Helen and John C. Hartmann Department of Electrical and Computer Engineering

Senior Design Project Report

Semester: Fall 2020

Project Title: FPGA-based Active Noise Control System

Team #9 Members:

Prashant Baliga (COE)	ID#31401756
Brian Cheng (EE)	ID#31403624
Kevin Horng (EE)	ID#31405990
Michael Nichols (EE)	ID#31386620

Advisors:

Dr. Marek Sosnowski

Date of Submission:

16 December 2020

FPGA-based Active Noise Control System

Team Members:

Prashant Baliga (COE)

Brian Cheng (EE)

Kevin Horng (EE)

Michael Nichols (EE)

Advisors:

Dr. Marek Sosnowski

ABSTRACT

This paper presents an FPGA-based Active Noise Control (ANC) system for HVAC air-ducts. A typical feedforward filtered-x LMS ANC system is initially modelled using MATLAB Simulink, then modelled in VHDL using Xilinx Vivado. Custom RAM-based digital FIR filters and LMS algorithms are created in VHDL using parallel pipelined multiply-accumulation (MAC) to alleviate FPGA resource usage. The core ANC system is programmed onto a Zynq-7000 xc7z020-1clg400c FPGA and tested on a mockup air-duct made of PVC pipes. Noise attenuation measurements along with FPGA resource usage are reported. Possible system improvements are suggested.

TABLE OF CONTENTS

LIST OF FIGURES

1 - INTRODUCTION

1.1 - Project Significance and Objectives

Heating, ventilation, and air-conditioning systems are a crucial part to all buildings from small houses to large office spaces. However, HVAC systems often generate unpleasant noise pollution which could propagate throughout the building through internal air ducts. Passive noise control solutions like foam padding are effective in attenuating mid to high frequency noise, but are generally ineffective against the low frequency noises which HVAC machinery can typically generate. To address this issue, we investigate active noise control solutions that are capable of attenuating a broad band of frequencies. The core objective of our project is to design and demonstrate an FPGA-based ANC system to attenuate undesired noise in HVAC air-ducts. Although ANC systems have been made before, we sought to develop a cost-effective FPGA-based version by creating hardware-efficient digital filters in VHDL.

1.2 - Project Novelty

FPGA-based ANC prototypes have been designed many times before. However, a trend we have observed while researching existing ANC prototypes [1] [2] is that the multiply-accumulation, which is crucial in digital signal processing, is conducted in one of two ways: totally parallel or totally pipelined multiply-accumulation (MAC). To illustrate this idea, consider two different implementations of a 4th order FIR Filter as shown in **Figure 1** below.

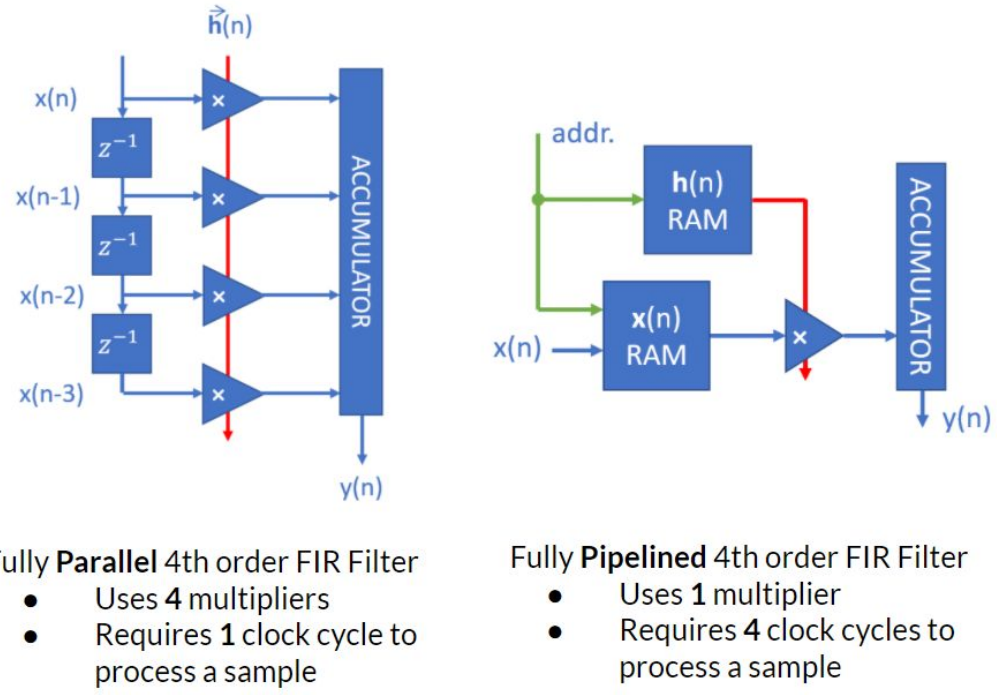


Figure 1. Examples of parallel and pipelined MAC

A fully parallel implementation uses 4 multipliers to process a sample on a single clock cycle. This approach uses more hardware but less processing time. On the other hand, a fully pipelined implementation uses 1 multiplier to process a sample over 4 clock cycles. This approach uses less

hardware but more processing time. Two constraints to keep in mind when programming FPGAs is that hardware resources are limited, and the performance of those resources are limited by clock speed and propagation delay. Another thing to note is that dedicated multipliers, which are implemented using DSP slices, are scarce compared to other resources on the FPGA.

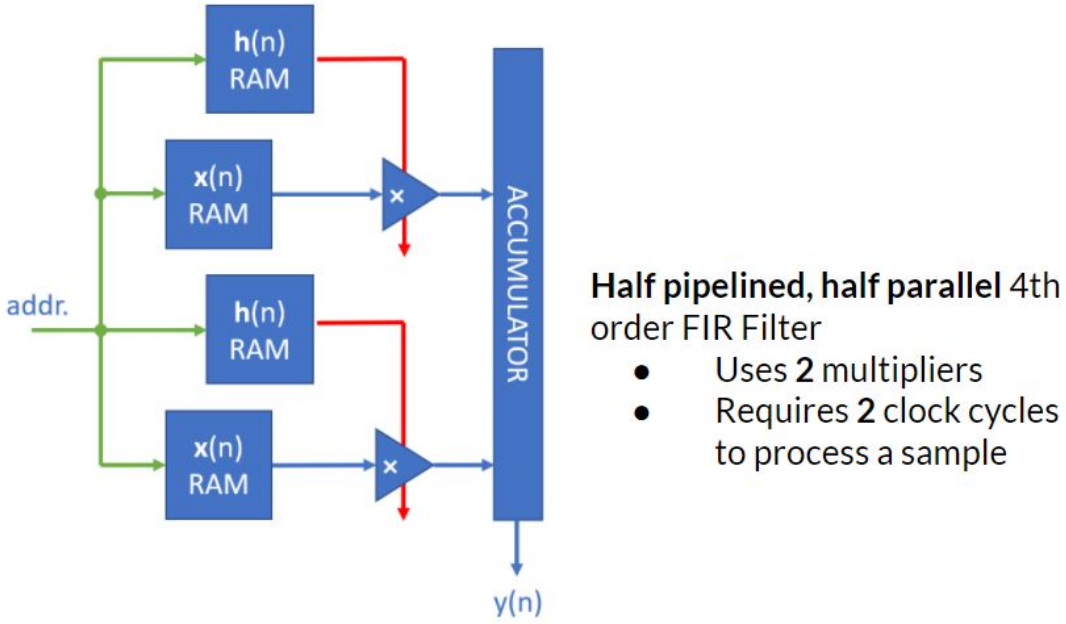


Figure 2. Example of parallel-pipeline MAC structure

Shown above in **Figure 2** is an implementation of a 4th order FIR filter using half-pipelined, half-parallel MAC structure, which uses 2 multipliers to process a sample over 2 clock cycles. Using this approach, we design VHDL components that strike a balance between pipeline and parallel MAC structure. We have also designed the components in such a way that the ratio between pipelined-ness and parallel-ness of the MAC structure can be reconfigured by modifying two generic parameters. This allows us to more effectively utilize the available resources and to operate within the hardware and timing constraints of the FPGA. Consequently, if a commercial FPGA-based ANC product were to be produced, the manufacturing cost could be reduced by using cheaper FPGAs.

2 - PROJECT ACCOMPLISHMENTS

2.1 - Technical Detail

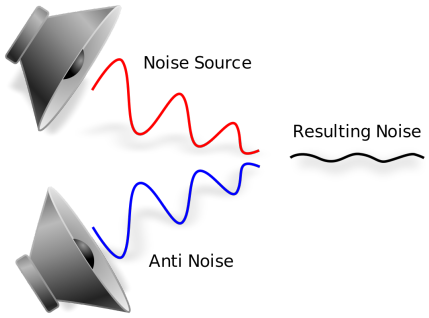


Figure 3. Illustration of Destructive interference between two waves

The basic principle of sound cancellation is predicated upon destructive interference. As depicted in **Figure 3**, two sound waves of equal amplitude but of opposite phase will sum up to zero when they interfere. So when a fan noise propagates through an air-duct, the main objective is to generate an equal but opposite noise through a loudspeaker to attenuate that noise. This objective has two challenges: to generate the anti-noise signal, and furthermore, to make the anti-noise signal adapt to an arbitrary noise source.

