Audio Denoising Coprocessor Based on RISC-V Custom Instruction Set Extension

JUN YUAN[1], QIANG ZHAO[1], WEI WANG[1], XIANGSHENG MENG[1], JUN LI[1], QIN LI[2]

[1]School of Optoelectronic Engineering, Chongqing University of Posts and Telecommunications, Chong Qing 400065, CHINA

[2]Chongqing Marketing Department of Southwest Oil & Gas Field Company, Chong Qing 401120, CHINA

Abstract—As a typical active noise control algorithm, FxLMS is widely used in the field of audio denoising. In this paper, an audio denoising coprocessor based on RISC-V custom instruction set extension was designed, and the idea of software and hardware co-design was adopted; based on the traditional pure-hardware implementation, the accelerator optimization design was carried out, and the accelerator was connected to RISC-V core in the form of coprocessor. Meanwhile, the corresponding custom instructions were designed, the compiling environment was established, and the library function of coprocessor acceleration instructions was established by embedded inline assembly. Finally, the ANC system was built and tested based on E203-SoC, and the test data was collected by audio analyzer. The results showed that the audio denoising algorithm could be realized by combining heterogeneous SoC with hardware accelerator, and the denoising effect was about 8dB. The number of instructions consumed by testing custom instructions for specific operations was reduced by about 60%, and the operation acceleration effect was significant.

Keywords—RISC-V, Custom instruction, ANC, Coprocessor.

Received: August 29, 2021. Revised: April 12, 2022. Accepted: May 9, 2022. Published: June 25, 2022.

1. Introduction

With the rapid development of economy, noise problems such as industrial noise and automobile noise have become increasingly prominent. The traditional passive noise control technology[1] is effective for medium and high frequency noise through passive control methods such as sound absorption and sound insulation. However, active noise control[2], namely the method of reducing noise signal through the principle of destructive interference of sound waves, is more effective for low-frequency narrow-band noise. Adaptive denoising system is generated, that is, while generating anti-noise signals, active repair of anti-noise signals is carried out according to the change of noise to complete active noise control.

FxLMS algorithm is widely used in active noise control system due to its simple circuit structure, simple implementation and small computation. Researchers through FPGA[3], DSP[4], MCU[5], ASIC[6] and other design methods for the hardware implementation of the algorithm. Reference [3] puts forward a hardware implementation of FxLMS algorithm based on FPGA, which divides the operation part of the algorithm into filtering part and update part, in which the filtering part is FIR filter, namely the process of one-dimensional convolution; the update part is the weight update part of LMS algorithm block, that is, the process of multiply accumulate (MAC). Reference [5] proposes an implementation of FxLMS algorithm using STM32F407 microprocessor of Cortex-M4, and proposes a fixed step size method to reduce the computation and solve the problem of floating-point operation.

An audio denoising coprocessor based on RISC-V custom instruction set extension is designed in this paper. According to the hardware implementation of traditional FxLMS algorithm, the software and hardware co-design of FxLMS algorithm is carried out, the work of filling and moving the data to be processed is handed over to MCU for processing. Meanwhile, the convolution and MAC operations with large computation are designed as hardware accelerators, and the coprocessor is designed in the way of instruction pipeline. Finally, the hardware acceleration is completed by coprocessor, and the heterogeneous SOC is combined with the hardware accelerator.

2. RISC-V and Hbird E203 Core

RISC-V instruction set has been widely welcomed all over the world since it was published in 2014. RISC-V instruction set design is simplified and efficient. At present, the setting of RISC-V modular instruction set makes RISC-V architecture have more choices, so that it can try to meet various applications through a u nified architecture, which is an advantage that X86 and ARM instruction set architecture do not possess. Extensibility of instructions is a prominent feature of RISC-V architecture. Users can customize instructions according to the reserved instruction coding space. So that the coprocessor has better portability.

In order to realize the audio denoising coprocessor based on RISC-V instruction set, it is necessary to select the appropriate RISC-V processor core as the carrier. Among many open source RISC-V cores, such as Rocket[7], BOOM[8], RI5CY[9] and others, Hbird E203 core adopts two-stage pipeline design and supports RV32I/E/A/M/C instruction subset configuration, and its supporting SoC provides a large number of IP modules, including UART, IIC, SPI, etc[10]. Benchmark ARM Cortex-M0+ in terms of performance, and its microarchitecture is shown in Figure 1.

