**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN - ĐIỆN TỬ**

**NĂM HỌC 2023 - 2024  
-------\*-------**

****

**BÀI TẬP LỚN**

**Đề tài:**

**Thiết Bộ Lọc FIR**

**Cho Ứng Dụng Nhạc BASS**

GVHD: Ths. Nguyễn Tuấn Hùng

Nhóm: 08– L01 - HK242

**TP.HCM, ngày 04 tháng 05 năm 2024**

**MỤC LỤC**

[CHƯƠNG 1: GIỚI THIỆU VỀ ĐỀ TÀI 5](#_Toc165853007)

[1.1. Bộ lọc FIR và ứng dụng 5](#_Toc165853008)

[1.2. KIT DE2 Altera 8](#_Toc165853009)

[1.3. Phần mềm Quartus và ModelSim 8](#_Toc165853010)

[CHƯƠNG 2: PHẦN CỨNG 10](#_Toc165853011)

[2.1. Giới thiệu về AUDIO Trên KIT DE2 10](#_Toc165853012)

[2.2. Giới thiệu về WM8731 11](#_Toc165853013)

[CHƯƠNG 3: THIẾT KẾ VÀ MÔ PHỎNG 20](#_Toc165853014)

[3.1. Thiết bộ lọc bằng matlab 20](#_Toc165853015)

[3.2. Kiểm tra bộ lọc và tạo các file âm thanh bằng Python 24](#_Toc165853016)

[3.3. Mô phỏng bộ lọc trên ModelSim 27](#_Toc165853017)

[3.4. Phân tích về các độ trễ của mạch 31](#_Toc165853018)

[3.4. Đánh giá hiệu quả của Filter 32](#_Toc165853019)

[3.5. Thiết kế bộ Serial to Parallel 35](#_Toc165853020)

[3.6. Thiết kế bộ Parallel to Serial. 36](#_Toc165853021)

[CHƯƠNG 4: KẾT QUẢ 38](#_Toc165853022)

[4.1. Sơ đồ kết nối của mạch 38](#_Toc165853023)

[4.2. Kết quả đổ KIT. 40](#_Toc165853024)

[TÀI LIỆU THAM KHẢO 41](#_Toc165853025)

**DANH MỤC HÌNH ẢNH**

[Hình 2-1: Một số dãy tần số và ứng dụng 4](#_Toc165803278)

[Hình 2-2: Sơ đồ nguyên lý kế nối Audio CODEC của Altera DE2 7](#_Toc165803279)

[Hình 2-3: Các chân tín hiệu của DE2. 7](#_Toc165803280)

[Hình 2-4: Cách hoạt động của các Clock 8](#_Toc165803281)

[Hình 2-5: Đặc điểm của một số tín hiệu 9](#_Toc165803282)

[Hình 2-6: Các chế độ hoạt động của WM8731 10](#_Toc165803283)

[Hình 2-7: Sử dụng I2S Mode với INVERT của clock BLCK 11](#_Toc165803284)

[Hình 2-8: Thanh ghi cấu hình Mode 11](#_Toc165803285)

[Hình 2-9: Bảng cấu hình clock ở chế độ lấy mẫu bình thường 12](#_Toc165803286)

[Hình 2-10: Thanh ghi cài đặc xung clock 13](#_Toc165803287)

[Hình 2-11: Sơ đồ khối nguyên lý hoạt động của dữ liệu đầu vào 13](#_Toc165803288)

[Hình 2-12: Thanh ghi cấu hình tín hiệu đầu vào 14](#_Toc165803289)

[Hình 2-13: Tổng hợp thanh ghi của WM8731 15](#_Toc165803290)

[Hình 2-14: Các tín hiệu của chip WM8731 16](#_Toc165803291)

[Hình 2-15: Sơ đồ nguyên lý chỉ có bộ lọc FIR Lowpass filer 35](#_Toc165803292)

[Hình 2-16: Sơ đồ nguyên lý bộ lọc Trung bình kết nối với bộ Lowpass filter 35](#_Toc165803293)

[Hình 3-1: Kiểm tra tín hiệu trong vùng tần số vào sẽ nghe tốt 17](#_Toc165803258)

[Hình 3-2: Thiết kế bộ lọc trên bằng công cụ Filter Design của Matlab 18](#_Toc165803259)

[Hình 3-3: Lấy hệ số và chuyển từ hệ số chuẩn hóa sang 16 bit có dấu 18](#_Toc165803260)

[Hình 3-4: Sơ đồ khối bộ lọc FIR cơ bản 20](#_Toc165803261)

[Hình 3-5: Khối FIR sau khi thiết kế. 20](#_Toc165803262)

[Hình 3-6: Phổ của tín hiệu ở kênh trái file âm thanh. 22](#_Toc165803263)

[Hình 3-7: Phổ tín hiệu kênh trái sau khi thêm nhiễu 23](#_Toc165803264)

[Hình 3-8: Tạo testbench mô phỏng cho bộ lọc 25](#_Toc165803265)

[Hình 3-9: Tín hiệu sau khi đi qua bộ lọc FIR 26](#_Toc165803266)

[Hình 3-10: Tín hiệu sau khi đi qua bộ lọc FIR 27](#_Toc165803267)

[Hình 3-11: Bộ lọc FIR trung bình 30](#_Toc165803268)

[Hình 4-1: RTL View chỉ có bộ lọc FIR Lowpass filer 36](#_Toc165803303)

[Hình 4-2: RTL View của bộ lọc Trung bình kết nối với bộ Lowpass filter 36](#_Toc165803304)

**TÓM TĂT ĐỀ TÀI**

**Trong đề tài này chúng tỗi đã thực hiện:** Thiết kế bộ lọc cho ứng dụng máy âm thanh bass (mục tiêu cho người muốn nghe nhạc bass).

+ Đưa ra yêu cầu thiết kế chính là: Bộ lọc FIR Lowpass filter: với tần số cắt là 5kHz, biên độ ngoài cùng tầng số cắt là -60dB.

+ Thiết kế bộ lọc bằng Matlab và lấy các hệ số và viết bộ lọc bằng verilog

+ Thiết kế bộ lọc trung bình bình 20 tap dựa trên core là bộ lọc FIR low pass filter được chỉnh sửa lại.

+ Mô phỏng bộ lọc bằng Python

+ Tạo các tập test .wave bằng Python

+ Tạo tín hiệu giả và nhiễu để làm tập test cho ModelSim

+ Mô phỏng bộ lọc FIR Lowpass filter bằng ModelSim

+ Thiết lập cấu hình chip WM8731

+ Viết driver để đọc dữ liệu từ chip WM8731, bao gồm 2 bộ chuyển đổi từ là từ nối tiếp sang song song và từ song song sang nối tiếp

+ Mô phỏng và xác minh khối parallel sang serial và ngược lại bằng Quartues.

+ Thực hiện triển khai bộ lọc trên KIT và kết quả chưa hiệu quả.

# **CHƯƠNG 1: GIỚI THIỆU VỀ ĐỀ TÀI**

## 1.1. Bộ lọc FIR và ứng dụng

Các thao tác của sử lý dùng để để làm biến dạng sự phân bố tần số của các thành phần của một tín hiệu theo các chỉ tiêu đã cho nhờ một hệ thống số được gọi là việc lọc số. Dải tần số mà bộ lọc cho đi qua được gọi là dải thông. Dải tần số mà bộ lọc không cho đi qua được gọi là dải chặn.

Bộ lọc số là một hệ thống dùng để làm biến dạng sự phân bố tần số của các thành phần của một tín hiệu theo các chỉ tiêu đã cho.

Các mạch lọc số cho tín hiệu số có phổ nằm trong 1 dải tần số nhất định đi qua và không cho tín hiệu có phổ nằm ngoài dải tần số đó đi qua.

Dựa vào đáp ứng tần số, có thể chia bộ lọc ra làm nhiều loại, trong đó có bộ lọc thông thấp. Bộ lọc thông thấp (LPF - Low Pass Filter) là một loại bộ lọc điện tử hoặc kỹ thuật số cho phép các tín hiệu có tần số thấp hơn một giá trị cắt đi qua và ngăn chặn các tín hiệu có tần số cao hơn. Nó được sử dụng để loại bỏ nhiễu tần số cao và làm mịn tín hiệu đầu vào

Đặc điểm của bộ lọc thông thấp:

+ Đáp ứng tần số: Đáp ứng tần số của bộ lọc thông thấp giảm dần theo tần số.

+ Tần số cắt: Tần số cắt là tần số mà tại đó độ suy giảm của bộ lọc đạt 3 dB (khoảng một nửa công suất).

+ Độ dốc: Độ dốc của bộ lọc thông thấp mô tả tốc độ giảm độ suy giảm của nó khi tần số tăng.

Bộ lọc số có đặc tính xung hữu hạn (bộ lọc số FIR) là một loại bộ lọc được phân biệt dựa vào dạng đặc tính xung. Bộ lọc FIR (Finite Impulse Response) là một loại bộ lọc kỹ thuật số được sử dụng để xử lý tín hiệu số. Nó được đặc trưng bởi:

+ Đáp ứng xung hữu hạn (Finite Impulse Response): Đáp ứng của bộ lọc đối với một xung đơn vị chỉ kéo dài trong một số hữu hạn các mẫu thời gian.

+ Sử dụng hệ số bộ lọc: Bộ lọc FIR được xác định bởi một tập hợp các hệ số bộ lọc, được sử dụng để nhân với các mẫu dữ liệu đầu vào.

+ Dễ dàng triển khai: Bộ lọc FIR có thể được triển khai bằng phần cứng hoặc phần mềm, và có thể được thiết kế với nhiều đặc tính khác nhau.

Một bộ lọc đáp ứng xung hữu hạn với hàm hệ thống có dạng:



Như vậy đáp ứng xung h(n) là:

Ảnh có chứa Phông chữ, văn bản, chữ viết tay, màu trắng

Mô tả được tạo tự động

Và phương trình sai phân là:

y(n) = b0 x(n) + b1x(n −1) +Λ + b M−1x(n − M +1)

Đây chính là tích chập tuyến tính của các dãy hữu hạn.

