

**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

**ꝏꝏꝏ**



****

**BÁO CÁO ĐỒ ÁN**

**NHÓM 7**

**BỘ TĂNG TỐC LỌC FIR SỬ DỤNG CHUẨN**

**GIAO THỨC WISHBONE**

**Môn học: Thiết kế hệ thống số với HDL – CE213.P23**

**GVHD: ThS. Tạ Trí Đức**

***TP.Hồ Chí Minh, ngày 12 tháng 6 năm 2025***

| **DANH SÁCH THÀNH VIÊN** | | |
| --- | --- | --- |
| **MSSV** | **Họ tên** | **Đánh giá** |
| 22520290 | Phan Tiến Dũng (C) | 100% |
| 22520275 | Trần Anh Đức | 100% |
| 22521384 | Huỳnh Công Thiên | 100% |
| 22520376 | Hồ Hồng Hắc | 100% |
| 22520389 | Phạm Ngọc Hải | 100% |
| 22521397 | Lương Đăng Thiều | 100% |
| 22521449 | Trương Gia Thuận | 100% |

# 

| PHÂN CÔNG | | | |
| --- | --- | --- | --- |
| **STT** | **Công việc** | **Mô tả** | **Phụ trách** |
| 1 | Nghiên cứu  lý thuyết | - Tìm hiểu FIR filter và giao thức Wishbone  - Chọn cấu trúc filter phù hợp (Direct Form)  - Xây dựng, thiết kế các hệ số cần thiết | Huỳnh Công Thiên |
| 2 | Thiết kế phần cứng  (cơ bản) | - Viết Verilog/VHDL cho bộ lọc FIR theo hệ số  - Thiết kế khối Wishbone slave cơ bản  - Tích hợp bộ lọc vào bus | Hồ Hồng Hắc  Phạm Ngọc Hải |
| 3 | Tối ưu hóa thiết kế | - Cải thiện hiệu năng/tài nguyên: giảm độ trễ (latency), tăng thông lượng (throughput)  - So sánh giữa bản tối ưu và chưa tối ưu | Phan Tiến Dũng  Lương Đăng Thiều |
| 4 | Mô phỏng, kiểm thử & triển khai FPGA | - Viết testbench cho FIR & Wishbone  - Chạy mô phỏng (ModelSim, Vivado)  - Synthesis + implementation trên FPGA | Trương Gia Thuận |
| 5 | Tổng hợp, báo cáo & đánh giá | - Tổng hợp kết quả, viết báo cáo đồ án  - Làm slide, bảng so sánh, nhận xét bản tối ưu | Trần Anh Đức |

# 

# LỜI MỞ ĐẦU

Trong thời đại công nghệ số hiện nay, các hệ thống xử lý tín hiệu số (DSP - Digital Signal Processing) đóng vai trò không thể thiếu trong nhiều lĩnh vực như viễn thông, y sinh, âm thanh và hình ảnh. Một trong những khối cơ bản và quan trọng trong hệ thống DSP là bộ lọc số, đặc biệt là bộ lọc đáp ứng xung hữu hạn (FIR - Finite Impulse Response). Để đáp ứng yêu cầu xử lý nhanh và hiệu quả trong thời gian thực, việc thiết kế các bộ tăng tốc (accelerators) cho bộ lọc FIR trở nên rất cần thiết.

Đồ án này tập trung vào việc thiết kế một bộ tăng tốc cho bộ lọc FIR, sử dụng ngôn ngữ mô tả phần cứng HDL (Hardware Description Language) và tích hợp theo chuẩn giao thức Wishbone – một chuẩn giao tiếp mở thường được dùng trong thiết kế các hệ thống trên chip (SoC - System on Chip). Thông qua đồ án, nhóm 7 không chỉ rèn luyện kỹ năng thiết kế và mô phỏng mạch số bằng HDL mà còn hiểu rõ hơn về cách tích hợp các thành phần phần cứng trong một hệ thống tổng thể.

**MỤC LỤC**

[**LỜI MỞ ĐẦU 4**](#_xsavc7d2eraz)

[**CHƯƠNG 1 – GIỚI THIỆU 7**](#_bbwjpzw6gjz3)

[1.1. Giới thiệu tổng quan 7](#_8r84eoxjwhjs)

[1.2. Mục tiêu 7](#_xdvwad5qx76s)

[1.3. Phạm vi thực hiện 7](#_4cl6brgi7vgn)

[**CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT 8**](#_dpaomubz0bci)

[2.1. Bộ lọc số FIR 8](#_gi6if0zfxues)

[2.1.1. Khái niệm 8](#_ak8ymuk1y9ug)

[2.1.2. Nguyên lý hoạt động 8](#_fhhf4j3q5u10)

[2.1.3. Cấu trúc và các loại bộ lọc FIR 9](#_bknzmee4rby8)

[2.2. Giao thức Wishbone 10](#_wpm2xt5gs41p)

[2.2.2. Khái niệm Wishbone 10](#_2fpkda8hf1pj)

[2.2.3. Kiến trúc kết nối trong Wishbone 10](#_m1qv5qptnax5)

[2.2.4. Giao tiếp giữa IP core (FIR) và Wishbone 12](#_d9qwbb46l7iq)

[**CHƯƠNG 3 – THIẾT KẾ BỘ LỌC FIR VÀ GIAO TIẾP WISHBONE 13**](#_uuteqikttvv9)

[3.1. Yêu cầu thiết kế hệ thống 13](#_mmol548flxz3)

[3.2. Kiến trúc tổng thể hệ thống 13](#_t7rnbpanf7hn)

[3.3. Mô tả thiết kế RTL bằng Verilog HDL 15](#_imzqi2l7vkcn)

[3.3.1. Thiết kế khối FIR 15](#_9vji4qlxzjt)

[3.3.2. Thiết kế giao tiếp Wishbone 15](#_n8yyexgouhz5)

[3.3.3. Kết nối FIR với Wishbone 16](#_rse7ojikynek)

[3.4. Mô phỏng và kiểm thử 17](#_70gznpmc9hhh)

[3.4.1. Cấu trúc testbench 17](#_flzyafjkoy5w)

[3.4.2. Quy trình kiểm thử 17](#_np40xfu5wc07)

[3.4.3. Kết quả mô phỏng 18](#_7t8xrhl5khxv)

[**CHƯƠNG 4 – TỐI ƯU THIẾT KẾ 19**](#_fadlpobelzx2)

[4.1. Mục tiêu tối ưu hóa 19](#_v6vjdwo1k7iq)

[4.2. Các phương pháp tối ưu được áp dụng 19](#_x625oxou5nv5)

[4.2.1. Nhân song song (Parallel Multiply) 19](#_qyqtqbkl8c7o)

[4.2.2. Cây cộng (Adder Tree) 19](#_pchgp0cleab1)

[4.2.3. Pipeline hóa (3 giai đoạn) 20](#_mrbbkeq5bdhc)

[4.2.4. Mô hình kiến trúc sau tối ưu 20](#_6ov9tre5i1ow)

[4.3. Mô phỏng và kiểm thử 21](#_6lkgs28drqfu)

[4.4. Đánh giá hiệu quả tối ưu 22](#_rg9tuwe7q96g)

[**CHƯƠNG 5 – TỔNG HỢP, TRIỂN KHAI VÀ KIỂM THỬ TRÊN FPGA 24**](#_1yb4nhpnonyz)

[5.1. Tổng hợp thiết kế bằng Quartus 24](#_ed4egmzhn96n)

[5.1.1. Phân tích kết quả 24](#_dzpnhdw2uc3x)

[5.1.2. Phân tích timing 28](#_y7y8wbc797e4)

[5.2. Nạp và kiểm thử trên FPGA 29](#_xiif23sfgc9g)

[**CHƯƠNG 6 – KẾT LUẬN 31**](#_m8gdrujpwppx)

[**PHỤ LỤC 32**](#_fwuiex9m9xlp)

[**TÀI LIỆU THAM KHẢO 36**](#_wac5v7tuqegy)

# 

# CHƯƠNG 1 – GIỚI THIỆU

## 1.1. Giới thiệu tổng quan

Trong xử lý tín hiệu số (DSP), bộ lọc FIR (Finite Impulse Response) là thành phần quan trọng nhờ tính ổn định và khả năng điều chỉnh đặc tuyến tần số. Tuy nhiên, khi yêu cầu xử lý thời gian thực tăng cao, việc sử dụng vi xử lý truyền thống không còn đáp ứng hiệu suất cần thiết.

Giải pháp đặt ra là thiết kế bộ tăng tốc phần cứng (hardware accelerator) giúp tăng tốc quá trình lọc. Trong các hệ thống tích hợp (SoC), việc kết nối các thành phần xử lý qua chuẩn giao tiếp Wishbone – một chuẩn bus mở, đơn giản – là hướng tiếp cận hiệu quả và linh hoạt.

Đồ án này nhằm thiết kế bộ tăng tốc cho bộ lọc FIR sử dụng HDL, tích hợp giao diện Wishbone để dễ dàng mở rộng và tái sử dụng trong các hệ thống phần cứng nhúng.

