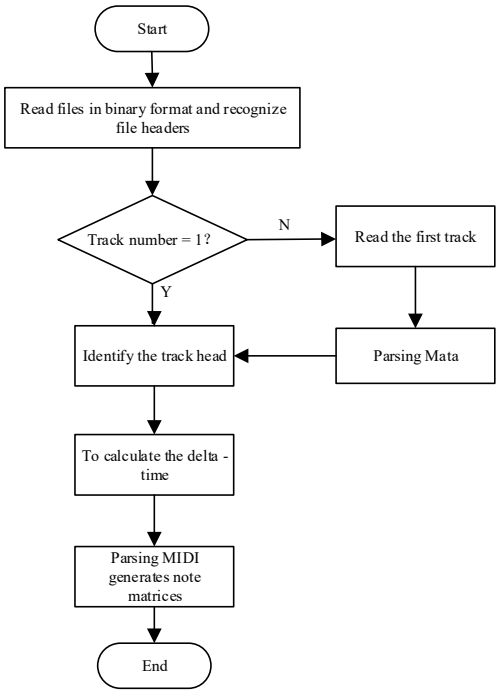

Figure 3 Creation of the note feature matrix

Reads the MIDI file in binary format. Since the header of a MIDI file is fixed, it is determined that the file has been opened correctly. Then you need to determine the number of tracks, which are contained in the header block. For MIDI files with a track count other than 1, i.e. if there are multiple tracks, the first track is usually recorded as a Meta event with some global information, such as the track name, instrument name, note tempo, beat information, etc., so we need to record this information before proceeding to the next step of basic note extraction. For a track number of 1, the basic note information is recorded directly in that track, so you can go directly to the basic note extraction.

In the basic note extraction, the track head is first determined to determine the starting position of the pointer traversal. Next the delta-time is calculated according to the flow in figure 3 to determine the start time and end time of each note. After the delta-time is thrown out, the rest of the content consists of MIDI events, which have a very fixed format of a status byte plus a number

of data bytes. The status byte represents the actual action, which is determined by a table lookup operation. The data byte represents some basic information about the note, such as pitch, timbre, velocity, etc. After extracting this information, the useful parts are written into the note feature matrix and the main melody contour is extracted. Based on the note feature matrix, the MIDI music sound field modulation is implemented using the FM modulation principle.

1.4 MIDI main melody contour extraction

The process of extracting the main theme contours is as follows.

The average pitch height of each track is first calculated as follows [19].

$$\overline{p}_i = \frac{\sum_{j=1}^{n} p_{ij}}{n} \quad (2)$$

Where, $\overline{p}_i$ is the average pitch value of track c_i; n is the total number of notes in a track c_i. It is necessary to make statistics of the pitch of the notes in each track block and get a distribution from 1 to 127. After the average pitch height and pitch statistics of each track are calculated, the clustering algorithm projects the pitch values of the notes on each track into a 12-dimensional note statistics table shown. The rule for projection is to divide modular twelve according to the law of equal twelve. Calculate the average value of 12-dimensional notes in the statistical table of the track as follows:

$$\overline{h}_i = \frac{\sum_{k=1}^{16} h_{ki}}{T} \quad (3)$$

Where, T is the total number of tracks with at least one valid note; h_{ki} is the average value of the statistical table of 12-dimensional notes of track k. After the statistical table of 12-dimensional notes of each track is finished, the operation of clustering is carried out. Calculate the distance between the 12-dimensional note statistics table and the average value of each track to be used as the clustering standard, and the calculation formula is as follows:

$$edist_j = d\left(h_i, \overline{h}\right) = \sqrt{\sum_{i}^{12} \left(h_{ij} - \overline{h}_j\right)^2} \quad (4)$$

Where, $edist_j$ is the distance between track i and the average value; $\overline{h}_j$ is the average of the tracks. After calculating the distance of the twelve dimensional note statistics table for each track, it is saved in array D. Arrange the data in D from low to high, categorize all tracks in MIDI files by setting one or more minimum distance difference thresholds. The threshold can be calculated using the following formula:

$$\xi_{edist} = \frac{d\left(\overline{h}_w, \overline{h}\right)}{2} \quad (5)$$

Where, weighted average 12-dimensional pitch statistics are as follows:

$$h_{w_i} = \sum_{k=1}^{16} h_{ki} f_i \quad (6)$$

In the above formula, f_i is the weight value of each track.

