

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG



ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

**CẢI THIỆN CHẤT LƯỢNG TÍN HIỆU BẰNG BỘ
LỌC THÍCH ỨNG LMS THỰC HIỆN TRÊN FPGA**

GVHD: PGS. TS. HÀ HOÀNG KHA

SVTH: PHẠM VIỆT HÙNG

MSSV: 2113592

TP. HỒ CHÍ MINH, THÁNG 9 NĂM 2025

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

Độc lập – Tự do – Hạnh phúc.

-----☆-----

-----☆-----

Số: _____ /BK.ĐDT

Khoa: **Điện – Điện tử**

Bộ Môn: **VIỄN THÔNG**

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. HỌ VÀ TÊN: Phạm Việt Hùng

MSSV: 2113592

2. NGÀNH: Kỹ thuật Điện tử - Viễn thông

LỚP: DD21DV2

3. Đề tài: Cải thiện chất lượng tín hiệu bằng bộ lọc thích ứng LMS thực hiện trên FPGA

4. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):

- Tìm hiểu và thiết kế bộ lọc thích nghi theo giải thuật LMS.
- Thiết kế phần cứng bộ lọc thích ứng và thực hiện trên kit FPGA DE10.
- Thiết kế các thành phần để kiểm thử bộ lọc thích ứng trên KIT FPGA.
- Thực hiện kiểm thử trên phần cứng FPGA để đánh giá hiệu quả lọc và mức độ cải thiện chất lượng tín hiệu.

5. Ngày giao nhiệm vụ đồ án: 15/06/2025

6. Ngày hoàn thành nhiệm vụ: 19/09/2025

7. Họ và tên người hướng dẫn:

PGS. TS. HÀ HOÀNG KHA

Phần hướng dẫn

100%

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp.HCM, ngày..... tháng năm 2025

CHỦ NHIỆM BỘ MÔN

NGƯỜI HƯỚNG DẪN CHÍNH

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):

Đơn vị:

Ngày bảo vệ:

Điểm tổng kết:

Nơi lưu trữ đồ án:

LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến các thầy cô giảng viên của Khoa Điện- Điện tử, Trường Đại học Bách khoa TP. Hồ Chí Minh, đã tận tình hướng dẫn, giúp đỡ và truyền đạt cho em những kiến thức quý báu trong suốt quá trình học tập, nghiên cứu trong bốn năm đại học vừa qua của em.

Đặc biệt, em xin bày tỏ lòng biết ơn sâu sắc đến Thầy HÀ HOÀNG KHA, người đã đồng hành với em trong suốt quá trình thực hiện đồ án này. Thầy đã không ngừng động viên, khích lệ và chỉ bảo em những phương pháp khoa học, những kỹ năng thực hành và những kinh nghiệm quý giá. Nhờ có sự hướng dẫn tận tâm của thầy, em đã có thể hoàn thành đồ án này một cách tốt nhất, đồng thời em xin gửi lời cảm ơn đến thầy Nguyễn Thanh Tuấn giảng viên phản biện đã đưa ra cho em những nhận xét đúng đắn để em kịp thời hiệu chỉnh đồ án tốt nghiệp của em trở nên hoàn chỉnh hơn.

Em cũng xin gửi lời cảm ơn đến các bạn bè, đồng nghiệp và gia đình đã luôn ủng hộ, động viên và tạo điều kiện thuận lợi cho em trong suốt quá trình học tập và làm đồ án.

Em xin chịu trách nhiệm về mọi sai sót trong đồ án này và rất mong nhận được những ý kiến đóng góp quý báu của các thầy cô giáo và các bạn đọc.

Tp. Hồ Chí Minh, ngày 15 tháng 09 năm 2025.

Sinh viên

TÓM TẮT LUẬN VĂN

Đồ án tốt nghiệp này của em sẽ chủ yếu trình bày về thiết kế và kiểm thử một bộ lọc thích ứng (Adaptive filter) sử dụng giải thuật LEAST MEAN SQUARE cá về mô phỏng phần mềm và phần cứng. Các thành phần trong bộ lọc thích ứng của em sẽ sử dụng các bộ cộng trừ, nhân và chuyển đổi từ số 24bit nhị phân dạng bù hai sang dạng số chấm động (Floating point) theo chuẩn IEEE 754 để có thể nhân và cộng tích lũy với hệ số học (Learning rate) cho ra kết quả đúng nhất đến từng số phận phân. Để tìm hệ số học hợp lý nhất cho bộ lọc và tiết kiệm thời gian nhất, em dùng công cụ mô phỏng MATLAB để mô phỏng trước khi thực hiện thiết kế phần cứng để thực hiện trên KIT FPGA. Để kiểm thử hiệu quả lọc trên của bộ lọc đã thiết kế bằng ngôn ngữ phần cứng (Verilog HDL), em phải làm bộ giao tiếp (Interface) để có thể đưa ngõ ra của bộ lọc vào kiểm tra ở oscilloscope hoặc loa. Đồng thời em thiết kế thêm bộ tạo nhiễu ngõ vào có thể tạo và điều chỉnh các thông số nhiễu như biên độ và tần số để kiểm thử hiệu quả lọc khi nhiễu ở ngõ vào thay đổi.

Đồ án tốt nghiệp này của em mong muốn góp phần vào việc phát triển các giải pháp sử dụng FPGA để xử lý tín hiệu số hướng đến sự tùy biến linh hoạt nhờ FPGA để dễ dàng thay đổi và kiểm thử thiết kế. Đồ án này cũng hy vọng mang lại những kiến thức và kinh nghiệm bổ ích cho các sinh viên và nhà nghiên cứu quan tâm đến lĩnh vực FPGA, thiết kế số (Digital design) và xử lý số tín hiệu (DSP).

Mục Lục

LỜI CẢM ƠN	i
TÓM TẮT LUẬN VĂN.....	ii
DANH SÁCH HÌNH MINH HỌA	v
DANH SÁCH BẢNG SỐ LIỆU	ix
DANH MỤC CÁC KÝ HIỆU VÀ VIẾT TẮT	x
1. GIỚI THIỆU	1
1.1 Tổng quan	1
1.2 Tình hình nghiên cứu trong và ngoài nước	2
1.2.1. Tình hình nghiên cứu trong nước	2
1.2.2. Tại nước ngoài:	2
1.3 Nhiệm vụ đồ án.....	4
1.3.1. Tìm hiểu tính chất thiết và nhu cầu về sử dụng bộ lọc thích ứng.....	4
1.3.2. Phân tích yêu cầu.....	6
1.3.3. Nhiệm vụ của đồ án.....	8
2. LÝ THUYẾT	10
2.1. Tiền đề cho bộ lọc thích nghi	10
2.1.1. Đặt vấn đề.....	10
2.1.2. Bộ lọc FIR với hệ số cố định và hạn chế so với khi sử dụng hệ số thích ứng.....	11
2.2. Lý Thuyết Về Giải thuật LMS	14
2.2.1. Giải thuật LMS.....	14
2.2.2. Các thang đánh giá chất lượng bộ lọc	19
3. THỰC HIỆN MÔ PHỎNG MATLAB TÌM CÁC THÔNG SỐ HIỆU QUẢ NHẤT CHO BỘ LỌC	22
3.1. Định hướng.....	22
3.2. Tìm hệ số μ khi thực hiện bộ lọc thích nghi cho các thành phần sóng cơ bản.....	23
3.2.1. Tạo các thành phần sóng ngõ vào	23
3.2.2. Thực hiện bộ lọc LMS.....	25

3.2.3. Kết quả sau khi thực hiện lọc	27
3.3. Thực hiện mô phỏng tìm thông số bộ lọc thích nghi để lọc nhiễu âm thanh	30
3.3.1. Tạo GUI Matlab để mô phỏng	30
3.2.2. Thực hiện bộ lọc LMS với μ tốt nhất cho audio.....	33
4. THIẾT KẾ PHẦN CỨNG BỘ LỌC THÍCH ỨNG LMS.....	38
4.1. Định hướng.....	38
4.2. SỬ DỤNG BỘ CHUYỂN ĐỔI ADC VÀ DAC TRÊN KIT FPGA.....	40
4.2.1. Giới thiệu bộ Audio Codec WM8731 trên DE-10	40
4.2.2. Thực hiện hóa phần cứng cấu hình Audio_codec	42
4.3. Thiết kế bộ lọc thích ứng.....	53
4.3.1 Cấu trúc bộ lọc thích ứng	53
4.3.2 Ván đè phát sinh khi triển khai lên kit FPGA	54
4.3.3. Thiết kế và mô phỏng phần cứng bộ lọc	63
5. KẾT QUẢ THỰC HIỆN.....	69
5.1. Thực hiện lọc các thành phần sóng cơ bản:	69
5.2. Kết quả thực hiện lọc audio bằng bộ lọc thích nghi	75
5.3. So sánh hiệu quả bộ lọc khi thay đổi các thông số bộ lọc	77
6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	78
6.1. Kết luận	78
6.2. Hướng phát triển	79
TÀI LIỆU THAM KHẢO.....	80
PHỤ LỤC.....	82

DANH SÁCH HÌNH MINH HỌA

Hình 1-1: Mô hình khử tiếng vọng âm thanh	2
Hình 1-2: Ứng dụng của DSP trong y sinh	3
Hình 1-3: Sơ đồ khối bộ tách tạp âm từ cơ thể mẹ lẫn vào khi đo tim thai.....	3
Hình 1-4: Các nghiên cứu khoa học trên trang ieeexplore.ieee.org	4
Hình 2-1: Mic có bộ lọc thích ứng giúp định hướng và khử nhiễu âm thanh	10
Hình 2-2: Mô hình thêm microphone phụ để khử nhiễu cho microphone chính.....	10
Hình 2-3: Sơ đồ tín hiệu của Direct FIR filter.....	11
Hình 2-4: Các dạng đáp ứng bộ lọc phổ biến	12
Hình 2-5: Đáp ứng tần số bộ lọc FIR với hệ số cố định	13
Hình 2-6: Mô hình bộ lọc FIR thêm thành phần cập nhật hệ số bộ lọc.....	13
Hình 2-7: Ví dụ minh họa chứng minh giải thuật LMS	14
Hình 2-8: Mặt phẳng phương vị của hàm tối thiểu J.....	16
Hình 2-9: Mô phỏng mặt phẳng hàm mục tiêu J.....	17
Hình 3-1: Sơ đồ ngõ vào ngõ ra của bộ lọc thích nghi dựa trên FIR và LMS	23
Hình 3-2: Tín hiệu sạch ban đầu.....	24
Hình 3-3: Dạng sóng ngõ vào chính.....	24
Hình 3-4: Dạng sóng của ngõ vào thứ cấp.....	25
Hình 3-5: Độ biến thiên của SNR theo biến μ	26
Hình 3-6: MSE giữa $s[n]$ và $e[n]$ khi $0.002 < \mu < 0.0045$	26
Hình 3-7: MSE giữa $s[n]$ và $e[n]$ khi $0.005 < \mu < 0.009$	27
Hình 3-8: Dạng sóng ước lượng $e[n]$ sau lọc.....	27
Hình 3-9: FFT ngõ vào gốc	28
Hình 3-10: FFT ngõ vào chính	28
Hình 3-11: FFT ngõ vào thứ cấp	28
Hình 3-12: FFT Ngõ ra ước lượng $s[n]$	28
Hình 3-13: Dạng sóng của ước lượng nhiễu.....	29

Hình 3-14: Đồ thị biến thiên hệ số bộ lọc.....	30
Hình 3-15: Gui để thao tác với các file âm thanh.....	30
Hình 3-16: Hàm thay đổi SNR theo μ của audio.....	33
Hình 3-17: MSE giữ $s[n]$ và $e[n]$ khi $\mu > 0.001$	33
Hình 3-18: MSE với $\mu > 0.004$	34
Hình 3-19: Dạng sóng âm thanh gốc	35
Hình 3-20: Dạng sóng đầu vào chính.....	35
Hình 3-21: Dạng sóng ngõ ra ước lượng tín hiệu.....	35
Hình 3-22: Phổ của âm thanh gốc	36
Hình 3-23: Đầu vào chính	36
Hình 3-24: Đầu vào thứ cấp	36
Hình 3-25: Phổ của tín hiệu sau lọc.....	37
Hình 3-26: FFT thành phần ước lượng nhiễu	37
Hình 3-27: Độ biến thiên Weight	37
Hình 4-1: Kết nối line in vào laptop và line out ra loa	38
Hình 4-2: Kết nối FPGA với oscilloscope.....	39
Hình 4-3: Block diagram của chip Codec (gồm cả ADC DAC)	40
Hình 4-4: Hạn chế của bộ Codec WM8731	41
Hình 4-5: Waveform giao thức I2C.....	42
Hình 4-6: Các i/o I2C và địa chỉ I2C của WM8731	43
Hình 4-7: Bộ chia tần số	43
Hình 4-8: Sơ đồ chuyển trạng thái của Module lái giao thức I2C	45
Hình 4-9: Testbench điều kiện bắt đầu	46
Hình 4-10: Testbench Module I2C.....	46
Hình 4-11: Testbench điều kiện kết thúc	47
Hình 4-12: Testbench truyền I2C.....	47
Hình 4-13: Testbench truyền cấu hình bằng I2C.....	50

Hình 4-14: Chế độ hiệu chỉnh phải lấy mẫu ADC	51
Hình 4-15: Module thanh ghi dịch từ song song sang nối tiếp	51
Hình 4-16: Kết nối các ngõ vào và ngõ ra data ra ADC và DAC trên WM8731.....	52
Hình 4-17: Nối pin ra các chân của DE10	52
Hình 4-18: kết nối thành công với FPGA	52
Hình 4-19: Sơ đồ hoàn chỉnh bộ Adaptive filter.....	53
Hình 4-20: Interface giữa bộ lọc thích nghi và ADC DAC.....	54
Hình 4-21: Chuyển đổi các dạng số sang fp32	55
Hình 4-22: Giải thuật cộng số floating point	56
Hình 4-23: Testbench bộ cộng FP32.....	57
Hình 4-24: Giải thuật nhân số floating point.....	58
Hình 4-25: Testbench các testcase của bộ nhân	59
Hình 4-26: Giải thuật chuyển đổi số S24 sang dạng số FP32	60
Hình 4-27: Testbench chuyển đổi từ số bù 2 sang floating point	61
Hình 4-28: Testbench chuyển đổi từ floating point sang bù 2 24 bit.....	62
Hình 4-29: Block diagram của module Adaptive filter	63
Hình 4-30: Thiết kế thành phần tạo nhiễu cho ngõ vào của bộ lọc thích nghi	64
Hình 4-31: Tương quan giữa CLK của FPGA và tần số mong muốn tạo ra.....	65
Hình 4-32: Module thiết kế bộ điều chỉnh tần số	66
Hình 4-33: Mô phỏng tăng tần số nhiễu	67
Hình 4-34: Các thành phần sóng ở ngõ vào	68
Hình 4-35: Thành phần ngõ vào chính và thành phần ngõ vào thứ cấp	68
Hình 4-36: $e[n]$ và $s[n]$	68
Hình 4-37: Dạng sóng thay đổi của hệ số bộ lọc	68
Hình 5-1: Dạng sóng sóng gốc 1.54Vpp.....	69
Hình 5-2: Dạng sóng của primary input 3.54 Vpp	70
Hình 5-3: Dạng sóng của reference signal 3.42 Vpp	70

Hình 5-4: FFT của sóng sine 1KHz gốc	71
Hình 5-5: Dạng sóng ước lượng tín hiệu gốc.....	71
Hình 5-6: FFT của ngõ vào chính -15,8dB.....	72
Hình 5-7: FFT của ngõ ra ước lượng.....	72
Hình 5-8: Tín hiệu có ích.....	73
Hình 5-9: Dạng sóng của ngõ vào chính	73
Hình 5-10: FFT của ngõ vào chính	74
Hình 5-11: FFT của tín hiệu sau lọc	74
Hình 5-12: Kết quả file lọc âm thanh.....	75
Hình 5-13: Spectrogram của âm thanh từ Audacity	75
Hình 5-14: Ước lượng nhiễu được mô phỏng ở tool với điều kiện lý tưởng.....	76
Hình 5-15: Các switch và nút bấm để tương tác với bộ lọc	76
Hình 5-16: Kết quả và thời gian tổng hợp mạch với số tap là 15	77
Hình 5-17: kết quả và thời gian tổng hợp mạch với số tap là 32	77

DANH SÁCH BẢNG SỐ LIỆU

Bảng 2-1: Thang đánh giá DCR	19
Bảng 2-2: Sự khác biệt giữa MSE và ME.....	19
Bảng 4-1: Các chân quan trọng của WM8731.....	41
Bảng 4-2: Bảng các i/o của module giao thức I2C.....	44
Bảng 4-3: Bảng cấu hình thanh ghi.....	48
Bảng 4-4: I/O Ports module cấu hình thanh ghi.....	48
Bảng 4-5: Bảng điều chỉnh tần số.....	66
Bảng 5-1: Bảng các switch và nút bấm tương tác với FPGA.....	76
Bảng 5-2: So sánh hiệu quả khi thay đổi thông số bộ lọc	77

DANH MỤC CÁC KÝ HIỆU VÀ VIẾT TẮT

STT	Ký hiệu, viết tắt	Ý nghĩa, chữ viết đầy đủ
1	LMS	Least mean squares
2	μ (Miu)	Hệ số học (learning rate)
3	$d(n)$	Ngõ vào chính
4	$x(n)$	Ngõ vào thứ cấp
5	$e(n), \hat{s_k}$	Ước lượng tín hiệu gốc
6	$y(n), \hat{n_k}$	Ước nhiễu trộn vào ngõ vào chính
7	Adaptive filter	Bộ lọc thích ứng
8	FIR filter	Finite impulse response (Đáp ứng xung hữu hạn)
9	Weight, Coefficient	Hệ số bộ lọc
10	MSE	Mean squared error
11	FFT	Fast Fourier transform

1. GIỚI THIỆU

1.1 Tổng quan

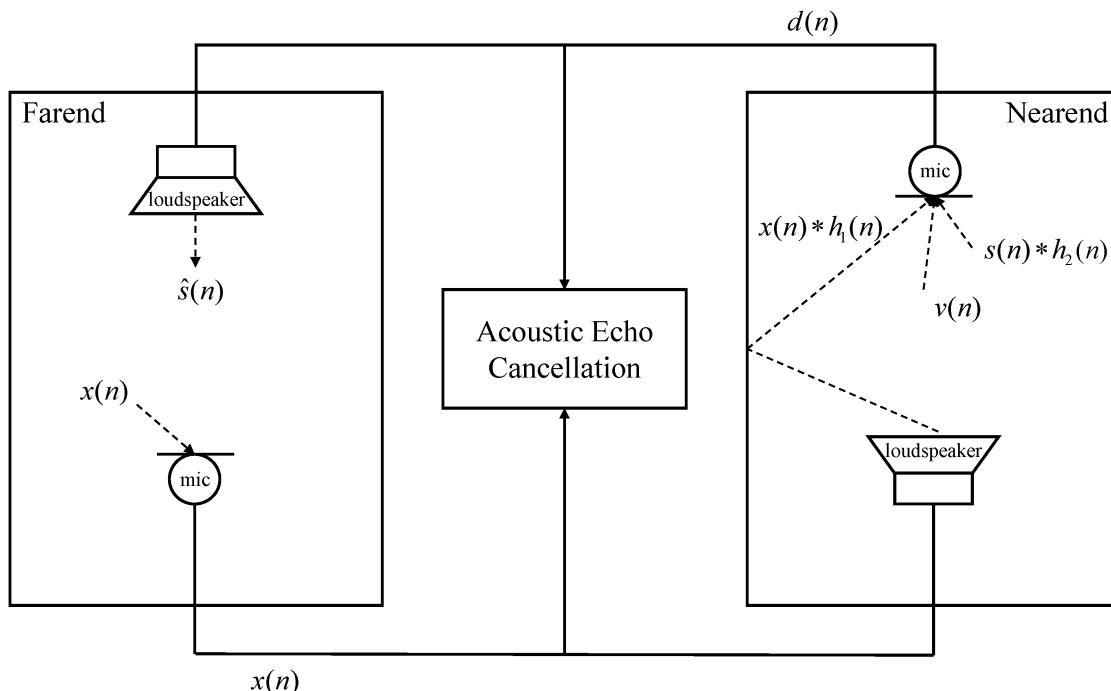
Trong bối cảnh chuyển đổi số sâu rộng hiện nay, xử lý tín hiệu số (Digital Signal Processing -DSP) trở thành nền tảng hạ tầng cho nhiều lĩnh vực trọng yếu như viễn thông, âm thanh đa phương tiện, y sinh, radar, sonar, công nghiệp ô-tô, năng lượng và các hệ thống nhúng/IoT. Một trong những kiến thức nền tảng của DSP là bộ lọc số (digital filter), công cụ dùng để định hình phổ, loại bỏ nhiễu, khử vọng, cân bằng kênh truyền, tách đặc trưng. Với ưu điểm nổi bật so với bộ lọc tương tự (Analog filter) là tham số chính xác và tái lập, dễ tích hợp vào chuỗi xử lý rác sau ADC, linh hoạt thay đổi cấu hình theo phần mềm và phần cứng, ít chịu ảnh hưởng của sai lệch linh kiện và nhiệt độ. Hai họ bộ lọc phổ biến nhất là FIR (ổn định, có thể đạt pha tuyến tính, thích hợp cho ứng dụng đòi hỏi bảo toàn dạng sóng cả về biên độ và pha) và IIR (đạt đáp ứng chính xác hơn với bậc thấp hơn so với FIR nhưng có pha phi tuyến, đòi hỏi kiểm soát ổn định). Việc lựa chọn kiến trúc, bộ lọc, miền thực thi (thời gian, tần số) luôn gắn với yêu cầu về độ trễ, thông lượng và ràng buộc tài nguyên của hệ thống thực tế.

Tuy nhiên, thực tiễn triển khai cho thấy nhiều môi trường tín hiệu là tín hiệu không dừng (nonstationary) ví dụ tiếng ồn môi trường thay đổi theo thời gian, kênh truyền vô tuyến biến động, đặc tính nguồn thu phát đổi khi người dùng di chuyển. Khi đó, một bộ lọc cố định dù tối ưu ở thời điểm thiết kế cũng nhanh chóng suy giảm hiệu quả. Bộ lọc thích nghi xuất hiện như lời giải cho các bất cập trên bằng cách cập nhật liên tục trọng số của bộ lọc dựa trên sai số ước lượng để duy trì hiệu quả lọc trong điều kiện môi trường biến thiên liên tục. Trong số các thuật toán thích nghi, Least Mean Squares (LMS) được ưa chuộng nhờ ba điểm mạnh cốt lõi là quy tắc cập nhật tuyến tính cập nhật hệ số bộ lọc theo hướng giảm dần gradient xấp xỉ, độ phức tạp khi tính toán thấp (chủ yếu là MAC multiplyaccumulate) từ đó tiết kiệm tài nguyên trên FPGA, phù hợp xử lý thời gian thực, và hiệu quả mang lại là rất cao nếu so sánh về hiệu quả và tiêu tốn tài nguyên trên FPGA với bộ lọc FIR direct form thông thường. Các biến thể như NLMS (chuẩn hoá theo công suất tín hiệu), Leaky LMS (chống trôi hệ số), Sign-LMS (giảm chi phí nhân), hay LMS (tối ưu cho xử lý khói/băng thông cao) mở rộng phạm vi ứng dụng từ khử vọng thoại/âm thanh (AEC), khử nhiễu chủ động, triệt nhiễu ECG/EMG trong y sinh, đến cân bằng kênh truyền, ước lượng hệ thống và điều khiển thích nghi. Chính vì thế em chọn đề tài thiết kế BỘ LỌC THÍCH NGHI SỬ DỤNG GIẢI THUẬT LMS VÀ THỰC HIỆN HÓA TRÊN FPGA này.

1.2 Tình hình nghiên cứu trong và ngoài nước

1.2.1. Tình hình nghiên cứu trong nước

Tại Việt Nam, các mục nghiên cứu về bộ lọc thích nghi trong xử lý số tín hiệu (DSP) trong những năm vừa qua đã rất phát triển, điển hình như: Khử tiếng vọng trên nền DSP/FPGA, các bài báo khoa học trong nước mô phỏng AEC và hiện thực trên kit DSP (TMS320C6713) của các trường đại học lớn hoặc các đề tài đồ án của sinh viên. Giải thuật được dùng trong nhiều trong các thiết bị đầu thu và phát âm thanh như trong phòng thu âm, phòng hội nghị.



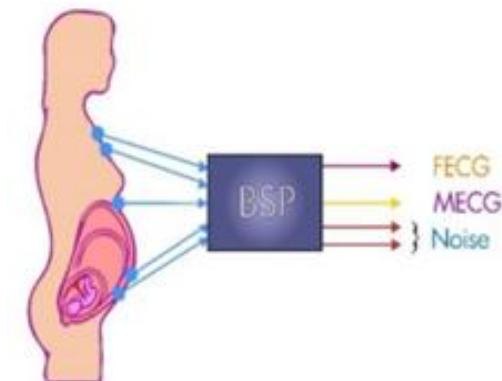
Hình 1-1: Mô hình khử tiếng vọng âm thanh

1.2.2. Tại nước ngoài:

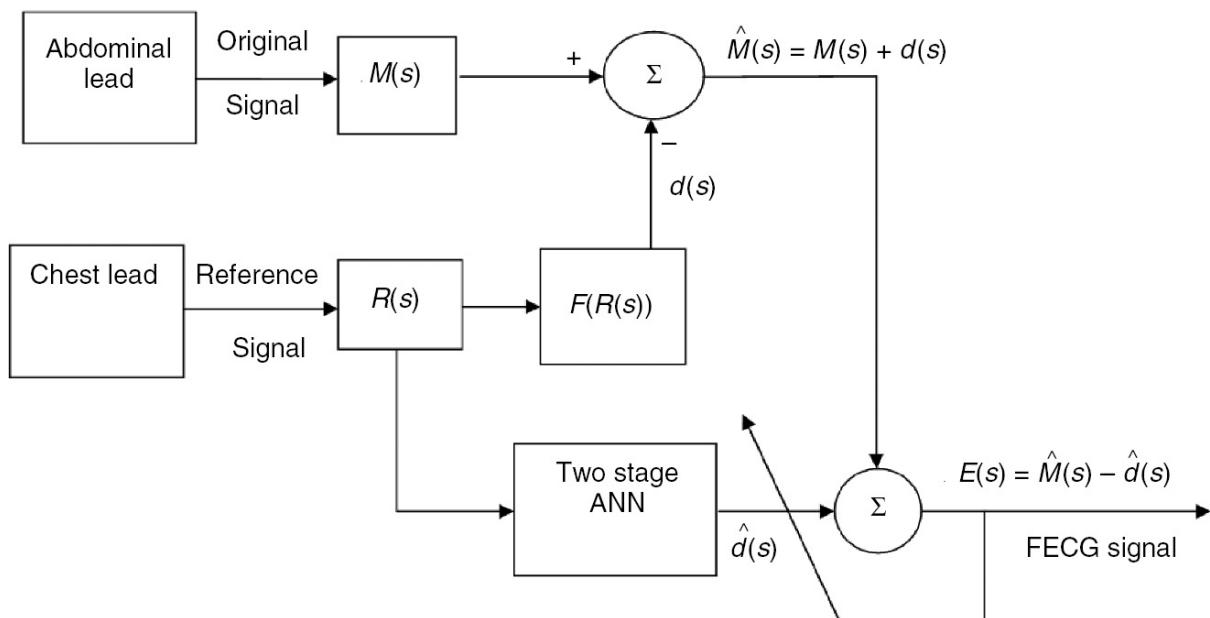
Tại nước ngoài, các sản phẩm được ứng dụng bộ lọc thích ứng đã được nghiên cứu phát triển và thương mại từ lâu, điển hình thường thấy nhất là các sản phẩm về điện tử y sinh, rada, sonar trong quốc phòng. Bên cạnh đó, nhiều bài báo khoa học, nhiều nghiên cứu mới về adaptive filter cũng đã được xuất bản lên IEEE Xplore và tăng cường giải thuật và tăng tốc phần cứng (Hardware Acceleration) để có thể thu nhỏ thiết kế nhiều hơn nhưng về hiệu năng vẫn không đổi từ đó có thể thu nhỏ kích thước module chip DSP để dễ dàng tích hợp vào những thiết bị yêu cầu độ chính xác cao nhưng kích thước phải không đổi. Tiêu biểu như:

- Tối ưu NLMS cho AEC: các nguyên lý tổng quan của Paleologu Benesty để xuất các cách điều khiển learning rate/ regularization để cân bằng giữa tốc độ hội tụ nhanh với sai lệch nhỏ, giúp bộ lọc theo giải thuật LMS tối ưu hơn.

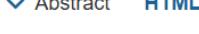
- AEC và ANC trên FPGA: nhiều công trình mô tả kiến trúc RTL LMS và NLMS, đánh giá tài nguyên và hiệu quả giải thuật trên các kit FPGA Virtex và Zynq, cho thấy ngoài những đề tài về tối ưu giải thuật như của Paleologu chúng ta còn có thể khai thác mạnh hơn về tối ưu phần cứng để có thể tính toán giải thuật một cách hiệu quả và mang lại một kết quả tốt hơn.
- FxLMS và biến thể hiện đại: nghiên cứu ANC dùng FxLMS (kể cả cấu trúc systolic, delay-tolerant và tăng tốc phần cứng), cùng các hướng VSS NLMS và SM-NLMS cho môi trường nhiễu biến thiên mạnh.
- Mô hình và thuật toán kết hợp: một số nhóm kết hợp LMS với tối ưu hoá (ACO, PSO) hay học sâu cho giai đoạn RES, đây là các nghiên cứu mới và có thể được phổ biến trong tương lai.



Hình 1-2: Ứng dụng của DSP trong y sinh



Hình 1-3: Sơ đồ khói bộ tách tạp âm từ cơ thể mẹ lẫn vào khi đo tim thai

-
- [A Variable Step-Size MSD-Based Diffusion LMS Algorithm With Noisy Input and Quantization Link Noise](#) 
- Jeongmin Park; Chan Park; PooGyeon Park
IEEE Wireless Communications Letters
Year: 2025 | Early Access Article | Publisher: IEEE
- ▼ Abstract  
-
- [Fourier Domain Gradient Descent Total Least Square/Fourth Algorithm for Efficient Adaptive Direction of Arrival Estimation](#) 
- Joel S.; Shekhar Kumar Yadav; M. L. N. S. Karthik; Nithin V. George
IEEE Transactions on Vehicular Technology
Year: 2025 | Early Access Article | Publisher: IEEE
- ▼ Abstract  
-
- [Optimized Interpolation-Based Methods With Inverse Filtering for Time Delay Estimation](#) 
- Chenhao Zhai; Youhao Kong; Chenxi Liao; Qing Shen
2025 8th International Conference on Electronics Technology (ICET)
Year: 2025 | Conference Paper | Publisher: IEEE
- ▼ Abstract   
-
- [Adaptive Filtering and Machine Learning Approaches to Radio Frequency Interference Mitigation and Bandwidth Optimization in Space Communication Systems](#) 
- Md Sahin Ali; Md Shahidul Islam Riaz; Md Meherab Khandokar Pulok; Chowdhury Rahat Absar; Md Kaium Hossain Fahim; Lei Chen
2025 8th International Conference on Electronics Technology (ICET)
Year: 2025 | Conference Paper | Publisher: IEEE
- ▼ Abstract   
-
- [An Adaptive Beamforming Method Based on Klms Algorithm](#) 
- Shuyu Lou; Xiangbo Sun; Yuebo Li
2025 IEEE 14th International Conference on Communications, Circuits and Systems (ICCCAS)
Year: 2025 | Conference Paper | Publisher: IEEE
- ▼ Abstract   

Hình 1-4: Các nghiên cứu khoa học trên trang ieeexplore.ieee.org

1.3 Nhiệm vụ đồ án

1.3.1. Tìm hiểu tính cấp thiết và nhu cầu về sử dụng bộ lọc thích ứng

Trong các hệ thống thực tế (âm thanh, viễn thông, radar, sonar, thiết bị y sinh), với điều kiện làm việc liên tục, nhiều sẽ luôn thay đổi theo thời gian, kênh truyền biến thiên, vị trí nguồn thu phát thay đổi, thiết bị dần lão hóa gây ảnh hưởng đến hiệu quả lọc và xử lý nhiều. Bộ lọc cố định dù được thiết kế tối ưu ở thời điểm thiết kế ban đầu cũng nhanh chóng mất tối ưu và độ hiệu quả khi môi trường đổi. Bộ lọc thích nghi (adaptive filter), đặc biệt là họ

LMS/NLMS/Leaky LMS, khắc phục điểm yếu đó bằng cách luôn cập nhật hệ số bộ lọc để duy trì sai số nhỏ giữa tín hiệu thực và ước lượng tín hiệu thực và nâng SNR trong dài điều kiện rộng, với tiêu chí tài nguyên thấp (MAC) và khả năng hiện thực tốt trên FPGA hay SoC. Những yêu cầu bao gồm:

-Yêu cầu sử dụng theo từng lĩnh vực tiêu biểu:

- ❖ Với các ứng dụng về xử lý âm thanh (AEC, ANC, khử nhiễu nền): Triệt vọng phòng hay loa micro, lọc tiếng ồn quạt, máy nén, chống gió ngoài trời. Yêu cầu: hội tụ nhanh mở máy là dùng được, ERLE từ 15 đến 25 dB.
- ❖ Radar, Sonar trong quốc phòng và dân sự: Beamforming nulling thích nghi, triệt clutter biển, địa hình, bù trễ kênh, theo vết mục tiêu yêu. Yêu cầu: nâng SINR tại đầu ra beamformer, kiểm soát Pd và Pfa, hoạt động trong tín hiệu nhiễu mạnh và phô thay đổi.
- ❖ Y sinh (ECG, EMG, đo tim thai, FECG): Tách tín hiệu tim thai từ kênh bụng với tham chiếu ngực mẹ, triệt nhiễu cơ của người mẹ, nhiễu lướt 50/60 Hz. Yêu cầu: bảo toàn hình thái sóng (QRS), SNRi từ 6 đến 12 dB, an toàn và trễ thấp cho hiển thị thời gian thực.
- ❖ Trong truyền dẫn thông tin, bộ cân bằng kênh trong truyền dẫn tốc độ cao: Cân bằng ISI, bù méo pha hoặc biên độ, giảm nhiễu giao thoa đồng kênh, yêu cầu: BER giảm rõ rệt, thích nghi theo thay đổi SNR fading.
- ❖ Công nghiệp và IoT: Giám sát rung, chẩn đoán máy, lọc tín hiệu cảm biến trong môi trường nhiễu cao, yêu cầu: năng lượng thấp, kết quả đáng tin cậy, dễ cấu hình và thiết lập nhanh.

-Yêu cầu hệ thống khi triển khai thực tế

- ❖ Thời gian thực và độ trễ: pipeline để tăng tần số hoạt động, với âm thanh cần tần số lấy mẫu fs từ 48 đến 96 kHz, trễ pipeline nhỏ hơn vài ms, radar sonar có thể tới hàng trăm kHz tới MHz tuỳ ứng dụng.
- ❖ Hội tụ và ổn định theo thời gian.
- ❖ Số cố định (fixed point) tin cậy: định dạng Q (ví dụ Q1. 23 cho audio), làm tròn lên và làm tròn gần nhất, kiểm soát tràn ở mọi node, đo tỉ lệ saturate để tinh chỉnh bước học, hoặc có thể dùng hệ thống số Floating point để đạt hiệu quả tính toán cao hơn.
- ❖ Kiến trúc phần cứng: Datapath ước lượng $y[n]$ dùng DSP slice (MAC), datapath cập nhật Δw tách riêng. BRAM/URAM cho weight, dùng thanh ghi dịch. Pipeline 2 tới 3 stage để đạt Fmax.

- ❖ Đo lường hiệu quả: MSE, ERLE, SNR, tốc độ hội tụ, độ bền double talk với âm thanh, SINR, Pd Pfa trong radar sonar, RMSE hình thái QRS (y-sinh).
- ❖ Ràng buộc kỹ thuật: công suất tiêu thụ, diện tích thiết kế (FPGA), chi phí B RAM, độ tin cậy, tiêu chuẩn an toàn (đặc biệt y sinh, quốc phòng).

1.3.2. Phân tích yêu cầu

1.3.2.1. Mục tiêu và phạm vi

Mục tiêu tổng quát: Xây dựng hệ thống xử lý tín hiệu thời gian thực trên FPGA nhằm giảm nhiễu và cải thiện chất lượng âm thanh bằng bộ lọc thích ứng LMS, và để kiểm chứng khả năng trên cần tạo ra thêm các khói tạo nhiễu, cấu hình nhiễu để quan sát và đối chiếu kết quả trực tiếp, để đánh giá chất lượng bộ lọc và khả năng cải thiện tín hiệu.

