ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG
BỘ MÔN MÁY TÍNH - HỆ THỐNG NHÚNG

TRẦN HỒNG SƠN
20200331

Đề tài:

THIẾT KẾ BỘ LỌC FIR VÀ THỰC HIỆN TRÊN FPGA

DESIGN FIR FILTER AND IMPLEMENTATION ON FPGA

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHUYÊN NGÀNH MÁY TÍNH - HỆ THỐNG NHÚNG

NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. HUỲNH HỮU THUẬN

TP. Hồ Chí Minh, tháng 12 năm 2024

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG
BỘ MÔN MÁY TÍNH - HỆ THỐNG NHÚNG

TRẦN HỒNG SƠN
20200331

Đề tài:

THIẾT KẾ BỘ LỌC FIR VÀ THỰC HIỆN TRÊN FPGA

DESIGN FIR FILTER AND IMPLEMENTATION ON FPGA

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHUYÊN NGÀNH MÁY TÍNH - HỆ THỐNG NHÚNG

NGƯỜI HƯỚNG DẪN KHOA HỌC

TS. HUỲNH HỮU THUẬN

TP. Hồ Chí Minh, tháng 12 năm 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin đặc biệt gửi lời cảm ơn tới thầy Huỳnh Hữu Thuận – người đã trực tiếp hướng dẫn, cung cấp các nguồn tài liệu tham khảo, chỉ dẫn cho quá trình thực hiện, các anh chị nghiên cứu sinh, các bạn trong phòng thí nghiệm CESLAB đã hết sức tạo điều kiện cho em có các hướng dẫn, các chỉ dạy, tiếp cận các trang thiết bị trong phòng để thực hiện và hoàn thành khóa luận này.

Kế đến, em xin được gửi lời cảm ơn đến Ban giám hiệu, các thầy cô trong trường Đại học Khoa học Tự nhiên – Đại học Quốc gia Thành phố Hồ Chí Minh nói chung, cũng như các thầy cô, giảng viên của khoa Điện tử - Viễn thông nói riêng đã tạo điều kiện cho em học tập trong ngôi trường này và hoàn thành khóa học.

Cuối cùng em xin cảm ơn tới gia đình, bạn bè đã hết sức động viên, tạo điều kiện giúp em chuyên tâm học tập và hoàn thành đề tài.

Trong quá trình thực hiện, em cảm thấy mình còn có nhiều thiếu sót, đề tài nghiên cứu còn nhiều hạn chế nên rất mong quý thầy cô và các anh chị góp ý để luận văn được hoàn thiện hơn.

Tôi xin chân thành cảm ơn!

Tp. Hồ Chí Minh, ngày tháng năm 20 .

Sinh viên

LỜI CAM KẾT

Tôi cam đoan khóa luận tốt nghiệp cử nhân ngành Điện tử - Viễn thông, với đề tài "Thực hiện và thiết kế lọc FIR trên FPGA" này là công trình khoa học/sản phẩm hoàn toàn do tôi thực hiện dưới sự hướng dẫn của TS. Huỳnh Hữu Thuận.

Những kết quả nghiên cứu/sản phẩm, đánh giá của khóa luận tốt nghiệp hoàn toàn trung thực và chính xác.

Sinh viên thực hiện

(Ký tên, ghi rõ họ tên)

Trần Hồng Sơn

TÓM TẮT ĐỒ ÁN

Bộ lọc đáp ứng xung hữu hạn (FIR) là một bộ lọc số thường được sử dụng trong các ứng dụng xử lý tín hiệu và hệ thống truyền thông như giảm nhiễu, tăng cường chất lượng hình ảnh, tổng hợp giọng nói và dạng sóng, v.v. Khi độ phức tạp của việc triển khai phụ thuộc vào số lượng các hệ số lọc và độ chính xác của tính toán với các số thực, việc xây dựng các bộ lọc này trong thời gian thực trên các cấu trúc của hệ thống FPGA với mức độ chính xác mong muốn trở thành một nhiệm vụ đầy thách thức do các hạn chế về tính toán phép nhân và số floating point. Vì vậy, việc thay đổi thuật toán của bộ lọc FIR mà không dùng phép nhân, số thực là cần thiết để nâng cao tốc độ và tài nguyên của hệ thống xử lý tín hiệu.

Trong luận văn này, bộ lọc số đã được thiết kế dạng thấp qua gồm 16 hệ số lọc đối xứng sử dụng cửa sổ Hamming. Mô hình đầu tiên được xây dựng trên MATLAB, C model và cuối cùng là trên FPGA. Với mục tiêu tối ưu hóa chức năng FPGA và giảm số phép toán nhân cộng tích lũy (MAC), nghiên cứu áp dụng kỹ thuật số học phân tán (DA) để cải thiện hiệu suất của bộ lọc, bao gồm hai phương pháp chính: Serial Distributed Arithmetic (SDA) và Parallel Distributed Arithmetic (PDA), cùng với việc sử dụng các bảng Look-Up Tables (LUTs).

Nghiên cứu đánh giá hiệu quả của thiết kế bộ lọc FIR, so sánh giữa phương pháp MAC truyền thống và hai phương pháp DA: SDA và PDA. So sánh hiệu suất nhằm chứng minh lợi ích của việc sử dụng DA và LUT trong việc giảm độ phức tạp phần cứng và tải tính toán, từ đó cải thiện hiệu suất tổng thể khi triển khai trên FPGA.
Nghiên cứu cung cấp cái nhìn sâu sắc về các khía cạnh thực tiễn của thiết kế bộ lọc FIR trên nền tảng FPGA và đóng góp vào việc phát triển các hệ thống xử lý tín hiệu số hiệu quả hơn. Kết quả nghiên cứu làm nổi bật những lợi ích của các kỹ thuật số học tiên tiến trong việc tối ưu hóa tài nguyên phần cứng và đạt được hiệu suất tốt hơn trong các ứng dụng thực tế.

ABSTRACT

A finite pulse response (FIR) filter is a digital filter commonly used in signal processing and communication system applications such as noise reduction, image quality enhancement, voice and waveform synthesis, etc. When implementation complexity depends on the number of filter coefficients and floating-point calculation accuracy, building these filters in real-time on FPGA systems with desired precision becomes challenging due to limitations in multiplication operations and floating-point numbers. Therefore, modifying the FIR filter algorithm without using multiplication and real numbers is necessary to improve the speed and resources of the signal processing system.

In this thesis, a low-pass digital filter was designed with 16 symmetric filter coefficients using the Hamming window. The model was first built in MATLAB, then as a C model, and finally on FPGA. To optimize FPGA functionality and reduce Multiply-Accumulate (MAC) operations, the research applies Distributed Arithmetic (DA) techniques to improve filter performance, including two main methods: Serial Distributed Arithmetic (SDA) and Parallel Distributed Arithmetic (PDA), along with the use of Look-Up Tables (LUTs).

The research evaluates the effectiveness of FIR filter design by comparing traditional MAC method with two DA approaches: SDA and PDA. Performance comparisons demonstrate the benefits of using DA and LUT in reducing hardware complexity and computational load, thereby improving overall performance when implemented on FPGA.

The study provides deep insights into practical aspects of FIR filter design on FPGA platforms and contributes to the development of more efficient digital signal processing systems. The research results highlight the advantages of advanced arithmetic techniques in optimizing hardware resources and achieving better performance in practical applications.

MỤC LỤC

DANH SÁCH HÌNH MINH HỌA

CHƯƠNG 1:

CHƯƠNG 2:

CHƯƠNG 3:

CHƯƠNG 4:

DANH SÁCH BẢNG SỐ LIỆU

DANH MỤC CÁC TỪ VIẾT TẮT

ALM	Adaptive Logic Module
ASIC	Application-Specific Integrated Circuit
CPLD	Complex Programmable Logic Device
DA	Distributed Arithmetic
DSP	Digital Signal Processing
FIR	Finite-Impulse Response
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IIR	Infinite-Impulse Response
MAC	Multiply/Accumulator
NaN	Not a Number
PDA	Parallel Distributed Arithmetic
PSC	Parallel to Serial Converter
SDA	Serial Distributed Arithmetic

CHƯƠNG 1: GIỚI THIỆU

Xử lý tín hiệu số (DSP) là việc xử lý tính toán trên các mẫu của tín hiệu, và xử lý tín hiệu số đã trở thành một môn học cơ sở cho nhiều ngành khoa học – kỹ thuật: điện tử, tự động hóa, viễn thông,…Việc xử lý tín hiệu số được kết hợp trong nhiều loại thiết bị kỹ thuật số (CD, DVD, card âm thanh, camera, IC,… cũng như trong nhiều loại hệ thống truyền thông số, xử lý thông tin). Việc xây dựng hệ thống xử lý tín hiệu số có nội dung khá rộng và dươc áp dụng trên nhiều lĩnh vực, đặc biệt là phân tích, thiết kế, lập trình và xây dựng phần cứng cho các ứng dụng và yêu cầu cụ thể.

Một bộ xử lý tín hiệu số (còn được gọi là hệ thống xử lý dữ liệu rời rạc thời gian) có ứng dụng phổ biến nhất là lọc số (digital filter). Lọc số là quá trình xử lý tín hiệu số nhằm cải thiện chất lượng hoặc trích xuất một vài đặc tính nào đó từ tín hiệu đầu vào. Việc lọc số được thực hiện bằng cách áp dụng các thuật toán toán học để biến dổi tín hiệu theo một hướng nhất định (loại bỏ nhiễu, làm mượt tín hiệu, phân tách tần số…). Xét từ phương trình hiệu số hay cấu trúc của lọc mà người ta phân chia lọc số thành 2 loại lớn là lọc đệ quy (hay quy hồi) (reursive) được thể hiện với hàm đáp ứng xung vô hạn (Infinite Impluse Response) và lọc phi đệ quy (phi quy hồi) (nonrecursive) được thể hiện bằng hàm đáp ứng xung hữu hạn (Finite Impulse Response). Ở đây sẽ chỉ nói tới và mô tả về loại lọc phi đệ quy (có hàm đáp ứng xung hữu hạn) hay chính là lọc FIR.

FPGA (Field Programable Gate Array – Mảng có thể lập trình được) là loại mạch tích hợp có thể lập trình, tái cấu trúc và thay đổi chức năng sau khi sản xuất. Nó là mạng lưới gồm các khối có thể lập trình được (CLBs – Configurable Logic Blocks) và các phần tử khác và có thể thể được lập trình để thực hiện nhiều chức năng số khác nhau trong đó có cả xử lý tín hiệu số, hay cả thực hiện lọc FIR. Về phân loại, FPGA được chia thành nhiều loại tùy theo cấu trúc.

1. Mảng đối xứng: Được tạo thành từ hàng và cột của những khối logic được kết nối với nhau, bao quanh bởi các khối đầu vào/đầu ra

2. Kiến trúc dựa trên hàng: Sắp xếp xen kẽ các khối logic và tài nguyên kết nối có thể lập trình được, với những khối đầu vào/đầu ra dọc theo các cạnh.
3. PLD phân cấp: Đây là thiết bị logic có thể lập trình. Những thiết bị này có bố cục phức tạp được tạo thành từ các khối logic kết nối với nhau. Các khối logic có chứa khối logic, các phần tử có chức năng tổ hợp và tuần tự.

1.1 Lý do chọn đề tài

Các công nghệ tiên phong hiện nay có thể kể đến như là ứng dụng truyền thông di động do sự bùng nổ mạng 5G, xử lý ảnh để xây dựng các mô hình sử dụng Trí tuệ nhân tạo (AI) hay học sâu (DL), xử lý hình ảnh trong các hệ thống thời gian thực (RTOS)… yêu cầu các bộ xử lý tín hiệu số (DSP) cần phải có tốc độ cao hơn và tiêu thụ điện năng thấp hơn để đáp ứng nhu cầu ngày càng cao về tính chính xác của dữ liệu. Trước đây, các ứng dụng DSP với hiệu suất cao thường được bắt gặp chủ yếu trong các thiết bị ASIC và các chip DSP chuyên dụng. Việc sử dụng các mạch FPGA tạo một giải pháp thay thế linh động cho các ứng dụng này nhờ sự đa dụng và có thể tái lập trình của FPGA, đặc biệt trong việc thử nghiệm và nghiên cứu trong các phòng thí nghiệm hoặc trong các trường đại học.

1.2 Phạm vi đề tài

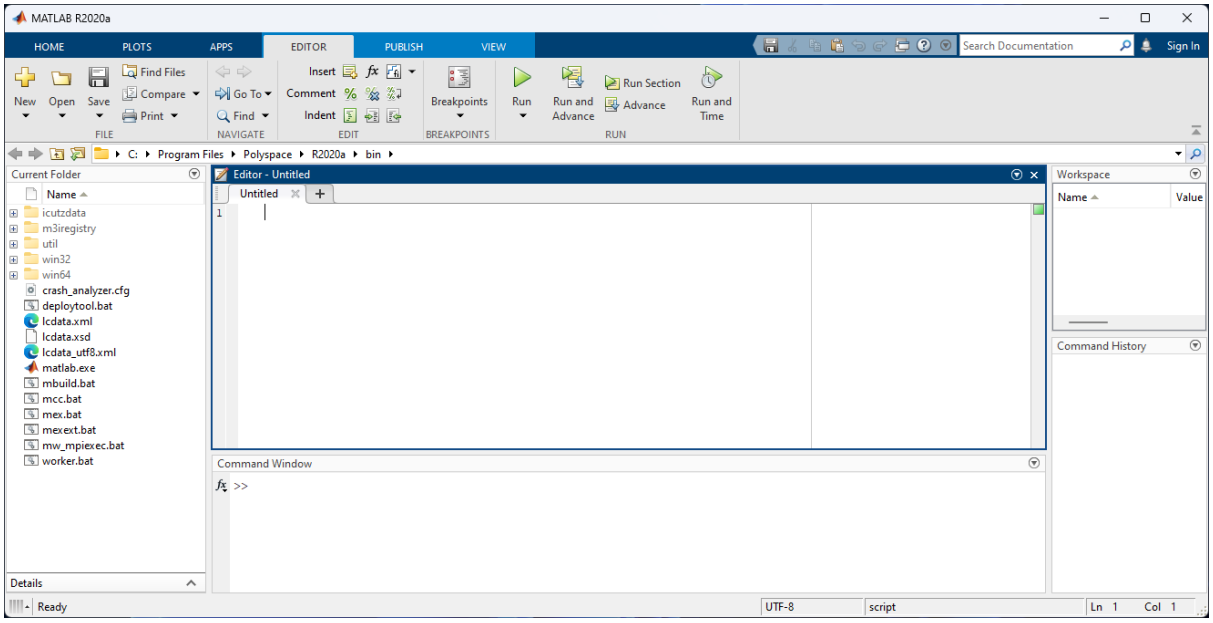
Điểm hạn chế của việc tính toán lọc FIR trên hệ thống FPGA việc xuất hiện tính toán số thực biểu diễn dạng số có dấu chấm động (floating-point) ở cả hệ số và dữ liệu, bên cạnh đó là phép nhân cộng tích lũy (MAC) và cả hai đều gặp nhiều hạn chế khi chạy trên FPGA. Để giải quyết thì trong đề tài này, một bộ lọc FIR 16 hệ số lọc đối xứng được xây dựng bằng phương pháp cửa sổ Hamming, các dữ liệu đầu vào và các hệ số học được tạo bởi các hàm MATLAB và được chuyển sang kiểu số nguyên nhị phân 8 bit. Trước tiên, ta sẽ xây dựng thử nghiệm với MATLAB, sau đó viết chương trình C để kiểm tra. Cuối cùng là xây dựng mã Verilog HDL sử dụng thuật toán thay thế cho việc sử dụng phép nhân cộng tích lũy (MAC) bằng cách sử dụng cơ sở của số học phân tán (DA). Kết quả được kiểm tra, xem xét bằng các sử dụng phần mềm MATLAB, Visual Studio Code; Intel Quartus Prime, ModelSim để thực hiện trên

board Terasic DE10-Standard Development Kit sử dụng chip Altera Cyclone V để kiểm tra hiệu năng và kết quả của mã Verilog HDL.