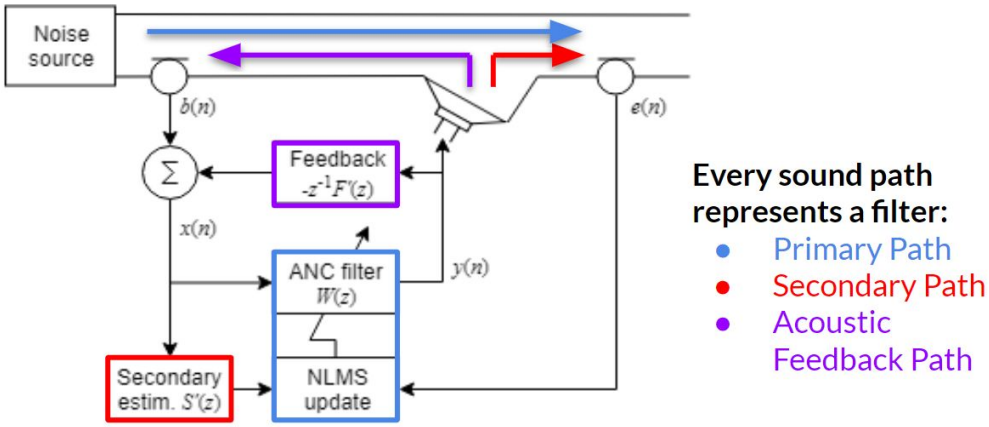


Figure 4. Block diagram of the Feedforward Filtered-x LMS ANC System

Our system follows a feedforward filtered-x LMS model that is typical of many active noise control systems. “Feedforward,” as opposed to “feedback,” refers to a control system that measures disturbances and compensates for them before they have time to affect the system. “LMS” refers to the well-known Least Mean Squares algorithm, while “Filtered-x LMS” refers to an LMS algorithm whose input is filtered by a secondary path [3]. **Figure 4** shows the functional block diagram in relation to a physical air-duct. A noise source propagates down the duct and is cancelled out by a loudspeaker. The noise source b(n) is recorded using a reference microphone positioned near the noise source, while the residual error e(n) is recorded using an error microphone at the air-duct’s exit.

There are three sound paths that must be considered: the primary path from the noise source to the error microphone, the secondary path from the loudspeaker to the error microphone, and finally, the acoustic feedback path from the loudspeaker back to the reference microphone. As the sound signals propagate through these paths, the air-duct acts as a filter that warps the sound. Thus, the noise signal as recorded by the reference microphone will not be the same signal after it propagates and reaches the air-duct's exit. Likewise, the anti-noise signal emitted by the loudspeaker will not be the same signal when it reaches the error microphone and the reference microphone. We cannot simply record the noise signal at the reference microphone, invert it, and play it back to the loudspeaker to achieve reliable sound attenuation. Rather, we must use adaptive algorithms, in this case, the LMS algorithm, to learn the filter coefficients which represent the air-duct's effect on each sound path. These coefficients are then fed into digital FIR filters to compensate for these effects.

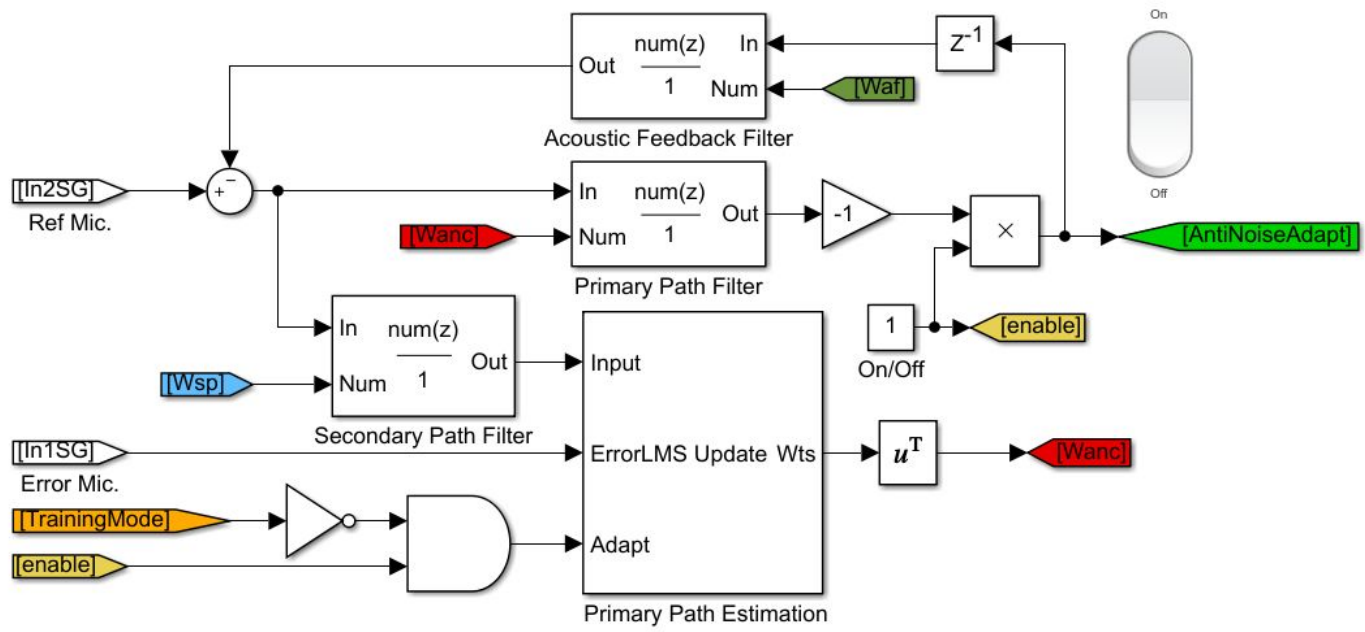


Figure 5 (above). Active noise control system (Simulink)

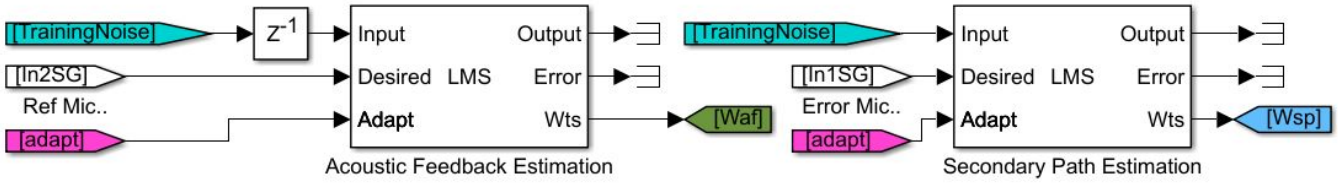


Figure 6. Secondary and feedback path weight estimation (Simulink)

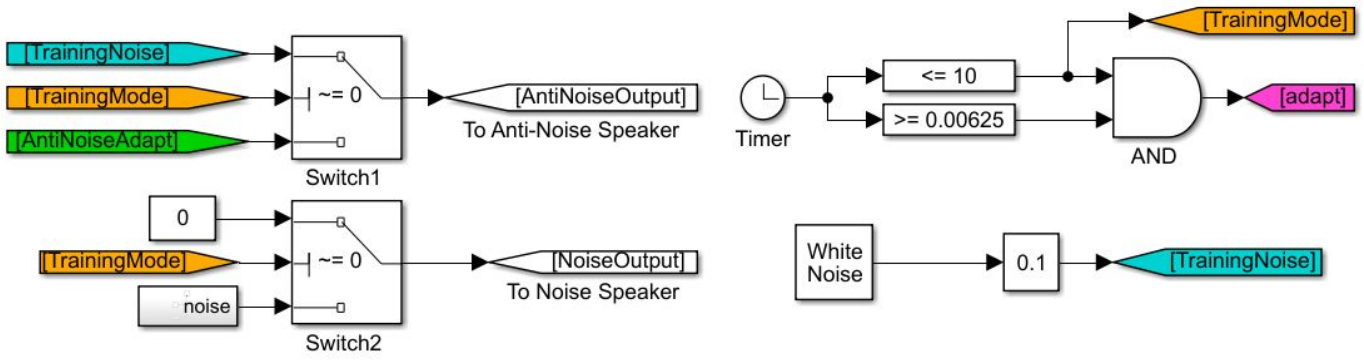


Figure 7. Signal routing, train/adapt sequence, training noise generator (Simulink)

Figures 5, 6, and 7, show the functional ANC System modelled in Simulink. It is closely modelled after MATLAB's ANC system described in their example project titled *Active Noise Control with Simulink Real-Time* [4]. When the system first initializes, it enters a ten second training period in which it estimates the filter coefficients for the secondary and acoustic feedback paths using the two LMS algorithm blocks shown in **Figure 6**. During the training period, a white noise signal is generated and fed to the "input" port of both LMS blocks. The white noise signal is also emitted from the loudspeaker and propagated down both the secondary and feedback paths to the respective microphones. The error microphone signal is fed to the "desired" port of the secondary path LMS block while the reference microphone signal is fed to the "desired" port of the acoustic feedback LMS block. The two LMS blocks calculate the filter coefficients for their respective paths based on the algorithm shown in **Figure 8.** After the training period ends, the LMS blocks are disabled and the filter coefficients are frozen and fed to the secondary and feedback FIR Filters. **Figure 7** shows the mechanism for routing signals during and after training period as well as the training sequence.