## 1.2. Mục tiêu

Mục tiêu của đồ án là nghiên cứu nguyên lý hoạt động và phương pháp thiết kế bộ lọc FIR, đồng thời tìm hiểu giao thức truyền thông Wishbone trong hệ thống nhúng. Trên cơ sở đó, tiến hành thiết kế bộ tăng tốc phần cứng cho bộ lọc FIR bằng ngôn ngữ mô tả phần cứng (HDL), tích hợp giao diện Wishbone nhằm kết nối linh hoạt với hệ thống. Cuối cùng, thực hiện mô phỏng, kiểm thử và đánh giá hiệu năng để đảm bảo hệ thống hoạt động chính xác, có khả năng mở rộng và tích hợp hiệu quả trong kiến trúc SoC.

## 1.3. Phạm vi thực hiện

Phạm vi thực hiện của đồ án bao gồm thiết kế bộ lọc FIR với hệ số cố định được xác định trước, tập trung vào kiến trúc song song hoặc pipeline nhằm tăng tốc độ xử lý. Hệ thống được cài đặt và mô phỏng hoàn toàn trong môi trường phần mềm như ModelSim, Vivado hoặc Quartus. Giao tiếp Wishbone được triển khai theo chuẩn cơ bản, hỗ trợ truyền nhận dữ liệu giữa bộ lọc và master. Nếu điều kiện cho phép, hệ thống sẽ được tổng hợp và kiểm thử trên FPGA thực tế.

# CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT

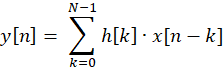
## 2.1. Bộ lọc số FIR

### 2.1.1. Khái niệm

Bộ lọc FIR (Finite Impulse Response) là một bộ lọc số tuyến tính có đáp ứng xung hữu hạn, nghĩa là đầu ra sẽ trở về 0 sau một số hữu hạn mẫu khi đầu vào là một xung đơn vị. Nhờ không sử dụng hồi tiếp nên bộ lọc FIR luôn ổn định, đồng thời có thể dễ dàng thiết kế để đạt được đáp ứng pha tuyến tính. Với cấu trúc đơn giản và chỉ bao gồm các phép nhân và cộng, FIR rất phù hợp để thực thi hiệu quả trên phần cứng số.

### 2.1.2. Nguyên lý hoạt động

Phương trình tổng quát của bộ lọc FIR bậc N được biểu diễn như sau:



Trong đó:

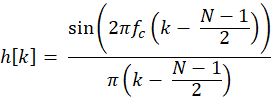
N : Số lượng taps

x[n] : Tín hiệu vào

y[n] : Tín hiệu ra

h[k] : Hệ số lọc (filter coefficients) (\*)

(\*) Hệ số lọc h[k] có thể được tính như sau:



Trong đó:

 : tần số cắt đã chuẩn hóa (cutoff frequency / sampling frequency)

Quá trình hoạt động của bộ lọc (theo từng mẫu) như sau:

(1) Khi tín hiệu x[n] đi vào, bộ lọc sẽ lấy các mẫu đầu vào gần nhất: x[n],x[n−1],...,x[n−(N−1)]

(2) Mỗi mẫu được nhân với một hệ số h[k]

(3) Các tích được cộng lại để cho ra đầu ra y[n]

\*Ví dụ minh họa:

Giả sử bộ lọc FIR bậc 2 với hệ số:

h = [0.25, 0.5, 0.25]

Đầu ra tại thời điểm n:

y[n] = (0.25 ⋅ x[n]) + (0.5 ⋅ x[n−1]) + (0.25 ⋅ x[n−2])

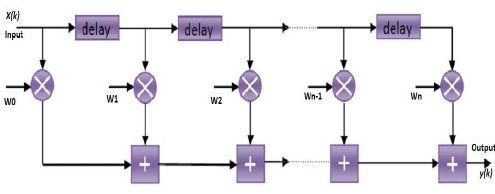
→ Là bộ lặp thông thấp đơn giản, làm mượt tín hiệu đầu vào

### 2.1.3. Cấu trúc và các loại bộ lọc FIR

Theo cấu trúc Direct Form, bộ lọc FIR gồm 3 thành phần chính:  
 + *Thanh ghi trễ (Delay elements)*: giữ các giá trị mẫu trước đó.

+ *Hệ số nhân (Multipliers)*: nhân mỗi mẫu với một hệ số h[k].

+ *Bộ cộng (Adder)*: cộng kết quả các tích lại với nhau để tạo ra đầu ra.



Bộ lọc FIR khá đa dạng, có nhiều ứng dụng khác nhau, cụ thể có một số loại như sau:  
 - Phân loại theo đáp ứng tần số:

* Bộ lọc thông thấp (Low-pass filter): Cho tín hiệu tần số thấp đi qua, chặn tín hiệu cao.
* Bộ lọc thông cao (High-pass filter): Cho tín hiệu tần số cao đi qua, chặn tín hiệu thấp.
* Thông dải (Band-pass filter): Cho một dải tần số cụ thể đi qua.
* Chặn dải (Band-stop filter): Loại bỏ một dải tần số cụ thể.

- Phân loại theo đáp ứng pha:

* Pha tuyến tính (Linear Phase FIR filter): không gây méo pha tín hiệu.
* Pha không tuyến tính (Nonlinear FIR filter): dùng khi độ trễ không phải vấn đề.

## 2.2. Giao thức Wishbone

**2.2.1. IP (Intellectual Property) core**

IP (Intellectual Property) core là một khối thiết kế phần cứng tái sử dụng được, được viết bằng HDL (Verilog, VHDL...) và có thể tích hợp vào các hệ thống số lớn hơn như SoC (System-on-Chip) hoặc FPGA.

### 2.2.2. Khái niệm Wishbone

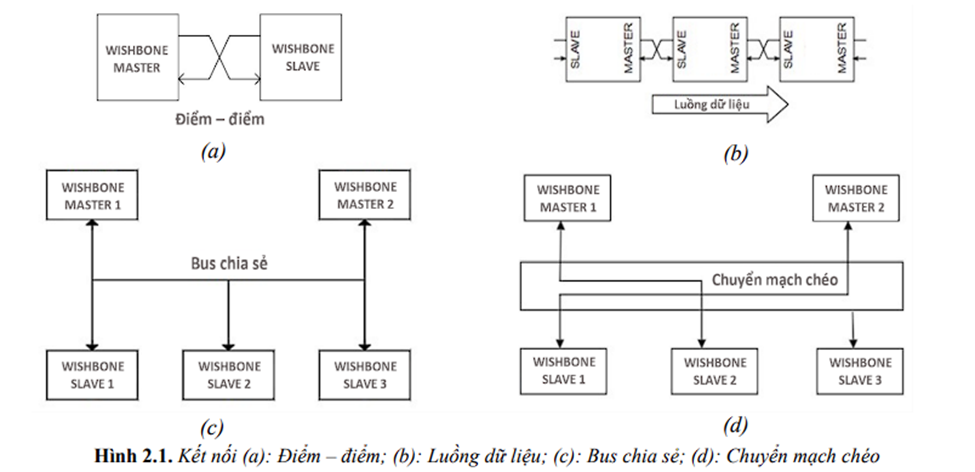
Wishbone là một chuẩn giao tiếp bus hệ thống mở dùng để kết nối các IP core trong thiết kế hệ thống trên chip (SoC) hoặc FPGA. Wishbone giúp chuẩn hóa giao diện giữa các khối phần cứng (như CPU, bộ lọc FIR, bộ nhớ, thiết bị ngoại vi...) thông qua một giao thức chung, nhằm đơn giản hóa việc tích hợp và tái sử dụng thiết kế.

### 2.2.3. Kiến trúc kết nối trong Wishbone

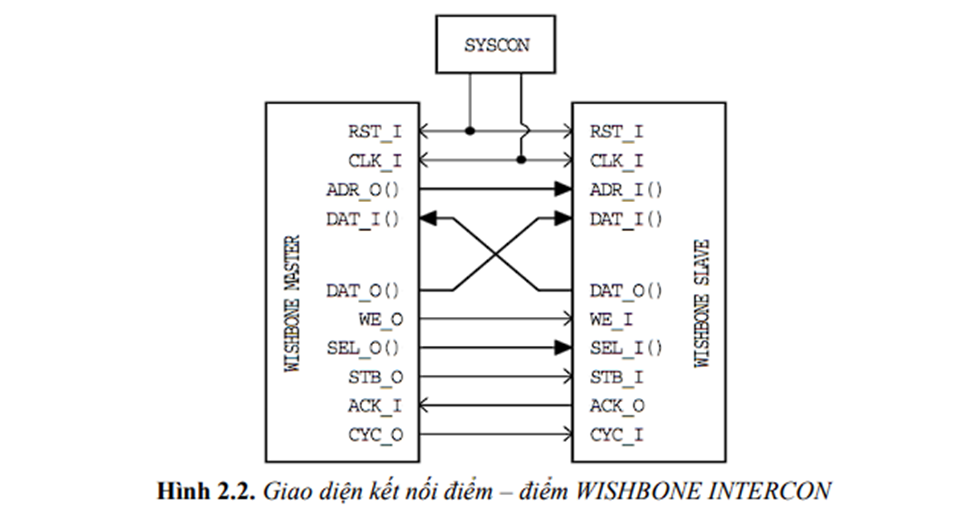
Wishbone hoạt động theo kiến trúc Master/Slave, trong đó:

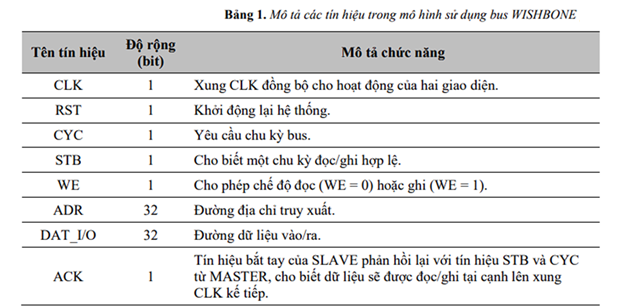
* *Master (chủ động)*: như CPU hoặc bộ điều khiển, khởi tạo giao dịch đọc/ghi.
* *Slave (bị động)*: như IP core FIR, chờ lệnh từ master và phản hồi yêu cầu.