Phạm vi: Làm việc với tín hiệu audio, tốc độ lấy mẫu tầm 48 kHz, xử lý, phát lại và hiển thị các đại lượng đặc trưng (tín hiệu sạch ước lượng, nhiễu ước lượng, sai số). Hệ thống chạy độc lập trên kit FPGA DE10, giao tiếp codec audio WM8731.

1.3.2.2. Yêu cầu chức năng

-Thu nhận và phát lại tín hiệu: Nhận tín hiệu audio từ ngõ vào chính (primary) và tín hiệu tham chiếu (reference) sau khi xử lý nhiễu đưa ra codec. Phát lại tín hiệu sau xử lý qua DAC, hỗ trợ xuất các ngõ ra ước lượng (signal estimate, noise estimate).

-Xử lý thích ứng:

Thực hiện lọc FIR thích ứng theo thuật toán LMS:

$$y[n] = \sum_{k=0}^{M-1} w_k[n] x[n - k], \quad (1.1)$$

$$e[n] = d[n] - y[n], \quad (1.2)$$

$$w_k[n + 1] = w_k[n] + \mu e[n] x[n - k], \quad k = 0, 1, \dots, M - 1. \quad (1.3)$$

Tham số cấu hình: số tap N, hệ số học μ .

-Quản lý tín hiệu và định dạng: Hỗ trợ chuyển đổi dữ liệu giữa dạng 24bit bù 2 và IEEE-754 FP32, chuẩn hoá. Đồng bộ theo LRCLK/BCLK của codec, đảm bảo lấy mẫu và phát mẫu đúng pha.

-Giao tiếp và cấu hình: Cấu hình codec qua I2C. Mô phỏng bằng các EDA trước khi đỗ kit để xem dạng sóng và kiểm tra xem các tín hiệu cấu hình đã chạy đúng với định dạng mà ta mong muốn hay chưa và khi đỗ KIT FPGA cần thêm một tín hiệu LED dùng để debug để biết trạng thái cấu hình hoàn thành hay chưa

-Giám sát và an toàn: Có tín hiệu reset, khoá trạng thái khi lỗi cấu hình, các ngưỡng chống tràn, chống bão hòa số Floating point.

1.3.2.3. Yêu cầu phi chức năng

-Thời gian thực: độ trễ đầu cuối thấp hơn chu kỳ lấy mẫu nhiều lần (từ vài µs tới ms) để không gây méo âm thanh đầu ra.

-Ôn định và tin cậy: hệ thống chạy liên tục, không mất đồng bộ I2S, thuật toán hội tụ ổn định.

-Chất lượng âm thanh: SNR sau lọc tăng so với trước lọc, hạn chế méo tín hiệu sạch.

-Khả năng mở rộng: kiến trúc module hoá, có thể tăng số tap, thay thuật toán (NLMS/RLS) mà không thay đổi lõi I/O.

-Khả năng bảo trì: mã nguồn Verilog rõ ràng chia theo t, có testbench cơ bản, có tham số hoá.

1.3.2.4. Ràng buộc kỹ thuật

-Phần cứng: Kit DE10 (nguồn 5 V, I/O 3.3 V), codec WM8731, dây nối I2S và I2C đúng tiêu chuẩn phần cứng.

-Chuẩn giao tiếp: I2S (BCLK, LRCLK, SDIN/SDOUT), cấu hình I2C, GPIO phục vụ debug.

-Tài nguyên FPGA: giới hạn số lượng ALM/LUT/DSP, ưu tiên pipeline ngắn, không tạo latch, tránh phép chia tốn kém.

-CLK: 50 MHz (clk được cấp sẵn trong FPGA DE10), xấp xỉ 48 kHz (audio). Bộ chia xung clk PLL phải chia đúng tần số để module I2C ổn định trước khi chạy xử lý.

-Đảm bảo an toàn: tuân thủ mức điện áp, nối đất, chống nhiễu crosstalk trên đường audio giao tiếp.

1.3.2.5. Các thông số cấu hình mong muốn

-Tốc độ mẫu: 48 kHz mono (tuỳ cấu hình).

-Cải thiện SNR: tăng SNR hoặc giảm năng lượng nhiễu đo được tại ngõ ra (mức cải thiện phụ thuộc mức độ nhiễu so với tín hiệu có nghĩa).

Tài nguyên: số DSP/LUT sử dụng trong giới hạn của DE10, tần số 50 MHz đạt timing.

1.3.2.6. Kiểm thử

-Mô phỏng RTL: kiểm tra tính đúng sai các khối chuyển đổi từ 24b sang FP32 và ngược lại, phép cộng trừ nhân FP32 phải được kiểm thử ở nhiều trường hợp đảm bảo luôn tính toán đúng để khi cộng dồn bộ FIR tránh gây sai lệch kết quả, cập nhật hệ số bộ lọc, độ hội tụ trọng số.

-Để kiểm chứng thực tế: phát nhiễu có kiểm soát (tam giác, white noise), đo dạng sóng $x[n]$, $d[n]$, $y[n]$, $e[n]$ trên oscilloscope, so sánh phô trước và sau lọc.

-Tiêu chí đánh giá: hệ thống chạy ổn định, không drop sample, kết quả lọc cải thiện rõ rệt theo câu hình μ , số tap, không có tràn định dạng.

1.3.2.7. Rủi ro và biện pháp khắc phục

-Giải thuật hội tụ chậm: chọn μ phù hợp, giới hạn $|w_k|$.

-Tràn: chuẩn hóa thang đo, bão hòa đầu ra, kiểm soát biên độ qua codec.

-Lệch pha: đồng bộ cứng theo LRCLK, pipeline nhất quán, kiểm tra timing closure.

-Nhiều phần cứng: chú ý layout, dây I2S/I2C ngắn và tách biệt analog/digital ground.

1.3.3. Nhiệm vụ của đồ án

-Phạm vi và nhiệm vụ của đồ án em sẽ chia thành ba hợp phần chính, từ mô phỏng đến triển khai phần cứng và đánh giá thực nghiệm như sau:

- Nghiên cứu – mô phỏng thuật toán LMS trên MATLAB. Hệ thống hóa cơ sở lý thuyết (hàm mục tiêu, miền ổn định của hệ số học, LMS, xây dựng mô hình lọc FIR thích ứng và khảo sát hội tụ theo bậc lọc, μ và phô tín hiệu. Tạo bộ dữ liệu kiểm thử gồm các thành phần sóng cơ bản (sin, vuông, tam giác, răng cưa) và tín hiệu âm thanh có nhiễu, đánh giá bằng các thước đo MSE/SNR, thời gian hội tụ, sai lệch biên–pha để xác định cấu hình hệ số lọc tối ưu.
- Hiện thực phần cứng cho việc đồ kit FPGA DE10. Thiết kế bộ lọc LMS dạng FIR trực tiếp bằng Verilog, chuẩn hóa đường dữ liệu từ 24bit bù 2 sang FP32 và ngược lại, xây dựng

các khối tính toán (cộng trừ nhân theo FP, bộ dịch, bộ chuẩn hóa) và điều chỉnh pipeline để đáp ứng tốc độ lấy mẫu 48 kHz. Mô phỏng RTL Testbench bảo đảm với MATLAB, sau đó tổng hợp mạch bằng Quartus và chạy trên kit FPGA DE10.

3. Tích hợp I/O âm thanh và đánh giá thực nghiệm. Cấu hình codec WM8731 qua I2C, truyền nhận số liệu âm thanh I2S (ADC từ máy tính sang FPGA, DAC từ FPGA ra loa/oscilloscope). Thực hiện các kiểm thử lọc nhiễu trộn vào sóng cơ bản và âm thanh thực, đánh giá bằng tai theo thang đo và đánh giá khách quan bằng FFT trên oscilloscope để xác nhận mức suy giảm nhiễu và mức bảo toàn tín hiệu gốc.

Kết quả mong đợi gồm: Giải thuật LMS tối ưu ở mô phỏng, kiến trúc RTL đáp ứng đủ tài nguyên thực trên DE10, và minh chứng thực nghiệm theo các thang đánh giá, cho thấy chất lượng tín hiệu được cải thiện rõ rệt và chỉ ra điểm còn hạn chế của thiết kế.

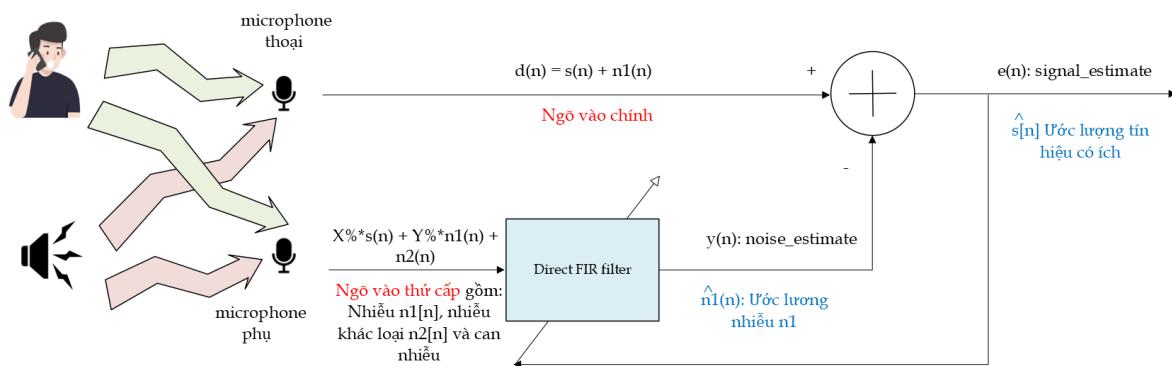
2. LÝ THUYẾT

2.1. Tiền đề cho bộ lọc thích nghi

2.1.1. Đặt vấn đề



Hình 2-1: Mic có bộ lọc thích ứng giúp định hướng và khử nhiễu âm thanh



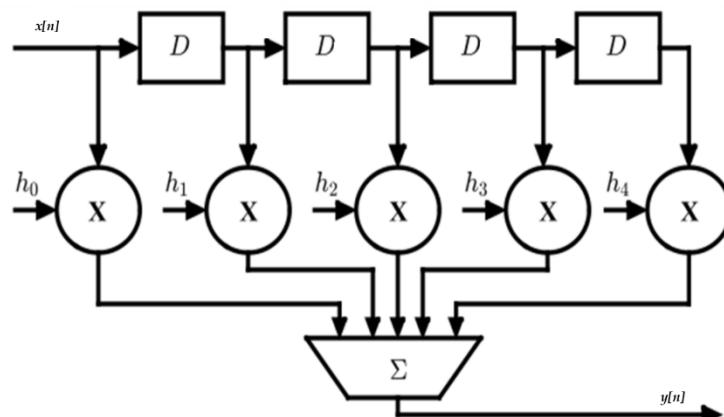
Hình 2-2: Mô hình thêm microphone phụ để khử nhiễu cho microphone chính

Ở các thiết bị cầm tay có chức năng thu âm nói chung, người dùng nói trước microphone chính (primary mic). Song song đó, các thiết bị còn bố trí microphone thứ cấp (reference mic) đặt ở một vị trí khác nơi mà có thể thu được tín hiệu nhiễu rõ ràng hơn thường ở mặt sau máy để nghe trường nhiễu môi trường. Mục tiêu của bộ lọc thích nghi là triệt nhiễu trong microphone chính bằng cách học sự tương quan giữa tín hiệu ở microphone thứ cấp và thành phần nhiễu trong microphone chính.

Trong đó:

- $s[n]$: tín hiệu lời nói (clean speech) mà ta muốn bảo toàn.
- $n1[n]$: nhiễu môi trường tác động lên mic chính (cần triệt).
- $n2[n]$: nhiễu ngoại lai chỉ xuất hiện ở mic thứ cấp để đảm bảo rằng nhiễu khác với nhiễu $n1[n]$ thì ta nên chọn nhiễu khác loại với nhiễu $n1$.
- Primary input: ngõ vào chính còn được gọi là Desire signal $d(n)$

2.1.2. Bộ lọc FIR với hệ số cố định và hạn chế so với khi sử dụng hệ số thích ứng



Hình 2-3: Sơ đồ tín hiệu của Direct FIR filter

Trong miền rời rạc, một hệ LTI (tuyến tính bất biến theo thời gian) được đặc trưng bởi đáp ứng xung $h[n]$. Với ngõ vào $x[n]$, ngõ ra $y[n]$ là phép chập:

$$y[n] = (h * x)[n] = \sum_{k=0}^{N-1} h[k] x[n-k]. \quad (2.1)$$

Nếu $h[n]$ hữu hạn độ dài N (Finite Impulse Response – FIR). Công thức trên cũng chính là phương trình sai phân dạng tích cộng dồn (MAC) dùng để hiện thực trên DSP/FPGA.

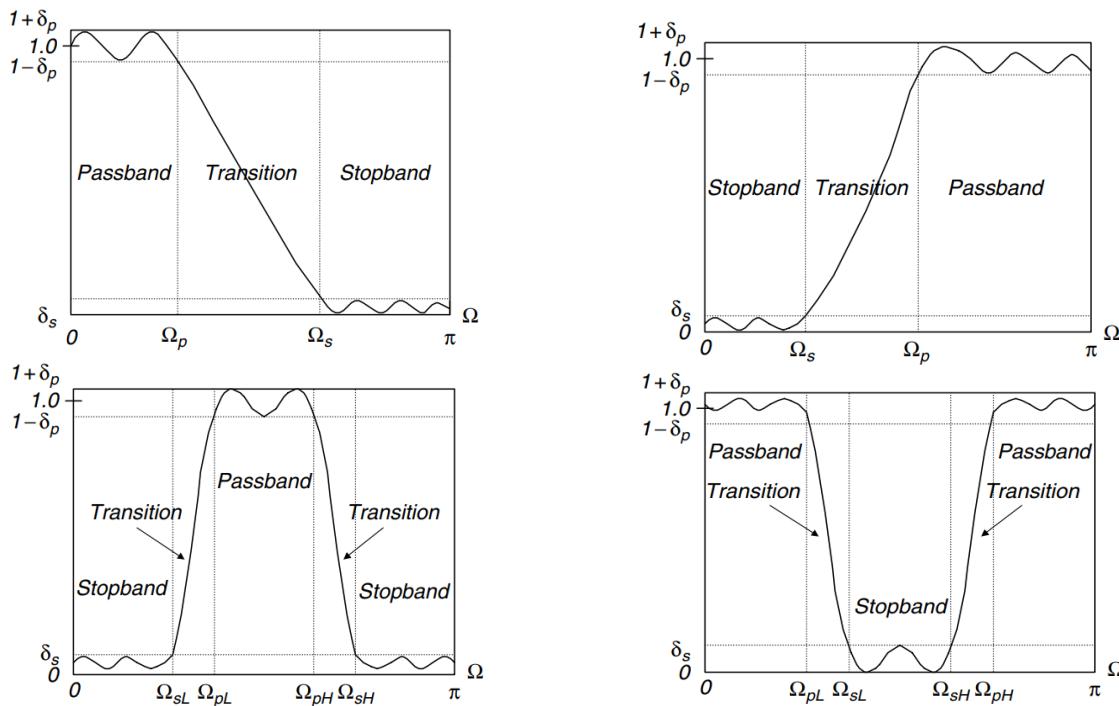
Trong miền tần số, đáp ứng pha của FIR là:

$$H(e^{j\Omega}) = \sum_{k=0}^{N-1} h[k] e^{-j\Omega k}, \quad \Omega \in [0, \pi]. \quad (2.2)$$

Việc thiết kế lọc nghĩa là chọn $h[k]$ sao cho $H(e^{j\Omega})$ gần với đáp ứng mong muốn: thông thấp, thông cao, thông dài, chấn dài... theo các ràng buộc:

- Dải thông: ($|H| \in [1 - \delta_p, 1 + \delta_p]$) trên miền $([0, \Omega_p])$
 - Dải chấn: ($|H| \leq \delta_s$) trên $([\Omega_s, \pi])$

- Vùng chuyển tiếp (pass band): bề rộng ($\Delta\Omega = \Omega_s - \Omega_p$), ($\Delta\Omega$) càng hẹp thì cần bậc bộ lọc càng lớn.



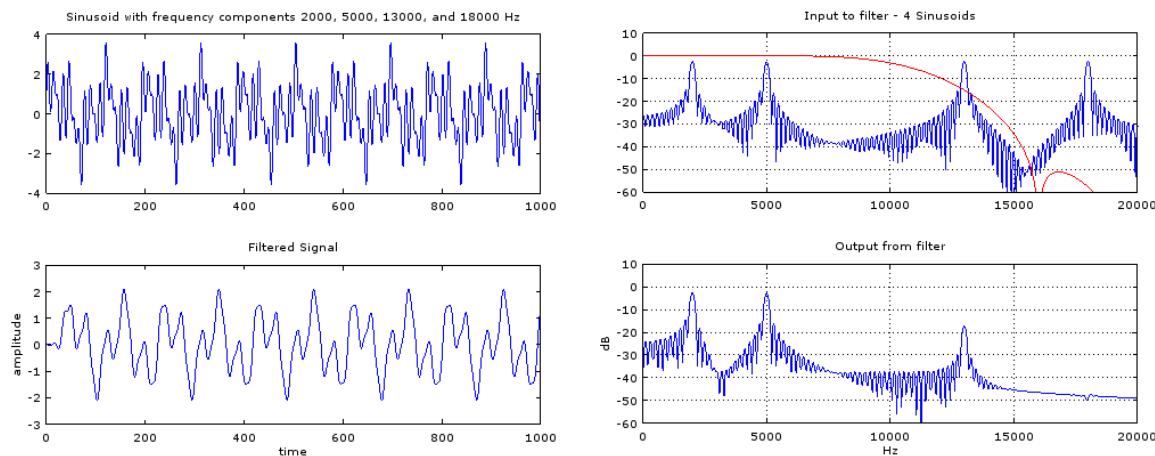
Hình 2-4: Các dạng đáp ứng bộ lọc phổ biến

Các bộ lọc số có hệ số cố định chỉ hoạt động tốt khi ta biết trước đầy đủ đặc tả: kiểu lọc (LPF/HPF/BPF/BSF), tần số cắt cao và thấp Ω_s, Ω_p và dung sai δ_p, δ_s . Một khi đã thiết kế và triển khai, các hệ số lọc không thay đổi theo thời gian. Chính tính bất biến này lại trở thành điểm bất lợi khi truyền tín hiệu trong môi trường thực, nơi nguồn nhiễu biến thiên, điều kiện kênh thay đổi, hoặc khi đặc tính hệ thống không thể xác định chính xác từ trước.

Trong thực tế, phổ nhiễu và hàm truyền kênh $h[n]$ thường thời biến: nguồn phát di chuyển, trường âm thay đổi do phản xạ, dội âm, hay xuất hiện can nhiễu mới. Khi đặc tả ban đầu không còn phù hợp, bộ lọc cố định nhanh chóng bị lệch pha và biên độ, dẫn tới lọt nhiễu dư thừa hoặc làm méo tín hiệu hữu ích $s[n]$. Nói cách khác, một thiết kế đúng ở thời điểm ban đầu có thể trở nên sai ở khoảng thời gian sau đó.

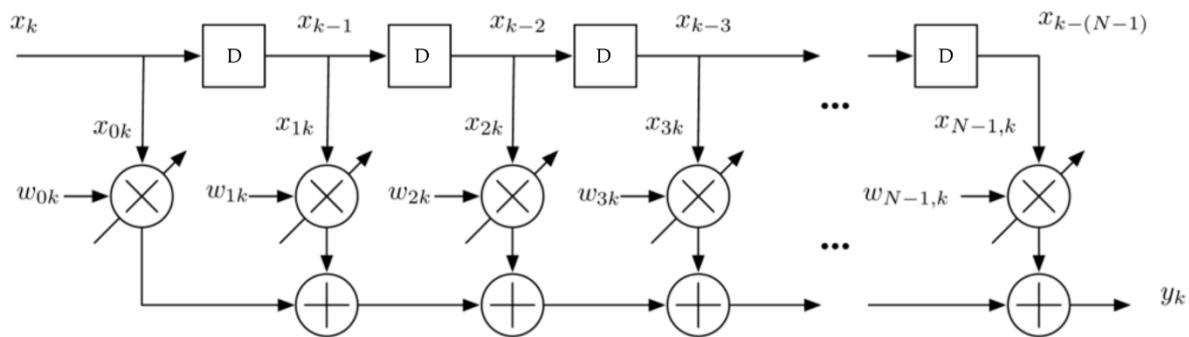
Vì vậy, ta cần một cơ chế tự thích nghi theo thời gian thực, cho phép học và cập nhật các hệ số bộ lọc phù hợp với môi trường ngay tại thời điểm vận hành. Giải pháp tiếp cận đó chính là bộ lọc thích nghi (adaptive filter). Trong nhiều thuật toán cập nhật hệ số, LMS (Least Mean Squares) được ưa chuộng nhờ: cấu trúc đơn giản, dễ hiện thực

hiện trên FPGA, tiêu tốn tài nguyên FPGA thấp, phù hợp thời gian thực, hiệu quả sau lọc, và khả năng ước lượng tốt nhiễu biến thiên của kênh và nhiễu. Nhờ đó, LMS trở thành lựa chọn an toàn khi yêu cầu vừa thực dụng vừa hiệu quả trong các hệ xử lý tín hiệu thời gian thực.



Hình 2-5: Đáp ứng tần số bộ lọc FIR với hệ số cố định

Như hình 2-5 ta có thể thấy bộ lọc FIR chỉ lọc tốt khi ta biết rõ đâu là tần số của tín hiệu ta muốn giữ lại và đâu là tần số tín hiệu ta muốn loại bỏ. Bộ lọc thích ứng (Adaptive filter) sinh ra để một phần nào đó khắc phục điều trên, tuy có thể khả năng loại bỏ tín hiệu không mong muốn chưa thể bằng bộ lọc cố định nhưng ở mức độ nào đó có thể chấp nhận được là cải thiện tín hiệu. Mô hình bộ lọc thích ứng đơn giản là sẽ có thêm thành phần cập nhật hệ số bộ lọc để hồi tiếp vào bộ Direct FIR như sau:



Hình 2-6: Mô hình bộ lọc FIR thêm thành phần cập nhật hệ số bộ lọc

2.2. Lý Thuyết Về Giải thuật LMS

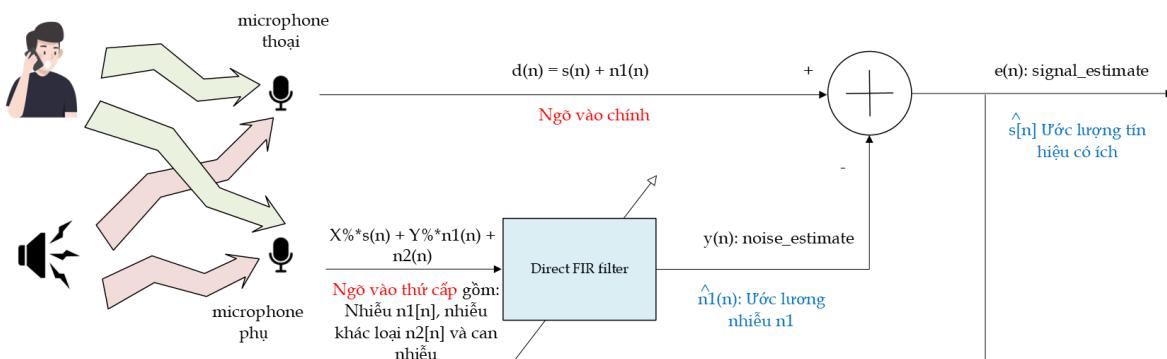
2.2.1. Giải thuật LMS

- LMS xuất phát từ công trình Adaptive Switching Circuits của Bernard Widrow và Marcian Hoff (còn gọi là Widrow Hoff hay Delta rule) là mốc khai sinh của bộ lọc thích nghi tuyến tính và là nền tảng cho nhiều thuật toán lọc thích nghi hiện đại. Ý tưởng cốt lõi là với một mô hình tuyến tính đơn giản, điều chỉnh các hệ số theo hướng làm giảm bình phương sai số trung bình (MSE) giữa tín hiệu gốc và tín hiệu ước lượng sau mỗi mẫu đưa vào bộ lọc.

Sau đó, các giáo trình của Haykin phát triển thêm khung lý thuyết từ bài toán Wiener (tối ưu bậc hai) bổ sung thêm phương pháp steepest descent và thay thế gradient thật bằng ước lượng tức thời để thu được LMS, đồng thời phân tích hội tụ, misadjustment, và các biến thể như NLMS, leaky LMS, sign LMS

-Trong các ứng dụng như khử vọng khử nhiễu âm thanh, cân bằng kênh, sonar/radar, tín hiệu y sinh. LMS vẫn là lựa chọn an toàn nhất vì đơn giản, tốn ít tài nguyên phần cứng, mà vẫn đủ mạnh với môi trường nhiễu thay đổi liên tục. Các cải tiến sau đó như là NLMS cải tiến độ chính xác của hệ số học (learning rate) đã được tổng quan chi tiết cho bài toán echo cancellation.

-Để chứng minh giải thuật LMS em sẽ bám theo ví dụ sau đây:



Hình 2-7: Ví dụ minh họa chứng minh giải thuật LMS

-Một bộ lọc thích ứng theo giải thuật LMS sẽ có 2 ngõ vào và 2 ngõ ra tương ứng như hình 2-7 ở trên. Ngõ vào gồm: ngõ vào chính (Primary input $d[n]$) và ngõ vào thứ cấp có nhiệm vụ tham chiếu nhiễu cho ngõ vào chính (Reference input $x[n]$). Ngõ ra gồm ước lượng tín hiệu gốc ($e[n]$) và ước lượng tín hiệu nhiễu là ngõ ra của bộ lọc FIR $y[n]$.

-Ngõ vào chính $d[n]$ sẽ gồm tín hiệu có ích kèm theo một nhiễu $n1[n]$, mục tiêu cuối cùng của giải thuật là làm sao để tối thiểu được nhiễu $n1[n]$ này và nhận tối đa tín hiệu có ích $s[n]$ ở đầu

ra ước lượng $e[n]$. Ở ngõ vào thứ cấp ta sẽ thu được một phần nào đó tín hiệu tương quan với tín hiệu nhiễu $n_1[n]$ ở ngõ vào chính, nhưng thực tế ngõ vào thứ cấp còn bị can nhiễu từ tín hiệu có ích từ ngõ vào chính lan sang và đồng thời chịu tác động của nhiễu $n_2[n]$ nhiễu khác loại hoàn toàn so với nhiễu $n_1[n]$. Cho nên ý tưởng của bộ lọc thích ứng theo giải thuật LMS đơn giản là dùng bộ lọc FIR tái lập các thành phần phô và biên độ của tín hiệu tham chiếu ở ngõ vào thứ cấp trở thành giống thành phần nhiễu $n_1[n]$ nhất có thể khi đó $d[n] - y[n]$ thì sẽ ra được ước lượng tín hiệu gốc $e[n]$, ở hình 2.7 tên tín hiệu nào có mũ ở trên đầu chính là thành phần ước lượng cho tín hiệu phí dưới nó.

$$\hat{s}_k = d_k - y_k \equiv s_k + n_k - \hat{n}_k \quad (2.3)$$

Trong đó: K là iteration index, còn N là time index

- Thành phần nhiễu n_k và sáp xỉ nhiễu \hat{n}_k càng tương quan với nhau thì đầu ra bộ lọc càng tốt hay nói cách khác nếu xem các thành phần sóng là thành phần liên tục thì công suất của ngõ ra $e[n] = \hat{s}_k$ càng bé thì tỉ số công suất trung bình của tín hiệu trên trong suất trung bình của nhiễu SNR càng lớn theo chứng minh sau đây.

$$\hat{s}_k = d_k - y_k \equiv s_k + n_k - \hat{n}_k \quad (2.4)$$

- Bình phương: $\hat{s}_k^2 = s_k^2 + (n_k - \hat{n}_k)^2 + 2 s_k(n_k - \hat{n}_k) \quad (2.5)$

- Trung bình: $E(\hat{s}_k^2) = E(s_k^2) + E((n_k - \hat{n}_k)^2) + E(2 s_k(n_k - \hat{n}_k)) \quad (2.6)$

- Ta có thể tương đương về cuối bằng không khi đó công suất trung bình ước lượng tín hiệu $E(\hat{s}_k^2)$ sẽ bằng tổng công suất trung bình cộng công xuất tàn dư $E((n_k - \hat{n}_k)^2)$ và công suất trung bình tín hiệu gốc $E(s_k^2)$. Do vậy để SNR càng lớn thì công xuất nhiễu tàn dư $E((n_k - \hat{n}_k)^2)$ phải bé nhất tương đương công suất trung bình $E(\hat{s}_k^2)$ cũng phải bé nhất.

- Nếu xét bộ lọc số FIR là dạng các mẫu rời rạc, với N tap thì ngõ ra ước lượng nhiễu sẽ là

$$e_k = d_k - \hat{n}_k = d_k - W^T X_k = d_k - \sum_{i=0}^{N-1} w(i)x_k \quad (2.7)$$

-Vì có N tap ta có thể hiểu W và X_k thành các vector cho nên cần chuyển vị một trong hai vector trên nếu muốn nhân hai vector với nhau.

$$X_k = \begin{bmatrix} x_{0k} \\ x_{1k} \\ x_{2k} \\ \vdots \\ x_{N-1,k} \end{bmatrix} \quad W_k = \begin{bmatrix} w_{0k} \\ w_{1k} \\ w_{2k} \\ \vdots \\ w_{N-1,k} \end{bmatrix}$$

-Tương đương với cách hiểu dưới dạng công suất ở trên muốn SNR là lớn nhất thì công suất trung bình của ngõ ra ước lượng nhiễu phải là bé nhất do đó hàm tối thiểu hóa trung bình bình phương J như sau:

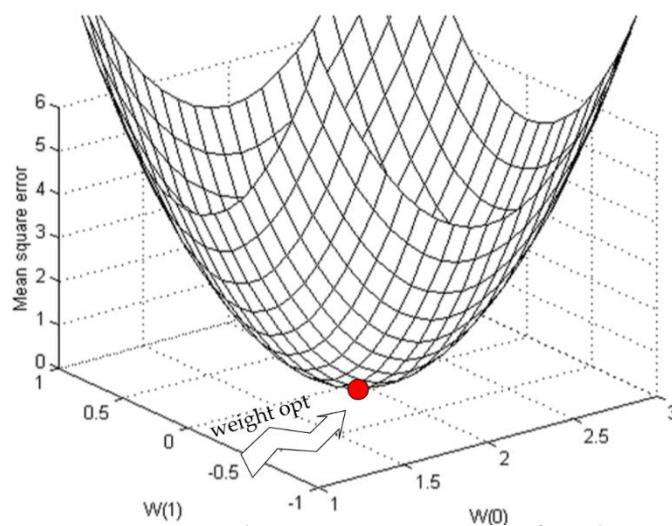
$$\begin{aligned} J &= E(e_k^2) = E(d_k^2) - 2E(d_k W^T X_k) + E(W^T X_k X_k^T W) \\ &= E(d_k^2) - 2P^T W + W^T R W \end{aligned} \quad (2.8)$$

Đặt biến: P : Vector tương quan chéo (cross-correlation vector) $P = E(d_k X_k)$

R : Ma trận tương quan của tín hiệu vào bộ lọc (Autocorrelation matrix) $R = E(X_k X_k^T)$

Hàm tối thiểu hóa: $J(W) = E(e_k^2) = E(d_k^2) - 2P^T W + W^T R W \quad (2.9)$

Có thể thấy trong hàm này X_k là vector đầu vào Reference signal, d_k là primary signal cũng là tín hiệu input, do đó chỉ còn lại biến cuối cùng là biến W có thể thay đổi để làm J biến thiên, do vậy hàm tối thiểu J là hàm theo biến W . Hàm $J(W)$ là hàm bậc 2 với hằng số đứng trước trước $E(d_k^2)$ là số dương nên nếu bộ lọc 1 tap thì hàm J sẽ tạo thành hình paraboloid với lõm hướng lên còn 2 tap thì sẽ tạo thành hình Parasolid, còn 2 tap trở lên sẽ tạo thành mặt yên ngựa hoặc mặt phẳng khác phức tạp hơn.



Hình 2-8: Mặt phẳng phương vị của hàm tối thiểu J

Để dễ hình dung hơn, ta có thể tiến hành thay số thực tế vào phương trình tối thiểu và vẽ bằng mô phỏng hàm mặt phẳng Geogebra để hiểu hơn về bước giảm (steepest descent).

$$\text{Ta có hàm tối thiểu hóa: } J(W) = E(d_k^2) = E(d_k^T d_k) = 2P^T W + W^T R W \quad (2.9)$$

- $E(d_k^2)$ là hằng số vì là ngõ vào chính ta không thể can thiệp.
- $P = E(d_k d_k^T)$ cũng là hằng số vì tương quan giữa 2 ngõ vào giả sử là cố định
- $R = E(X_k X_k^T)$ cũng là hằng số vì tương quan giữ ngõ vào thứ cấp cũng giả sử sẽ là cố định
- Do vậy hàm tối thiểu J sẽ chỉ còn phụ thuộc W

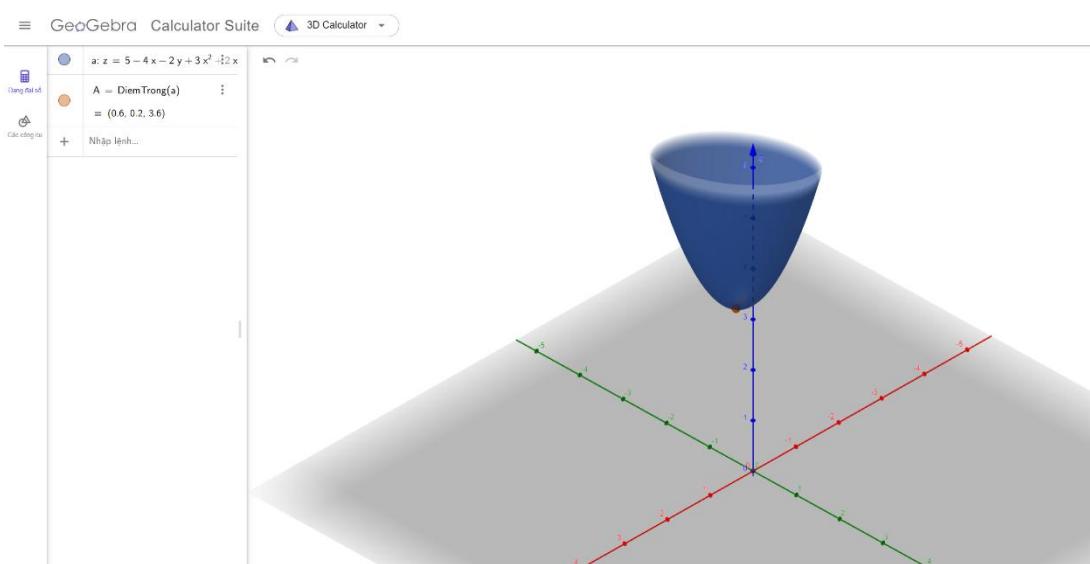
Giả sử các vector trong phương trình trên có giá trị như sau:

- $E(d_k^2) = 5$
- $P = E(d_k d_k^T) = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$
- $R = E(X_k X_k^T) = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$
- $W = \begin{pmatrix} W_0 \\ W_1 \end{pmatrix}$

$$\text{Thê vào hàm tối thiểu hóa } J(W_0, W_1) = 5 - 2(2 \cdot 1) \begin{pmatrix} W_0 \\ W_1 \end{pmatrix} + (W_0 \cdot W_1) \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} W_0 \\ W_1 \end{pmatrix}$$

Tiến hành rút gọn ta được phương trình sau:

$$J(W_0, W_1) = 5 - (4W_0 + 2W_1) + (3W_0^2 + 2W_0W_1 + 2W_1^2) \quad (2.10)$$



Hình 2-9: Mô phỏng mặt phẳng hàm mục tiêu J

Theo như mô phỏng mặt phẳng hàm mục tiêu trên có trên cho ta thấy, với hàm 2 tap sẽ tạo thành mặt phẳng có dạng hình Elliptic Parabolic với 1 hệ số bộ lọc tối ưu nhất làm cho công suất trung bình ngõ ra là bé nhất, để tìm được điểm này đầu tiên ta cần xác định chiều Gradien, để hàm trở về cực tiểu thì để sẽ nằm ngược chiều Gradien, hệ số bộ lọc tối ưu (Weight opt) sẽ là điểm làm cho Gradien(J) = 0

$$\nabla J = \frac{dJ}{dW} = -2P + 2RW \quad (2.11)$$

$$\rightarrow \text{Weight opt} = \frac{P}{R}$$

Để hệ số bộ lọc (Weight) hiện tại có thể tiến tới hệ số bộ lọc tối ưu (Weight opt) theo bước giảm (steepest descent) để cập nhật hệ số bộ lọc mới như sau.

$$W_{k+1} = W_k - \mu \frac{dJ}{dW} = W_k + \mu(2P - 2RW) \quad (2.12)$$

Với μ là learning rate: có chức năng là kiểm soát bước nhảy theo chiều Gradien

$$W_{k+1} = W_k + \mu \frac{dJ}{dW} = W_k + \mu(2P - 2RW) \quad (2.13)$$

Gọi $\mu(-2P + 2RW)$ là Delta weight, là mức thay đổi của hệ số bộ lọc.