$$y(n) = \mathbf{w}^T(n-1)\mathbf{u}(n)$$
$$e(n) = d(n) - y(n)$$
$$\mathbf{w}(n) = \alpha\mathbf{w}(n-1) + f(\mathbf{u}(n), e(n), \mu)$$
$$\textbf{LMS: } f(\mathbf{u}(n), e(n), \mu) = \mu e(n)\mathbf{u}^*(n)$$

n	The current time index
$\mathbf{u}(n)$	The vector of buffered input samples at step n
$\mathbf{w}(n)$	The vector of filter weight estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
μ	The adaptation step size
α	The leakage factor ($0 < \alpha \leq 1$)

Figure 8. LMS Filter Algorithm and description of variables

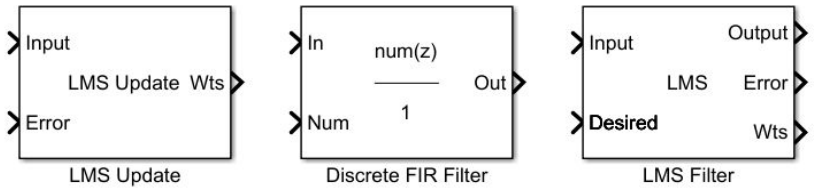


Figure 9. Relevant Simulink Blocks: LMS Update, FIR Filter, LMS Filter

Our original plan was to design the ANC system in Simulink, then use MATLAB's HDL Coder add-on to directly translate the relevant Simulink blocks shown in **Figure 9** into synthesizable VHDL components. However, a major limitation of HDL Coder is that it will only generate the relevant VHDL components using fully parallel multipliers. This means that if we were to use HDL Coder to generate VHDL for a 256th order FIR Filter, it will infer 256 multipliers operating in parallel. Since our FPGA has only 220 DSP slices, and because our design calls for multiple high-order filters operating in tandem, we cannot use the MATLAB-generated VHDL components. Thus, we create our own VHDL components for the FIR Filter, LMS Update, and LMS Filter using the methods described in the introduction in **Figure 2**. The VHDL code for the components as well as for the rest of the system can be found in our Github repository [5].

Our top level design shown in **Figure 10** is the connection between the core ANC system and the two analog-digital converters. The VHDL code driving the converters was mostly inherited from the manufacturer's code repository. **Figure 11** displays the exploded ANC System entity, which resembles the Simulink model in figures 5 to 7. **Figure 12** displays the exploded path entity which includes the filters, algorithms, and the block RAMs which store the buffered input vectors and the weight vectors. The bolded arrows which represent 24-bit signals while the thin arrows represent single bit values.

Figures 14, 15, and **16** show the finite state machines which govern the behavior of the FIR Filter, LMS Update algorithm, and LMS Filter algorithm. The main difference between the two algorithms is that the LMS Filter algorithm calculates the error signal using the input signal and the desired signal, while the LMS Update algorithm simply has a port for the error signal. The finite state machines update on the DSP clock which runs at 5.12MHz, while the ANC system updates on the ANC clock running at 10KHz, which is the sampling frequency of the analog-digital converters.

Each entity has two generic parameters that define its MAC structure: length (L) and width (W). The length represents the order of the filter, which is also the size of the input buffer. For example, in order to instantiate a 256th order FIR filter, L would have to be assigned the value 256. On the other hand, the width represents the number of multipliers operating in parallel. It also defines how many SPRAMs are required to store the input buffer and how many DPRAMs are required to store the weight vector. For example, the 4th order FIR filter in **Figure 2** would have had L=4 and W=2. Another constant each entity has is the ratio (R) between length and width, which is assigned the value L/W. The ratio defines how many words each RAM block must be able to store. **Figure 12** shows the memory structure between the filter and the LMS algorithm for when W=1, while **Figure 13** shows the expandable memory structure for when the width is assigned an arbitrary value W.

There are some constraints to be wary of when assigning these generics. First, L and W must be assigned such that R is a whole number. Second, if the sampling frequency is 10KHz and the DSP clock is 5.12MHz, then R must be less than 64 for the FIR Filter and the LMS Update. With the same clock frequencies, R must be less than 32 for the LMS Filter. This is given by the equations

R<T_clk_anc/(8*T_clk_dsp) for the FIR Filter and LMS Update, and **R<T_clk_anc/(16*T_clk_dsp)** for the LMS Filter. T_clk_anc and T_clk_dsp denote the time period of the respective clocks. Refer to **Figure 17** for a timing diagram of a FIR Filter with L=8 and W=2, clk_anc=10KHz, clk_dsp=5.12MHz. The DSP state machine is only active when clk_anc is low. If the ratio R exceeds these limitations, the state machine will not have enough DSP clocks in between the ANC clock highs to process the sample in time.

2.2 Testing and Performance Measurements

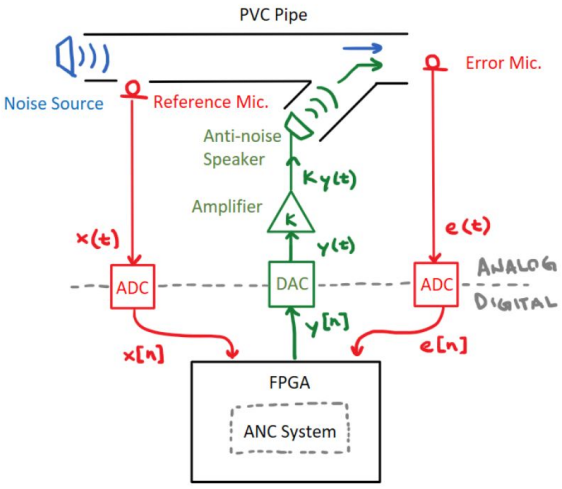


Figure 18. Hardware Overview

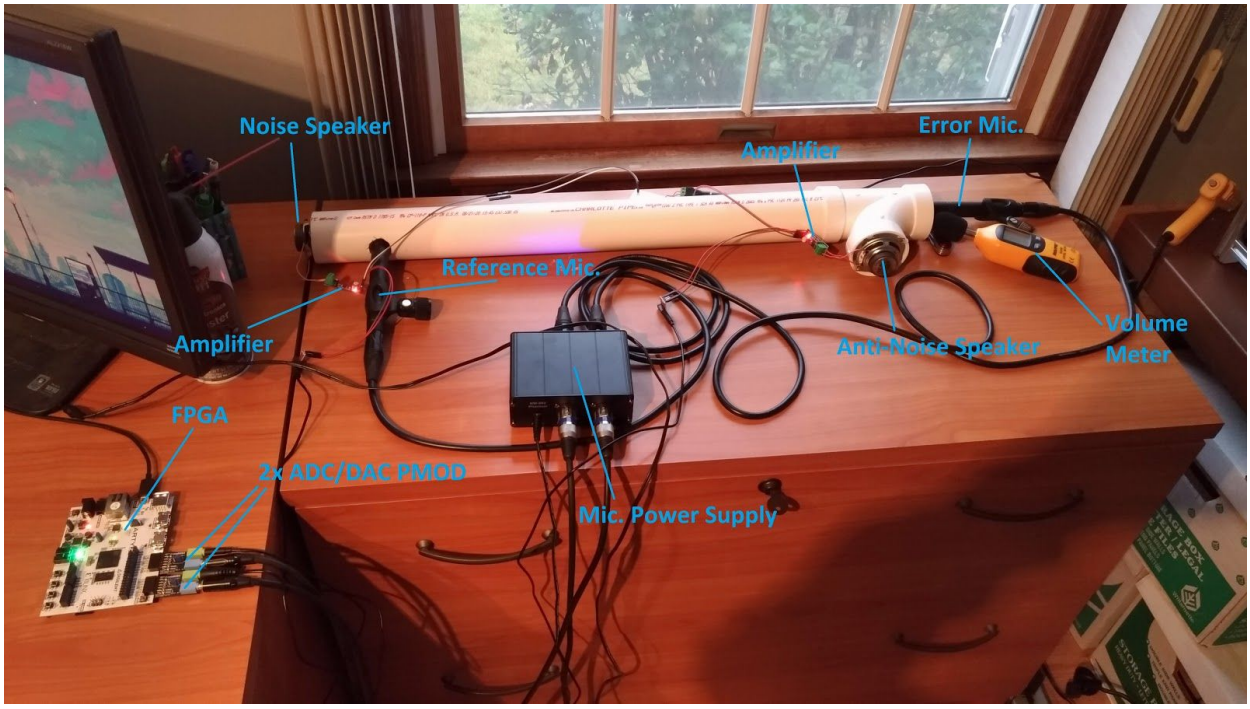


Figure 19. Physical Realization of Hardware

We tested our FPGA-based ANC system on a PVC pipe setup as shown in **Figure 17 and 18.** Against a 150Hz + 225Hz noise propagating down the PVC pipe, our system was able to reduce the noise recorded at the error microphone from 86dBA down to 67dBA, which is a 19dB of attenuation measured using a Risepro sound level meter. The ambient noise level was 45dBA. Further testing is required to contextualize this measurement by finding the band of frequencies for which the system is effective.

	xc7z020clg400-1	top_level	
Resource	**Max Available**	**Usage**	**Usage (% of max)**
LUT	53200	2064	3.9
FF	106400	1596	1.5
BRAM	140	32	22.9
DSP Slices	220	48	21.8

Figure 20. Vivado synthesis results (hardware usage)

Component	Length	Width	Count
LMS Update algorithm	384	16	1
FIR Filter	384	16	1
LMS Filter algorithm	128	8	2
FIR Filter	128	8	2

Figure 21. Components instantiated

We recorded the FPGA's hardware usage as reported by Vivado. Our design fits comfortably on the xc7z020clg400-1 FPGA, utilizing a low percentage of the maximum count for each resource. DSP Slices are used to implement every multiplication operation. BRAMs are block RAMs which store the input buffers and weight vectors. Flip flops (FF) or registers are crucial for implementing sequential circuits, while Lookup Tables (LUT) are responsible for implementing combinational circuits. If we had used the raw MATLAB-generated VHDL components for this design, it would fail implementation due to lack of resources on the FPGA.