1.3 Giới thiệu về MATLAB

MATLAB là bản viết tắt của Matrix Laboratory. Đây là một ngôn ngữ hiệu suất cao được sử dụng để tính toán kỹ thuật. Nó được phát triển bởi Cleve Molar của công ty MathWorks.Inc vào năm 1984. Nó được viết bằng C, C ++, Java. Nó cho phép các thao tác ma trận, vẽ biểu đồ các hàm, phát triển khai báo thuật toán và tạo giao diện người dùng. Đây là một trong những công cụ quan trọng nhất trong lĩnh vực tính toán kỹ thuật và khoa học.

MATLAB là ngôn ngữ ma trận cấp cao, cú pháp lệnh dựa trên C/++, với nhiều tính năng nổi trội như hỗ trợ lập trình hướng đối tượngú pháp đơn giản, dễ học và sử dụng, tích hợp các thuật toán toán học và kỹ thuật tối ưu, khả năng xử lý song song và phân tán.



Hình 1.1: Giao diện chính phần mềm MATLAB

Các thành phần trong cửa sổ giao diện của MATLAB

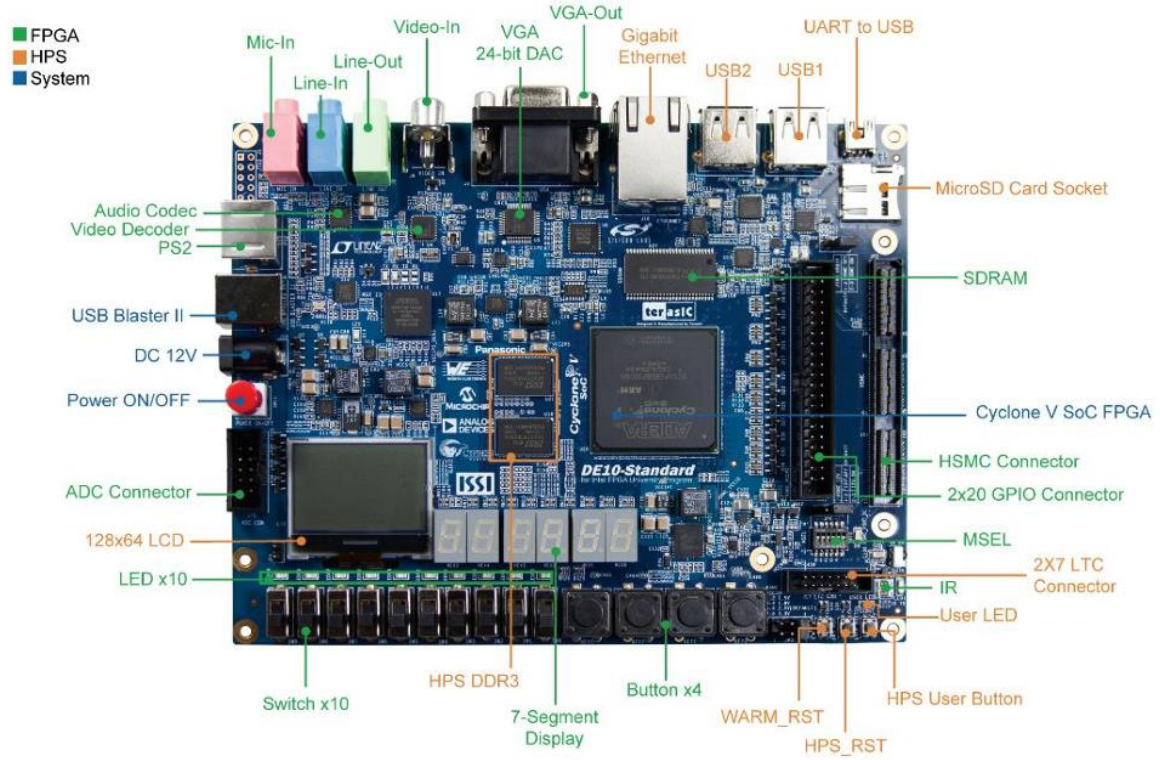
- MATLAB Desktop: Giao diện chính tích hợp
- Command Window: Cửa sổ dòng lệnh tương tác, ở đây người dùng sẽ gõ lệch trực tiếp và chương trình sẽ thực hiện ngay lập tức, phù hợp với chương trình nhỏ, dễ thực thi.

- Command History: Hiển thị nhật ký các lệnh trong các phiên làm việc hiện tại và trước đó. Ở đây sẽ liệt kê ngày, giờ theo thời gian của hệ thống và danh sách các lệnh đã thực thi.
- Editor: Soạn thảo mã nguồn với tính năng gợi ý và debug, phù hợp cho các chương trình lớn, sử dụng nhiều lệnh, hàm phức tạp, và có thể được lưu trữ dài hạn với phần mở rộng ".m".
- Workspace Browser: Quản lý biến và dữ liệu, khu vực lưu trữ các biến, dữ liệu… được tạo ra trong quá trình sử dụng.
- Help Browser: Tài liệu hướng dẫn tích hợp

MATLAB có thư viện được tích hợp nhiều hàm sẵn có, hỗ trợ các phép toán học, tính toán ma trận, vẽ đồ thị, nhập xuất, đọc/ghi file, hiển thị ký tự, hình ảnh… Ngoài ra, MATLAB còn có các Toolbox hỗ trợ các ứng dụng chuyên biệt như xử lý tín hiệu, xử lý hình ảnh, mã hóa, phân tích toán học, thống kê, thiết kế hệ thống truyền thông, điều khiển, lập trình giao diện ứng dụng…

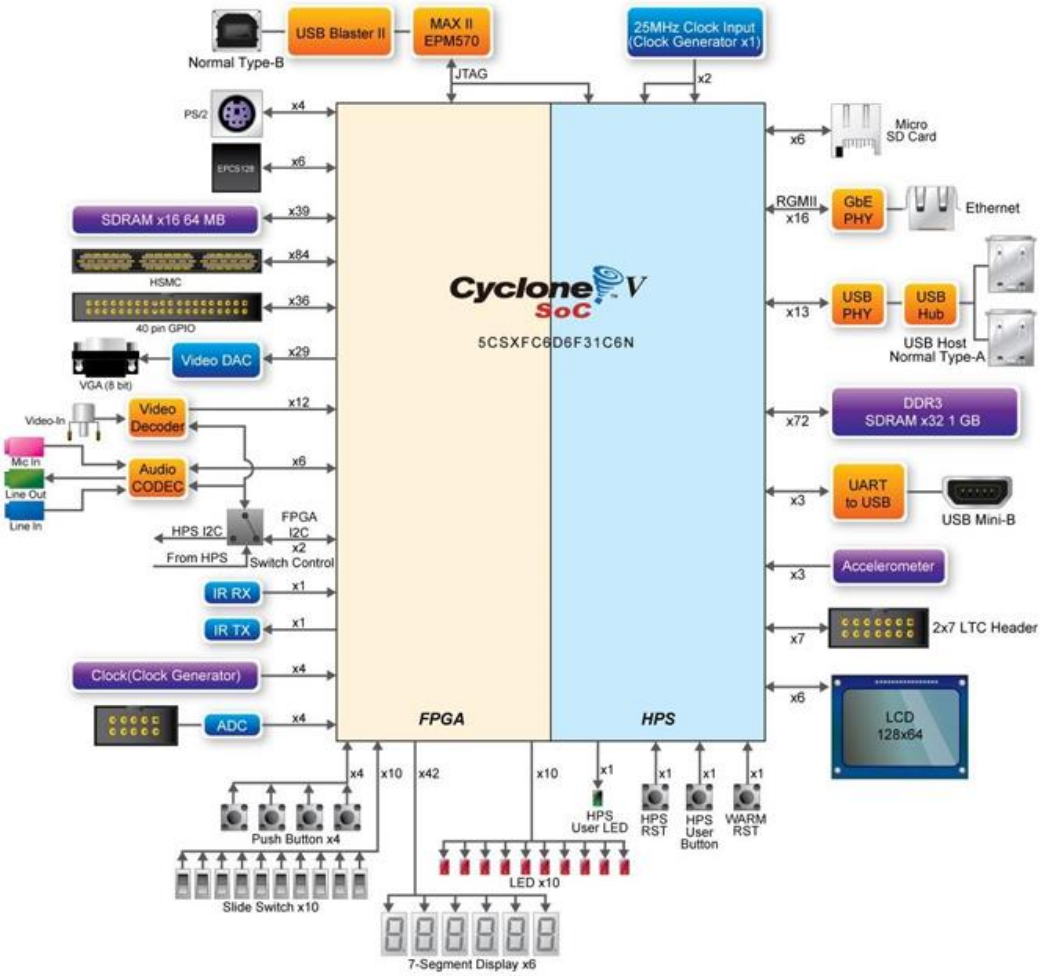
1.4 Giới thiệu về board Terasic DE10- Standard

Board DE10-Standard của hãng Terasic (Đài Loan) là một nền tảng thiết kế phần cứng mạnh mẽ xây dựng xung quanh vi xử lý hệ thống (System on Chip) FPGA của Intel. SoC của Intel tích hợp một hệ thống xử lý cứng dựa trên ARM (Hard Processor System) bao gồm bộ xử lý, các thiết bị ngoại vi và giao diện bộ nhớ liên kết mượt mà với fabric FPGA bằng cách sử dụng một mạng lưới kết nối trục lớn băng thông. Bảng phát triển DE10-Standard trang bị bộ nhớ DDR3 tốc độ cao, khả năng video và âm thanh, mạng Ethernet, …



Hình 1.2: Các thành phần trên board DE10 Standard

Bảng phát triển DE10-Standard có nhiều tính năng cho phép người dùng triển khai một loạt các mạch được thiết kế, từ mạch đơn giản đến các dự án đa phương tiện đa dạng.



Hình 1.3: Sơ đồ khối của board DE10 Standard

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Lọc phi đệ quy (FIR)

2.1.1 Giới thiệu

Như đã đề cập trên, lọc là quá trình loại bỏ các tín hiệu nhiễu ra khỏi tín hiệu thông tin. Một bộ lọc số thường được cấu tạo bởi các mạch điện tử (phần cứng) hoặc dùng chương trình (phần mềm) hoặc kết hợp cả hai. Có hai loại lọc số: loại độ dài hàm đáp ứng xung hữu hạn (hay tín hiệu ra chỉ phụ thuộc vào một số hữu hạn các tín hiệu vào) được gọi là lọc FIR và loại lọc số độ dài xung vô hạn (IIR) . Các bộ lọc FIR được sử dụng rộng rãi trong xử lý tín hiệu số do tính ổn định, có pha tuyến tính (trừ một số trường hợp đặc biệt) và khả năng thiết kế dễ dàng bởi chỉ cần hữu hạn bộ nhớ để lưu trữ. Tuy nhiên, với cùng giá trị hàm đáp ứng tần số thì lọc FIR lại kém hiệu quả hơn nhiều so với lọc IIR. Phương trình thể hiện lọc FIR là:

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \qquad (1)$$

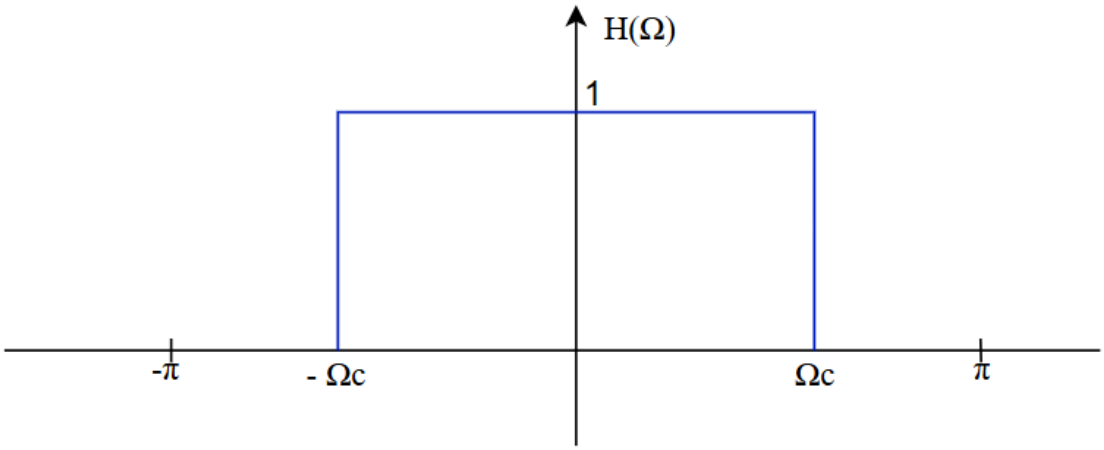
Thông thường, bộ lọc FIR thường được coi là nhân quả (chỉ xét các thời điểm kể từ khi n = 0 trở đi) nên phương trình (1) có thể viết lại thành

$$y(n) = \sum_{k=0}^{N} h(k)\,x(n-k) \qquad (2)$$

Với N là bậc của lọc, và chiều dài hàm đáp ứng (số các hệ số lọc) là M = N +1. Trên lý thuyết N có thể lớn vô hạn.

2.1.2 Lọc thấp qua

Đáp ứng tần số và đáp ứng xung lý tưởng của lọc thấp qua là một đôi liên hệ nhau thông qua biến đổi DTFT và IDTFT. Vì đáp ứng tần số tuần hoàn ở chu kỳ 2π nên chỉ cần tính đáp ứng xung trong khoảng Nyquist (-π, π).



Hình 2.1: Độ lớn hàm đáp ứng xung của lọc lý tưởng thấp qua.

Với tần số số cắt của một lọc thấp qua là Ωc = $\frac{2\pi f_c}{f_s}$ (fc là tần số cắt tương tự), hàm đáp ứng xung lý tưởng là :

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\Omega)e^{jn\Omega}d\Omega = \frac{1}{2\pi}\int_{-\Omega_c}^{\Omega_c} 1e^{jn\Omega}d\Omega$$

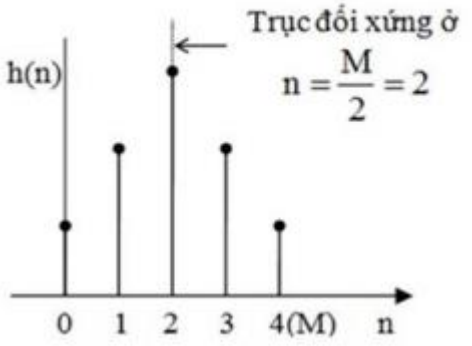
- Với n = 0 :

$$h(n) = \frac{1}{2\pi}\int_{-\Omega_c}^{\Omega_c} 1e^{jn\Omega}d\Omega = \frac{1}{2\pi}\Omega\Big|_{-\Omega_c}^{\Omega_c} = \frac{\Omega_c}{\pi} \qquad (3)$$

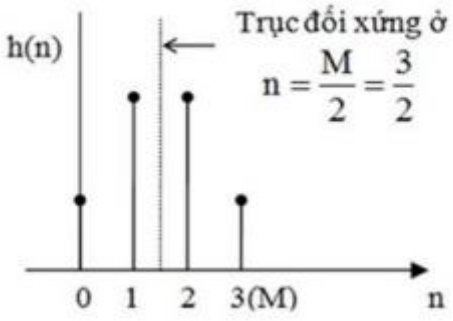
- Với n ≠ 0:

$$h(n) = \frac{1}{2\pi}\left[\frac{e^{jn\Omega}}{jn}\right]_{-\Omega_c}^{\Omega_c} = \frac{1}{2\pi}\frac{1}{jn}(e^{j\Omega_c n} - e^{-j\Omega_c n}) = \frac{\sin(n\Omega_c)}{n\pi} \qquad (4)$$

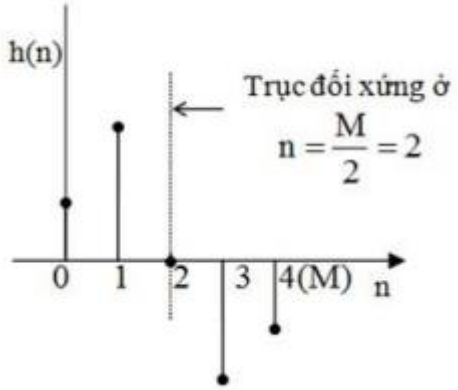
Không chỉ đối với lọc thấp qua mà các loại FIR nhân quả khác cũng được mô tả bởi phương trình hiệu số, tùy thuộc vào bậc lọc của lọc là chẵn hay thẻ mà hàm đáp ứng xung (còn gọi là hệ số lọc) có thể đối xứng (hay đối xứng chẵn) và đối đối xứng (đối xứng lẻ). Tùy vào tính chất đối xứng chẵn lẻ này mà loại lọc tương ứng có những ứng dụng khác nhau.



