University College of Southeast Norway
Faculty of Technology and Maritime Sciences
-
Master Thesis in Systems Engineering with Embedded Systems
Kongsberg Department of Engineering
June 7, 2017

Zhili SHAO

Real-time Audio Signal Processing for UAV Based Applications Using the Xilinx Zynq SoC

Acknowledgements

I want to express my great gratitude to my supervisors Dr. António L. L. Ramos and Dr. José A. Apolinário Jr., for their enthusiastic encouragement, patient tutoring and useful criticism throughout my research. They provided me precious chance to work in their research field, and I appreciate their selfless help.

I would also like to extend my thanks to my lovely fellow master students for their cooperation, encouragement, and friendship. Thank my college for the comfortable research environment and abundant resources.

At last, I wish to thank my parents for their support and encouragement throughout my master program.

Abstract

Unmanned aerial vehicles (UAVs) have been widely used in both civilian and military fields nowadays. Early applications mainly focus on imaging area due to the difficulty in collecting clean audio data. However, there are huge demands for airborne audio applications. In this thesis, the possible solutions for airborne audio applications of UAV are investigated. Here, a number of frequently used filters for audio signal processing such as median filter, FIR filter, adaptive filters are addressed. It also reveals how these filters will help UAV audio applications according to their specific features. The hardware and software implementation of a real-time audio signal processing system is designed on ZYBO, which is a development board using Xilinx Zynq-7000 all programmable ARM/FPGA SoC. This design follows the concept of hardware and software co-design. The results show that ZYBO platform is efficient and robust enough to support UAV airborne real-time audio signal processing.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	**A**nalog **D**igital **C**onverter
AMBA	**A**dvanced **M**icrocontroller **B**us **A**rchitecture
Codec	**Co**der-**dec**oder
DAC	**D**igital **A**nalog **C**onverter
DSP	**D**igital **S**ignal **P**rocessing
DOA	**D**irection **O**f **A**rrival
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**esign textbfLanguage
IC	**I**ntegrated **C**hip
GCS	**G**round **C**ontrol **S**tation
IP	**I**ntellectual **P**roperty
IDE	**I**ntegrated **D**evelopment **E**nvironment
LMS	**L**east **M**ean **S**quares
NLMS	**N**ormalised **L**east **M**ean **S**quares
PL	**P**rogramable **L**ogic
PS	**P**rocessing **S**ystem
SDK	**S**oftware **D**evelopment **K**it
SoC	**S**ystem **o**n **C**hip
SNR	**S**ignal-to-**N**oise **R**atio
UAV	**U**nmanned **A**erial **V**ehicle
ZYBO	**ZY**nq **BO**ard

Chapter 1

Introduction

This thesis work aims at exploring audio signal processing on Unmanned Aerial Vehicle (UAV) applications. UAVs have been widely used in both civilian and military fields nowadays. Early applications mainly focus on imaging area due to the difficulty in collecting clean audio data. However, there are huge demands for airborne audio applications. In this thesis, the possible solutions for airborne audio applications of UAV are investigated. Several frequently used filters for audio signal processing such as median filter, FIR filter, adaptive filters are introduced. It also reveals how these filters will help UAV audio applications according to their specific features. The hardware and software implementation of a real-time audio signal processing system is designed on ZYBO, which is a development board using Xilinx Zynq-7000 all programmable ARM/FPGA SoC. This design follows the concept of hardware and software co-design. The results show that ZYBO platform is efficient and robust enough to support UAV airborne real-time audio signal processing.

This chapter will discuss basic concepts about UAV and how its applications affect our lives at first section. Then it provides a brief introduction to the real-time system. Possible audio applications of UAV will be discussed in the following section where corresponding problems are summarized. The last section introduces the structure of this report.

1.1 Unmanned Aerial Vehicle

UAV, also commonly referred to as drone, is an aircraft with no human pilot or passengers. These vehicles rely on many different sensors, i.e., altimeter, GPS receiver, cameras, etc., to control the flight through an onboard computer when a programmed flight plan is available. The vehicle can also be operated remotely by a pilot at the ground control station (GCS). Drones can be powered using jet propulsion, piston engine, or

electric motors. Construction wise, they can be classified as fixed-wing, rotary-wing, or multi-rotor, and Figure 1.1 shows their typical appearances.

FIGURE 1.1: Three types of UAV: fiexed-wing, rotary-wing, multi-rotor.

UAVs can be used in both military and civil applications. Their popularity has grown fast in the last decade, and new applications are still being developed at a steady pace. Besides using UAV directly as the attack weapon, there are many other applications for drones in the battlefield, e.g., military network relay and battlefield circumstance surveillance;or may be even used as a cyber-power in the future. For civilian use, fixed-wing UAVs like aviation models are very popular among amateurs, and they are also used in agriculture and mapping. With the rise of multi-copter, more applications are explored based on its stable, easy control, and high payload, e.g., aerial photography, disaster victims rescue (Wolfe et al., 2015), cellular network reinforce (Afonso et al., 2016), Wi-Fi internet access (Gu et al., 2015), video surveillance (Qazi, Siddiqui, and Wagan, 2015), sensor network data collection (Say et al., 2016), sniper localization systems (Fernandes et al., 2015), etc. These applications show highly demands on signal processing because of the reliability requirement and rugged working environment of UAVs. A former research (Shao, Ramos, and Apolinário Jr., 2016) about UAV applications specialized in data transmission field has been made, in this research, the focus is audio signal processing on UAV applications.

1.2 Real-time Systems

Real-time is one critical characteristic for most embedded systems. It requires the input signal has to be responded within a specified period. The end of this finite period is called deadline and the delay time between input signal creation and response is called latency. A real-time system must be designed carefully to ensure no input signal are missed. Considering the meeting deadline condition, real-time systems can be classified into three patterns: hard real-time system, firm real-time system and soft real-time

system. The hard real-time system requires the deadline must be hit. Only a few systems like nuclear weapon, pacemaker have this requirement. Firm and soft real-time systems allow failure to meet the deadline. For the firm real-time system, the missed response to a request will be meaningless. A good example is forecast system. While in the soft real-time system, the missed response is not worthless, but it will degrade with the time passes, the telephone meeting system is a typical soft real-time system.

UAV flight control system is a real-time system, and it has a critical time requirement for controlling command response. Furthermore, more and more real-time applications based on UAV platform are appearing, since UAVs, primarily multi-rotors become cheap and reliable recently. (Wu and Zhou, 2006) explored the real-time UAV video processing for quick response to natural disaster. (Shi et al., 2016) built a vision-based real-time 3D mapping system on UAV. (Varela et al., 2011) shows a swarm intelligence based approach for real-time UAV team coordination in search operations. For now, we can hardly find research related to real-time audio signal processing applications on UAV. In this study, a real-time audio signal processing hardware platform is built, and some digital audio filters will be tested on this real-time audio signal processing system specifically for UAV applications.

1.3 Audio Signal Processing demands on UAVs

Though UAVs have been used in a variety of applications, there are limited researches on audio related applications. (Santano, 2014) shows a research demo in audio visual data acquisition for first person view (FPV), but there is no special processing for audio data. (Furukawa et al., 2013) depicts an improvement on sound source localization (SSL) system from multi-rotor UAV, but it is not focused on audio signal itself. To bring more possibility to audio signal processing on UAV, this research is conducted. There are three types of digital audio filters will be employed in this paper. According to their characteristics, some possible applications on UAV are illustrated.

The median filter is famous for its using on noise reduction of imaging processing, it can also be used in noise reduction of specific audio signal like old vinyl records. (Borzino et al., 2015) shows an median filter application on enhancing gunshot audio recorded by UAVs for direction of arrival estimation. This thesis will focus on how to implement a real-time gunshot enhancement median filter on UAV platform, and related experiments is also designed to test the results.

Finite impulse response (FIR) filter is a simple and easy to implement signal processing component. FIR filter can be implemented to low-pass, high-pass or band-pass filters in digital audio signal processing. One important application of audio application on

UAV is providing speech communication between help seeker and rescuers. The sound collected by airborne microphone normally contains other noise from the environment. The human auditory system is just sensitive to the audio frequency range from 20Hz to 20KHz, so we can use FIR band-pass filter to decrease noise, or extracting the desired frequency domain without noise from other frequencies. Moreover, knowing the most of the energy in a speech signal is present in lower frequencies, may below 2kHz, so we can use an FIR low-pass filter to narrow interference above the frequency away.

Adaptive FIR filter is another kind of FIR filter that can literately adjust its coefficients to output desired signals. One of the most important reasons that UAV is not popular in audio applications is the noise from UAV working environment including propellers, wind, etc. Propeller vibration is the primary source of noise that affecting sound collection from UAV platform. So removing the propeller noise will significantly improve the quality of audio signals, adaptive noise filter will be addressed in this research .

1.4 Structure of the thesis report

The second chapter introduces methodologies and technologies used by this research. The concept about hardware and software co-design will be introduced at first. Then the relevant technologies about audio signal processing are introduced, including digital signal processing processors, floating and fixed arithmetics, and circular buffer. At last, it interprets communication protocols used by this design.

Chapter three demonstrates detailed information about digital audio signal processing procedure. Median filter, FIR filter, and adaptive filter based on the normalized least mean square (NLMS) algorithm are frequently used on audio signal processing applications. On this chapter, the basic concepts of these filters are depicted. Their possible applications in different fields will be discussed separately.

Chapter four focuses on the implementation of real-time digital signal processing on ZYBO. It is the combination of hardware and software design. The hardware design is implemented on Xilinx Zynq AP SoC. The complete hardware platform contains threes parts: ARM Contex-A9 processer, SSM2603 coder-decoder (Codec) chip, and its controller implemented on FPGA using Verilog Language. Software design section shows program realization of fixed-point filter algorithms.

In chapter five, the experiments of different filters on real-time digital audio platform are conducted, and final results are evaluated.

The last chapter summarizes the whole research. It concludes that the performance of real-time signal processing platform can totally fulfill the requirements of some specific audio processing applications on UAV. The possible future work is also presented on the end.

Chapter 2

Methodologies and Technologies

2.1 Hardware and Software Co-design

In the embedded system field, hardware means single clock synchronous digital circuits created using word-level combinational logic and flip-flops. These circuits can be modeled with building blocks such as registers, adders, and multiplexers. Cycle-based hardware modeling is often called register-transfer-level (RTL) modeling. Software means single-thread sequence programs, written in C or assembly. They will be able to implement the various sections of memory (global, stack, heap), and provide the techniques to control different kinds of memory (registers, caches, RAM, and ROM), Normally we choose C because it matches so well with actual execution model of a typical microprocessor. Figure 2.1 shows a simple hardware design, an RTL circuit; list 2.1 shows a brief software design, a list of C codes. As defined in book (Schaumont, 2010), hardware and software co-design is the partitioning and design of an application regarding fixed and flexible components.

LISTING 2.1: Software example.

```
#include<stdio.h>
int natural_generator
{
    int a = 1;
    static int b = -1;
    b += 1;
    return a+b;
}
```

Table 2.1 depicts the history of hardware and software co-design (Teich, 2012). Hardware software co-design emerged basically as a new discipline to design complex integrated circuits (ICs) in the early 1990s. From that time to today, it has experienced three generations of evolution, from 2012 to now, the fourth generation of hardware software co-design is on the way.

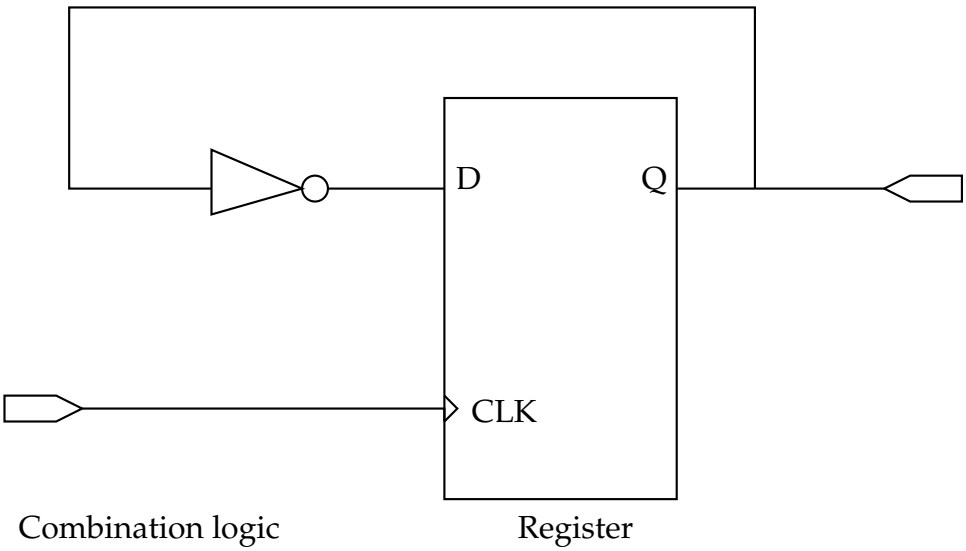

FIGURE 2.1: Hardware example.

TABLE 2.1: Hardware and software co-design history

Generation	Years	Major achievements
1	-1995	hardware software bi-partitioning for CUP-ASIC target
2	1995-2004	Co-simulation & PBD (platform-based design) for complex targets
3	2005-2011	DSE(design space exploration)& Co-synthesis for Het(Heterogeneous). Multi-core Designs
4	2012-	New variants of co-design emerge; Online co-design for adaptive Systems; Co-design of systems of Systems; Co-design for dependability; ...