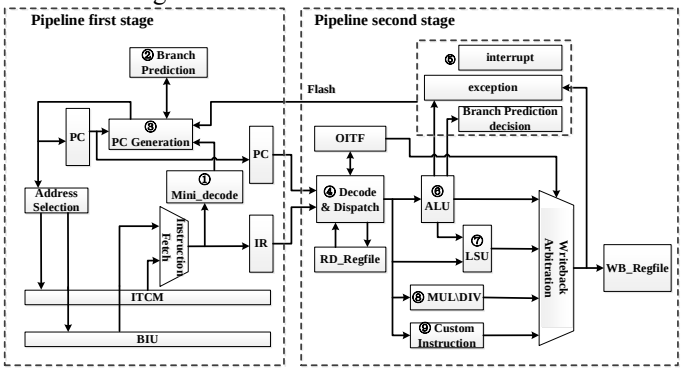


Figure 1 Schematic diagram of E203 microarchitecture

E203 core adopts two-stage pipeline structure, the first stage of which is value taking, and the second stage of which is instruction decode (ID), execute (EX), writeback (WB) and memory (MEM).

The first stage pipeline includes simple ID function block, branch predictor and PC generator. The simple ID function block (Tag 1 in the figure) partially decodes the obtained instructions to obtain some instruction information, including the classification of instructions, whether they are ordinary instructions or branch jump instructions, and the types and details of branch jump instructions. For branch jump instruction, it is necessary to use static branch predictor (Tag 2 in the figure) to predict the jump and get the predicted jump address of the instruction. The PC generator (tag 3 in the figure) generates the PC value of the next instruction to be fetched, generates PC according to different types such as fetching after reset, sequential fetching, branch instruction fetching and pipeline flushing fetching, and accesses instruction tightly coupled memory (ITCM) or bus interface unit (BIU) to fetch fingers through ICB bus. The PC value and the corresponding instruction value are stored in the PC register and the IR register.

The secondary pipeline mainly includes ID and dispatch (tag 4 in the figure), arithmetic logic operation unit (tag 6 in the figure), memory access unit (tag 7 in the figure), long instruction (tag 8 in the figure), custom instruction (tag 9 in the figure), delivery and pipeline flushing (tag 5 in the figure). ID and dispatching realize ID of instructions and dispatching related information to arithmetic logic operation unit, and ALU unit dispatches specific information to different execution units for execution. One-cycle instructions such as logic operation, addition and subtraction, shift, etc. are handed over to ordinary ALU unit for processing. The branch jump instruction is delivered to judge the prediction, and the prediction error needs to be flushed by the instruction pipeline. The memory access instruction is allocated to the memory loading unit for loading and accessing data. Long-term coprocessor instructions will be assigned to coprocessor units for execution.

3. Fxlms Algorithm and Its Hardware and Software Co-design Conception

The schematic diagram of active noise control system architecture is shown in Figure 2, and the operation processing part is the most classical FxLMS algorithm[11-13].

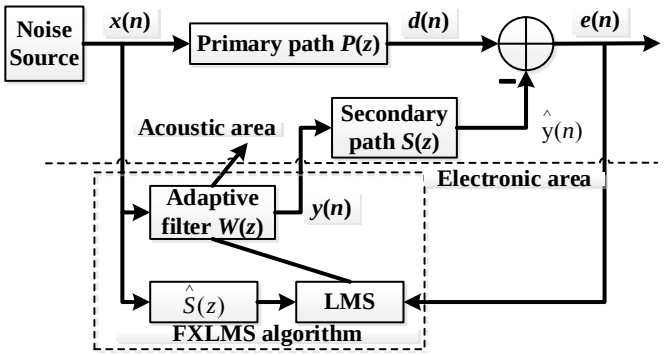


Figure 2 Schematic diagram of ANC system structure

The implementation of FxLMS algorithm has two different acoustic paths. The main signal is sampled with a r eference microphone, then the speaker emits an anti-noise signal, and the error sensor measures the residual error signal. In this process, the acoustic path between the reference noise source and the error sensor is called the primary path, and the electrical to acoustic path between the speaker and the microphone is called the secondary path. FxLMS algorithm contains two parts, one is the least mean square algorithm, and the other is adaptive filtering.

3.1 LMS Algorithm Principle

The least mean square (LMS) algorithm is based on the minimum mean square error criterion and the gradient method. By improving the calculation method of the gradient value of the mean square error, the algorithm can be shown by recursive formulas such as Equations (1), (2) and (3)[14-16]:

$$y(n) = \boldsymbol{W}^H(n)\boldsymbol{X}(n) \tag{1}$$

$$e(n) = d(n) - y(n) \tag{2}$$

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n) + 2\mu\boldsymbol{X}(n)e^*(n) \tag{3}$$

Where, $\boldsymbol{W}(n)$ represents the weight vector of the filter; $\boldsymbol{X}(n)$ represents a set of vectors composed of input signals; $y(n)$ represents the output signal; $d(n)$ represents the desired signal; $e(n)$ represents the error signal; μ represents the step size factor, where the larger μ is, the faster the convergence speed of the algorithm is, and vice versa. However, the faster the convergence speed, the worse the steady-state performance, so it is necessary to constrain the step size factor. In this design, considering the reduction of algorithm complexity and processing flexibility, the selection of step factor is based on the

fixed step proposed in [3].