2.3 Specifications

Specifications of our prototype:

- 1x PVC Pipe: 3-inch diameter, 48-inch length
    - 1-inch diameter hole positioned 3-inches away from PVC opening for the reference microphone
- 1x PVC T-junction: 3-inch diameter
- 2x Speakers: 4 Ohm, 3 Watt, 1.5-inch diaphragm
- 2x 48V Condenser Microphones
- 1x 2-input 2-output 48V Phantom Power Supply
- 2x Phantom to Phantom power cable
- 2x Phantom to 3.5mm cable

- 2x 3.5mm to 3.5mm cable
- 2x LM386 Amplifier set at 0.5 Gain
- 1x 5V power adapter for amplifiers
- 1x USB to micro-USB cable for FPGA
- 2x i2s2-PMOD modules for ADC and DAC voltage range -1V to +1V
- 1x Xilinx Arty Z7-20 FPGA (or any FPGA with the below specifications)
    - Lookup Tables: >5000
    - Flip Flops: >5000
    - Block RAMs: >50
    - DSP Slices: >50

2.4 Budget

Item	Price/Unit	Quantity	Total Cost w/Shipping
Xilinx FPGA (Arty Z7-20)	$200	1	$200
Condenser Microphones (2-pack)	$99	1	$106
Phantom Power Supply	$30	1	$30
Speakers (4-pack)	$13.00	1	$12.99
ADC/DAC (Digilent i2s2 PMODs)	$22.00	2	$53.97
3.5mm Audio Cable	$6.00	2	$24.56
Phantom to 3.5mm audio cable	$8.00	2	$16.00
Phantom to Phantom audo cable	$8.00	2	$16.00
3.5mm audio pin breakout board (2-pack)	$6.00	1	$5.99
LM386 amplifier board (10-pack)	$11.00	1	$10.99
			Total: $476.5

Figure 22. Prototype budget

2.5 Future Work

There is certainly much more work to be done with our project. Since we had only recently gotten a prototype to work, we have yet to collect enough data to describe the effectiveness or limitations of our system. We were able to achieve a decent attenuation for a low frequency two-tone noise source, but it would be worthwhile to find the band of frequencies for which our system is effective. We know that the theoretical upper limit is the sampling frequency which is 10KHz, however, aliasing effects could still manifest well within this limit. It could also be worthwhile to test our system with an actual fan at the opening of the PVC and introduce some air flow.

There is also more work to be done improving the hardware efficiency of our VHDL models. We could change the clk_anc duty cycle to free up more clocks for the finite state machines. We could also find a way to make the multipliers in the LMS Filter reusable during

and after the training period. Freeing up more resources can allow us to increase the order of our FIR filters to further reduce the ripple effect in their frequency response.

Systematic changes could also allow us to attenuate more than 15-20dB. It could be worthwhile to experiment with different step size or leakage factor in the LMS algorithm, or experiment with other algorithms like the Normalized LMS algorithm.

3. Engineering Standards, Risks, and Constraints

This project's objective was to design an FPGA-based active noise control system which could be applied to HVAC air-ducts. What we have achieved so far is a prototype which demonstrates the proof of concept. No plans have been made so far for designing a commercial product.

We could not identify formal engineering standards or constraints related to our project. We do know, however, that positive feedback of sound could potentially cause ear damage, and that a system failure could possibly result in positive feedback. Thus, if a commercial product is to be produced, the voltage levels of the speakers must be capped by either using an external circuit or by capping the digital signal within the FPGA.

Research on other ANC products fluctuate heavily in price, so the predicted product cost will be estimated based on cost of prototype. For our prototype, the total price amounted to $476 and estimating the addition of plastic casings to be an additional $10, the price to produce comes to $486. The most expensive components were the FPGA and the microphones, which were $200 and $106, respectively. It is very possible to use cheaper versions of these components. With some modifications, we could easily fit a working design on a $100 Artix-7 FPGA. We could also try swapping out the expensive 48V condenser microphones for cheap 5V dynamic microphones.

For retroactive installation, additional costs would derive from installation fees, if necessary. The price of these fees is hard to gauge at this time of writing, and it would be variable in the future depending on the client and use case. The installation of the product would also be relatively easy. Holes for both the reference microphone and the loud speaker would need to be drilled into the ventilation ducts. The rest of the installation would involve wiring management.

4. Learning Experience

This project was very much a "learn-as-we-go" experience. We started out with the general idea of doing active noise control because it unified multiple fields in electrical and computer engineering. This includes signals and systems, feedback control, and computer architecture. However, with limited experience in these fields and even less experience with actual hardware and software, we had to teach ourselves many skills to accomplish what we wanted. We had also vastly underestimated the scale of such a project and had constantly fought against the hardware constraints of the FPGA. While frustrating, this process helped us as students to understand the importance of efficient digital design from first-hand experience. This project has also taught us the importance of having an orderly design process, and that a few more hours spent in the planning stage could save weeks of troubleshooting down the line.

5. Acknowledgements

We would like to thank the NJIT faculty and staff for the feedback and advice provided while we worked on this project. We would like to thank Dr. Sosnowski for advising us and helping us to steer the direction of our project whenever managerial or technical problems would arise. Our group would also like to thank Professor Pramod Abichandani for advising us in the early stages of project brainstorming.

6. References

[1] W. H. Mahmoud, N. Zhang: *Software/Hardware Implementation of an Adaptive Noise Cancellation System*.
https://www.researchgate.net/publication/288443017_Softwarehardware_implementation_of_an_adaptive_noise_cancellation_system

[2] Wolfgang Fohl, Jorn Matthies, Bernd Schwarz: *A FPGA-BASED ADAPTIVE NOISE CANCELLING SYSTEM*.
https://www.semanticscholar.org/paper/A-FPGA-BASED-ADAPTIVE-NOISE-CANCELLING-SYSTEM-Fohl-Matthies/9e4616d9aee973c2d1c8a1b854b23d83b5589e61

[3] https://zone.ni.com/reference/en-XX/help/371988G-01/lvaftconcepts/aft_filteredx/

[4] MATLAB: *Active Noise Control with Simulink Real-Time.*
www.mathworks.com/help/audio/ug/active-noise-control-with-simulink.html.

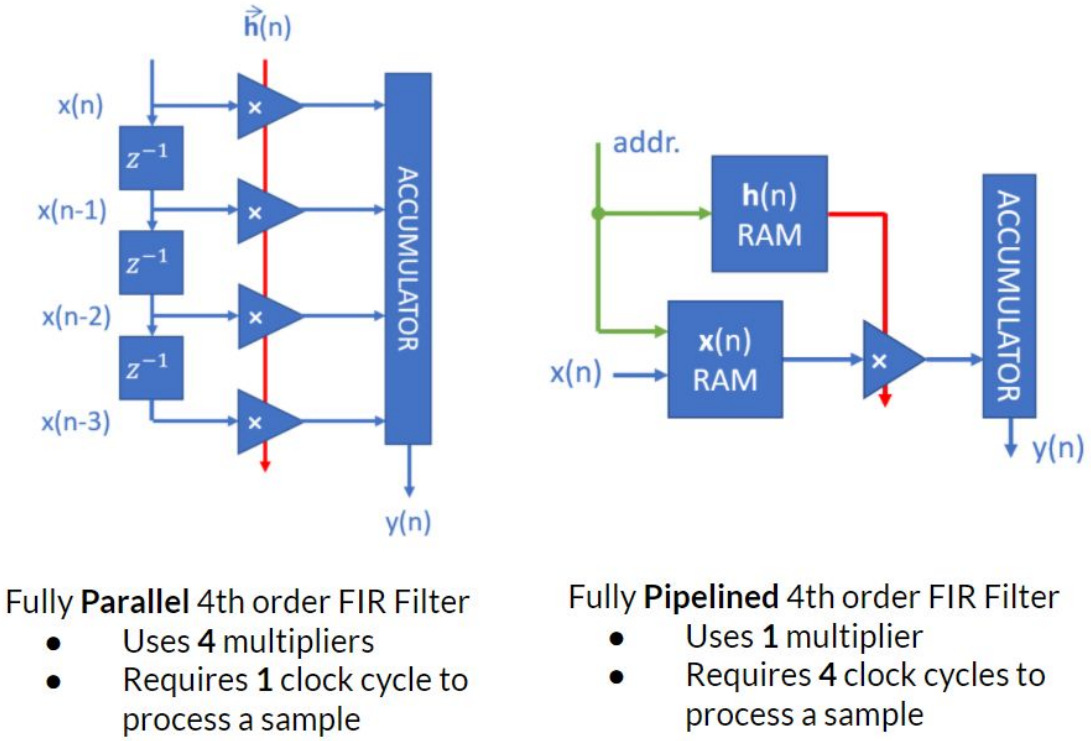
[5] https://github.com/TotoroTron/ANC_System