According to the average pitch value of each track and the entropy of the track, the main melody is selected. The calculation formula is as follows:

$$v_i = \overline{p}_i + H\left(c_i\right)w \quad (7)$$

Where, v_i is the score of the sound track; $\overline{p}_i$ is the average pitch value of the track, which ranges from 0 to 127. $H\left(c_i\right)$ is the entropy of sound track, which ranges from 0 to 1. w is a weight ranging from 100 to 140. Therefore, if w is less than 127, the average tonal value has a higher weight, if w is 127, the weight is the same, and if w is greater than 127, the Euclidean distance from the whole is more important.

The calculation method of track entropy is as follows:

$$H\left(c\right) = -\sum_{i=1}^{12} p\left(h_i\right)\log_2 p\left(h_i\right) \quad (8)$$

Where , $p\left(h_i\right)$ is the probability of h_i as the main melody note, and the calculation formula is as follows:

$$p\left(h_i\right) = 1 - \frac{edist\left(h_i\right)}{edist_{\max}} \quad (9)$$

For each cluster after clustering, the track with the largest score is selected as the representative track, and then all the selected tracks are put together, and then the results are changed to mono using the improved contour line algorithm to obtain the main melody of the sound field modulation.

1.5 Implementing MIDI music production sound field modulation

On the FPGA-based modulator, MIDI music after initial processing can be achieved by adjusting the frequency modulation circuit parameters. The FM operation unit is located in the top-level module of the FPGA modulator, as shown in figure 4 below [20].

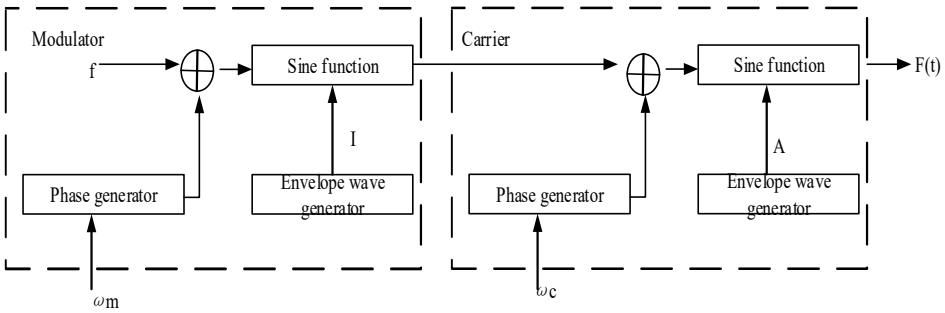

Figure 4 FM operating unit

When both the modulating and carrier signals are sine waves, the modulated output signal can be expressed as

$$F\left(t\right) = A\cos\left(\omega_c t + I\sin\omega_m t\right) \quad (10)$$

Where, $F(t)$ is the instantaneous amplitude of modulated wave output frequency; A is carrier amplitude, ω_c is carrier frequency; I is the modulation amplitude (modulation index); ω_m is the modulation frequency. For the sound field modulation of MIDI music, which is a complex signal, the following formula is applied:

$$F(t) = A\cos\left(\omega_c t + I_1 \sin\omega_m t + I_2 \sin\omega_m t + \ldots + I_n \sin\omega_m t\right) (11)$$

In the above formula, $I_1, I_2 \ldots, I_n$ represents the carrier signal corresponding to each track during MIDI music production. In the process of sound field modulation, the amplitude of modulated carrier is determined according to MIDI note feature matrix. Fourier transform is applied to the above formula, and feedback factor is added to adjust the strength of self-feedback of sound field modulator, then MIDI music signal output after modulation is as follows:

$$F'(t) = A\sin\left[\omega t + \beta F(t)\right] (12)$$

Where, β is the feedback factor. Through the feedback FM modulation method generated with sawtooth wave type harmonic spectrum, can enhance the MIDI music expression. So far, the sound field modulation method of MIDI music production based on FPGA is studied.