Bậc của bộ lọc là M-1, trong khi chiều dài của bộ lọc là M (bằng với số lượng các hệ số). Các cấu trúc bộ lọc FIR luôn luôn ổn định, và tương đối đơn giản hơn so với các cấu trúc bộ lọc IIR. Hơn thế nữa, các bộ lọc FIR có thể được thiết kế để có một đáp ứng pha tuyến tính và đó là điều cần thiết trong một số ứng dụng.

Chúng ta sẽ xem xét lần lượt các cấu trúc của bộ lọc FIR sau đây:

Cấu trúc dạng trực tiếp : Cấu trúc dạng trực tiếp được cho trong hình 1.2 với M = 5: Ảnh có chứa hàng

Mô tả được tạo tự động

Hình 1-1: Cấu trúc lọc FIR dạng trực tiếp

Cấu trúc dạng ghép tầng: Cấu trúc dạng ghép tầng được cho trong hình 1.3 với M = 7:

Ảnh có chứa hàng, biểu đồ, văn bản

Mô tả được tạo tự động

Hình 1-2: Cấu trúc lọc FIR dạng ghép tầng

Cấu trúc dạng pha tuyến tính: Đối với M lẻ: M = 7, còn đối với M chẵn: M = 6

Ảnh có chứa biểu đồ, hàng, nghệ thuật gấp giấy origami

Mô tả được tạo tự động

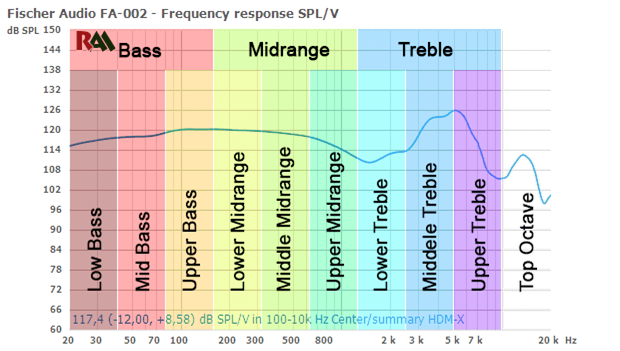
Hình 1-3: Cấu trúc lọc FIR pha tuyến tính với các hệ số M chẵn và lẻ

**Một số ứng dụng âm thanh thông thường như:**

**Jazz**: Tần số cơ bản trong âm nhạc jazz thường nằm trong khoảng trung bình và cao. Các nhạc cụ như piano, saxophone, trumpet thường tạo ra các tần số trung bình và cao. Bass và trống thường được sử dụng để tạo ra các nhịp và âm trầm ổn định, nhưng chúng không thường xuất hiện ở mức tần số cao.

**Pop**: Âm nhạc pop thường có các thành phần âm thanh đa dạng, từ các nhạc cụ truyền thống đến các hiệu ứng âm thanh điện tử. Tần số trung bình và cao thường được sử dụng nhiều để tạo ra các giai điệu và âm thanh chính trong các bài hát pop. Bass thường được tăng cường để tạo ra nhịp và âm trầm mạnh mẽ, nhưng không quá chiếm phần lớn so với các tần số khác.

**Bass**: Thể loại âm nhạc bass thường tập trung vào các tần số thấp hơn. Bass được coi là yếu tố chính trong các bài hát bass, với các tần số dưới 250Hz thường được tăng cường. Các tần số cao hơn cũng có thể xuất hiện, nhưng thường ít quan trọng hơn so với tần số bass.

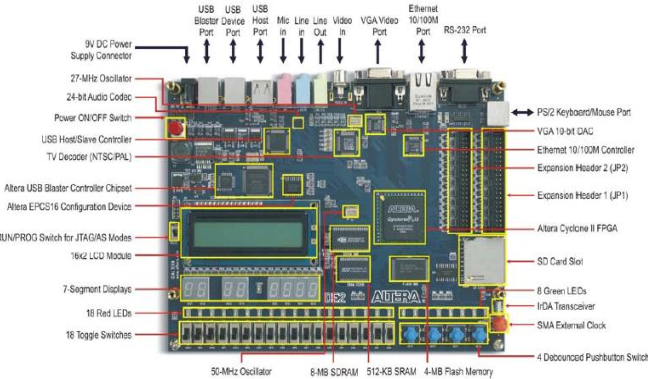


Hình 2-1: Một số dãy tần số và ứng dụng

***Do đó chúng tôi chọn kế bộ loc thông thấp có tấn số cắt là 5kHz. Ứng dụng trong bộ Bass, dành cho nhữn người thích nghe nhạc pass.***

## 1.2. KIT DE2 Altera

Board Altera DE2 được thiết kế với nhiều tính năng đa dạng dựa trên số lượng lớn các ngoại vi nhằm phục vụ cho các nghiên cứu khác nhau (ví dụ như: nghiên cứu và phát triển về các lĩnh vực luận lý số học (digital logic), tổ chức máy tính (computer organization) và FPGA). Trên kit này, một FPGA CHIP họ Cyclone II được tích hợp, và các ngõ vào/ra (IO) của Chip được kết nối với tất cả các CHIP khác trên kit DE2 (như CHIP TV Decoder, Ethernet 10/100M Controller, SRAM..) nhằm giúp cho người dùng có thể thay đổi các ứng dụng hay cấu hình nhằm mong muốn nhằm hướng đến một ứng dụng cụ thể. Trong bài báo cáo này, nhóm tác giả sử dụng Kit DE2 nhằm thực hiện việc lọc nhiễu của một tín hiệu âm thanh.



Hình. Kit Altera DE2

## 1.3. Phần mềm Quartus và ModelSim

***Phần mềm Quatus***

Phần mềm Quartus II được phát triển bởi công ty Altera là một trong những phần mềm đi kèm với các thiết bị phần cứng của công ty này nhằm tạo nên giao diện tương tác giữa người dùng. Quartus cung cấp một số tính năng hữu ích trong việc thiết kế logic sau:

 + Môi trường thiết kế gồm các bản vẽ, sơ đồ khối, công cụ soạn thảo các ngôn ngữ: VHDL, Verilog HDL.

+ Khả năng mô phỏng chức năng và thời gian.

+ Phân tích thời gian.

+ Phân tích logic nhúng với công cụ phân tích SignalTap II.

+ Cho phép xuất, tạo và kết nối các file nguồn để tạo ra các file chương trình.

+ Tự động định vị lỗi.

+ Khả năng lập trình và nhận diện linh kiện.

Trong bài báo cáo này, nhóm tác giả thực hiện các thiết kế trên phần mềm Quartus II 13.0 sp1 (64-bit).

***Phần mềm Model Sim***

ModelSim là chương trình mô phỏng và debug cho các thiết kế ASIC và FPGA cực kỳ mạnh mẽ và hiệu quả, khi thiết kế FPGA chúng ta có thể dùng Modelsim để mô phỏng, kiểm tra chương trình…

Model Sim cung cấp một số tính năng hữu ích trong việc mô phỏng, kiểm tra chương trình như:

+ Mô phỏng hành vi, RTL và mã cấp cổng.

+ Tự động cập nhật các nguồn, tín hiệu, Quy trình, và biến cửa sổ.

+ Hỗ trợ kết hợp với HDL Designer và HDL Author để quản lý dự án.

+ ModelSim giúp giảm bớt quá trình tìm lỗi thiết kế với môi trường gỡ lỗi được thiết kế thông minh, hiển thị hiệu quả dữ liệu thiết kế để phân tích và gỡ lỗi tất cả các ngôn ngữ mô tả phần cứng. Một loạt các khả năng trực quan cho Verilog, VHDL và SystemC làm cho nó trở thành lựa chọn lý tưởng cho thiết kế ASIC và FPGA.

Trong bài báo cáo này, nhóm tác giả thực hiện quá trình mô phỏng và kiểm tra chương trình trên phần mềm ModelSim-Altera 10.1d.

# **CHƯƠNG 2: PHẦN CỨNG**

## 2.1. Giới thiệu về AUDIO Trên KIT DE2

Ảnh có chứa văn bản, biểu đồ, Kế hoạch, sơ đồ

Mô tả được tạo tự động

Hình 2-2: Sơ đồ nguyên lý kế nối Audio CODEC của Altera DE2

Bảng mạch DE2 cung cấp âm thanh chất lượng cao 24-bit thông qua chip mã hóa/ giải mã âm thanh Wolfson WM8731 (enCOder/DECoder). Chip này hỗ trợ cổng microphone-in, line-in và line-out, với tốc độ mẫu có thể điều chỉnh từ 8 kHz đến 96 kHz. WM8731 được điều khiển thông qua giao diện bus I2C nối tiếp, được kết nối với các chân trên FPGA Cyclone II. Thông tin chi tiết về việc sử dụng codec WM8731 có sẵn trong tài liệu kỹ thuật, có thể tìm thấy trên trang web của nhà sản xuất, và từ thư mục Datasheet trên CD-ROM Hệ thống DE2.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Hình 2-3: Các chân tín hiệu của DE2.

Do DE2 sử dụng chip WM8731 do đó ta sẽ đọc datasheet của nó để biết rõ cách để giao tiếp.

## 2.2. Giới thiệu về WM8731

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Việc giao tiếp âm thanh sẽ bao gồm tổng cộng 2 kênh là kênh trái và kênh phải. Khi đọc âm thanh ta cũng phải đọc cả hai kênh, và khi xuất dữ liệu ra, chúng ta cũng xuất dữ liệu trên cả hai kênh. Mỗi kênh sẽ có tần số lấy mẫu là fs=fclock/1024=50MHz/1024 =48.828kHz.

Sau đây tôi sẽ giới thiệu các chân tín hiệu để giao tiếp với chip WM8731:

* Mclk => A 12.5 MHz master clock. It’s the WM8731’s internal operation clock.
* bclk  *=>* A 3.125 MHz bit clock (mclk 4 ).
* adclrc *=>* A left/right selector for adcdat. adclrc=’1’ for left.
* adcdat *=>* Serial bits from the ADCs (one bit per bclk pulse).
* daclrc *=>* A left/right selector for dacdat. daclrc=’1’ for left.
* dacdat *=>* Serial bits to the DACs (one bit per bclk pulse)

Ảnh có chứa văn bản, biểu đồ, ảnh chụp màn hình, hàng

Mô tả được tạo tự động

Hình 2-4: Cách hoạt động của các Clock

Tín hiệu ADCLRC và DACLRC giống như tín hiệu lấy mẫu. Trong một chu kỳ của xung clock, ở nữa chu kỳ đầu tiên nó sẽ cung cấp dữ liệu ở kênh trái, ở nữa chu kỳ còn lại sẽ cung cấp dữ liệu ở kênh phải. Việc thiết lập dữ liệu kênh trái, kênh phải xuất hiện ở chu kỳ âm hay dương có thể thiết thập bằng các thanh ghi cấu hình sẽ được trình bày sau.