3.2 Adaptive Filtering

The adaptive filtering part is a FIR filter, and the formula is shown in Equation (4):

$$y(n) = \boldsymbol{W}^T(n)\boldsymbol{X}(n) \tag{4}$$

Where, $y(n)$ represents $\boldsymbol{X}(n)$ generated by a FIR filter with a weight coefficient $\boldsymbol{W}(n)$, because every time a stage sound source $y(n)$ is generated, the weight coefficient $\boldsymbol{W}(n)$ is updated by LMS operation. Therefore, updated time-varying coefficients are obtained, i.e. the coefficients are automatically and continuously adapted to a given signal to obtain a desired response to complete adaptive filtering.

3.2 Software and Hardware Co-design

The coprocessor part is connected with the main processor in the mode of instruction pipeline through NICE circuit interface, and the hardware acceleration function is mobilized in the mode of custom instructions in the software flow, which is shown in Figure 3.

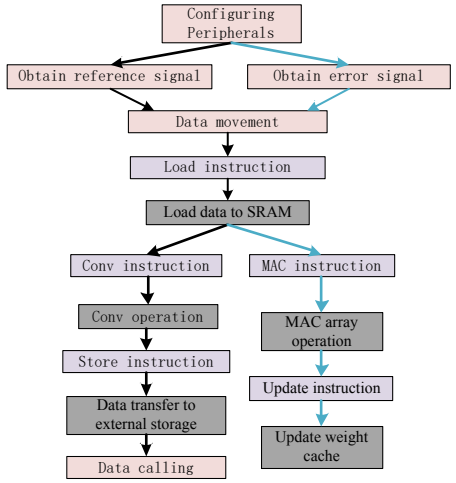


Figure 3 Flow chart of software and hardware co-design

The gray and pink parts in Figure 3 belong to the software flow, of which specific significance is the external acquisition of ANC system and the configuration of function blocks. Then it includes storing the data at the corresponding address after collection. Subsequently, the light gray part is the related custom instructions. Used for realizing software and hardware interaction between the main processor and the coprocessor, the last dark gray part is the defined hardware acceleration part, which specifically includes the adaptive filtering part in the algorithm corresponding to convolution operation, the weight update part corresponding to LMS algorithm in the algorithm corresponding to multiplication and accumulation array operation, and the corresponding cache unit used in data handling.

At the same time, due to the particularity of serial operation of the algorithm itself, this process has two steps. First, the black flow line is the adaptive filtering operation in the main path, and then the electrical to acoustic transformation is needed through relevant peripherals to generate secondary sound sources. The second step is the acquisition of error signals and the updating operation of weight coefficients, which will have a sequence relationship. Therefore, this design process includes two paths, and only when both paths run out can ANC system denoising be completed once.

4. Hardware Design Part

The audio denoising accelerator designed in this paper optimizes the updating weight and filtering module in the traditional design. The parallel one-dimensional convolution structure in the form of addition tree is used to replace the serial MAC arithmetic unit to realize the filtering part, and the parallel MAC array is used to replace the original updating weight part, and the related modules of coprocessor are added.

4.1 Optimization of Operation Structure Design

In the traditional hardware implementation of FxLMS algorithm, the filter module adopts MAC arithmetic unit, namely multiply accumulate arithmetic unit and realizes filtering in serial mode, which will reduce the arithmetic performance of the filter module, and a lot of repeated operations are needed when the filter order is long. Therefore, this design adopts the strategy of sacrificing area in exchange for performance improvement, and uses the addition tree structure, which will greatly improve the parallel operation ability and realize one-dimensional convolution operation. At the same time, this design adopts MAC array parallel operation to update the coefficients of weight matrix. Finally, this design adopts the idea of data multiplexing, and uses data distributor to reduce the resource consumption of weight coefficient storage SRAM. The circuit structure is shown in Figure 4..