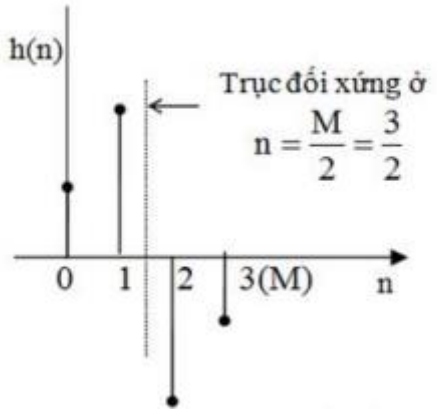
a)Bậc lọc chẵn, h(n) đối xứng
h(n) = h(M-n)



b) Bậc lọc lẻ, h(n) đối xứng)
h(n) = h(M-n)



c) Bậc lọc chẵn, h(n) đối đối xứng
h(n) = -h(M-n)



d) Bậc lọc lẻ, h(n) đối đối xứng
h(n) = -h(M-n)

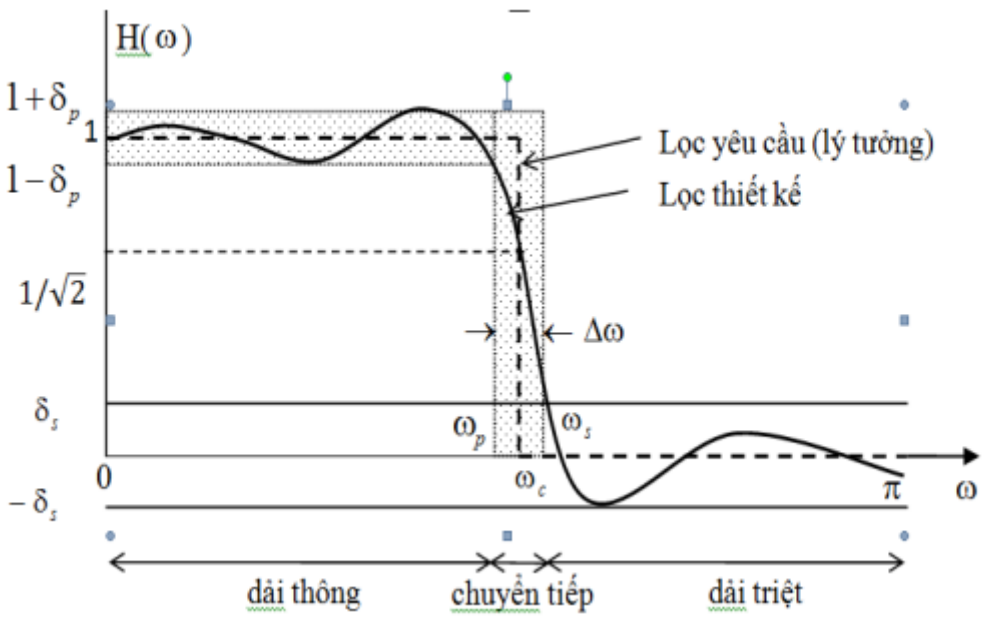
Hình 2.2: Biểu diễn các loại hàm đáp ứng xung

2.1.3 Phương pháp cửa sổ

Trong kết quả trên, ta sử dụng biến đổi Fourier rời rạc thời gian và các tính chất của nó nên các làm trên còn được gọi là phương pháp Fourier. Tuy nhiên, phương pháp Fourier thường có sai số lớn và dẫn đến đáp ứng xung lâu vô hạn khi thiết kế, bên cạnh đó, dạng thực tế khi thiết kế thì hình dạng hàm đáp ứng tần số không được như lý tưởng mà sai lệch khá nhiều. Các chỉ tiêu sai lệch khi thiết kế được xét tới là

- Tần số cạnh dải thông (passband edge frequency) ω_p, tần số cạnh dải triệt (stopband edge frequency) ω_s, tần số cắt (cutof frequency) ω_c mà thường được xem là $\omega_c = (\omega_p + \omega_s)/2$
- Dải thông $[0, \omega_p]$, dải chuyển tiếp (transition band) $\Delta\omega = \omega_s - \omega_p$, dải triệt $[\omega_s, \pi]$

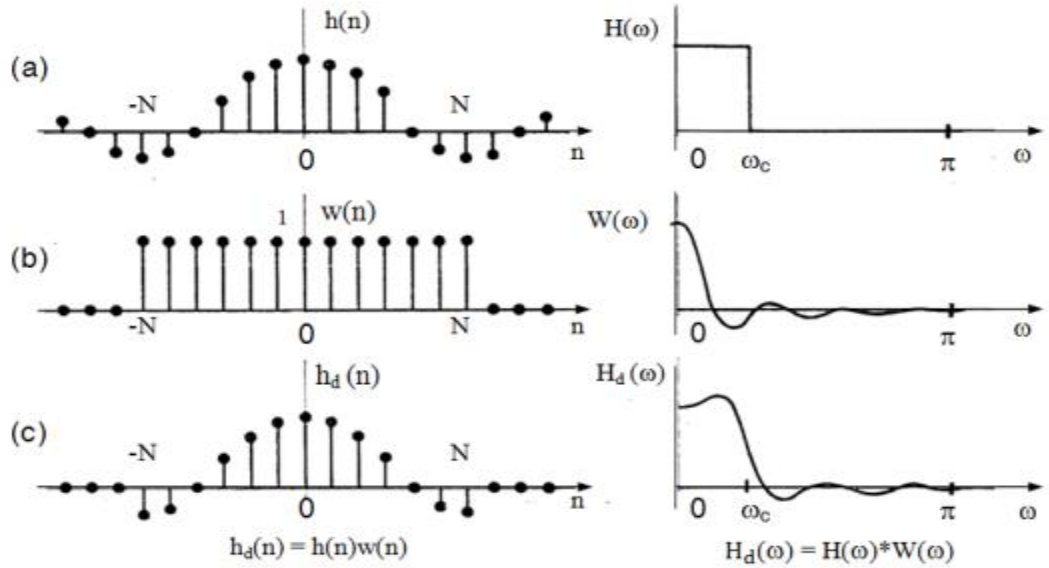
- Dợn sóng dải thông (passband ripple) δp, dợn sóng dải triệt (stopband ripple) δs.Hai dợn sóng này thường được xem bằng nhau, nhưng trong nhiều thiết kế chúng được yêu cầu khác nhau.



Hình 2..3: Hàm đáp ứng tần số thực tế và các yêu cầu thiết kế

Vì vậy, người ta thường loại bỏ các mẫu ở xa gốc của hàm đáp ứng xung (còn gọi là cắt cụt) và cố gắng đạt tới các yêu cầu thiết kế bằng cách nhân hàm đáp ứng xung h(n) với một hàm cửa sổ w(n) rộng hữu hạn.

$$b_n = h(n)w(n) \qquad (4)$$



Hình 2.4: Tác động của áp dụng cửa sổ trong miền thời gian (bên trái) và miền tần số (bên phải).

Phương pháp Fourier kết hợp với của sổ này thường được gọi là phương pháp cửa sổ. Có nhiều loại hàm cửa sổ khác nhau, với các ưu nhược điểm khác nhau như cửa sổ tam giác (cửa sổ Bartlett), của sổ Hanning, cửa sổ Hamming, cửa sổ Blackmann,…

Chữ nhật $w(n)=1 \quad 0\le n\le M$ (2.33a)

Tam giác (Barlett) $w(n)=\frac{2n}{M}, \quad 0\le n\le \frac{M}{2}$ (2.33b)

$2-\frac{2n}{M}, \quad \frac{M}{2}\le n\le M$ (2.33c)

Hanning (von Hann): $w(n)=0.5-0.5\cos\left(\frac{2\pi n}{M}\right), \quad 0\le n\le M$

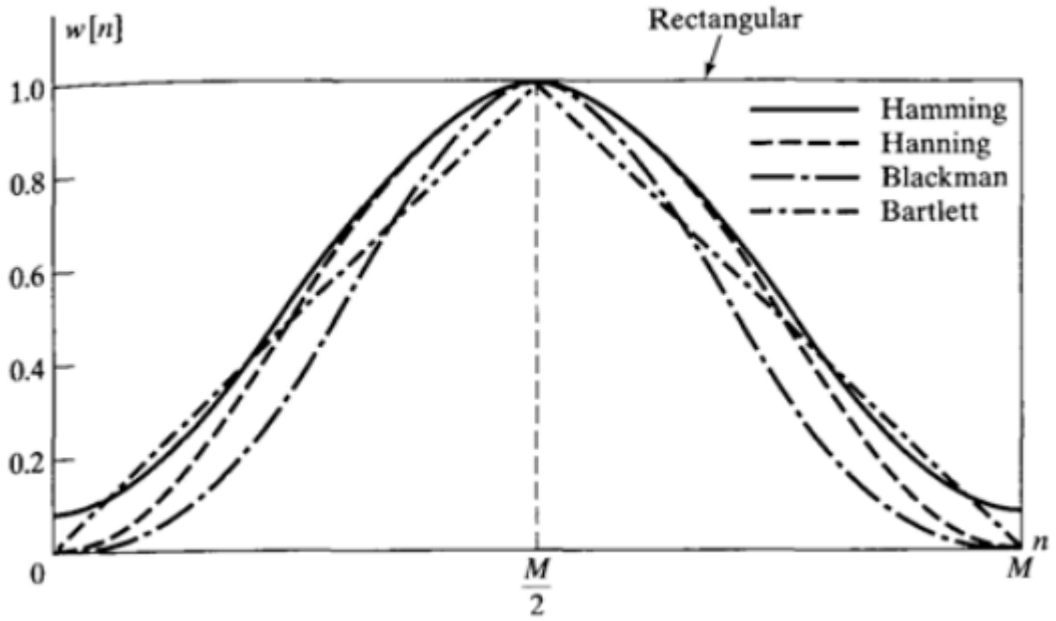
Hamming: $w(n)=0.54-0.46\cos\left(\frac{2\pi n}{M}\right), \quad 0\le n\le M$ (2.33e)

Blackman: $w(n)=0.42-0.5\cos\left(\frac{2\pi n}{M}\right)+0.08\cos\left(\frac{4\pi n}{M}\right), \quad 0\le n\le M$ (2.33f)

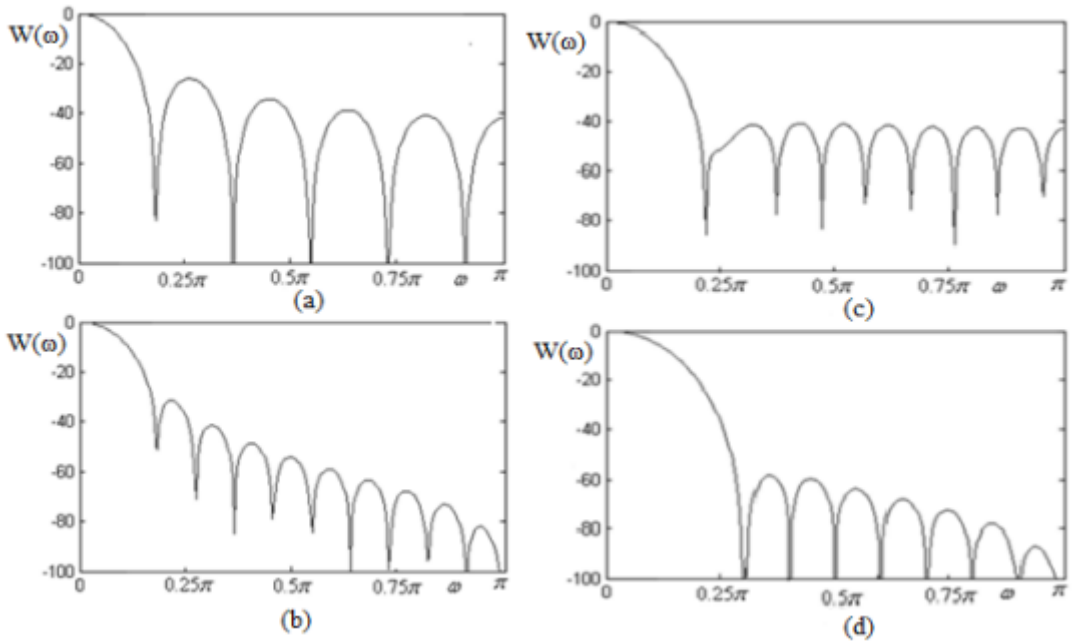
Hình 2.5: Biểu thức hàm cửa sổ

Loại cửa sổ	Độ rộng múi chính (rad/mẫu)	Độ rộng chuyển tiếp (rad/mẫu	Độ suy giảm từ múi chính tới múi phụ (dB)	Suy giảm dải triệt tối đa
Rectangular	4π/M	1.8π/M	-13	21
Barlett	8π/M	-	-25	-
Hanning	8π/M	6.2π/M	-31	44
Hamming	8π/M	6.6π/M	-41	53
Blackmann	12π/M	11.1π/M	-57	74

Bảng 2.1: So sánh tổng quan thông số các hàm cửa sổ



Hình 2.6: So sánh độ biến thiên của các loại cửa sổ



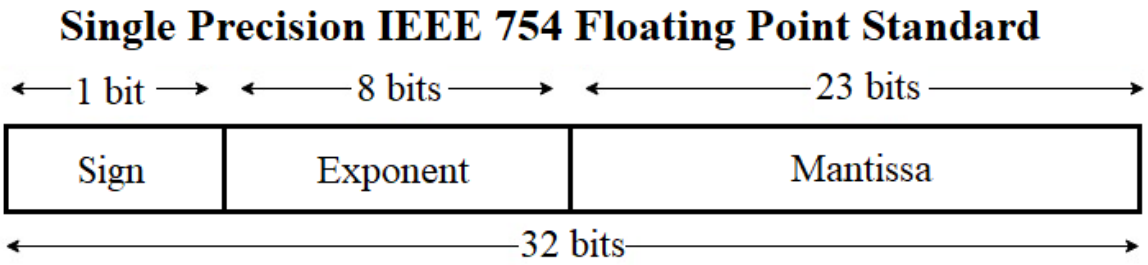
Hình 2.7: Đáp ứng dB của các cửa sổ cố đinh với M = 20

(a) Bartlett, (b) Hanning, (c) Hamming, (d) Blackman

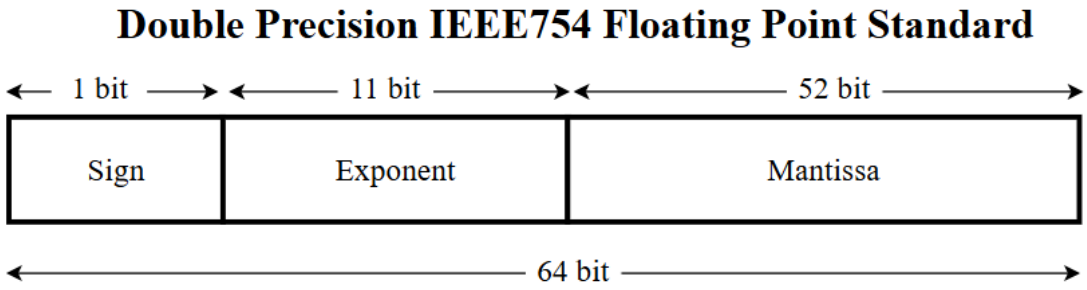
Từ những mô tả ở trên về các loại cửa sổ, ta thấy rằng không có có cửa số nào là tốt nhất cho mọi yêu cầu kỹ thuật nhưng là sự bù trừ lẫn nhau. Ví dụ cửa sổ chữ nhật có độ rộng múi chính hẹp nhất, Blackman có múi bên suy giảm nhiều nhất, và cửa sổ ở giữa Hamming dung hòa tốt nhất các yếu tố ưu nhược điểm của các cửa sổ nên thường được dùng trong thiết kế lọc FIR.