Khi khi J đạt tới W_{opt} thì $\nabla J = 0$ tương đương với phương trình trên W không còn cập nhật thành phần Delta nữa hệ số bộ lọc sẽ giữ nguyên và không tăng nữa, ta gọi hiện tượng đó là bộ lọc đã hội tụ sẽ thấy rõ hơn khi mô phỏng Matlab.

$$W_{k+1} = W_k + \mu \nabla J = W_k + \mu(2P - 2RW) \quad (2.14)$$

Với $e_k = y_k - W^T X_k$

P : Vector tương quan chéo (cross-correlation vector) $P = E(d_k X_k)$

R : Ma trận tương quan của tín hiệu vào bộ lọc Autocorrelation matrix $R = E(X_k X_k^T)$

$$W_{k+1} = W_k + \mu \nabla J = W_k + \mu e[n] x[n] \quad (2.15)$$

Với μ được tính theo $0 < \mu < \frac{2}{\lambda_{max}}$ và Được sắp xỉ $\mu \leq \frac{1}{N \cdot P_x}$ (2.16)

Trong đó:

- N : số hệ số (số tap) của bộ lọc thích ứng

- Px: công suất trung bình của tín hiệu đầu vào thứ cấp $x(n)$, tức là:

$$P_x = E[x^2(n)] \quad (2.17)$$

2.2.2. Các thang đánh giá chất lượng bộ lọc

2.2.2.1. Thang đánh giá DCR

Rating	Degradation
1	Very annoying
2	Annoying
3	Slightly annoying
4	Audible but not annoying
5	Inaudible

Bảng 2-1: Thang đánh giá DCR

-Nếu bộ lọc sử dụng cho xử lý nhiễu âm thanh thì thang đánh giá đầu tiên sẽ là cảm nhận bằng thính giác dựa trên thang đo trên. Hoặc nhìn và đánh giá dạng sóng và phô cho ra trước và sau khi lọc.

2.2.2.2. Mean Squared Error (MSE)

-Mean Squared Error (MSE) là một trong những tiêu chí phổ biến nhất để đánh giá hiệu quả của một giải thuật, đặc biệt trong bộ lọc thích ứng. Công thức MSE được định nghĩa như sau:

$$MSE = E(s[n] - e[n])^2 = \frac{1}{N} \sum_{i=0}^{N-1} (s[n] - e[n])^2 \quad (2.16)$$

-Để có thể thấy được các sai lệch nhỏ hơn giữa tín hiệu gốc và ước lượng tín hiệu gốc thay vì ta đánh giá trên thang Mean error, thì đánh giá trên thang MSE sẽ làm cho trọng số của các sai

Actual	Predicted	Error	Square Error
10	13	-3	9
16	20	-4	16
13	12	1	1
19	18	1	1
7	3	4	16
		13	43

Bảng 2-2: Sự khác biệt giữa MSE và ME

lệch bé giữa 2 tín hiệu lớn hơn từ đó biết được các tín hiệu có đang sai lệch nhau quá lớn hay không. MSE thường dưới dạng đơn vị là dB.

2.2.2.3. Tốc độ hội tụ

Như em đã trình bày ở phần lý thuyết giải thuật, bộ lọc hội tụ là khi output của bộ lọc FIR đã ước lượng được tiệm cận nhiễu ở primary signal do đó tỉ số tín hiệu trên nhiễu SNR từ thời điểm đó sẽ là cao nhất. Để có thể quan sát và nhận biết thời điểm hội tụ, đầu tiên có thể nhìn vào Delta weight nếu delta weight gần như không tăng nữa mà chỉ sắp xỉ 0 đó là khi giải thuật đã hội tụ, hoặc ta có thể nhìn vào ước lượng nhiễu $y[n]$, ước lượng nhiễu $y[n]$ sẽ liên tục tăng tới tối đa và sau đó giữ nguyên không tăng nữa.

2.2.2.4. Signal to Noise Ratio (SNR)

Signal to Noise Ratio (SNR) được sử dụng để đánh giá độ mạnh của tín hiệu mong muốn so với nhiễu trong hệ thống. SNR được tính bằng công thức:

$$\text{SNR}_{\text{output}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise}}} \right) (\text{dB}) \quad (2.17)$$

Trong đó:

- ❖ P_{signal} : Công suất tín hiệu sạch.
- ❖ P_{noise} : Công suất nhiễu.

Ý nghĩa:

- Giá trị SNR đầu ra cao cho thấy nhiễu đã được loại bỏ hiệu quả và tín hiệu mong muốn được tái tạo chính xác.
- Thang đo này thường đi kèm với Delta SNR (ΔSNR) để đánh giá hiệu quả lọc.

2.2.2.5. Delta Signal to Noise Ratio (ΔSNR)

Delta SNR (ΔSNR) đánh giá mức cải thiện SNR của tín hiệu sau khi qua bộ lọc thích ứng. Công thức:

$$\Delta \text{SNR} = \text{SNR}_{\text{output}} - \text{SNR}_{\text{input}} \quad (2.18)$$

Trong đó:

- SNR_{output} : SNR của tín hiệu đầu vào trước khi qua bộ lọc.
- SNR_{output}: SNR của tín hiệu sau khi qua bộ lọc.

Ý nghĩa:

- Nếu ΔSNR lớn, điều đó thể hiện bộ lọc thích ứng đã hoạt động hiệu quả.
- Đây là chỉ số rất quan trọng trong các hệ thống triệt nhiễu (anti noise).

2.2.2.6. Các thang đánh giá khác

$$\text{ERLE} = 10 \cdot \log_{10} \left(\frac{P_{echo_{in}}}{P_{echo_{out}}} \right) (\text{dB}) \quad (2.19)$$

3. THỰC HIỆN MÔ PHỎNG MATLAB TÌM CÁC THÔNG SỐ HIỆU QUẢ NHẤT CHO BỘ LỌC

3.1. Định hướng

Trong bất kỳ hệ thống xử lý tín hiệu số nào có độ phức tạp nhất định, đặc biệt là những hệ thống ứng dụng thuật toán thích nghi như LMS (Least Mean Squares), việc mô phỏng trên môi trường phần mềm trước khi hiện thực hóa trên phần cứng luôn là một bước bắt buộc, không chỉ nhằm mục đích kiểm tra chức năng mà còn đóng vai trò là giai đoạn tối ưu hóa thiết kế. Vì khi tổng hợp mạch (Synthesis), hoặc chỉ là mô phỏng dạng sóng mạch trên các EDA chuyên dụng thì thời gian thực thi thường rất lâu khoảng từ 30 phút đến vài tiếng, nếu ta không mô phỏng trước bằng Matlab thì thời gian hoàn thành đồ án sẽ bị kéo dài gây chậm tiến độ.

3.1.1. Tính cần thiết của mô phỏng trước triển khai phần cứng

Khi triển khai một thuật toán phức tạp như LMS trên nền tảng phần cứng số (cụ thể là FPGA), nhà thiết kế phải đối mặt với nhiều thách thức: từ việc kiểm soát độ chính xác số học, chi phí tài nguyên logic, độ trễ xử lý theo chu kỳ xung nhịp, đến tối ưu hóa mức tiêu thụ năng lượng. Trong khi đó, môi trường mô phỏng MATLAB lại cho phép người thiết kế thực hiện thử nghiệm linh hoạt, không bị ràng buộc bởi các giới hạn vật lý của phần cứng. Điều này cho phép mô hình hóa chính xác các thành phần toán học như vector trọng số, tín hiệu lỗi, tín hiệu nhiễu và tín hiệu gốc để đánh giá chất lượng lọc một cách định lượng thông qua các chỉ số như MSE (Mean Square Error), SNR (Signal to Noise Ratio), hoặc tốc độ hội tụ.

3.1.2. Vai trò của MATLAB trong quá trình thiết kế

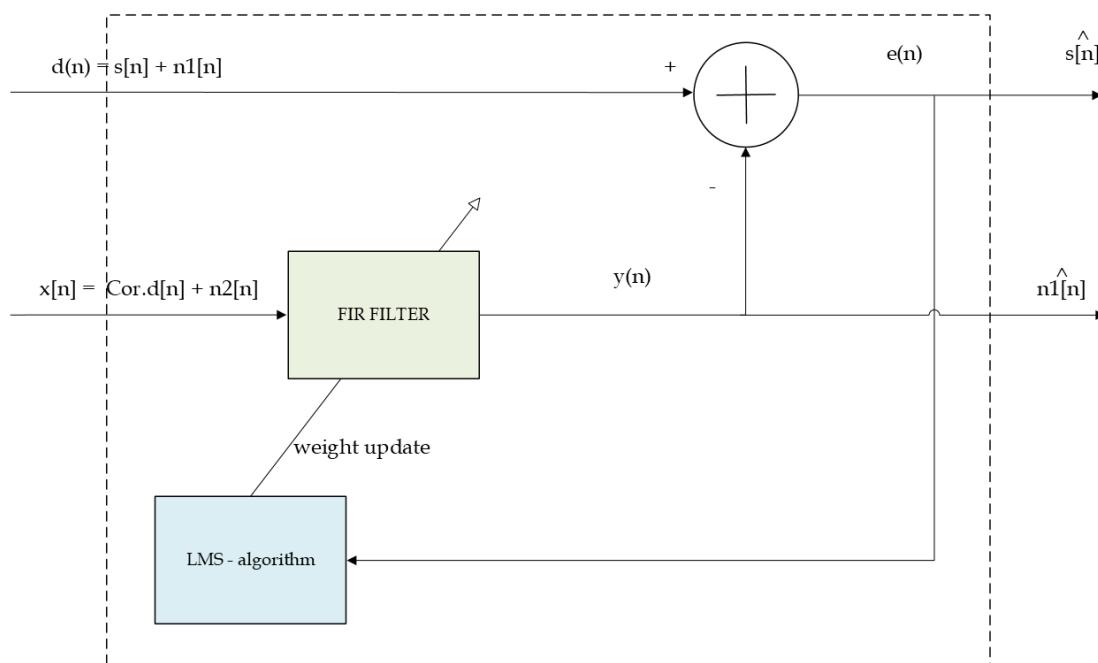
MATLAB được xem là một môi trường lý tưởng để kiểm tra lý thuyết hoạt động của hệ thống lọc thích nghi. Với khả năng biểu diễn dữ liệu dưới dạng vector ma trận, hỗ trợ trực quan hóa thời gian thực các tham số và dễ dàng tích hợp các mô hình tín hiệu thực tế, MATLAB giúp nhà thiết kế xác định chính xác tác động của từng yếu tố ảnh hưởng đến kết quả lọc. Đặc biệt, khi triển khai thuật toán LMS vốn là một thuật toán lặp có độ nhạy cao với learning rate việc mô phỏng trong MATLAB là điều kiện tiên quyết để xác định dải hội tụ ổn định và tránh các trạng thái phân kỳ không mong muốn.

3.1.3. Tối ưu hóa hiệu suất trước khi lập trình FPGA

Khác với môi trường phần mềm, nơi các thao tác có thể thực hiện tuần tự mà không lo đến tài nguyên vật lý, môi trường phần cứng như FPGA đòi hỏi việc lập trình phải phù hợp với số lượng LUTs, FFs, BRAMs và DSPs sẵn có. Một thuật toán chưa được kiểm chứng kỹ lưỡng có thể dẫn đến việc tiêu tốn không cần thiết tài nguyên logic, gây trễ hoặc thậm chí không thể tổng hợp (synthesize) thành công. Nhờ mô phỏng trước trên MATLAB, toàn bộ luồng dữ liệu, mô hình toán học, quá trình cập nhật trọng số, và phản hồi sai số có thể được kiểm nghiệm và tinh chỉnh kỹ càng để đảm bảo rằng thiết kế sau này khi được chuyển sang môi trường phần cứng sẽ hội tụ nhanh, tiêu tốn ít tài nguyên và có độ tin cậy cao.

3.2. Tìm hệ số μ khi thực hiện bộ lọc thích nghi cho các thành phần sóng cơ bản

3.2.1. Tạo các thành phần sóng ngõ vào



Hình 3-1: Sơ đồ ngõ vào ngõ ra của bộ lọc thích nghi dựa trên FIR và LMS

Dựa vào sơ đồ ở trên và phân lý thuyết em có đề cập, ngõ vào bộ lọc thích ứng sẽ gồm 2 ngõ vào một ngõ vào chính và một ngõ vào thứ cấp. Ngõ vào chính sẽ gồm tín hiệu gốc có ích bị cộng thêm một thành phần nhiễu $n1[n]$. Để sát với thực tế rằng nhiễu sẽ ảnh hưởng tới nhiều dải tần số nên ngoài việc nhiễu $n1[n]$ gồm thành phần sine tần số cao hoặc các thành phần sóng cơ bản có thể hiệu chỉnh mà em sẽ trình bày ở phần 4, em sẽ công thêm nhiễu trắng để đảm bảo nhiễu phân bố đều các dải tần số tạo nên một nền nhiễu.

Để sát với thực tế thay em sẽ cho nhiễu ở reference signal chỉ tương quan 1 phần nào đó so với primary signal và chèn thêm can nhiễu từ tín hiệu có ích và tín hiệu nhiễu ngoại lai $n2[n]$ hoàn toàn khác với nhiễu $n1[n]$.

```
% === 1. CẤU HÌNH TÍN HIỆU ===
```

```
fs = 48000,
```

```
N = 2024,
```

```
t = (0:N-1) / fs,
```

```
f_sine = 500,
```

```
s = 1 *sin(2 * pi * f_sine * t),
```

```
% === 2. TẠO NHIỆU ===
```

```
trắngian_noise = 0.73 * randn(1, N),
```

```
sine_noise = 0.5 * sin(4 * pi * 1000 * t),
```

```
n = trắngian_noise + sine_noise,
```

```
f_triangle = 350,
```

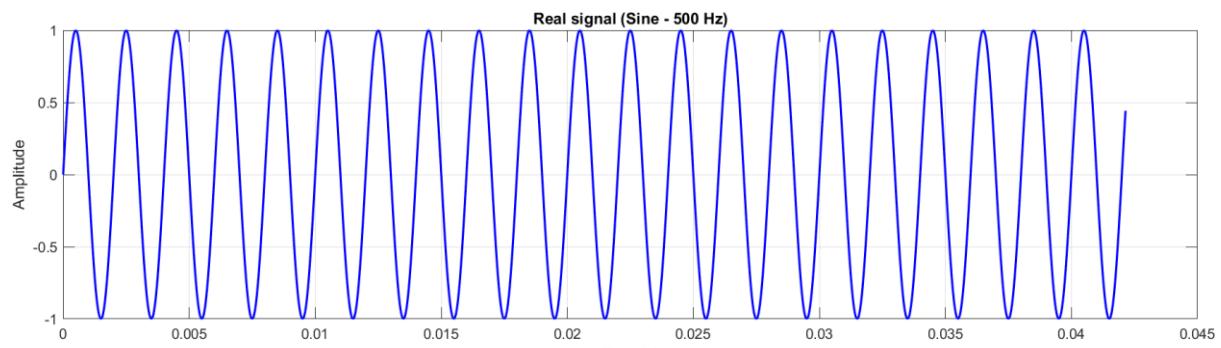
```
amplitude_triangle = 0.26, % Biên độ sóng tam giác
```

```
n2 = amplitude_triangle * sawtooth(2 * pi * f_triangle * t),
```

```
d = s + n,
```

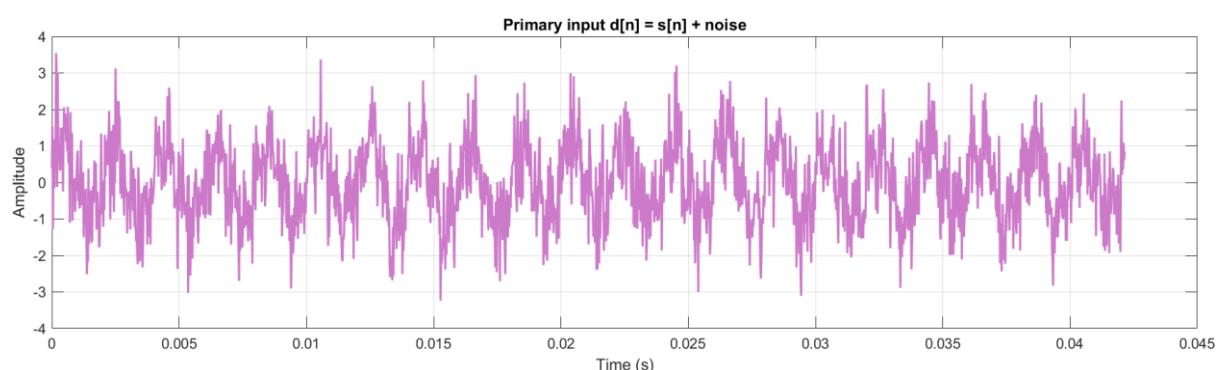
```
x = 0.7*n + 0.01*s + n2 ,
```

Từ code Matlab trên ta sẽ tạo được các thành phần tín hiệu đầu vào gồm có tín hiệu gốc có ích được thêm các thành phần nhiễu khác nhau để tạo thành 2 ngõ vào chính và thứ cấp như sau:

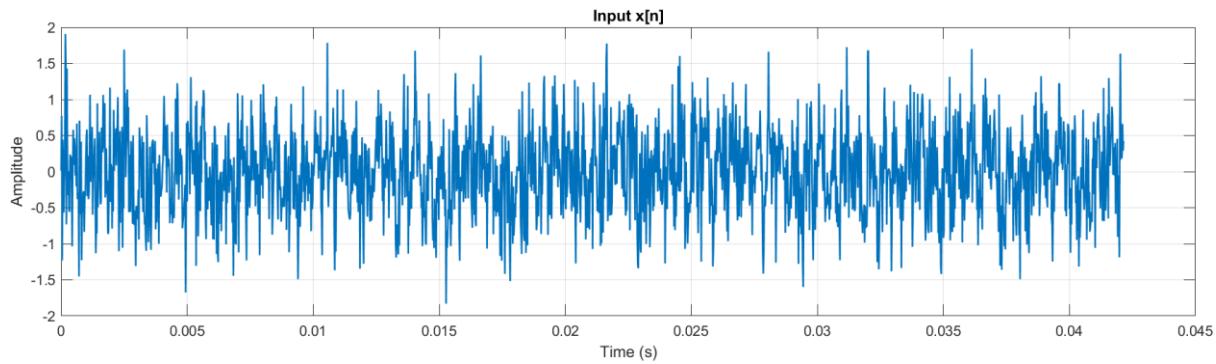


Hình 3-2: Tín hiệu sạch ban đầu

Tín hiệu có ích được chèn nhiễu tạo nên ngõ vào chính như sau



Hình 3-3: Dạng sóng ngõ vào chính



Hình 3-4: Dạng sóng của ngõ vào thứ cấp

Nhận xét: Ngõ vào bộ lọc thích ứng mất hoàn toàn dạng sóng sine ban đầu

Để tìm ra hệ số μ hợp lý nhất em sẽ bám theo các tiêu chí đánh giá hiệu quả lọc ở mục 2.2.2. Đầu tiên nếu xét về SNR, em sẽ dùng hàm có sẵn trong Matlab để tính toán cho ra được SNR trước và sau khi qua bộ lọc thích ứng. Với đầu vào ban đầu là $s[n]$ và được cộng nhiễu tạo nên $d[n]$ nên tỉ số tín hiệu trên nhiễu sẽ là

$$\text{SNR}_{\text{output}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise}}} \right) (\text{dB}) \quad (3.1)$$

Vì $d[n] = s[n] + n_1[n]$ nên nhiễu còn sót lại sẽ được tính như sau

$$\text{snr_before} = \text{snr}(s_cut, s_cut - d_cut),$$

SNR Trước lọc: -1.12 dB

3.2.2. Thực hiện bộ lọc LMS

Đầu tiên nếu xét lý thuyết ước lượng hệ số lọc (learning rate) với số tap là N như sau

$$\mu \lesssim \frac{1}{N \cdot P_x} \quad (3.2)$$

Trong đó:

- N: số hệ số (số tap) của bộ lọc thích ứng
- P_x : công suất trung bình của tín hiệu đầu vào thứ cấp $x(n)$, tức là:

Để ước lượng hàm trên em tính toán Matlab và cho ra kết quả như sau:

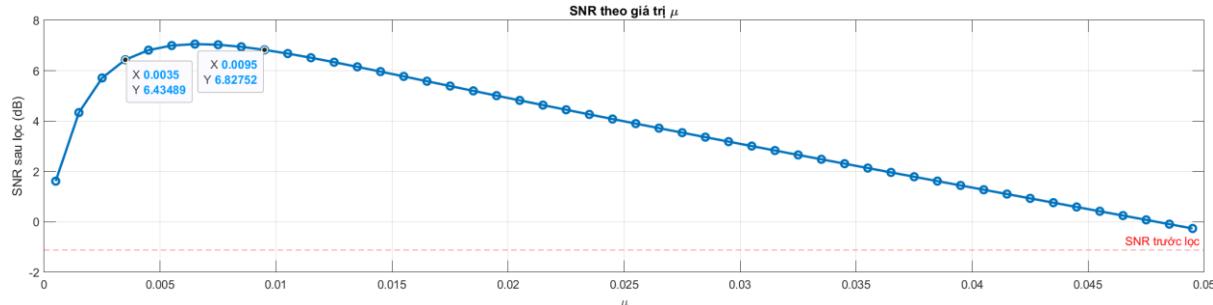
Ước lượng công suất đầu vào $P_x \approx 0.64519$

Giới hạn trên của μ theo lý thuyết: $\mu < 0.00484$

Tù ước lượng trên để chắc chắn hơn em sẽ lần lượt thay đổi biến μ và giữ nguyên các thành phần tín hiệu đầu vào để vẽ hàm SNR:

```
function [filtered_signal, w, noise_estimate] = LMS_filter(in, d, o, miu)
```

Cùng với kết quả tính toán SNR trước lọc là -1.12dB ta có biểu đồ hàm SNR theo biến μ như sau:



Hình 3-5: Độ biến thiên của SNR theo biến μ

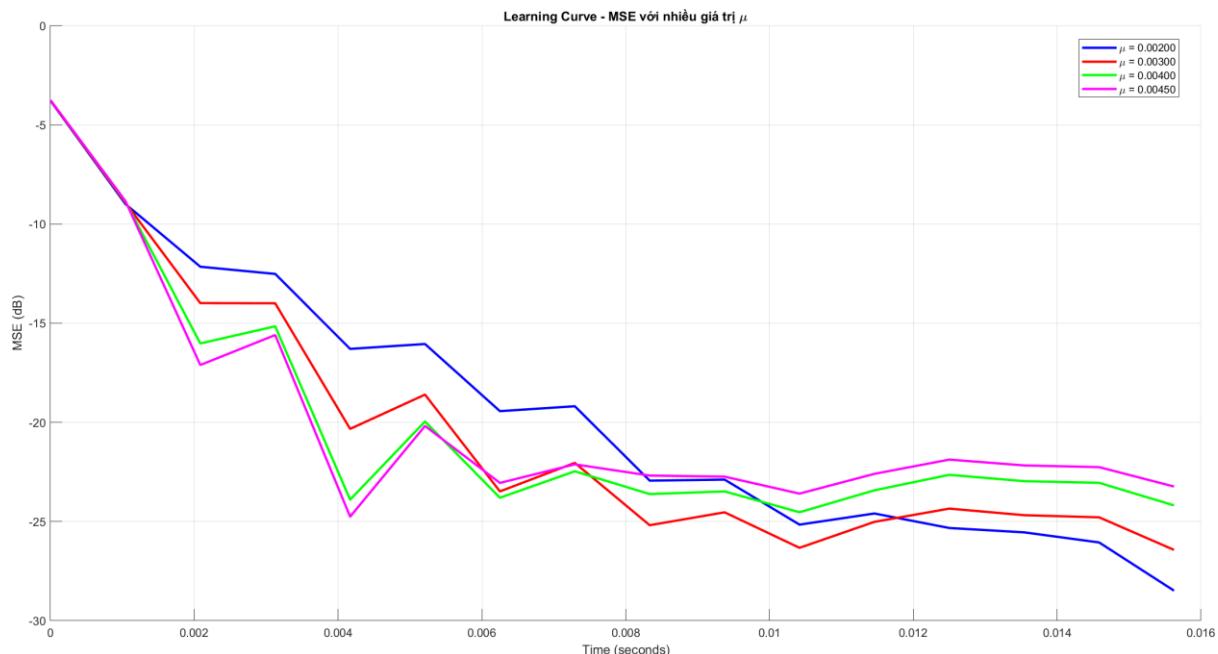
-Nhận xét: vậy nếu xét theo tiêu chí SNR càng cao càng tốt thì ta có thể chọn $0.0035 < \mu < 0.0095$

Tiêu chí Thứ 2 cần xét đến là MSE càng thấp:

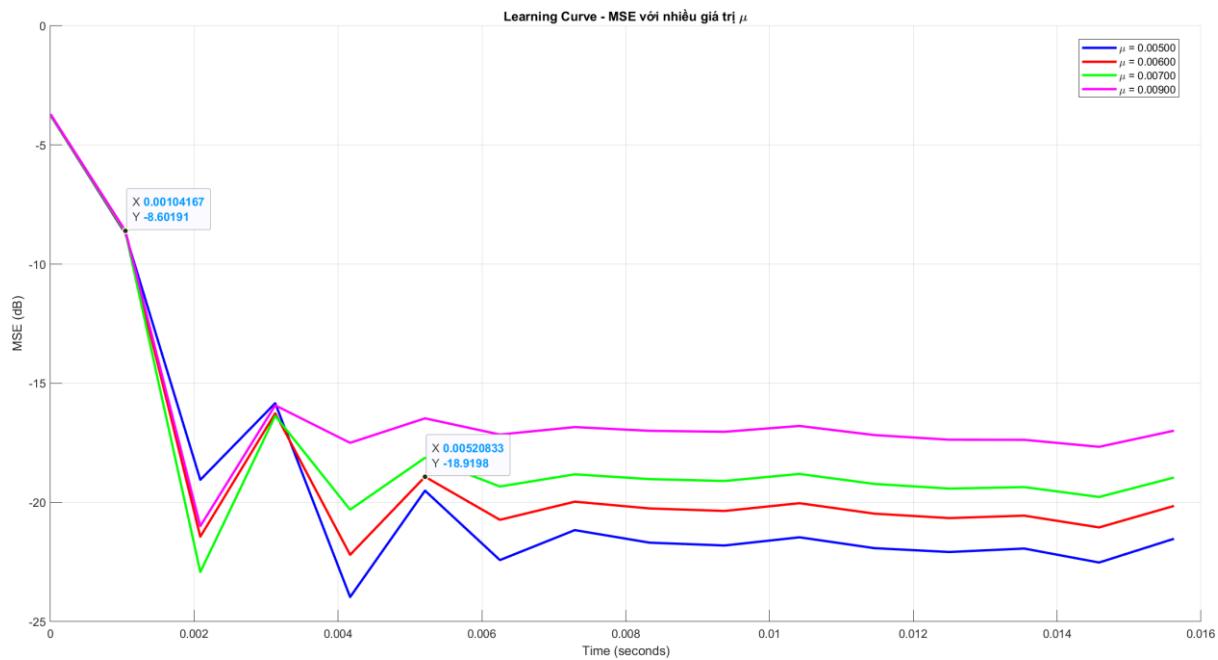
MSE ở đây đang xét là trung bình bình phương giữa tín hiệu gốc và tín hiệu ước lượng gốc

$$MSE = E(s[n] - e[n])^2 = \frac{1}{N} \sum_{n=0}^{N-1} (s[n] - e[n])^2 \quad (3.3)$$

Nếu $e[n]$ càng giống $s[n]$ thì MSE sẽ càng bé



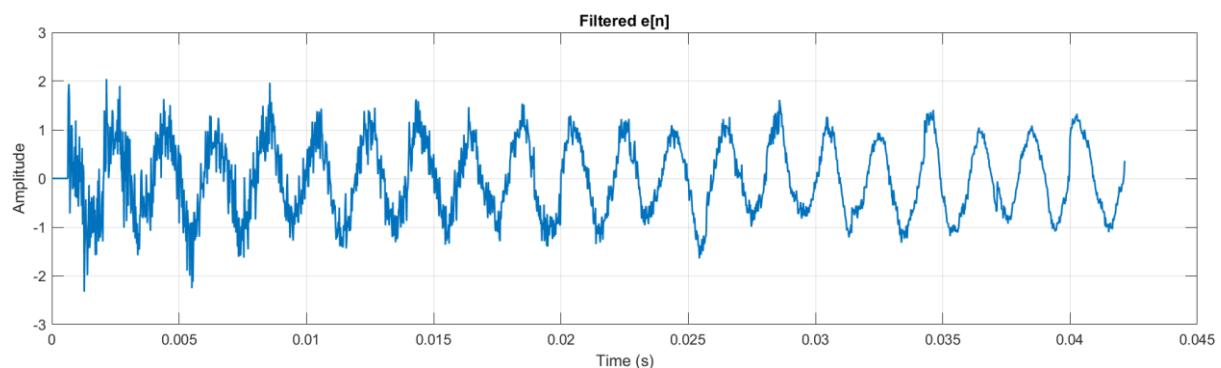
Hình 3-6: MSE giữa $s[n]$ và $e[n]$ khi $0.002 < \mu < 0.0045$

Hình 3-7: MSE giữa $s[n]$ và $e[n]$ khi $0.005 < \mu < 0.009$

Nhận xét: có thể thấy nếu xét về độ hội tụ theo lý thuyết là MSE sẽ không tăng mà giữ nguyên thì $MSE \mu = 0,005$ rất tốt khi so với $\mu < 0.0045$, nhưng nếu xét về MSE càng thấp càng tốt thì chọn $\mu = 0,0045$ an toàn hơn μ

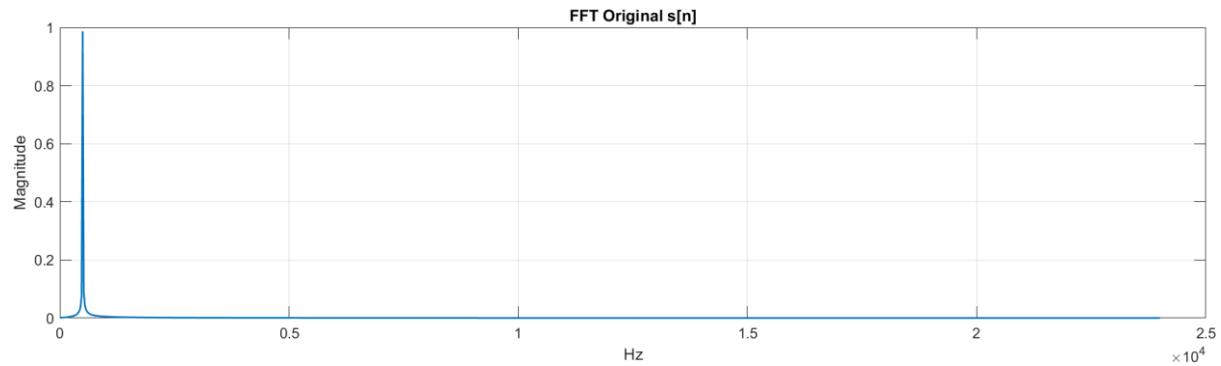
Kết luận từ 2 thang đánh giá trên em chọn $\mu = 0,0045$ để thực hiện lọc cả trên Matlab và khi implement lên FPGA.

3.2.3. Kết quả sau khi thực hiện lọc

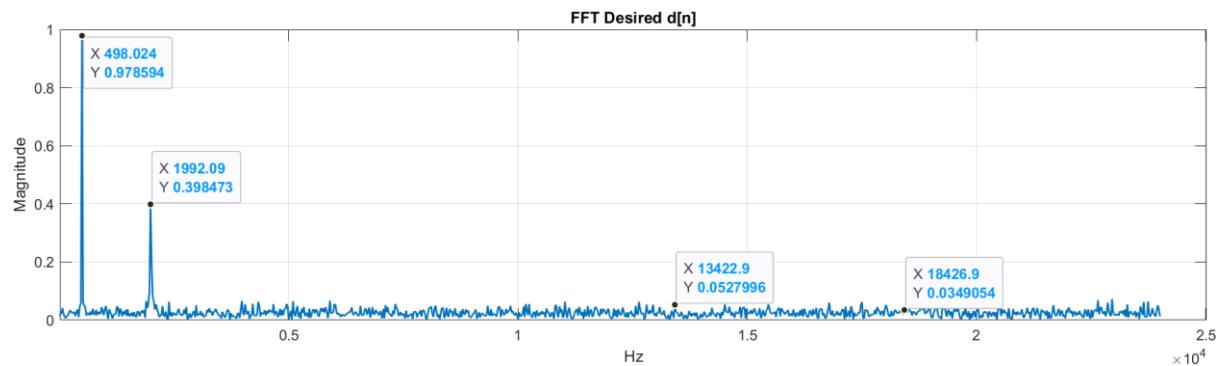
Hình 3-8: Dạng sóng ước lượng $e[n]$ sau lọc

Nhận xét: Dạng sóng sau lọc đã có biên độ phẳng hơn $d[n]$ rất nhiều. Dạng sóng $e[n]$ tại thời điểm $t < 0.015$ biên độ nhiễu nặng nên biên độ vượt qua khỏi mức biên độ 1V của sóng sine gốc, tại thời điểm $t > 0.015$ thì dạng sóng ổn định quanh 1V và xét về dạng sóng thì đã trông

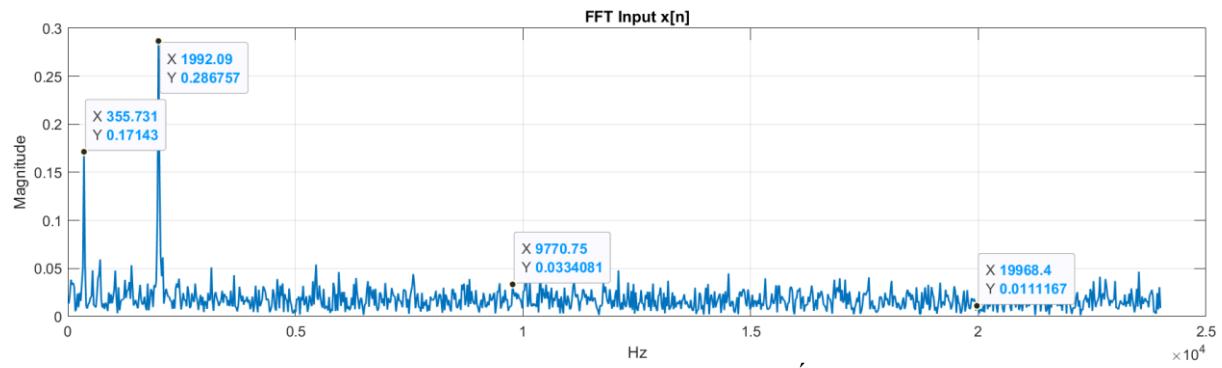
giống sóng sine hơn $d[n]$ ban đầu. Nhưng để rõ ràng hơn về mặt tần số sẽ em sẽ vẽ các FFT trước và sau để so sánh miền tần số của các thành phần tín hiệu trước và sau lọc:



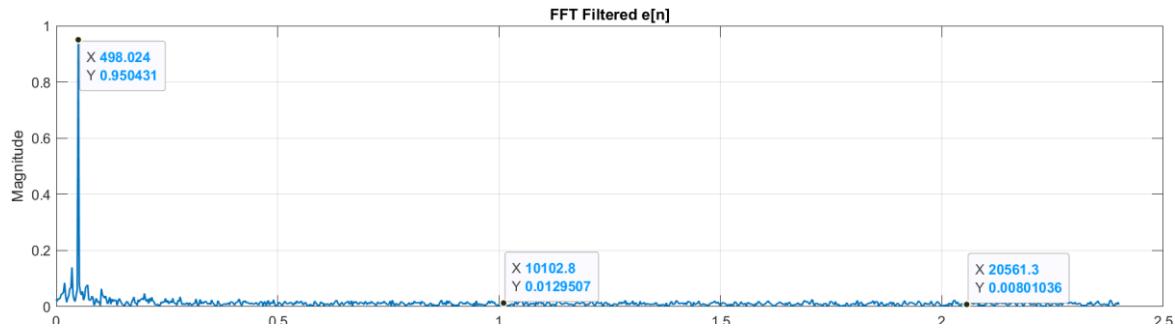
Hình 3-9: FFT ngõ vào gốc



Hình 3-10: FFT ngõ vào chính



Hình 3-11: FFT ngõ vào thứ cấp

Hình 3-12: FFT Ngõ ra ước lượng $s[n]$

Nhận xét: Đáp ứng tần số cho ra cho thấy nền nhiễu trắng đã được giảm đáng kể. Thành phần hài nhiễu tam giác ở $x[n]$ cũng biến mất. Sau khi qua bộ lọc thích ứng ngũ thành phần sóng sine có ích bị suy giảm đi một phần biên độ nhưng không đáng kể. Nếu xét trên đáp ứng tần số thì bộ lọc thích ứng thực hiện rất tốt.