2 Experimental studies

2.1 Experimental content

In this section, the modulation effect of the proposed FPGA-based sound field modulation method for MIDI music production above will be tested by comparison experiments. The conventional sound field modulation method is chosen as the comparison group, and the original music fragments are processed based on the MIDI production software. And the produced MIDI music was processed according to the requirements of the two sound field modulation methods, respectively. The modulation effect of the two methods was evaluated by comparing the difference between the measured sound pressure level and the simulated sound pressure level of the MIDI music processed by the two methods.

2.2 Experimental preparation

The parameters of the music selections used for the experiments are shown in table 3 below.

Table 3 Parameters of the original music selection for the experiment

Music selection number	Length of selected segment (s)	Music genre	Number of notes
1	48	World music	56
2	67	New age	89
3	52	Bossa nova	48
4	69	Acappella	62
5	87	Classical	80
6	49	Post-rock	37
7	61	World music	59
8	75	New age	65
9	80	Classical	77

10	50	World music	45

Play the modulated MIDI music using the same equipment and measure the sound field sound pressure of the modulated MIDI music using a sound pressure meter at the same location.

2.3 Experimental results

Table 4 below shows the measured and simulated sound pressures after modulation

Table 4 Comparison of measured sound pressure and simulated sound pressure

Sound pressure meter position number	Methodology of this paper		Traditional methods	
	Measured sound pressure/dB	Simulated sound pressure/dB	Measured sound pressure/dB	Simulated sound pressure/dB
1	0	0	0	0
2	-4.26	-4.18	-2.87	-3
3	-20.76	-20.11	-10.98	-10.21
4	-0.03	0	-0.04	0
5	-4.15	-4	-3.14	-2.94
6	-15.21	-14.75	-10.17	9.49
7	-0.02	0	-0.02	0
8	-4.37	-4	-3.05	-2.94
9	-13.68	-14.75	-9.43	-9.49

Analysis of the above table 4 shows that after the modulation of the method in this paper, the measured sound pressure level of music is consistent with the simulated sound pressure level, the sound frequency reaches above 2.5 Mhz, and the sound directivity is stronger. The music modulated by the traditional method fails to obtain a stronger directivity, but instead gets a more uniform sound field, with an overall higher sound pressure and only amplification effect. Therefore, the FPGA-based sound field modulation method for MIDI music production proposed in this paper can increase the directivity of MIDI music and enhance the music sound field effect.

3 Conclusion

The small size of MIDI files makes it ideal for distribution over the Internet. With the rapid development of wireless communication technology, especially the use of multimedia on wireless communication devices, such as mobile phones, in turn will further facilitate the development of MIDI music synthesis technology. Because a MIDI file is a description of a piece of music and does not itself include playback-ready sound information, computer music needs to be synthesized through various forms of synthesis techniques if it is to be played back by a sound card. To improve the purpose of music, this paper proposes an FPGA-based sound field modulation method for MIDI music production. Through comparison experiments with traditional methods, it is verified that the modulation by the method in this paper results in more directional MIDI and better modulation.

Reference

[1] Järpe Eric, Weckstén Mattias. (2020). Velody 2—Resilient High-Capacity MIDI Steganography for Organ and Harpsichord Music. Applied Sciences, 11(1): 39-39.