2.2 Số floating-point và fixed-point

Như đã bàn trong mục trên, khi thiết kế lọc FIR dùng cửa sổ hoặc không dùng cửa sổ thì các hệ số lọc cũng sẽ là một giá trị theo hàm sin (cos), và là một số thực. Đối với một số thực bất kỳ, để hệ thống máy tính có thể hiểu và biểu diễn, số ấy sẽ được chuyển sang dạng nhị phân floating-point. Theo tiêu chuẩn IEEE754, có hai dạng biểu diễn số floating-point, một là Single Precision (32bit) và Double Precision (64bit).



Hình 2.8: Single Precision theo chuẩn IEEE754



Hình 2.9: Double Preccision theo chuẩn IEE754

Một số **floating-point** dạng Single hoặc Double Precision được biểu diễn dưới dạng:

$$x=(-1)^{\text{Sign}} \times 2^{\text{Exponent - Bias}} \times (1.\text{Mantissa})$$

- **Sign**
  - Xác định dấu của số.
  - $(-1)^{\text{Sign}}$: Sign = 0 biểu diễn số dương, Sign = 1 biểu diễn số âm.

- **Exponent**:
  - Biểu diễn giá trị mũ theo dạng cơ số 2.
  - Bias (độ lệch): Để đảm bảo exponent luôn dương khi lưu trữ trong các bit.
    - **Bias của Single Precision**: 127.
    - **Bias của Double Precision**: 1023.
  - Giá trị thực của exponent là **Exponent−Bias**
- **Mantissa**:
  - Biểu diễn phần thập phân nhị phân của số, có dạng 1.Mantissa (normalized – dạng chuẩn hóa).

Về phạm vi biểu diễn sẽ tùy thuộc vào khoảng giá trị của exponent và Mantissa cũng như là bit dấu. Với trường hợp khi biểu diễn số dạng đặc biệt (NaN, Inf)

- **Single Precision**
  - NaN: Exponent = 255, Mantissa ≠ 0
  - Inf: Exponent = 255, Mantissa = 0
- **Double Precision**
  - NaN: Exponent = 2047, Mantissa ≠ 0
  - Inf: Exponent = 2047, Mantissa = 0

Mặc dù các số floating-point có vùng biểu diễn phạm vi giá trị rất rộng từ gần 0 tới một giá trị rất lớn, với độ chính xác cao nhờ vùng biểu diễn giá trị của exponent và mantissa, tuy nhiên việc tính toán các giá trị này lại yêu cầu tài nguyên tính toán cao (logic, bộ nhớ, băng thông, tốc độ tính) khiến việc tính toán và xử lý với số floating-point trên các nền tảng phần cứng hạn chế như FPGA là hết sức khó khăn và phức tạp, đặc biệt là yêu cầu về FPU – Floating Point Unit để xử lý chuyên biệt, làm tốc độ tính toán rất chậm chạp, và không phù hợp cho các ứng dụng tốc độ cao như lọc số, xử lý thời gian thực, xử lý ảnh và video… trên FPGA.

Để giải quyết vấn đề này thì người ta sẽ nhân số floating-point với một hệ số tỷ lệ (scaling factor) và làm tròn kết quả, và gọi nó là số fixed-point. Số fixed-point chỉ yêu cầu một số bit nhất định để biểu diễn giá trị, giúp tiết kiệm bộ nhớ và tài nguyên xử lý trên FPGA hoặc các hệ thống nhúng. Điều này làm cho số fixed-point rất phù hợp cho các hệ thống với tài nguyên hạn chế. Các phép toán fixed-point giống như là tính toán cho các số nguyên có dấu thông thường, và có thể thực hiện trực tiếp trên các phần cứng cơ bản của FPGA mà không cần các phần cứng phức tạp. Điều này làm tăng tốc độ tính toán và giảm độ trễ, rất quan trọng trong các ứng dụng DSP yêu cầu xử lý thời gian thực.

Tuy vậy, các số fixed-point có phạm vi biểu diễn hẹp hơn so với số floating-point, độ chia nhỏ nhất của số fixed-point cũng là khá cao do không có cơ chế như exponent và mantissa như số floating-point. Việc lựa chọn số bit cho phần nguyên và phần thập

phân là rất quan trọng và cần phải được tính toán chính xác, nếu lựa chọn không hợp lý, có thể xảy ra các lỗi làm tròn hoặc mất mát dữ liệu.

Các bước để chuyển từ số floating-point sang fixed-point là:

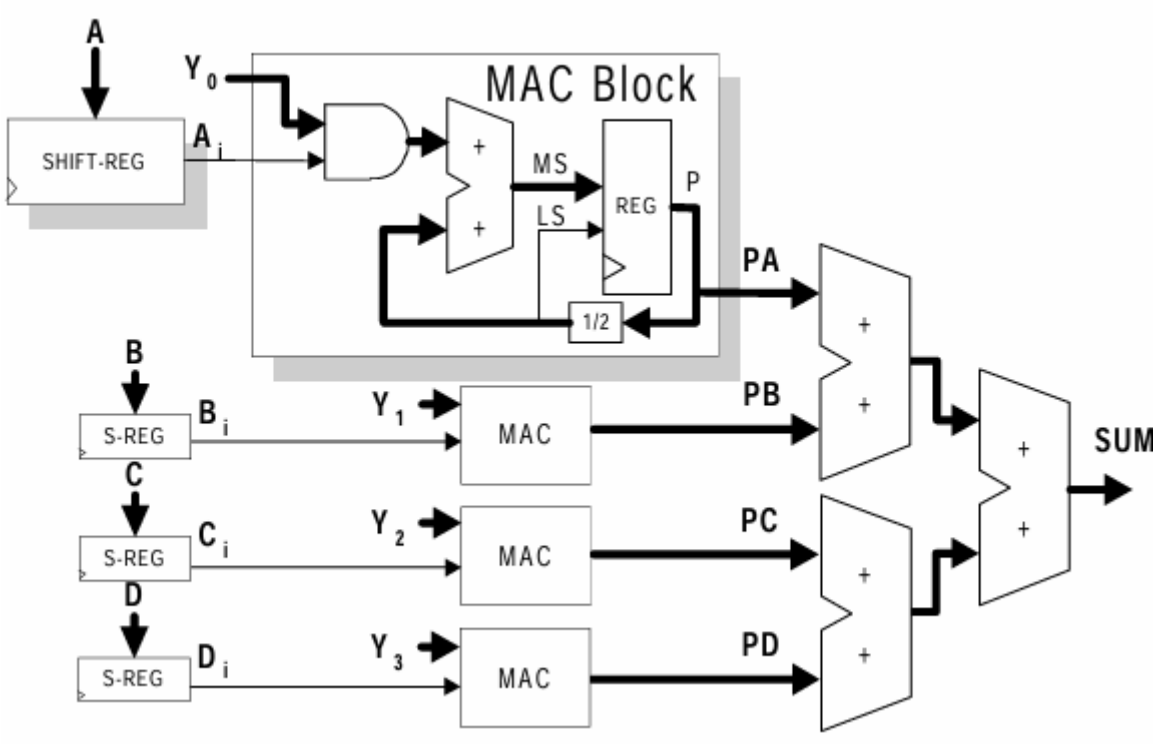
Giả sử ta có một số có dấu hoặc không dấu floating-point a bất kỳ

- Bước 1: Tính giá trị b = a x 2^F, với F là số bit làm tròn, và F là số nguyên dương.
- Bước 2: Làm tròn giá trị b về số nguyên gần nhất, ví dụ b = 3.7 thì làm tròn lên 4, b là -0.7 thì làm tròn về -1, gọi là giá trị c.
- Bước 3: So sánh giá trị c với khoảng khoảng giá trị yêu cầu. Giả sử nếu yêu cầu là biểu diễn số có dấu fixed-point 8 bit như vậy khoảng biểu diễn từ -127 tới 128. Khi này nếu c lớn hơn 128 thì chuẩn hóa cho c là 128, nếu c nhỏ hơn -127 thì cho c là -127.

2.3 Cơ sở lý thuyết về số học phân tán (DA)

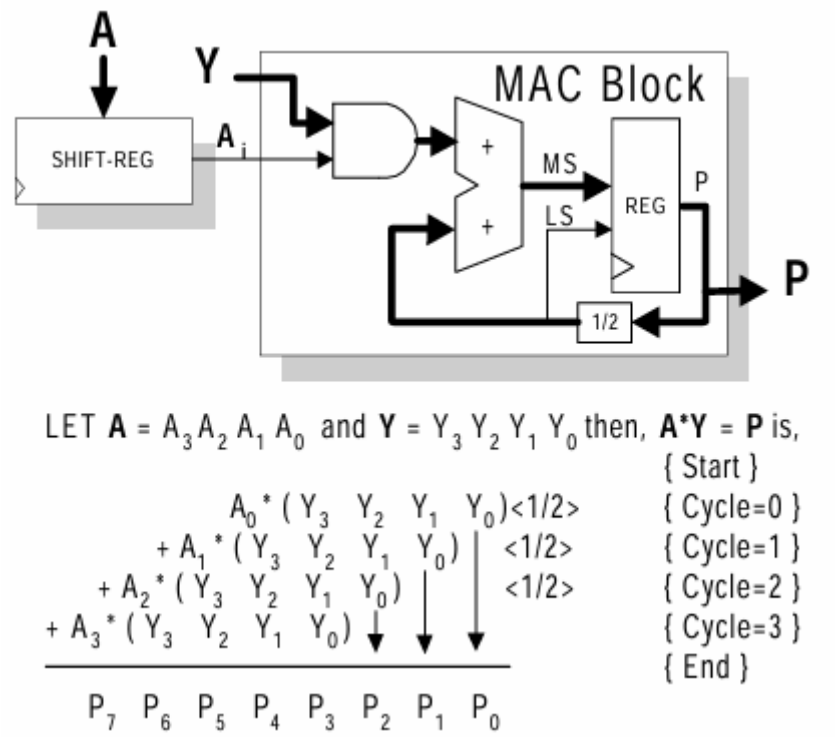
Trong phương trình (2) của lọc FIR nhân quả ở trên, giá trị đầu ra là kết quả của các phép nhân cộng tích lũy (MAC) bao gồm các phép nhân giữa các hệ số lọc và các giá trị tín hiệu vào, sau đó cộng các giá trị tích vào tổng. Khi thay đổi số lượng hệ số lọc của bộ lọc, số lượng các phép nhân cộng tích lũy thay đổi theo. Khi số phép nhân cộng tích lũy tăng lên, yêu cầu tài nguyên cho bộ lọc FIR tăng lên do số lượng phép tính và bộ nhớ lưu trữ lớn hơn, đặc biệt là phép nhân với các kiến trúc dựa trên CPU, đặc biệt là FPGA.

Số học phân tán (DA) là một phương pháp tối ưu cho việc thực hiện các phép toán số học trong các hệ thống xử lý tín hiệu số (DSP), đặc biệt là với phép nhân cộng tích lũy MAC. Số học phân tán chuyển đổi các phép nhân và cộng phức tạp thành các phép toán tra cứu bảng (LUT) và cộng đơn giản, làm giảm đáng kể khối lượng tính toán cần thiết. DA tận dụng tính chất của các phép toán nhị phân để biểu diễn các phép toán nhân như là các phép tra cứu trong bảng LUT.
Đầu tiên, xét mô hình hàm gồm 4 phép nhân cộng tích lũy như sau



Hình 2.10: Mô hình hàm 4 phép nhân cộng tích lũy

Trong đó, Y đóng vai trò như số nhân (multiplicand) và A là số bị nhân (multiplier). Phép nhân được tính theo từng cặp bằng cách cộng dồn từng giá trị của phép and từng bit của A với Y, sau đó là cộng dồn giá trị và thực hiện dịch bit giá trị thanh khi kết quả. Giá trị cuối cùng là tổng của 4 phép nhân này. Ví dụ về phép nhân hai số 4-bit không dấu như trong hình dưới.

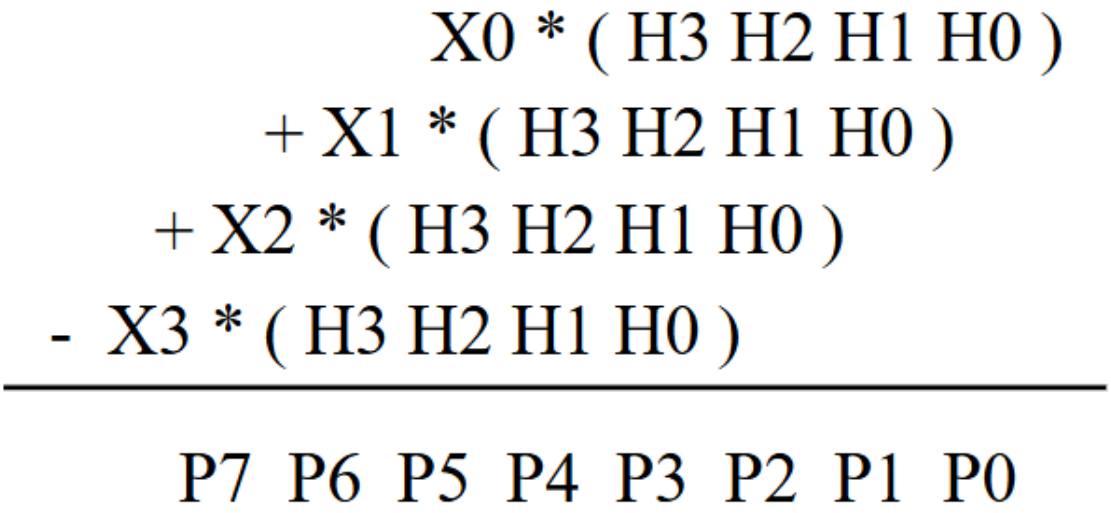


Hình 2.11: Chi tiết cách tính phép nhân

Về mặt tổng quan, với một giá trị số nguyên có dấu x(n) thứ k bất kỳ được biểu diễn dưới dạng nhị phân bù 2 như sau:

$$x_k = -x_{k(B-1)}2^{B-1} + \sum_{b=0}^{B-2} x_{kb}2^b \qquad (5)$$

Khi này, ở bước cuối cùng thì thay vì cộng thì ta sẽ thực hiện phép trừ như sau



Hình 2.12: Phép nhân hai số có dấu dạng bù 2

Với B là số bit biểu diễn của của các giá trị x(n), x_{kb} là giá trị bit thứ b của x thứ k ($0 \leq b \leq B-1$). Từ phương trình (2), ta viết nó lại một lần nữa:

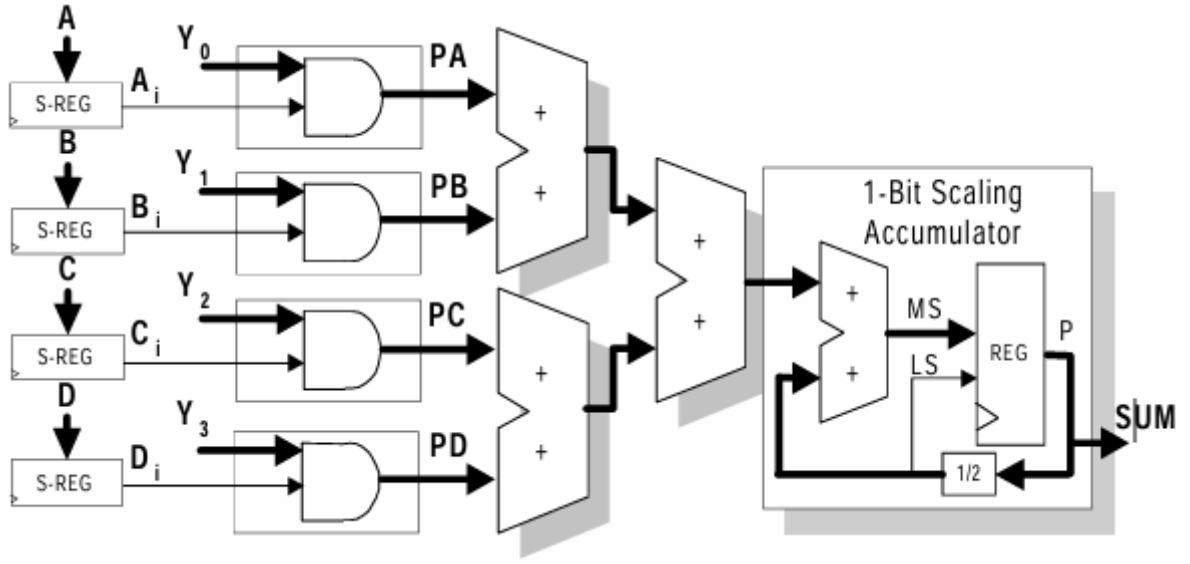
$$y(n) = \sum_{k=0}^{N} h_k \, x_{n-k}$$

Với x_k là giá trị vào thứ k tại thời điểm n. Từ phương trình trên và (5) mô tả trước đó, ta có:

$$y(n) = \sum_{k=1}^{K} h_k \left[- x_{(n-k)(B-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{(n-k)b} 2^b \right]$$

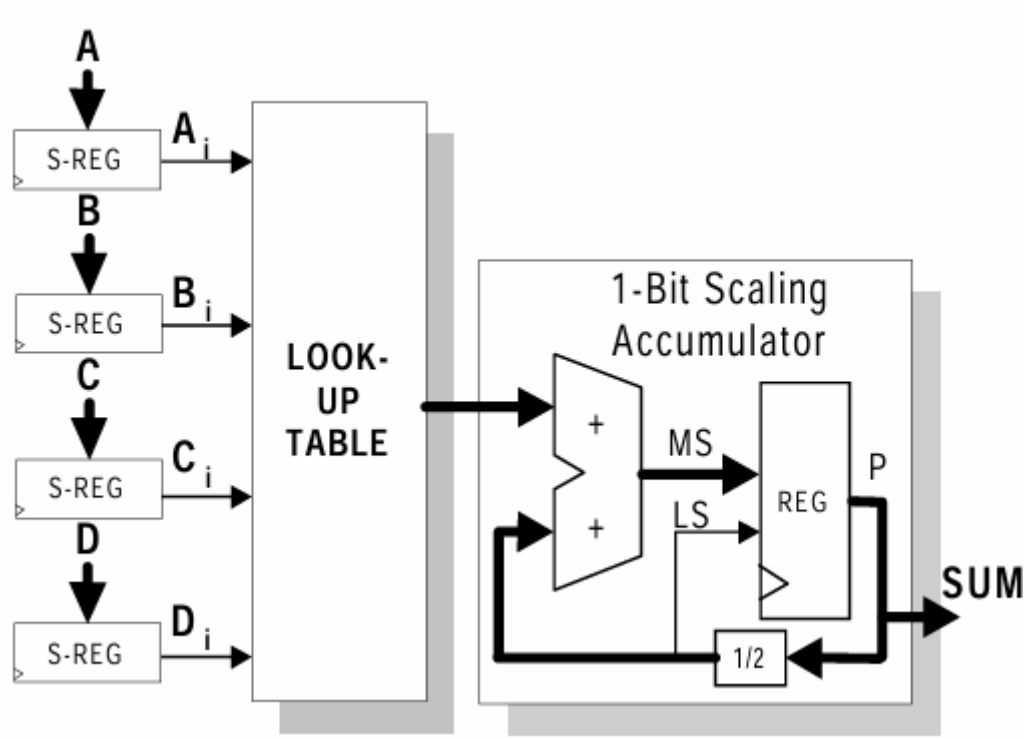
$$= - \sum_{k=1}^{K} x_{(n-k)(B-1)} 2^{B-1} h(k) + \sum_{k=1}^{K} \sum_{b=0}^{B-2} x_{(n-k)b} 2^b \, h(k)$$

Và mô hình phép nhân cộng tích lũy có thể biểu diễn thành



Hình 2.13: Phép nhân cộng tích lỹ theo số học phân tán

Khi này, thay vì phải xét kết quả phép nhân (hay chính là AND) của các giá trị Y với từng bit input rồi tính tổng, thì ta sẽ tổng hợp lại các giá trị có thể xảy ra của phép toán này theo tổ hợp giá trị của các bit input đầu vào, và tạo thành bảng tra cứu LUT để đơn giản hóa quá trình tính toán.



Hình 2.14: Mô hình phép nhân cộng dùng LUTs

Đầu vào				**Kết quả**
Ai	Bi	Ci	Di	
0	0	0	0	0
0	0	0	1	Y0
0	0	1	0	Y1
0	0	1	1	Y0 + Y1
0	1	0	0	Y2
0	1	0	1	Y2 + Y0
0	1	1	0	Y2 + Y1
0	1	1	1	Y2 + Y1 + Y0
1	0	0	0	Y3
1	0	0	1	Y3 + Y0
1	0	1	0	Y3 + Y1
1	0	1	1	Y3 + Y1 + Y0
1	1	0	0	Y3 + Y2
1	1	0	1	Y3 + Y2 + Y0
1	1	1	0	Y3 + Y2 + Y1
1	1	1	1	Y3 + Y2 + Y1 + Y0

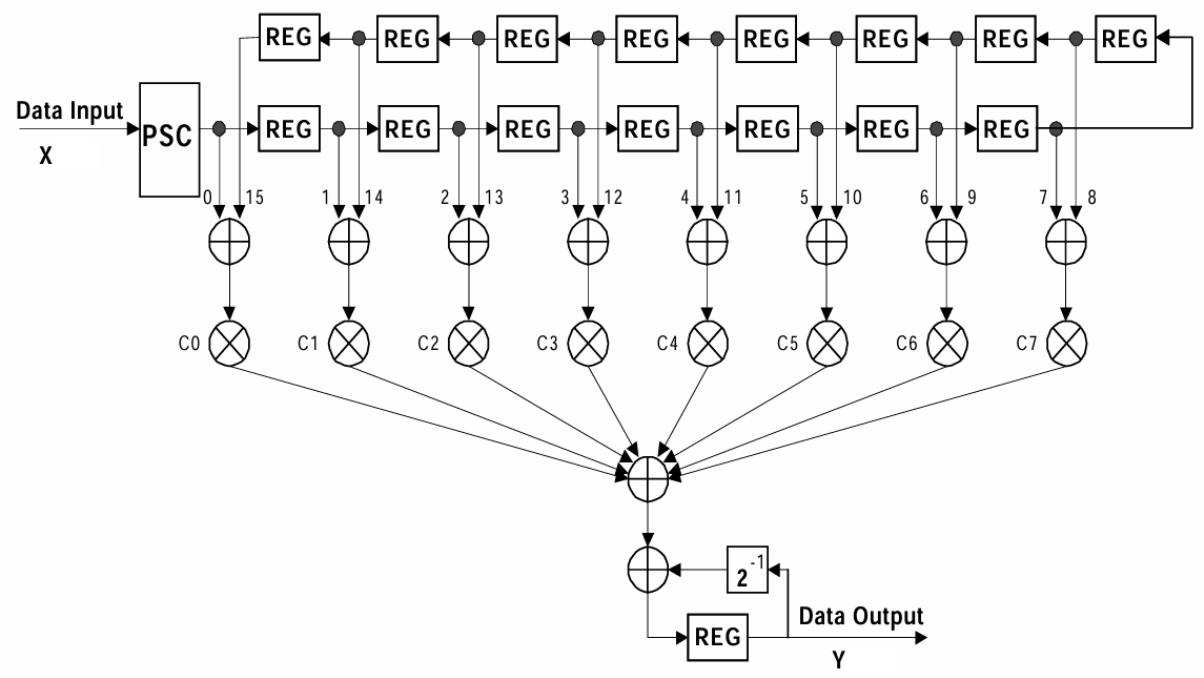
Bảng 2.2: *Tổng quan bảng LUT*

2.4 Mô hình thuật toán

Vì đang thực hiện các mô hình trên kiến trúc của FPGA nên còn có một cách khác để giảm thiểu số phép nhân phải sử dụng, đó là thiết kế các hệ số lọc đối xứng, khi này các mẫu đầu vào được lưu trữ và cộng theo từng cặp. Từ đó giảm đi một nửa số phép nhân khi sử dụng nhân cộng tích lũy, nhưng sẽ nảy sinh ra một số vấn đề khác khi sử dụng số học phân tán. Điều này sẽ được bàn tới ở các mục sau.

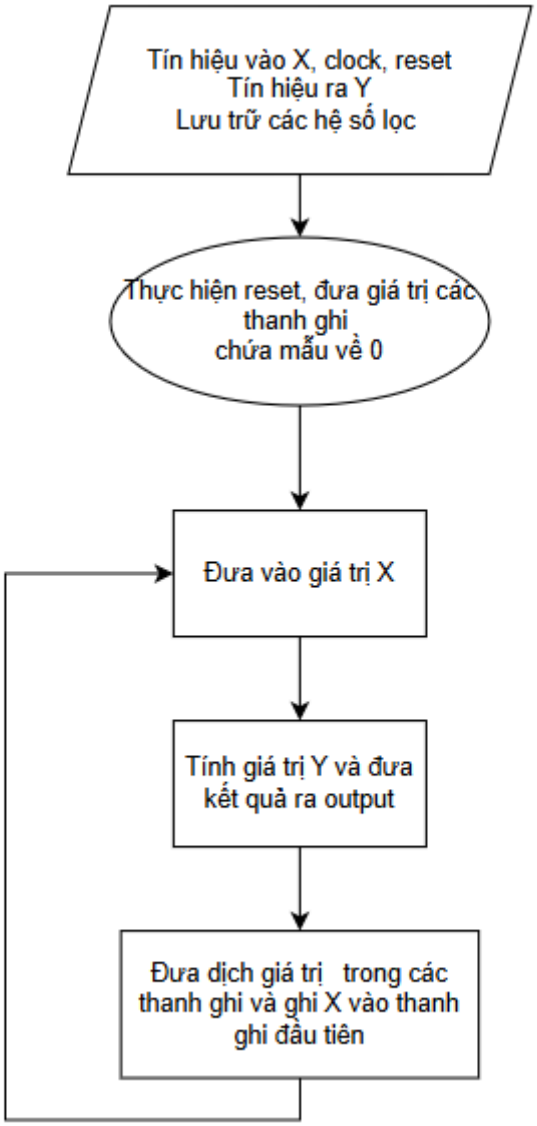
2.4.1 Sử dụng phép nhân cộng tích lũy (MAC)

Về cơ bản, lọc FIR 8 bit, 16 hệ số là lọc rời rạc thời gian, với tín hiệu ra là một hàm liên hệ giữa các giá trị mẫu ở hiện tại và trước đó. Và khi thực hiện trên FPGA, các giá trị mẫu ở hiện tại và trước đó được lưu trữ trong các thanh ghi, và được cộng theo từng cặp (do các hệ số lọc là đối xứng) trước khi nhân với hệ số lọc và được cộng dồn để thành kết quả ra. Để áp ứng việc đưa và lưu trữ các giá trị mẫu trước đó và hiện tại cho đúng, ta sẽ thực hiện dịch giá trị của mẫu được lưu trữ trong các thanh ghi mỗi khi tính ra kết quả, để nhường chỗ cho việc lưu trữ giá trị mới



Hình 2.15: Mô hình khối dữ liệu sử dụng nhân cộng tích lũy

Từ mô hình khối dòng dữ liệu của lọc FIR ở trên, ta xây dựng lưu đồ giải thuật khi thiết kế với ngôn ngữ Verilog HDL như sau



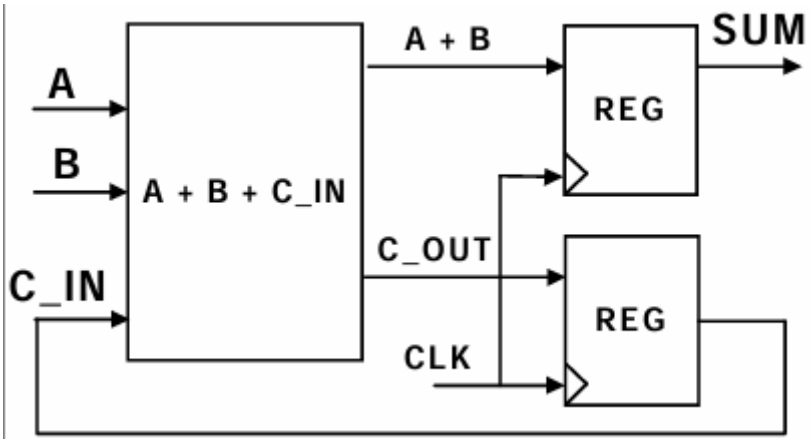
Hình 2.16: Lưu đồ giải thuật với phép nhân cộng tích lũy

2.4.2 Sử dụng số học phân tán (DA)

Như đã đề cập ở trên, với việc sử dụng số học phân tán (DA) thì sẽ chuyển các phép nhân thảnh các phép cộng để thực hiện tra cứu qua bảng LUT. Và tùy vào loại thực hiện và cộng cách song song hoặc tuần tự thì sẽ có 2 trường phái khi sử dụng số học phân tán này là thực hiện tuần tự (SDA) và thực hiện song song (PDA).