Hardware can provide high performance, energy efficient solutions, while software can significantly reduce cost, shrink schedule and decrease complexity. A good managed hardware and software co-design will balance their advantages and disadvantages, so the concept of hardware and software co-design is applied on this research. The signal processing algorithms and configuration are conducted by program running on processor, while audio Codec chip controller is implemented on FPGA.

2.2 DSP processors versus ARM-based processing systems

ARM and DSP are two types of microprocessors. A microprocessor is an integrated circuit implemented on a silicon chip with central computing unit (CPU), and it has internal memory, and the whole system operates based on binary logic. The purpose of a microprocessor based signal processing system is getting digital data from the input, processing them and outputting processed data.

DSP stands for digital signal processing and it can be any kinds of processes working on digital data operating. The DSP processor is specially designed for digital signal processing. The main goals of a DSP process is to filter, measure, or compress a digital signal or analog. It normally includes an analog-to-digital converter (ADC) which can transform real-world analog signals to digital signals. Moreover, the processed digital signals may be converted back to an analog signals with a digital-to-analog converter (DAC). DSP processors are widely used in audio signal processing, imaging signal processing, and sensor signal processing, among other fields. There are many manufacturers specializing in DSP processors, including Analog Devices, famous for the SHARC processor, and Texas Instruments, famous for their TMS320 family of products.

DSP processors are specially designed for manipulating a large number of complex mathematic calculations. It is almost two or three times faster than a general purpose microprocessor. This performance benefits from its special hardware. A DSP processor has a different arithmetic unit architecture with specialized units like highly parallel multipliers. Its memory uses Harvard architecture which can manipulate data and program at the same time; also, the hardware-based circular buffer allows fetching a stream of data at one time. These characteristics make DSP processors very popular in the field of digital signal processing field.

ARM means Advanced RISC Machine, as its name, ARM is one number of reduced instruction set computing (RISC) architectures microprocessors, which also including Atmel AVR, MIPS, PowerPC, etc. The architecture of ARM processor is designed and licensed by ARM Holdings. For now, ARM processors might be the mostly used 32-bit instruction set architecture in the world. The most famous electronic companies including Apple, Nvidia, Qualcomm are all using ARM microprocessor in their products.

Compared with other microprocessors used in traditional, ARM needs significant fewer transistors, resulting in energy efficient, less heat and low cost. These features make ARM processor fit for energy and cost sensitive products like mobile phone, tablet, etc.

Also, the simpler architecture of ARM helps manufacturers to customize microprocessor according their applications. For the application has high-performance requirements, ARM provides multi-cores or high-frequency solutions to enhance microprocessor performance. The flexibility of ARM processor design attracts more and more manufacturers join this family.

Though most ARM microprocessors are designed for general usage, ARM architecture provides extend options for strengthening digital signal processing performance. ARM Cortex-M and Cortex-A architectures can provide DSP capabilities and other technologies like NEON, multi-core, FPU. These features provide powerful support for signal processing. Furthermore, there are lots of UAV platforms like Paparazzi UAV, 3DR, their autopilot is based on ARM microprocessor, so it worth to do research on real-time signal processing based on ARM microprocessor. During our design, some technologies like fixed-point calculation, circular buffer is used in our design.

2.2.1 Floating-point versus fixed-point

There are basically two kinds of calculations in digital signal processing: floating-point computing and fixed-point computing. Floating-point computing normally need a floating point unit (FPU) inside the processor. Also precision of IEEE-754 standard floating number scales with the order of magnitude of your operands. For FPGA and processor without FPU, fixed-point can provide higher performance and precision. In the Zynq device, if we want to use FPGA accelerate signal processing, fixed-point calculation is the best solution.

To represent a fraction number with the integer, we need to image that it is scaled by a specific "scale factor." The scale factor is usually a power of 2, i.e. 2, 4, 8, 16. So we can easily adjust the result through bits shifting in C Language. For an unsigned data scaled by N, it will range from 0 to $(2^b - 1)/N$, b is the number of bits to be used for storing this number. The distance between two consequent numbers will be $1/N$. For a signed, binary scaled fixed point number, it is presented as Q format: $QM.N$, in which Q stands for the sign, M is the bit amount to the left of the imaginary decimal, N is the bit quantity to the right of the imaginary decimal. N is also the binary scale factor. $Q7.8$ can represent a range from -128.996 to $+127.996$, its precision is $1/256$. In chapter 4 DSP filter software design part, more details about fixed-point filter algrathiom will be introduced.

2.2.2 Circular buffer

A circular buffer is built on a sequential segment of memory, which has a specific capacity, but we can input data without limitation. Circular buffer is one type of first-in first-out (FIFO) buffer. It has one *head* to point to the newest data, and a *tail* to point to the oldest data. The data writing and reading for circular is conducted by moving the position of *head* and *tail*.

In this research, a special circular buffer is designed. There is only one *head* used by this circular buffer. When the circular buffer memory is full, the newest data will always overwrite the oldest data. We can use circular buffer to store a sequence of digital signal data, which can be utilized as input array of a filter. Every time we read all of the data in circular buffer, so *tail* is not necessary. Through filter algorithm calculation, the output signal will be updated. When the delay is acceptable, we can say this is a real-time digital signal processing process.

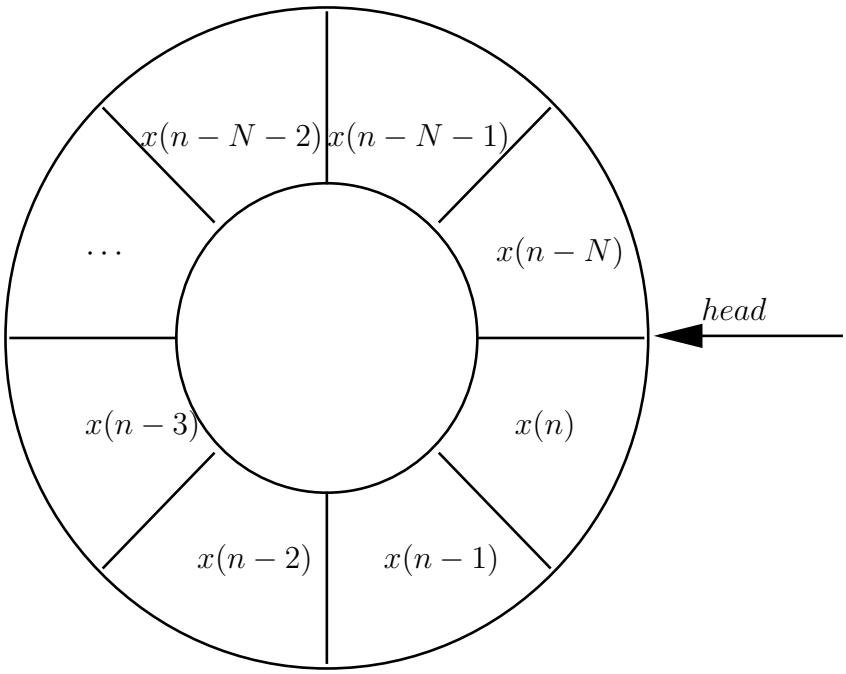

FIGURE 2.2: Circular buffer.

In DSP processor, specific hardware is optimized to for circular buffer management, while there is no special designed circuits in ARM processor. So circular buffer must be implemented on memory at first. Listing 2.2 reveals the typical structure of a circular buffer implemented in C language. In this structure, *buffer_start* and *buffer_end* store the start and end addresses of this circular buffer respectively in memory, they will be initialized when a circular buffer is defined. The *capacity* is to clarify the maximum of data can be stored in this circular data, if the oder of filter coefficients is N, then *capacity* is $N + 1$. The data size, meaning how many memory units will be used for storing one

item of data, is kept by *sz*, for example, the *int16_t* data has a size of 2. The head will always point to the address of the newest data in this circular buffer.

LISTING 2.2: Software circular buffer.

```
typedef struct circular_buffer
{
    void *buffer_start;     // data buffer
    void *buffer_end; // end of data buffer
    size_t capacity;  // maximum number of items in the buffer
    size_t sz;        // size of each item in the buffer
    void *head;       // pointer to head
} circular_buffer;
```

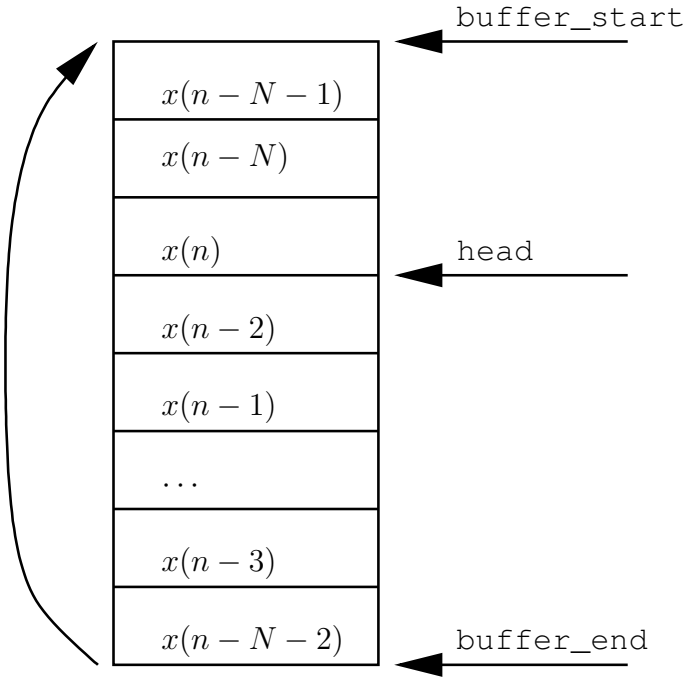

FIGURE 2.3: Circular buffer in memory.

Figure 2.3 shows how the circular buffer work in memory. When new data come, *head* will increase one unit of data size. In the real memory, *buffer_start* and *buffer_end* have different addresses, So *head* has to be relocated to *buffer_start* when it reaches *buffer_end*. When the circular is full, the newest input data $x(n+1)$ will overwrite the oldest data $x(n-N)$, so the arrangement of input data array $\mathbf{x}(k)$ always start from newest data and end with oldest data, as shown on Equation 2.1.

$$\mathbf{x}(k) = [x(k) \quad x(k-1) \quad \cdots \quad x(k-N)]^{\mathrm{T}}. \tag{2.1}$$

2.3 Inter-chip communication protocols

Communication protocols are standards or rules that defined for information exchange between two or more different systems. There are vast of protocols designed for specific situations. In this design, mainly three type communication protocols are used: AXI bus for communication between the ARM processor and FPGA fabric based Intellectual Property (IP), I2C and I2S for communication between SoC and Codec chip, and they are utilized for chip configuration and audio data transmission respectively.

2.3.1 Advanced eXtensible Interface

AXI, which means Advanced eXtensible Interface, is part of ARM advanced microcontroller bus architecture (AMBA) open standard. AMBA standard is the de facto standard for on-chip communication designed by ARM. The current version of AXI is AXI4. Xilinx is a significant contributor to AXI standard. In Zynq device, AXI4 is used for communication between the processor and IP blocks implemented on FPGA. There are three kinds of AXI4 connections as shown in Table 2.2 (Crockett et al., 2014). They can be selected according to requirements for communication.

TABLE 2.2: AXI patterns

Name	Description
AXI4	It is one memory-mapped link, one address can support up to 256 data words transfer.
AXI4-Lite	It is also a memory-mapped link, but one address only supports one data transfer.
AXI4-stream	It is not memory-mapped link. This kind of link can transfer unlimited size of data, and it is designed especially for high-speed data streaming without address mechanism.

In the above definitions, AXI4 and AXI4-Lite links are memory mapped, that means an address in memory space has been assigned to the master of this link.

In the SSM2603 Controller IP design, AXI4-stream is used for DMA data transmission, which requires hight speed transaction. The IP can directly transfer digital audio data without the control of the processor. AXI-Lite is used for SSM2603 register configuration. Every register has specific addresses on memory space. So we can directly write configuration to the SSM2603 register. Moreover, we can read register to get SSM2603 setting or state. In this design, digital audio data are directly read from register through the AXI4-Lite link.

2.3.2 Inter-integrated circuit protocol

The inter-integrated circuit (I2C) protocol is originally invented by NXP. It is designed for low speed and short distance communication between chips on the same board or nearby. There are two wires needed for the I2C bus: SDA for serial data transaction and SCL as the serial clock.

I2C is a widely used protocol for IC communication and one important reason is that its hardware connection is pretty simple. Only two lines with pull-up registers are needed, so the chip will also just use two pins for I2C communication. In most situation, there are one master and several slave chips connected with the I2C bus, but each slave should have a unique address. The master will build communication with one slave through this address at one period. The time sequence of I2C is quite simple, even microcontroller without I2C interface can easily implement this protocol by using two GPIO pins and a timer.

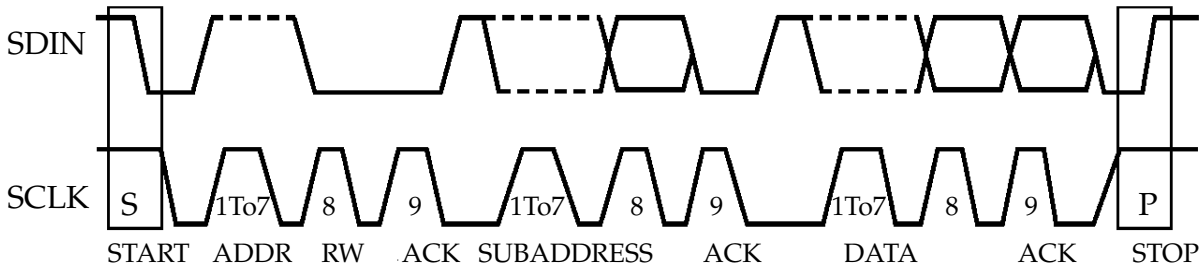

FIGURE 2.4: I2C generalized timing diagram.