[6] DIGILENT: ADC/DAC i2s2 PMOD module documentation.
https://reference.digilentinc.com/reference/pmod/pmodi2s2/reference-manual

[7] DIGILENT: Arty Z7-20 FPGA Documentation.
https://reference.digilentinc.com/reference/programmable-logic/arty-z7/start

7. APPENDIX

Figure 1. Examples of parallel and pipelined MAC



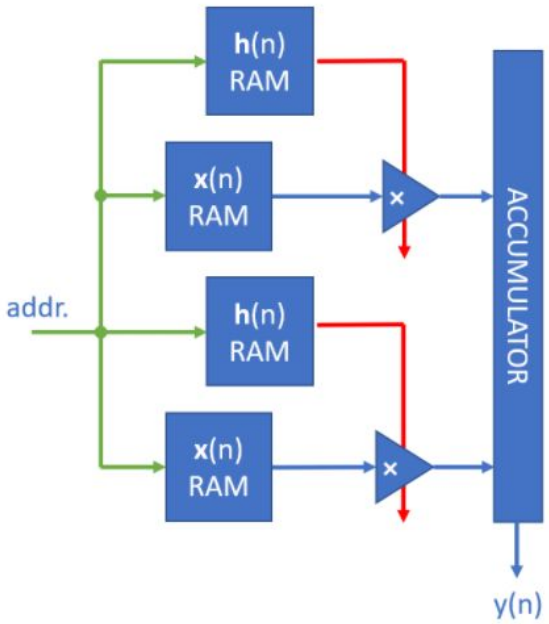
Fully **Parallel** 4th order FIR Filter

- Uses **4** multipliers
- Requires **1** clock cycle to process a sample

Fully **Pipelined** 4th order FIR Filter

- Uses **1** multiplier
- Requires **4** clock cycles to process a sample

Figure 2 (below). Example of parallel-pipeline MAC structure



Half pipelined, half parallel 4th order FIR Filter

- Uses **2** multipliers
- Requires **2** clock cycles to process a sample

Figure 3. Illustration of destructive interference between two waves

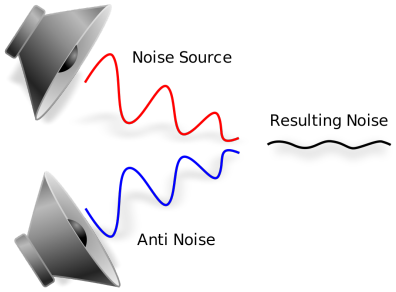


Figure 4 (below). Block diagram of the Feedforward Filtered-x LMS ANC System

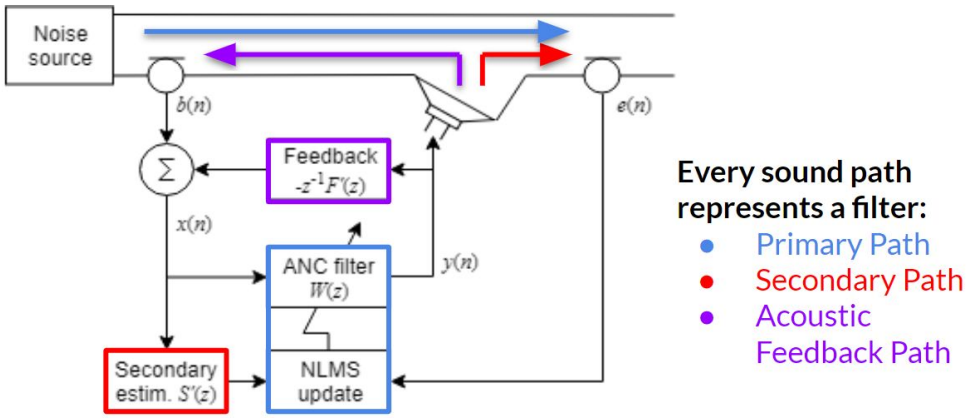


Figure 5 (below). Active noise control system (Simulink)

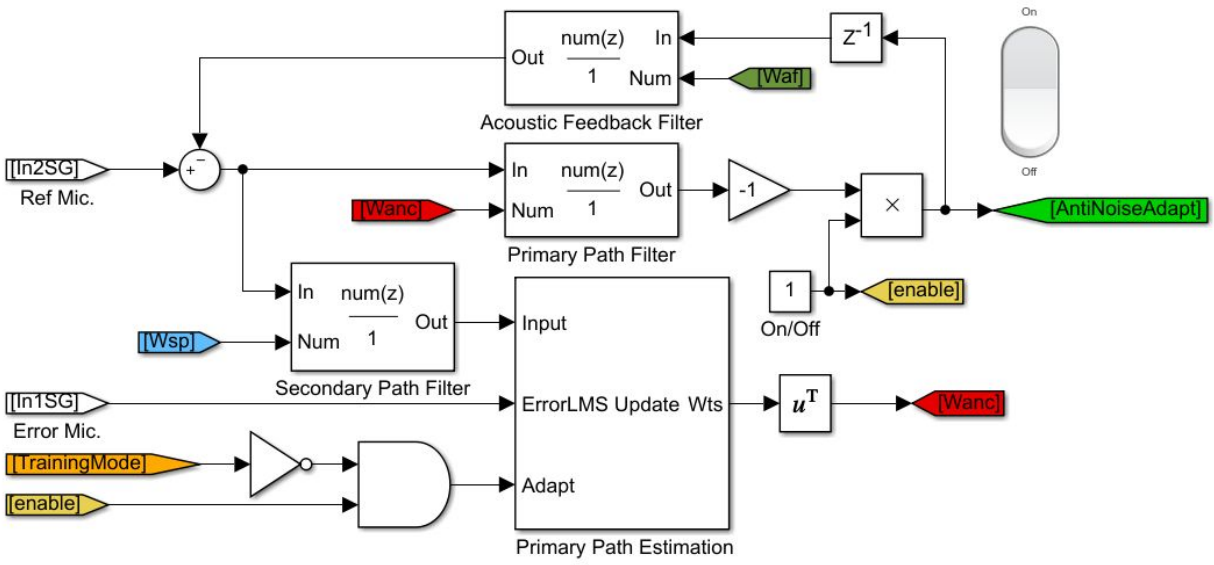


Figure 6. Secondary and feedback path weight estimation (Simulink)

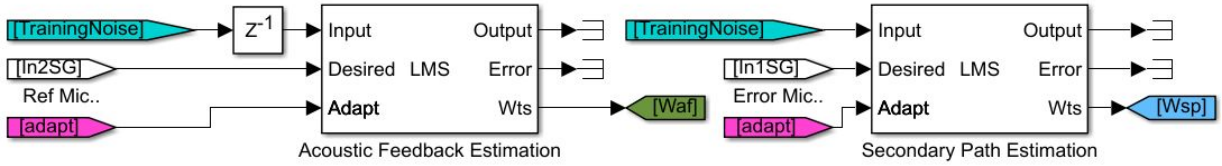


Figure 7 (below). Signal routing, train/adapt sequence, training noise generator (Simulink)

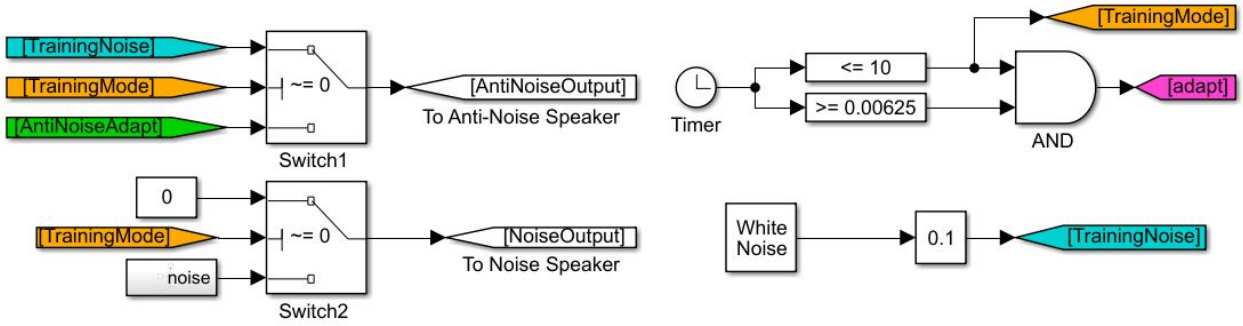


Figure 8. LMS filter algorithm and description of variables

$$y(n) = \mathbf{w}^T(n-1)\mathbf{u}(n)$$
$$e(n) = d(n) - y(n)$$
$$\mathbf{w}(n) = \alpha\mathbf{w}(n-1) + f(\mathbf{u}(n), e(n), \mu)$$
$$\textbf{LMS: } f(\mathbf{u}(n), e(n), \mu) = \mu e(n)\mathbf{u}^*(n)$$

n	The current time index
$\mathbf{u}(n)$	The vector of buffered input samples at step n
$\mathbf{w}(n)$	The vector of filter weight estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
μ	The adaptation step size
α	The leakage factor ($0 < \alpha \leq 1$)