Wishbone hỗ trợ nhiều kiểu kết nối linh hoạt như point-to-point, data flow, shared bus và crossbar, giúp phù hợp với nhiều cấu trúc SoC khác nhau.



Giao thức Wishbone sử dụng các tín hiệu tiêu chuẩn như CLK\_I, RST\_I, DAT\_I, DAT\_O, ADR\_I, WE\_I, STB\_I, CYC\_I, ACK\_O,... và hoạt động đồng bộ theo cạnh lên của xung clock với tín hiệu tích cực mức cao, đảm bảo tính nhất quán và dễ tích hợp.





### 2.2.4. Giao tiếp giữa IP core (FIR) và Wishbone

Trong hệ thống SoC hoặc FPGA, giao diện Wishbone đóng vai trò là cửa giao tiếp chính giữa IP core bộ lọc FIR và các thành phần còn lại của hệ thống. Thông qua bus Wishbone, CPU có thể tương tác trực tiếp với FIR IP core để thực hiện các chức năng như: ghi các hệ số lọc (coefficients), gửi dữ liệu đầu vào và đọc kết quả đầu ra sau khi xử lý. Giao diện này giúp việc truyền dữ liệu giữa CPU và bộ lọc diễn ra theo một chuẩn thống nhất, đảm bảo tính tương thích và mở rộng trong thiết kế hệ thống.

Wishbone sử dụng tập hợp các chân tín hiệu chuẩn để điều phối hoạt động giao tiếp, bao gồm: CLK\_I (xung clock đầu vào), RST\_I (tín hiệu reset), DAT\_I/O (dữ liệu vào và ra), ADR\_I (địa chỉ), WE\_I (ghi dữ liệu), STB\_I (bắt đầu giao dịch), CYC\_I (chu kỳ bus), và ACK\_O (phản hồi từ slave). Những tín hiệu này hoạt động đồng bộ với xung clock và tuân theo logic mức cao, tạo điều kiện thuận lợi cho việc tích hợp IP core vào các thiết kế phần cứng phức tạp.

# 

# 

# CHƯƠNG 3 – THIẾT KẾ BỘ LỌC FIR VÀ GIAO TIẾP WISHBONE

## 3.1. Yêu cầu thiết kế hệ thống

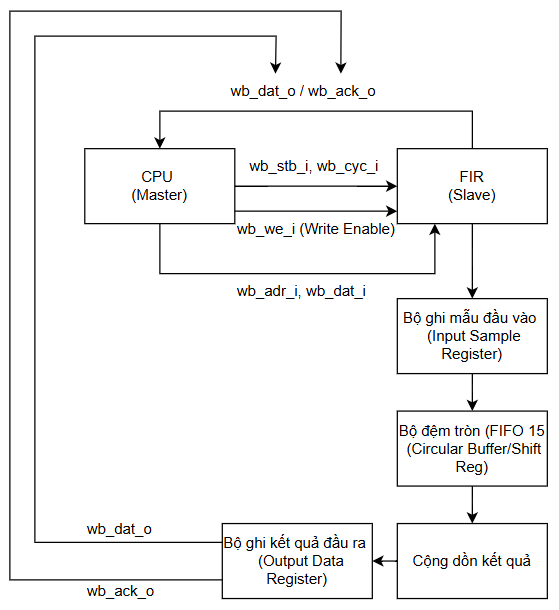
Trong các hệ thống xử lý tín hiệu số hiện đại, việc sử dụng bộ lọc Finite Impulse Response (FIR) là một phương pháp phổ biến để loại bỏ nhiễu, trích xuất đặc trưng và xử lý tín hiệu đầu vào. Nhằm nâng cao hiệu năng và tối ưu hóa tài nguyên phần cứng, đề tài hướng tới thiết kế một bộ lọc FIR ở mức phần cứng, có thể hoạt động như một IP core độc lập và dễ dàng tích hợp vào các hệ thống lớn hơn như SoC hoặc FPGA.

Yêu cầu đặt ra cho hệ thống bao gồm:

1. Thiết kế bộ lọc FIR có khả năng hoạt động theo thuật toán lọc tuyến tính tuyến tính thời gian bất biến
2. Giao tiếp được với CPU hoặc các thành phần khác thông qua chuẩn giao tiếp Wishbone, tuân thủ mô hình Master-Slave.
3. Thiết kế phải được mô tả bằng ngôn ngữ Verilog HDL để thuận tiện cho việc mô phỏng, tổng hợp và triển khai trên FPGA.
4. Cho phép CPU nạp hệ số lọc (coefficients), gửi mẫu đầu vào (input samples), và nhận kết quả đầu ra (output data) thông qua bus chuẩn Wishbone.

## 3.2. Kiến trúc tổng thể hệ thống

Kiến trúc tổng thể của hệ thống được trình bày trong Hình 3.1. Trong đó, CPU đóng vai trò là khối Master, khởi tạo các giao dịch đọc/ghi, còn FIR filter đóng vai trò là Slave, thực hiện chức năng lọc tín hiệu theo các yêu cầu từ CPU.



Hình 3.1 – Sơ đồ khối hệ thống kết nối giữa CPU và bộ lọc FIR thông qua giao tiếp Wishbone

Hệ thống gồm các khối chức năng chính:

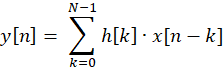
1. Bộ ghi mẫu đầu vào (Input Sample Register): lưu mẫu tín hiệu mới nhận từ CPU.
2. Bộ đệm tròn (FIFO hoặc Circular Buffer): lưu trữ liên tục 15 mẫu gần nhất phục vụ tính toán tích chập.
3. Khối nhân và cộng dồn (MAC – Multiply and Accumulate): thực hiện phép nhân từng hệ số với mẫu tương ứng và cộng dồn kết quả.
4. Bộ ghi kết quả đầu ra (Output Register): lưu trữ kết quả lọc để CPU đọc lại.

Giao tiếp giữa CPU và FIR filter được thực hiện thông qua các tín hiệu chuẩn của giao thức Wishbone như *wb\_adr\_i, wb\_dat\_i, wb\_dat\_o, wb\_we\_i, wb\_stb\_i, wb\_cyc\_i* và *wb\_ack\_o*. Các tín hiệu này đóng vai trò điều phối các chu kỳ đọc/ghi dữ liệu và kiểm soát quá trình truyền nhận trong hệ thống.

## 3.3. Mô tả thiết kế RTL bằng Verilog HDL

### 3.3.1. Thiết kế khối FIR

Bộ lọc FIR được thiết kế dựa trên công thức lọc tuyến tính:



Trong đó:

* h[k]: hệ số lọc, được nạp từ CPU.
* x[n−k]: các mẫu tín hiệu đầu vào, được lưu trong bộ đệm FIFO.
* y[n]: kết quả đầu ra sau khi thực hiện lọc.

Khối FIR bao gồm các thành phần:

* Bộ ghi mẫu đầu vào (Input Sample Register): để nhận mẫu dữ liệu từ CPU thông qua Wishbone.
* Bộ đệm tròn (FIFO hoặc Circular Buffer): kích thước 15 phần tử (theo số hệ số được chọn), hoạt động như một FIFO vòng, hỗ trợ việc dịch chuyển mẫu theo thời gian thực.
* Khối nhân và cộng dồn (MAC – Multiply and Accumulate): thực hiện từng phép nhân và cộng dồn liên tục trong nhiều chu kỳ clock.
* Bộ ghi kết quả đầu ra (Output Register): lưu trữ giá trị kết quả và sẵn sàng gửi về CPU thông qua bus Wishbone.

Tất cả các thành phần được mô tả bằng Verilog HDL theo phương pháp mô tả hành vi tuần tự và song song để đảm bảo tính đồng bộ và khả năng tổng hợp.

\*Mã nguồn: Phụ lục 1

### 3.3.2. Thiết kế giao tiếp Wishbone

Giao thức Wishbone được sử dụng làm chuẩn bus kết nối giữa CPU và bộ lọc FIR. Trong thiết kế này:

* CPU đóng vai trò Master, khởi tạo các giao dịch đọc/ghi.
* FIR filter là một Slave, phản hồi lại các yêu cầu tương ứng.

Các tín hiệu chuẩn của Wishbone được tích hợp trong thiết kế như sau:

* CLK\_I: tín hiệu clock hệ thống.
* RST\_I: tín hiệu reset bất đồng bộ.
* ADR\_I: địa chỉ truy cập (dùng để xác định vùng ghi hệ số, mẫu đầu vào, hoặc vùng đọc kết quả).
* DAT\_I: dữ liệu từ CPU ghi xuống FIR.
* DAT\_O: dữ liệu từ FIR gửi ngược về CPU.
* WE\_I: tín hiệu ghi (1: ghi, 0: đọc).
* STB\_I, CYC\_I: điều khiển bắt đầu giao dịch.
* ACK\_O: phản hồi xác nhận giao dịch hoàn tất.