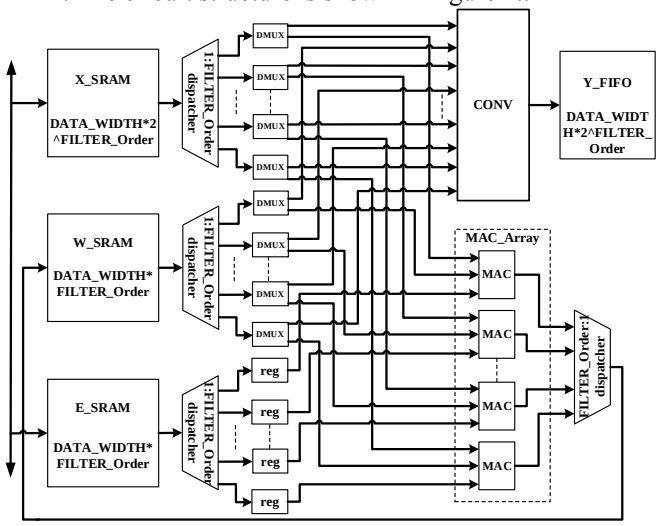


Figure 4 Structure diagram of hardware accelerator

The whole acceleration circuit comprises a reference signal data buffer module (X_RAM), a weight coefficient data buffer module (W_RAM), an error signal data buffer module (E_RAM), a data distributor, a data distributor, a one-dimensional convolution operation block, a MAC array operation block and a data integrator.

The most critical parts in the accelerator circuit are one-dimensional convolution operation block and MAC array operation block. The one-dimensional convolution operation block realizes FIR filtering part of the algorithm, and the MAC array operation block realizes parallel weight coefficient update.

Its circuit structure is shown in Figure 5.

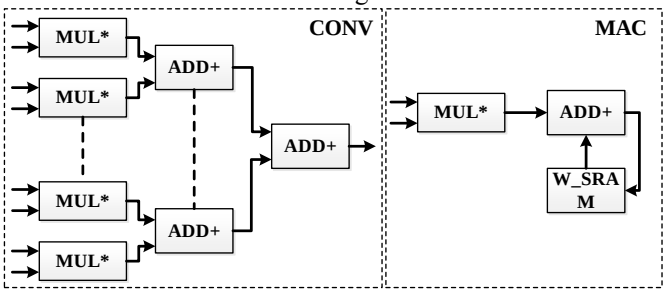


Figure 5 Circuit structure diagram of arithmetic unit

Because the audio denoising algorithm needs to quickly generate secondary sound sources after collecting reference signals, and the generation of secondary sound sources needs to be operated by adaptive filtering, so in this design, the addition tree parallel structure is used to design one-dimensional convolution operation for filtering, which can improve the operation speed and high parallelism, so that the secondary sound sources can be produced faster. When the secondary sound source is generated, it is necessary to collect error signals to update the filter weight coefficients, so the algorithm has the characteristics of sequential processing, and the speed of weight updating will play an important role in the generation of secondary sound sources. Therefore, in this design, MAC array is used to realize the updating operation of each weight, and the updated weight data needed by the next convolution operation can be obtained in the same period. The reasonable use of data distributor and data integrator makes the operation speed greatly improved.

4.2 Coprocessor Design

After the operation structure design is completed, the core instruction cooperation unit should be added to e xpand the coprocessor design, and the decoder, data extractor and configuration enabling function block should be added to complete the hardware design of the audio denoising coprocessor. Its circuit structure is shown in Figure 6.