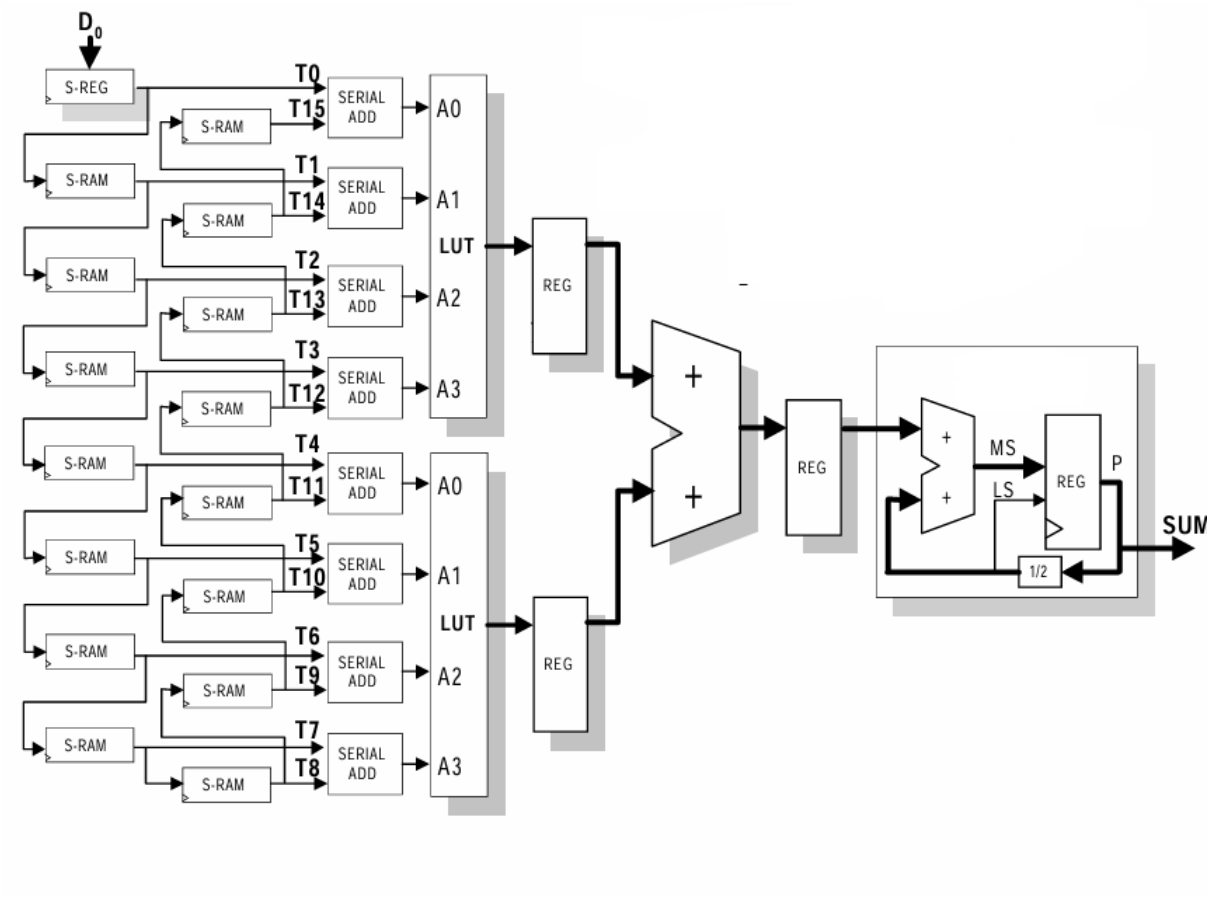
Với việc thực hiện tính toán tuần tự, việc cộng từng bit giá trị của mẫu và cộng dồn kết hợp dịch cho kết quả cuối cũng được thực hiện tuần tự. Để cộng tuần tự từng bit theo SDA, ta sẽ kết hợp thêm khối cộng tuần tự (Serial Add) để cộng lần lượt từng

bit của cặp hai mẫu. Do ở đây thực hiện cộng tuần tự từng bit nên để xử lý chống tràn hoặc tính toán sai, khi đưa vào giá trị X thì đưa vào giá trị đã được mở rộng bit dấu (MSB).

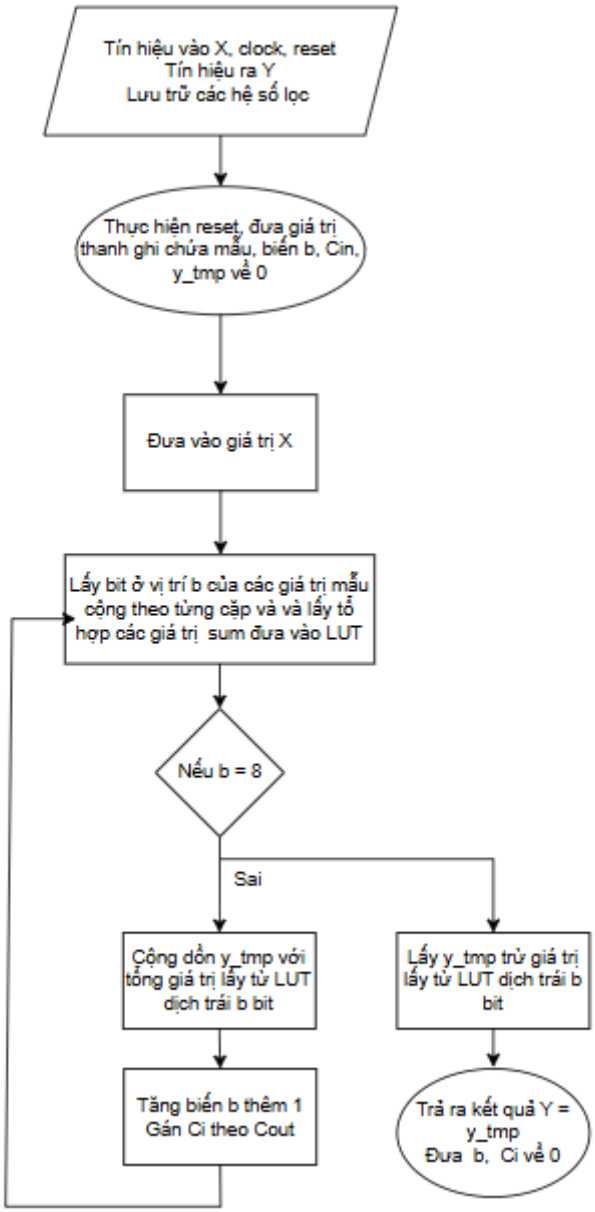


Hình 2.17: Khối cộng tuần tự

Sử dụng các khối cộng tuần tự ấy, khi này thì mô hình thuật toán của lọc FIR 16 hệ số lọc, các hệ số đối xứng sẽ như sau:



Hình 2.18: Mô hình SDA các hệ số lọc đối xứng

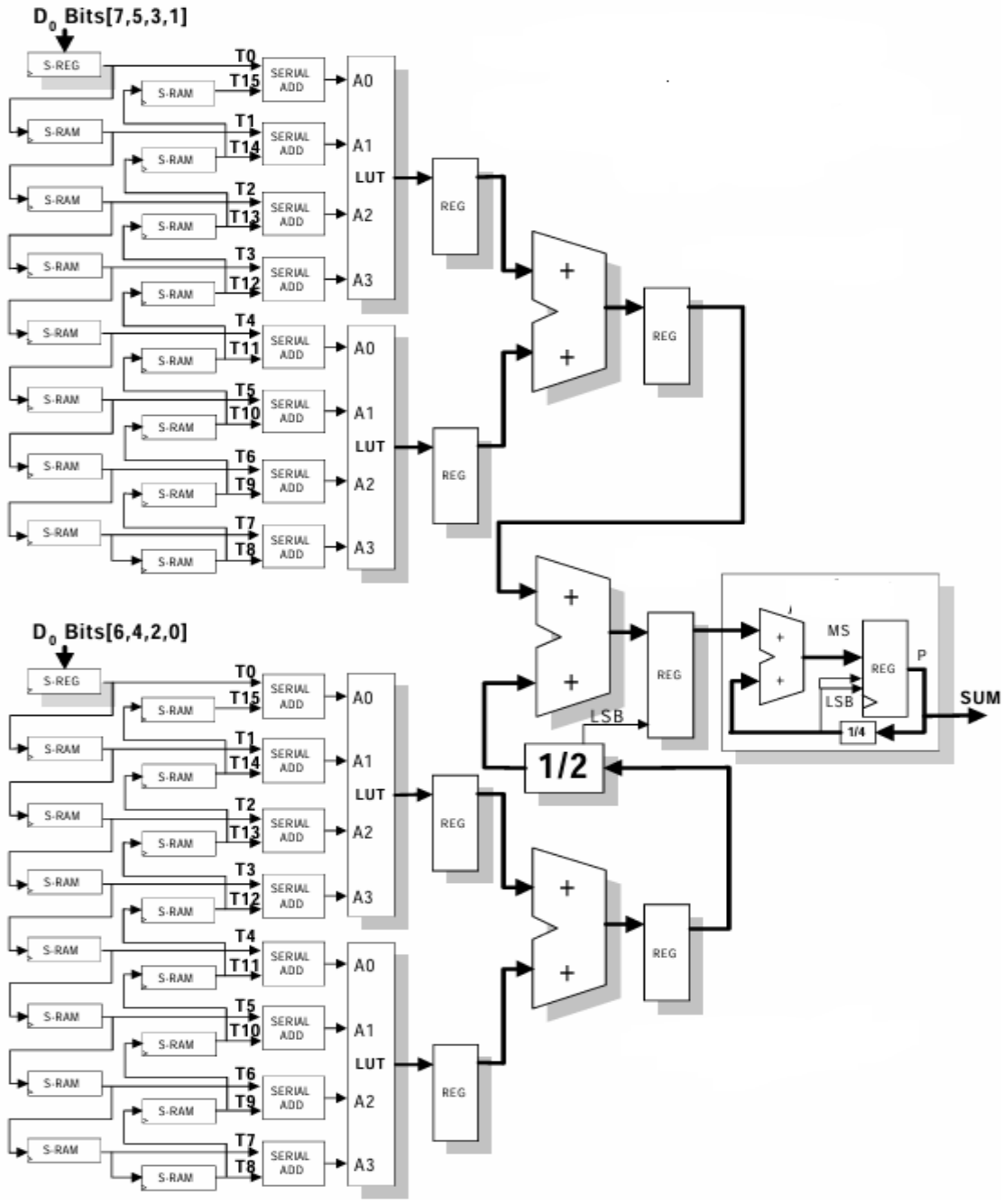


Hình 2.19: Lưu đồ giải thuật cho thuật toán SDA

Với PDA, số bit của dữ liệu xử lý trong cùng 1 xung đồng hồ sẽ tăng lên, tùy thuộc vào yêu cầu về tốc độ, tài nguyên. Xử lý càng nhiều bit cùng 1 lúc thì tốc độ càng cao nhưng cũng yêu cầu tài nguyên phần cứng lớn hơn. Với PDA, việc cộng giá trị các mẫu cũng sẽ được thực hiện từng bit dùng Serial Adder giống với SDA, chỉ khác ở đây thì các tín hiệu ra của khối Serial Add (S, Cout) sẽ được tính toán và cập nhật ngay lập tức khi mà các tín hiệu vào (A, B, Ci) thay đổi mà không phải chờ tín

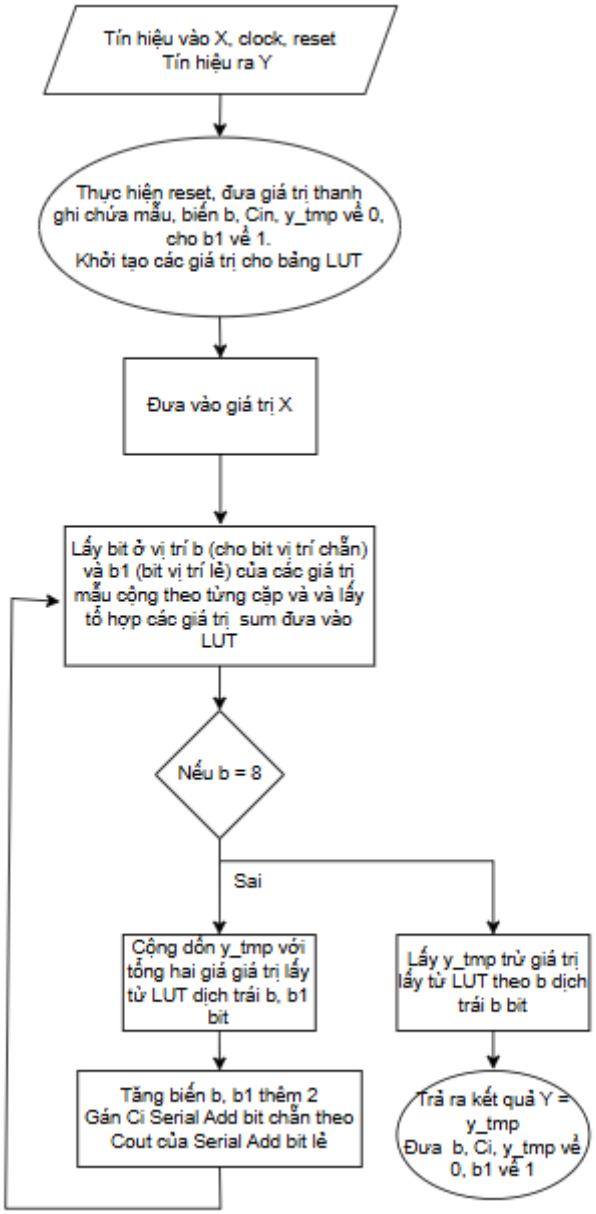
hiệu đồng hồ để đồng bộ hóa và tất nhiên các mẫu đầu vào sẽ được lưu vào với bit dấu mở rộng cho các giá trị mẫu.

Ở đây, ta sẽ bàn về thuật toán PDA 2 bit. Với PDA 2 bit thì trong cùng một xung đồng hồ hệ thống sẽ xử lý 2 bit, bắt đầu từ LSB. Và để chống tràn số, khi đưa giá trị mẫu vào thì đó cũng là giá trị đã được mở rộng bit dấu. Và ở đây xét 2 bit cùng lúc nên để xử lý tính toán cộng với Serial Add cho đúng thì các giá trị Cout của các bit chẵn sẽ là Cin cho Serial Adder của các bit lẻ.



Hình 2.20: Mô hình thuật toán PDA 2 bit các hệ số lọc đối xứng

Từ các mô tả trên, ta xây dựng lưu đồ giải thuật cho thuật toán dùng PDA như sau



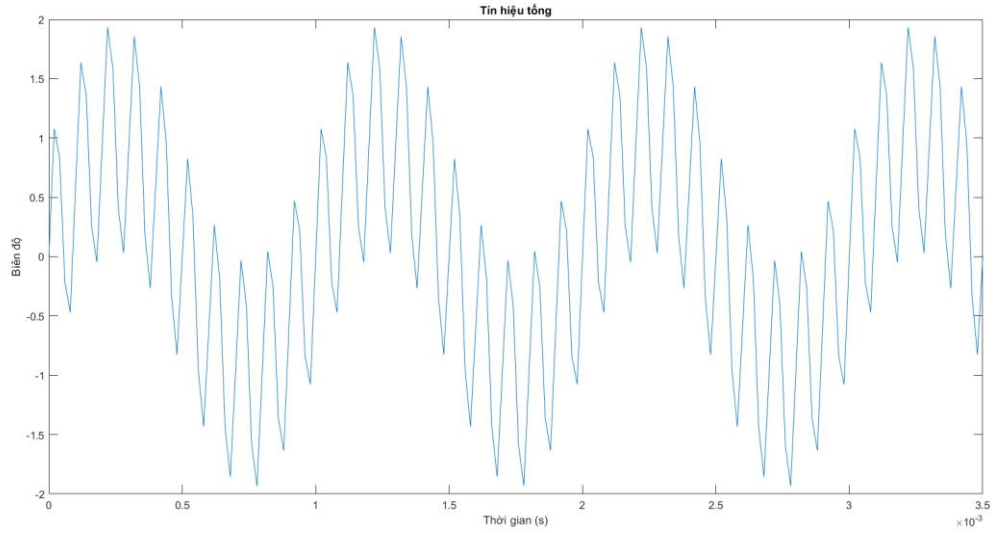
Hình 2.21: Lưu đồ giải thuật cho thuật toán sử dụng PDA

CHƯƠNG 3: THIẾT KẾ HỆ THỐNG TRÊN PHẦN MỀM

Chi tiết mô hình được sử dụng là một bộ lọc FIR thấp qua sử dụng phương pháp cửa sổ Hamming gồm 16 hệ số lọc đối xứng, với tín hiệu vào là tín hiệu tổng của 2 sóng sin tần số 1 kHz và 10 kHz, biên độ đều là 1 và pha ban đầu là 0, tần số lấy mẫu là 50 kHz và lấy 176 mẫu. Tần số cắt là 4.5 kHz để giữ lại tín hiệu có tần số 1 kHz.