Phân vùng địa chỉ:

* Địa chỉ 0x00: ghi hệ số lọc (coefficients).
* Địa chỉ 0x01: ghi mẫu tín hiệu đầu vào.
* Địa chỉ 0x02: đọc kết quả lọc đầu ra.

Mỗi giao dịch chỉ được xử lý sau khi FIR nhận đầy đủ tín hiệu khởi tạo (STB\_I và CYC\_I ở mức cao), và phản hồi bằng ACK\_O để thông báo đã xử lý xong lệnh.

\*Mã nguồn: Phụ lục 2

### 3.3.3. Kết nối FIR với Wishbone

Toàn bộ khối FIR cùng với logic giao tiếp Wishbone được đóng gói thành một IP Core độc lập.

IP Core được thiết kế với khả năng:

* Tái sử dụng cao, cấu hình linh hoạt số lượng hệ số.
* Đáp ứng đầy đủ chuẩn giao tiếp Wishbone, tương thích với các SoC hoặc CPU mềm
* Cho phép mở rộng thêm các chức năng như ngắt, DMA hoặc cấu hình động.

Việc kết nối IP Core FIR với các khối khác trong hệ thống giúp hiện thực hóa một kiến trúc lọc tín hiệu số có hiệu năng cao, phù hợp triển khai thực tế trên FPGA.

\*Mã nguồn: Phụ lục 3

## 3.4. Mô phỏng và kiểm thử

Để kiểm chứng chức năng của khối IP Core FIR đã thiết kế, một mô-đun testbench độc lập được xây dựng bằng ngôn ngữ Verilog HDL. Mô-đun này có vai trò mô phỏng hoạt động của hệ thống xử lý tín hiệu thông qua giao tiếp Wishbone, bao gồm các thao tác ghi hệ số lọc, gửi mẫu tín hiệu đầu vào và đọc kết quả đầu ra.

### 3.4.1. Cấu trúc testbench

Testbench khởi tạo các tín hiệu điều khiển và dữ liệu, mô phỏng vai trò CPU (Master) điều khiển IP Core FIR (Slave) thông qua các tín hiệu chuẩn của giao tiếp Wishbone như: clk, rst, adr\_i, dat\_i, dat\_o, we\_i, stb\_i, cyc\_i, ack\_o.

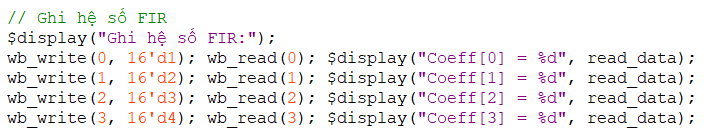
Các tác vụ kiểm thử chính được chia thành ba phần:

* wb\_write: Ghi dữ liệu vào địa chỉ được chọn (dùng để ghi hệ số lọc hoặc mẫu tín hiệu).
* wb\_read: Đọc dữ liệu từ địa chỉ chỉ định (dùng để kiểm tra lại hệ số đã ghi hoặc lấy kết quả lọc).
* check\_result: So sánh kết quả thực tế với giá trị mong đợi để xác minh tính đúng đắn của IP Core.

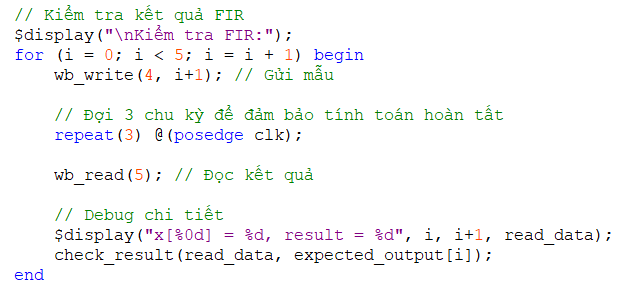
### 3.4.2. Quy trình kiểm thử

Quy trình kiểm thử được tiến hành như sau:

1. Khởi tạo và reset hệ thống: Tín hiệu *rst* được kích hoạt trong 100ns đầu tiên để đảm bảo các thanh ghi trong hệ thống trở về trạng thái mặc định.
2. Ghi hệ số FIR: Bốn hệ số lọc FIR được ghi vào các địa chỉ tương ứng từ 0 → 3. Sau khi ghi, mỗi hệ số được đọc lại để xác minh tính toàn vẹn dữ liệu ghi.



1. Gửi mẫu tín hiệu đầu vào và đọc kết quả: Năm mẫu tín hiệu lần lượt được gửi vào địa chỉ 4, tương ứng với các giá trị từ 1 đến 5. Sau mỗi mẫu gửi vào, testbench đợi một số chu kỳ đồng hồ (3 chu kỳ) để kết quả được tính toán và lưu vào thanh ghi kết quả. Sau đó, dữ liệu kết quả được đọc từ địa chỉ 5.



1. So sánh kết quả với giá trị mong đợi: Giá trị đầu ra của FIR được so sánh với kết quả tính toán tay theo công thức của bộ lọc FIR. Ví dụ, với hệ số {1, 2, 3, 4} và đầu vào {1, 2, 3, 4, 5}, đầu ra mong đợi là:

y[0] = 1

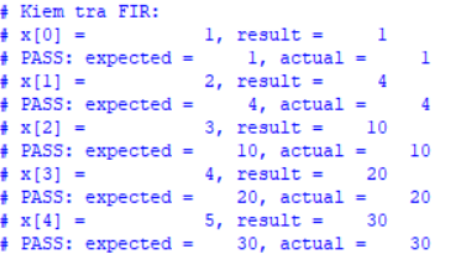
y[1] = 2 + 2 = 4

y[2] = 3 + 4 + 3 = 10

y[3] = 4 + 6 + 6 + 4 = 20  
 y[4] = 5 + 8 + 9 + 8 = 30

### 3.4.3. Kết quả mô phỏng

Trong quá trình mô phỏng, nếu IP Core hoạt động chính xác, tất cả các dòng kiểm thử check\_result sẽ hiển thị "PASS" với giá trị thực tế khớp với giá trị mong đợi.



Nếu có sai lệch, dòng "FAIL" sẽ xuất hiện kèm theo dữ liệu thực tế và mong đợi, giúp người thiết kế dễ dàng xác định lỗi trong thuật toán hoặc giao tiếp.

# CHƯƠNG 4 – TỐI ƯU THIẾT KẾ

## 4.1. Mục tiêu tối ưu hóa

Thiết kế bộ lọc FIR ban đầu được xây dựng nhằm đảm bảo tính đúng đắn về mặt chức năng. Tuy nhiên, để đáp ứng yêu cầu về hiệu năng trong các ứng dụng thời gian thực, đặc biệt khi triển khai trên FPGA, việc tối ưu là cần thiết. Mục tiêu cụ thể bao gồm:

* Giảm độ trễ xử lý (latency) từ thời điểm nhận dữ liệu đầu vào đến khi có kết quả đầu ra.
* Tăng thông lượng (throughput), nghĩa là có thể xử lý được một mẫu dữ liệu đầu vào mỗi chu kỳ đồng hồ sau giai đoạn khởi động.
* Tối ưu hóa sử dụng tài nguyên phần cứng như LUTs, thanh ghi và DSP blocks.
* Cải thiện khả năng mở rộng và tái sử dụng IP Core trong các ứng dụng có quy mô lớn hơn.

## 4.2. Các phương pháp tối ưu được áp dụng

### 4.2.1. Nhân song song (Parallel Multiply)

Trong thiết kế ban đầu, các hệ số FIR có thể được nhân tuần tự với các mẫu đầu vào, dẫn đến độ trễ lớn và thông lượng thấp. Để khắc phục điều này, thiết kế được cải tiến bằng cách thực hiện nhân song song – mỗi hệ số được nhân đồng thời với một mẫu tương ứng. Việc này được thực hiện trong cùng một chu kỳ đồng hồ bằng cách sử dụng nhiều khối nhân song song.

Ưu điểm của phương pháp này là toàn bộ các phép nhân được hoàn tất trong một chu kỳ xung nhịp, độ trễ xử lý tín hiệu được giảm đáng kể so với phương pháp nhân tuần tự và khai thác tối đa tài nguyên DSP tích hợp sẵn trong FPGA.

### 4.2.2. Cây cộng (Adder Tree)

Sau khi thực hiện nhân song song, các kết quả cần được cộng lại để cho ra đầu ra cuối cùng. Thay vì thực hiện phép cộng theo kiểu tuyến tính (tuần tự cộng từng giá trị một), thiết kế được tối ưu bằng cách sử dụng cây cộng nhị phân (adder tree).

Cây cộng được tổ chức thành các tầng, tại mỗi tầng các cặp giá trị được cộng lại đồng thời. Đối với bộ lọc có 4 bậc (N=4), cây cộng bao gồm 2 tầng với cấu trúc như sau:

Tầng 1: sum\_stage[0] = mult\_stage[0] + mult\_stage[1]

sum\_stage[1] = mult\_stage[2] + mult\_stage[3]

Tầng 2: sum\_stage[2] = sum\_stage[0] + sum\_stage[1]

Ưu điểm của phương pháp này là giúp giảm độ sâu xử lý từ O(N) xuống O(log₂N), tăng tốc độ tính toán tổng đầu ra và dễ dàng mở rộng với các bộ lọc FIR có số bậc lớn hơn.