Còn đối với tín hiệu BLCK, đây là tín hiệu để chuyển từng bit ra cho ta đọc. Tín hiệu này giúp ta đọc các bit data, trong một chu kỳ BLCK sẽ có một bit data ở DACDAT và ADCDAT được xuất hiện, từ đó ta sẽ đọc các tín hiệu đó vào trong các thanh ghi của chúng ta để lưu lại mẫu.

Trong báo cáo này chúng ta chủ yếu sử dụng 2 clock là chính là BLCK và LRCK, chỉ cần hai clock này là chúng ta có thể đọc dữ liệu từ chip WM8731, còn các clock khác ta không quan tâm nhiều.

Dưới đây là một số thông số về clock có thể được dùng trong timming để xác nhận khối chúng ta viết đã đúng hay chưa.

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động

Hình 2-5: Đặc điểm của một số tín hiệu

Đầu tiên để điều khiển được WM8731, ta cần biết các chế độ hoạt động của nó cài đặc nó như thế nào.

**Cấu hình MODE hoạt động của WM8731**

Ảnh có chứa văn bản, biểu đồ, ảnh chụp màn hình, hàng

Mô tả được tạo tự độngẢnh có chứa văn bản, số, biên lai, Phông chữ

Mô tả được tạo tự động

Hình 2-6: Các chế độ hoạt động của WM8731

Có 4 chế độ hoạt động chính là Left Justified Mode, I2S Mode, Right Justified Mode và DSP Mode. Ở trong báo cáo này, chúng tôi sẽ sử dụng I2S Mode để đọc dữ liệu, đây là cách để đọc dễ nhất. Ở trong Mode này, khi xung clock LRCK (ADCLRC) chuyển trạng thái, thì clock đầu tiên của BLCK chưa có dữ liệu xuất hiện mà sẽ xuất hiện ở clock thứ hai sau khi LRCK chuyển trạng thái. Lúc này kết hợp với CLOCK ngược của BLCK( chế độ invert BLCK). Thì khi đó, khi mà tín hiệu INVERT BLCK có cạnh xuống, thì ta sẽ đọc dữ liệu của của chân ADCDAT. Cách đọc này là an toàn là bởi vì khi mà INVERT BLCK cạnh xuống, thì nó nằm ngay giữa thời gian xuất hiện của Data, từ đó ta sẽ đọc chính xác và không nhầm lẫn.

**Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, biểu đồ

Mô tả được tạo tự động**

Hình 2-7: Sử dụng I2S Mode với INVERT của clock BLCK

Quan sát hình trên và những gì tôi đã viết ở phía trên ảnh bạn sẽ hiểu tại sao tôi sử dụng đảo của BLCK và hiệu quả của nó.

Tiếp theo ta sẽ quan tâm đến thanh ghi cấu hình Mode

Ảnh có chứa văn bản, ảnh chụp màn hình, Song song, số

Mô tả được tạo tự động

Hình 2-8: Thanh ghi cấu hình Mode

Như vậy ta sẽ cấu hình thanh ghi như sau:

Giá trị là: 0EC2=00001110 1100 0010

BIT7: 1 ==> INVERT CLOCK BCLK

BIT6: 1 => MASTER MODE

BIT5: 0 ==> DAC Left Righ Clock Swap -> Right channel DAC Data Right

BIT4: 0 ==> Right channel DAC data when DACLRC low (thiết lập dữ liệu kênh trái ở chu kỳ nào của LCRLK)

BIT3: 0==>16bit format

BIT2: 0

BIT1: 1 => I2S Formart

BIT0: 0

Hoặc nếu không dùng INVERT CLOCK, số lượng bit khác, chúng ta có thể điều chỉnh tại thanh ghi này. Ở phần sau tôi sẽ cung cấp địa chỉnh I2C của thanh ghi này để chúng ta viết dữ liệu vào đó.

***Cấu hình CLOCK***

Ảnh có chứa văn bản, số, trò chơi giải ô chữ, Song song

Mô tả được tạo tự động

Hình 2-9: Bảng cấu hình clock ở chế độ lấy mẫu bình thường

Chúng tôi sử dụng tần số cả hai kênh với tần số là 48kHz, do đó các bit của tôi sẽ có giá trị là BOSR=0, SR3=0, SR2=0, SR1=0, SR0=0. Và ta sẽ chuyển dữ liệu này vào địa chỉ I2C chứa nó.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Hình 2-10: Thanh ghi cài đặc xung clock

***Cấu hình dữ liệu đầu vào***

Chúng ta có nhiều đầu vào là LineIn, Audio và giữa chúng là có nhiều switch để chọn lượng, và dẫn tới cũng có nhiều mode hoạt động, và đôi khi ta có thể switch sang bộ lọc có sẵn trên chip.

Ảnh có chứa biểu đồ, văn bản, Kế hoạch, Bản vẽ kỹ thuật

Mô tả được tạo tự động

Hình 2-11: Sơ đồ khối nguyên lý hoạt động của dữ liệu đầu vào

Quan sát thấy, chúng ta sẽ cấu hình các khỏa SIDETONE, BYPASS, DACSEL để cấu hình tín hiệu đầu vào của chúng ta sẽ đi từ đâu và đi qua những gì.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Hình 2-12: Thanh ghi cấu hình tín hiệu đầu vào

***Cài đặc volume và cách hoạt động trên kênh trái và phải***

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động

**Bảng tổng hợp các thanh ghi để cấu hình toàn bộ chip WM8731**

Ảnh có chứa văn bản, số, Phông chữ, Song song

Mô tả được tạo tự động

Hình 2-13: Tổng hợp thanh ghi của WM8731

Do đó mà tôi đã cấu hình vào trong Code file CLOCK500 của nhà sản xuất cung cấp là:

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình

Mô tả được tạo tự động

Mối giá trị của rom sẽ bao gồm 16bit, 7 bit địa chỉ, 1 bit do I2C tạo để báo dữ liệu sẽ xuất ra, và 8 bit là dữ liệu sẽ được nạp vào thanh ghi.

Một tài liệu tham khảo thêm, đây là sơ đồ khối driver được vẽ bởi Petter K¨allstr¨om, Mario Garrido trong Lab4 về CODEC, liên kết tại: [Lab4\_Audio.pdf (liu.se)](https://www.isy.liu.se/edu/kurs/TSIU03/Documents2021/Lab4_Audio.pdf)

Ảnh có chứa văn bản, biểu đồ, Kế hoạch, sơ đồ

Mô tả được tạo tự động

Ảnh có chứa văn bản, biểu đồ, Song song, đen và trắng

Mô tả được tạo tự động

Hình 2-14: Các tín hiệu của chip WM8731

# **CHƯƠNG 3: THIẾT KẾ VÀ MÔ PHỎNG**

## 3.1. Thiết bộ lọc bằng matlab

Để tạo bộ lọc, ta sẽ thiết kế trên Matlab hoặc Python. Ở đây tôi sẽ sử dụng phần mềm Matlab để thiết kế. Filter Designer cung cấp một môi trường đồ họa tương tác để thiết kế các bộ lọc số IIR hoặc FIR dựa trên các tiêu chuẩn do người dùng xác định.

* Các loại bộ lọc có thể thiết kế: Thông thấp, thông cao, thông dải, chắn dải.
* Các phương pháp thiết kế bộ lọc FIR: Equiripple, Least squares, Window
* Các phương pháp thiết kế bộ lọc IIR: Butterworth, Chebyshev loại I, Chebyshev loại II, Elliptic

Ở đây chúng tôi đang thực hiện thiết kế bộ lọc thông thấp, bởi vì ứng dụng của chúng tôi là bộ âm thành dành riêng biệt cho nhu cầu nghe nhạc bass. Và do đó tần số cắt sẽ rơi vào khoảng 5kHz. Và chúng tôi đã thử nghiệm tại tần số cắt đó, thì âm thanh vẫn nghe được, và tín hiệu bass nghe ổn.

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Sơ đồ

Mô tả được tạo tự động

Hình 3-1: Kiểm tra tín hiệu trong vùng tần số vào sẽ nghe tốt

Chúng tôi có nhu cầu muốn lọc bỏ hoàn toàn những tần số khoảng từ 8kHz trở đi, bơi vì nhưng âm thanh này sẽ nghe sẽ bị chói. Do đó biên độ của bộ lọc trong vùng băng thông này khoảng -80 đến -60dB.

Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Biểu tượng máy tính

Mô tả được tạo tự động

Hình 3-2: Thiết kế bộ lọc trên bằng công cụ Filter Design của Matlab

Các giá trị thu được của vector đáp ứng xung sẽ được sử dụng để thực hiện bộ lọc số lên trên kit DSP. Do chương trình khảo sát bộ lọc số lên trên kit DSP sử dụng chế độ 16 bit có dấu trong khi chương trình thiết kế bộ lọc bằng MATLAB chuẩn hóa các hệ số đáp ứng xung trong khoảng [-1;1] nên các hệ số đáp ứng xung này cần nhân với 215 và làm tròn về số nguyên trước khi đưa vào thực hiện bộ lọc số lên trên kit DSP như sau:

Ảnh có chứa ảnh chụp màn hình, Phần mềm đa phương tiện, đồ điện tử, phần mềm

Mô tả được tạo tự động

Hình 3-3: Lấy hệ số và chuyển từ hệ số chuẩn hóa sang 16 bit có dấu

Sau khi có các thông số bộ lọc, tôi tiến hành ta viết bộ lọc bằng system verilog.

Ta phân tích, bộ lọc sẽ lấy từ mẫu vào bộ FIR, các mẫu sẽ vào tuần tự theo chu kỳ lấy mẫu là của xung LRCLK. Sau mỗi chu kỳ, các mẫu sẽ được dịch chuyển qua một delay, và sẽ có một biến lưu lại giá trị kết quả nhân của các hệ số với mẫu. Và sau khi có xung lấy mẫu là LRCLK, chúng ta sẽ có một mẫu đi vào và một mẫu đi ra.