Figure 2.4 from the SSM2603 data sheet (Devices, 2013) depicts the generalized timing diagram of I2C bus. As we can see, in ideal condition, SDA and SCL are keeping heigh level. The master always creates the SCL serial clock when the I2C bus starts to work. There are start condition and stop condition to start or stop an I2C command. In both conditions, SCL must keep high, and a high to low change of SDA indicates command start, while a low to high of SDA stands for command end. When one-bit data transmitted on I2C bus, the SCL must keep high level stable. In our design, we use SSM2603 controller to control SSM2603 through the I2C bus.

2.3.3 Integrated Inter-IC Sound bus

I2S stands for Integrated Inter-IC Sound bus, and it is designed for transmitting digital audio data between devices, e.g. analog-to-digital converter (ADC), digital-to-analog converter (DAC) , digital signal processor, etc. In our design, I2S is in charge of audio data stream between SSM2603 and its controller. This bus is also designed with

concept of flexibility and simplicity. For a two channel time-mixed audio data, there are 3 wires be used: SDA and SCL for serial data and clock, and a word select line for channel identification, which is also called called "left-right clock (LRCLK)". Figure 2.5 illustrates the basic timing diagram of I2S from the SSM2603 data sheet (Devices, 2013).

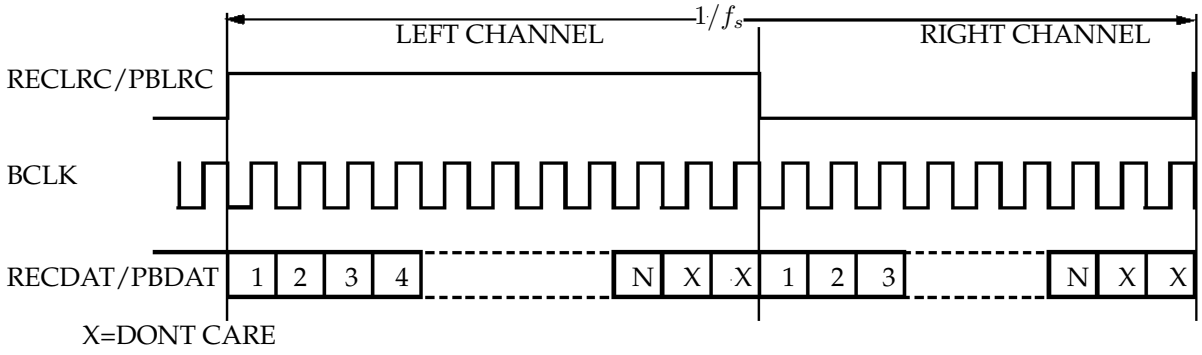

FIGURE 2.5: I2S timing diagram.

The SSM2603 data sheet also introduces three kinds of structures for I2S connection, as shown on Figure 2.6. Their only different is which one is used as the master. In our design, there are two I2S channels are built for original digital audio signal input and processed data output. The SSM2603 controller works as the master in both conditions.

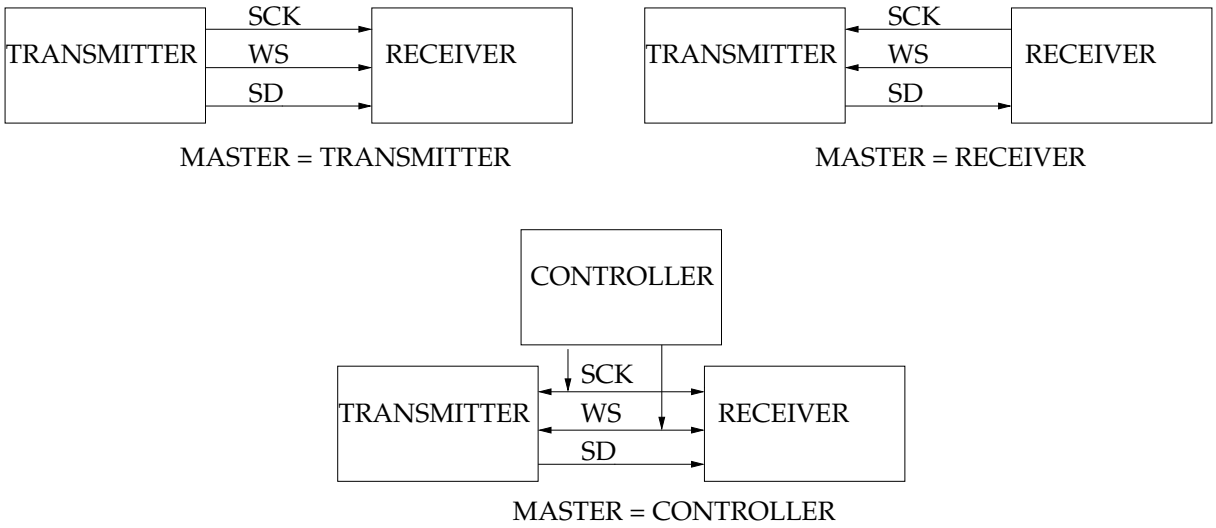

FIGURE 2.6: I2S structures.

Chapter 3

Overview of Digital Signal Processing

Signal processing is a branch of technology that makes data collected from real word around us meaningful. It is the heart of many modern high devices technologies used in our life today. Signal processing powers a range of fields including voice assistant, autopilot car, 5G telecommunication, robotics, etc, among the newest technologies that are making our future. It may be classified by the field of application in video signal processing, imaging processing, audio signal processing, wirelessing communication processing, financial signal processing, etc. Another possible classification can be analog and digital signal processing. In both domain, continuous-time and discrete-time signal, a signal processing system can be labeled linear and nonlinear system.

In this chapter, we will focus on digital audio signal processing and several commonly used filters: meidan filter, FIR filter, and adaptive filters.

3.1 Digital Audio Processing

Digital signal processing (DSP) manipulates digitized signals with the purpose of filtering, measuring, compressing and sound reproducing. An analog signal processing system is built by electronic components like resistors, capacitors, transistors, etc; it is cheap and easy to assemble, but it would be difficult to modify after the design is finished. While a digital signal processing system is a little complex on hardware, it normally contains four components: computing unit, data memory, program memory and I/O interfaces. Table 3.1 shows a detailed explanation about them. A well-designed digital signal processing system can provide flexibility of modification, since it is fully configured and controlled by software. A DSP system also has better performance than an analog system because it usually has more flexibility when compared to hardware limitations.

Digital audio signal processing is one of signal processing applications that specializes on intentional alteration of acoustic signals to achieve a specific goal. The audio signal

TABLE 3.1: Digital signal processing system components

Name	Description
Computing Unit	is responsible for mathematical calculation, processing tasks and managing data and program memory.
Data Memory	stores digital data, works closely with program memory.
Program Memory	stores programming instructions.
I/O interfaces	provides different kinds of communication interfaces with the outside world.

processing is conducted by using digitized data. Figure 3.1 shows a general procedure of digital audio signal processing. Analog input collects the analog signals from devices like microphone, MP3 player, or electronic musical instruments. Then the ADC will transform them to digital signals. The digitized signal will be manipulated in the discrete-time systems, e.g. in a DSP processor, an ARM processor, or in an FPGA fabric for a specific purpose. The processed digital signal will then be converted to an analog signal and reproduce in an analog output device like speaker, headphone. There are dedicated microchips which combine ADC, DAC, and analog signal amplification functions. SSM2603 is one kind of coder-decoder(Codec) chip specially designed for digital audio signal processing, and it will be used in our hardware implement.

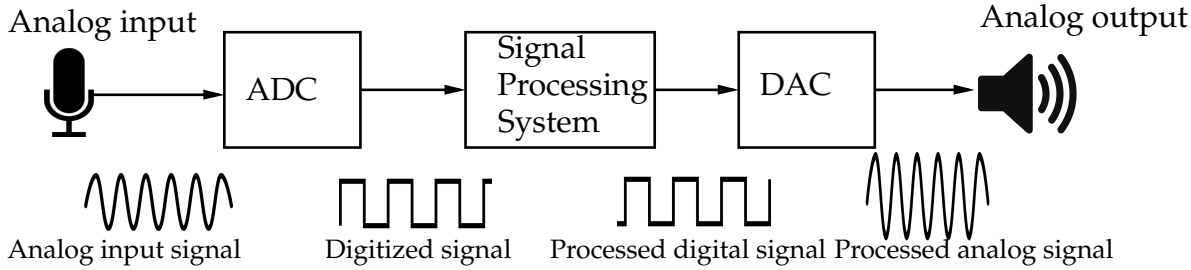

FIGURE 3.1: Digital audio signal processing.

In the field of signal processing, a filter is a device or process that can add some effect to the original signal. The effect can be either signal separation or signal restoration. A filter can attenuate some specific frequencies from the original signal, which can help to reduce interfering signals and suppress background noise. It can also be used for signal restoration, which means a corrupted signal might be enhanced with the help of a properly designed filter.

As previously mentioned, there are two categories of filters in signal processing: analog filters and digital filters. An analog filter has high-speed, low-cost and provides wide-range support on frequency and amplitude, while digital filter can provide very

high performance; it is also easy to simulate and design. Therefor, in a great number of situations, digital filters are used for signal processing applications.

The reminding sections of this chapter will mainly focus on reviewing to three kinds of commonly used digital signal filters, and points out their potential on UAV applications.

3.2 Digital Filters

A discrete-time system using a sequence $x(k)$ signals as input, has an output $y(k)$ through a transformation expressed as in

$$y(k) = T\{x(K)\}, \tag{3.1}$$

$T\{\}$ is the system function. A digital filter corresponds to a linear and linear and time-invariant function which can be expressed as by a different equation.

There are two primary types of discrete-time filters which can be used on digital signal filtering: infinite impulse response (IIR) and finite impulse response (FIR). FIR filter has finite duration impulse response, which will settle to zero in a finite period. Table 3.2 shows pros and cons of FIR filters. Because the advantages of FIR filters is more outstanding than disadvantages, so they have more applications than IIR. In this research, FIR filter is used to implement specific application.

TABLE 3.2: Advantages and disadvantages of FIR filters

Advantages	always stable
	simple to implement compared with IIR
	might achieve linear phase
	startup transients have finite duration
	design methods are generally linear
Disadvantages	filter order is higher than IIR to achieve same performance
	delay is bigger than IIR since higher order

3.2.1 FIR Filters

If the discrete-time FIR filter has order N. The output $y(k)$ is the weighted sum of input $x(n)$ with filter coefficients array:

$$[w_0 \quad w_1 \quad \cdots \quad w_N], \tag{3.2}$$

as shown in

$$y(k) = w_0 x(k) + w_1 x(k-1) + \cdots + w_N x(k-N) = \sum_{i=0}^{M} w_i x(k-i). \tag{3.3}$$

The impulse response of x discrete convolution is

$$h(n) = \begin{cases} w_n & n = 0, 1, \cdots, N, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

Figure 3.2 from (Oppenheim and Schafer, 2011) shows the direct form realization of an FIR system.

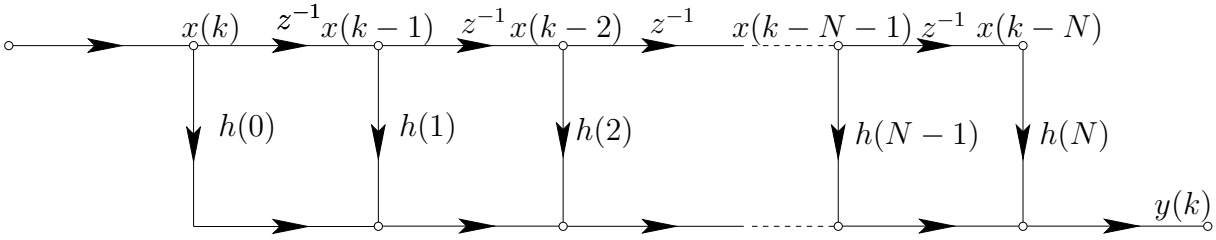

FIGURE 3.2: FIR filter.

There are many kinds of FIR design methods, multiband with transition bands method is used in this thesis. To design an FIR filter for real-time audio digital streaming data, we need to decide coefficients array at first. MATLAB® provides tools usued in this work (functions "firpmord" and "firpm") that helps the design of efficient (in the minimax sense) FIR filters.

FIR filters are usually used for frequency selection, e.g. low-pass filter, band-pass filter, hight-pass filter. These filters have wide range of applications in radio, audio, imaging fields. Also, it is an important part of adaptive filters, which has continually changing coefficients adjusted by a specific adaptive algorithm, more details will be introduced in next section.

3.2.2 Median Filter

For a set of values, the median M is the value that can separate this set to two halves, with that the number of values smaller than M is equal to the number of values lager than M. Considering that the sorted set $x(n)$ has N values; the median M of this set should be:

$$M = \begin{cases} x(\frac{N-1}{2}) & \text{if } N \text{ is odd,} \\ \frac{1}{2}[x(\frac{N}{2}) + x(\frac{N}{2}+1)] & \text{if } N \text{ is even.} \end{cases} \tag{3.5}$$

As we can see, in order to obtain the median value, the values should be sorted at first, and then median can be decided by Equation (3.5). Hence, the process to calculate the median value from a group of numbers is a nonlinear operation. Single value in the list with a very small or large magnitude does not influence the median, so it is robust regarding to outliers.