3.1 Thiết kế trên MATLAB và C

Sau tạo tín hiệu, lấy mẫu vào tổng hợp thì kết quả vẽ tín hiệu tổng hợp trên MATLAB



Hình 3.1: Hình dạng tín hiệu tổng

Dựa trên lý thuyết nêu trên, ta xây dựng được hàm tạo các hệ số lọc lý tưởng theo biến đổi Fourier và hàm cửa sổ Hamming là

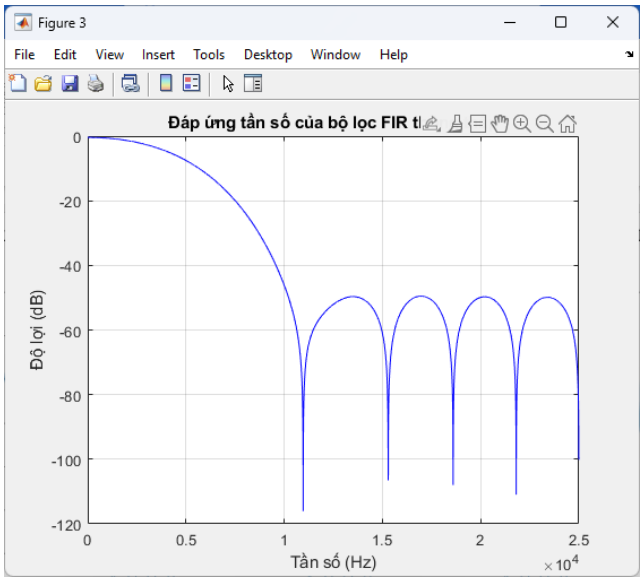
```
function h_ideal = Lowpass(f_c, f_s, N)
    wc = 2 * pi * f_c / f_s;
    n = 0:N-1;
    n_center = (N-1) / 2;
    h_ideal = wc / pi * sinc((n - n_center) * wc / pi);
end

function w_hamming = Hamming(N)
    n = (0:N);
    w_hamming = 0.54 - 0.46 * cos(2 * pi * n / N);
end
```

Vì trong bài đang xây dựng một bộ lọc FIR, là một hệ thống tuyến tính bất biến và nhân quả thì nó phải có giá trị là 0 khi n < 0, nên ở đây biến n từ 0 tới N - 1, và đã

được dịch sang phải n_center giá trị. Ngoài ra khi áp dụng hàm cửa sổ thì tham số vào cho hàm cửa sổ Hamming là N - 1.

Từ những gì đề cập tới việc xây dựng hàm tạo các hệ số lọc, ta có dạng đáp ứng tần số của nó là



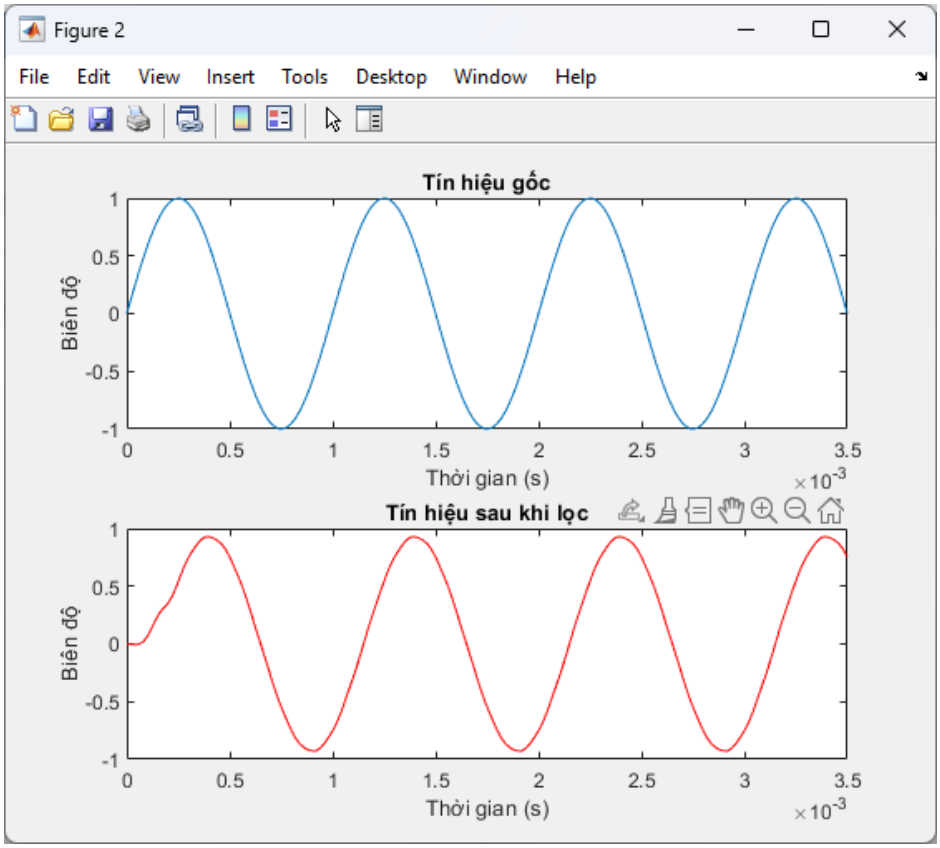
Hình 3.2: Hình dạng hàm đáp ứng tần số

Các giá trị hệ số lọc thu được, và kết quả chuyển sang fixed-point 8 bit

STT	Dạng số thực	Fixed-point
1	-0.00298562072450287	-1
2	-0.00298562072450287	-1
3	0.000422112232006720	0
4	0.0158183106704109	4
5	0.0490848376689463	13
6	0.0968324166902078	25
7	0.145194333029042	37
8	0.175825887678234	45
9	0.175825887678234	45
10	0.145194333029042	37
11	0.0968324166902078	25
12	0.0490848376689463	13
13	0.0158183106704109	4
14	0.000422112232006720	0
15	-0.00298562072450287	-1
16	-0.00298562072450287	-1

Bảng 3.1: Giá trị các hệ số lọc thu được

Để kiểm tra, ta sẽ in ra và so sánh kết quả giữa sóng ban đầu và sóng kết quả sau lọc



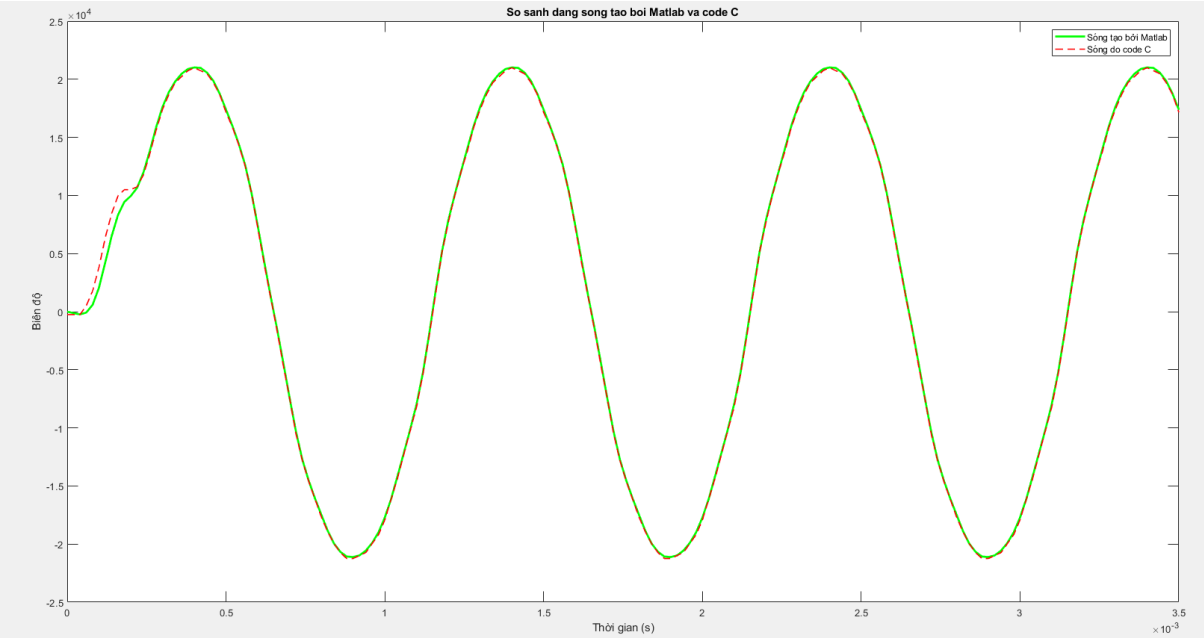
Hình 3.3: So sánh dạng sóng ban đầu và sau khi lọc

Ta thấy rằng, dạng sóng ban đầu và sau lọc đã khá tương đồng, chỉ là dạng sóng sau lọc có khoảng trễ hơn và vùng một số giá trị ban đầu hơi sai lệch. Điều này cho thấy rằng các hệ số lọc được tạo có độ tin cậy khá cao. Từ đó, ta lưu các giá trị này để kiểm tra với ngôn ngữ C.

Cần lưu ý là trong phần khởi tạo, ta tạo là tín hiệu sin, và các giá trị sẽ là dạng số thực và được hiểu là dạng floating-point, nên sẽ cần phải chuyển sang số fixed-point bằng hàm dưới (tương tự với MATLAB)

```
int8_t float_to_fixed_8bit(float x, int frac_bits) {
    float scale = (float)pow(2, frac_bits) ; //(1 << frac_bits);
    int16_t fixed_point = (int16_t)round(x * scale);
    if (fixed_point > INT8_MAX) {
        fixed_point = INT8_MAX;
    } else if (fixed_point < INT8_MIN) {
        fixed_point = INT8_MIN;
    }
    return (int8_t)fixed_point;
}
```

Từ các giá trị kết quả dạng số nguyên 16 bit (do các hệ số lọc và mẫu đều là 8 bit) tạo bởi mã C trên môi trường Visual Studio Code, dùng MATLAB đọc và so sánh với kết quả tạo bởi chính hàm MATLAB

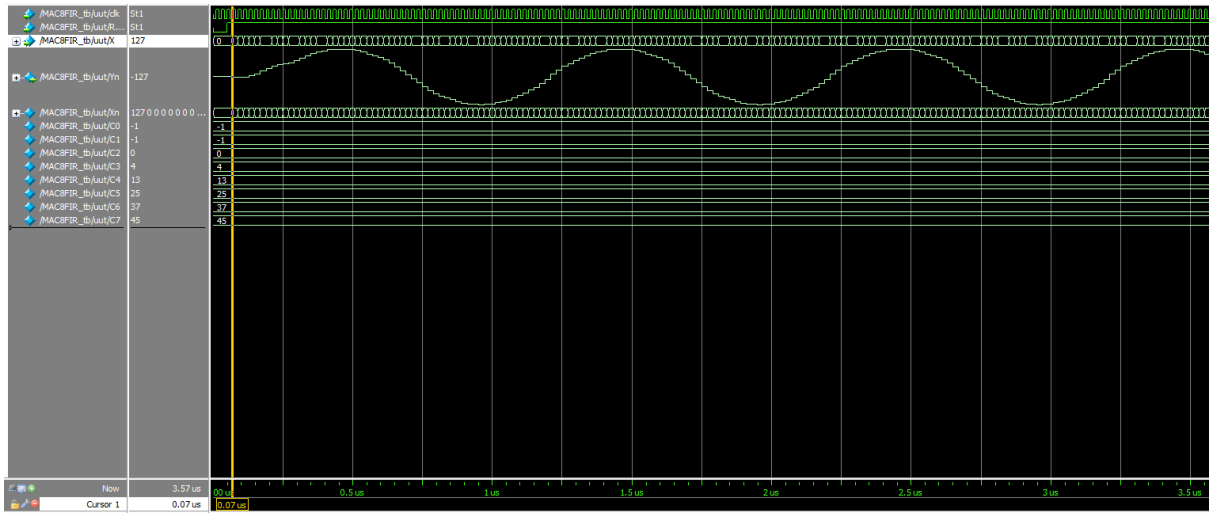


Hình 3.4: So sánh kết quả tạo bởi C và MATLAB dạng số nguyên 16 bit.

CHƯƠNG 4: MÔ PHỎNG VÀ THỰC HIỆN TRÊN FPGA.

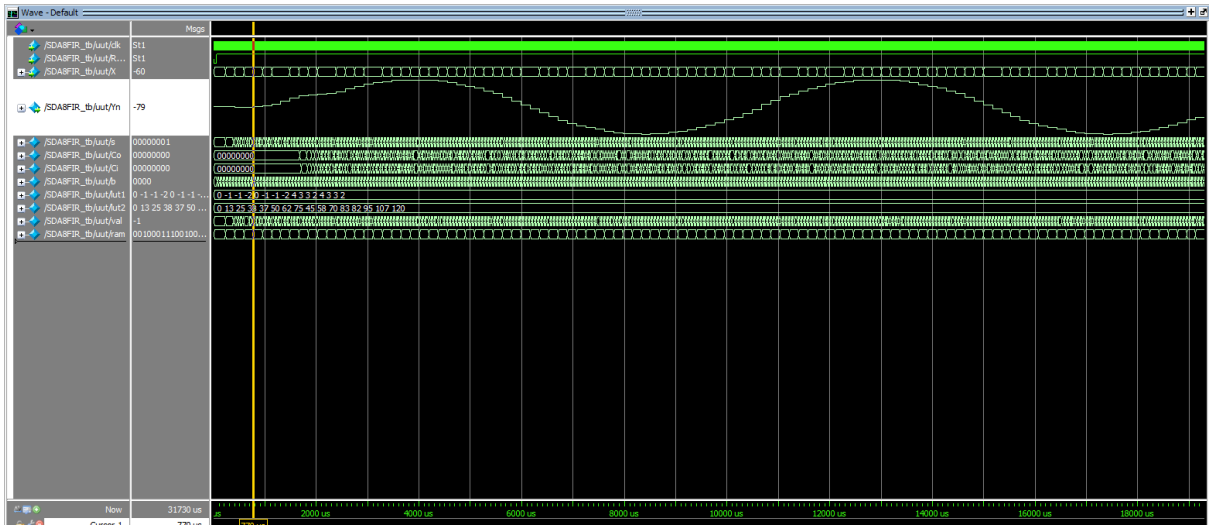
4.1 Mô phỏng ModelSim

- Với thuật toán dùng các phép nhân cộng tích lũy (MAC), reset tích cực mức thấp cho thấy rằng kết quả tính toán là chính xác (dạng analog của sóng ra Yn có hình sin), và Yn tính được ngay sau khi đưa vào Yn.