Khi thiết kế, theo như khuếch khích các tài liệu đọc, thì việc cấu hình chip ***WM8731*** hoạt động ở chế độ 16 bit và tần số lấy mẫu là 48ksps sẽ ổn định nhất.

Do đó bản thiết kế bộ FIR sẽ có:

Input: tín hiệu 16 bit, xung clock LRCLK

Output: tín hiệu 16 bit.

Ngoài ra bên trong sẽ cần bộ FIR sẽ cần có các thanh ghi nhớ đóng vai trò như Delay, để lưu trữ các mẫu và kết quả nhân.

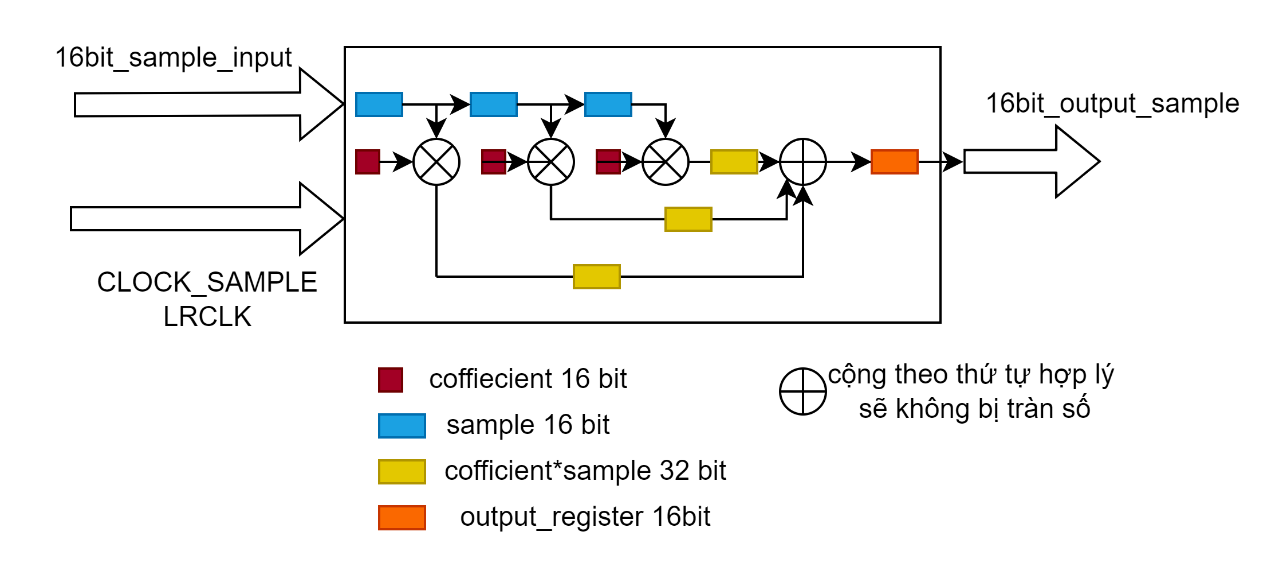
Vậy bên trong khối FIR tôi sẽ cần:

+ N 16 bit thanh ghi để lưu hệ số của bộ lọc (N-tap): coffecient

+ N 16 bit thanh ghi để lưu mẫu (sample) từ x(n) đến x(n-N):

+ N 32 bit để lưu kết quả nhân của từng mẫu với hệ số (Bời vì hệ số là 16 bit nhân với mẫu có kích thước là 16 bit nên thanh ghi này có kích thước tối thiểu là 32 bit).

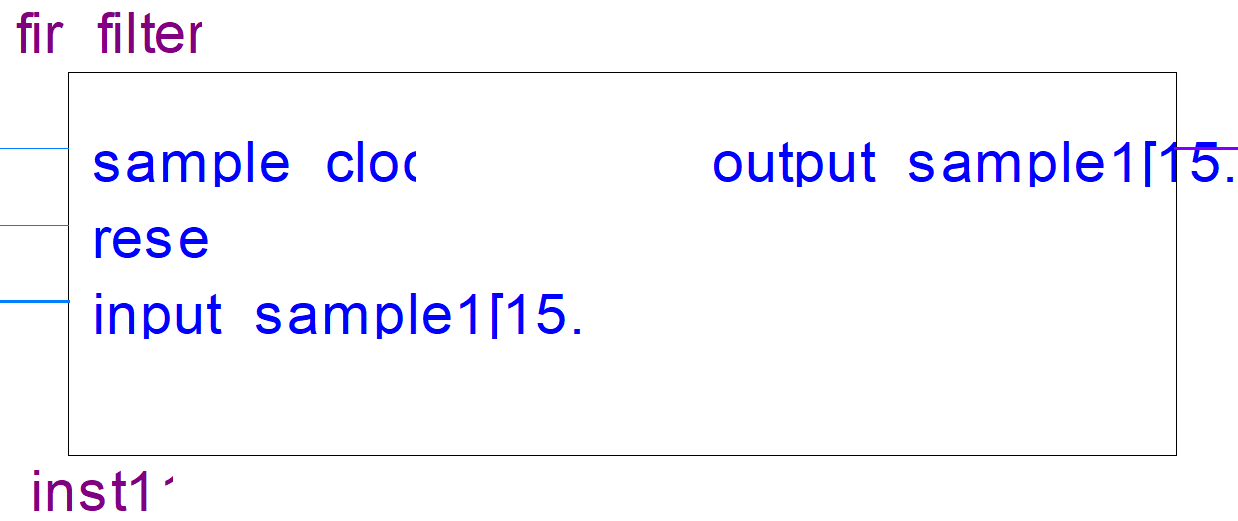
+ 1 thanh ghi 16 bit để lưu kết quả cộng cuối cùng với nhau (Đây là một câu hỏi tại sao 16 bit, là bởi vì ban đầu hệ số của chúng ta cofficient đã được nhân với 215 để cho dễ tính toán với số nguyên, do đó khi tín toán xong ta phải chia kết quả lại cho 215 để chuyển về 16 bit và xuất ra ngõ ra 16 bit. Ngoài ra ta sẽ cộng từ kết quả 32 bit với nhau, nhưng làm sao để chứa với thanh ghi 16 bit, đơn giản chúng ta sẽ dùng thêm một thanh ghi đệm, thanh ghi đó sẽ giúp chia kết quả nhân của từng cặp mẫu với hệ số với 215 trước, sau đó mới dồn vào thanh ghi kết quả).



Hình 3-4: Sơ đồ khối bộ lọc FIR cơ bản

Như tôi đã trình bày, phép cộng cuối cùng sẽ cộng theo thuật toán do đó tôi không thể trình bày cụ thể khối đó như thế nào.

Và ngoài ra chúng ta cũng có thể thêm chân Reset vào với chức năng reset tất cả giá trị sample trong mạch về không.



Hình 3-5: Khối FIR sau khi thiết kế.

## 3.2. Kiểm tra bộ lọc và tạo các file âm thanh bằng Python

Để hiểu rõ hơn các file âm thanh của chúng ta ở miền tần số vào, và biết chọn file âm thanh mẫu cho bộ lọc của chúng ta. Nhờ vào python, chúng ta sẽ biết phổ của tín hiệu trong miền tần số, từ đó biết được chúng ta sẽ biết chọn các thông bộ lọc như thế nào. Đồng thời hình thành được các tập test phù hợp cho bộ lọc chúng ta.

Sau đây là code để phân tích một file .wave

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

from scipy.fft import fft,fftfreq

sampling\_rate, audio\_data = wavfile.read("/content/drive/MyDrive/DSp/Asample1.wav")

# print(audio\_data[0:100])

audio\_left = audio\_data[:, 0]  # Kênh trái

# print(audio\_left[0:10])

audio\_right = audio\_data[:, 1]  # Kênh phải

# print(audio\_right[0:10])

# Tính toán âm lượng trung bình của hai kênh âm thanh

audio\_average = (audio\_left + audio\_right) / 2

# print(audio\_right[0:10])

# Chuyển đổi tín hiệu âm thanh thành dạng thời gian

time = np.arange(0, len(audio\_data)) / sampling\_rate

# Fourier Transform

audio\_spectrum\_average = fft(audio\_average)

audio\_spectrum\_left = fft(audio\_left)

audio\_spectrum\_right = fft(audio\_right)

noisy\_audio\_average = np.fft.ifft(audio\_spectrum\_average).real.astype(np.int16)

noisy\_audio\_left = np.fft.ifft(audio\_spectrum\_left).real.astype(np.int16)

noisy\_audio\_right = np.fft.ifft(audio\_spectrum\_right).real.astype(np.int16)

# Savefilewave

wavfile.write('/content/drive/MyDrive/DSp/Asample1\_Average.wav', sampling\_rate, noisy\_audio\_average)

wavfile.write('/content/drive/MyDrive/DSp/Asample1\_Left.wav', sampling\_rate, noisy\_audio\_left)

wavfile.write('/content/drive/MyDrive/DSp/Asample1\_Right.wav', sampling\_rate, noisy\_audio\_right)

# print(audio\_spectrum\_average[0:100])

# print(np.size(audio\_spectrum\_left))

N=np.size(audio\_spectrum\_left);

T=1/sampling\_rate

# Drawspectrum.

plt.figure(figsize=(10, 4))

xf = fftfreq(N, T)[:N//2]

plt.plot(xf, 2.0/N \* np.abs(audio\_spectrum\_left[0:N//2]))

plt.title('Spectrum  of file wave')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

Thông thường dữ liệu âm thanh của chúng ta sẽ có hai kênh là kênh trái và kênh phải. Tuy nhiên tôi chỉ sử dụng một kênh để cho đơn giản là kênh trái. Sau khi tiến hành tách dữ liệu của từng kênh từ dữ liệu gốc. Ta sẽ sử dụng Fourier Transform để chuyển từ miền thời gian về miền tần số. Như hình dưới đây, là phổ của tín hiệu kênh trái. (Một thông tin thêm là các âm thanh trên thị trường thường được lấy mẫu ở tần số 44.1ksps, do đó Fs/2 sẽ là 22.05ksps).