Median filters are well-known for their applications on image and audio processing. It is a nonlinear digital filtering technique for noise reduction. The most common use of median filter might be removing some noise, salt and pepper noise, in imaging signal processing. In the field of digital audio signal processing , median filters are also routinely used on noise reduction. One example is signal enhancement for gunshot direction of arrival (DOA) estimation (Borzino et al., 2015). The recorded audio data are typically processed on a powerful computer and there is no problem in off-line applications. Otherwise, if the median filter is to be implemented in real-time, there could be a problem due to the need of a delay to store the set of samples and sorting it . In this research, we will explore the use of real-time median filter for UAV applications.

3.3 Adaptive Filters

Similar to FIR filter, an adaptive filter system has a group of coefficients, it can create the output signal similarly to Equation (3.3). However, it also can continuously compare its output signal with the desired signal, and modify coefficients with the comparison result through an adaptive algorithm. With the power of digital signal processing system increasing, more and more applications are using adaptive filters.

Figure 3.3 shows the block diagram of a general adaptive filter. There are two inputs in this diagram: $x(k)$ stands for the original input signal; $d(k)$ stands for reference input signal, which is the desired signal we hope output to be. The output signal is $y(k)$, and another signal $e(k)$, which is called error signal is used to adjust coefficients of the filter. m represents the discrete sample number. If the order of coefficients or weights is N, the weight vector $\mathbf{w}$ of adaptive filter would be:

$$\mathbf{w} = [w_0 \quad w_1 \quad \cdots \quad w_N]^{\mathrm{T}}. \tag{3.6}$$

since,

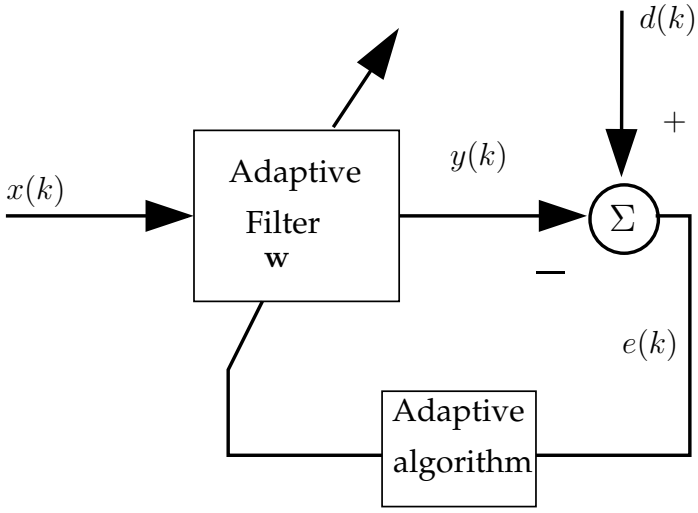

FIGURE 3.3: Basic configuration of an adaptive filter.

$$\mathbf{x}(k) = [x(k) \quad x(k-1) \quad \cdots \quad x(k-N)]^{\mathrm{T}}. \tag{3.7}$$

Then we can calculate the output of $y(k)$:

$$y(k) = \sum_{i=0}^{N} w_i x(k-i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}(k). \tag{3.8}$$

The error signal is generated by removing the output signal $y(k)$ from the desired signal $d(k)$:

$$e(k) = d(k) - y(k) \tag{3.9}$$

The objective of an adaptive filter is to minimize the function applied to the error signal which is carried out for the adaptive algorithm. In some specific applications, the adaptive algorithm tries to make error signal $e(k)$ approximate to zero and finally generate a signal at the output without undesired noise or interference signals. However, in other application, the error signal might be the interested signal, as the adaptive noise cancellation. It will addressed in the following subsection.

3.3.1 The LMS and NLMS Algorithms

To achieve the above objective, we need to design the adaptive algorithms to optimize the coefficients. LMS and NLMS are two algorithms routinely used by adaptive filters.

Least Mean Squares (LMS) algorithm was raised by Professor Bernard Widrow and his student, Ted Hoff in 1960 at Standford University. It is one kind of adaptive filter that can literately adjust its coefficients to create the desired filter, which will produce the least mean square of error signals.

For a conventional LMS algorithm, Equation (3.10) shows how the weight vector is updated. Here, $\mathbf{x}(k)$ is the input signal and $e(k)$ is the error signal. μ is the step-size or convergence parameter.

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mu \mathbf{x}(k) e(k), \tag{3.10}$$

LMS is mostly used algorithm because of its stable and easy to implement. However, its convergence is slow in many applications. The performance of the LMS algorithm is highly dependent on the input signal condition, on the filter order and on the step-size μ.

Another important drawback of LSM is that it is quite sensitive to the scale of the input. A variant of the LMS algorithm called Normalized least mean squares (NLMS) is proposed. At first, in LMS algorithm, the step-size μ is fixed for a specific scale of the input signal, while in the general environment, the scale of the input signal is unknown. In the NLMS algorithm, the size of μ is variable, depending on the square mean of the input signal vector. The updating equation of the NLMS algorithm expressed as

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \frac{\mu}{\epsilon(k) + \mathbf{x}^T(k)\mathbf{x}(k)} \mathbf{x}(k) e(k). \tag{3.11}$$

$\epsilon(k)$ is a small positive constant, keeping $\epsilon(k) = \epsilon$ leads to a so-called ϵ-NLMS algorithm (Yousef and Sayed, 2001). This version of NLMS algorithm is usually used by practical implementations because the ϵ can prevent the occurrence of division by zero. (Ramos et al., 2017) shows real-time implementations of acoustic signal enhancement techniques for aerial based surveillance and rescue applications powered by NLMS algorithm.

3.4 Adaptive Noise Cancellation

The basic concept of adaptive noise cancellation is to remove the noise from a received signal to improve the signal-to-noise ratio (SNR). One of its most useful applications is to reduce the propeller noise from audio collected by microphones on UAVs. In the rescue scenario, if the rescue team can directly communicate with people who needs help through UAV, it will be significantly helpful. However, the precondition in this

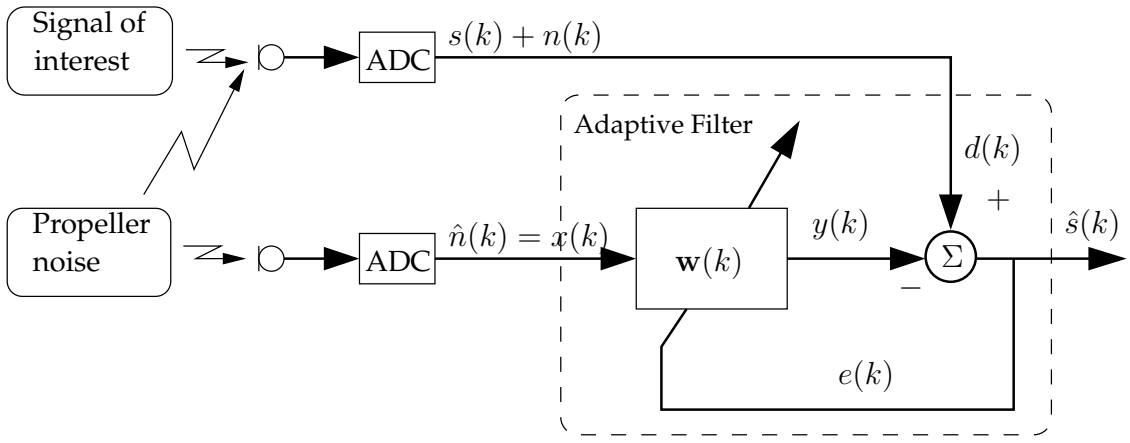

FIGURE 3.4: Adaptive filter noise cancellation setup.

application is that the input signal to the adaptive filter is correlated to the noise collected by the airborne microphone while uncorrelated to the signal of interest. In this application, we can use the adaptive filter to remove propeller noise while retaining the audio we need.

A propeller noise cancellation system is shown on Figure 3.4. In this figure, $s(k)$ is the audio signal we want to acquire, but we can not get it directly, since it is corrupted by noise signal $n(k)$. We only collect $s(k) + n(k)$. To separate $n(k)$ from $s(k)$, we need to use an adaptive filter.

In this adaptive noise cancellation setup, $s(k) + n(k)$ is used as reference signal or desired signal. Another highly correlated version of the noise component, $\hat{n}(k)$, is used as input signal, $x(k)$. The coefficients $\mathbf{w}(k)$ are iteratively modified by the adaptive algorithm, the NLMS in this work. Such that the adaptive filter output $y(k)$, will be close to the noise component $n(k)$. The error signal $e(k)$, denoted as $\hat{s}(k)$, will approximate the signal of interest $s(k)$. A NLMS algorithm based adaptive filter will be addressed for propeller noise cancellation application on UAV.

Chapter 4

Real-time Digital Audio Filters Implementation

4.1 Development environment

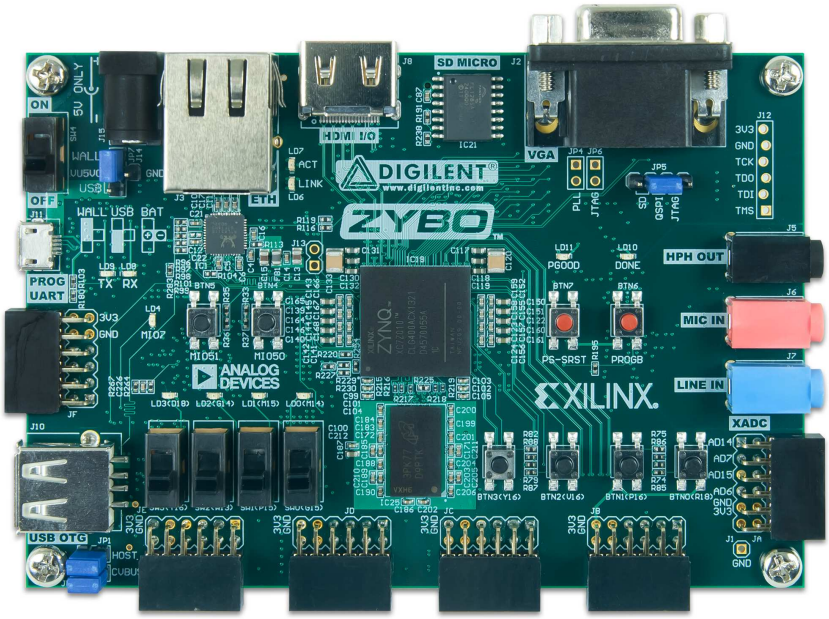

FIGURE 4.1: ZYBO.

The ZYBO (ZYnq BOard) is a development board designed for Xilinx Zynq-7000 All Programmable SoC. It is built based on Z-1010, one enter-level chip of the whole Zynq family. The Z-1010 has a duel-core ARM processor tightly combined with FPGA fabric. There are abundant resources e.g., audio and video I/O, USB and Ethernet interfaces, on-board memory, SD slot, etc., are ready on board, which can support users implement their design easily. It also provides six Pmod ports for extending needs. The support for JTAG programming and debugging, UART to USB conversion makes the work of developer easy. The size of ZYBO is pretty small compared with other Zynq

platforms like Zedboard, which make it fit for airborne usage. Based on the above reasons, ZYBO is chosen as the hardware platform of this design.

To handle development on ZYBO, we need an efficient and user-friendly development environment. Luckily, Xilinx provides a series of integrated development environment (IDE) to help us in the design flow. In this design, Vivado is used for hardware design and SDK is used for software design.

4.1.1 Vivado

Vivado is released in April 2012 by Xilinx for edit, simulation, synthesis, analysis and implement of Hardware Description Language (HDL) design. Vivado is a design environment for Xilinx FPGA products, and it supports devices including Ultrascale, Virtex-7, Kintex-7, Artix-7, and Zynq-7000. Vivado changes design flow for HDL and adds new features for SoC development and high-level synthesis comparing with ISE, which is an old fashion IDE for Xilinx products before Vivado appearing.

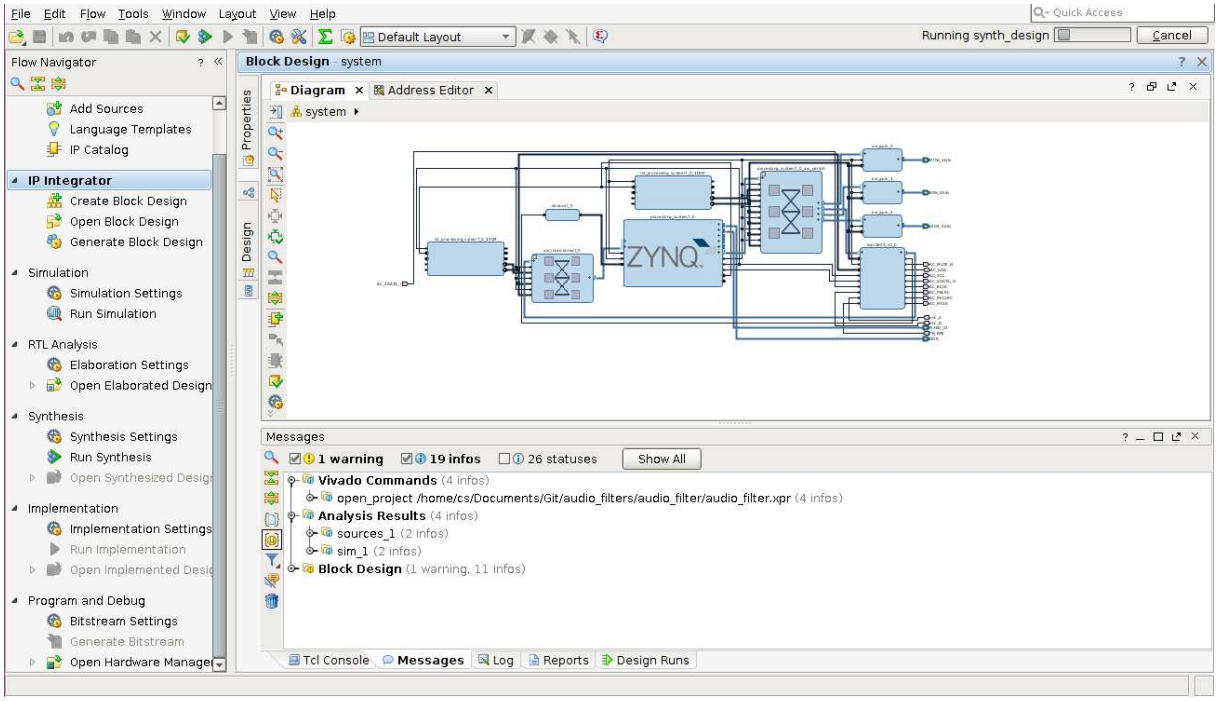

FIGURE 4.2: Vivado IDE overview.