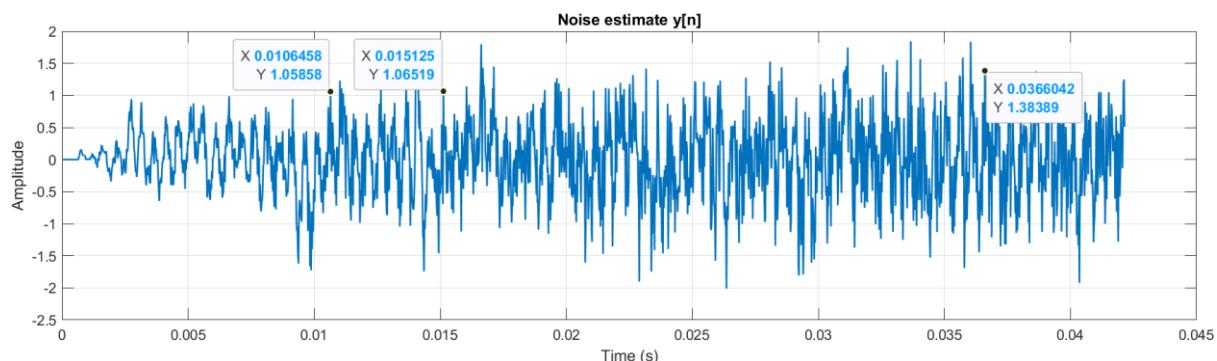
Phổ của $d[n]$ cho thấy có một thành phần phổ có tần số cao hơn tần số của sóng sine có ích và nằm trên nền phổ rộng do nhiễu trắng. Và phổ $x[n]$ có đỉnh tại nhiều điều hòa (tại vị trí phổ 2 kHz) và chùm hài của sóng tam giác (ở phổ 400 Hz), kèm nền rộng do trắng. Ở $x[n]$ còn có thành phần phổ bị can nhiễu từ tín hiệu có ích (tần số sine gốc). Nhưng sau khi lọc định của tín hiệu có ích vẫn được bảo toàn (biên độ chỉ suy hao rất nhỏ, cỡ vài phần trăm), Nền phổ nhiễu suy giảm đồng đều trên dải nhiễu, đỉnh nhiễu thành phần nhiễu điều hòa và các hài giảm đáng kể. Hình thái phổ này phản ánh MSE đã được hạ thấp và SNR sau lọc tăng rõ rệt. Với số tap và bước học đang dùng, kết quả đạt đến trạng thái hội tụ ổn định mà không xuất hiện dao động.

Đánh giá độ hội tụ: để đánh giá độ hội tụ ta cần tham chiếu 2 dạng sóng là ước lượng nhiễu và DeltaW vì:

$$W_{k+1} = W_k + \mu \frac{dJ}{dW} = W_k + \mu(2P - 2RW) \quad (3.4)$$

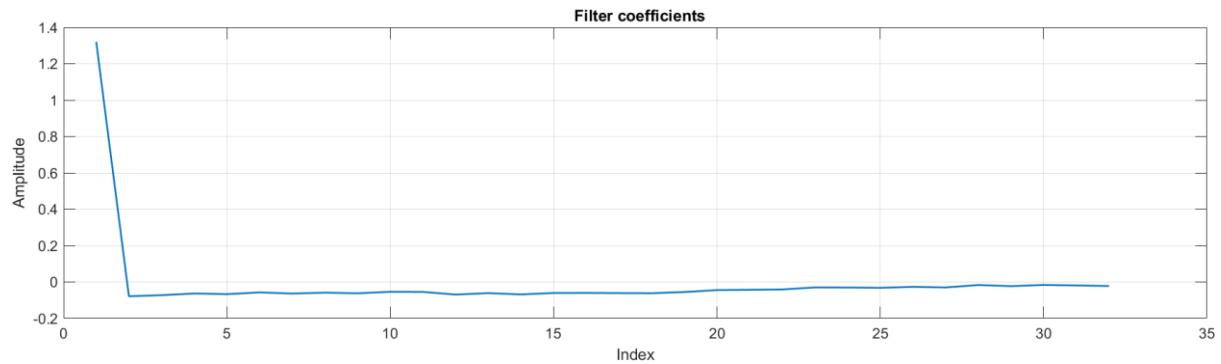
Gọi $\mu(-2P + 2RW)$ là Delta weight, là mức thay đổi của hệ số bộ lọc.

Khi khi J đạt tới W_{opt} thì $\nabla J = 0$ tương đương với phương trình trên W không còn cập nhật thành phần Delta nữa hệ số bộ lọc sẽ giữ nguyên và không tăng nữa.



Hình 3-13: Dạng sóng của ước lượng nhiễu

Nhận xét: ta có thể thấy ước lượng nhiễu tăng dần tới thời điểm $t=0.015$ thì mới gần tương đương với nhiễu $n1(n)$



Hình 3-14: Đồ thị biến thiên hệ số bộ lọc

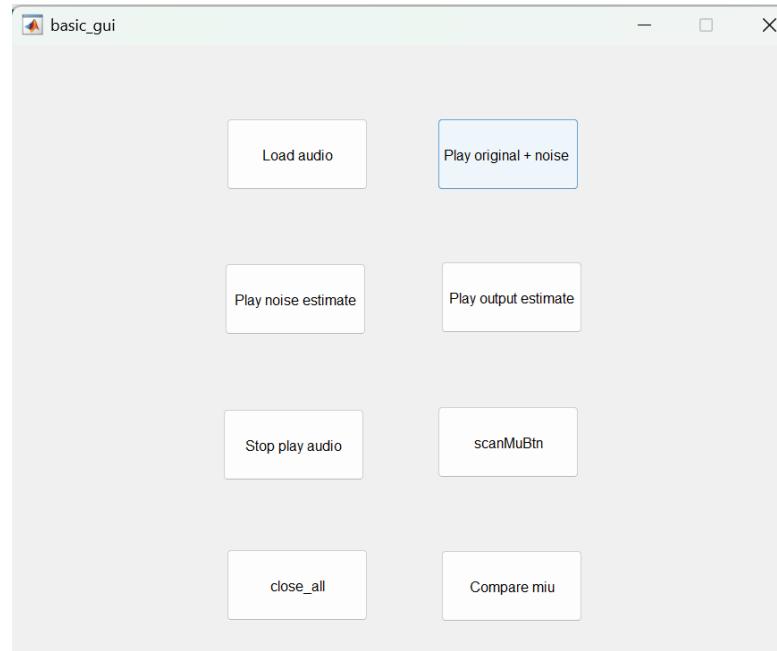
Khi đã tìm được hệ số bộ lọc thích hợp tại Tap đó rồi thì hệ số bộ lọc sẽ hầu như không tăng mà giữ nguyên.

3.3. Thực hiện mô phỏng tìm thông số bộ lọc thích nghi để lọc nhiễu âm thanh

3.3.1. Tạo GUI Matlab để mô phỏng

Tương tự phần mô phỏng các thành phần sóng cơ bản, điều đầu tiên ta cần thực hiện là lấy mẫu âm thanh bằng việc nạp (load) tệp audio có thể là .wav và chuẩn hóa tốc độ lấy mẫu về 48 kHz chuẩn cho tần số âm thanh.

Để thao tác thuận tiện khi kiểm thử, sửa lỗi và đánh giá theo thang DCR, em xây dựng một GUI cơ bản trong Matlab. GUI đóng vai trò mặt là nút nhát gọi các hàm: nạp file, trộn nhiễu, chạy LMS FIR, nghe lại các tín hiệu ở từng điểm đo, cũng như quét tham số để tìm miền hội tụ tốt.



Hình 3-15: Gui để thao tác với các file âm thanh

Các chức năng chính trên GUI (Hình 3-16) gồm:

- Load audio: Nạp file *.wav/*flac, sau đó tự động chuyển mono, chuẩn hóa biên độ và resample về 48 kHz. Lưu metadata để tái lập thí nghiệm.
- Play original và noise: Phát tín hiệu gốc đã trộn nhiễu (Trắngian + thành phần điều hòa/hài).
- Play noise estimate ($y[n]$): Phát ước lượng nhiễu tại ngõ ra bộ lọc thích nghi, giúp nghe trực quan thành phần ước lượng nhiễu bị trừ đi để cho ra ước lượng tín hiệu gốc.
- Play output estimate ($e[n]$): Phát tín hiệu ước lượng sạch sau khi triệt nhiễu.
- Stop play audio: Dừng phát tức thì, tránh treo kênh âm thanh khi nhấn nhiều nút bấm.
- scanMuBtn: Quét hệ số học μ trên một dải cấu hình, tính MSE, SNR, từ đó gợi ý miễn hỏi tự an toàn. Kết quả hiển thị dạng đồ thị kèm trị số khuyến nghị.
- Compare miu: So sánh nhanh hai cấu hình μ
- close_all: Đọn tài nguyên, đóng toàn bộ handle âm thanh và hình ảnh xuất ra, đảm bảo phiên làm việc sạch sẽ trước khi thử nghiệm mới.

Luồng xử lý: *Audio in → Resample/Normalize → Noise mixer → LMS-FIR ($y[n], e[n]$) → Playback/Plots/Metrics.*

Tương tự như mô phỏng lọc thành phần sóng cơ bản thực hiện tương tự với lọc nhiễu được trộn vào file âm thanh. Dựa vào sơ đồ cấu hình bộ lọc thích nghi ở trên và phân lý thuyết em có đề cập, ngõ vào bộ bộ lọc thích nghi sẽ gồm 2 ngõ vào một ngõ vào chính và một ngõ vào thứ cấp. Ngõ vào chính sẽ gồm tín hiệu gốc và bị cộng thêm một thành phần nhiễu $n1[n]$. Để sát với thực tế rằng nhiễu sẽ ảnh hưởng tới nhiều dải tần số nên để sát với thực tế nhất em ngoại trừ việc nhiễu $n1[n]$ gồm thành phần sine tần số cao sẽ cộng thêm nhiễu trắng để phân bố đều các dải tần số tạo nên một nền nhiễu. Để sát với thực tế em sẽ cho nhiễu ở reference signal chỉ tương quan một phần nào đó so với nhiễu ở ngõ vào chính và chèn thêm can nhiễu và tín hiệu nhiễu ngoại lai khác.

```
% ===== Tạo các thành phần =====
N = handles.N,
s = y(1:N).'; % s[n] – clean
t = 1:N,
n = 0.01*randn(1,N) + 0.1*sin(2*pi*(1/64)*t), % noise phụ
n1 = 0.09*randn(1,N) + 0.321*sin(2*pi*(1/128)*t), % primary noise
n2 = sawtooth(2*pi*400*(0:N-1)/handles.fs, 0.5) * 0.26, % triangle 400 Hz

d = s + n1, % d[n] = s + n1 (primary)
x = 0.1*s + 0.2*n + 0.7*n1, % x[n] (reference)
```

Để tìm ra hệ số μ hợp lý nhất em sẽ bám theo các tiêu chí đánh giá hiệu quả lọc ở mục 2.2.2

Đầu tiên nếu xét về SNR, em sẽ dùng hàm có sẵn trong Matlab để tính toán SNR trước khi qua bộ lọc thích ứng.

Với đầu vào ban đầu là $s[n]$ và được cộng nhiều tạo nên $d[n]$ thì tỉ số tín hiệu trên nhiễu sẽ là

$$\text{SNR}_{\text{output}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise}}} \right) (\text{dB}) \quad (3.5)$$

Vì $d = s + n1$ nên nhiều còn sót lại sẽ được tính như sau

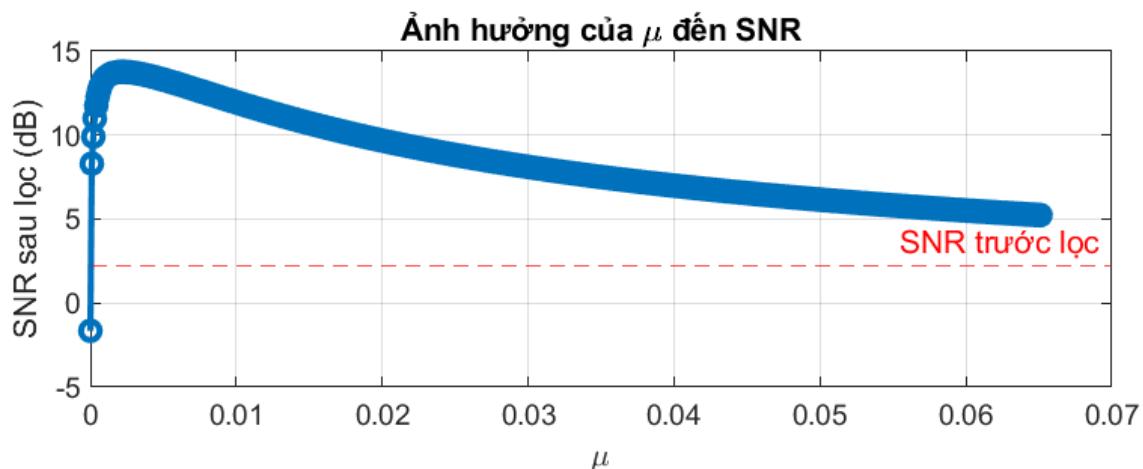
$$\text{snr_before} = \text{snr}(s_cut, s_cut - d_cut),$$

SNR Trước lọc: 2.20 dB

Ước lượng công suất đầu vào $P_x \approx 0.04866$

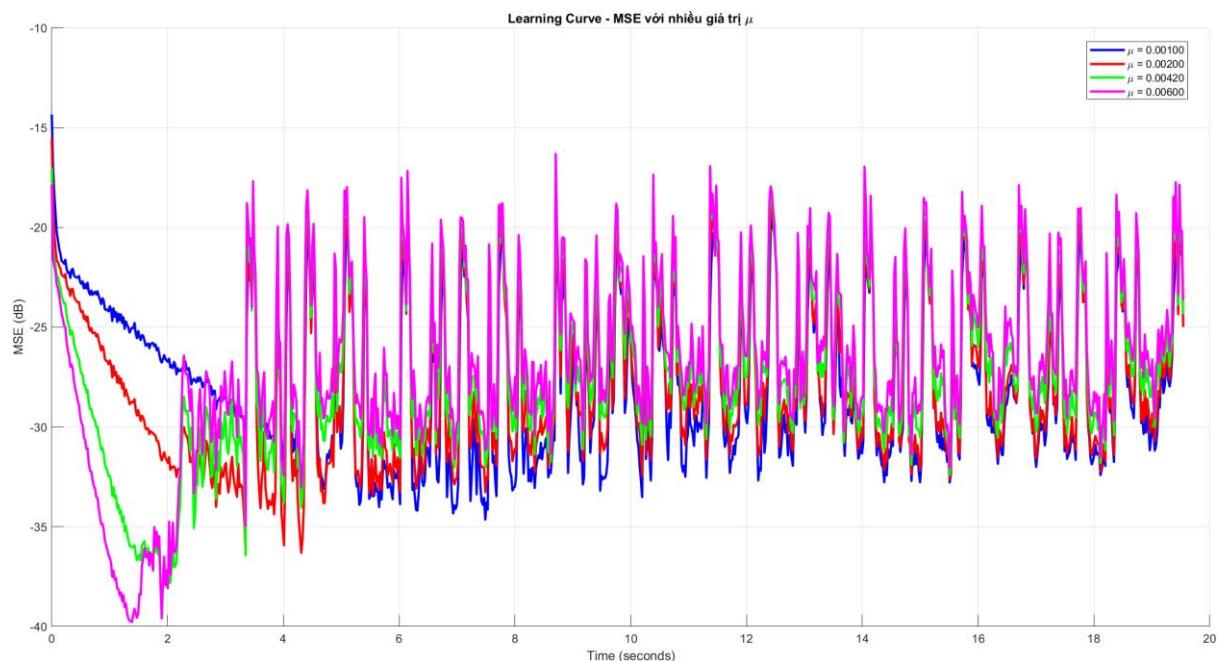
Giới hạn trên của μ theo lý thuyết: $\mu < 0.06422$

3.2.2. Thực hiện bộ lọc LMS với μ tốt nhất cho audio



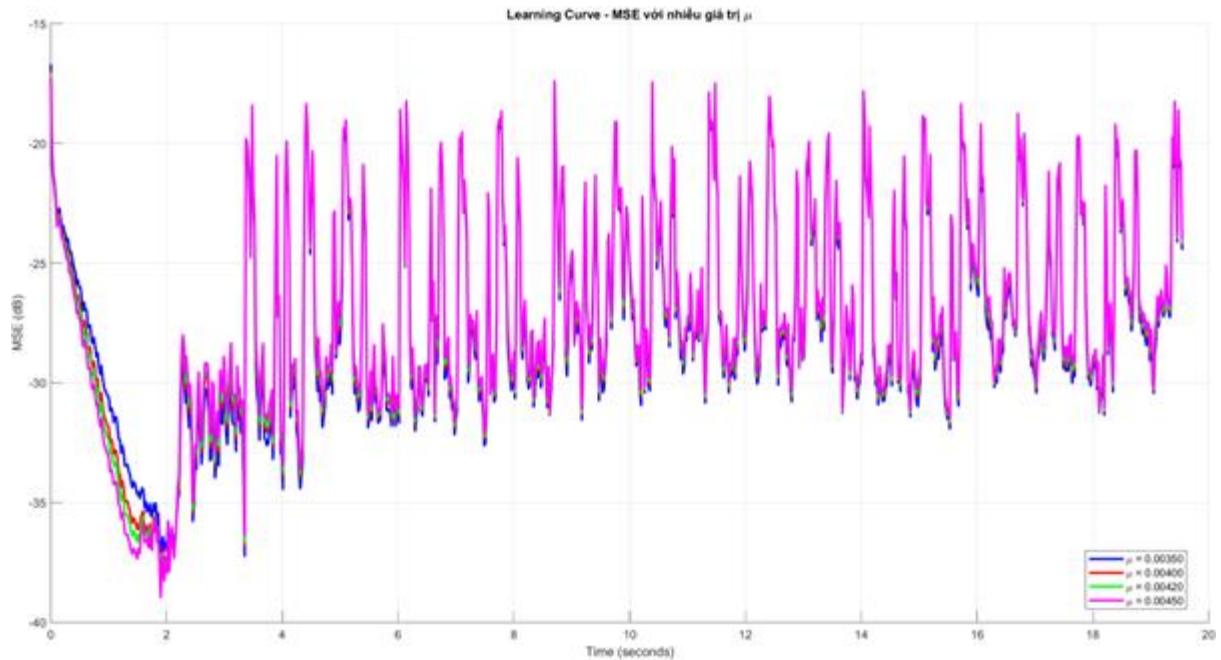
Hình 3-16: Hàm thay đổi SNR theo μ của audio

Nhận xét: Audio cần hệ số learning rate thấp hơn đáng kể so với ước lượng $\mu < 0.06422$ khoảng 0.001 thì mới cho ra SNR là cao nhất. Tiếp tục so sánh MSE với từng μ trong khoảng trên



Hình 3-17: MSE giữ $s[n]$ và $e[n]$ khi $\mu > 0.001$

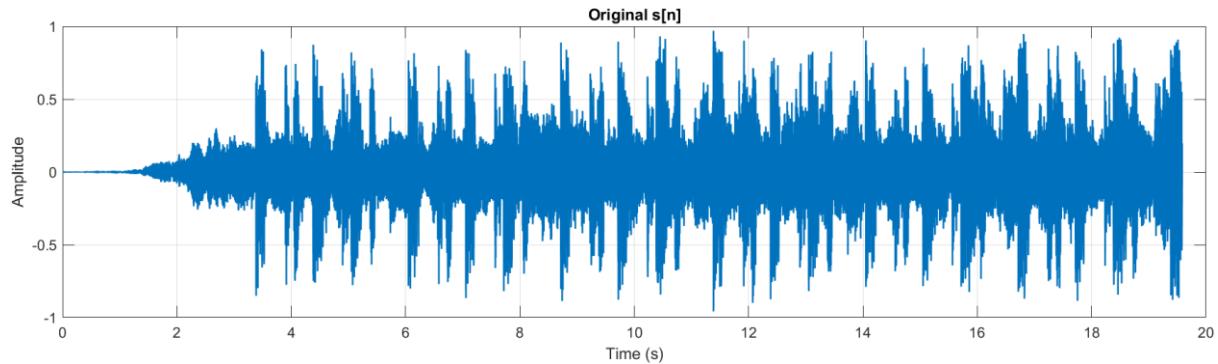
Nhận xét: có một vài sample nhiễu kiến cho MSE dạo động sau khi đã hội tụ nhưng MSE cao nhất vẫn khoảng -20dB được coi là chấp nhận được.

Hình 3-18: MSE với $\mu > 0.004$

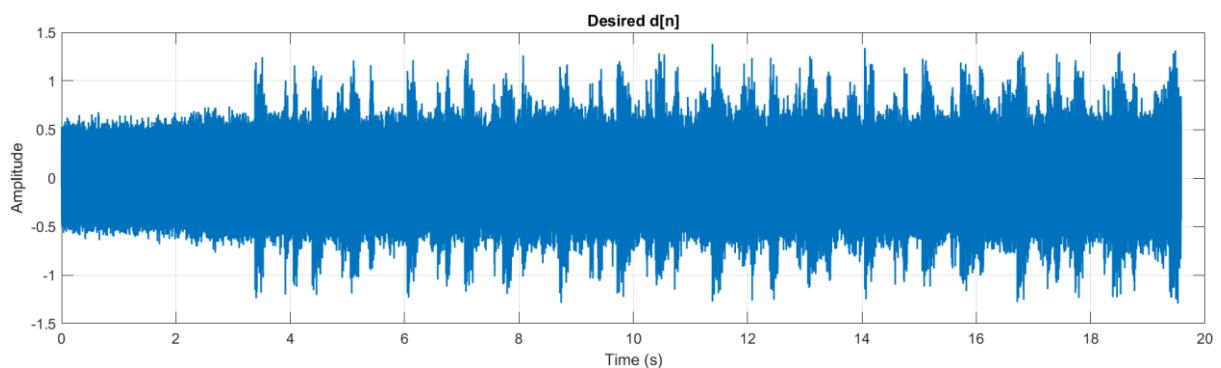
Khi càn tăng μ , min của μ dường như bị dâng lên không được tốt như $\mu < 0.001$

Do vậy để cân bằng giữa thang đánh giá SNR và MSE em chọn μ là:

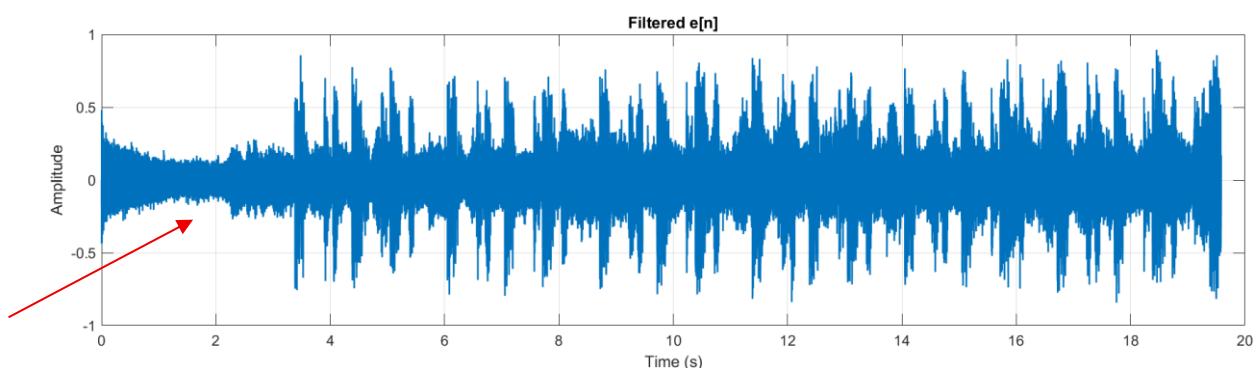
$$\boxed{\mu = 0.002}$$



Hình 3-19: Dạng sóng âm thanh gốc



Hình 3-20: Dạng sóng đầu vào chính



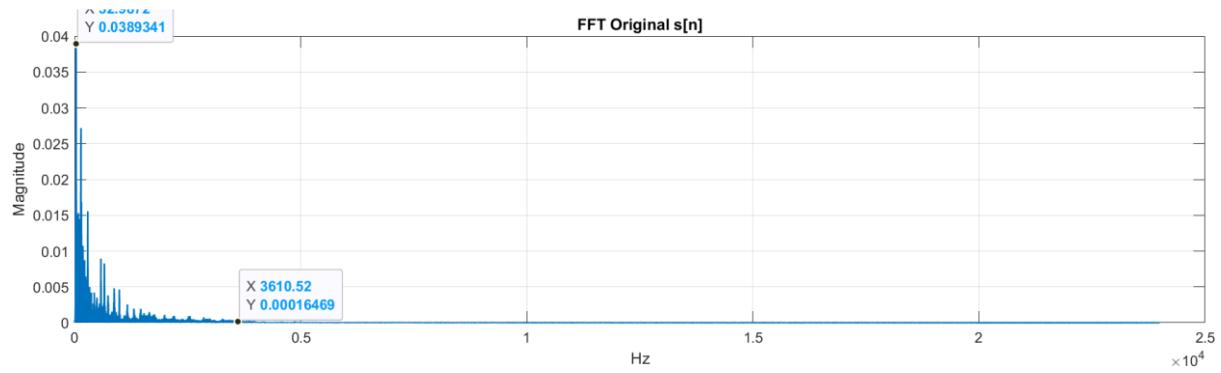
Hình 3-21: Dạng sóng ngõ ra ước lượng tín hiệu

-Tương tự với việc chèn nhiễu ở mô phỏng các thành phần sóng cơ bản để đảm bảo rằng nhiễu suất hiện ở nhiều tần số khác nhau nên em đã thêm nhiễu trắng trái đều ở các dài tần số

-Sau khi qua bộ lọc thành phần bộ lọc FIR sẽ tạo ra tín hiệu ước lượng tín hiệu nhiễu để từ đó nhiễu sẽ bị triệt tiêu dần khi ra khỏi ngõ ra ước lượng.

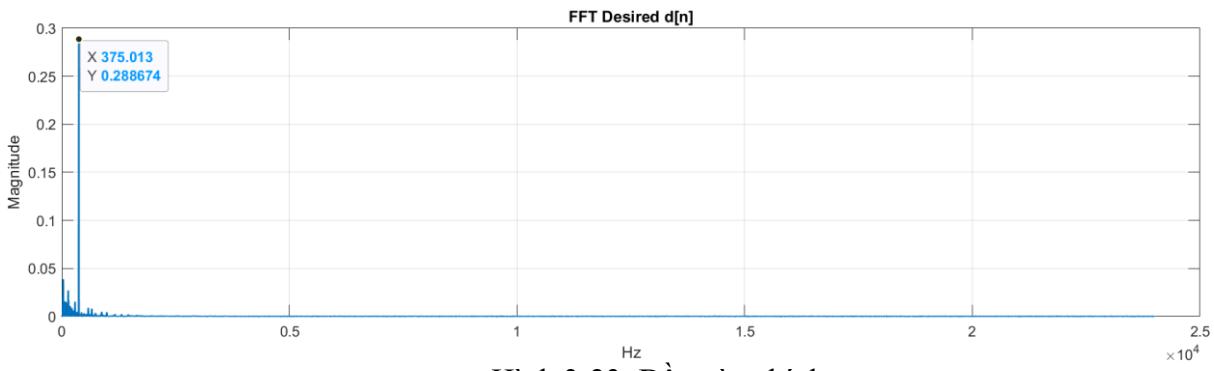
-Nhận xét: sau khi lọc biên độ nhiễu dần dần bị bóp, đến khi âm thanh gần tương đương âm thanh gốc, ta có thể kiểm chứng điều này bằng cách play estimate_audio ở gui. Nếu xét về

biên độ thì với $\mu = 0,002$ đang lọc rất tốt và không có dấu hiệu méo dạng. Để kiểm chứng rõ hơn thì ta xét FFT của các thành phần âm thanh.



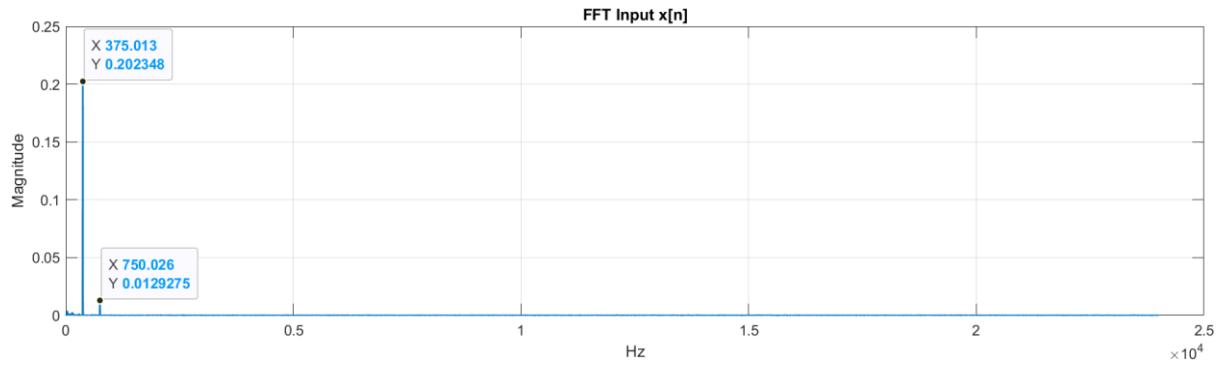
Hình 3-22: Phổ của âm thanh gốc

-Phân xét phổ của âm thanh gốc trải từ 0 tới gần 4Khz

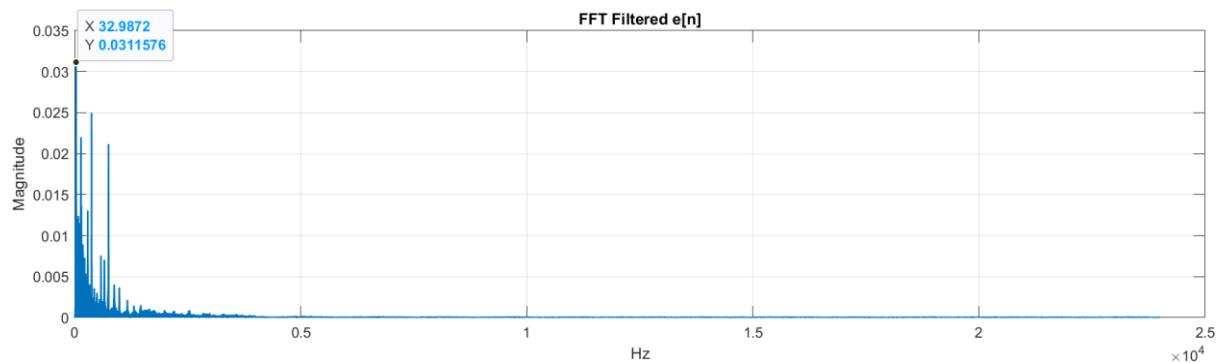


Hình 3-23: Đầu vào chính

-Nhận xét với đầu vào chính có 1 thành phần phổ nhiễu lớn hơn rất nhiều so với băng tần của âm thanh. Và các thành phần nhiễu trăng lẩn trong phổ của tín hiệu âm thanh gốc.

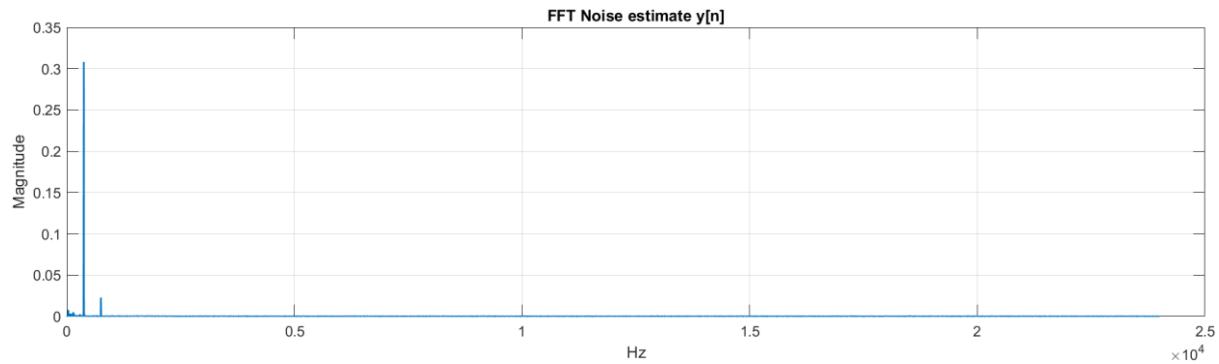


Hình 3-24: Đầu vào thứ cấp



Hình 3-25: Phổ của tín hiệu sau lọc

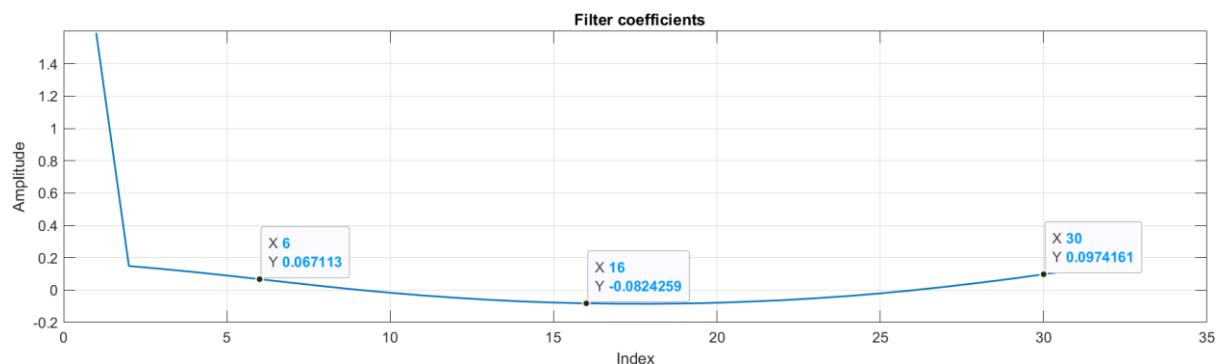
-Nhận xét: về dạng phổ của ước lượng tín hiệu gốc hầu như là được khôi phục nhưng vẫn còn có một vài thành phần nhiễu tam giác vẫn lẫn vào âm thanh, tạo thành hai phô hài với biên độ lớn rõ rệt.



Hình 3-26: FFT thành phần ước lượng nhiễu

-Nhận xét: về mặt biên độ lắn phô LMS ước lượng nhiễu gần đúng với 3 nhiễu thành phần chèn vào.

Về mặt tốc độ hội tụ.



Hình 3-27: Độ biến thiên Weight

-Nhận xét: Khi đạt đạt được hệ số bộ lọc thích hợp thì hệ số bộ lọc sẽ hầu như không tăng nữa đúng với lý thuyết delta weight hầu như cập nhật với số rất bé.