Ảnh có chứa văn bản, hàng, Sơ đồ, ảnh chụp màn hình

Mô tả được tạo tự động

Hình 3-6: Phổ của tín hiệu ở kênh trái file âm thanh.

Để tạo nhiễu, chúng tôi thực hiện tạo bằng cách tạo một hàm tạo nhiễu, đây chỉ là một hàm tạo ra một số phức ngẫu nhiên với biên độ đã được chúng tôi hiệu chỉnh nhiều lần để phù hợp với bộ lọc. Nếu không căn chỉnh bộ lọc có thể tín hiệu không thể khôi phục được (SNR quá cao dẫn tới không thể khôi phục tín hiệu). Mặc dù có các hàm tạo nhiễu Gausian,… nhưng các hàm này nó tạo nhiễu trên tất cả tần số, nhưng tôi chỉ muốn tạo trên một số tần số nhất định. Do đó mà tôi tạo ra hàm dưới đây.

def generate\_complex\_number():

    real\_part = np.random.uniform(-5e6, 5e7)

    imag\_part = np.random.uniform(-5e7, 5e6)

    return complex(real\_part, imag\_part)

random\_complex\_number = generate\_complex\_number()

print("Random:", random\_complex\_number)

Sau khi có được hàm tạo nhiễu, tôi tiến hành cộng nó vào tại các tần số ngoài khoảng của bộ lọc của tôi. Ví dụ bộ lọc tôi lọc tín hiệu trong khoản từ 5kHz, vậy thì tôi sẽ thêm nhiễu vào các miền có tần số trên 5kHZ. Như đoạn code dưới đây, chúng tôi thêm nhiễu ở tần số 7kHz đến 21kHz.

audio\_spectrum\_left\_with\_noise = audio\_spectrum\_left

for i in range(86075,1715415):

  audio\_spectrum\_left\_with\_noise[2692163+i]+=generate\_complex\_number();

  audio\_spectrum\_left\_with\_noise[2692163-i]=audio\_spectrum\_left\_with\_noise[2692163+i];

plt.figure(figsize=(15, 6))

plt.plot(xf, 2.0/N \* np.abs(audio\_spectrum\_left\_with\_noise[0:N//2]))

plt.title('Spectrum of file wave after adding noise')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

noisy\_audio\_left = np.fft.ifft(audio\_spectrum\_left\_with\_noise).real.astype(np.int16)

wavfile.write('/content/drive/MyDrive/DSp/Asample1\_Left\_with\_noise.wav', sampling\_rate, noisy\_audio\_left)

Sau khi thực hiện thêm nhiễu vào file wave đó, bạn có thể thấy vùng biên độ từ 7kHz đến 21kHz đã tăng lên rất nhiều, do đó tín hiệu của chúng ta lúc này sẽ bị rè rè do nhiễu. Có thể xem phổ tần số sau khi thêm nhiễu dưới đây:

Ảnh có chứa ảnh chụp màn hình, văn bản, hàng, Sơ đồ

Mô tả được tạo tự động

Hình 3-7: Phổ tín hiệu kênh trái sau khi thêm nhiễu

Từ phổ của tín hiệu sau khi nhiễu, ta tiến hành biến đổi ifft, tức là Fourier Transform Inversre, biến đổi ngược lại miền thời gian, là chúng ta dược tín hiệu có nhiễu. Và lưu file nhiễu đó vào Google Drive và làm âm thanh để truyền vào KIT DE2.

## 3.3. Mô phỏng bộ lọc trên ModelSim

***Tạo tín hiệu nhiễu giả để tạo testbench cho ModelSsim***

Để tạo tín hiệu giả hình sine đi vào bộ lọc, ta phải tạo bằng python từ đó lấy các mẫu của nó để tạo ra testbench trên ModelSim. Ta dùng python để tạo mẫu test cho testbench để mạch của ta có thể lấy mẫu với tần số của tín hiệu ban đầu bao gồm 2kHz và 20kHz tạo thành với biên độ của tần số 2kHZ là 20000 và 20kHz là 10000.

Ảnh có chứa văn bản, Sơ đồ, hàng, biểu đồ

Mô tả được tạo tự động

import numpy as np

import matplotlib.pyplot as plt

sampling\_rate = 800  # Sample Frequency

duration = 2  # Time (s)

frequency\_1 = 2  #  2KHz

frequency\_2 = 5  #  5kHz

frequency\_3 = 10 #  10kHz

frequency\_4 = 20 #  20kHz

frequency\_5 = 80 # Frequency of sine5

amplitude\_1 = 18000# Amplitude of sine3

amplitude\_2 = 0 # Amplitude of sine3

amplitude\_3 = 0 # Amplitude of sine3

amplitude\_4 = 9000 # Amplitude of sine3

amplitude\_5 = 5000 # Amplitude of sine3

t = np.linspace(0, duration, int(sampling\_rate \* duration), endpoint=False)

y = amplitude\_1 \* np.sin(2 \* np.pi \* frequency\_1 \* t) + amplitude\_2 \* np.sin(2 \* np.pi \* frequency\_2 \* t) + amplitude\_3 \* np.sin(2 \* np.pi \* frequency\_3 \* t)+ amplitude\_4 \* np.sin(2 \* np.pi \* frequency\_4 \* t) + amplitude\_5 \* np.sin(2 \* np.pi \* frequency\_5 \* t)

plt.plot(t, y)

plt.title('Signal 10000\*2kHz + 9000\*20kHz + 5000\*80kHz')

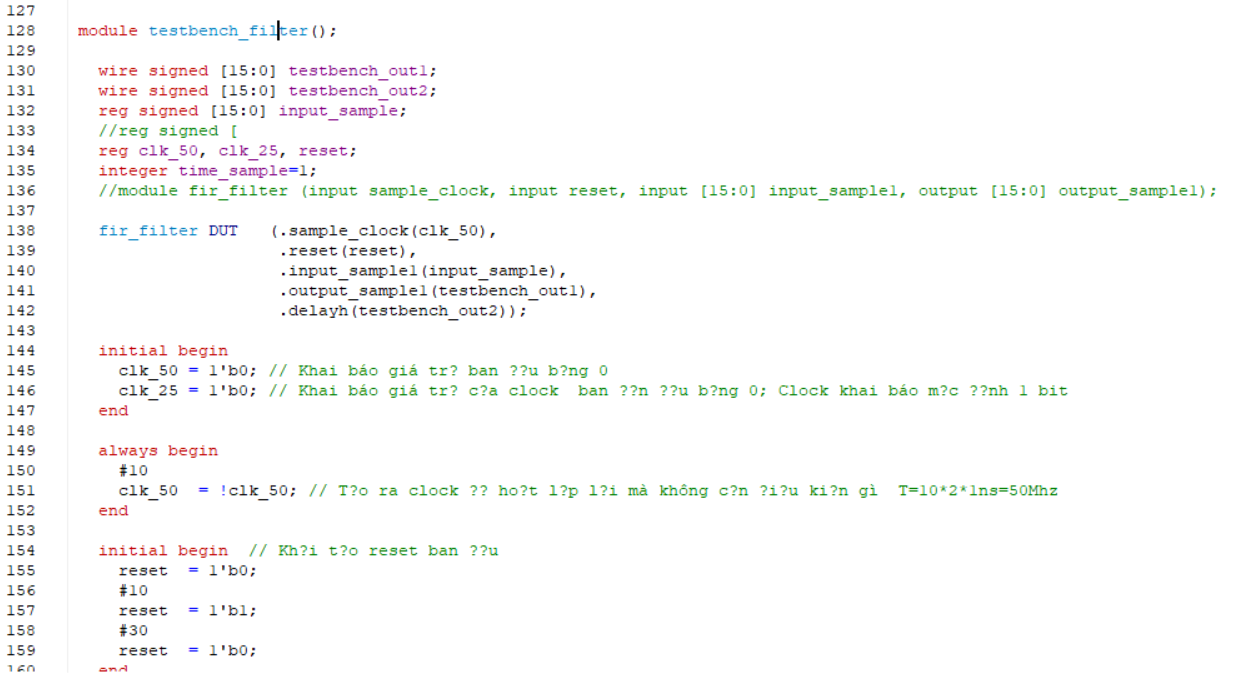
plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

Sau khi thực hiện code trên, ta tiến hành lẫy các mẫu của để tạo testbecnch cho modelsim.



Đây là sample lấy từ tín hiệu python đã tạo code python đã tạo tín hiệu, giờ ta lấy các sample để chạy trên ModelSim



Hình 3-8: Tạo testbench mô phỏng cho bộ lọc

Để dễ hiểu, trong testbench tôi sẽ tạo ra 1 clock có tần số bằng fs=48ksps của KIT Altera để gắn vào sample\_clock. Input sample sẽ lấy từ tín hiệu tôi tạo từ python và tôi sẽ cho tín hiệu này chạy với tốc độ rất nhanh, và tạo ra dạng sóng sao cho nó giống tín hiệu thực tế, để cho firfilter lấy mẫu.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Sau khi thực hiện testbench, tôi được kết quả như sau, ban đầu tín hiệu tôi cho bao gồm có 2kHz và 20kHz. Sau khi qua bộ lọc, tín hiệu 20kHz đã bị lọc mất đi, chứng tỏ bộ lọc đang hoạt động tốt.

Ảnh có chứa ảnh chụp màn hình, màn hình, văn bản, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 3-9: Tín hiệu sau khi đi qua bộ lọc FIR

***Tạo một tín hiệu đầu vào khác***

Tiếp theo để kiểm chứng mạch tốt hơn tôi sẽ thêm các thành phần tần số khác để làm nhiễu vào tần số 2kHZ. Ta cũng dùng python để tạo mẫu test cho testbench để mạch của ta có thể lấy mẫu với t ần số của tín hiệu ban đầu bao gồm 2kHz và 20kHz tạo thành với biên độ của tần số 2kHZ là 20000 và 20kHz là 10000 và 80kHz với biên độ là 5000.

Ảnh có chứa văn bản, Sơ đồ, hàng, biểu đồ

Mô tả được tạo tự động

Ta thấy tín hiệu này bị nhiễu rất nặng, hầu như đã bị biến dạng, tuy nhiên hình dạng của 2kH vẫn còn do nó có biên độ lớn hơn so với các tín hiệu nhiễu.