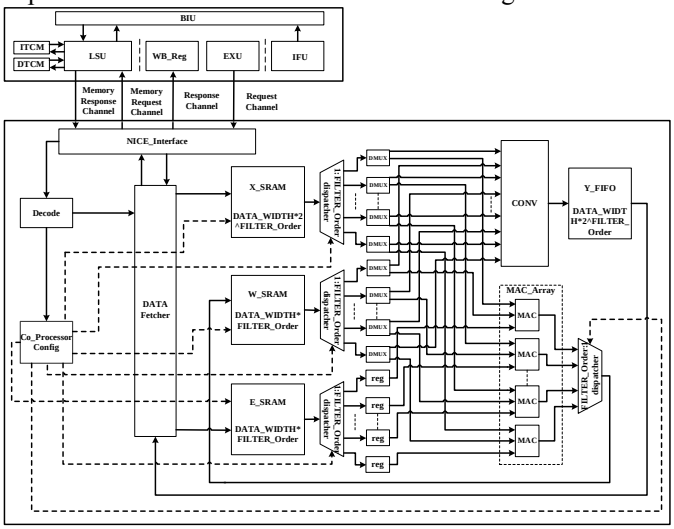


Figure 6 Circuit structure diagram of audio denoising coprocessor

The NICE controller processes the time sequence related to the interface of the coprocessor, and transmits the instruction information and source operands obtained from the request channel to the decoder for ID. Decoder is used to decode custom instructions. This design is mainly divided into two types of instructions, one is configuration instructions, and the other is data loading and storage instructions. For the configuration instruction, the configuration information is transmitted to the configuration module, and the configuration module will give the enabling signal and control signal required by the corresponding response module to realize the configuration of each operation function part. For the data load store instruction, the memory access information is transmitted to the data extractor for processing. When the memory access information is a loading instruction, the address information and the read signal are transmitted through the memory request channel and the data is obtained from the corresponding memory module. Then the read data is transmitted to the data extractor through the memory feedback channel and distributed to the corresponding cache module by the data extractor. When the memory access information is a write memory instruction, the address information is transmitted through the memory request channel, and the write data is obtained from the data extractor and transmitted to the memory location corresponding to the address. If the instruction is the write-back result, the write-back data is transmitted to the feedback channel through the data extractor to complete the write-back of the general register.

After the software program is burned to the MCU, the main processor obtains the instructions in sequence, decodes the instructions, and judges whether the instructions are custom instructions according to their operation codes. In this design, the operation codes of custom1-4 defined by RISC-V are used as custom instruction operation codes, and R-type instructions are used for custom instruction coding. Its format is shown in Figure 7.

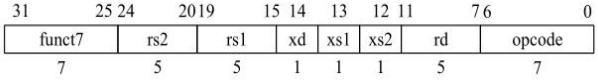


Figure 7 32-bit custom instruction encoding format

For custom instructions, it is judged whether to read the source operand according to xs1 and xs2. In this process, the main processor maintains the data correlation, and if there is a data conflict, the data channel will be closed until the data correlation is released. If there is data written back, the destination register of the rd bit is also a consideration of data correlation. After that, the instruction information is transmitted to the coprocessor for processing through the NICE interface, The coprocessor decodes the instructions and distributes them to different units for execution according to the type of instructions. Finally, the coprocessor writes the instruction execution results back to the main processor through the response channel, and writes the execution results back to the rd target register or transmits the results to the corresponding storage locations through the memory request channel.

The data stream of the audio noise reduction coprocessor designed in this subject is shown in Figure 8, and the processing signals are obtained by external sensors or receivers. Data is transmitted to ICB peripheral bus through interface IP mounted

on SoC. When the relevant data needs to be processed, the data is acquired and written through the memory request and feedback channel. After the processing is finished, the processing result is written back to the general register of the main processor or into the corresponding memory, and the main processor sends it to the external module through the interface IP to obtain the generated signal.

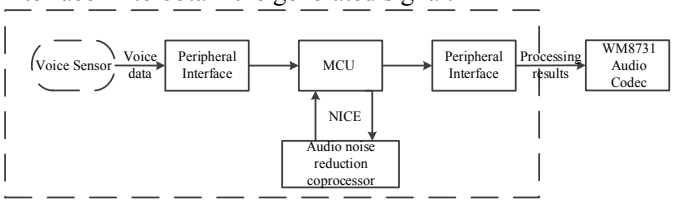


Figure 8 System data flow diagram

Through memory access custom instructions, a lot of data stored in the main processor is moved to the coprocessor, which reduces the access of the coprocessor to the main processor memory and greatly reduces the power consumption. At the same time, the parallel operation units in the coprocessor will ensure the operation speed. Finally, compared with the SoC plug-in accelerator, the coprocessor with instruction pipeline mode does not need frequent data access, reduces data movement, and has better real-time processing performance.

5. Software Design Part

5.1 Custom Instruction Design

The custom instructions of the audio denoising coprocessor based on FxLMS algorithm are shown in Table 1.

Table 1 Custom instruction table of coprocessor

Instruction	Funct7	Rd	Xd	Rs1	Xs1	Rs2	Xs2
Load.X	1	-	0	X_MemoryAddress	1	Length|X_BaseAddress	1
Load.E	2	-	0	E_MemoryAddress	1	Length|E_BaseAddress	1
Store.Y	3	-	0	Y_ MemoryAddress	1	-	0
Cfg.Conv	4	-	0	Filter order	1	En_Conv	1
Cfg.MAC	5	-	0	Filter order	1	En_MAC	1
Updata.W	6	-	0	Filter order	1	En_Up.W	1
Rst	7	-	0	-	0	-	0

There are 7 custom instructions, namely data load storage instruction and configuration enable instruction. The data loading instruction is responsible for loading the reference signal and the error signal from the corresponding address and storing them in the corresponding buffer of the coprocessor. The data storage instruction is responsible for transmitting the secondary sound source signal and writing it to the corresponding memory address through the memory request channel. The configuration enable instruction is responsible for configuring the filter order and enabling the relevant functional modules.