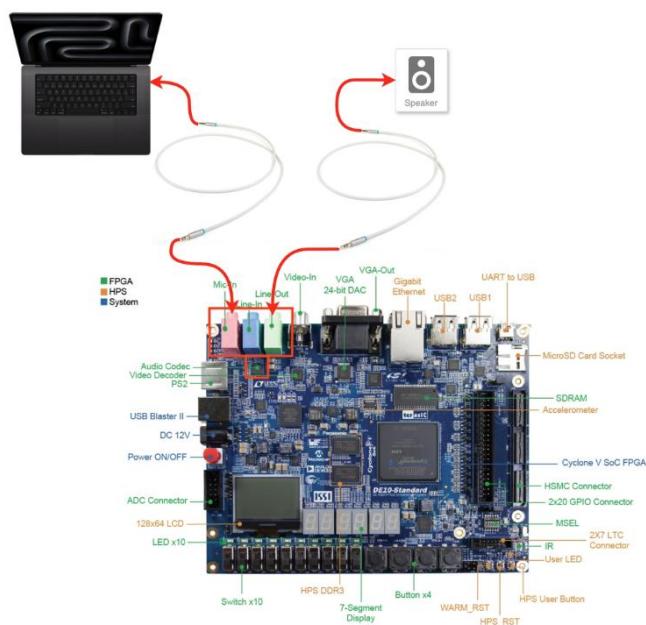
4. THIẾT KẾ PHẦN CỨNG BỘ LỌC THÍCH ÚNG LMS

4.1. Định hướng

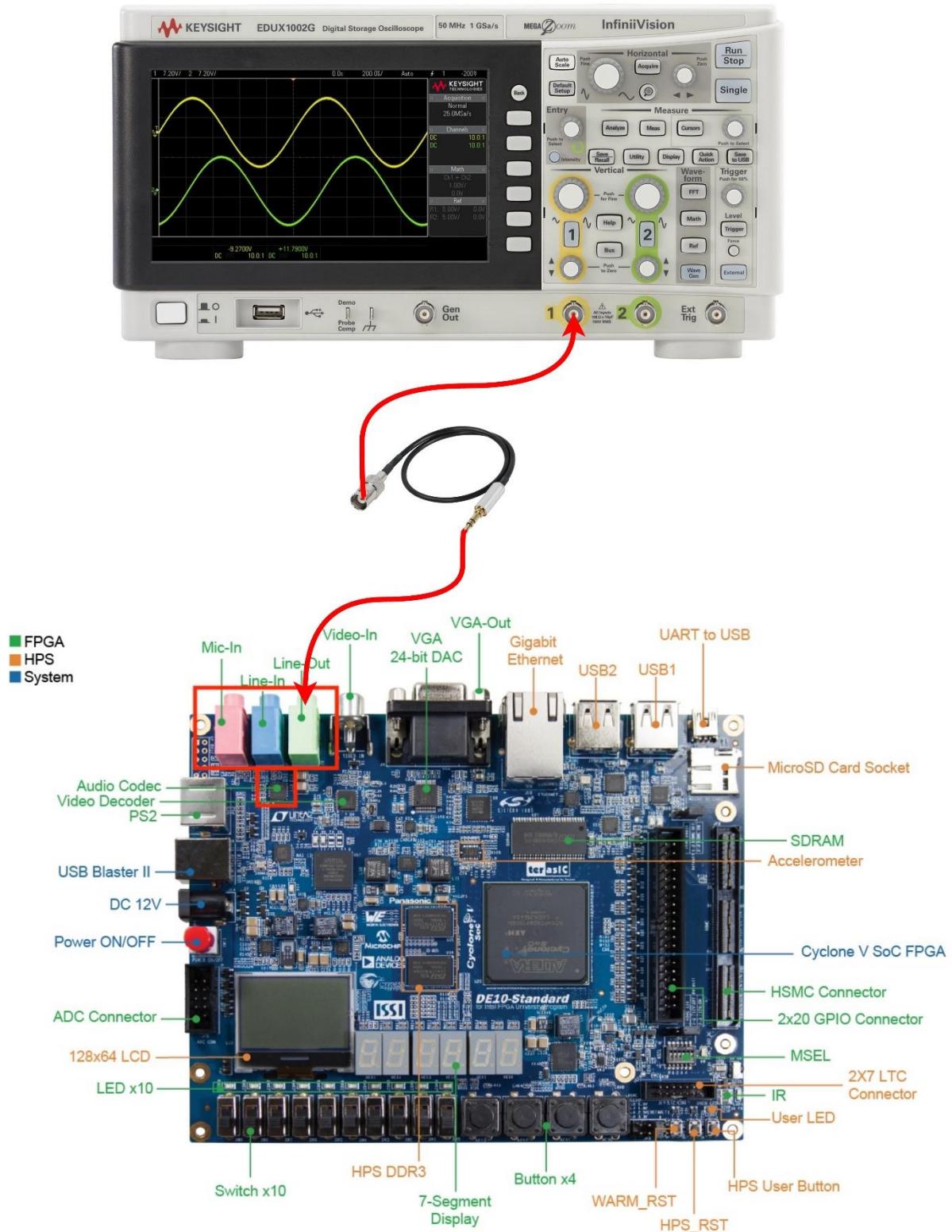
Trong lĩnh vực xử lý tín hiệu số (DSP), việc triển khai các thuật toán thích ứng như LMS (Least Mean Square) lên phần cứng là một xu hướng tất yếu nhằm đáp ứng yêu cầu ngày càng cao về tốc độ, độ trễ thấp và khả năng xử lý song song. Giữa nhiều lựa chọn về nền tảng phần cứng, FPGA (Field Programmable Gate Array) nổi bật là một giải pháp linh hoạt, mạnh mẽ và hiệu quả trong nhiều ứng dụng thời gian thực. Thực hiện trên FPGA cũng là một cách để kiểm thử hiệu quả thiết kế chip trước khi tới công đoạn tape out (sản xuất chip).

FPGA cho phép triển khai song song và pipelining các khối xử lý, giúp giảm thiểu độ trễ và tăng tốc độ xử lý đáng kể so với giải pháp trên vi điều khiển hoặc DSP truyền thống. Ngoài ra, người thiết kế có thể tùy chỉnh cấu trúc phần cứng theo đúng yêu cầu của thuật toán LMS từ hệ số learning rate, số lượng tap của bộ lọc, đến cách thức cập nhật trọng số theo từng chu kỳ xung CLK. Điều này giúp tối ưu hóa hiệu suất hoạt động cả về tốc độ và mức tiêu thụ tài nguyên.

Đặc biệt, FPGA có khả năng hiển thị và kiểm chứng đầu ra theo thời gian thực bằng cách xuất tín hiệu ngõ ra ngoài line out để đo dạng sóng bằng oscilloscope hoặc Audacity, từ đó giúp rút ngắn chu trình phát triển và kiểm thử hệ thống. Quá trình kiểm chứng này mang tính trực quan cao, dễ dàng phát hiện lỗi và đánh giá chất lượng đáp ứng tần số của bộ lọc thích ứng một cách chính xác.



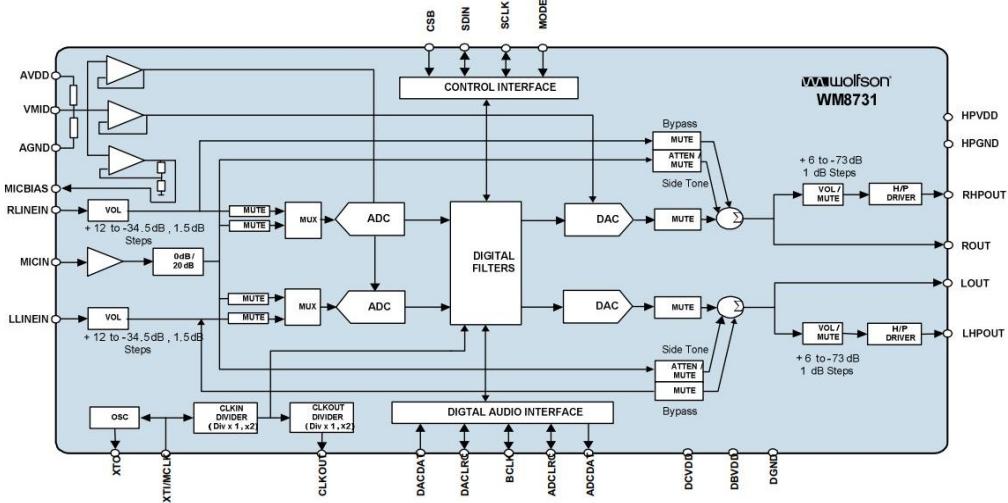
Hình 4-1: Kết nối line in vào laptop và line out ra loa



Hình 4-2: Kết nối FPGA với oscilloscope

4.2. SỬ DỤNG BỘ CHUYỂN ĐỔI ADC VÀ DAC TRÊN KIT FPGA

4.2.1. Giới thiệu bộ Audio Codec WM8731 trên DE-10



Hình 4-3: Block diagram của chip Codec (gồm cả ADC DAC)

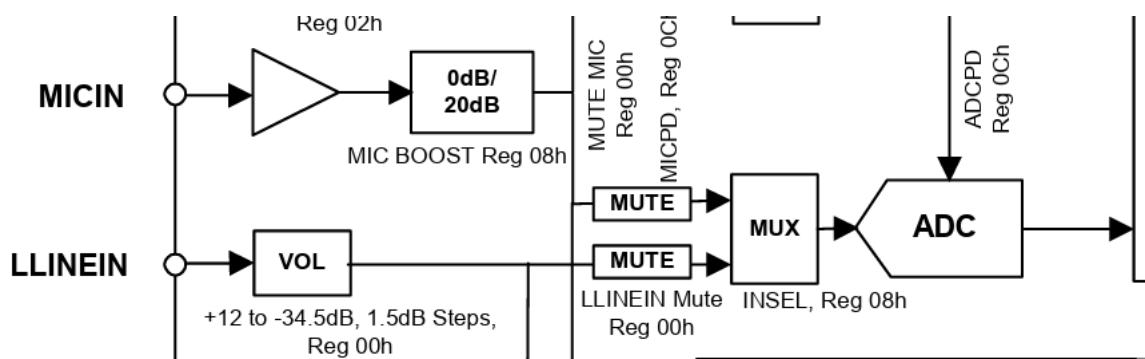
Vai trò: Là bộ chuyển đổi ADC/DAC audio tích hợp, hỗ trợ xử lý tín hiệu âm thanh stereo.

Ứng dụng: Thu/phát âm thanh qua giao tiếp I2S và cấu hình qua I2C/SPI.

Bộ WM8731 hỗ trợ nhiều chế độ ứng dụng:

- + ADC và DAC 24bit sử dụng kiến trúc sigma delta với bộ lọc số giúp đạt SNR cao và giảm nhiễu ngay từ ngõ vào.
- + Giao diện điều khiển thông qua giao tiếp 2 dây hoặc 3 dây (thường là giao thức I2C) cho việc truy cập đầy đủ các thanh ghi cấu hình (bao gồm volume, mute, deemphasis)
- + Giao diện dữ liệu âm thanh số: hỗ trợ nhiều định dạng (I2S, Left Justified, Right Justified, DSP mode) với độ dài dữ liệu từ 16 đến 24bit, và tần số lấy mẫu từ 8 khz đến 96 KHz. Từ đầu ra cổng line out 3.5 ở FPGA ta có thể kết nối với headphone, chức năng mute và hệ thống quản lý nguồn linh hoạt với nhiều chế độ tắt/ mở từng khối. Trong hệ thống của chúng ta, FPGA (trên kit De10 Standard) sẽ đảm nhận vai trò master cho cả giao tiếp điều khiển và truyền dữ liệu âm thanh đến WM8731 (slave).

Hạn chế của bộ codec WM8731 trên FPGA: Điểm hạn chế của bộ tích hợp ADC và DAC này trên FPGA là cùng một thời điểm chỉ được sử dụng một ngõ vào line in hoặc Mic in, dẫn tới khi em muốn kiểm thử hiệu quả của bộ lọc thích nghi không thể chèn nhiều thu được từ môi trường bằng Mic in để chèn vào tín hiệu đầu vào có ích do đó em phải tạo nhiều trực tiếp trên FPGA để chèn vào tín hiệu đầu vào có ích.



Hình 4-4: Hạn chế của bộ Codec WM8731

Bảng 4-1: Các chân quan trọng của WM8731

WM8731 Pin	Kết nối tới DE10	Chức Năng
SCLK	I2C_SCLK (I2C clock)	Clock I2C
SDIN	I2C_SDAT (I2C data)	Data I2C
MCLK(XTI)	AUD_XCK [3:0]	Clock chính codec (thường là 12MHz hoặc 18.432MHz)
BCLK	AUD_BCLK [3:0]	Bit Clock (I2S) 3.072MHz (48kHz × 64)
DACLRC	AUD_DACLRCK [3:0]	Left/Right clock cho DAC
DACDAT	AUD_DACDAT [3:0]	Dữ liệu I2S (phát), Data audio từ FPGA → WM8731

-Chân SCLK và SDIN cần điện trở pull-up $2.2k\Omega$ – $10k\Omega$ để đảm bảo tín hiệu I2C hoạt động ổn định. MCLK không đồng bộ: MCLK phải được tạo từ PLL của DE10 với tần số chính xác (ví dụ: 12.288MHz cho 48kHz sampling rate).

-Trong giao thức I2C, mỗi thiết bị được gán một địa chỉ 7bit (hoặc 10bit, nhưng với DE10 là 7bit). Khi DE10 muốn giao tiếp với WM8731, nó cần biết địa chỉ thiết bị để khởi tạo truyền dữ liệu, ở đây module Audio Codec WM8731 theo như Datasheet phần schematic thì địa chỉ I2C chọn WM8731 là slave với địa chỉ là 0x34 với chế độ chỉ ghi

-WM8731 chỉ hỗ trợ ghi dữ liệu cấu hình qua I2C (chế độ write-only) để cấu hình các thanh ghi bên trong. Vậy để chọn dùng module Audio Codec WM8731 trên DE10 thì ta cần truyền địa chỉ trên chân I2C_SDAT của DE10 với địa chỉ là 7bit:

$0x34$ (hex) = 0110100 (binary). Địa chỉ I2C được truyền dưới dạng 8bit ($0x68$), bao gồm 7bit địa chỉ và 1bit chọn R/W (=0) = 0110100 (binary).

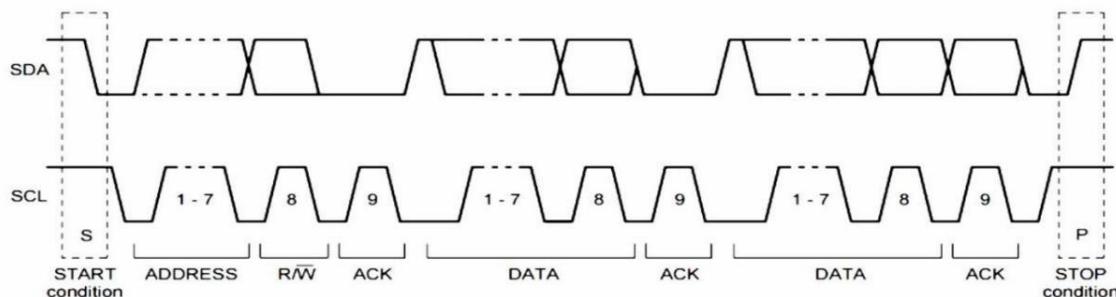
Cấu trúc lệnh ghi qua I2C

Để ghi giá trị vào thanh ghi của WM8731:

1. Start Condition: Bắt đầu giao tiếp.
2. Gửi byte địa chỉ: $0x68$ ($0x34 << 1 | 0$).
3. Gửi địa chỉ thanh ghi: 8bit (ví dụ: $0x02$ cho Left Headphone Volume).
4. Gửi dữ liệu: 8bit (ví dụ: $0x79$ để set volume).
5. Stop Condition: Kết thúc giao tiếp.

4.2.2. Thực hiện hóa phần cứng cấu hình Audio_codec

4.2.2.1. Module I2C_protocol



Hình 4-5: Waveform giao thức I2C

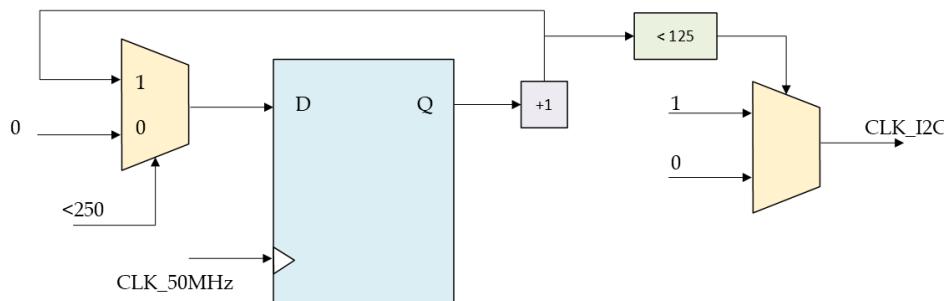
-Khi sử dụng FPGA DE10 để cấu hình WM8731 qua giao thức I2C, tần số CLK của I2C nên được thiết lập ở mức 400 kHz. Đây là tần số tối đa được hỗ trợ cho giao tiếp 2 dây MPU serial interface của WM8731, tương đương với chuẩn I2C tốc độ cao (Fast mode).

- ❖ Standard Mode: 100 kHz (phổ biến nhất).
- ❖ Fast Mode: 400 kHz (FCLK I2C tối đa của DE10).

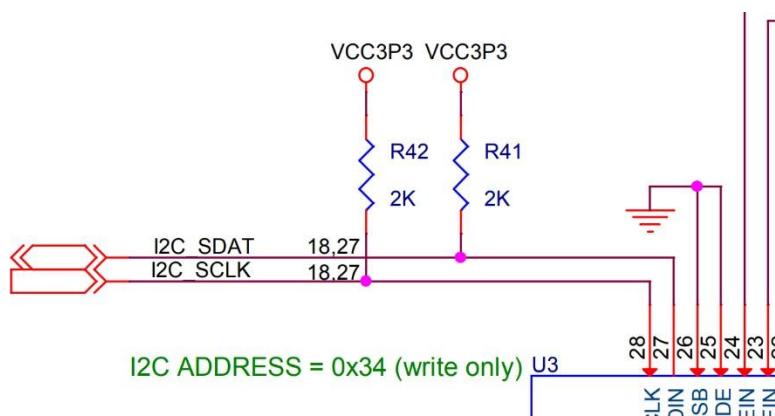
-Chọn tần số hoạt động: 400 kHz hợp lệ vì ta có thể sử dụng bộ chia để scale CLK của DE10 từ 50Mhz xuống 400kHz

```
// generate two clocks for i2c and data transitions
//Counting from 0 to 249 (a total of 250 values), the counting period is calculated as 250 x 20ns (50MHz) = 5μs, resulting in a frequency of 200kHz.
//The duty cycle is 50%, with the high state lasting for 125 counts and the low state for 125 counts.
always @(posedge clk_50Mhz) begin: SCALE_50M
    if      ( clk_counter < 250 )  clk_counter <= clk_counter + 1;
    else    clk_counter <= 0;
end

always @(posedge clk_50Mhz) begin: CLK_I2C_50_duty_cycle
    if      ( clk_counter < 125 )  clk_i2c <= 1'b1;
    else    clk_i2c <= 1'b0;
end
```



Hình 4-7: Bộ chia tần số



Hình 4-6: Các i/o I2C và địa chỉ I2C của WM8731

- Module I2C_protocol triển khai giao thức I2C để truyền 3byte dữ liệu (24bit liên tiếp gồm 7bit địa chỉ, 1bit chỉ đọc ghi, 2byte data) đến slave (ở đây là WM8731) và thêm 3bit ack để nhận điện đúng sai kênh truyền. Thiết kế dựa trên điện trở kéo lên và open drain, tuân thủ chuẩn I2C.

- Do 2 chân I2C nối với WM8731 với 2 điện trở kéo lên do đó khi muốn truyền 1 thì ta cần thả nối tín hiệu (Z), khi muốn về không thì ta nối bit 0, đảm bảo đúng phàn cứng vật lý I2C với điện trở kéo lên

- Bit 1 (HIGH): FPGA thả nối chân I2C (High-Z). Điện trở kéo lên bên ngoài sẽ kéo điện áp lên 3.3V (mức logic 1).
- Bit 0 (LOW): FPGA chủ động kéo xuống GND (mức logic 0).

- Điện trở kéo lên (bên ngoài): Giữ sclk/sdin ở mức 1 khi không truyền.

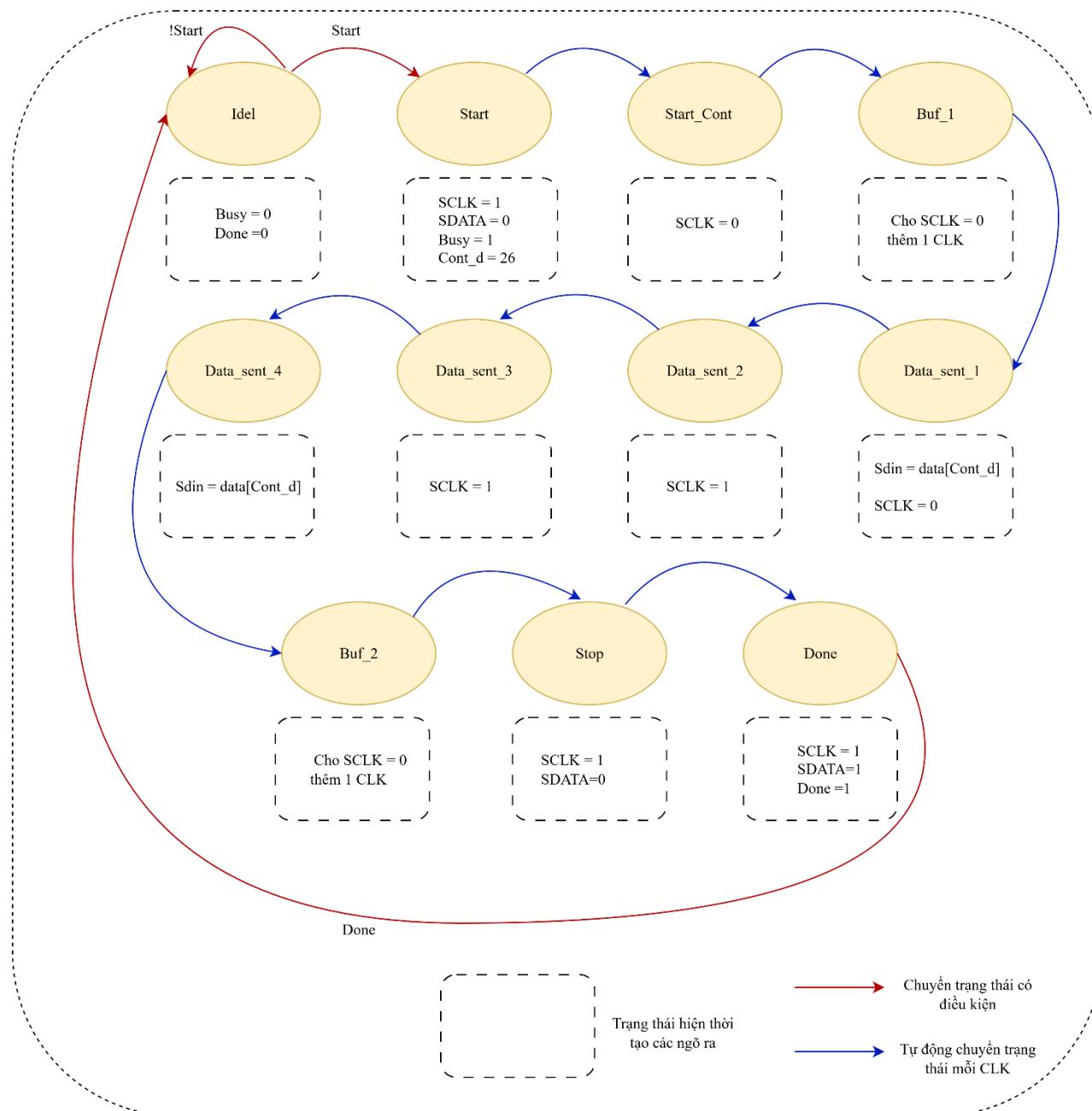
```
assign sclk = (sclk_tmp) ? 1'b1 : 1'b0;
assign sdin = (sdin_tmp) ? 1'b1 : 1'b0;
```

- Như vậy để cấu hình bộ Audio codec bằng giao thức I2C bằng cách tạo module giao thức I2C trực tiếp trên FPGA DE10 cần một số chân I/O như sau ở bảng 4-2.

Bảng 4-2: Bảng các i/o của module giao thức I2C

Port Name	Direction	Bit Width	Mô tả
clk	Input	1	Xung nhịp hệ thống (DE10: 50MHz).
reset_n	Input	1	Tín hiệu reset tích cực mức thấp (0: reset, 1: hoạt động).
start	Input	1	Kích hoạt quá trình truyền I2C (1: bắt đầu).
addr	Input	7	Địa chỉ 7bit của thiết bị I2C slave (WM8731: 0x34).
wr_rd	Input	1	Chế độ ghi/đọc (0: ghi, 1: đọc) nhưng WM8731 chỉ hỗ trợ ghi (0).
data_st	Input	8	Byte dữ liệu đầu tiên FIRst(thường là địa chỉ thanh ghi cấu hình).
data_nd	Input	8	Byte dữ liệu thứ hai second (giá trị cấu hình).
busy	Output	1	Báo hiệu module đang bận (1: đang truyền, 0: sẵn sàng).
done	Output	1	Báo hiệu truyền hoàn tất (1: xong, 0: đang xử lý).
sclk	Inout	1	Chân xung clock I2C (SCL), sử dụng open drain với điện trở kéo lên.
sdin	Inout	1	Chân dữ liệu I2C (SDA), sử dụng open drain với điện trở kéo lên.

Sơ đồ chuyển trạng thái của Module lái giao thức I2C



Hình 4-8: Sơ đồ chuyển trạng thái của Module lái giao thức I2C

1. IDLE:

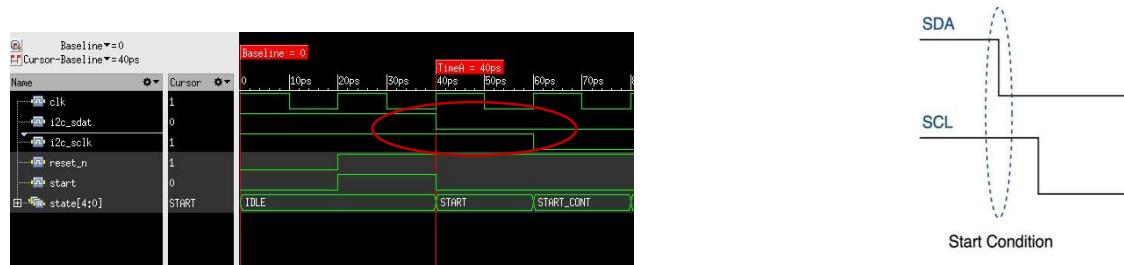
Chờ tín hiệu start.

SCL và SDA ở mức cao (thả nôi, kéo lên).

2. START:

Bắt đầu điều kiện Start: SCL cao, SDA xuống thấp.

3. START_CONT:



Hình 4-9: Testbench điều kiện bắt đầu

Hoàn thành Start: SCL và SDA đều thấp.

4. BUF_1:

- Chuẩn bị truyền bit đầu tiên.

- Bật cờ busy.

5. DATA_SEND_1 tới DATA_SEND_4:

- Để truyền 27bit gồm có 7bit địa chỉ, 1 R/W, 3ACK, 2byte data thì ta dùng biến đếm count_q để trả giá trị vào data[26:0], sau khi truyền 1bit đi bộ đếm giảm 1 đơn vị.

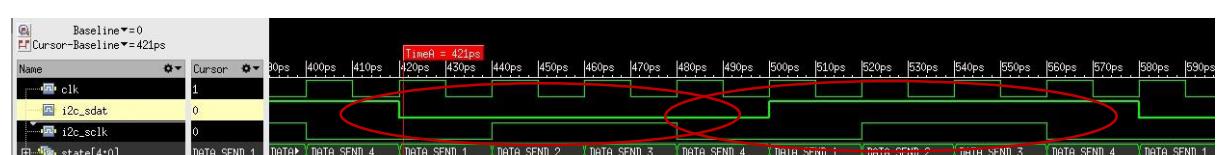
- Để slave có thể lấy mẫu data đúng timing khi tín hiệu SDAT đã ổn định, thì xung SCLK tích cực cao phải ở giữa xung SDAT truyền 1bit. Do đó để truyền 1bit SDAT lên I2C_SDAT ta chia làm 3 state để kéo dài xung SDAT làm sao cho SCLK vào giữa SDAT:

DATA_SEND_1: Đặt SDA theo bit dữ liệu (MSB trước), SCLK tích cực thấp.

DATA_SEND_2: SCLK lên tích cực cao, giữ SDA ổn định.

DATA_SEND_3: Giữ SCLK tích cực cao.

DATA_SEND_4: SCLK xuống tích cực thấp, giảm biến đếm count_q.



Hình 4-10: Testbench Module I2C

Như mô phỏng thì phần tích cực cao của SCLK nằm giữa SDAT để đảm bảo slave lấy mẫu chính xác.

6. BUF_2:

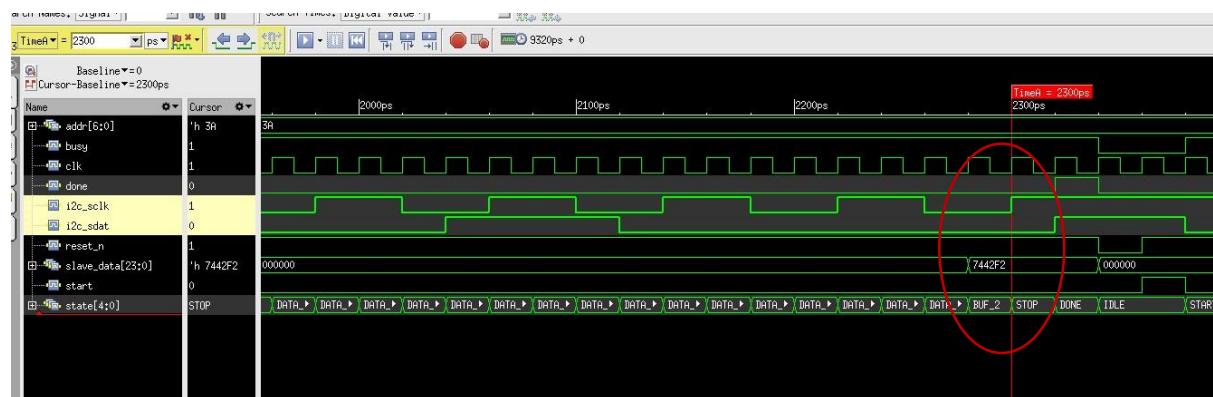
-Chuẩn bị tạo điều kiện Stop.

-Phục hồi biến điêm

7. STOP:

-Điều kiện Stop: SCLK tích cực cao và SDA lên cao.

-STOP condition: SDAT từ tích cực thấp sang tích cực cao khi SCLK đang tích cực cao.



Hình 4-11: Testbench điều kiện kết thúc

-Hoàn tất, trả bus về trạng thái idle.

Testbench và Debug

```
// Check if the slave received the correct data
$display("DEBUG DATA FROM I2C: Expected = %h, Received = %h", {test_addr, 1'b0}, test_data_st, test_data_nd);
assert (slave_data == {test_addr, 1'b0, test_data_st, test_data_nd});
else $error("!!!!!!Test failed: Incorrect data received by slave!!!!!!!");
end
endtask

// Test sequence
initial begin
    $display("!!!!!!!!!!!!!!Starting I2C Testbench.....!!!!!!!");
    // Test case 1
    i2c_test(7'h3A, 8'h42, 8'hF2);
    // Test case 2
    i2c_test(7'b0011010, 8'h13, 8'hFF);
    $display("!!!!!!All tests passed successfully!!!!!!!");
    $finish;
end

xcelium> source /opt/cadence/XCELIUM2009/tools/xcelium/files/xmsimrc
xcelium> run
!!!!!!!!!!!!!!Starting I2C Testbench.....!!!!!!
DEBUG DATA FROM I2C: Expected = 7442f2, Received = 7442f2
DEBUG DATA FROM I2C: Expected = 3413ff, Received = 3413ff
!!!!!!All tests passed successfully!!!!!!
Simulation complete via $finish() at time 9280 PS + 0
..../01_tb/testbench_i2c_protocol.sv:97      $finish;
xcelium> exit
```

Hình 4-12: Testbench truyền I2C

4.2.2.2. Module config_codec

Chức năng: Nối trực tiếp với module I2C_protocol để thực hiện cấu hình thanh ghi trong WM8731.

-Địa chỉ slave WM8731: 0x34 (7bit).

-Dữ liệu ghi: 16bit (7bit địa chỉ thanh ghi + 9bit dữ liệu).

Ta chạy I2C protocol với tần số clk cấp vào là 200kHz được tạo từ chia tần số FPGA 50Mhz.

Bảng 4-3: Bảng cấu hình thanh ghi

STT	Thanh Ghi	Giá trị (Hex)	Mô tả
1	R15	0x1E00	Reset toàn bộ thanh ghi.
2	R6	0x0C10	Power Up: Bật EX, OUTPD, và các khối cần thiết.
3	R2	0x0579	Âm lượng kênh trái (Left Volume): 0dB, không mute.
4	R3	0x0779	Âm lượng kênh phải (Right Volume): 0dB, không mute.
5	R4	0x0812	Tắt ADC, bật DAC, bypass MIC.
6	R7	0x0E08	Định dạng audio: 24bit, MSB-FIRst, DSP mode.
7	R8	0x1001	Chế độ USB, MCLK = 12MHz, tần số mẫu 48kHz.
8	R9	0x1201	Kích hoạt giao tiếp (Active Mode).
9	R6	0x0C00	Tắt OUTPD để xuất âm thanh ra loa/headphone.

Bảng 4-4: I/O Ports module cấu hình thanh ghi

Port	Direction	Mô Tả
clk	Input	Xung nhịp hệ thống (VD: 50MHz).
reset_n	Input	Reset tích cực mức thấp. Khi reset_n = 0, module trở về trạng thái ban đầu.
busy	Input	Tín hiệu từ I2C Master, báo hiệu đang truyền dữ liệu.
is_config	Input	Tín hiệu kích hoạt quá trình cấu hình WM8731.
done_config	Output	Báo hiệu cấu hình hoàn tất.
ack_I2C	Output	Xác nhận ACK từ slave (WM8731).
wr_rd	Output	Chế độ ghi (0) hoặc đọc (1). Ở đây luôn là 0 do chỉ ghi.
addr	Output [6:0]	Địa chỉ I2C của WM8731 (7bit). WM8731 có địa chỉ 0x34 (7'b0110100)

addr_reg	Output [7:0]	Byte cao của dữ liệu gửi, chứa địa chỉ thanh ghi WM8731.
data_config	Output [7:0]	Byte thấp của dữ liệu gửi, chứa giá trị cấu hình thanh ghi.

State diagram

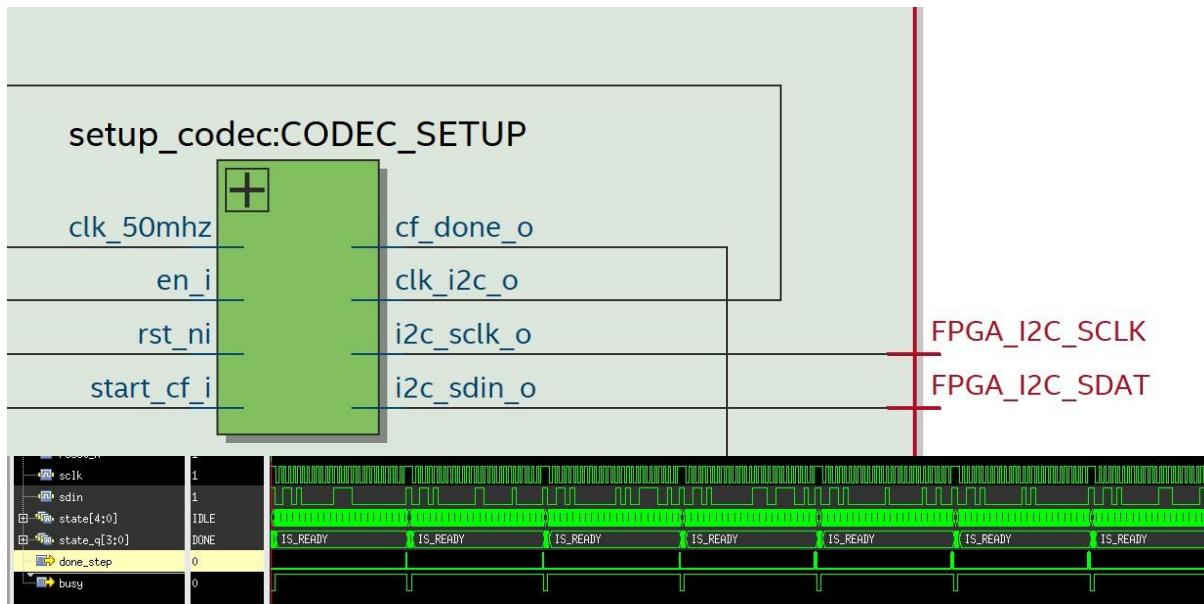
- IDLE: Chờ tín hiệu busy = 0 để bắt đầu.
- IS_CONFIG: Kiểm tra is_config = 0 để chuyển trạng thái.
- SEND_1 và SEND_2: Gửi lần lượt 2 byte (địa chỉ thanh ghi + giá trị).
- IS_READY: Chờ I2C Master sẵn sàng.
- SEND_LAST: Quyết định gửi thanh ghi tiếp theo hoặc kết thúc.
- DONE: Báo done_config = 1.

Biến đếm count_q để đếm số lượng thanh ghi đã cấu hình như trong bảng cấu hình thanh ghi ở trên.

Module Setup_codec

- Top module của 2 module trên. Module có nhiệm vụ điều chế xung I2C_clk 200kHz từ clk của FPGA DE10 là 50Mhz bằng bộ chia tần số.
- Setup_codec sẽ nối chân 2 module trên và nối chân với phần cứng bên ngoài là 2 chân I2C trên DE10.