The newly designed Vivado is very friendly to Zynq device development. The developer can even finish their hardware design by connecting standard IP blocks and configuring them without writing HDL codes. It also provides IP tools so that developers can develop their IP block with the desired function. These features can provide good support for our hardware design. The newest version of Vivado is 2016.04 for now,

which is used as our hardware development environment. The overview of Vivado IDE is shown in Figure 4.2.

4.1.2 SDK

Xilinx Software Development Kit (SDK) is another useful IDE specialized for embedded software development on Zynq-7000 AP SoCs, MicroBlaze soft-core implemented on FPGA fabrics and Zynq UltraScale+ MPSoC. The SDK provides C/C++ source code editor and compilation environment. Moreover, it has a powerful debugger which supports common debug functions, e.g., breakpoint setting, stepping execution, variable value check, etc. It allows developers conduct software application development on the hardware platform created through Vivado IDE. After finishing hardware design through Vivado IDE, the developer can export the customized hardware design to SDK, then SDK will atomically initialize software development environment parameters for the developer, including bitstream file for FPGA programming, peripheral drivers, compiler setting, library path, memory map, etc. These automatic configurations will help developer focus on their application development, and boost development process.

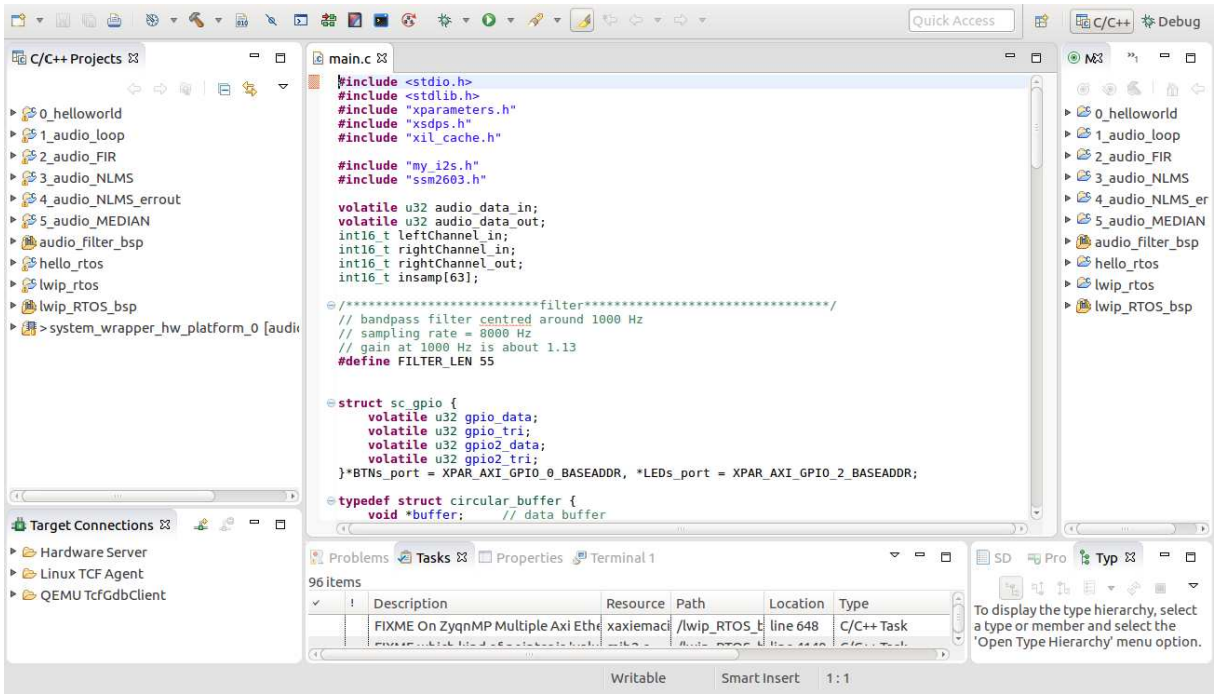

FIGURE 4.3: SDK IDE overview.

The same version of SDK is used in this design to keep compatible with Vivado, which should be installed with Vivado from the same installation package. The overview of SDK IDE is shown in Figure 4.3.

4.2 Hardware Design

The whole system is implemented on ZYBO, which has been introduced in section 4.1. It is obvious that we do not need all of the components on board for our real-time digital audio processing platform. Figure 4.4 shows the system architecture and table 4.1 explains three main components of it.

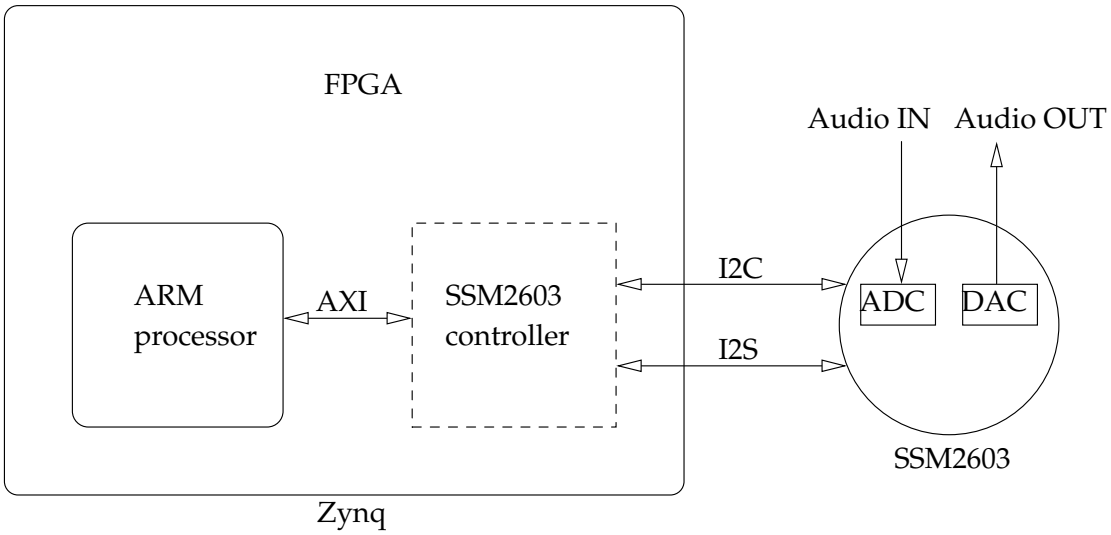

FIGURE 4.4: Real-time signal processing system architecture.

TABLE 4.1: Real-time digital audio signal processing components

Name	Description
Processor	It is a dual-core ARM Cortex-A9 processor, one part of Xilinx Zynq-7000, AP SoC architecture.
Audio Codec IC	SSM2603 from Analog Devices, providing stereo, 48 kHz sampling rate, 16-bit ADC and DAC conversion in this case.
Codec IC controller	It is implemented on FPGA part of the Zynq SoC, specially designed for I2C and I2S protocols communication between ARM processor and SSM2603.

In this section, the three critical components and how they are used in our design will be introduced.

4.2.1 Zynq SoC

The core of the whole system is Zynq SoC, which consists of two parts: Processing System (PS) is a dual-core ARM Cortex-A9 processor, and Programmable Logic (PL) is the FPGA fabric.

The PS section is not just composed of an ARM processor. Instead, it is a whole processing system which contains a series of resources forming application processing unit and peripheral interfaces, memory interfaces, different kinds of ports to communicate with FPGA, etc. PS is a "hard core", which is fixed and optimized on the silicon wafer. Comparing with MicroBlaze processor which can be located in the logic fabric of an FPGA, PS can provide higher performance with lower power consumption.

The PL section in Zynq is mainly composed of general purpose FPGA logic fabric and Input/Output Blocks(IOBs) for interfacing. It also contains DSP48E1s and Block RAMs for special use. The PL has some hard IP components for dedicated functions: analogue to digital conversion, clocks and JTAG for programming and debug.

In our design, PS support software design so that we can using C language programming the whole process for signal processing. PL is used for implementing high-speed data transmission. PS and PL are connected through AXI protocol.

4.2.2 ARM Processor

As we have introduced in former content, a dual-core ARM Cortex-A9 processor is the critical component of PS. The business model of ARM company is to authorize Original Equipment Manufactures (OEMs) using ARM processor IP in their product. There are flexible configurations OEMs can choose when designing their product. Xilinx is one of the OEMs, and it integers ARM Cortex-A9 processor into their Zynq device with FPGA fabrics. Figure 4.5 shows the architecture of ARM processor inside Zynq. ARM Cortex-A9 can have 1 to 4 cores, Xilinx specifies a dual-core processor. The Level 1 cache used as data and instructions buffer between processor and memory, it could be 16KB, 32KB or 64 KB, and Zynq chose 32KB Level 1 cache. MMU is memory management unit. There are also some optional elements that ARM processor can have, Zynq includes NEON extension FPU which is very help for digital signal processing (DSP).

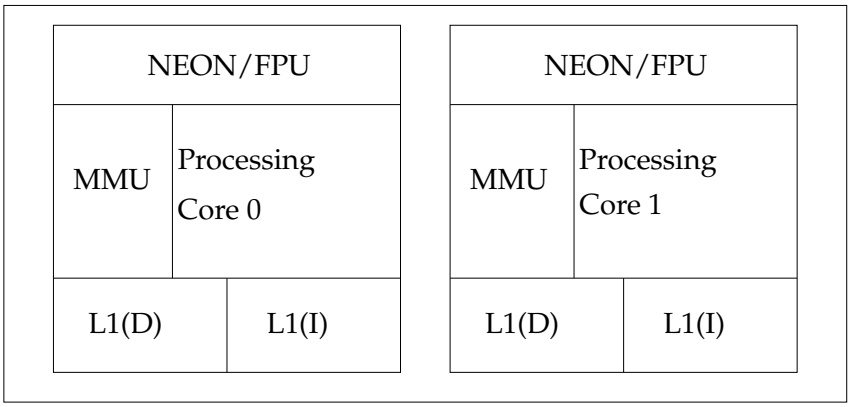

FIGURE 4.5: ARM Cortex-A9 processor in Zynq.

The ARM processor supports a set of assembly instructions. Through Xilinx Software Development Kit (SDK), the developer can implement programs or algorithms by using C/C++ language. The SDK will help user compile codes to assembly instructions. Details about the software development will be introduced in software design part.

4.2.3 Audio Codec Chip

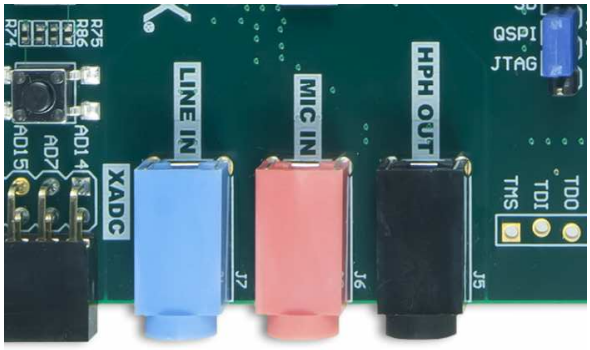

FIGURE 4.6: Audio interfaces on ZYBO.

SSM2603 is one audio chip from Analog Devices, which is widely used on portable electric products, e.g. mobile phone, mp3 player, etc. SSM2603 supports low power consumption and high-quality stereo audio Codec. The sampling rate is up to 96kHz and working environment temperature vary from -40°C to +85°C. Its analog input supports stereo line and monaural microphone inputs. The DAC output signals can be presented at both the stereo line outputs and stereo headphone outputs. The digital result of ADC can be up to 24bits per channel.

As shown on Figure 4.6, there are three 3.5mm standard audio jacks on ZYBO. Two inputs, stereo line input and mono microphone input have blue and pink color respectively. The black jack is headphone output. In this design, stereo line input and headphone output are chosen for audio signal input and output. The digital signal format is 16bits, and the sampling rate is 48kHz. SSM2603 is configured by setting internal registers, and there are nineteen 9bits registers used for SSM2603 configuration. I2C protocol provides access to SSM2603 registers, and I2S protocol is in charge of audio digital data transmission.

4.2.4 SSM2603 controller

SSM2603 Codec chip has been introduced in the previous section. To configure SSM2603 and implement communication with the processor, an IP called SSM2603 Controller

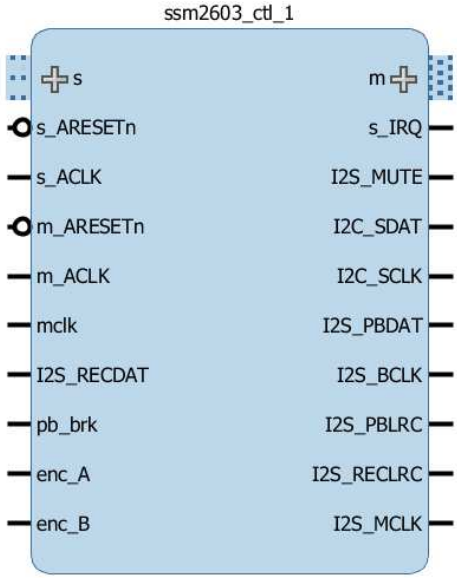

FIGURE 4.7: SSM2603 controller IP.

has been designed. It is entirely implemented in FPGA part of Zynq device using Verilog Language. Table 4.2 shows the interface definitions.