[2] Abboud, R. , Tekli, J. . (2020). Integration of nonparametric fuzzy classification with an evolutionary-developmental framework to perform music sentiment-based analysis and composition. Soft Computing, 24(13), 9875-9925.
[3] Hubert Eichner , Alexander Borst. (2017). Hands-on parameter search for neural simulations by a MIDI-controller.. PLoS ONE, 6(10):e27013
[4] Craig Anderton. (2016). The MIDI Comeback Continues. Pro Sound News, 38(5): 25,57.
[5] Avila, C. . (2018). Midi technology for live performances in quito. Journal of the Audio Engineering Society, 66(12): 1131-1131.
[6] Uzhansky, E. , Katsnelson, B. , Lunkov, A. , Ostrovsky, I. . (2019). Variability of the sound field in the presence of internal kelvin waves in a stratified lake: the sea of galilee as a case study. The Journal of the Acoustical Society of America, 145(3):1671-1671.
[7] Hu, X. , Liu, X. , Wan, X. , Y Shan, Yi, J. . (2020). Experimental analysis of sound field in the tire cavity arising from the acoustic cavity resonance. Applied Acoustics, 161(1):107172.
[8] Brown, L. , Mahomed-Asmail, F. , Sousa, K. , Swanepoel, D. W. . (2019). Performance and reliability of a smartphone digits-in-noise test in the sound field. American Journal of Audiology, 28(3S): 736-741.
[9] V. M. Kotov , G. N. Shkerdin. (2018). Pulse Modulation of Multicolor Radiation Via Light Diffraction by Sound. Journal of Communications Technology and Electronics, 63(9): 1005-1008.
[10] Sushma S , Koushika Ravindran Smruthi, Rajendar Nadagoudar Pavan, Augusta Sophy P.. (2020). Implementation of a 32 – bit RISC processor with floating point unit in FPGA platform. Journal of Physics: Conference Series, 1716(1): 012047-.
[11] Lee Joo Hwan , Zhang Hui, Lagrange Veronica, Krishnamoorthy Praveen, Zhao Xiaodong, Ki Yang Seok. (2020). SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. IEEE COMPUTER ARCHITECTURE LETTERS, 19(2):110-113.
[12] Kil Song Won , Kim Dong Ho , Ri Hyo Il. Design and Verification of SDRAM Controller Based on FPGA[J]. Journal of Computer and Communications, 2020, 08(07) : 14-22.
[13] Taenzer Michael ,Mimilakis Stylianos I. , Abeßer Jakob. (2021). Informing Piano Multi-Pitch Estimation with Inferred Local Polyphony Based on Convolutional Neural Networks. Electronics, 10(7):851-851.
[14] Qiu Lvyang , Li Shuyu , Sung Yunsick. (2021). DBTMPE: Deep Bidirectional Transformers-Based Masked Predictive Encoder Approach for Music Genre Classification. Mathematics, 9(5): 530-530.
[15] Antonio Polo , Xavier Sevillano. (2019). Musical Vision: an interactive bio-inspired sonification tool to convert images into music. Journal on Multimodal User Interfaces, 13(3):231-243.
[16] Dong Hyun Lee ,James W. Beauchamp. (2019). Automatic transcription of solo audio into music notation. The Journal of the Acoustical Society of America, 145(3): 1709.
[17] Peiwei Zhang , Xin Sui. (2017). Application of Digital Music Technology in Music Pedagogy. International Journal of Emerging Technologies in Learning (iJET), 12(12): 4-13.

[18] Kim Hojun , Shang Yulong, Kim Seunghyeon, Jung Taejin. (2021). Improved Pair-Wise Detections of Differential Quasi-Orthogonal Space-Time Modulation with Four Transmit Antennas. Electronics, 10(14) :1675-1675.
[19] Liokumovich Leonid , Muravyov Konstantin, Sochava Aleksandr, Skliarov Philipp, Ushakov Nikolai. (2021). Signal detection algorithms for interferometric sensors with harmonic phase modulation: evaluation of additive noise effects.. Applied optics, 60(20): 5959-5966.
[20] Andrei Popleteev. (2019). Improving ambient FM indoor localization using multipath-induced amplitude modulation effect: A year-long experiment. Pervasive and Mobile Computing, 58(1):101022.