Testbench and debug



Hình 4-13: Testbench truyền câu hình bằng I2C

4.2.3. Thực hiện hóa phần cứng giao tiếp P2S với Audio_codec

Module p2s_buffer_rjm_codec

-Chức năng: Chuyển đổi dữ liệu âm thanh 24bit từ song song sang nối tiếp (P2S) theo chuẩn Right-Justified Mode (RJM) cho codec WM8731.

-Ứng dụng: Xuất âm thanh digital qua giao tiếp I2S (hoặc RJM) đến codec WM8731 trên board DE10.

-Right-Justified Mode (RJM): là một định dạng truyền dữ liệu âm thanh số, là một chế độ của I2S, dữ liệu 24bit được căn lề phải trong khung 32bit

Ví dụ: Với dữ liệu 24bit, trong khung $\frac{1}{2}$ Fs clock cho 1bit, 24bit âm thanh sẽ nằm ở vị trí 24bit LSB).

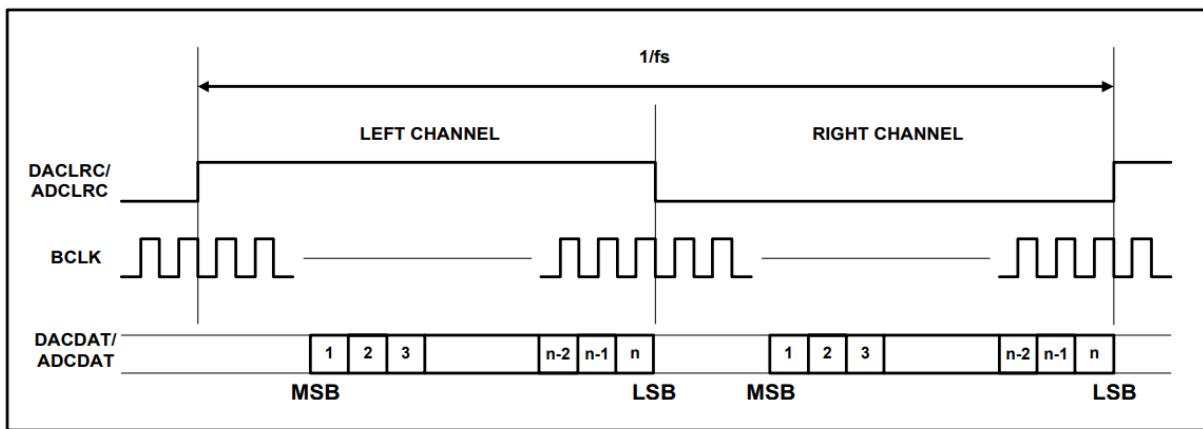


Figure 28 Right Justified Mode

Hình 4-14: Chế độ hiệu chỉnh phải lấy mẫu ADC

Giải Thuật**Khởi tạo:**

Khi `load_i = 1`, dữ liệu 24bit `parallel_data_i` được nạp vào thanh ghi dịch (PISO).

Tạo CLK:

$BCLK: f = \frac{clk_i}{2}$ (6MHz với clk_i từ PLL= 12MHz).

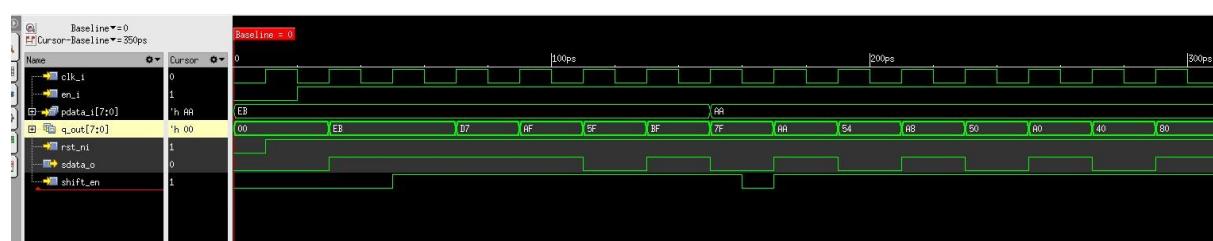
$LRCK: f = \frac{clk_i}{256}$ sáp xỉ 48kHz (chuẩn I2S từ clk_i chia 256).

Chuyển đổi từ song song sang nối tiếp:

Dữ liệu được dịch ra từng bit trên sườn xuống của BCLK.

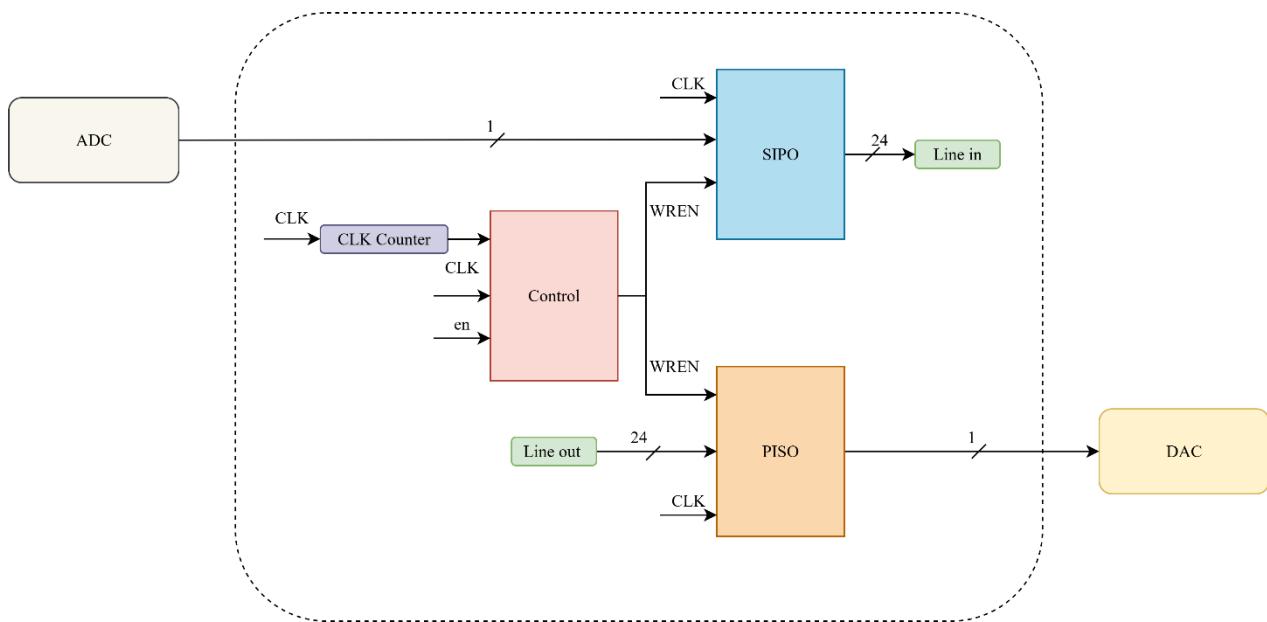
PISO dịch dữ liệu ra `serial_data_o` trên sườn xuống của BCLK.

Right Justified: Dữ liệu 24bit nằm ở 24bit cuối của khung 32bit.

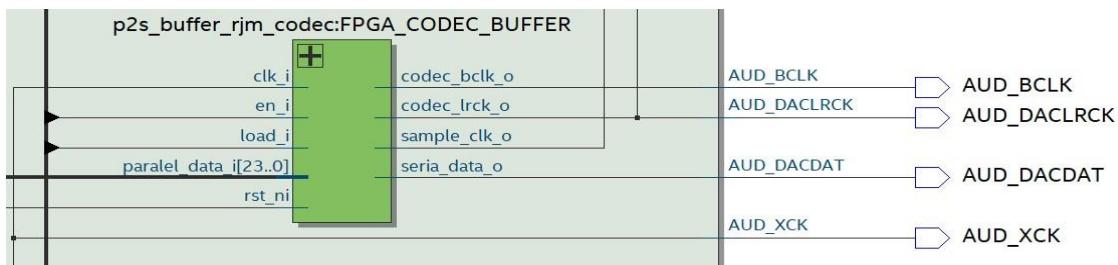


Hình 4-15: Module thanh ghi dịch từ song song sang nối tiếp

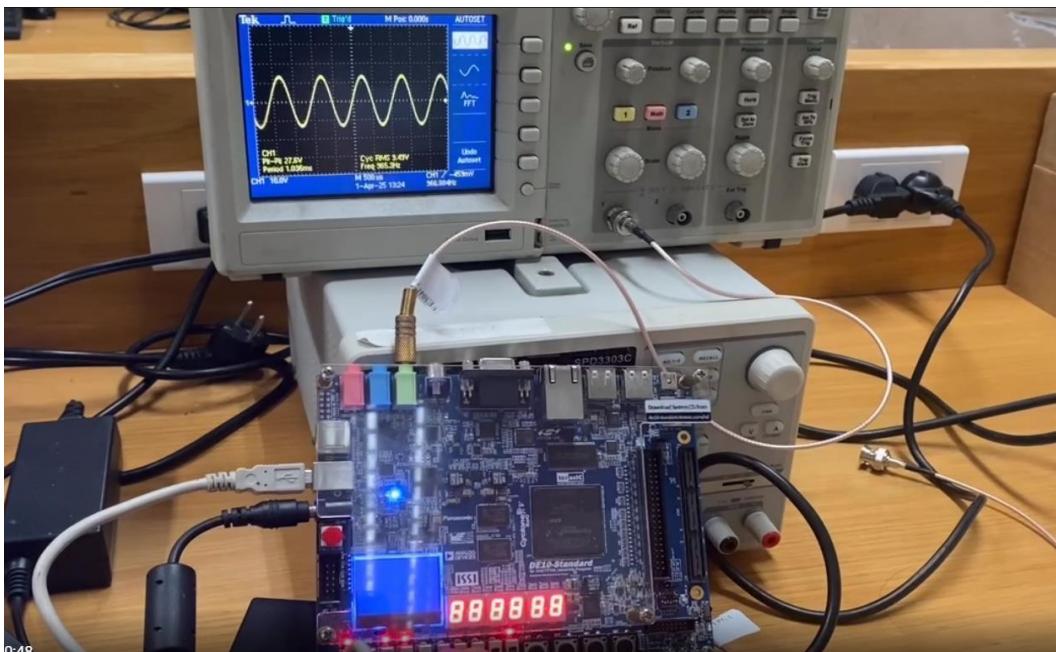
Nhận xét: như waveform data được shift từ từ parallel sang serial



Hình 4-16: Kết nối các ngõ vào và ngõ ra data ra ADC và DAC trên WM8731



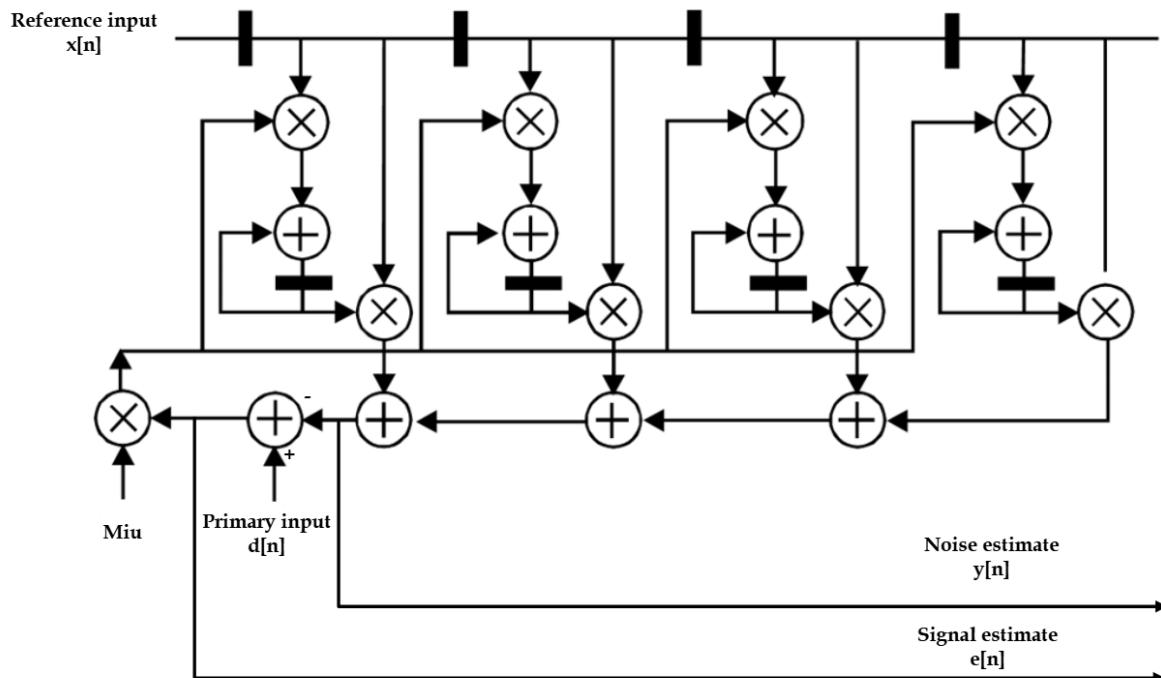
Hình 4-17: Nối pin ra các chân của DE10



Hình 4-18: kết nối thành công với FPGA

4.3. Thiết kế bộ lọc thích ứng

4.3.1 Cấu trúc bộ lọc thích ứng



Hình 4-19: Sơ đồ hoàn chỉnh bộ Adaptive filter

Tín hiệu đầu vào

- $x[n]$ (Reference Input): Đây là tín hiệu tham chiếu
- $d[n]$ (Primary Input): Đây là tín hiệu đầu vào chính,

Các thành phần chính bộ FIR:

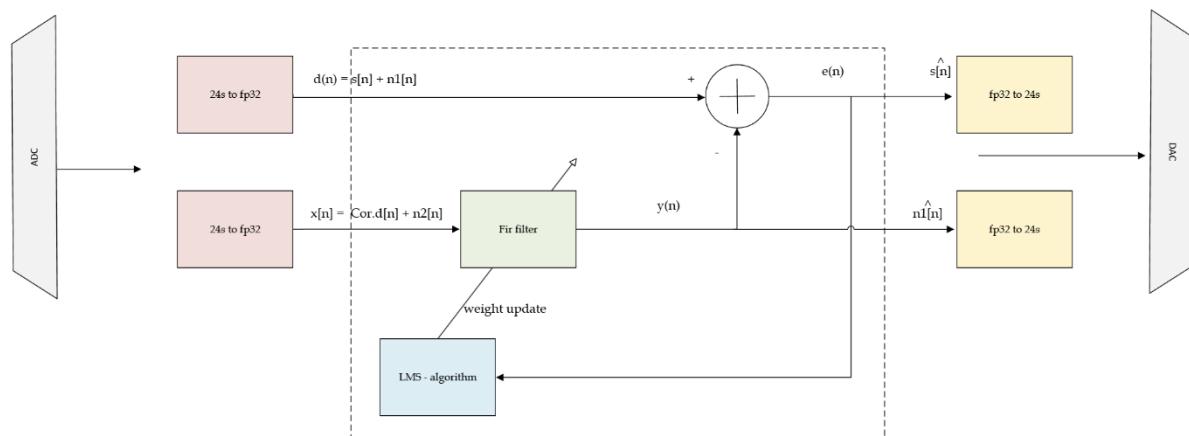
1. Chuỗi phần tử trễ D1: Mỗi khối D1 đại diện cho một thanh ghi dịch (delay register), thường là phần tử lưu trữ đơn giản có thể ánh xạ bằng D flip-flop hoặc bộ nhớ tạm trong FPGA và ASIC. Chuỗi này giữ lại các mẫu đầu vào trước đó, giúp cung cấp tập dữ liệu $x[n], x[n-1], \dots, x[n-N+1]$ cho bộ lọc ở mọi thời điểm.
2. Khối nhân \times : Mỗi giá trị đầu vào sau trễ được đưa vào một khối nhân, nơi nó được nhân với hệ số tương ứng hk. Trong thiết kế VLSI, các bộ nhân kiểu Booth multiplier, Wallace tree hoặc các khối MAC chuyên biệt thường được sử dụng nhằm tối ưu tốc độ và diện tích bề mặt silicon.
3. Bộ cộng Σ : Toàn bộ kết quả từ các khối nhân được đưa vào khối tổng (accumulator hoặc adder tree), nơi thực hiện phép cộng song song hoặc tuần tự để tạo đầu ra rn.

Kiến trúc bộ cộng phụ thuộc vào yêu cầu thời gian đáp ứng: có thể dùng adder tree (song song) hoặc systolic structure (pipeline tuyến tính).

4. Chuỗi thanh ghi dịch Trong các hệ thống xử lý tín hiệu số rời rạc, đặc biệt là bộ lọc FIR (Finite Impulse Response), việc lưu trữ và sử dụng lại các mẫu tín hiệu trong quá khứ là một yêu cầu bắt buộc nhằm phục vụ phép tích chập với tập hệ số (tap coefficients). Để đáp ứng yêu cầu này, một cấu trúc bộ đệm tín hiệu dạng đường truyền trễ (delay line) nhiều cấp, còn gọi là chuỗi thanh ghi dịch (shift register chain), được triển khai với mục đích giữ lại trạng thái lịch sử của tín hiệu đầu vào theo đúng trình tự thời gian.

Module group_delay đóng vai trò như một bộ đệm tín hiệu tuyến tính gồm 32 vị trí mẫu, tương ứng với 32 mức độ trễ. Bộ đệm này thực hiện chức năng lưu giữ giá trị của tín hiệu đầu vào trước đó bên cạnh mẫu hiện tại, nhằm phục vụ cho tác vụ nhân-tích lũy trong bộ lọc FIR. Mỗi phần tử trong chuỗi thanh ghi hoạt động như một tầng trễ một chu kỳ đồng hồ, nơi mà mỗi mẫu tín hiệu được truyền liên tiếp từ cấp hiện tại sang cấp kế tiếp vào mỗi chu kỳ đồng bộ. Với dữ liệu đầu vào có độ rộng 24bit, toàn bộ cấu trúc đảm bảo khả năng duy trì độ trung thực của tín hiệu qua từng chu kỳ dịch tuyến tính.

4.3.2 Vấn đề phát sinh khi triển khai lên kit FPGA

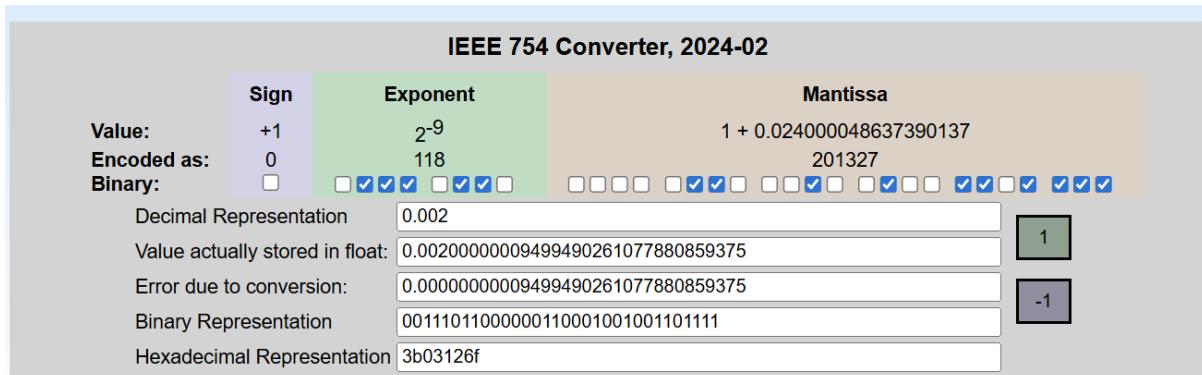


Hình 4-20: Interface giữa bộ lọc thích nghi và ADC DAC

Do sau khi qua bộ Codec WM8731 như phần 4.2 em đã đề cập, ADC scale tín hiệu analog xuống thành 24bit có dấu dạng bù 2, do đó cần có một cơ chế scale để bộ lọc thực hiện đúng mức.

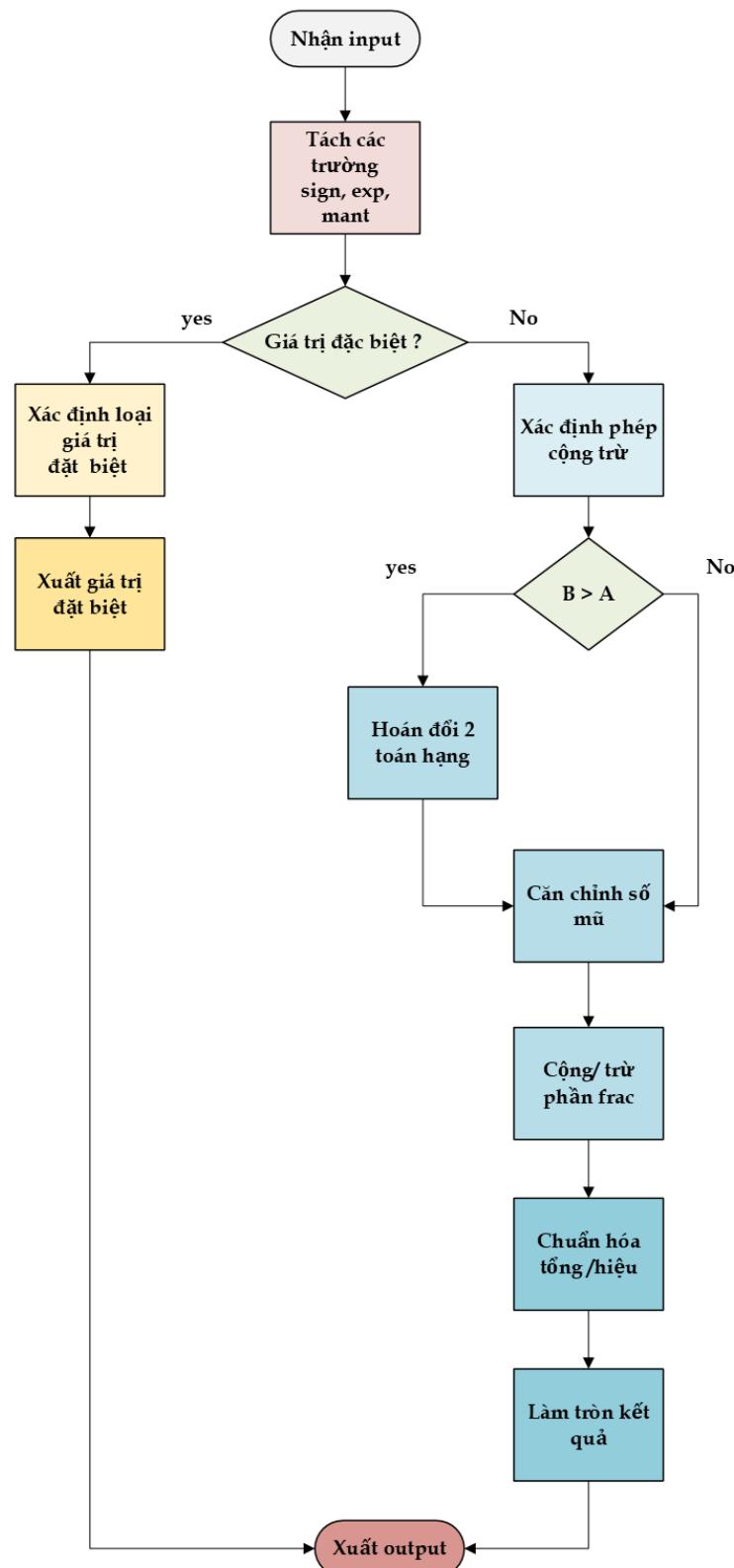
Hơn nữa theo như lý thuyết $\mu \approx \frac{1}{N \cdot P_x}$ có gì gó bé hơn một nén để bảo đảm quá trình toán cộng trừ và chia hòa động chuẩn nhất thì tất cả các giá trị khi thực hiện trong giải thuật LMS đều phải ở dạng dấu chấm động 32bit theo chuẩn IEEE 754.

Như vậy trong nội bộ khối floating point chúng ta cần thêm các module cộng và nhân theo dạng floating point như đã đề cập.



Hình 4-21: Chuyển đổi các dạng số sang fp32
Floating point có thể đại diện số rất bé tới mũ 2^{-23} do đó với kiểu nhân và cộng tích lũy số thập phân như LMS sẽ không lo bị sai ở bất kì bước nào.

4.3.2.1. Bộ cộng floating point



Hình 4-22: Giải thuật cộng số floating point

Testcase kiểm thử

```

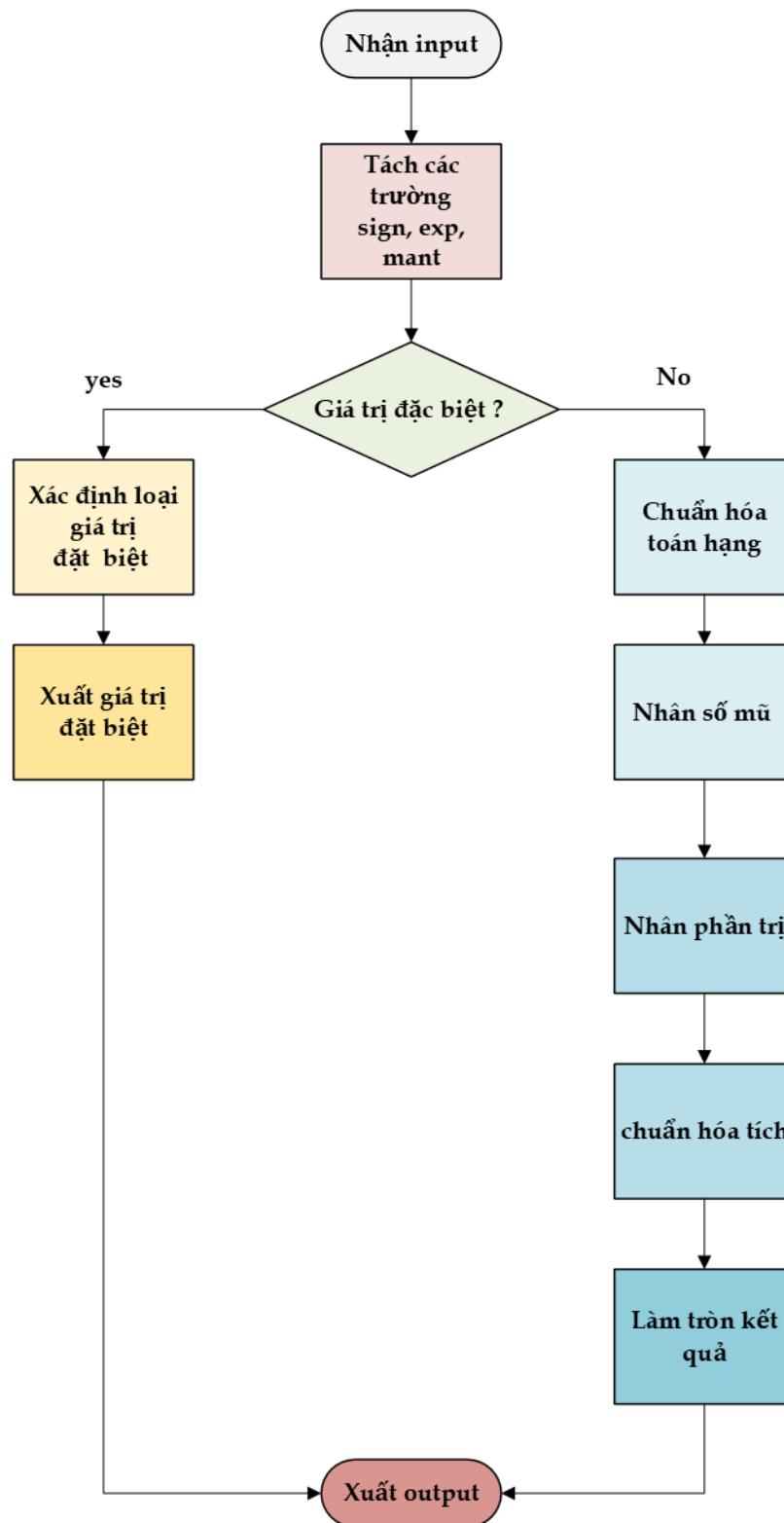
Checking: A=0xcfa5c338, B=0xfd834afc, SUB=0 | Got=0xfd834afc | Expected=0xfd834afc
Checking: A=0x80bea3aa, B=0x375ba4a3, SUB=0 | Got=0x375ba4a3 | Expected=0x375ba4a3
Checking: A=0xd5a638f9, B=0x13f9dd5a, SUB=1 | Got=0xd5a638f9 | Expected=0xd5a638f9
Checking: A=0xd6e54a2b, B=0xe1af5b29, SUB=0 | Got=0xe1af5b2c | Expected=0xe1af5b2d
xmsim: *E,ERRSEV (.../01_tb/testbench.sv,71): (time 998 NS).
fpu_add_sub_tb.check
Mismatch: A=0xd6e54a2b, B=0xe1af5b29 | Got=0xe1af5b2c | Expected=0xe1af5b2d
Checking: A=0xa0879d80, B=0xf16a6d51, SUB=1 | Got=0x716a6d51 | Expected=0x716a6d51
Checking: A=0xc9b70be9, B=0xb973f6a, SUB=0 | Got=0xc9b70be9 | Expected=0xc9b70be9
Checking: A=0x5b9d7ceb, B=0x402808e9, SUB=0 | Got=0x5b9d7ceb | Expected=0x5b9d7ceb
Checking: A=0xde3fe121, B=0x7b1757a, SUB=1 | Got=0xde3fe121 | Expected=0xde3fe121
Checking: A=0x35f6723e, B=0xf5719151, SUB=0 | Got=0xf5719151 | Expected=0xf5719151
Checking: A=0xd1b49a3e, B=0x05c84d68, SUB=1 | Got=0xd1b49a3e | Expected=0xd1b49a3e
Checking: A=0xf22d1825, B=0x1264a23d, SUB=0 | Got=0xf22d1825 | Expected=0xf22d1825
Checking: A=0x0ba9f77f, B=0xf52da487, SUB=1 | Got=0x752da487 | Expected=0x752da487
Checking: A=0x5ffce87a, B=0x54b8360b, SUB=1 | Got=0x5ffce878 | Expected=0x5ffce877
xmsim: *E,ERRSEV (.../01_tb/testbench.sv,71): (time 1007 NS).
fpu_add_sub_tb.check
Mismatch: A=0x5ffce87a, B=0x54b8360b | Got=0x5ffce878 | Expected=0x5ffce877
Checking: A=0x5c022958, B=0x98c98d6b, SUB=1 | Got=0x5c022958 | Expected=0x5c022958
Checking: A=0xf38fae87, B=0x2b2d4aa8, SUB=1 | Got=0xf38fae87 | Expected=0xf38fae87
Checking: A=0x98ab3d9b, B=0x50f062b2, SUB=1 | Got=0xd0f062b2 | Expected=0xd0f062b2
Checking: A=0x6e9cf00a, B=0x01bf8b36, SUB=0 | Got=0x6e9cf00a | Expected=0x6e9cf00a
Checking: A=0x6f08bd54, B=0x1ef42889, SUB=0 | Got=0x6f08bd54 | Expected=0x6f08bd54
Checking: A=0x4f3470fa, B=0xb99ec26a, SUB=0 | Got=0x4f3470fa | Expected=0x4f3470fa
Checking: A=0x7d2c7f75, B=0xa67fe0eb, SUB=1 | Got=0x7d2c7f75 | Expected=0x7d2c7f75
Manual Check: A=0x40490fdb, B=0x40000000, SUB=0 | Result=0x40a487ed

===== All Tests Passed =====

```

Hình 4-23: Testbench bộ cộng FP32

4.3.2.2. Bộ nhân floating point



Hình 4-24: Giải thuật nhân số floating point

Ví dụ nhân 2 số floating point:

	Số thập phân	Số nhị phân (bỏ qua dấu)	Scale về dạng chuẩn
A	-0.3	0.010011	1.0011 x2 ⁻²
B	500.25	111110100.01	1.1111010001 x2 ⁸
Đáp số	-150.075	10010110.11	-1.001011011 x2 ⁷

-Xét về dấu: Dấu của tích 2 phép nhân Floating point là XOR 2bit đầu của số A và số B.

-Về mặt exponent sẽ là tổng exponent của A và B, Như ví dụ kết quả sẽ là số âm

-Fraction sẽ là tích của 2 fraction.