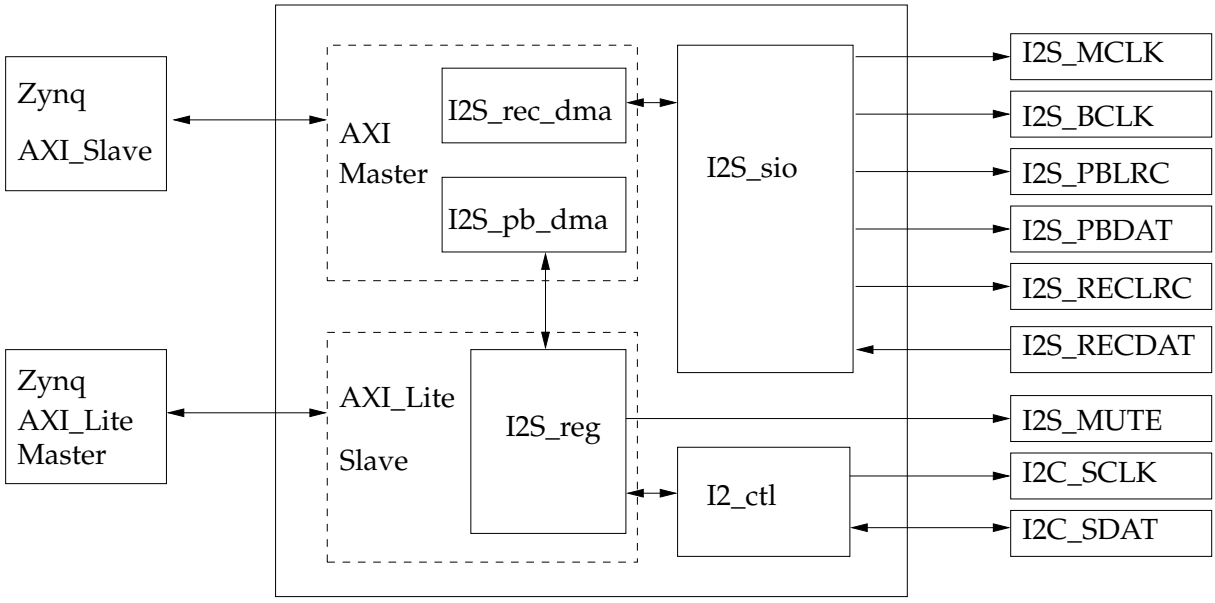

FIGURE 4.8: SSM2603 controller structure.

As shown on the Figure 4.8, the SSM2603 controller IP has five parts which help to realize different functions. The I2S_rec_dma and I2S_pb_dma blocks are used in record mode and playback mode for DMA data transmission, they are not utilized by this design. The I2S_reg specifies register position and definition for SSM2603 configuration. I2S_sio is the I2S interface implement, and I2C_ctl realize the I2C control interface for SSM2603. On the top level of this IP, it provides AXI-Lite and AXI-stream interface to communicate with the ARM processor. Different blocks in this IP cooperate with each other to make the audio signal processing work smoothly.

TABLE 4.2: IP interfaces description

Name	Description
s, s_ARESETn, s_ACLK	AXI4_lite Slave interface
m, m_ARESETn, m_ACLK	AXI4_stream master interface
mclk	Clock input, connecting to FCLK_CLK2 12.288MHz, used to create digital audio bit clock.
I2S_RECDAT	Input signal, I2S SDA wire for transferring record data (ADC process).
I2S_MUTE	Output signal, DAC mute.
I2S_PBDAT	Output signal, I2S SDA wire for transferring playback data (DAC process).
I2S_BCLK	Output bit clock for SSM2603, I2S SCL wire for transferring digital audio data (ADC and DAC processes).
I2S_PBLRC	Output signal and specifies left-channel or right-channel of digital audio on I2S bus in playback mode.
I2S_RECLRC	Output signal and specifies left-channel or right-channel of digital audio on I2S bus in record mode.
I2S_MCLK	Master clock output to SSM2603, it equals to mclk, and it affects the sampling rate of ADC.
I2C_SDAT	I2C 2-Wire Control Interface Data Input/Output
I2C_SCLK	I2C 2-Wire Control Interface Clock Input.

4.3 Software Design

Chapter 2 has discussed the comparisons about floating-point versus fixed-point, circular buffers in DSP processor and general processor. In this research, filers are designed by C Language and executed on ARM processor, so fixed-point arithmetic and software circular buffer are the basic techniques used on software design. Since the memory based software circular buffer design has been introduced in section 2.2.2, its structure is simple and general, so it can be used by any kind of filter input data management. The software design now focus on fixed-point implement of median filer, FIR filter and NLMS adaptive filter.

4.3.1 Fixed-point FIR filter implementation

The implement structure of fixed-point FIR filter is shown in Figure 4.9. The digital audio data samples created by ADC is n-bits signed integer and n is 16 in our case.

The top one bit stands for sign and other 15 bits represent absolute value. According to equation 3.3, $y(k)$ is multiply accumulate result of $\mathbf{x}(k)$ and $\mathbf{W}(k)$.

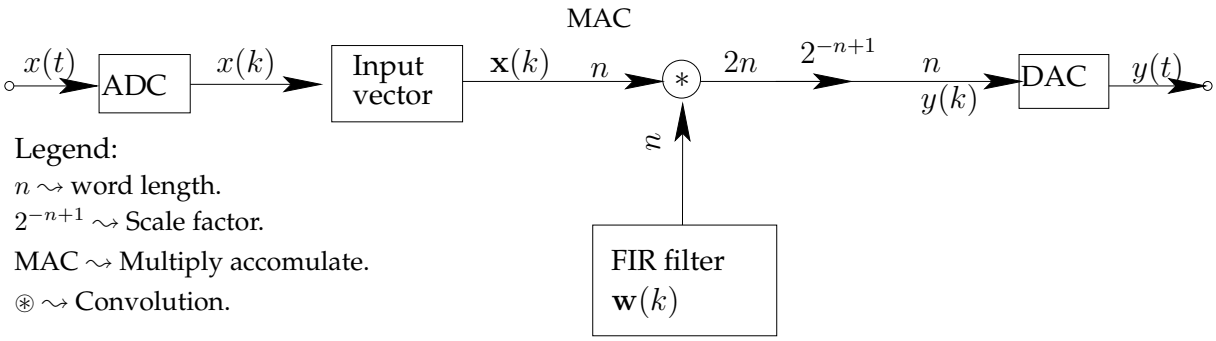

FIGURE 4.9: Fixed-point FIR filter implementation structure.

In this FIR software design, fixed-point arithmetic is used in multiply accumulate calculation. The coefficients created by MATLAB is a series of decimal numbers, and they multiply with a scale of 2^{15} before doing filter calculation. After weighted sum we get the result, rounding it by add half of the scale factor which is 2^{14}, then scale it with 2^{15} by shifting right 15 bits before output. Listing 4.1 reveals fixed-point FIR filter function implemented by C Language. There are three inputs required by this function. The "coeffs" points to the start of scaled coefficients array made by MATLAB. The "cb_input" is the pointer of circular buffer structure, it will provide the entry of $\mathbf{x}(k)$. The "filterLength" is the value of k. In the end of this code listing, the final result of $y(k)$ is downscaled by shifting 15 bits to right.

LISTING 4.1: Fixed-point FIR filter implemented by C Language.

```
// FIR filter function
int16_t firFixed(int16_t *coeffs,circular_buffer *cb_input,
    int filterLength) {
  int32_t acc;      // accumulator for MACs
  int16_t *coeffp; // pointer to coefficients
  int16_t *inputp; // pointer to input samples

  int k;
  // calculate output n
  coeffp = coeffs;
  //point inputp to the oldest data
  if (cb_input->head == cb_input->buffer_end) {
    inputp = cb_input->buffer;
  } else {
    inputp = cb_input->head;
  }
  // load rounding constant
  acc = 1 << 14;
```

```
  // perform the multiply-accumulate
  for (k = 0; k < filterLength; k++) {
    //when the input array pointer point to the end of buffer, jump to
    buffer start
    if (inputp == (cb_input->buffer_end)) {
      inputp = cb_input->buffer;
    }
    acc += (int32_t)(*coeffp++) * (int32_t)(*inputp++);
  }
  // saturate the result
  if (acc > 0x3fffffff) {
    acc = 0x3fffffff;
  } else if (acc < -0x40000000) {
    acc = -0x40000000;
  }
  // convert from Q30 to Q15
  return (int16_t)(acc >> 15);
}
```

4.3.2 Fixed-point NLMS adaptive filter implementation

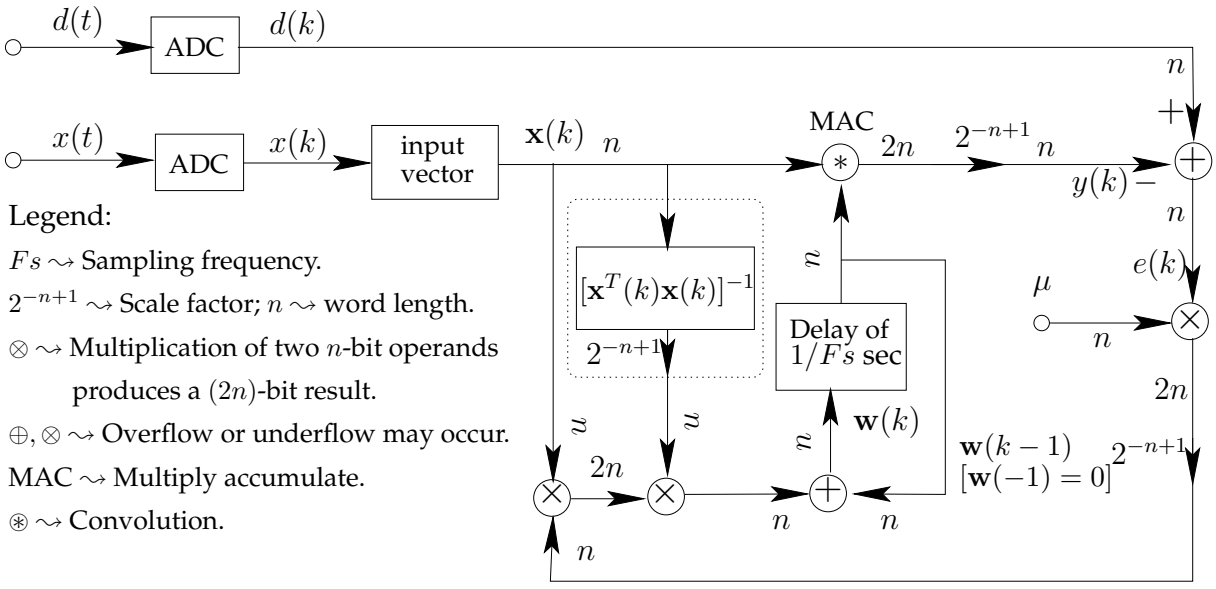

FIGURE 4.10: Fixed-point arithmetic implementation structure for the NLMS algorithm.

When implementing the algorithm using 32-bit fixed- or floating-point arithmetic, precision is not an issue, and one can focus solely on achieving the maximum computational efficiency. However, in a 16-bit fixed-point implementation, the reduced precision becomes an issue and needs to be addressed. Overflow or underflow in intermediate calculations may cause the algorithm to diverge or not converging at all. For

instance, in a floating-point implementation, a small step size can ensure a small steady state error. However, a step size of the same magnitude might be too small for a 16-bit fixed-point implementation of an adaptive filter.

Figure 4.10 illustrated the sequence that has been used in the 16-bit fixed-point implementation of the NLMS algorithm. It was assumed one bit of sign, and a scaling factor of 2^{15}. Note that downscaling ($\gg 15 \equiv \div 2^{15}$), implemented using a simple right-shift (see Listing 4.2), is performed after each 16-bit multiplication, except prior to the final division by input power $\mathbf{x}^T(k)\mathbf{x}(k)$. In this implementation we considered a straightforward computation of the inner product $\mathbf{x}^T(k)\mathbf{x}(k)$. However, a more computationally efficient implementation can achieved by updating the normalized input power $p(k) = \mathbf{x}^T(k)\mathbf{x}(k)/N$ instead of $\mathbf{x}^T(k)\mathbf{x}(k)$ using (Sayed, 2003)

$$p_x(k) = \beta p_x(k-1) + (1-\beta)x^2(k), p_x(-1) = 0, \tag{4.1}$$

for some forgetting factor $0 < \beta < 1$.

LISTING 4.2: Pseudocode for the 16-bit implementation of the NLMS algorithm.

```
for each new sample(k) do { acc = xx(k) = 0;
for ( i = 0; i < filterLength; i++){
  xx(k) +=  ((int32_t)(x[i]) * (int32_t)(x[i]))>>15; // MAC with scaling
  acc += (int32_t)((int32_t)*w[i]) * (int32_t)(x[i]); // MAC
}
y(k) = (int16_t)(acc >> 15); // Converting from Q30 to Q15
e(k) =  d(k) - y(k);
delta = (int32_t)((mu  *  (int32_t)e(k))>>15);
for(j = 0; j < filterLength; j++)
    weights[j] += (int16_t)((delta * (int32_t)x[j])/(1 + xx(k)));
  }
```

4.3.3 Median filter implementation

The critical component of median filter is how to get median value from an array containing 16-bits audio signal data. Listing 4.3 depicts the C function of median value fetching. There are two arguments in this function, x is the start address of circular buffer, n is the same with the length of circular buffer. Every time when it is executed, all digital data inside circular buffer will be sorted in ascending order, the median value will be returned according to equation 3.5. The value of n should be neither to big or to small. A big n will bring unnecessary load to processor and increase delay time between input and output. A small n will decrease median filer performance and make it useless.