Ảnh có chứa ảnh chụp màn hình, màn hình, văn bản, phần mềm

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

Hình 3-10: Tín hiệu sau khi đi qua bộ lọc FIR

## 3.4. Phân tích về các độ trễ của mạch

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

latency

Phân tích về các độ trễ của mạch, như hình trên, tôi sử dụng theo kỹ thuật FIR thông thường không dùng kỹ thuật FIR transform. Do đó mà Latency của tôi chỉnh bằng số lượng Delay trong mạch của tôi. Hay chính xác nó bằng số lượng tap của tôi.

Hay Latency bằng thời gian mẫu đầu tiên vào, cho đến lúc mẫu đầu tiên ra. Thì nó đã phải đi qua N Delay. Với N=65 thì latency của chúng tôi sẽ là 65\*Tsample = 65\*chu kỳ clock của LRCLK. Mà LRCLK chạy với chu kỳ lấy mẫu là 48ksps. Do đó Latency=65\*1/48000=1.354ms.

Điều kiện Tcrital của bộ FIR phải bé hơn thời gian chu kỳ clock xuất tín hiệu, nhưng tần số suất tín hiệu là ở clock LRCLK, cứ mỗi lần có xung LRCLK sẽ xuất một sample ra DAC\_DATA và nó có tần số bằng tần số lấy mẫu là 48ksps. Do đó mà Tcritcal phải bé hơn 1/48000ksps = 20.8us. Nếu không, bộ fir chưa kịp xử lý tín hiệu xong mà có xung báo muốn xuất tín hiệu thì chúng ta sẽ không có tín hiệu để xuất ra.

Hay ta có được Tcirtal=(N-1)\*TA+TM=64TA+TM < 20.8us.

## 3.4. Đánh giá hiệu quả của Filter

Bạn hãy quan sát hình sau đây, đây là tín hiệu ***trong miền thời gian*** ban đầu của tôi.

Ảnh có chứa hàng, Sơ đồ, biểu đồ, văn bản

Mô tả được tạo tự động

**Tín hiệu sau khi thêm nhiễu:**

Ảnh có chứa Sơ đồ, văn bản, hàng, ảnh chụp màn hình

Mô tả được tạo tự động

**Tín hiệu sau khi được lọc nhiễu bởi bộ lọc FIR (số tap hơn 500 tap)**

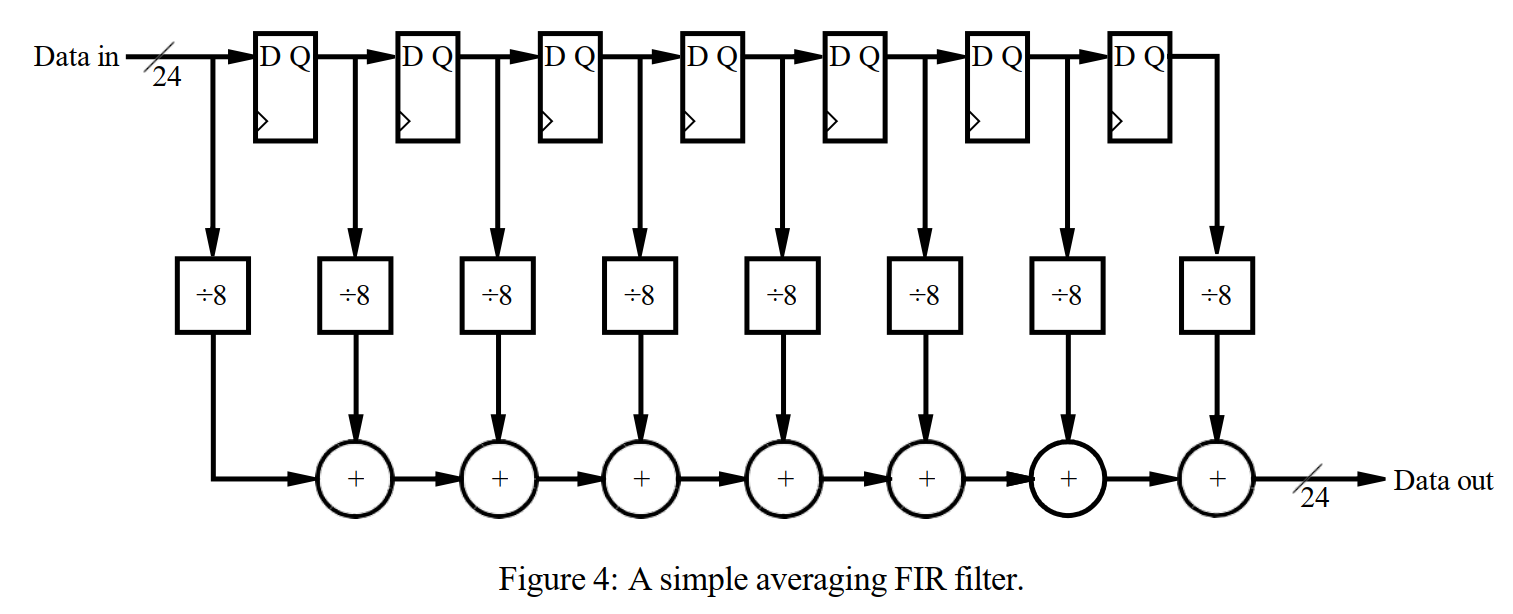
Ảnh có chứa văn bản, hàng, Sơ đồ, biểu đồ

Mô tả được tạo tự động

**Tín hiệu vẫn còn nhiễu với biên độ lớn**

Mặc dù số lượng tap rất lớn, tuy nhiên tín hiệu sau cùng, vẫn bị ảnh hưởng bởi nhiễu, và âm thanh nghe rất rè. Lý do trên miền tần số, nhiễu rất nhiều tần số, và ở tần số cao có biên độ khá lơn, dẫn đến tín hiệu ban đầu bị nhiều gai. Tuy nhiên hình dạng tín hiệu ban đầu vẫn nhìn thấy được. Các tín hiệu gai (giống như nhiễu hạt tiêu) này nó làm cho bộ lọc cũng bị nhiễu theo, và không thể nào lọc được hoàn toàn.

Do đó để loại bỏ bớt gai tôi đề xuất sử dụng bộ lọc trung bình, hoặc bộ lọc trung bị. Nhưng để đơn giản, tôi sử dụng bộ lọc trung bình trước khi bộ lọc FIR.



Hình 3-11: Bộ lọc FIR trung bình

Bộ lọc FIR tôi sẽ thiết kế giống như trên, tuy nhiên số tap tôi sử dụng là 16 tap, do đó giá trị /8 của tôi sẽ được điều chỉnh là “16”.

Chúng ta có thể xây dựng bộ lọc average dựa trên bộ lọc FIR Low Pass Filter bằng cách điều chỉnh tất cả hệ số của chúng lại bằng 1, hoặc 1/8 nếu thiết ở dạng số thực.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, máy tính

Mô tả được tạo tự động

Hình 3-12: Thực hiện test bộ lọc trung bình bằng python

***Sau đây là kết quả khi ta sử dụng thêm bộ lọc trung bình (Lọc qua bộ FIR medium rồi ra qua bộ FIR Lowpass)***

**Tín hiệu ban đầu:**

Ảnh có chứa hàng, Sơ đồ, biểu đồ, văn bản

Mô tả được tạo tự động

**Tín hiệu sau khi thêm nhiễu:**

Ảnh có chứa Sơ đồ, văn bản, hàng, ảnh chụp màn hình

Mô tả được tạo tự động

**Tín hiệu sau qua medium**

Ảnh có chứa hàng, văn bản, Sơ đồ, biểu đồ

Mô tả được tạo tự động**Tín hiệu sau bộ lọc FIR**

Ảnh có chứa hàng, văn bản, Sơ đồ, biểu đồ

Mô tả được tạo tự động

Tín hiệu nhiễu với biên độ thấp

Ta thấy rằng, chỉ cần thêm một bộ lọc trung bình có 16 tap, việc lọc tín hiệu của chúng ta trở nên hiệu quả hơn rất nhiều. Một điểm tôi phát hiện thêm rằng: nếu có bộ lọc trung bình thì bộ FIR Low pass filter chỉ cần khoảng 60 tap là có thể lọc được tốt. Còn nếu không có bộ lọc trung bình thì cho dù có 1000 tap thì kết quả cũng vẫn có nhiễu và âm thanh bị rè.

## 3.5. Thiết kế bộ Serial to Parallel

Ta cần thiết kệ một bộ Serial to Parallel bởi vì tín hiệu dữ liệu xử lý của chúng ta là từng mẫu, không phải từng bit. Mỗi lần clock, ta sẽ đưa một mẫu với độ rộng 16 bit vào. Bởi vì, nếu không có bộ SerialtoParalllel, thì bộ FIR phải xử lý serial từng bit của mẫu, thì bộ fir sẽ không hoạt động được. Bởi vì bộ fir hoạt động theo từ mẫu vào, và mỗi mẫu có kích thước là 16bit. Do đó, ta cần có một ngõ vào parallel 16bit để chuyển của mẫu từ Serial sang Parallel 16 bit vào bộ lọc FIR cùng một lúc.