### 4.2.3. Pipeline hóa (3 giai đoạn)

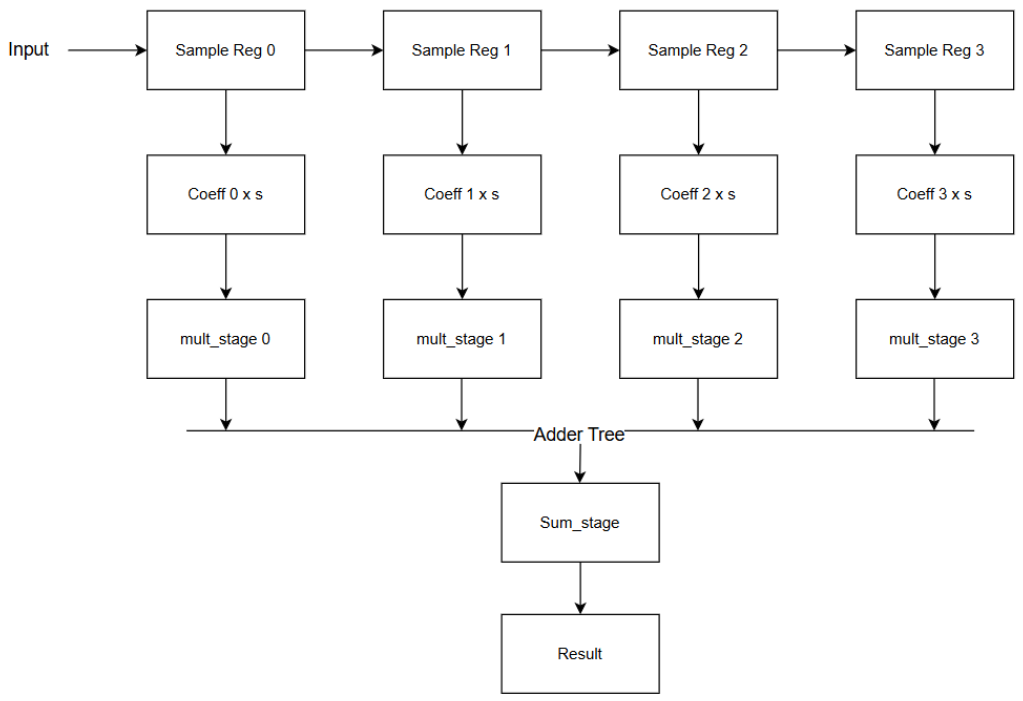
Để đạt được đầu ra ổn định mỗi chu kỳ xung nhịp và nâng cao hiệu suất toàn hệ thống, thiết kế được pipeline hóa thành ba giai đoạn riêng biệt:

1. Giai đoạn 1 – Nhân song song: Các hệ số nhân đồng thời với các mẫu → mult\_stage[]
2. Giai đoạn 2 – Cộng theo tầng: Áp dụng cây cộng để cộng dồn kết quả nhân → sum\_stage[]
3. Giai đoạn 3 – Xuất kết quả: Ghi giá trị kết quả cuối cùng vào thanh ghi đầu ra → result

Việc phân chia rõ ràng các giai đoạn xử lý giúp cải thiện khả năng hoạt động liên tục của hệ thống. Sau ba chu kỳ khởi động đầu tiên, hệ thống có thể xuất ra một kết quả đầu ra mỗi chu kỳ xung nhịp.

### 4.2.4. Mô hình kiến trúc sau tối ưu

Hình dưới đây mô tả sơ đồ kiến trúc của bộ lọc FIR sau khi áp dụng các kỹ thuật tối ưu:



Mỗi khối trong kiến trúc đảm nhiệm một vai trò riêng biệt và được thiết kế với các thanh ghi trung gian nhằm đảm bảo việc truyền dữ liệu ổn định giữa các giai đoạn pipeline.

\*Mã nguồn: Phụ lục 4

## 4.3. Mô phỏng và kiểm thử

Để đánh giá hoạt động của khối lọc FIR sau khi tối ưu, một mô-đun kiểm thử (testbench) được xây dựng. Thiết kế tối ưu này áp dụng các kỹ thuật như nhân song song, cây cộng nhị phân và pipeline hóa ba giai đoạn, do đó testbench cần đảm bảo:

* Đúng chức năng với chuỗi đầu vào mẫu.
* Kết quả đầu ra chính xác và đúng thời điểm theo pipeline (sau 3 chu kỳ khởi động).
* Xác nhận rằng hệ thống có thể xuất kết quả liên tục mỗi chu kỳ sau giai đoạn khởi động.
* Cấu hình:
* Độ rộng dữ liệu: 16-bit.
* Số hệ số FIR: 4 (ứng với N = 4).
* Tần số đồng hồ: 100 MHz (tương ứng 10 ns chu kỳ).
* Dữ liệu vào: Chuỗi mẫu đầu vào gồm 5 giá trị: [1, 2, 3, 4, 5].
* Hệ số FIR: [1, 2, 3, 4] đã được nạp sẵn.
* Chiến lược kiểm thử

Testbench thực hiện như sau:

* Áp tín hiệu reset trong 10 chu kỳ để đảm bảo hệ thống về trạng thái ban đầu.
* Sau khi reset, lần lượt gửi vào 5 mẫu dữ liệu đầu vào.
* Mỗi mẫu được gửi cách nhau 10 chu kỳ để đảm bảo pipeline hoạt động ổn định.
* Đầu ra được giám sát qua cổng data\_out\_valid và so sánh với kết quả mong đợi.
* Kết quả mong đợi

y[0] = 1

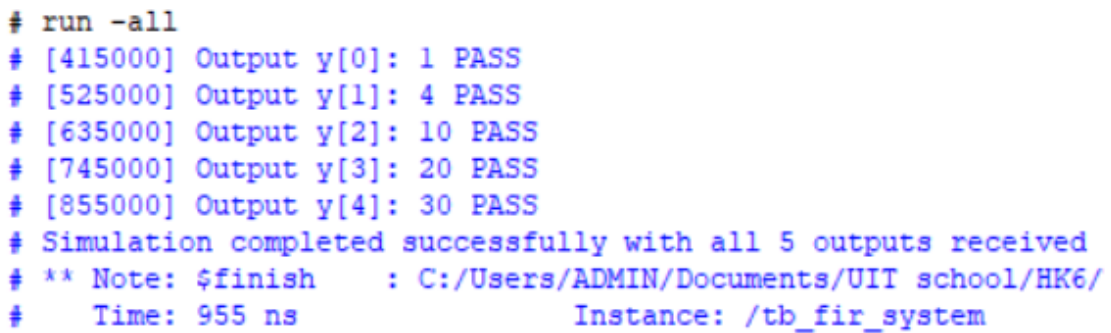
y[1] = 2 + 2 = 4

y[2] = 3 + 4 + 3 = 10

y[3] = 4 + 6 + 6 + 4 = 20  
 y[4] = 5 + 8 + 9 + 8 = 30

* Kết quả kiểm thử





## 4.4. Đánh giá hiệu quả tối ưu

Sau quá trình thiết kế và tối ưu hóa, hệ thống lọc FIR cho thấy sự cải thiện rõ rệt về hiệu năng. Cụ thể, thời gian thực thi cho một phép lọc tín hiệu đầy đủ (từ lúc nhập dữ liệu đến khi có kết quả đầu ra hợp lệ) đã giảm đáng kể từ 1780 ns ở thiết kế ban đầu xuống còn 955 ns sau khi áp dụng các kỹ thuật tối ưu. Điều này tương ứng với mức giảm khoảng 46% độ trễ xử lý, cho thấy hiệu quả rõ rệt của các giải pháp kỹ thuật.

Ba hướng tối ưu chính đã được triển khai bao gồm: (1) nhân song song, (2) cây cộng nhị phân và (3) pipeline hóa thiết kế. Việc thực hiện phép nhân song song giúp toàn bộ các hệ số được nhân cùng lúc với mẫu đầu vào, từ đó rút ngắn thời gian xử lý ban đầu. Tiếp theo, thay vì cộng tuyến tính N lần, thiết kế sử dụng cây cộng nhị phân để giảm độ sâu từ O(N) xuống O(logN), giúp tổng hợp kết quả nhanh hơn. Cuối cùng, hệ thống được pipeline hóa thành ba giai đoạn chính (nhân – cộng – ghi kết quả), cho phép sau thời gian khởi động ngắn, hệ thống có thể cung cấp đầu ra liên tục mỗi chu kỳ đồng hồ.

Những cải tiến này không chỉ giúp giảm đáng kể độ trễ tổng thể mà còn cải thiện throughput của hệ thống, giúp bộ lọc FIR trở nên phù hợp hơn với các ứng dụng yêu cầu thời gian thực. Đây là minh chứng cho tính hiệu quả của các phương pháp tối ưu hóa kiến trúc trong thiết kế phần cứng số, đồng thời đảm bảo cân bằng giữa tốc độ xử lý và mức sử dụng tài nguyên.