LISTING 4.3: Median fetch function implemented by C Language.

```
int16_t median(int16_t n, int16_t* x) {
  int16_t temp;
  int16_t i, j;
    // the following two loops sort the array x in ascending order
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if(x[j] < x[i]) {
                // swap elements
                temp = x[i];
                x[i] = x[j];
                x[j] = temp;
            }
        }
    }
    if(n%2==0) {
    // if there is an even number of elements, return mean of the two
   elements in the middle
        return((x[n/2] + x[n/2 - 1]) / 2);
    } else {
    // else return the element in the middle
        return x[n/2];
    }
}
```

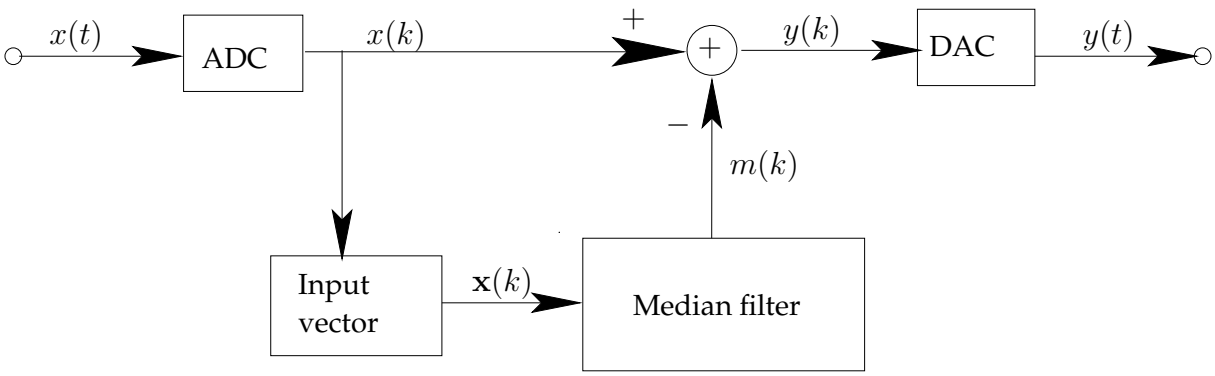

FIGURE 4.11: Gunshot enhancement median filter implementation.

In the UAV based gunshot DOA introduced by (Borzino et al., 2015), the input vector $\mathbf{x}(k)$ is not replaced by its median directly. The purpose of this median filter is to enhance the muzzle blast component to improve the follow-up DOA estimation. The original sound is modeled as the combination of pure gunshot signal and unwanted residual. Through the median filter, we get the estimate unwanted residual. So the enhanced gunshot signal can then be easily computed as $x(k) - m(k)$. Figure 4.11 shows the whole process of the gunshot enhancement median filter implementation.

Chapter 5

Experiments and results

On this chapter, we will discuss filters design and simulation on MATLAB. A series of experiments are designed and conducted to verify the performance of real-time audio signal processing on ZYBO. The results are recorded by the oscilloscope, and corresponding analysis about performance will be made.

5.1 Median filter experiments and results

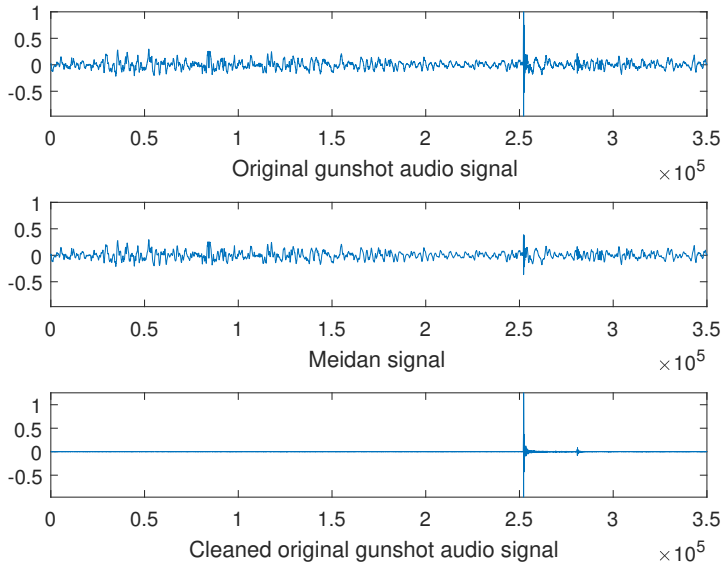

FIGURE 5.1: Gunshot audio signal cleaned by median filter.

In this experiment, the median filter is used to clean the gunshot audio signal noise. The objective of this application is to get a clearer edge between the gunshot happened time and other time, and also decrease the white noise. Here we use a gunshot audio created by a rifle. This audio file is recorded at a close distance (236m) from the shooting position. As explained on equation 3.5, N is chosen at first. It should be neither

too big nor too small. After a series of test, 30 is chosen. Figure 5.1 shows the results of conducting median filter cleaning on MATLAB. The filtered signal shows more clear edge compared with the original signal.

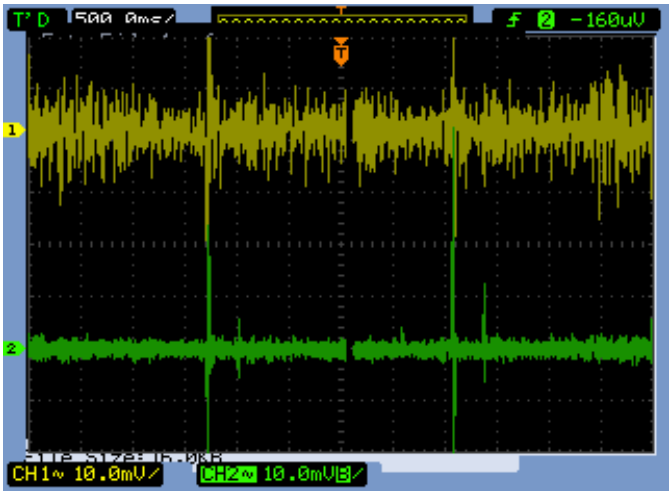

FIGURE 5.2: Median filter real-time comparison results.

An experiment is designed to verify the real-time median filter. Figure 5.2 show the real-time comparison between original gunshot audio signal and the signal cleaned by median filter on ZYBO. It clearly shows the gunshot happening part and significantly decreases other noises.

5.2 FIR filter experiments and results

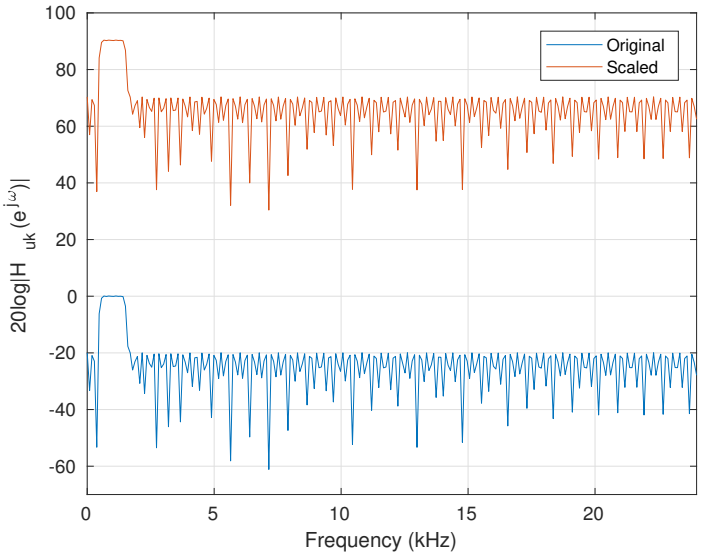

FIGURE 5.3: FIR frequency response.

This experiment will help us to test FIR filter performance on this real-time digital audio processing platform. By using multi-band with transition bands method on MATLAB, I designed a band-pass FIR filter with frequency response around $1KHz$. Since we use fixed- point arithmetic on ZYBO platform, the final coefficients are multiplied by 2^{15}. Figure 5.3 depicts the frequency response of this band-pass FIR filter.

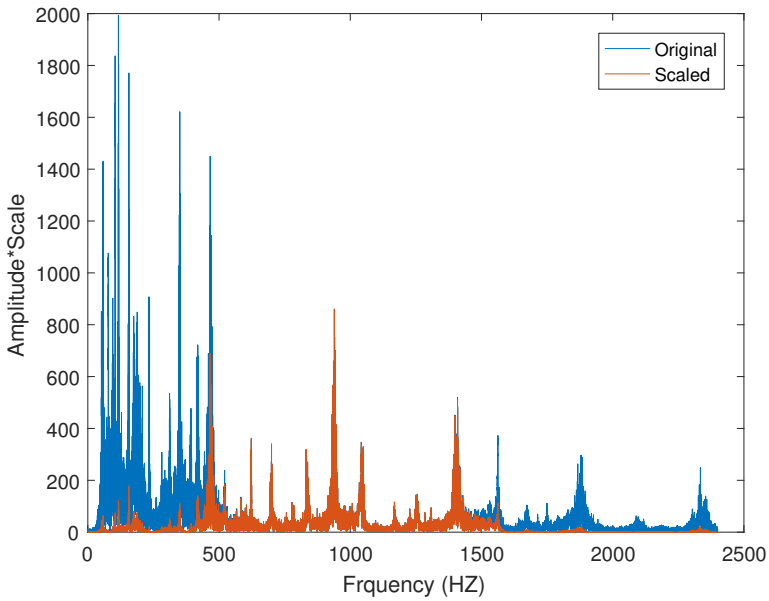

FIGURE 5.4: Audio samples comparisons on time and frequency domains.

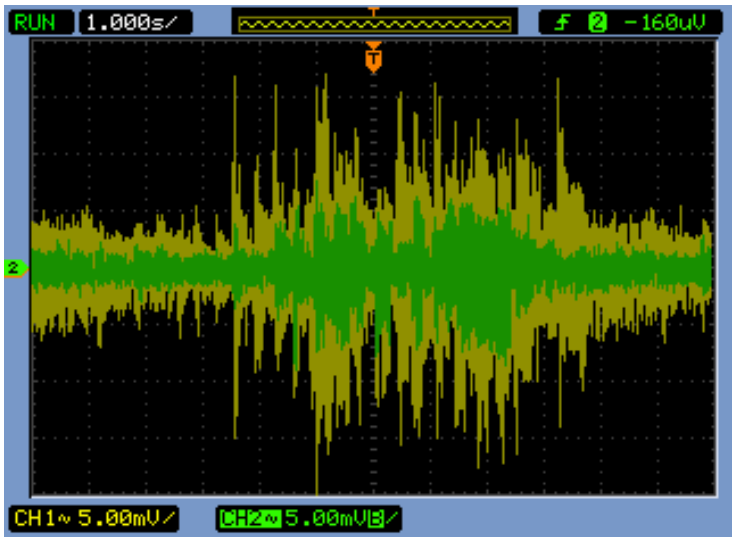

FIGURE 5.5: FIR original channel compared with real-time filtered channel.

This FIR filter is implemented and debugged on computer at first. It can only filter fixed audio file. By inputing an audio sample, we can get the filtered audio sample, which should only keep the frequencies around $1KHz$. Figure 5.4 shows the comparisons between original audio signal and filtered audio signal on frequency domains.

The blue signal is the original signal, and the brown signal is filtered signal. As we can see, the frequencies around $1KHz$ are kept, while other frequencies are significantly weakened. This test reveals that the coefficients of this FIR can properly achieve our original target.

The final purpose of this experiment is test the FIR filter performance on the real-time audio signal processing platform. We hope that real-time FIR filter on ZYBO has the same performance with FIR filter implemented on laptop. In order to verify this hypothesis, a comparison is made. As shown in Figure 5.6, two signals are connected with oscilloscope channels. The green signal is filtered signal created by the real-time signal signal processing platform, and the yellow signal is previously filtered by computer program. They are played on the same time track. As we can see, the two signals almost overlap with each other. It shows that real-time FIR filter on ZYBO almost has the same performance with FIR filter implement on the laptop.

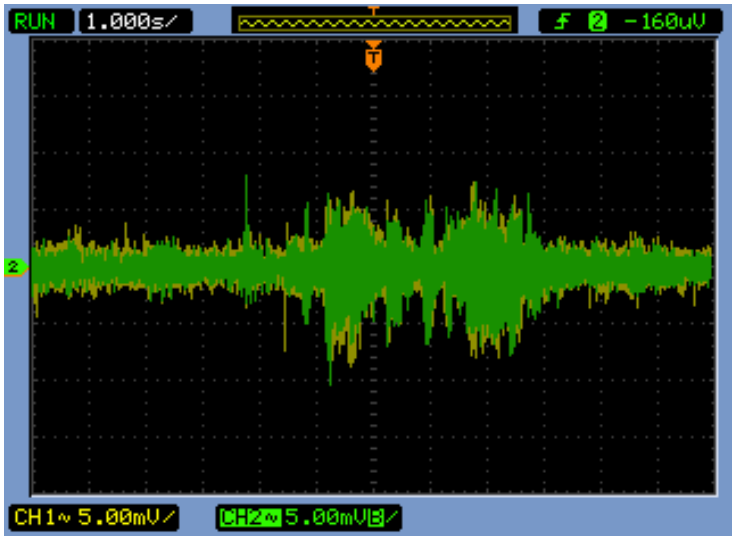

FIGURE 5.6: Comparison between filtered signal and real-time filtered signal.