Figure 9. Relevant Simulink Blocks: LMS Update, FIR Filter, LMS Filter

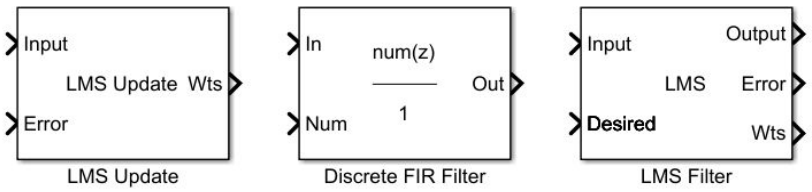


Figure 10. Circuit diagram of the top level entity

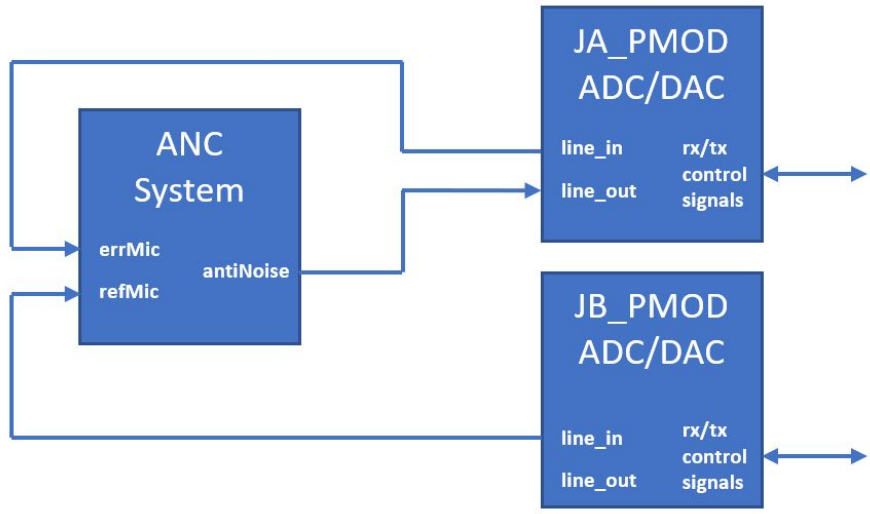


Figure 11. Circuit diagram of ANC_System entity

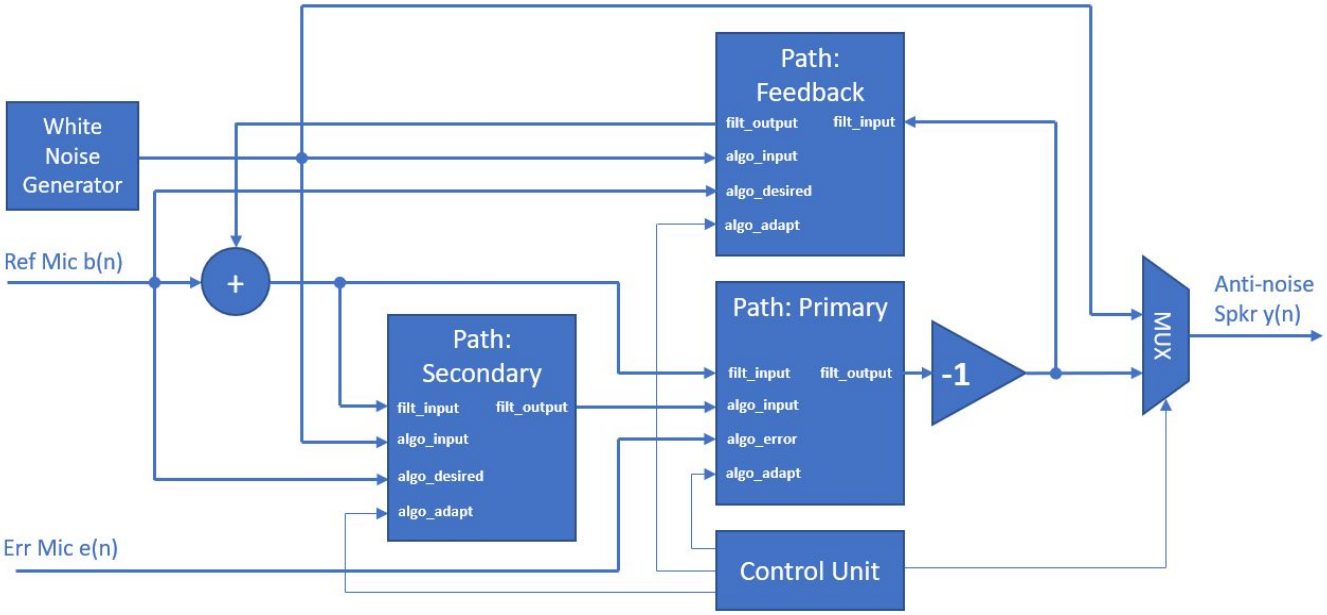


Figure 12. Circuit diagram of a path entity, Width = 1

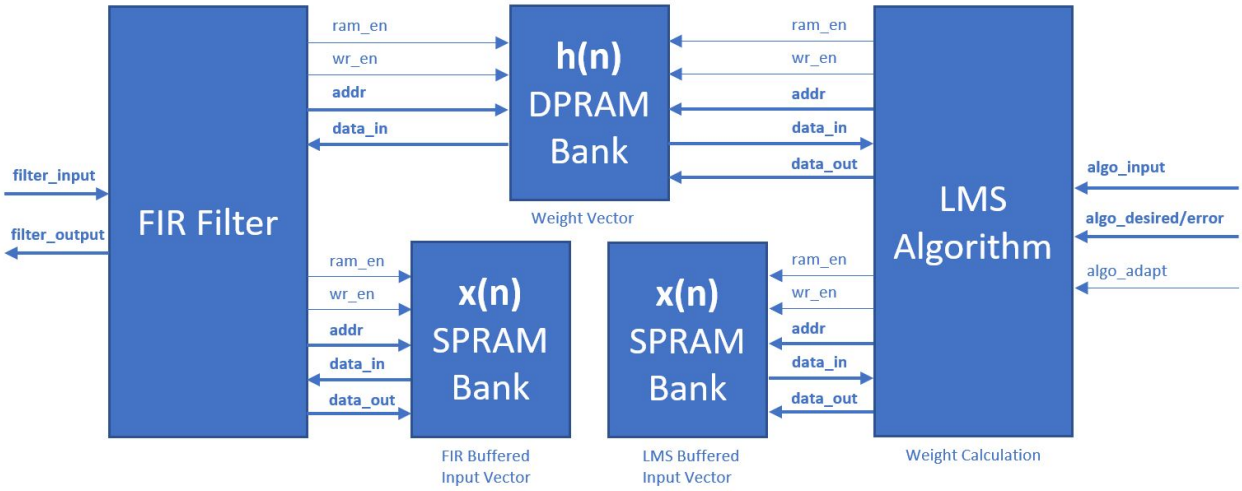


Figure 13 (below). Circuit diagram of path entity, arbitrary Width = W

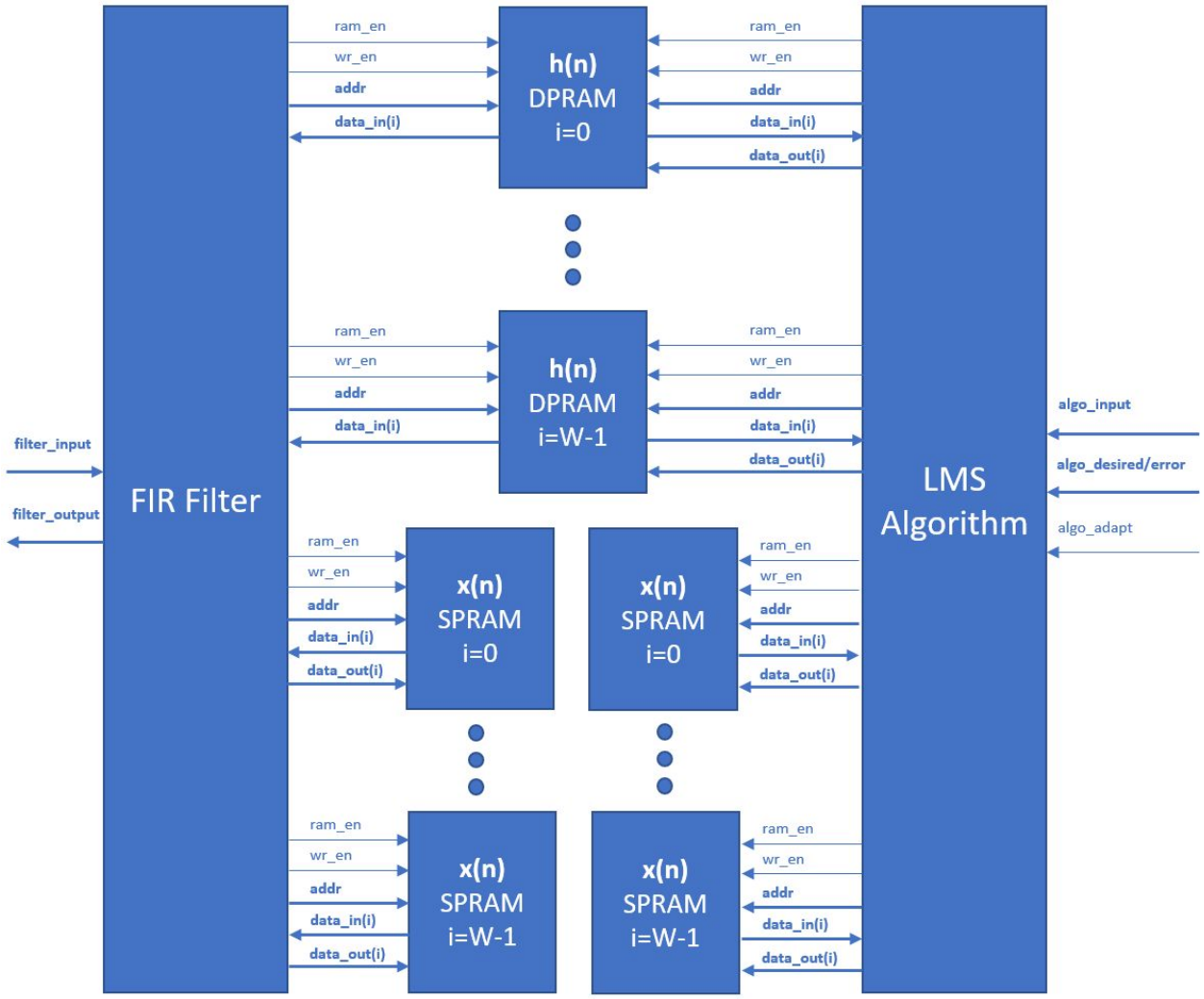


Figure 14. FIR Filter state machine diagram

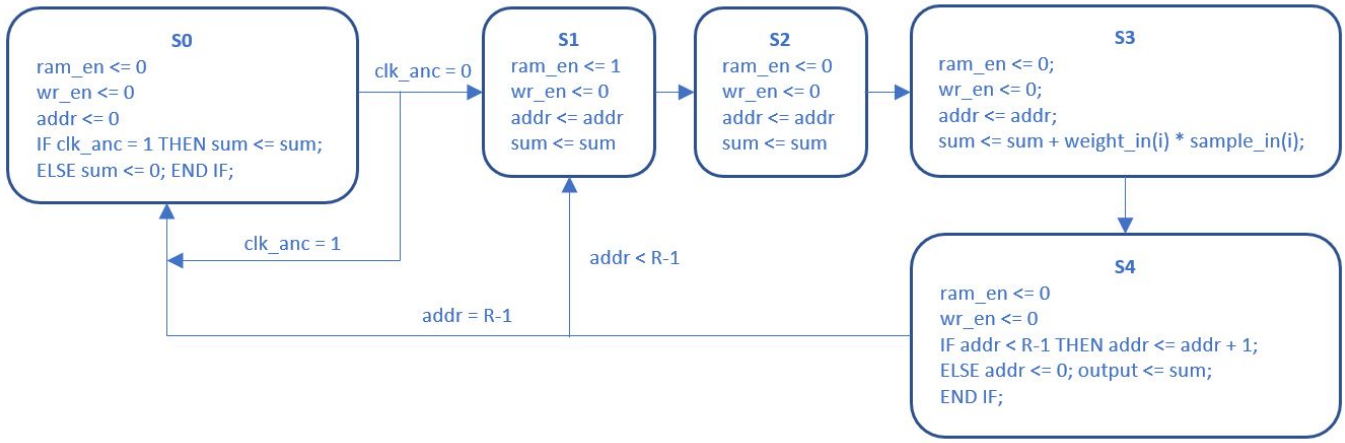


Figure 15. LMS Update algorithm state machine diagram