# 

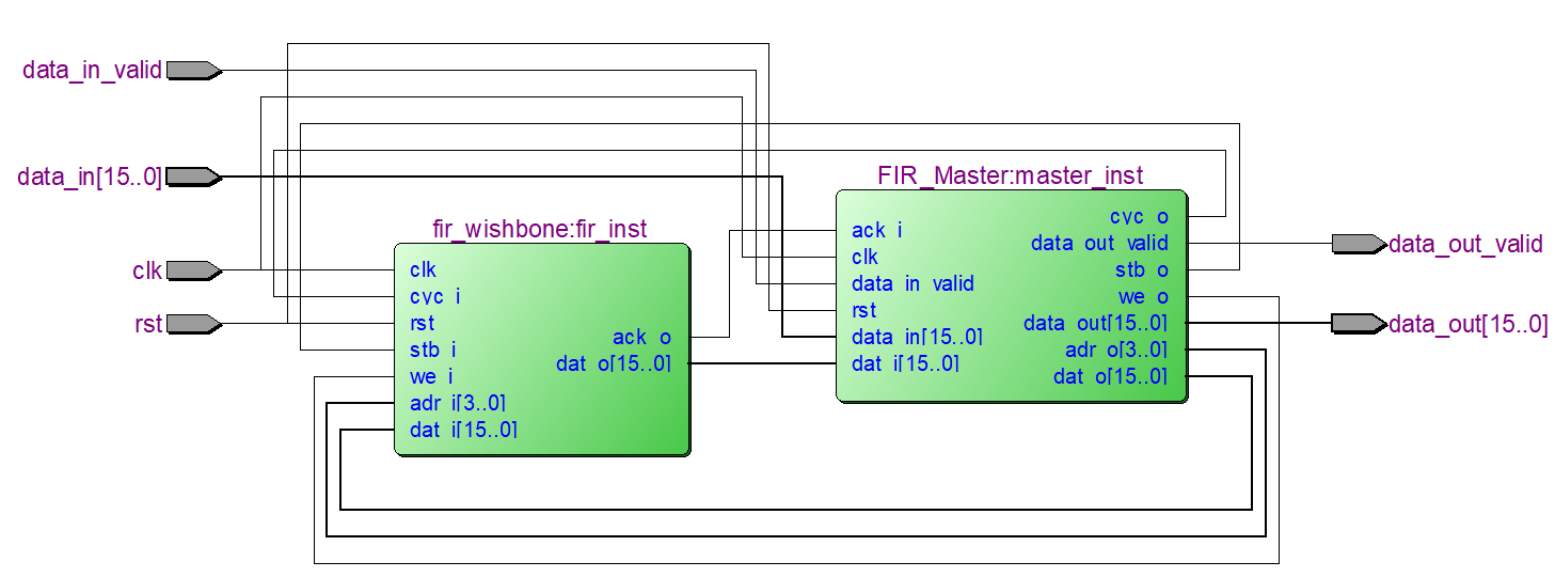
# CHƯƠNG 5 – TỔNG HỢP, TRIỂN KHAI VÀ KIỂM THỬ TRÊN FPGA

## 5.1. Tổng hợp thiết kế bằng Quartus

Thiết kế được tổng hợp bằng phần mềm Intel Quartus II với mục tiêu:

* Chuyển đổi mô tả Verilog thành sơ đồ cổng logic (netlist).
* Gán chân (pin assignment) tương ứng với board FPGA mục tiêu.
* Phân tích, ánh xạ tài nguyên (sử dụng LUT, register, DSP block...).

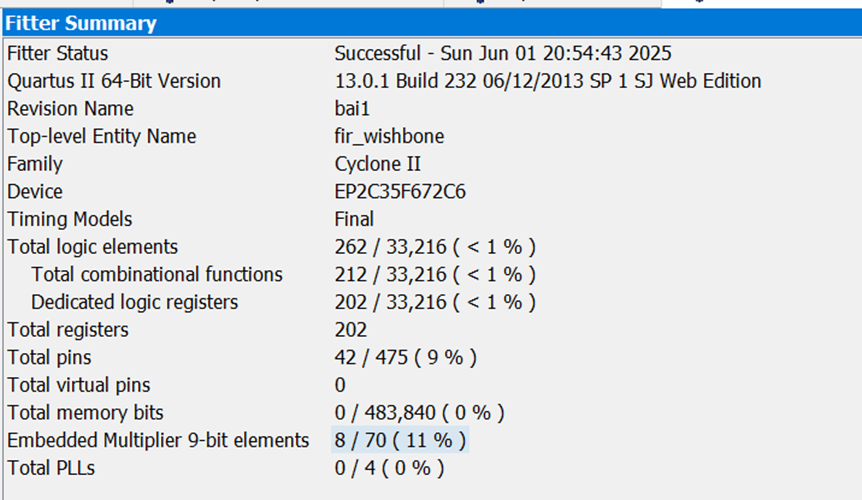
Kết quả tổng hợp:

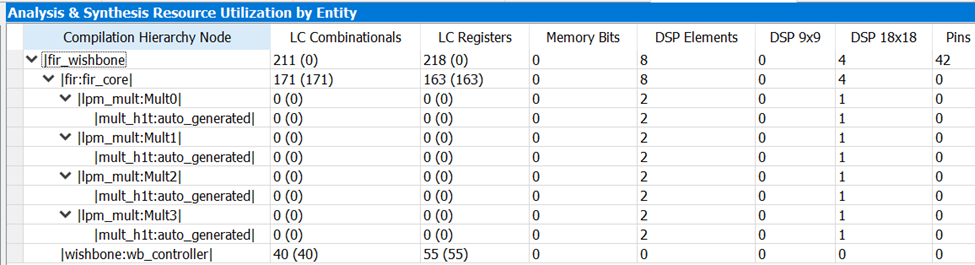


### 5.1.1. Phân tích kết quả

\***Chưa tối ưu**

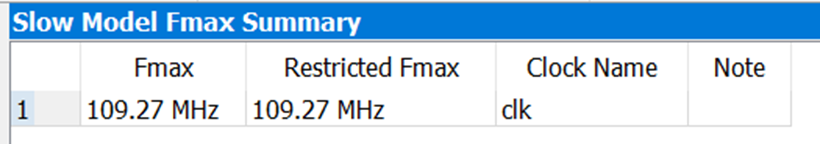
1. **Tài nguyên sử dụng**





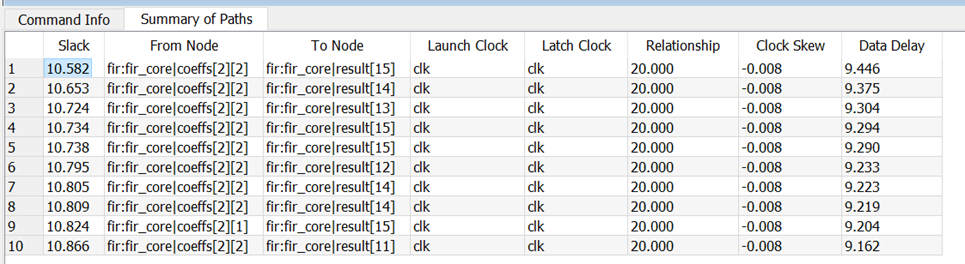
* Tổng LUTs: 262.
* Fir\_wishbone dùng tài nguyên lớn nhất với: Tổng LUTs sử dụng : 211 và 218 LC Register
* Bộ điều khiển Wishbone chiếm ít tài nguyên ( 40 LUTs và 55 LC registers)
* 8 khối DSP, phù hợp với số lượng phép nhân trong FIR

1. **Fmax và Slack**

****

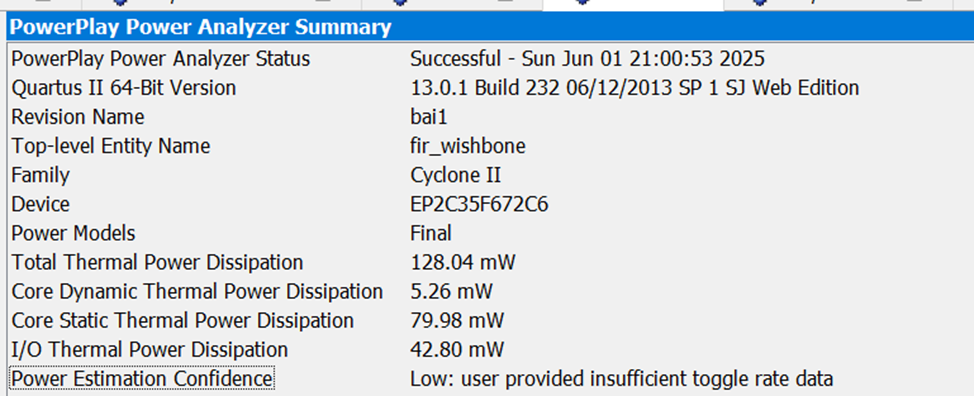
* Fmax: +109.27 MHz



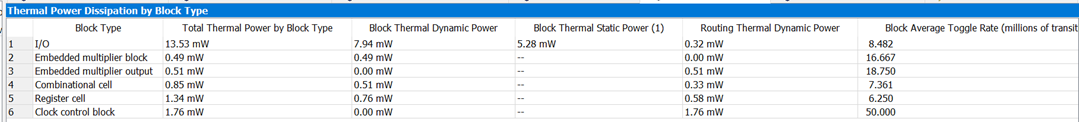


* Slack : + 10.582ns → Thiết kế đáp ứng yêu cầu timing
* Độ trễ dữ liệu (data delay) trung bình khoảng 9.2–9.4 ns.