The use steps of custom instruction are as follows: firstly, the reference signal is loaded through Load.X instruction, and the data is accessed through memory request channel and read through memory feedback channel, and then loaded into X_SRAM cache. After that, the reference signal and weight coefficient are read from X_SRAM and W_SRAM by Cfg.Conv instruction and sent to the corresponding DMUX through data distributor. DMUX performs convolution operation and generates secondary sound source under the control of enable signal until the convolution of reference signal ends. After that, the secondary sound source data in FIFO is written back to the corresponding address through the memory request channel through the Store. Y instruction. Then Load.E instruction loads error signal data like Load.X instruction, and Cfg.MAC instruction configures MAC operation array and updates weight coefficients. Finally, W_SRAM is configured by Updata.W instruction to write the update weight, which completes an adaptive denoising operation acceleration. In addition, the reset of the coprocessor can be performed by Rst instruction.

5.2 Coprocessor Library Function Design

After completing the instruction of custom coprocessor, we can use assembly language to transfer the work of coprocessor. However, the efficiency of assembly language development is too low, so embedded inline assembly is often used in C\ C++. Therefore, the first task is to package instructions into C language library functions by using inline assembly syntax format, and complete the library function design of coprocessor. The designed library function interface is shown in Table 2.

Table 2 Library functions of custom instructions and their introduction

Function Interface	Function
int Load_X(unsigned int X_MemoryAddress, unsigned int X_BaseAddress, unsigned int Length)	Load reference signal X into X.SRAM
int Load_E(unsigned int E_MemoryAddress, unsigned int E_BaseAddress, unsigned int Length)	Load error signal E into E.SRAM
int Store_Y(unsigned int Y_ MemoryAddress)	Store secondary source Y
int Cfg_Conv(int Filter_order, int En_Conv)	Configure convolution operation length and enable
int Cfg_MAC(int Filter_order, int En_MAC)	Configure MAC operation length and enable
int Updata_W(int Filter_order, int En_UP.W)	Configure weight coefficient length and enable
void Rst()	Reset coprocessor

Cfg.Conv library function is taken as an example, and its specific inline assembly syntax format is shown in Figure 9.

```
//   Cfg.Conv
 int Cfg_Conv(int Filter_order, int En_Conv)
{
    int source1=Filter_order;
    int source2=En_Conv;
    asm volatile(
        "Cfg.Conv %[src1],%[src2]"
        :[src1]"r"(source1), [src2]"r"(source2)
    );

  return 1;
}
```

Figure 9 Library function of Cfg.Conv instruction

6. Application and Evaluation of Fxlms Algorithm

6.1 Overall Design of Anc System

After completing the hardware and software design of the coprocessor, it is the design part of the whole ANC system. This design is based on Hbird E203_SoC platform, modifies the original SoC, deletes unnecessary peripheral interfaces, and adds IIS interface peripherals needed for audio data transmission. The whole system structure is shown in Figure 10.

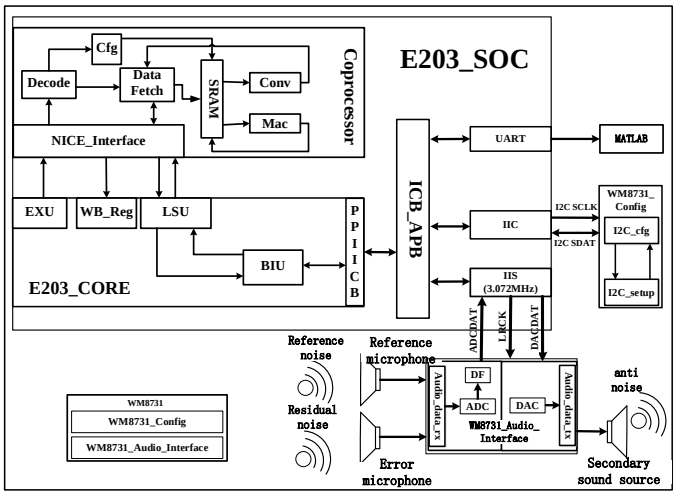


Figure 10 Circuit structure diagram of ANC system

The main processor configures the initial information through IIC bus to make WM8731 audio codec module work normally, and uses the probe to collect audio signals and convert them into digital audio signals through ADC built in the module. Then it is transmitted to ICB bus through IIS audio transmission interface, and the coprocessor reads IIS audio data on the bus through LSU and loads it on the corresponding cache. After configuring enabling instructions, the convolution operation can be carried out smoothly and anti-noise signals can be generated. After that, the anti-noise signal is written to the address where the IIS interface data is located through the memory request channel, and the analog signal is obtained by digital-to-analog conversion through the built-in DAC of the module and secondary noise is generated. Then the module collects the residual noise signal again until it is loaded on the corresponding buffer. After configuring the enabling instruction, the MAC array can update the weight coefficients, so as to complete the denoising and acceleration of ANC system once.