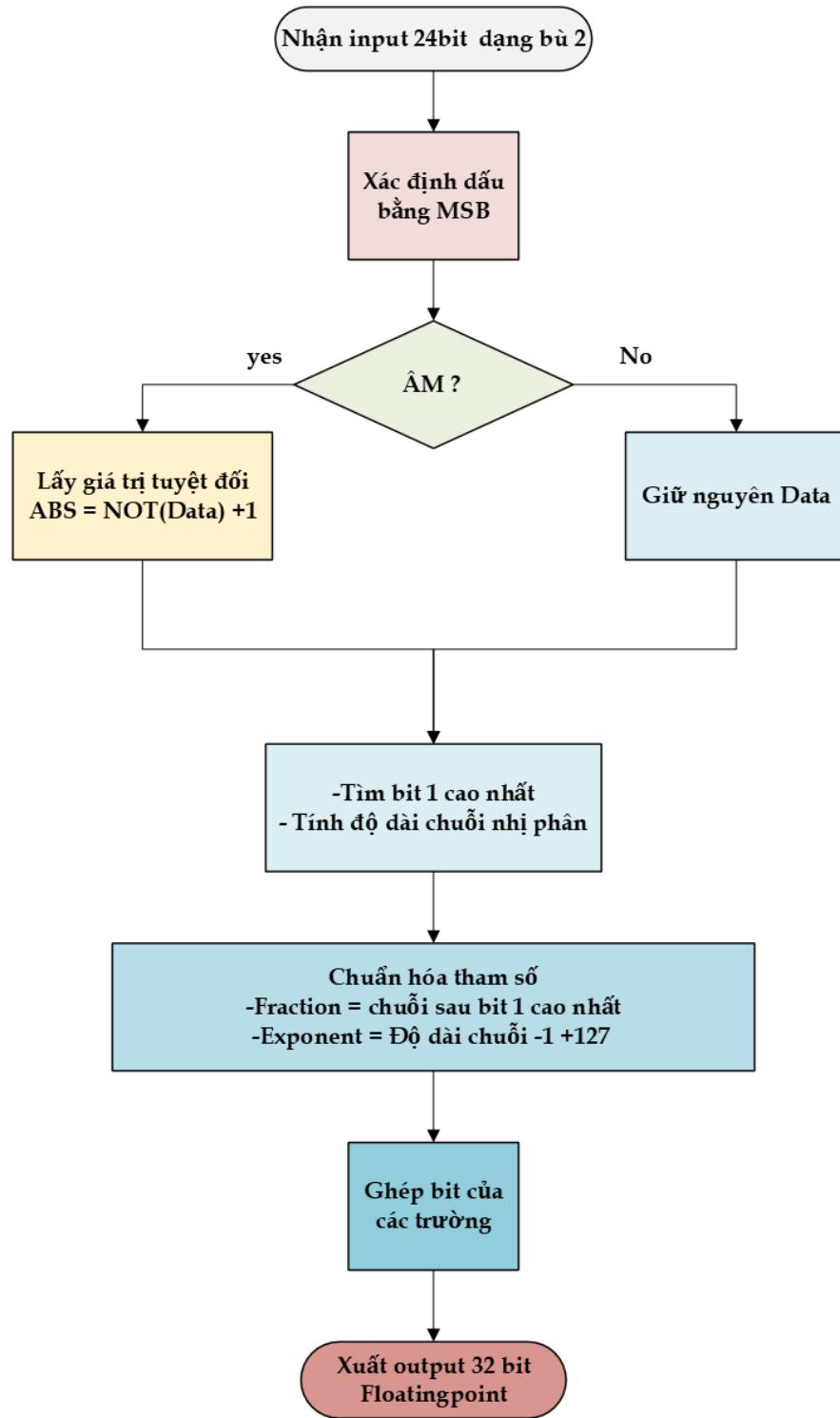
- Kết quả sẽ được scale về dạng chuẩn 1. x2^{-y}

```
===== Floating-Point Multiplier Testbench =====
=
      0 | A = 3f800000 | B = 3f800000 | Result = 3f800000
=
     10 | A = 40000000 | B = 40400000 | Result = 40c00000
=
     20 | A = 3f400000 | B = 3f800000 | Result = 3f400000
=
     30 | A = 3f800000 | B = bf800000 | Result = bf800000
=
     40 | A = 40400000 | B = c0000000 | Result = c0c00000
=
     50 | A = bf800000 | B = bf800000 | Result = 3f800000
=
     60 | A = c0000000 | B = c0000000 | Result = 40800000
=
     70 | A = 7f800000 | B = 3f800000 | Result = 7f800000
=
     80 | A = 7f800000 | B = bf800000 | Result = ff800000
=
     90 | A = 7f800000 | B = 00000000 | Result = 00000000
=
    100 | A = 00000001 | B = 00000002 | Result = 00000000
=
    110 | A = 00800000 | B = 41c00000 | Result = 00000000
=
    120 | A = 3e4ccccd | B = 3dccccc | Result = 3ca3d70a
=
    130 | A = 3f800000 | B = 40000000 | Result = 40000000
=
    140 | A = 40490fdb | B = 40400000 | Result = 4116cbe4
=
    150 | A = 47c35000 | B = 00000001 | Result = 00000000
=
    160 | A = 47c35000 | B = 42f00000 | Result = 4b371b00
=
    170 | A = c3800000 | B = 42f00000 | Result = c6f00000
=
    180 | A = 00000000 | B = 3f800000 | Result = 00000000
=
    190 | A = 80000000 | B = bf800000 | Result = 00000000
=
    200 | A = b8d1b717 | B = bc当地6f9e | Result = 361b2f9b
=
    210 | A = bc当地993d | B = c0417adf | Result = 3dc12d34

=====
 Test Results =====
```

Hình 4-25: Testbench các testcase của bộ nhân

4.3.2.3. Chuyển đổi từ số bù 2 sang floating point



Hình 4-26: Giải thuật chuyển đổi số S24 sang dạng số FP32

Testbench chuyển đổi từ số bù 2 sang floating point

```

Checking: din=0x773f9c (7815068 ) | DUT dout=0x4aee7f38 | Expected gold=0x4aee7f38
Checking: din=0x9a1422 (-6679518) | DUT dout=0xcacbd7bc | Expected gold=0xcacbd7bc
Checking: din=0x5e204b (6168651 ) | DUT dout=0x4abc4096 | Expected gold=0x4abc4096
Checking: din=0x28b171 (2666865 ) | DUT dout=0x4a22c5c4 | Expected gold=0x4a22c5c4
Checking: din=0xec5f16 (-1286378) | DUT dout=0xc99d0750 | Expected gold=0xc99d0750
Checking: din=0xe2d434 (-1911756) | DUT dout=0xc9e95e60 | Expected gold=0xc9e95e60
Checking: din=0x7bca94 (8112788 ) | DUT dout=0x4af79528 | Expected gold=0x4af79528
Checking: din=0x1ebf8 (1973240 ) | DUT dout=0x49f0dfc0 | Expected gold=0x49f0dfc0
Checking: din=0x0e316c (930156 ) | DUT dout=0x496316c0 | Expected gold=0x496316c0
Checking: din=0x90bc1d (-7291875) | DUT dout=0xcade87c6 | Expected gold=0xcade87c6
Checking: din=0x283389 (2634633 ) | DUT dout=0x4a20ce24 | Expected gold=0x4a20ce24
Checking: din=0x8643e2 (-7978014) | DUT dout=0xcaf3783c | Expected gold=0xcaf3783c
Checking: din=0x71d401 (7459841 ) | DUT dout=0x4ae3a802 | Expected gold=0x4ae3a802
Checking: din=0xe3c532 (-1850062) | DUT dout=0xc9e1d670 | Expected gold=0xc9e1d670
Checking: din=0x09b7c5 (636869 ) | DUT dout=0x491b7c50 | Expected gold=0x491b7c50
Checking: din=0xde262b (-2218453) | DUT dout=0xca076754 | Expected gold=0xca076754
Checking: din=0xa9bd06 (-5653242) | DUT dout=0xcaac85f4 | Expected gold=0xcaac85f4
Checking: din=0x7a2971 (8006001 ) | DUT dout=0x4af452e2 | Expected gold=0x4af452e2
Checking: din=0x7e576c (8279916 ) | DUT dout=0x4afcaed8 | Expected gold=0x4afcaed8
Checking: din=0xc37936 (-3966666) | DUT dout=0xca721b28 | Expected gold=0xca721b28
Checking: din=0xc3dcf8 (-3941128) | DUT dout=0xca708c20 | Expected gold=0xca708c20
Checking: din=0xccc107 (-3358457) | DUT dout=0xca4cfbe4 | Expected gold=0xca4cfbe4
Checking: din=0x21afb3 (2207667 ) | DUT dout=0x4a06becc | Expected gold=0x4a06becc
Checking: din=0x6aa256 (6988374 ) | DUT dout=0x4ad544ac | Expected gold=0x4ad544ac
Checking: din=0xc9f01e (-3543010) | DUT dout=0xca583f88 | Expected gold=0xca583f88
Checking: din=0xaf0fca (-5304374) | DUT dout=0xcaa1e06c | Expected gold=0xcaa1e06c
Checking: din=0x4c2457 (4990039 ) | DUT dout=0x4a9848ae | Expected gold=0x4a9848ae
Checking: din=0x275b48 (2579272 ) | DUT dout=0x4a1d6d20 | Expected gold=0x4a1d6d20
Checking: din=0x40000c (4194316 ) | DUT dout=0x4a800018 | Expected gold=0x4a800018
Checking: din=0xd402be (-2882882) | DUT dout=0xca2ff508 | Expected gold=0xca2ff508
Checking: din=0x933923 (-7128797) | DUT dout=0xcad98dba | Expected gold=0cad98dba
Checking: din=0xe907b4 (-1505356) | DUT dout=0xc9b7c260 | Expected gold=0xc9b7c260
Checking: din=0xab1524 (-5565148) | DUT dout=0xcaa9d5b8 | Expected gold=0caa9d5b8
Checking: din=0x475d93 (4677011 ) | DUT dout=0x4a8ebb26 | Expected gold=0x4a8ebb26
Checking: din=0x441b6c (4463468 ) | DUT dout=0x4a8836d8 | Expected gold=0x4a8836d8
Checking: din=0xbabab1 (-4539727) | DUT dout=0xca8a8a9e | Expected gold=0ca8a8a9e
Checking: din=0x081812 (530450 ) | DUT dout=0x49018120 | Expected gold=0x49018120
All tests passed.
MANUAL: din=0x18fc8d (1637517) | dout=0x49c7e468 | ref=0x49c7e468
MANUAL: din=0xf45875 (-763787) | dout=0xc93a78b0 | ref=0xc93a78b0
MANUAL: din=0x000001 (1) | dout=0x3f800000 | ref=0x3f800000
MANUAL: din=0xfffff68 (-152) | dout=0xc3180000 | ref=0xc3180000

===== Convert-to-FP32 Test Completed =====

Simulation complete via $finish(1) at time 10025 NS + 0
.../01_tb/testbench.sv:92      $finish;
xcelium> exit
TOOL: xrun(64)      20.09-s001: Exiting on Sep 22, 2025 at 23:25:50 EDT (total: 00:00:27)
[admin@centos7 02_sim]$ 

```

Hình 4-27: Testbench chuyển đổi từ số bù 2 sang floating point

Testbench chuyển đổi từ floating point sang số 24bit dạng bù 2

```

Checking: fp_in=0xcdc46572 | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x3230903d | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xc5ce8996 | Got=int_out=-6609 (0xffe62f) | Expected=gold=-6609 (0xffe62f)
Checking: fp_in=0x73b122bb | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x7490e033 | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x26437519 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x85b2540b | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x20193bcc | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xbb39e44c | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xf3d57d82 | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x878cbc61 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x5ae56ad4 | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x7e36067c | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0xeba09eae | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0xe65f04e0 | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x371f3380 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xb0abd1 | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x2adb923 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xa50e46fc | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x5209d78c | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0xfad479c5 | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x52b7aa31 | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x91d5a0d4 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xc2a4a03d | Got=int_out=-82 (0xfffffae) | Expected=gold=-82 (0xfffffae)
Checking: fp_in=0xcdd0667d | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x2fe1ad74 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x99cc12b2 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xf92b57e6 | Got=int_out=-8388608 (0x800000) | Expected=gold=-8388608 (0x800000)
Checking: fp_in=0x76d3e38b | Got=int_out=8388607 (0x7fffff) | Expected=gold=8388607 (0x7fffff)
Checking: fp_in=0x52056e55 | Got=int_out=-8388607 (0x7fffff) | Expected=gold=-8388607 (0x7fffff)
Checking: fp_in=0x0c690c90 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0x8e97f365 | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
Checking: fp_in=0xb6d0a8cb | Got=int_out=0 (0x000000) | Expected=gold=0 (0x000000)
MANUAL Check: fp_in=0x3f710a13 | int_out=0 (0x000000)

===== All Tests Passed =====

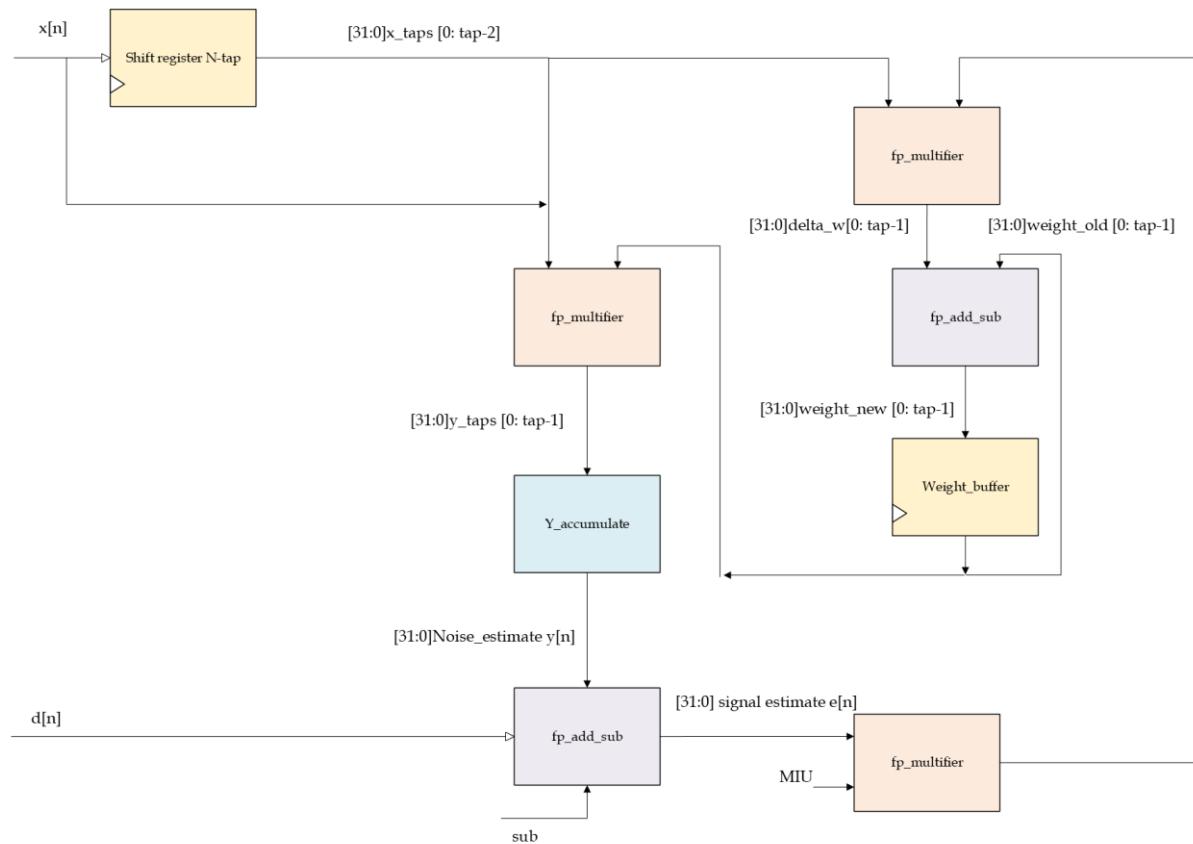
Simulation complete via $finish() at time 2014 NS + 0
../01_tb/testbench.sv:115      $finish;
xcelium> exit
TOOL:  xrun(64)      20.09-s001: Exiting on Sep 22, 2025 at 23:00:45 EDT (total: 00:00:23)
[admin@centos7 02_sim]$ █

```

Hình 4-28: Testbench chuyển đổi từ floating point sang số 24 bit

4.3.3. Thiết kế và mô phỏng phần cứng bộ lọc

4.3.3.1. Thiết kế bộ lọc thích ứng trên nền bộ lọc FIR với giải thuật LMS



Hình 4-29: Block diagram của module Adaptive filter

-Như em đã trình bày ở phần trên, toàn bộ tín hiệu đi vào và đi ra của bộ lọc thích ứng đều ở dạng floating point 32bit theo chuẩn IEEE 754.

Cấu Trúc Tổng Quát của Adaptive Filter

Adaptive Filter này bao gồm các thành phần và bước xử lý chính:

1/ Đầu vào:

- $x[n]$: Tín hiệu đầu vào thứ cấp (reference signal).
- $d[n]$: Đầu vào tín hiệu chính (primary signal).
- μ : Hệ số học (learning rate).

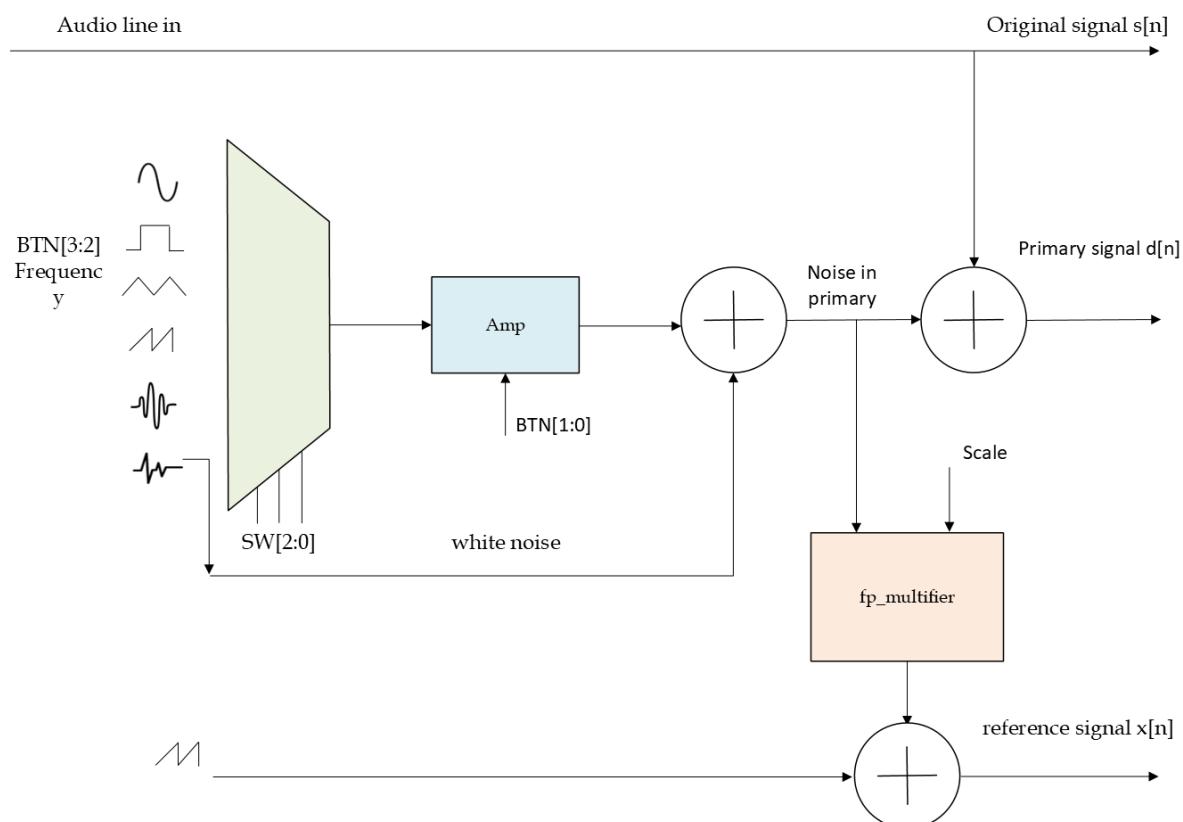
2/ Thành phần chính:

- Shift Register (N-Tap): Bộ lưu trữ cái mẫu (sample) được lấy mẫu của tín hiệu đầu vào thứ cấp để tạo thành vector đầu vào $[x_taps]$, cập nhật data theo CLK.

- fp_multiplier: Phép nhân số học có dấu dạng dấu chấm động (floating point) theo chuẩn IEEE-754 32bit.
- Y_accumulate: cộng tích lũy các tích trọng số (weights) với tín hiệu x_taps, Y_acc tương ứng với ngõ ra của bộ direct FIR.
- fp_add_sub: Khối cộng/trừ số học dấu chấm động.
- Weight Update (Learning): Cập nhật mỗi chu kì CLK

3/ Đầu ra:

- y[n]: Tín hiệu ước lượng từ bộ lọc.
- e[n]: Tín hiệu lỗi giữa d[n] và y[n] (giúp cập nhật trọng số).



Hình 4-30: Thiết kế thành phần tạo nhiễu cho ngõ vào của bộ lọc thích nghi

Để kiểm chứng khả năng lọc nhiễu tự điều chỉnh, thì phải tạo các tín hiệu nhiễu có thể thay đổi điều chỉnh thủ công được cả về mặt biên độ và tần số, switch [2:0] để lựa chọn tín hiệu nhiễu giữa các dạng sóng cơ bản, nút BTN[1:0] điều chỉnh độ khuếch đại của biên độ nhiễu.

Nguồn tín hiệu nhiễu

Để kiểm chứng khả năng lọc nhiễu của bộ lọc, tín hiệu nhiễu đầu vào được mô phỏng từ nhiều loại tín hiệu điển hình như nhiễu White Noise trộn với các tín hiệu dạng sóng cơ bản như sóng vuông (square wave), sóng răng cưa (sawtooth wave), hoặc sóng hình sin (sine wave). Các dạng sóng này được lấy từ bảng tra LUT lưu trong bộ nhớ BRAM của FPGA giúp giảm thiểu tiêu tốn logic element.

Người dùng có thể lựa chọn loại nhiễu mong muốn thông qua các switch SW[2:0], giúp mô phỏng các kịch bản cụ thể cho bài toán.

Điều chỉnh biên độ

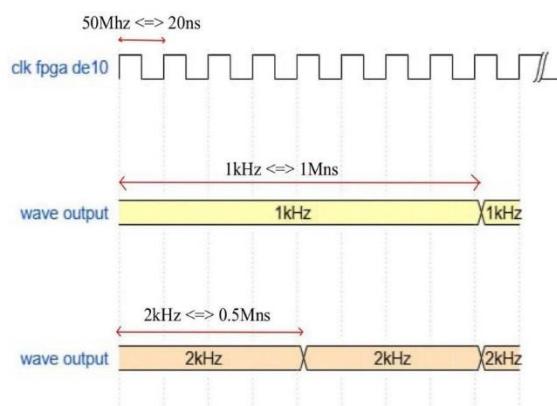
Biên độ của tín hiệu nhiễu có thể được điều chỉnh thông qua nút nhấn BTN[1:0]. Thành phần khuếch đại (Amp) giúp điều chỉnh mức nhiễu từ nhỏ tới lớn (scalable noise amplitude), mô phỏng các mức nhiễu nhẹ đến cực lớn. Bộ khuếch đại đơn giản là nhân tín hiệu đầu vào với các giá trị cho trước được chọn bằng MUX.

Việc điều khiển biên độ giúp phân tích ảnh hưởng của cường độ nhiễu đến hiệu suất của bộ lọc thích nghi.

Điều chỉnh tần số

BTN[3:2] được sử dụng để thay đổi tần số của tín hiệu nhiễu. Tần số là một tham số quan trọng, cho phép hệ thống thử nghiệm trong các môi trường nhiễu khác nhau, từ tần số thấp (nhiều chập biến đổi) đến tần số cao.

Để có thể điều chỉnh tần số, em sẽ dùng counter các biên nhảy pha để kéo giãn hoặc bóp chu kì từng sample lấy mẫu



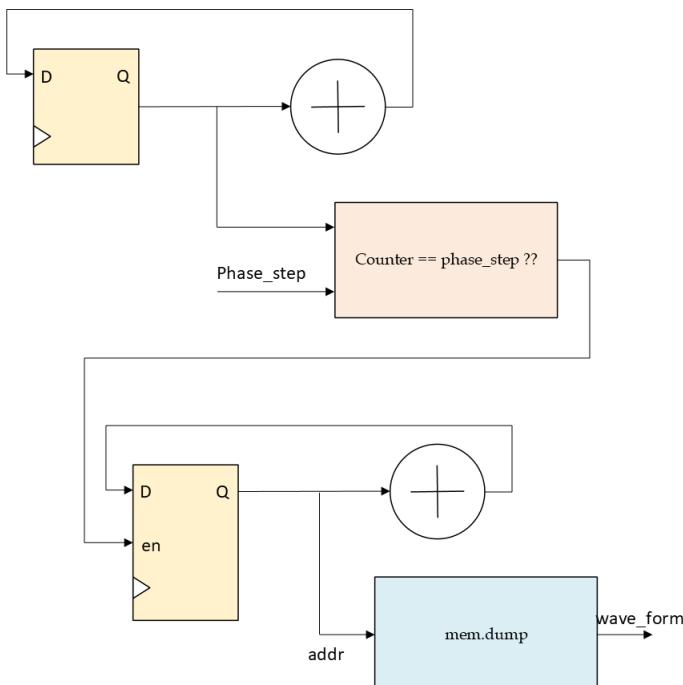
Hình 4-31: Tương quan giữa CLK của FPGA và tần số mong muốn tạo ra

Giả sử một xung được cấu hình là 1024 mẫu thì trong một chu kì 1Mns, thời gian chạy 1 mẫu sẽ là 977ns. Vậy 1 mẫu sẽ tồn 49 chu kì clock của DE10. Từ đó nếu muốn có được tần số mong muốn ta chỉ cần nạp phase step từ công thức sau

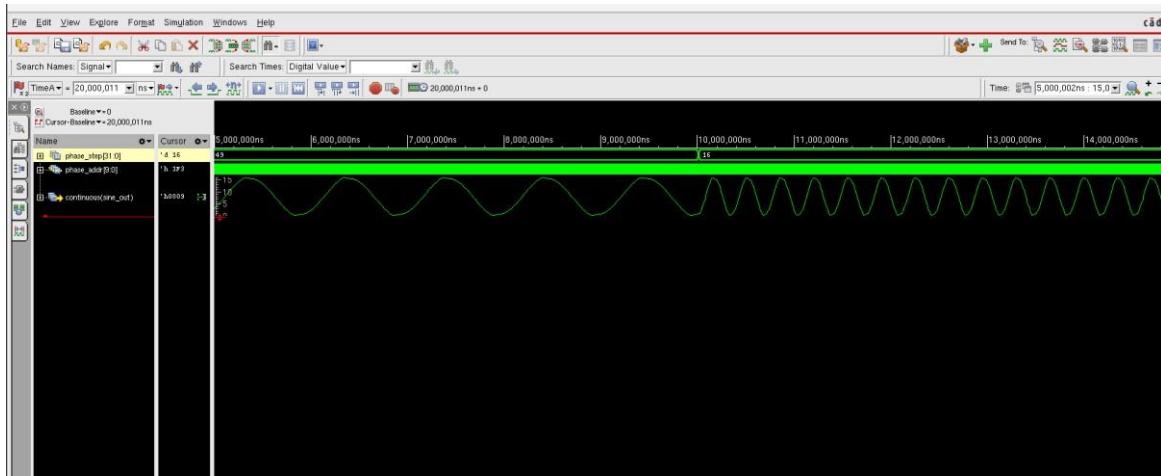
$$\text{Phase step} = \frac{F_{\text{CLK}} \text{ của FPGA}}{F_{\text{out}} \times \text{số mẫu lượng tử}} \quad (4.1)$$

Bảng 4-5: Bảng điều chỉnh tần số

F_{out}	Phase step (thang decimal)
1KHz	49
2KHz	24
3KHz	16
4KHz	12
8KHz	6
16KHz	3



Hình 4-32: Module thiết kế bộ điều chỉnh tần số

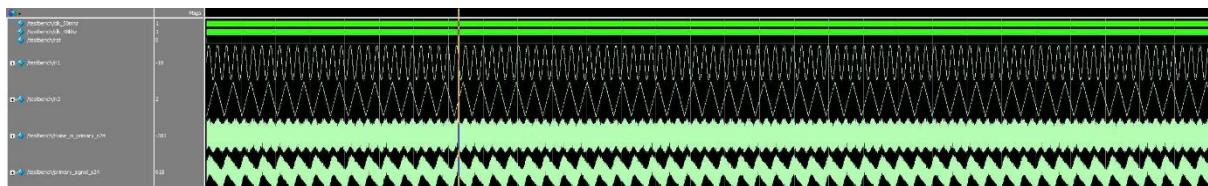


Hình 4-33: Mô phỏng tăng tần số nhiễu

Luồng tín hiệu

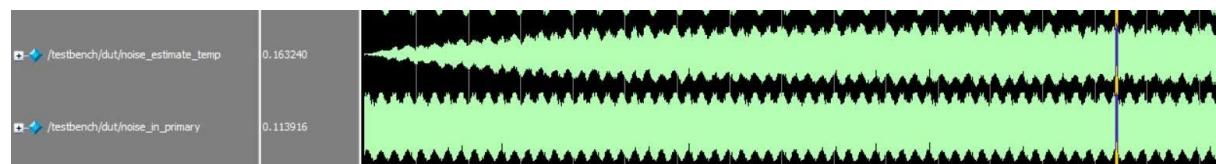
- Tín hiệu Audio Line-In (Tín hiệu âm thanh đầu vào): Đây là tín hiệu âm thanh gốc $s[n]$, được gửi vào đường truyền chính.
- Tạo tín hiệu ngõ vào chính: Tín hiệu nhiễu được tạo bởi mạch tạo nhiễu được đưa qua bộ khuếch đại (Amp) để điều chỉnh biên độ nhiễu, sau đó được cộng vào tín hiệu gốc để tạo thành tín hiệu $d[n]$ tín hiệu ngõ vào chính gồm tín hiệu có ích và nhiễu. Tín hiệu này đại diện cho tín hiệu thực tế mà bộ lọc thích nghi cần xử lý.
- Tạo tín hiệu tham chiếu ngõ vào thứ cấp (Reference Signal): Tín hiệu nhiễu sau khi được khuếch đại còn được gửi tới hệ thống tạo tín hiệu tham chiếu $x[n]$. Tín hiệu tham chiếu chỉ tương quan một phần với nhiễu $n1[n]$ ở ngõ vào chính, đã được chia nhỏ hoặc tái định dạng bởi thành phần $fp_multiplier$ để phù hợp với yêu cầu xử lý.

4.3.3.2. Mô phỏng bộ lọc thích ứng trên nền bộ lọc FIR với giải thuật LMS



Hình 4-34: Các thành phần sóng ở ngõ vào

Nhiều ở $d[n]$ sẽ gồm nhiều trắng và sine tần số cao.



Hình 4-35: Thành phần ngõ vào chính và thành phần ngõ vào thứ cấp

Nhận xét nếu bình thường 2 input này đi vào mà không có LMS thì sẽ không biết dạng sóng chính xác của nó là gì.



Hình 4-36: $e[n]$ và $s[n]$

Ta có thể thấy được biên độ nhiễu sẽ giảm dần tới lúc hội tụ là nhiễu nhỏ nhất. Điều này cũng chứng minh là bộ cộng trừ nhân fp32 hoạt động đúng.

Độ hội tụ của giải thuật:

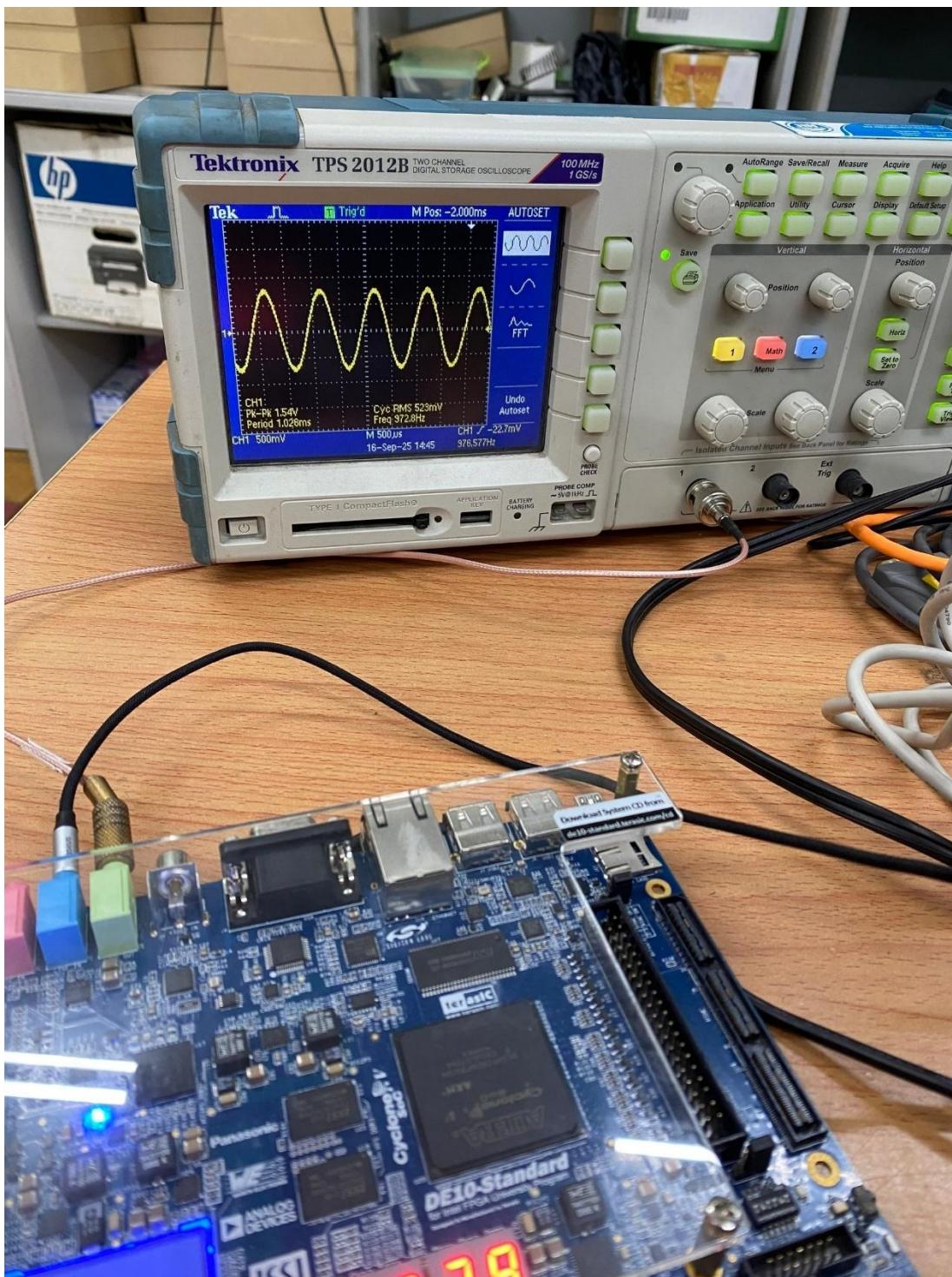


Hình 4-37: Dạng sóng thay đổi của hệ số bộ lọc

Ta có thể quan sát thấy delta weight gần như không tăng và chỉ dao động nhỏ sau khi đã hội tụ. Cho thấy rằng weight đang đi vòng quanh điểm W_{opt} ở mặt phẳng phương vị.

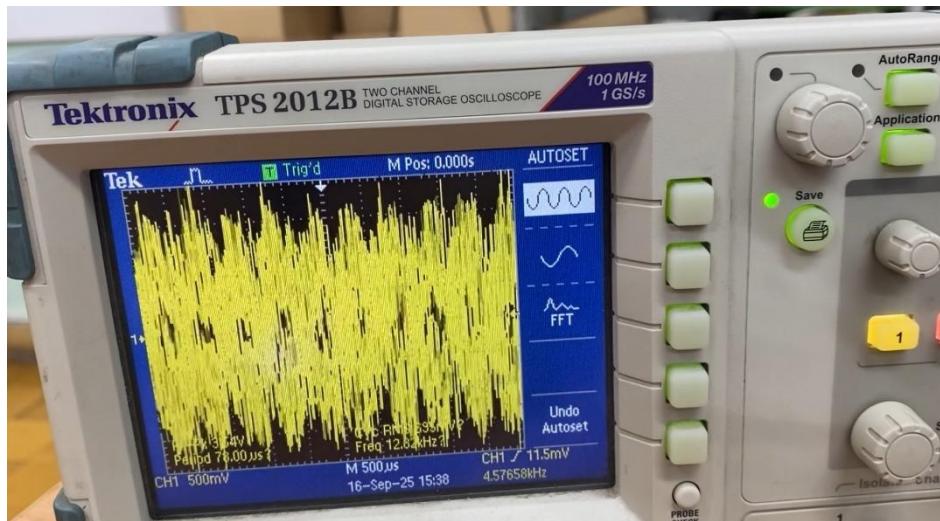
5. KẾT QUẢ THỰC HIỆN

5.1. Thực hiện lọc các thành phần sóng cơ bản:

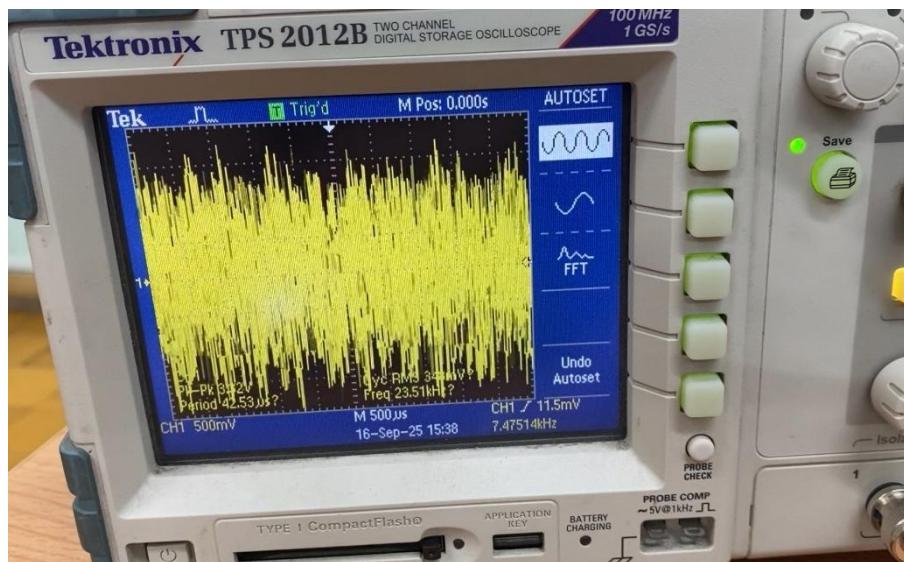


Hình 5-1: Dạng sóng sóng gốc 1.54Vpp

Ban đầu dạng sóng gốc là hình sine tần số thấp, được chèn thêm nhiễu $n1[n]$ để thành primary signal. Và reference signal được chèn thêm can nhiễu và nhiễu $n2[n]$. Ngõ vào chính và ngõ vào thứ cấp có tạo thành wave form như sau:

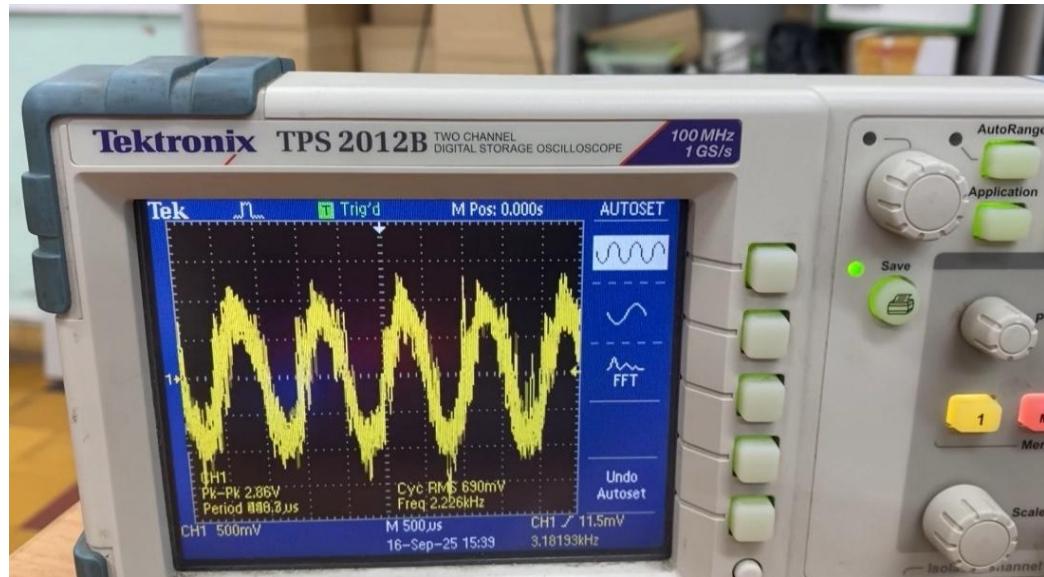


Hình 5-2: Dạng sóng của primary input 3.54 Vpp



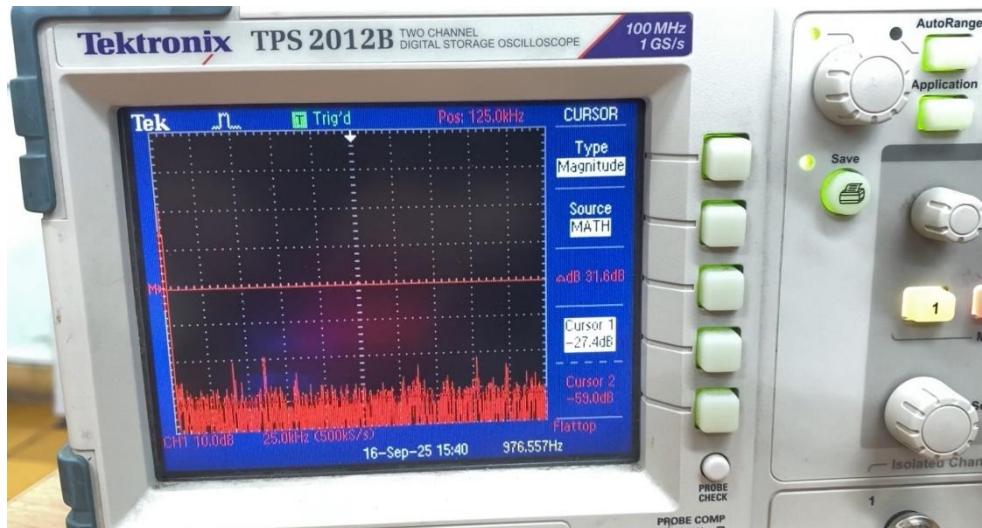
Hình 5-3: Dạng sóng của reference signal 3.42 Vpp

Nhận xét: Nhìn vào dạng sóng ta có thể thấy, cường độ nhiễu rất nặng. Biên độ vào các input gần như là được tăng lên gấp đôi với sóng sine gốc, 2 input đầu vào hoàn toàn mất đi dạng sóng của sóng sine ban đầu.