**Ngõ ra của bộ Serial to Parallel là 16 bit[0-15]:**

**A diagram of a circuit

Description automatically generated**

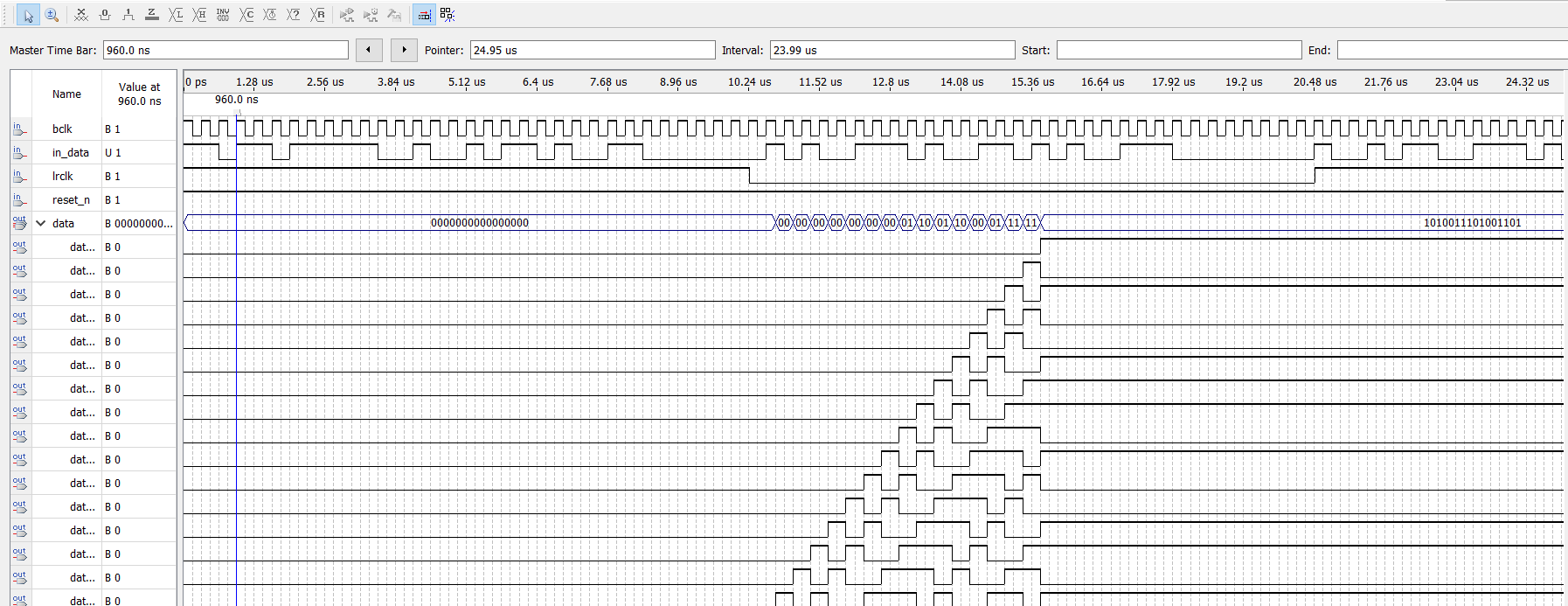
Ngõ vào ***in\_data:*** sẽ được gắn với PIN AUD\_ADCDAT (PIN\_A4): Audio CODEC DAC Data

Ngõ vào ***bclk:*** sẽ được gắn với AUD\_BCLK (PIN\_B4): Audio CODEC Bit-Stream Clock

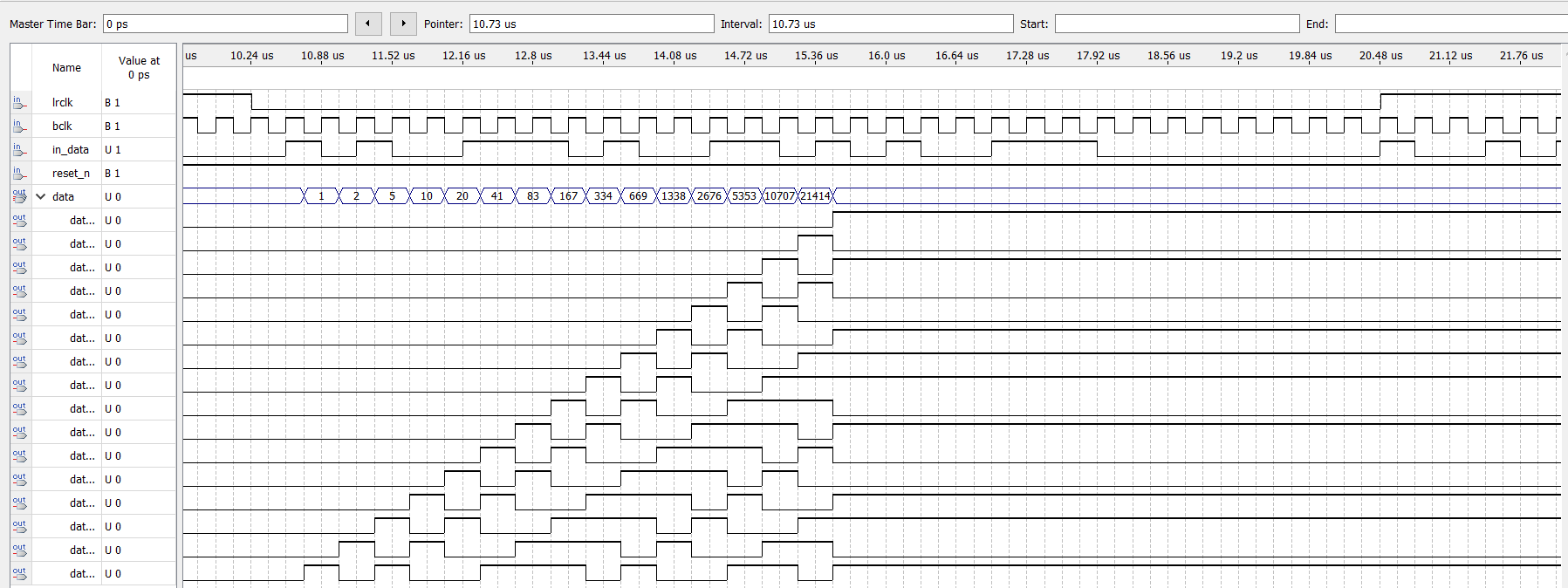
Ngõ vào ***Irclk***: sẽ được gắn với AUD\_ADCLRCK (PIN\_C6) Audio CODEC DAC LR Clock

**Sau đây là waveform bộ Serial to Parallel:**

Vì mình sử dụng I2S Mode, nên mỗi lần LRCLK có cạnh xuống thì sau 1 chu kì BCLK, bắt đầu có bit dịch đi vào



Mỗi lần di chuyển một bit vào, nên nó hình như vậy, chú ý chỗ lrck xuống 0 thì bắt đầu có bit dịch vào,



## 3.6. Thiết kế bộ Parallel to Serial.

Sau khi tín hiệu đã qua bộ lọc FIR, tín hiệu được lọc cần đưa về dạng 16bit serial ban đầu để xuất ra speaker.

**A diagram of a circuit

Description automatically generated**

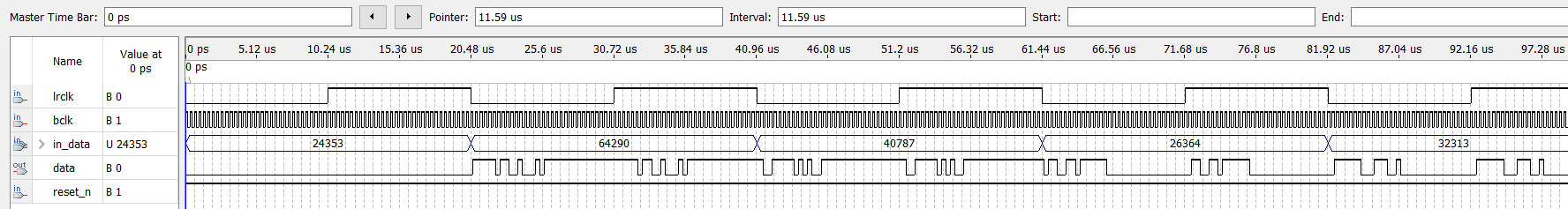
In\_data: là dữ liệu out sau khi mà đi qua bộ lọc FIR

Irclk cũng sẽ được gắn với AUD\_ADCLRCK (PIN\_C6), (Audio CODEC DAC LR Clock)

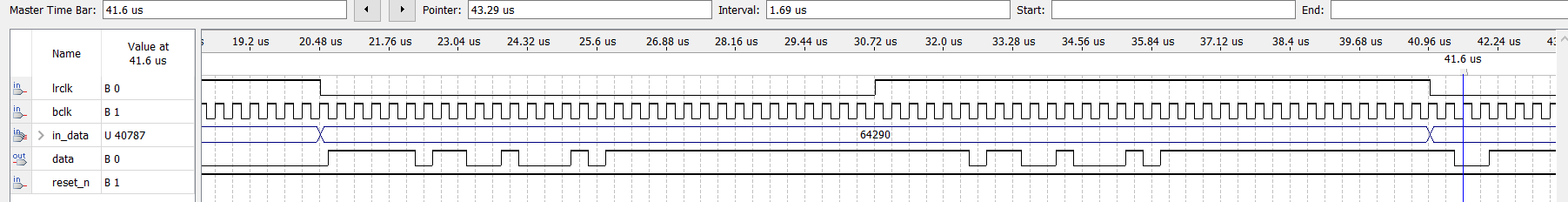
Bclk cũng sẽ được gắn với: sẽ được gắn với AUD\_BCLK (PIN\_B4), (Audio CODEC Bit-Stream Clock).

***Sau đây là waveform bộ Parallel to Serial:***

Unsigned integer 64290 ⬄ 1111101100100010 binary



Kiểm tra xem ngõ ra nó có bằng 1111101100100010 khi ngõ vào là 64290 không



Ta sẽ kiểm tra dữ liệu đầu vào 16bit có dịch từ từ ra ngõ ra data không, và tôi chọn giá trị 64290 và kiểm tra đầu ra thì tất cả các bit của nó đã được dịch ra đúng với CLOCK BCLK và LRCK

**3.4. Clock 500**

Đối với CLOCK\_500, khối này có chức năng tạo ra các xung CLOCK cho mạch, và ngoài ra nó còn tạo ra dữ liệu để truyền qua khối I2C để giao tiếp với chip WM8731, từ đó cấu hình hoạt động cho chip. Sau đây là các chân tín hiệu của CLOCK500

**Inputs**

input CLOCK: Tín hiệu clock chính của hệ thống.

input END\_TR: Tín hiệu kết thúc truyền dữ liệu.

input RESET: Tín hiệu reset.

input KEYON: Tín hiệu bật/tắt chính của hệ thống/

**Outputs**

output CLOCK\_500: Tín hiệu clock 500Hz.