5.3 NLMS adaptive filter experiments and results

The convergence property is very critical to an adaptive filter. A 16-bit fixed-point implementation of NLMS adaptive filter was carried out using the C programming language at first. This implementation was tested and debugged in a laptop and then adapted to the ZYBO. Figures 5.7 and 5.8 show results for a 16-bit fixed point implementation of the NLMS algorithm. In this particular example, a system identification setup is considered in order to evaluate the convergence properties of this algorithm. Note that, in this case, the theoretical dynamic range is of approximately 90 dB, equivalent to 6 dB per bit. The adaptive filter parameters are the same on both experiments

and were set as follows: filter order N = 500; sampling frequency Fs = 48 kHz; step size $\mu = \texttt{round}(0.05 \cdot 2^{15}) = 1638$. The unknown system is a bandpass filter centered around 1 kHz. Two different scenarios for the stop band attenuation have been considered: 40 dB (Figure 5.7) and 60 dB (Figure 5.8). In the first case, the system was identified with great accuracy, whereas in the second one, the effects of misadjustment and roundoff errors due to the reduced precision are evident on the side lobes. However, both cases exhibited similar performances in terms of the minimum Mean Square Error (MSE) attained.

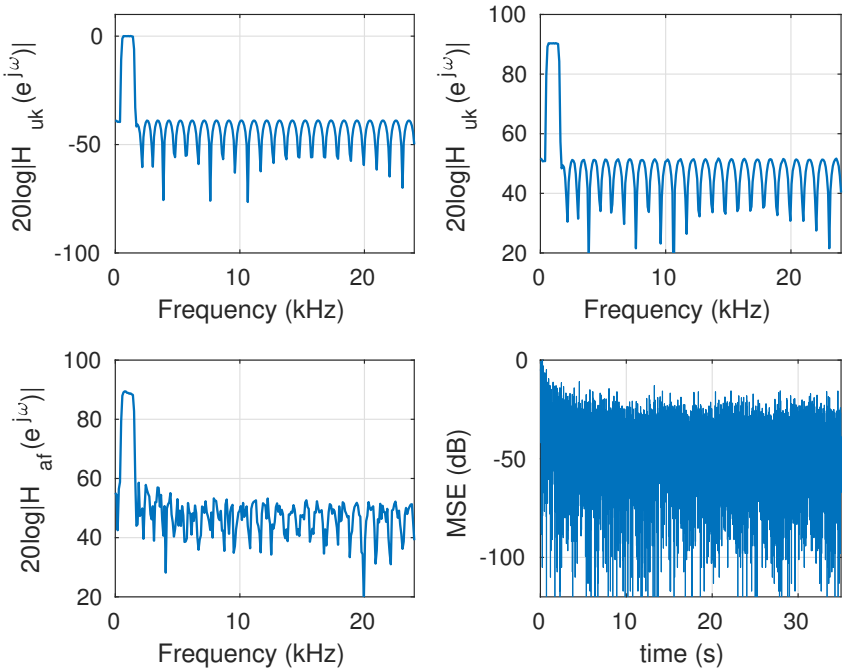

FIGURE 5.7: Fixed-point 16-bit implementation results using the NLMS adaptive filter to identify an 'unknown' passband filter with a stop band attenuation of 40 dB.

Another experiment is built to verify the performance of NLMS adaptive filter on UAV noise cancellation application. The setup for this experiment is shown in the Figure 3.4. A short period of music will be used as signal $s(k)$, and recorded propeller noise is used as noise $n(k)$ and $x(k)$. The output signal would be $e(k)$, if this filter has good performance, the output signal should sound quite similar with original music signal. Figure 5.9 shows the real-time comparison between *Signal* and *Error output*, channel 1 is the clear audio signal, channel 2 is the *Error output*, which is created from NLMS adaptive filter though input propeller noise and audio signal corrupted by propeller noise. As we can see, the two single we collected form different channels is almost same. It indicates that it is possible for UAV to collect real-time clear surrounding audio signal without the effects of propeller noise.

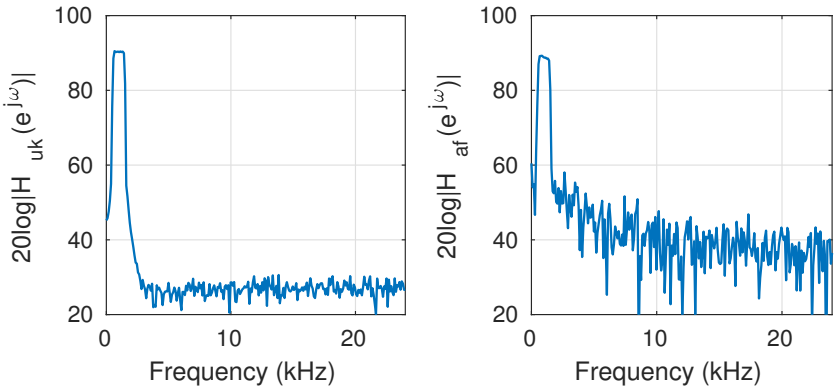

FIGURE 5.8: Fixed-point 16-bit implementation results using the NLMS adaptive filter to identify an 'unknown' passband filter with a stop band attenuation of 60 dB.

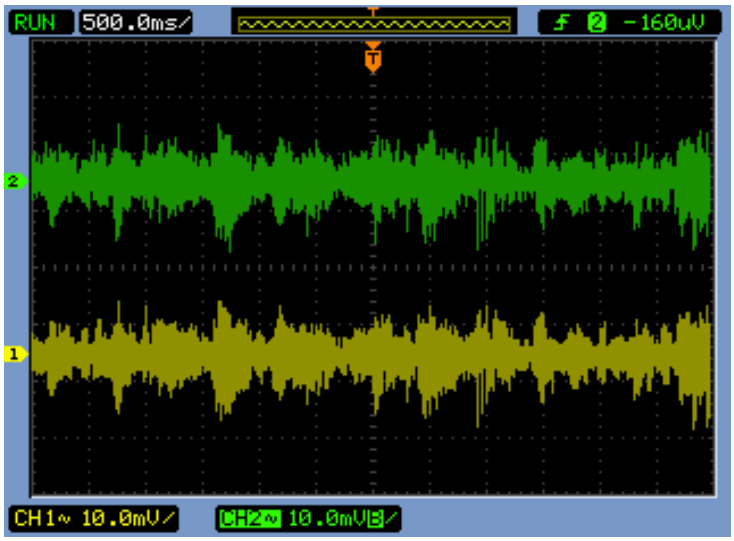

FIGURE 5.9: NLMS adaptive filter real-time comparison results.

Chapter 6

Conclusion and Future Work

During the research to explore digital signal processing applications on UAVs, a real-time digital signal processing hardware platform has been built on ZYBO. It includes ARM processor, SSM2603 codec chip and its controller with I2C and I2S protocols implemented on FPGA part of ZYNQ AP SoC. Then three routinely used digital filters, median filter, FIR filter, NLMS filter, are designed and simulated on Matlab with specific UAV applications respectively. To implement these filters on the real-time digital signal processing hardware platform, circular buffer and fixed-point arithmetic technologies are used in C language based software design. At last, A range of experiments aimed to verify the performance of these filters on real-time UAV audio signal processing applications is executed. As the chapter 5 discussed, the performance of real-time audio signal filters are almost equal to audio file based filters implemented on the general computer. These experiments conclude that this platform is qualified for airborne real-time audio signal processing applications on UAVs.

Though this platform can satisfy most of the basic needs for real-time signal processing requirements, there are still lots of improvements can be done to make it better. In this thesis, we just focus on the audio signal processing, but it might be used in a wide range of digital signal processing applications on UAVs, i.e., sensor data collection, communication data processing and compressing, etc. Since these filters are designed for general ARM processor, which is also the choice of most UAV controller, it is possible to integrate this design into existing autopilot solutions of UAVs or implement an autopilot on ZYBO containing these filters. With the application complexity increasing, this platform will face challenges on performance. Thanks to the powerful ZYNQ AP SoC on ZYBO, we have more options to choose. In this platform, an audio codec chip controller already has been implemented in FPGA successfully, so it is possible to implement the signal processing filters on FPGA. In fact, the FPGA-based hardware filter will achieve higher performance than software filter operating on the processor. For now, this platform can only provide processed audio signal locally on UAV, while

most of the situation the processed signal should be translated to ground control station (GCS), so the user can make decision according to real-time data. Digital data processing and UAV data transmission technologies are both critical in this situation. With the power of digital signal processing, more and more challenges on UAV based applications will be conquered in the near future.

Bibliography

Afonso, L. et al. (2016). "Cellular for the skies: Exploiting mobile network infrastructure for low altitude air-to-ground communications". In: *IEEE Aerospace and Electronic Systems Magazine* 31.8, pp. 4–11. ISSN: 0885-8985. DOI: 10.1109/MAES.2016.150170.

Borzino, A. M. C. R. et al. (2015). "Signal enhancement for gunshot DOA estimation with median filters". In: *2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS)*, pp. 1–4. DOI: 10.1109/LASCAS.2015.7250439.

Crockett, L.H. et al. (2014). *The Zynq Book: Embedded Processing Withe ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC*. Strathclyde Academic Media. ISBN: 9780992978709. URL: https://books.google.no/books?id=9dfvoAEACAAJ.

Devices, Analog (2013). *SSM2603: Low Power Audio Codec Data Sheet (Rev. C)*. original document from Analog Devices. URL: http://www.analog.com/media/en/technical-doc

Fernandes, Rigel Procópio et al. (2015). "A First Approach To Signal Enhancement For Quadcopters Using Piezoelectric Sensors". In: *Proceedings of the 20th International Conference on Transformative Science and Engineering, Business and Social Innovation*. Society For Design And Process Science. SDPS, pp. 536–541.

Furukawa, K. et al. (2013). "Noise correlation matrix estimation for improving sound source localization by multirotor UAV". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3943–3948. DOI: 10.1109/IROS.2013.6696920.

Gu, Y. et al. (2015). "Airborne WiFi networks through directional antennae: An experimental study". In: *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1314–1319. DOI: 10.1109/WCNC.2015.7127659.

Oppenheim, A.V. and R.W. Schafer (2011). *Discrete-Time Signal Processing*. Pearson Education. ISBN: 9780133002287. URL: https://books.google.no/books?id=EaMuAAAAQBAJ.

Qazi, S., A. S. Siddiqui, and A. I. Wagan (2015). "UAV based real time video surveillance over 4G LTE". In: *2015 International Conference on Open Source Systems Technologies (ICOSST)*, pp. 141–145. DOI: 10.1109/ICOSST.2015.7396417.

Ramos, António. L. L. et al. (2017). "Real-time implementations of acoustic signal enhancement techniques for aerial based surveillance and rescue applications". In: *Proc. SPIE, Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense X*. Vol. 8019, 80190U 1–9.

Santano, D. (2014). "Unmanned Aerial Vehicle in audio visual data acquisition; a researcher demo intro". In: *2014 International Conference on Virtual Systems Multimedia (VSMM)*, pp. 318–322. DOI: 10.1109/VSMM.2014.7136696.

Say, S. et al. (2016). "Priority-Based Data Gathering Framework in UAV-Assisted Wireless Sensor Networks". In: *IEEE Sensors Journal* 16.14, pp. 5785–5794. ISSN: 1530-437X. DOI: 10.1109/JSEN.2016.2568260.

Sayed, Ali H. (2003). *Fundamentals of Adaptive Filtering*. New Jersey: Wiley-IEEE Press.

Schaumont, P. (2010). *A Practical Introduction to Hardware/Software Codesign*. Springer US. ISBN: 9781441960009. URL: https://books.google.no/books?id=ngENR5O6fusC.

Shao, Zhili, António L. L. Ramos, and José Antonio Apolinário Jr. (2016). "data transmission strategies for UAV: an overview and future perspectives". In: *Proceedings of the 21th International Conference on Transformative Science and Engineering, Business and Social Innovation*. Society For Design And Process Science. SDPS, pp. 228–234.

Shi, J. et al. (2016). "Vision-based real-time 3D mapping for UAV with laser sensor". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4524–4529. DOI: 10.1109/IROS.2016.7759666.

Teich, J. (2012). "Hardware/Software Codesign: The Past, the Present, and Predicting the Future". In: *Proceedings of the IEEE* 100.Special Centennial Issue, pp. 1411–1430. ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2182009.

Varela, G. et al. (2011). "Swarm intelligence based approach for real time UAV team coordination in search operations". In: *2011 Third World Congress on Nature and Biologically Inspired Computing*, pp. 365–370. DOI: 10.1109/NaBIC.2011.6089619.

Wolfe, V. et al. (2015). "Detecting and locating cell phone signals from avalanche victims using unmanned aerial vehicles". In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 704–713. DOI: 10.1109/ICUAS.2015.7152353.

Wu, J. and G. Zhou (2006). "Real-Time UAV Video Processing for Quick- Response to Natural Disaster". In: *2006 IEEE International Symposium on Geoscience and Remote Sensing*, pp. 976–979. DOI: 10.1109/IGARSS.2006.251.

Yousef, N. R. and A. H. Sayed (2001). "A unified approach to the steady-state and tracking analyses of adaptive filters". In: *IEEE Transactions on Signal Processing* 49.2, pp. 314–324. ISSN: 1053-587X. DOI: 10.1109/78.902113.

Appendix A

Hardware Design