6.2 Evaluation Analysis

After the whole software and hardware design and system design are completed, the denoising performance is measured based on MCU200T development board, and the schematic diagram of the measured scene is shown in Figure 11.

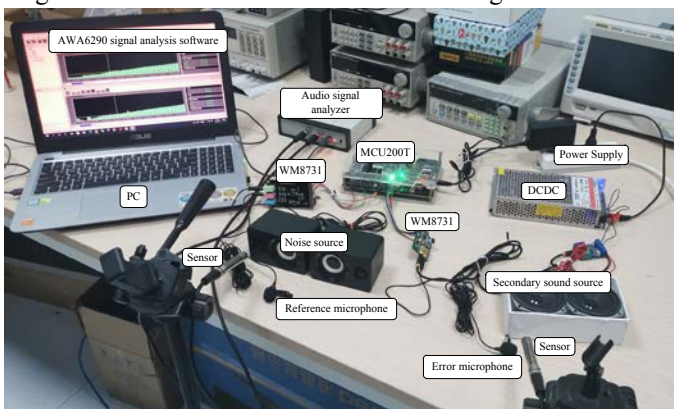


Figure 11 Real scene diagram of test scene

The noise signal is collected before and after denoising by special instruments, and the pink noise is used for acoustic test, so as to obtain the relevant collected data and visualize the data through Matlab to obtain the change schematic diagram before and after denoising in the ear frequency band as shown in Figure 12.

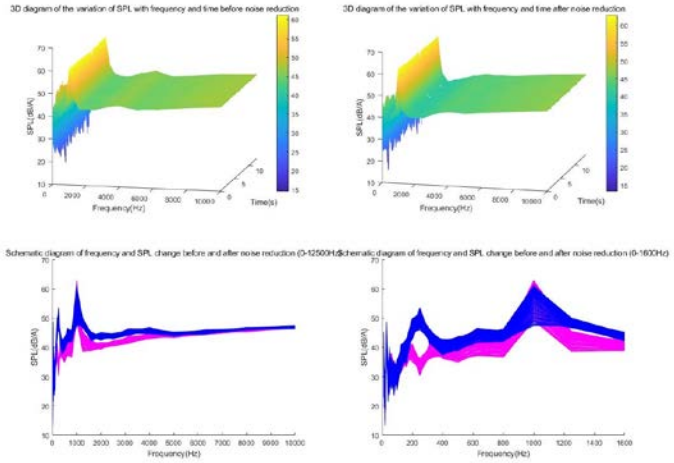


Figure 12 Schematic diagram of noise acquisition at the same position before and after denoising

As can be seen from the schematic diagrams A and B in Figure 12, the sound pressure levels of the same noise are different in continuous time periods, so the denoising effect is unstable. It can achieve good denoising effect in some individual positions but cannot adapt to the whole low frequency band. At the same time, it can be known from the schematic diagram C in Figure 12 that the denoising effect of the algorithm in the middle and high frequency band is not ideal, which is related to the denoising principle of the active denoising system itself. From the analysis of schematic diagram D in Figure 12, it can be seen that the average denoising effect of 8dB can be achieved in the frequency range of 200-2000Hz under the condition of pink noise, which proves that the algorithm can be realized by combining heterogeneous SOC with hardware accelerator.

In order to evaluate the performance of the coprocessor, this paper adopts two methods to implement convolution and MAC operations, one is implemented by the standard RISC-V I\ M instruction set, the other is implemented by using the coprocessor custom instructions designed in this paper and the RISC-V I\ M instruction set together, and compares the number of instructions executed by the two methods. Through IDE tools to write the software code and burn it to the development board, you can print out the corresponding execution results and calculate the number of instructions through serial port. The experimental results are shown in Table 3.

Table 3 Number of instructions required by different arithmetic units to run under different instruction sets

Algorithm	Rv32 I\M Instruction	Coprocessor Instruction
Conv	4582	1324
MAC	656	256

Through the instruction number, we can see that Conv and MAC operation can save instruction space more than standard instruction set under the action of coprocessor, and the instruction number is greatly reduced. This is because on the one hand, the coprocessor realizes convolution and MAC through a special hardware acceleration unit, while the main processor can only realize convolution and MAC through

software methods such as addition, subtraction, multiplication and division; on the other hand, from the system data flow diagram in Figure 7, it can be seen that the coprocessor implementation reduces the repeated movement of data and further improves the processing speed of the algorithm.