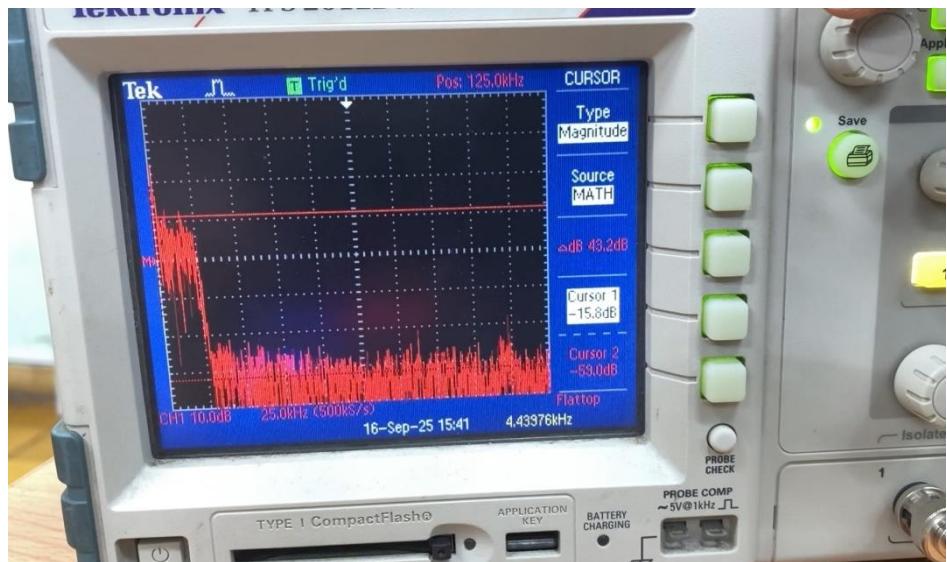


Khi cho 2 đầu vào của adaptive filter có $\mu = 0.0045$ như đã tìm được ở Matlab với số tap là 5, có thấy thấy kết quả ước lượng tín hiệu sau khi lọc đã giảm đáng kể thành phần nhiễu. Nhưng

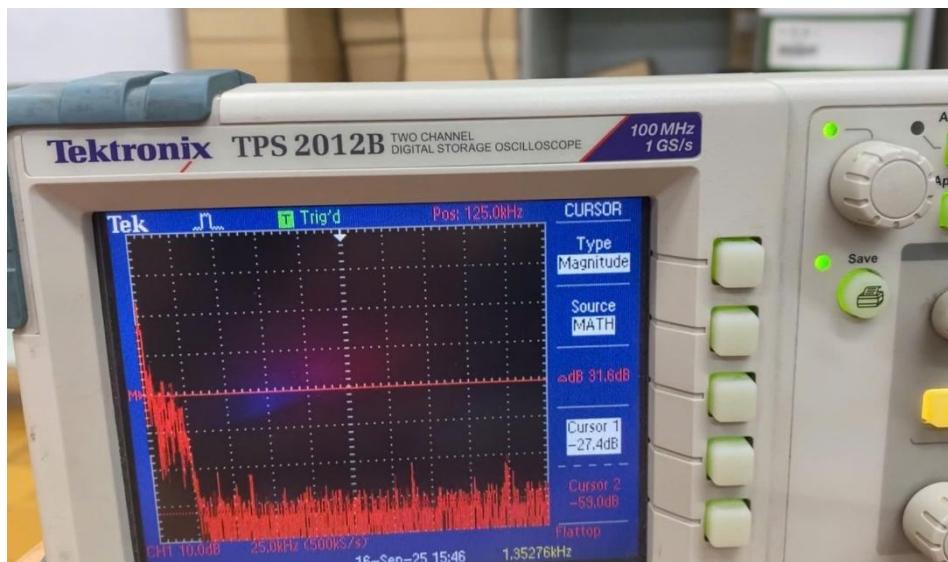
Hình 5-5: Dạng sóng ước lượng tín hiệu gốc vẫn còn nhiều thành phần nhiễu còn tồn tại làm biến dạng sóng sine gốc và đồng thời làm biến độ sóng lên 2.86Vpp. Do khi implement lên kit sẽ có nhiều vấn đề hơn khi mô phỏng wave form code RTL trên ModelSim nên ngõ ra này về mặt biên độ vẫn chấp nhận được. Nhưng để rõ hơn tiếp tục quan sát FFT của các thành phần tín hiệu xem biên độ miền nhiễu không mong muốn giảm đi bao nhiêu.



Hình 5-4: FFT của sóng sine 1KHz gốc



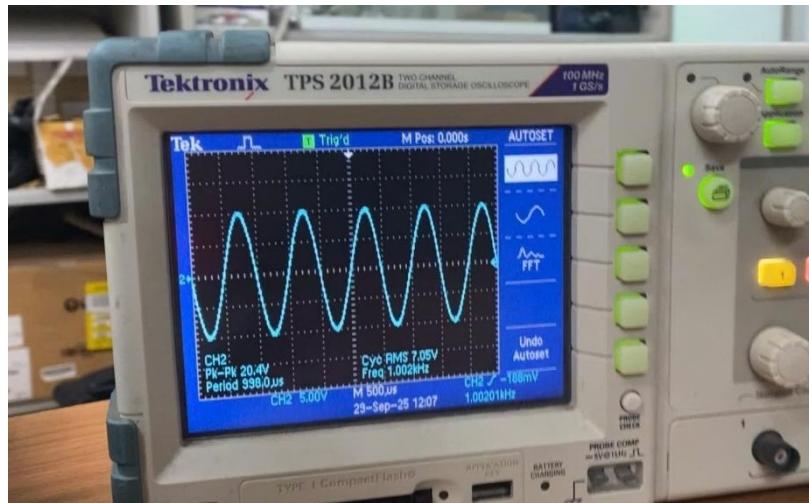
Hình 5-6: FFT của ngõ vào chính -15,8dB



Hình 5-7: FFT của ngõ ra ước lượng

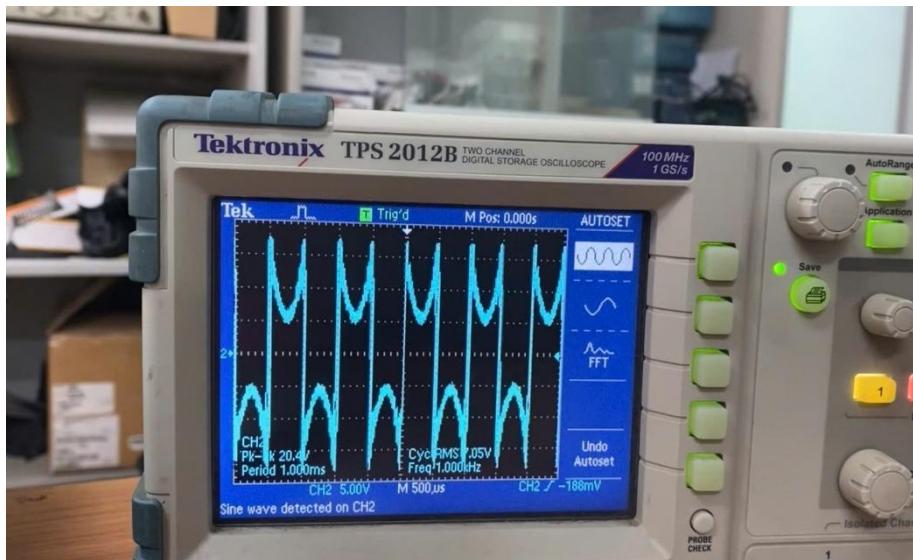
Nhận xét: có thấy thấy sau khi lọc phô của nhiễu đã được giảm gần 12dB trong khi thành phần sóng sine hầu như không bị giảm. Kết quả cho thấy adaptive filter đã thực hiện đúng chức năng của mình là giảm nhiễu và tăng cường chất lượng tín hiệu.

Thực hiện khử nhiễu sóng sine bị chèn nhiễu với phô nhiễu rỗng và biên độ nhiễu cao với $\mu = 0.002$ và số tap là 5.



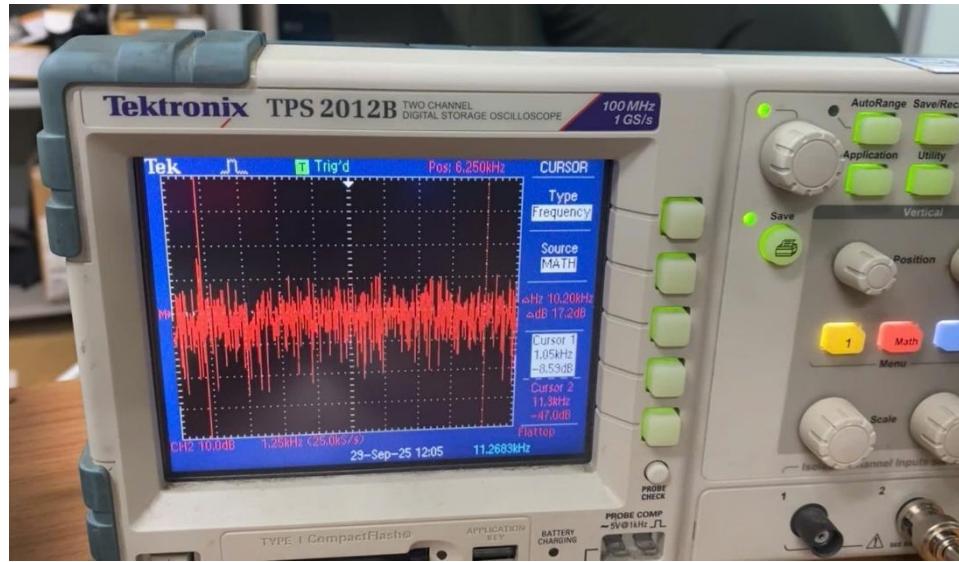
Hình 5-8: Tín hiệu có ích

Tín hiệu gốc có ích là sóng sin có tần số là 1KHz và biên độ là 10.2V được chèn thêm nhiễu trắng và xung vuông 1KHz có biên độ lớn dẫn tới ngõ vào chính là sóng buông bị đảo thành phần âm dương và bị nhiễu trắng nặng.



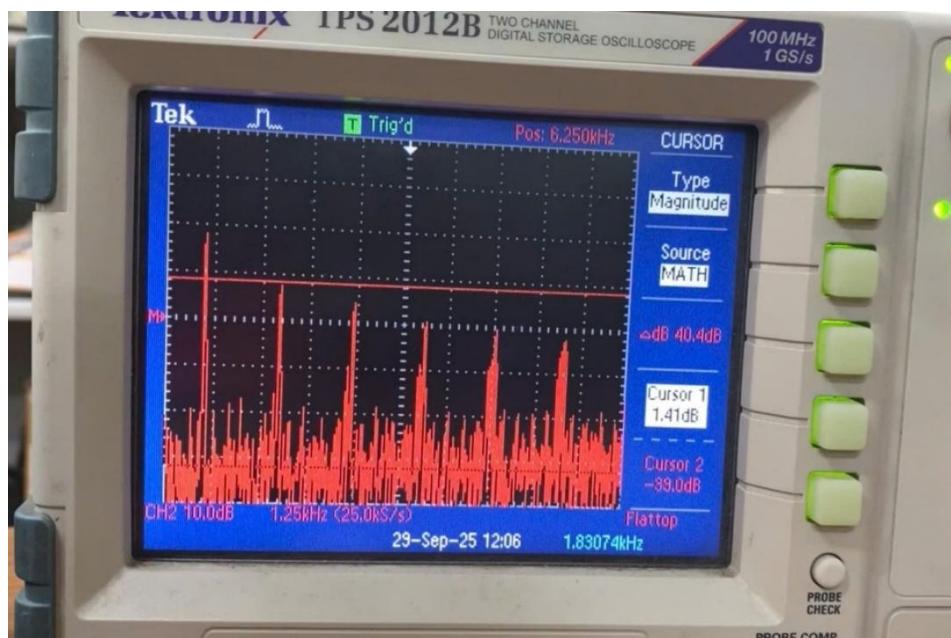
Hình 5-9: Dạng sóng của ngõ vào chính

Nhận xét: Với ngõ vào chính mất hoàn toàn hình dạng sóng sine có ích ban đầu và bị nhiễu bởi sóng vuông cùng tần số là rất nặng và thêm vào đó là nhiễu trắng làm cho hình dạng các đỉnh sóng còn sót lại bị đậm hơn do bị cộng các biên độ nhiễu trắng.



Hình 5-10: FFT của ngõ vào chính

Thực hiện dùng con trỏ để đo tần số cho thấy có 2 thành phần phổ có tần số sấp xỉ 1KHz do sai số khi tạo phần cứng nên 2 tín hiệu chưa tạo thành 1 phổ 1KHz duy nhất để chồng lên nhau. Với cường độ nhiễu có thể sấp xỉ 2dB.



Hình 5-11: FFT của tín hiệu sau lọc

Nhận xét: mặc dù vẫn có vài phổ nhiễu vẫn không bị suy giảm vẫn ở mốc là 1.41dB nhưng các phần phổ khác thì hầu như đã được triệt tiêu hoàn toàn. Với bộ lọc thích nghi có 5 tap, so với 10 tap như ở trên bộ lọc vì số tap ít hơn nên ít bị tràn khi tính FP hơn và ổn định hơn.

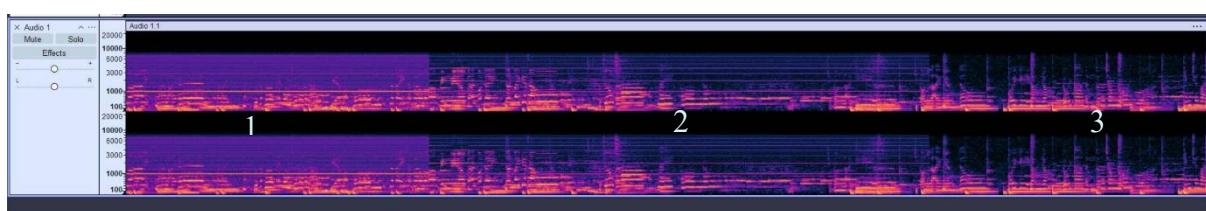
5.2. Kết quả thực hiện lọc audio bằng bộ lọc thích nghi



Hình 5-12: Kết quả file lọc âm thanh

-Âm thanh nghe được sẽ được đánh giá theo thang DCR ở mục 2.2.2

-Để rõ hơn về mặt biên độ và tần số một cách dễ nhất để kiểm chứng hiệu quả lọc là xem spectrogram của âm thanh bằng Audacity. Với Spectrogram sẽ có 3 chiều: Ngang là thời gian đọc là tần số và màu là biên độ càng sáng tức là biên độ càng lớn.



Hình 5-13: Spectrogram của âm thanh từ Audacity

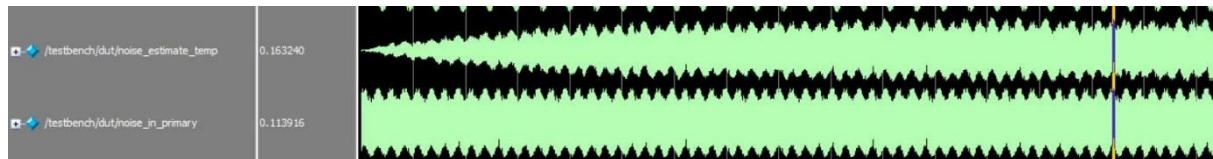
-Nhận xét: miền 1 là miền Primary input có thể thấy rằng gần như mọi thành phần tần số đều có biên độ mạnh. Miền 2 là miền của âm thanh gốc. Miền 3 là miền ước lượng tín hiệu gốc so với miền 1 thì miền 3 đã giống miền 2 hơn rất nhiều nhưng vẫn chưa đáng kể. Và miền cuối cùng là miền ước lượng nhiễu cho thấy nhiễu khá dày và mạnh ở tần số cao.

-Nếu âm thanh nghe được sẽ được đánh giá theo thang DCR ở mục 2.2.2, với biên độ nhiễu nhỏ có thể lọc được khá tốt, nhưng với biên độ nhiễu lớn hơn nhiều so với âm thanh gốc, tín

hiệu âm thanh ước lượng ngõ ra bị méo dạng, và khi nghe ước lượng nhiễu có một vài âm thanh gốc cho thấy bộ lọc đã học sai một phần nào đó.

Vấn đề khi thực hiện trên FPGA.

Khác với mô phỏng trên tool, do chạy trên kit các mức tín hiệu không được lý tưởng dẫn đến bộ lọc có thể lọc sai và ước lượng nhiễu thay vì tăng tới mức đúng với mức nhiễu được trộn vào tín hiệu gốc và tăng tới vô cực.

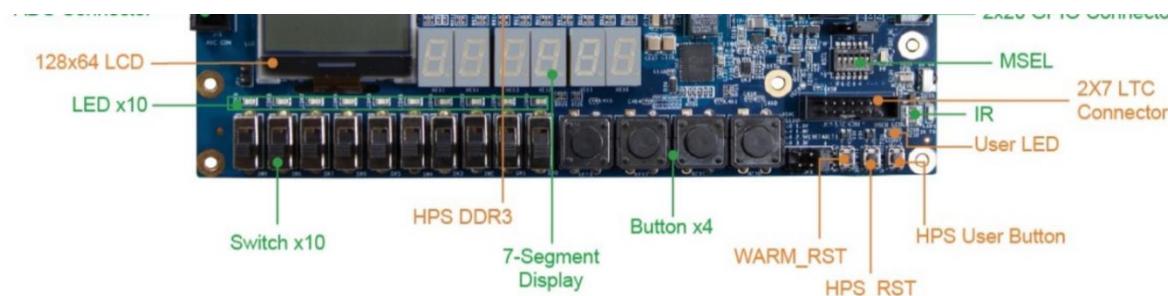


Hình 5-14: Ước lượng nhiễu được mô phỏng ở tool với điều kiện lý tưởng

Do đó $y[n]$ tăng tới vô cùng thì khi trừ Floating point với $d[n]$ sẽ ra – vô cùng, và theo như design thì các giá trị vô định và vô cùng em sẽ gán bằng 0 điều đó dẫn tới âm thanh ngõ ra bị mất hoàn toàn kênh dẫn, và cần bấm nút reset để tính lại các thành phần bộ lọc.

Bảng 5-1: Bảng các switch và nút bấm tương tác với FPGA

Port	Điều kiện	Mô Tả
sw[9]		Reset hệ thống
sw[8]		Cho phép chọn vào điều chỉnh biên độ nhiễu
sw[7:5]		Chọn ngõ ra của bộ lọc thích ứng
sw[4:0]	sw[8]	Chọn thành phần nhiễu
btn[2:0]	sw[8]=0	Tiến hành cấu hình codec (ADC và DAC)
btn[1:0]	sw[8]	Tăng/giảm biên độ nhiễu
btn[3:2]	sw[8]	Tăng/giảm tần số nhiễu
sw[7:5] =0	sw[8]	Sóng Sine
sw[7:5] =1		Sóng vuông biên độ lớn
sw[7:5] =2		Sóng tam giác
sw[7:5] =3		Sóng răng cưa



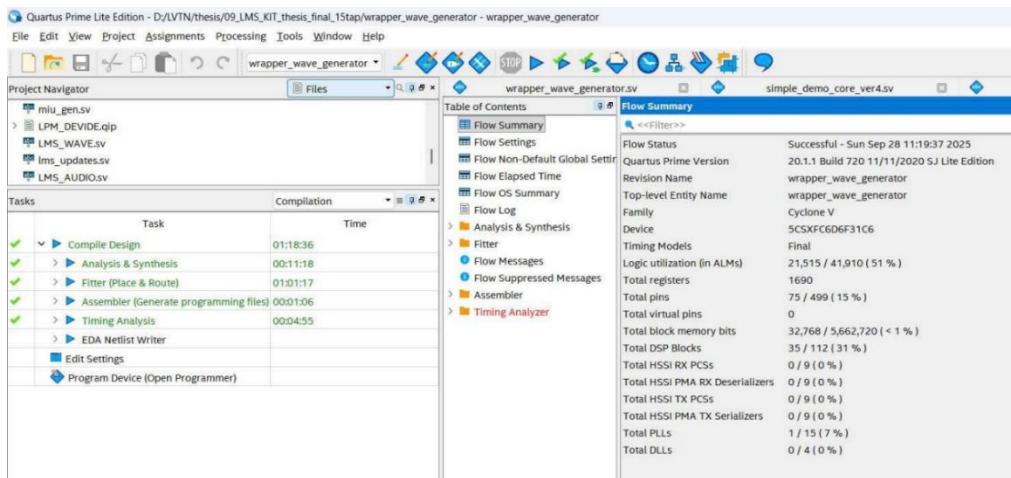
Hình 5-15: Các switch và nút bấm để tương tác với bộ lọc

5.3. So sánh hiệu quả bộ lọc khi thay đổi các thông số bộ lọc

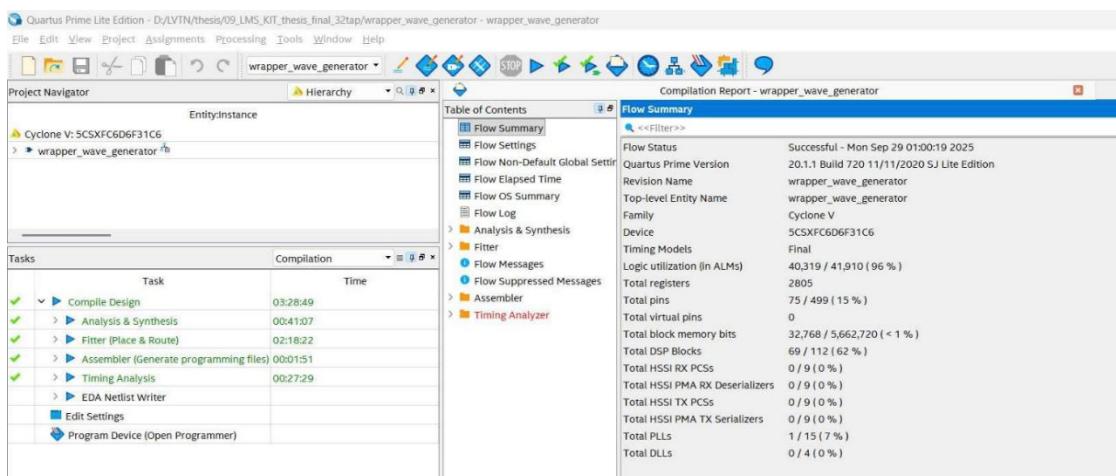
Để so sánh hiệu quả bộ lọc khi tăng dần số lượng tap với $\mu = 0.002$ cố định.

Bảng 5-2: So sánh hiệu quả khi thay đổi thông số bộ lọc

	Bộ lọc 5 tap	Bộ lọc 10 tap	Bộ lọc 32 tap
Tràn khi tính FP	Ít khi tràn	Thường xuyên tràn	Luôn luôn tràn
Khả năng lọc về mặt biên độ	Lọc tốt	Lọc tốt	Mất kênh dẫn nên không thể lọc
Khả năng lọc xét về mặt đáp ứng	Các tần số gần với tín hiệu sạch có khả năng bị sót lại	Các tần số gần với tín hiệu sạch có khả năng bị sót lại nhưng ở vùng xa hơn nên so với chỉ 5 tap	
Thời gian tổng hợp mạch	30 phút	1 tiếng	3 tiếng
Tiêu tốn tài nguyên	10%	15%	96%



Hình 5-16: Kết quả và thời gian tổng hợp mạch với số tap là 15



Hình 5-17: kết quả và thời gian tổng hợp mạch với số tap là 32

6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

6.1.1. Đánh giá kết quả lọc tín hiệu sóng cơ bản

Kết quả thực nghiệm trên các tín hiệu sóng cơ bản (chẳng hạn như sóng sin bị nhiễu) cho thấy bộ lọc LMS hoạt động ổn định và hiệu quả trong việc triệt tiêu nhiễu:

- Khả năng hội tụ nhanh sau một số lượng mẫu nhất định, phản ánh đặc tính thích ứng của thuật toán LMS trong việc ước lượng và loại bỏ thành phần nhiễu.
- Tín hiệu đầu ra sau lọc có biên độ và pha gần như khớp hoàn toàn với tín hiệu gốc lý tưởng, cho thấy độ chính xác cao trong điều kiện nhiễu đơn giản và môi trường giả lập.
- Điều này xác nhận tính đúng đắn của mô hình toán và khả năng hiện thực hóa phần cứng của thuật toán LMS trên nền FPGA, cả về mặt chức năng và thời gian thực.

6.1.2. Đánh giá kết quả lọc tín hiệu âm thanh thực

Khi áp dụng bộ lọc cho tín hiệu âm thanh thực thu từ môi trường, kết quả cho thấy một số hạn chế cần được xem xét:

- Bộ lọc thực hiện rất tốt và làm suy hao đúng tàn số nhiễu đối với các nhiễu có biên độ tương đối nhỏ, cho ra đáp ứng gần giống như ban đầu mặc dù vẫn còn thấy các thành phần hài của nhiễu nhưng ở biên độ nhiễu rất thấp. Còn các tín hiệu nhiễu lớn thì bộ lọc có dấu hiệu bị méo dạng thậm chí là bộ lọc nhận điện sai âm thanh gốc là nhiễu và cho ra ngoài ngõ ra ước tính nhiễu.
- Xét về độ hội tụ, em chưa có các nào để kiểm chứng độ hội tụ khi thực hiện trên FPGA bởi vì oscilloscope không cho tua ngược biên độ ở thời điểm trước đó.
- Một số yếu tố có thể ảnh hưởng bao gồm: cấu hình tham số ban đầu, giới hạn về độ rộng bit trong xử lý số cố định trên FPGA, và tốc độ lấy mẫu chưa tương thích tối đa với phổ tín hiệu đầu vào.
- Từ đó cho thấy, cần cải tiến thêm về thuật toán chẳng hạn sử dụng Normalized LMS hoặc Variable Step LMS cũng như tối ưu phần cứng nhằm phù hợp hơn với các tín hiệu âm thanh phức tạp.

6.2. Hướng phát triển

Sau quá trình nghiên cứu và triển khai bộ lọc thích nghi (adaptive filter) sử dụng giải thuật LMS (Least Mean Squares) trên nền tảng FPGA DE10, đề tài đã chứng minh tính hiệu quả của phương pháp trong việc cải thiện chất lượng tín hiệu trong môi trường có nhiễu. Tuy nhiên, để hệ thống có thể ứng dụng thực tiễn và mở rộng quy mô, vẫn còn nhiều hướng phát triển tiềm năng cần được tiếp tục khai thác trong các nghiên cứu sau:

- Trước hết, cần hướng đến tối ưu hóa phần cứng nhằm cải thiện hiệu năng và giảm thiểu mức tiêu thụ tài nguyên trên FPGA. Các phép toán nhân và cộng trong giải thuật LMS hiện đang sử dụng các khối logic cơ bản, vốn chiếm nhiều tài nguyên và có thể không phù hợp với các ứng dụng yêu cầu độ trễ thấp. Do đó, việc thay thế nhân số học bằng nhân dịch cộng (bit-serial multiplier), kết hợp chuẩn hóa LMS (Normalized LMS - NLMS) hoặc biến thể sử dụng dấu hiệu (Sign LMS) sẽ giúp cải thiện tốc độ hội tụ mà vẫn đảm bảo hiệu quả phần cứng.

- Một hướng phát triển quan trọng khác là xây dựng giao diện phần mềm hỗ trợ người dùng. Hiện tại, cấu hình hệ số và quan sát kết quả lọc chủ yếu dựa trên việc nạp dữ liệu offline. Trong tương lai, việc phát triển một ứng dụng đồ họa (GUI) hoặc trang web kết nối với FPGA sẽ cho phép người dùng điều chỉnh các tham số lọc như hệ số học (learning rate) hoặc khởi tạo trọng số ban đầu, đồng thời theo dõi chất lượng tín hiệu lọc một cách trực quan. Điều này đặc biệt cần thiết khi sản phẩm hướng tới ứng dụng thương mại hoặc triển khai diện rộng.

- Bên cạnh đó, việc so sánh và đánh giá các thuật toán thích nghi khác cũng là một hướng nghiên cứu cần thiết. Mặc dù LMS có độ phức tạp thấp và dễ triển khai, nó không phải là tối ưu trong mọi trường hợp, đặc biệt khi đối mặt với tín hiệu có nhiễu trắng không dừng hoặc các điều kiện biến đổi nhanh. Các thuật toán như Recursive Least Squares (RLS), Kalman Filter hoặc Affine Projection Algorithm (APA) nên được triển khai song song và đánh giá dựa trên các tiêu chí như: tốc độ hội tụ, độ phức tạp phần cứng, khả năng thích nghi và hiệu quả lọc.

Tóm lại, đề tài không chỉ dừng lại ở việc minh chứng nguyên lý hoạt động của LMS trên FPGA, mà còn đặt nền móng cho các hướng nghiên cứu sâu hơn trong lĩnh vực xử lý tín hiệu số, điều khiển thích nghi, và thiết kế hệ thống thông minh ứng dụng thực tiễn.

TÀI LIỆU THAM KHẢO

- [1]. Nguyen H.H., Pham X.T., Doan V.S. Hoang M.K., "Combining MUSIC Algorithm and Adaptive Beamforming to Improve Online Call Quality," in Ad Hoc Networks (ADHOCNETS 2023), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 558, Springer, Cham, 2024.
- [2]. W. B. Kleijn, F. Lim, "Robust and low-complexity blind source separation for meeting rooms," in 2017 Hands-free Speech Communications and Microphone Arrays (HSCMA), San Francisco, CA, USA, 156-160, 2017. doi: 10.1109/HSCMA.2017.7895581.
- [3]. F. Ma, W. Zhang, T. D. Abhayapala, "Active Control of Outgoing Broadband Noise Fields in Rooms," IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28, 529-539, 2020. doi: 10.1109/TASLP.2019.2960716.
- [4]. Q. Qiongfang, G. Bo, T. Mengnan, C. Yongchao, "Statistical Energy Analysis-Based Sound Absorption Coefficient Calculation Model and Noise Reduction," in 2023 IEEE 11th International Conference on Computer Science and Network Technology (ICCSNT), Dalian, China, 503-509, 2023. doi: 10.1109/ICCSNT58790.2023.10334611.
- [5]. Yuze Sun, Ji Zhang, Xiaopeng Yang, "Design of experimental adaptive beamforming system utilizing microphone array," in IET International Radar Conference 2013, Xi'an, 1-5, 2013. doi: 10.1049/cp.2013.0242.
- [6]. Y. Sun, X. Yang, L. Guo, T. Long, "Experimental array signal processing demonstration system by utilizing microphone array," in 2016 CIE International Conference on Radar (RADAR), Guangzhou, China, 1-5, 2016. doi: 10.1109/RADAR.2016.8059533.
- [7]. H. S. Vu, K. T. Truong, L. T. Bang, V. Y. Vu, M. T. Le, "An Investigation of Adaptive Digital Beamforming Antenna for gNodeB 5G," in 2019 International Conference on Advanced Technologies for Communications (ATC), Hanoi, Vietnam, 221-224, 2019. doi: 10.1109/ATC.2019.8924509. [8]. R. Suleesathira, "Direction of Arrival Identification Using MUSIC Method and NLMS Beamforming," in 2020 15th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP), Bangkok, Thailand, 1-6, 2020. doi: 10.1109/iSAI-NLP51646.2020.9376838.
- [8]. Jingdong, C., Jacob, B., Arden, Huang. (2007). On the optimal linear filtering techniques for noise reduction. *Speech Communications*, 49(2), 305-316.

- [9]. Akagi, M., Mizumachi, M. (1997). Noise reduction by paired microphones. In Proceedings of EUROSPEECH, 1997, 335-338.
- [10]. Cohen, I., Berdugo, D. (2001). Speech enhancement for nonstationary noise environments. *Signal Processing*, 81(11), 2403-2418.
- [11]. Hamid, H. (2008). A time-frequency approach for noise reduction. *Digital Signal Processing*, 18(5), 728-738.
- [12].bitzer, J., Simmer, K. U. (2001). Super directive microphone arrays. In: *Microphone arrays signal processing techniques and applications*. Springer, Berlin, 19-38.
- [13]. Simmer, K. M.bitzer, J.Marro, C. (2001). Post-filtering techniques. In: *Microphone arrays signal processing techniques and applications*. Springer, Berlin, 39-60.
- [14]. Akagi, M., Kago, T. (2002). Noise reduction using a small-scale microphone array in multi noise source environment. In Proceedings of ICASSP, 2002, 909-912.
- [15]. Ming, J., Srinivasan, R., Crookes, D. (2011). A Corpus-based approach to speech enhancement from nonstationary noise. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4), 822-836.
- [16]. Mcaulay, R. J., Malpass, K. L. (1980). Speech enhancement using a minimum mean-square error short time spectral amplitude estimator. *IEEE Transactions on Acoustic, Speech, Signal Processing*, 28(2), 137-145.
- [17]. Haykin, S (2001). Minimum mean square error adaptive filter. In: *Adaptive Filter Theory*, 4th ed. Prentice Hall, Upper Saddle River, 183-228.
- [18]. Lotter, T., Vary, P. (2005). Speech enhancement by MAP spectral amplitude estimation using a super-Trāngian speech model. *EURASIP Journal in Applied Signal Processing*, 2005(1), 1110-1126.
- [19]. Alouane, M., Jaïdane, M. (2006). A new nonstationary LMS algorithm for tracking Markovian time varying systems. *Signal Processing*, 86(1), 50-70.

PHỤ LỤC

Trong phần này, sinh viên có thể trình bày:

- ❖ Những kết quả nghiên cứu bổ sung mà trong phần Kết quả đồ án chưa trình bày hết.
- ❖ Phần mã nguồn chương trình, sinh viên cũng có thể trình bày trong mục này. Để ngắn gọn, sinh viên chỉ đưa những mã nguồn chính vào phần Phụ lục.
- ❖ Sơ đồ toàn mạch chi tiết

```

function [filtered_signal, w, noise_estimate] = LMS_filter(in, d, o, miu)
    % Adaptive Noise Canceller - NLMS
    N = length(in),
    w = zeros(o, 1),
    y = zeros(N, 1),
    e = zeros(N, 1), % noise estimate

    if miu > 0
        for i = o:N
            x = in(i:-1:i-o+1)';
            y(i) = w' * x,
            e(i) = d(i) - y(i), % Estimate of noise in d

            den = x' * x + 1e-6,
            % w = w + miu * e(i) * x / den,
            w = w + miu * e(i) * x,
        end
    end

    filtered_signal = e(:,1), % Final signal ≈ s
    noise_estimate = y(:,1), % Noise we've subtracted
end

```