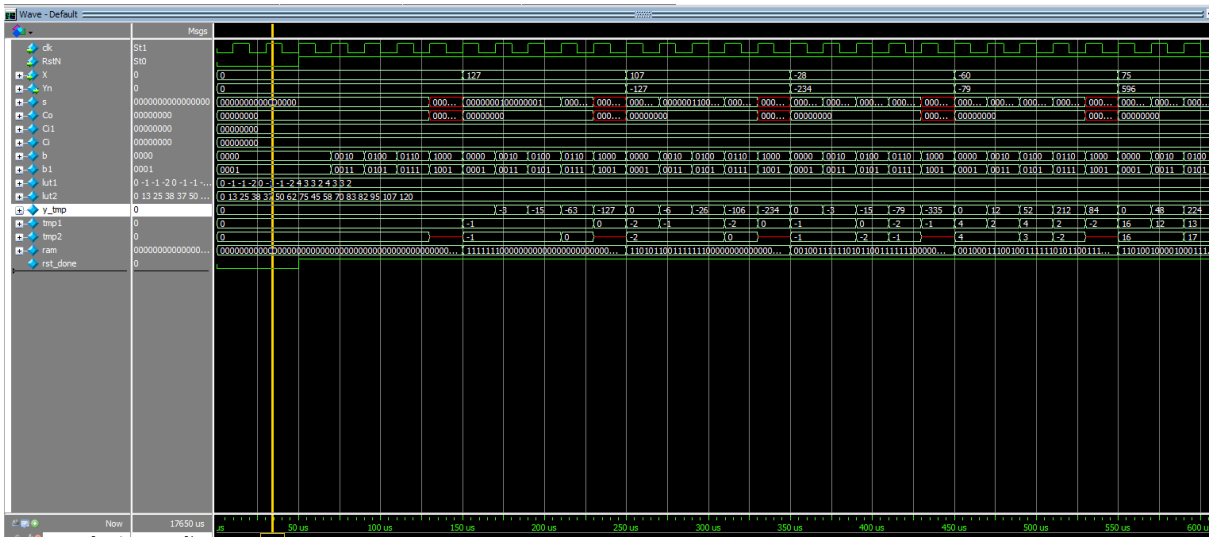
Hình 4.1: Kết quả mô phỏng của MAC

- Kết quả mô phỏng của thuật toán SDA, và giống với các mô tả lý thuyết, mỗi một mẫu cần có 9 xung đồng hồ để tính toán, và các kết quả ra cũng đã đảm bảo độ chính xác.



Hình 4.2: Kết quả mô phỏng của SDA

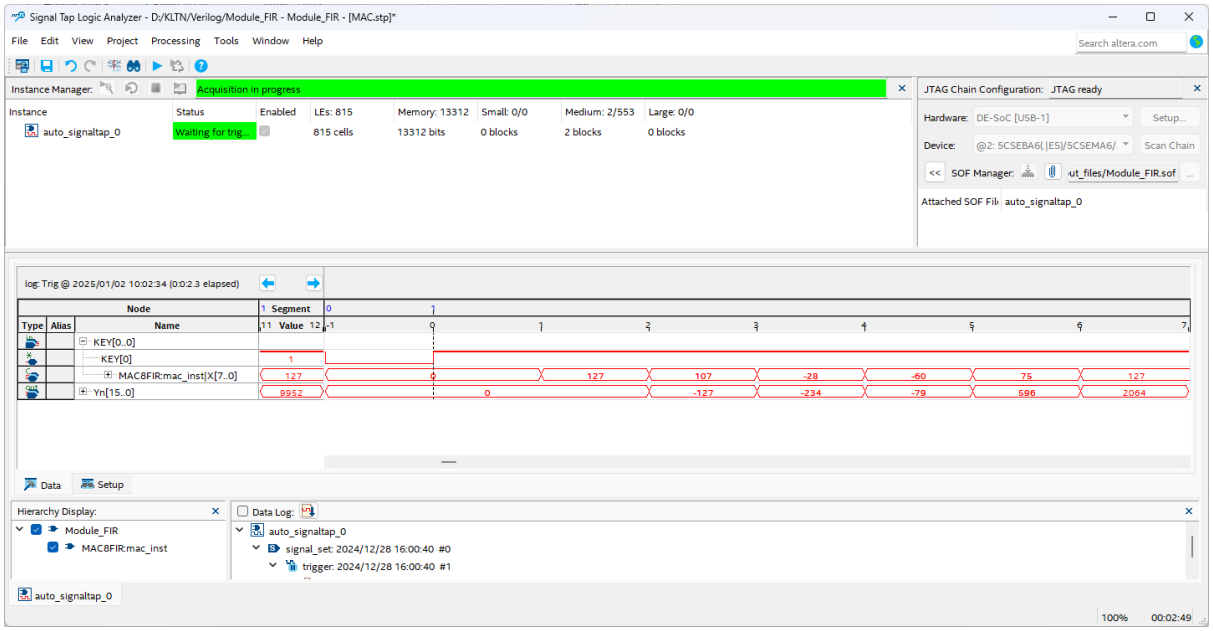
- Dạng sóng mô phỏng theo thuật toán PDA, với s là tín hiệu 16 bit là kết quả của 16 Serial Add, ram là thanh ghi chứa các mẫu. Một mẫu cũng cần 5 xung clock để tính đúng theo yêu cầu.



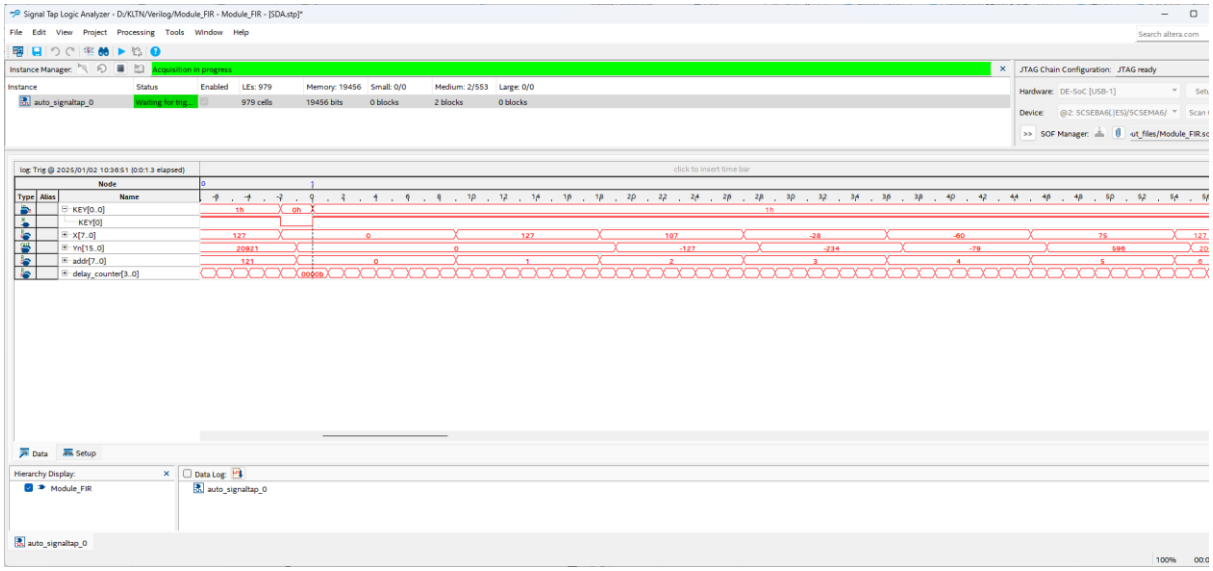
Hình 4.3: Kết quả mô phỏng với PDA

4.2 Chạy trên FPGA với Signal Tab Logic Analyzer

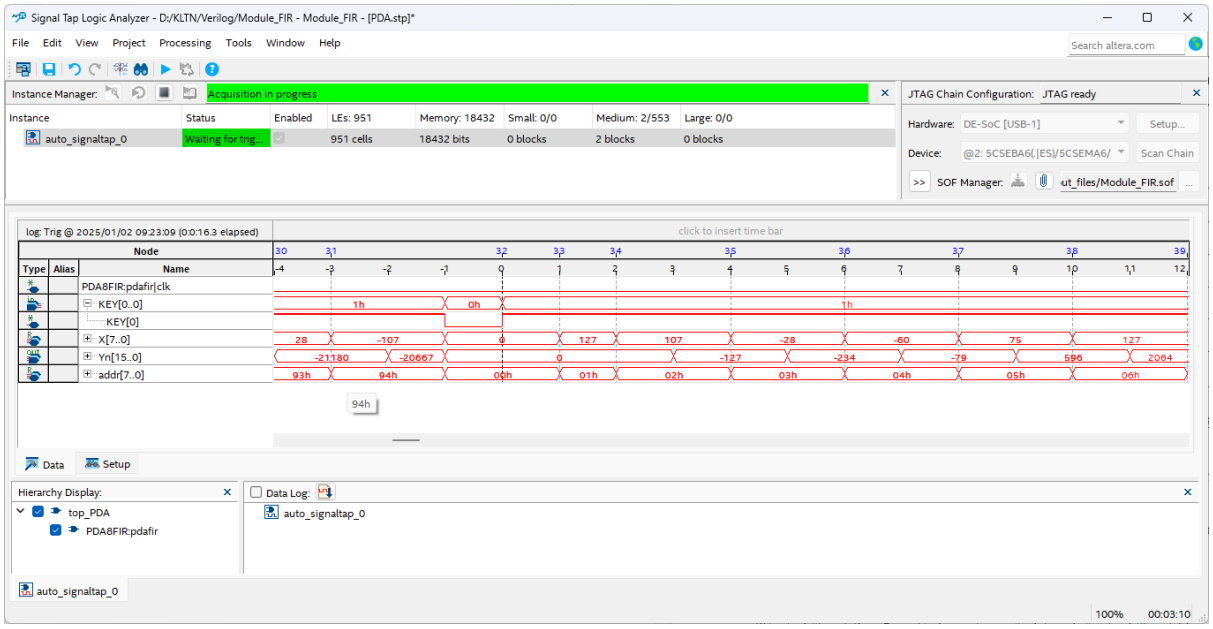
Trong tất cả các trường hợp, tín hiệu clk của các thiết kế gắn với tín hiệu xung CLOCK_50, tần số 50 MHz trên board, tín hiệu reset gắn với KEY[0], và đặt điều kiện Trigger là khi có xung lên của KEY[0] (tức là sau khi nhấn KEY[0] và thả).



Hình 4.4: Kết quả chạy Signal Tab của MAC



Hình 4.5: Kết quả chạy Signal Tab của SDA.



Hình 4.6: Kết quả chạy Signal Tab của PDA

CHƯƠNG 5: KẾT LUẬN VÀ ĐÁNH GIÁ

5.1 Kết luận

Trong đề tài, lọc FIR được xây dựng dựa trên mô hình MATLAB và được thực hiện và thiết kế trên FPGA bằng ngôn ngữ Verilog HDL và biên dịch, tổng hợp mạch với Quartus Prime Lite để thực hiện so sánh, đánh giá. Qua kết quả quá trình tổng hợp mạch và kiểm tra báo các tổng hợp của phần mềm Intel Quartus Prime, ta thấy được đặc điểm của từng loại thiết kế như sau

	MAC	PDA	SDA
f_{max} (MHz)	66.08	101.72	80.87
ALM	50	163	131
DSP blocks	3	0	0
Số thanh ghi	153	199	209
Số xung đồng hồ xử lý 1 mẫu	1	5	9

Bảng 5.1: *So sánh giữa MAC, PDA, SDA*

Từ bảng trên cho thấy các rằng các kiểu thiết kế đã đáp ứng dúng yêu cầu và kết quả của nó, ví dụ với PDA thì nhiều tài nguyên nhưng tần số tối đa cao, SDA thì tốn ít tài nguyên hơn nhưng tính toán chận hơn; riêng với phương pháp dùng nhân cộng tích lũy (MAC) thì mạch đã tự tổng hợp và sử dụng các DSP Block là các khối chuyên dụng trong các ứng dụng xử lý tín hiệu số trên nền tảng FPGA, và nếu phải so sánh thì các khối DSP Blocks chiếm lượng tài nguyên rất cao.

5.2 Đánh giá

Từ các kết quả thu được từ dạng sóng mô phỏng, kiểm tra kết quả khi chạy trực tiếp trên board bằng công cụ Signal Tab Logic Analyzer, kết quả cho thấy các thiết kế đều hoạt động đúng, và các kết quả tính toán phù hợp và chính xác, đồng thời các thiết kế cũng thể hiện đúng đặc điểm của mình.

TÀI LIỆU THAM KHẢO

[1] PGS TS.. Nguyễn Hữu Phương, *Xử lý tín hiệu số (Digital Signal Processing), Dùng cho sinh viên các ngành Điện tử, Tự động hóa, Viễn thông, Tin học*. Nhà xuất bản Thống Kê, 2003.

[2] Nguyễn Hữu Nhân (2015), Luận văn tốt nghiệp "Research and design a SOC based on DBNS FIR filter applied for DSP applications" Khoa Điện tử - Viễn Thông, Trường Đại học Khoa học Tự nhiên – ĐHQG TP.HCM, truy cập từ https://ir.vnulib.edu.vn/handle/VNUHCM/11778 [Truy cập lần cuối: 12/2024].

[3] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance," Digital Signal Processing Program Manager, Xilinx, Inc., San Jose, CA, 95124.

[4] S. S. Shah, S. Yaqub, and F. Suleman, "Distributed Arithmetic for the Design of High Speed FIR Filter using FPGAs," Chameleon Logics, #301, Kiran Plaza F-8 Markaz, Islamabad. Email: chameleon.logics@isb.comsats.net.pk.

[5] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 4th ed. Springer, 123. Truy cập từ: http://www.springer.com/series/4748. [Truy cập lần cuối: 12/2024].

[6] GeeksforGeeks, "Introduction to MATLAB," Truy cập từ: https://www.geeksforgeeks.org/introduction-to-matlab/. [Truy cập lần cuối: 12/2025].

[7] TutorialsPoint, "Fixed Point and Floating Point Number Representations," TutorialsPoint, Truy cập từ: https://www.tutorialspoint.com/fixed-point-and-floating-point-number-representations. [Truy cập lần cuối: 12/2024].

[8] **DE10-Standard Quick Start Guide**: "DE10-Standard Quick Start Guide," Terasic Technologies. Truy cập từ: https://www.terasic.com/. [Truy cập lần cuối: 12/2025].

[9] **DE10-Standard User Manual**:"DE10-Standard User Manual," Terasic Technologies, Jan. 7, 2022. Truy cập từ: https://www.terasic.com/. [Truy cập lần cuối: 12/2024].

[10] Altera Corporation, *Implementing FIR Filters and FFTs with 28-nm Variable-Precision DSP Architecture*, White Paper WP-01140-1.0, Sept. 2010.Truy cập từ: www.altera.com. [Truy cập lần cuối: 12/2024].

[11] S. C. Prasanna and S. P. Joy Vasantha Rani, "Area and Speed Efficient Implementation of Symmetric FIR Digital Filter through Reduced Parallel LUT Decomposed DA Approach," Scientific Research Publishing, vol. X, pp. XX-XX, Jun. 2016.

[12] S. Khan and Z. A. Jaffery, "Low Power FIR Filter implementation on FPGA using Parallel Distributed Arithmetic," Dept. of Electrical Engineering, Jamia Millia Islamia, N. Delhi, India, 2021. Email: shaheen.khan.2@gmail.com , zjaffery@jmi.ac.in.

[13] Intel Corporation, "Signal Tap Logic Analyzer Introduction," Intel, 2021. Truy cập từ:https://www.intel.com/content/www/us/en/docs/programmable/683819/21-3/logic-analyzer-introduction.html. [Truy cập lần cuối : 12/2024].

[14] Intel Corporation, "Signal Tap Logic Analyzer Tutorial," Intel. Truy cập từ: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Verilog/SignalTap.pdf. [Truy cập lần cuối : 12/2024].

[15] Intel Corporation, "Quartus Prime Documentation," *Intel*, Truy cập từ: https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/docs.html. [Truy cập lần cuối : 12/2024].

[16] V. K. Ingle and J. G. Proakis, Digital Signal Processing Using MATLAB®, Third Edition, Northeastern University, CENGAGE Learning, 2017.

[17] N. L. Trung, T. D. Tân, and H. H. Tuệ, Xử lý tín hiệu số, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, Nhà xuất bản Đại học Quốc gia Hà Nội, 2021.