7. Conclusion

Based on the design optimization of hardware accelerator, The coprocessor is designed, the ANC system is built on the basis of E203_SoC, and the denoising test is carried out in a quiet indoor environment. The sound pressure level data before and after denoising are obtained by audio analysis and acquisition instrument. After data analysis, it can be seen that the FxLMS algorithm realized by combining heterogeneous SoC with hardware accelerator has remarkable effect and can achieve nearly 8dB denoising effect. Subsequently, two different test methods are used to test the acceleration effect of coprocessor, and it is concluded that the implementation of coprocessor custom instruction set has significant acceleration effect for convolution and MAC operations.

Acknowledgment

This research was supported by the Science and Technology Major Project of Chongqing Municipal Science and Technology Bureau (cstc2018jszx-cyztzxX0054), and the Chongqing Municipal Science and Technology Commission Major Project of Integrated Circuit Industry (cstc2018jszx-cyztzx0217)

References

[1] Meng H, Chen S. Particle swarm optimization based novel adaptive step-size FxLMS algorithm with reference signal smoothing processor for feedforward active noise control systems[J]. Applied Acoustics, 2021, 174: 107796.

[2] Sookpuwong C, Chompoo-inwai C. A Multi-Channel Feedforward ANC with FXLMS Algorithm for Aviation-Noise Suppression[C]//2019 53rd Asilomar Conference on Signals, Systems, and Computers. IEEE, 2019: 1374-1378.

[3] Abdi F, Amiri P. Design and implementation of adaptive FxLMS on FPGA for online active noise cancellation[J]. Journal of the Chinese Institute of Engineers, 2018, 41(2): 132-140.

[4] Liu L, Su Q, Li W, et al. Real Time Implementation and Experiments of Multi-channel Active Noise Control System for ICU[C]//2021 IEEE International Conference on Electro Information Technology (EIT). IEEE, 2021: 395-400.

[5] Shyu K K, Ho C Y, Chang C Y. A study on us ing microcontroller to design active noise control systems[C]//2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). IEEE, 2014: 443-446.

[6] Vu H S, Chen K H, Sun S F, et al. A 6.42 mW low-power feed-forward FxLMS ANC VLSI design for in-ear headphones[C]//2015 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2015: 2585-2588.

[7] Asanovic K, Avizienis R, Bachrach J, et al. The rocket chip generator[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, 4.

[8] Asanovic K, Patterson D A, Celio C. The berkeley out-of-order machine (boom): An i ndustry-competitive, synthesizable, parameterized risc-v processor[R]. University of California at Berkeley Berkeley United States, 2015.

[9] Traber A, Zaruba F, Stucki S, et al. PULPino: A small single-core RISC-V SoC[C]//3rd RISCV Workshop. 2016.

[10] Wu N, Jiang T, Zhang L, et al. A reconfigurable convolutional neural network-accelerated coprocessor based on RISC-V instruction set[J]. Electronics, 2020, 9(6): 1005.

[11] Félix F B, de Castro Magalhães M, de Souza Papini G. An improved Anc algorithm for the attenuation of industrial fan noise[J]. Journal of Vibration Engineering & Technologies, 2021, 9(2): 279-289.

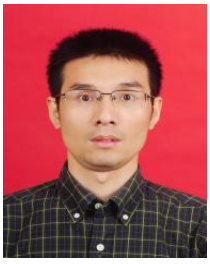
[12] Munir M W, Abdulla W H. On FxLMS scheme for active noise control at remote location[J]. IEEE Access, 2020, 8: 214071-214086.

[13] Kang M S. FxLMS Algorithm for Active Vibration Control of Structure By Using Inertial Damper with Displacement Constraint[J]. Journal of the Korea Institute of Military Science and Technology, 2021, 24(5): 545-557.

[14] Rabiman R, Nurtanto M, Kholifah N. Design and Development E-Learning System by Learning Management System (LMS) in Vocational Education[J]. Online Submission, 2020, 9(1): 1059-1063.

[15] Yang F, Guo J, Yang J. Stochastic analysis of the filtered-x LMS algorithm for active noise control[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2020, 28: 2252-2266.

[16] Jalal B, Yang X, Liu Q, et al. Fast and robust variable-step-size LMS algorithm for adaptive beamforming[J]. IEEE Antennas and Wireless Propagation Letters, 2020, 19(7): 1206-1210.



Jun Yuan, received B.E. and M.E. degrees in Electrical Engineering in 2006, 2009 respectively, from Southwest Jiaotong University, China. And then in 2012 he received D.Eng. degree from Kochi University of Technology, Japan. Then he joined School of Optoelectronic Engineering, Chongqing University of P osts and Telecommunications, China. His areas of research interests are analog-digital mixed signal IC design, DFT research and noise processing IC design.