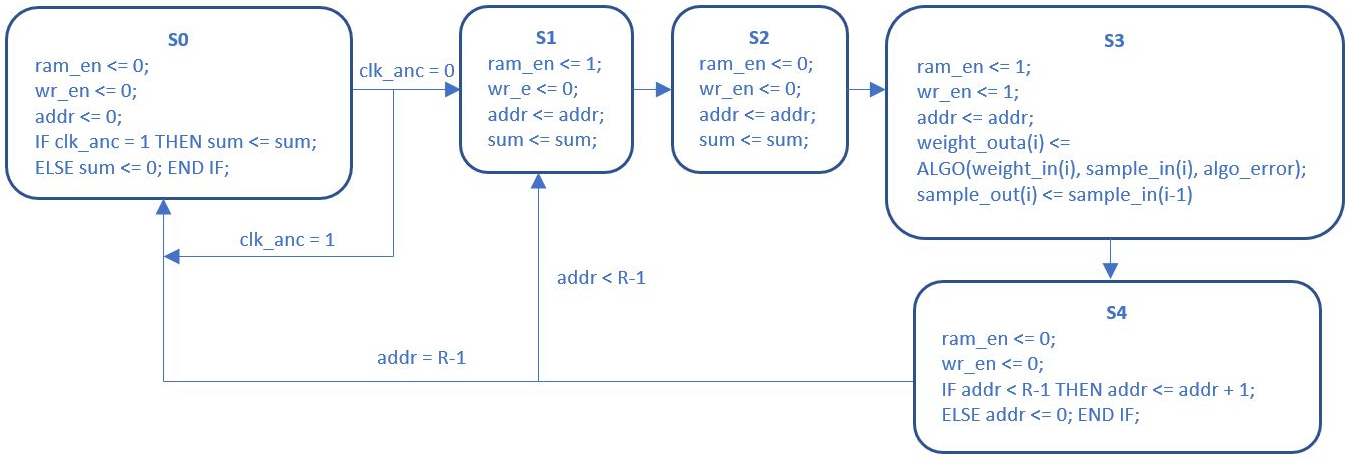


Figure 16. LMS Filter algorithm state machine diagram

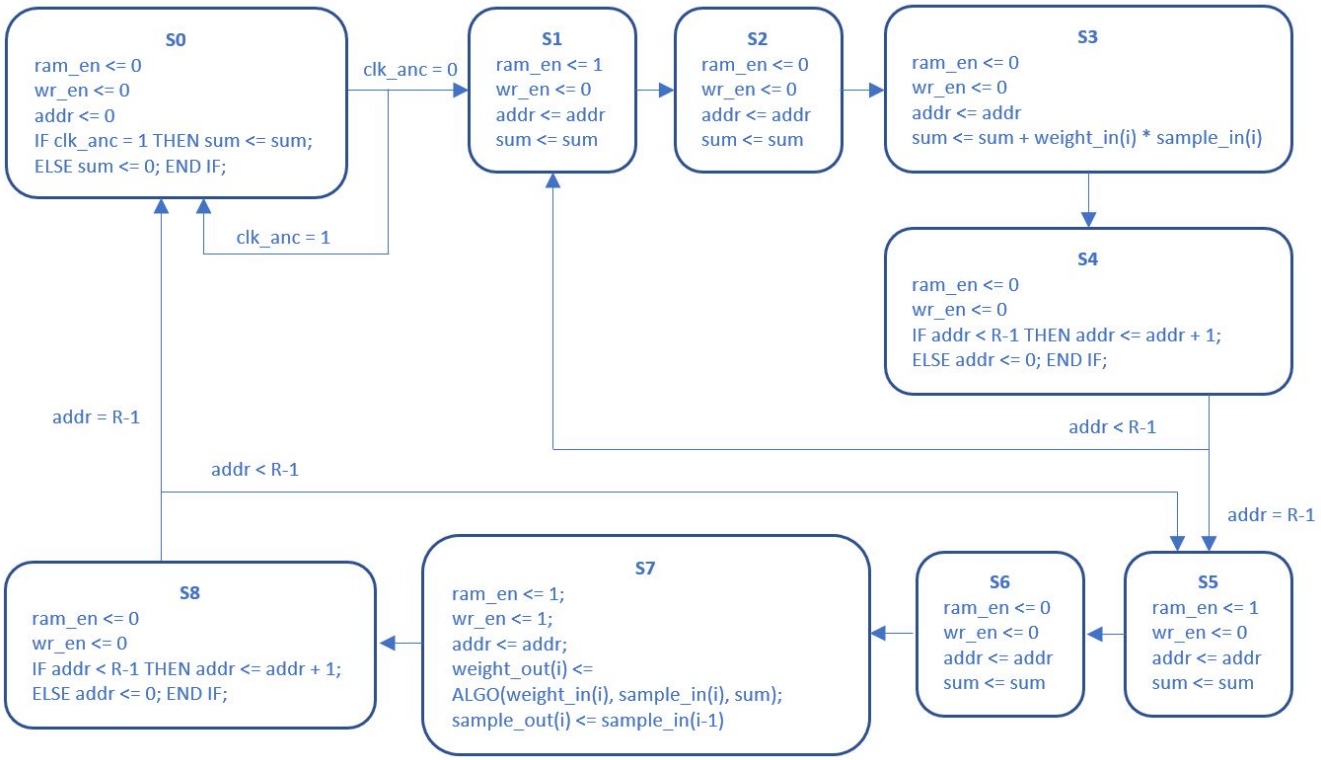


Figure 17. Timing diagram of FIR Filter L=8, W=2

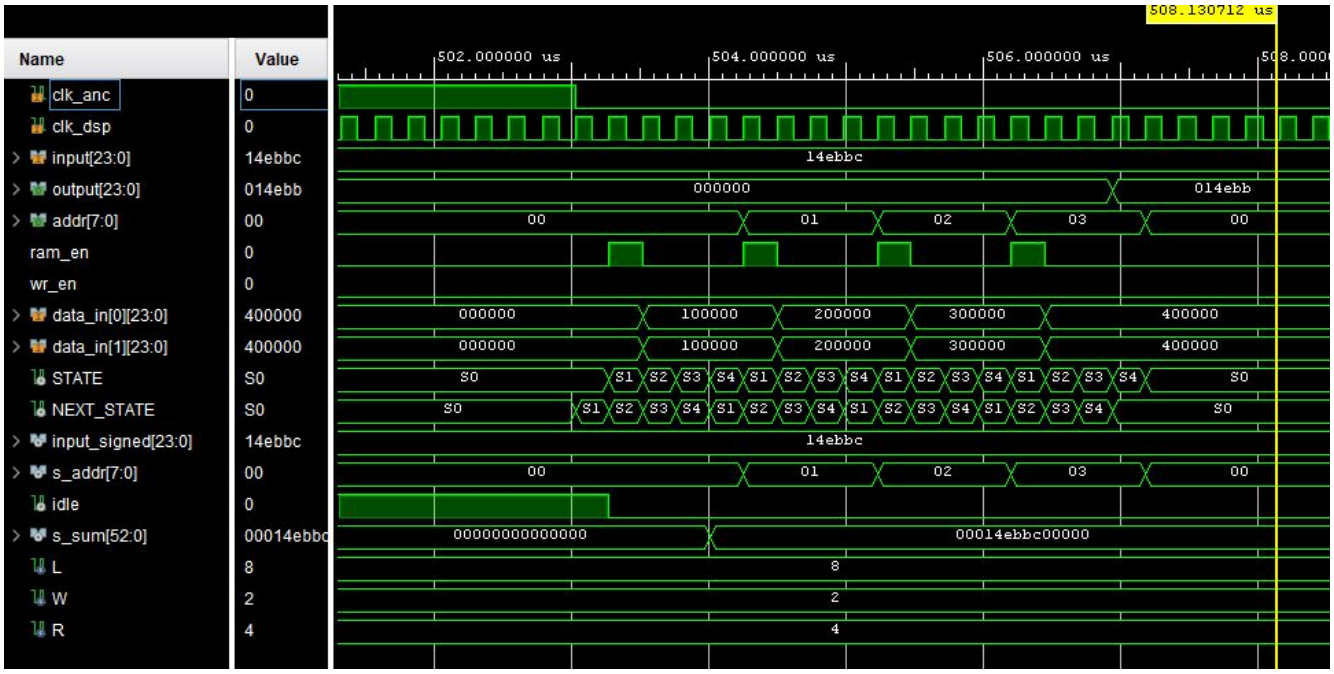


Figure 18. Hardware Overview Diagram

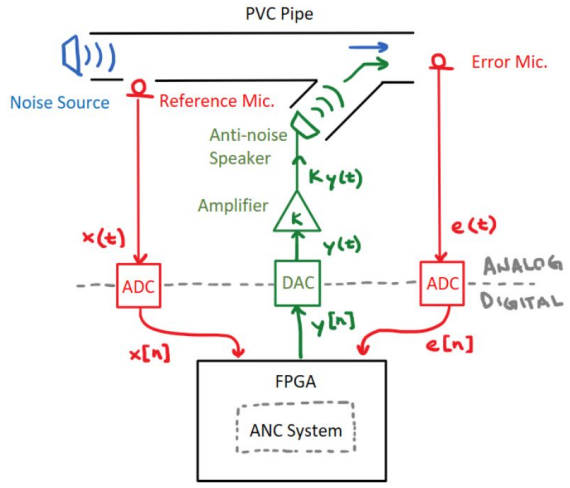


Figure 19. Physical Realization of Hardware

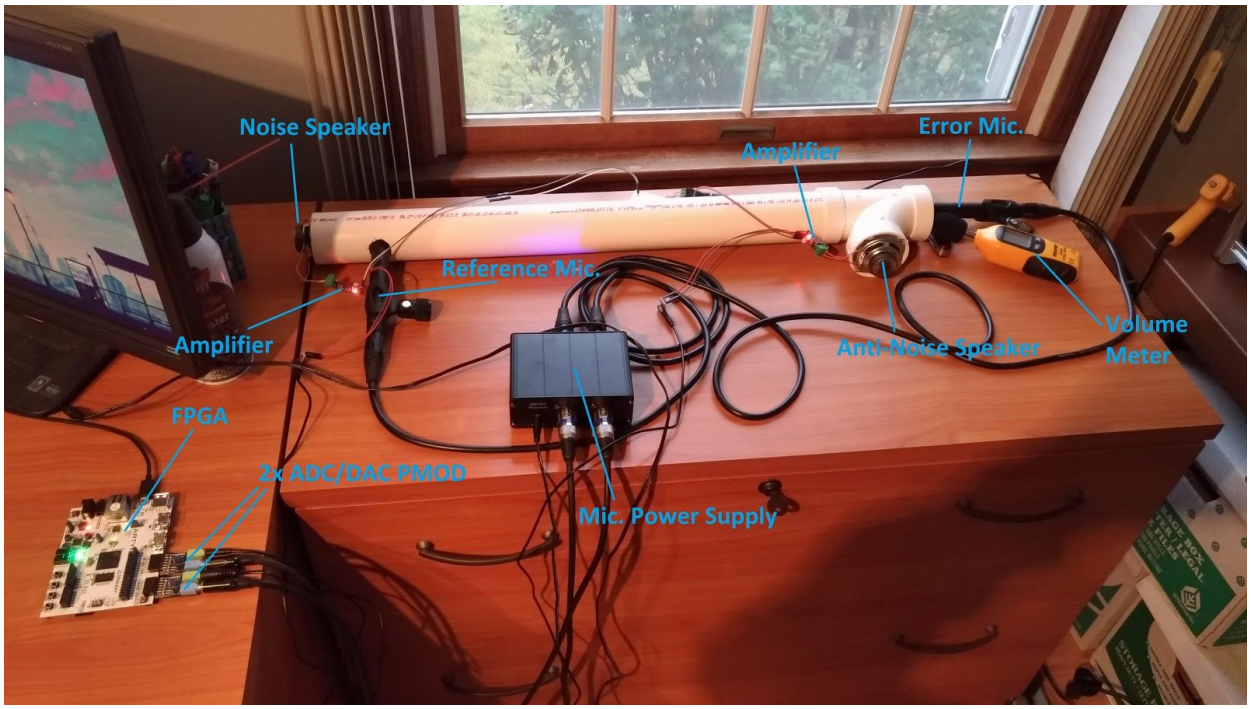


Figure 20. Vivado synthesis hardware usage results

	xc7z020clg400-1	top_level	
Resource	**Max Available**	**Usage**	**Usage (% of max)**
LUT	53200	2064	3.9
FF	106400	1596	1.5
BRAM	140	32	22.9
DSP Slices	220	48	21.8

Figure 21. Components instantiated during synthesis

Component	Length	Width	Count
LMS Update algorithm	384	16	1
FIR Filter	384	16	1
LMS Filter algorithm	128	8	2
FIR Filter	128	8	2

Figure 22. Prototype budget

Item	Price/Unit	Quantity	Total Cost w/Shipping
Xilinx FPGA (Arty Z7-20)	$200	1	$200
Condenser Microphones (2-pack)	$99	1	$106
Phantom Power Supply	$30	1	$30
Speakers (4-pack)	$13.00	1	$12.99
ADC/DAC (Digilent i2s2 PMODs)	$22.00	2	$53.97
3.5mm Audio Cable	$6.00	2	$24.56
Phantom to 3.5mm audio cable	$8.00	2	$16.00
Phantom to Phantom audo cable	$8.00	2	$16.00
3.5mm audio pin breakout board (2-pack)	$6.00	1	$5.99
LM386 amplifier board (10-pack)	$11.00	1	$10.99
			Total: $476.5