1. **Công suất tiêu thụ**



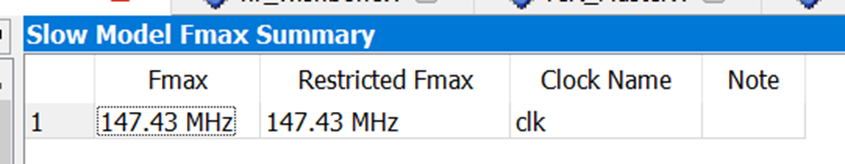
* Tổng công suất : 128.04mW
* Core Dynamic: 5.26 mW
* Core Static: 79.98 mW
* I/O: 42.80 mW



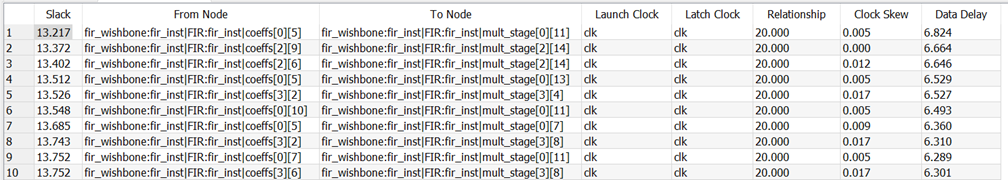
* Thiết kế tổng thể tiết kiệm năng lượng(128.04mW), I/O chiếm công suất lớn nhất(13.53mW)

**\*Đã tối ưu**

1. **Fmax và Slack**

****

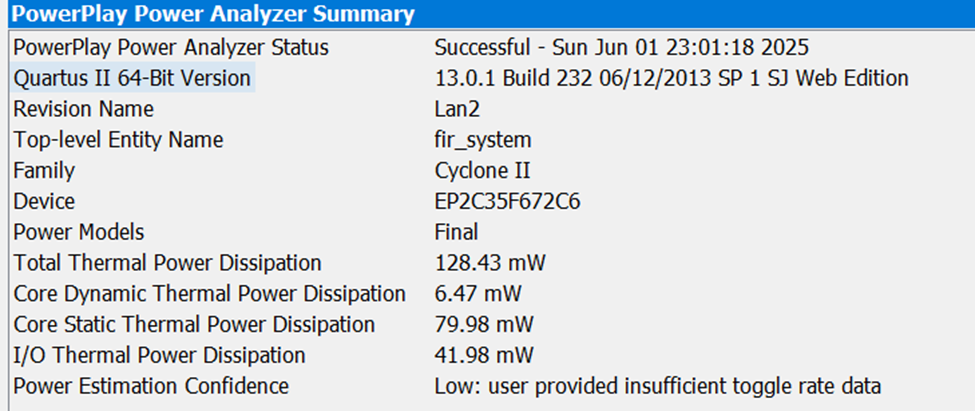
* Fmax = 147.43MHz

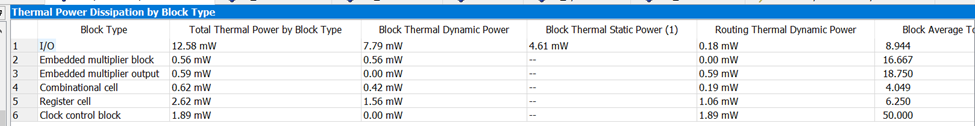


* Slack: + 13.217ns . Data delay ổn định : (6.3 – 6.8 ns)

(Tín hiệu fir\_inst|coeffs[] → mult\_stage[][] cho thấy: dữ liệu hệ số đang được đưa vào các giai đoạn nhân, tức là bộ lọc FIR đã phân chia pipeline theo từng hệ số / phần tử tích.)

1. **Công suất tiêu thụ**

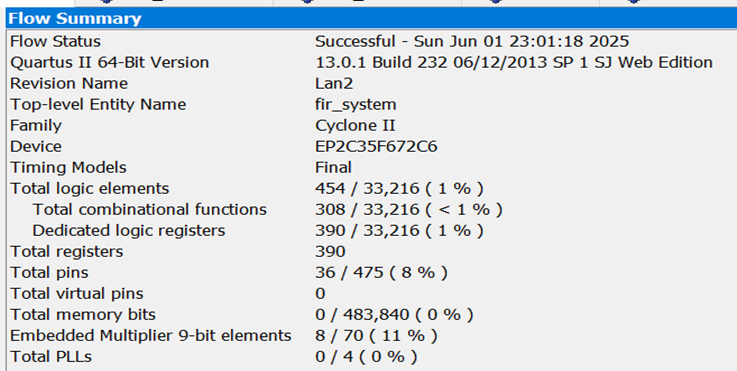
****

****

* Tổng công suất : 128.43mW
* Core Dynamic: 6.47 mW
* Core Static: 79.98 mW
* I/O: 41.98 mW

**Nhận xét:**

* Tổng công suất tăng nhẹ so với phiên bản chưa pipeline (+0.39 mW).
* Công suất động lõi tăng 1.21 mW (~23%), chứng tỏ pipeline hoạt động logic nhiều hơn.



* Tổng số LUTs: 454

### 5.1.2. Phân tích timing

|  | **Chưa pipeline** | **Đã pipeline** |
| --- | --- | --- |
| **Fmax** | 109.27 MHz | 147.42 MHz |
| **Slack** | +10.582ns | +13.217ns |

→ Việc Fmax và Slack tăng lên trong lần chạy 2 là kết quả của: Tối ưu tuyến đường (routing) tốt hơn, Critical path ngắn hơn, Sắp xếp logic hiệu quả hơn

Tuy có sự khác biệt nhưng cả hai lần đều đạt slack dương, chứng tỏ thiết kế đáp ứng yêu cầu timing và hoạt động ổn định với tần số yêu cầu (50 MHz).

**Phân tích công suất tiêu thụ**

|  | **Chưa pipeline** | **Đã pipeline** |
| --- | --- | --- |
| **Tổng công suất** | 128.04mW | 128.43mW |
| **Core Dynamic** | 5.26mW | 6.47mW |
| **Core Static** | 79.98mW | 79.98mW |
| **I/O** | 42.80mW | 41.98mW |

- Tổng công suất tiêu thụ giữa hai lần chạy gần như không thay đổi, chỉ tăng 0.3%

=> Hệ thống vẫn tiết kiệm năng lượng, không có ảnh hưởng lớn đến nhiệt hoặc yêu cầu nguồn.

Tăng thanh ghi và giảm logic tổ hợp cho thấy thiết kế đã thực sự pipeline hiệu quả.

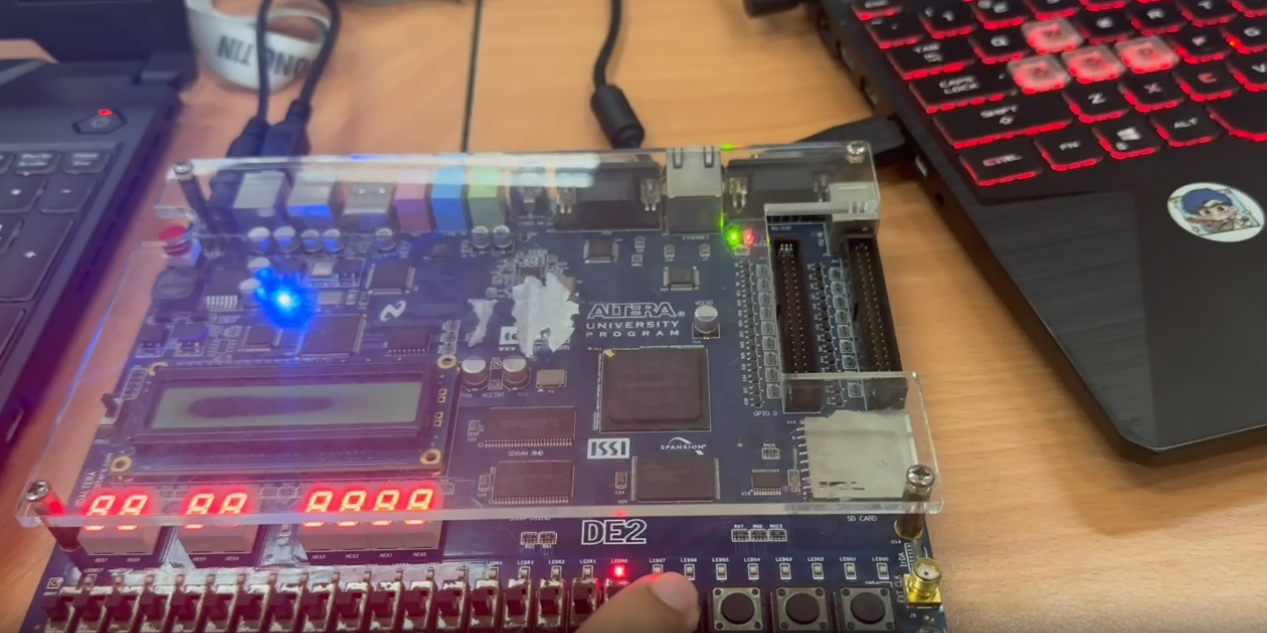
Công suất động tăng lên là điều hợp lý và thường thấy khi Fmax tăng, hoặc logic hoạt động hiệu quả hơn.