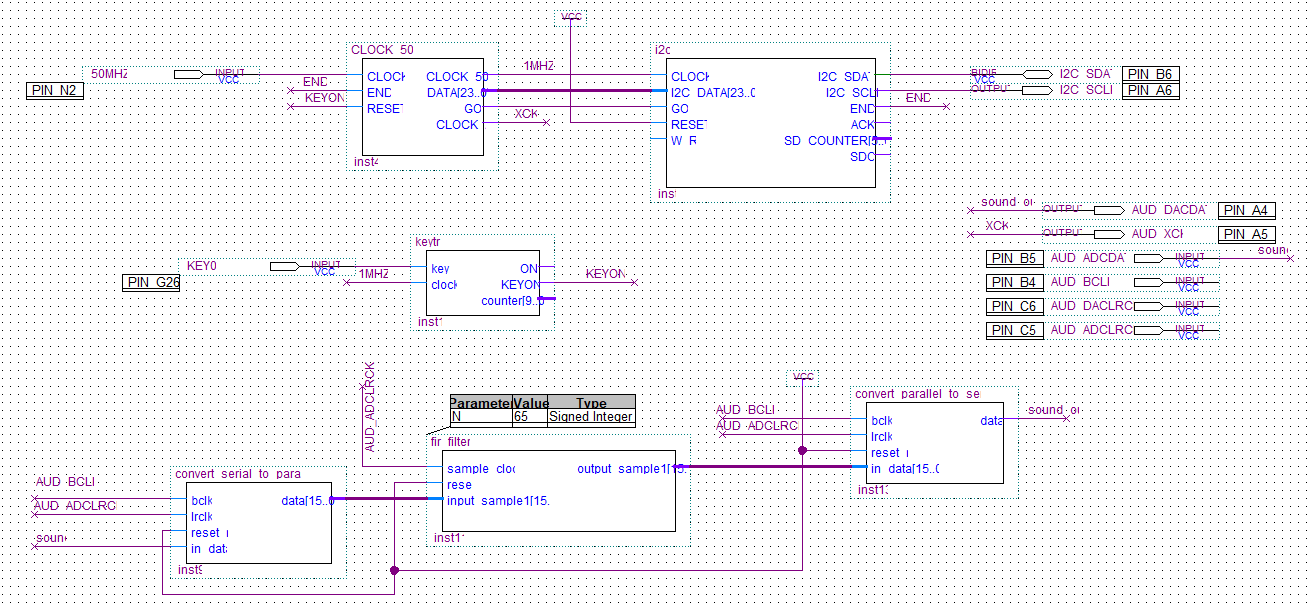
output [23:0] DATA: Dữ liệu đầu ra.

output GO: Tín hiệu bắt đầu truyền dữ liệu.

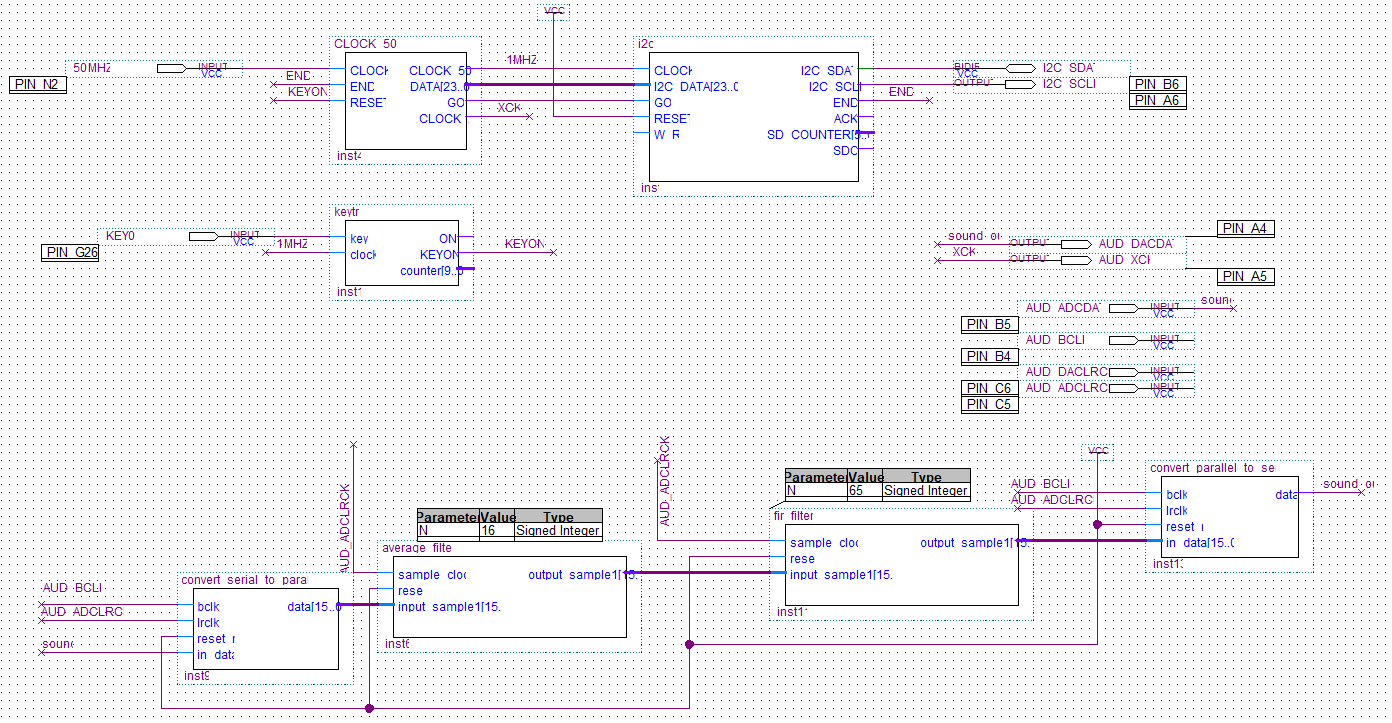
output CLOCK\_2: Tín hiệu clock 2Hz.

# **CHƯƠNG 4: KẾT QUẢ**

## 4.1. Sơ đồ kết nối của mạch



Hình 4-1: Sơ đồ nguyên lý chỉ có bộ lọc FIR Lowpass filer

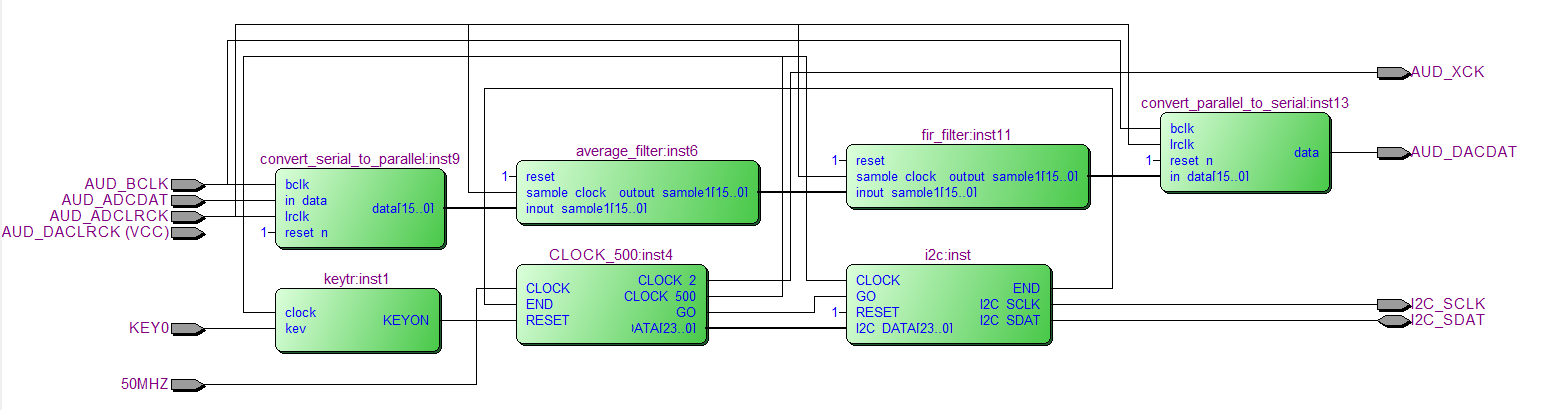


Hình 4-2: Sơ đồ nguyên lý bộ lọc Trung bình kết nối với bộ Lowpass filter

Như hình trên, ta thấy được bộ lọc average được xây dựng dựa trên bộ lọc Lowpass filter như đổi hệ số lại, do đó kiến trúc ngõ vào ngõ ra của nó cũng giống với bộ FIR Lowpass Filter



Hình 4-3: RTL View chỉ có bộ lọc FIR Lowpass filer



Hình 4-4: RTL View của bộ lọc Trung bình kết nối với bộ Lowpass filter

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, số

Mô tả được tạo tự động

Hình 3-13: Kết quả sau khi compile project

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Hình 3-14: Tiêu tốn tài nguyên của project

Ta thấy, project chiếm khoảng 5676 cổng logic chiếm gần 17% mà chip FPGA cung cấp. Đây là một số lượng lớn, chỉ một chức năng nhỏ mà tốn quá nhiều tài nguyên. Do đó chúng ta cần tối ưu lại bằng cách thay đổi thuật toán đọc.

Ví dụ thay vì sử dụng FIR thường, thì chúng ta sẽ sử dụng FIR Transform để giảm Tcitical mà số lượng cổng logic không đổi, do đó ta sẽ đỡ phải chèn thêm delay vào để pipeline mạch.

Loại bỏ bớt các thanh ghi tạm ở trong code, thay thế bằng các thuật toán tối ưu hơn.

## 4.2. Kết quả đổ KIT.

Ảnh có chứa đồ điện tử, Kỹ thuật điện, Dây điện, dây cáp

Mô tả được tạo tự động

Hình 3-15: Kết quả đổ KIT

Mặc dù mạch đã hoạt động, tuy nhiên vẫn tín hiệu không được lọc hiệu quả, dẫn đến âm thanh còn rè. Cách cải thiện là thiết kế lại bộ lọc, và đồng thời dùng các dụng cụ đo để kiểm tra các xung của tín hiệu có đang hoạt động đúng hay không.

# **TÀI LIỆU THAM KHẢO**

[1] https://github.com/MostafaOkasha/ECHO-and-FIR-FILTER-implementation-on-Verilo g-HDL

[2] Petter K¨allstr¨om, Mario Garrido trong Lab4 về CODEC, liên kết tại: [Lab4\_Audio.pdf (liu.se)](https://www.isy.liu.se/edu/kurs/TSIU03/Documents2021/Lab4_Audio.pdf)