## 5.2. Nạp và kiểm thử trên FPGA

Sau khi tổng hợp thành công thiết kế trên phần mềm Quartus II, IP core bộ lọc FIR cùng giao tiếp Wishbone đã được nạp xuống kit FPGA (sử dụng nền tảng Intel-Altera). Quá trình nạp sử dụng công cụ Quartus Programmer thông qua kết nối JTAG với thiết bị thực. Việc cấu hình FPGA được thực hiện từ file .sof sinh ra trong quá trình tổng hợp.



Để kiểm thử, sử dụng bộ điều khiển Wishbone giả lập trong testbench, thực hiện ghi hệ số bộ lọc và cung cấp các mẫu dữ liệu đầu vào. Dữ liệu đầu ra được đọc và so sánh với giá trị mong đợi để đánh giá tính đúng đắn của hệ thống trên phần cứng thực.



Kết quả kiểm thử cho thấy hệ thống hoạt động ổn định và cho kết quả chính xác, tương ứng với kết quả mô phỏng trước đó. Tín hiệu điều khiển giao tiếp Wishbone như ACK\_O, STB\_I, CYC\_I, và WE\_I đáp ứng đúng đặc tả giao thức, đảm bảo khả năng tích hợp vào các hệ thống SoC phức tạp.

Việc triển khai và kiểm thử thực tế giúp xác nhận rằng thiết kế không chỉ đúng về mặt chức năng mà còn đáp ứng tốt về mặt thời gian và khả năng tương thích phần cứng. Đây là bước quan trọng chứng minh tính khả thi và độ tin cậy của hệ thống khi ứng dụng trong môi trường thực tế.

\*Video triển khai trên KIT: link ở Phụ lục 5

# 

# 

# CHƯƠNG 6 – KẾT LUẬN

Trong khuôn khổ đồ án "Thiết kế bộ tăng tốc lọc tín hiệu Finite Impulse Response (FIR) theo chuẩn giao tiếp Wishbone", nhóm đã hoàn thành việc tìm hiểu, thiết kế, mô phỏng và triển khai một hệ thống lọc FIR hoạt động theo kiến trúc phần cứng số, đáp ứng chuẩn giao tiếp Wishbone và tối ưu hóa hiệu năng để phù hợp triển khai trên FPGA.

Trước hết, đồ án đã phân tích chi tiết nguyên lý hoạt động của bộ lọc FIR và các phương pháp triển khai hiệu quả trên phần cứng. Chuẩn giao tiếp Wishbone cũng được nghiên cứu nhằm đảm bảo khả năng tích hợp IP core FIR vào các hệ thống SoC thông dụng. Trên cơ sở đó, một thiết kế phần cứng mô tả bằng ngôn ngữ Verilog HDL đã được xây dựng với khả năng cấu hình hệ số lọc, truyền dữ liệu vào/ra và tuân thủ đầy đủ đặc tả giao thức Wishbone.

Tiếp theo, nhóm đã tiến hành xây dựng testbench để mô phỏng và kiểm thử hoạt động của IP core FIR. Kết quả mô phỏng xác nhận tính đúng đắn của thiết kế. Sau đó, thiết kế được tối ưu thông qua các kỹ thuật như nhân song song, cây cộng (adder tree) và pipeline ba giai đoạn, giúp giảm đáng kể độ trễ xử lý tín hiệu từ 1780ns xuống còn 955ns – tương ứng giảm hơn 46% thời gian thực thi, trong khi vẫn đảm bảo tính chính xác tuyệt đối.

Cuối cùng, thiết kế được tổng hợp, triển khai và kiểm thử thành công trên kit FPGA của Intel-Altera. Kết quả thu được từ phần cứng trùng khớp với mô phỏng, chứng tỏ tính khả thi, độ tin cậy và hiệu suất cao của hệ thống FIR được thiết kế.

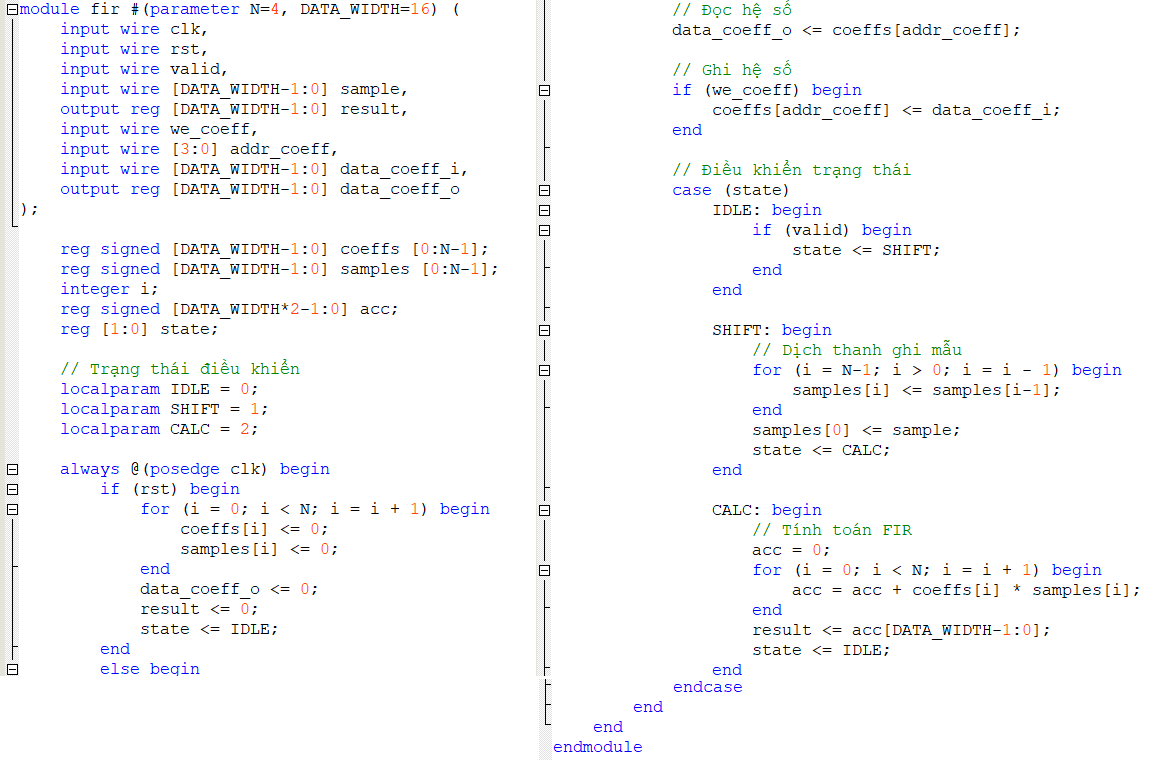
# 

# 

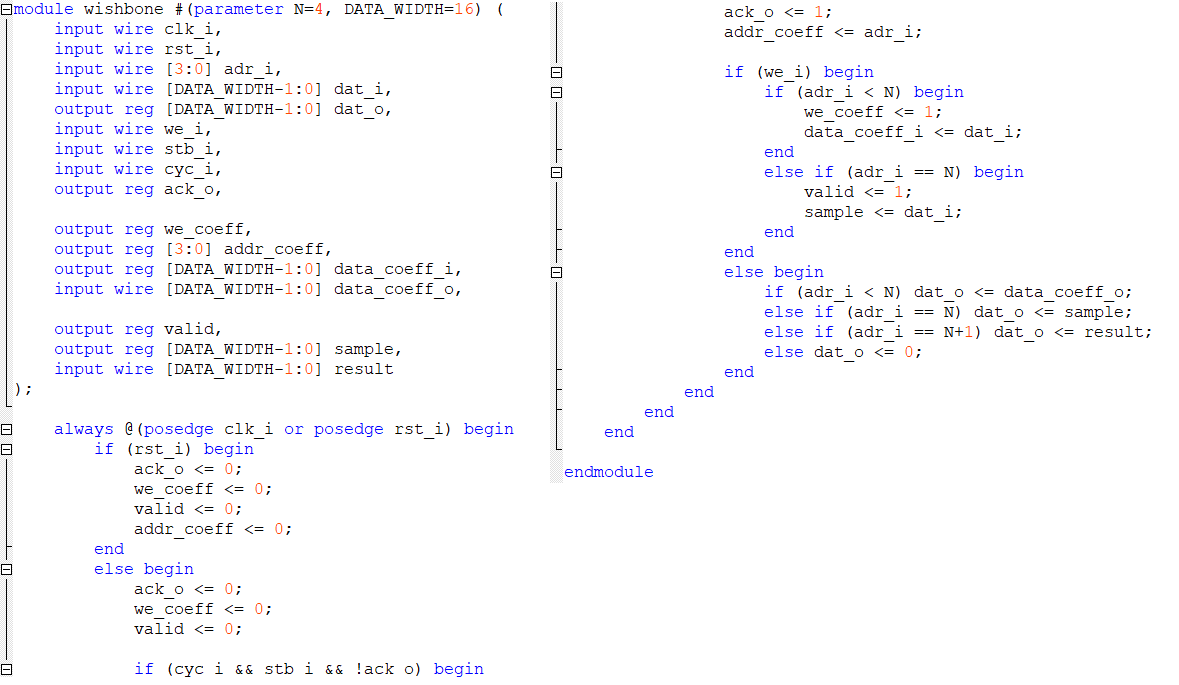
# 

# PHỤ LỤC

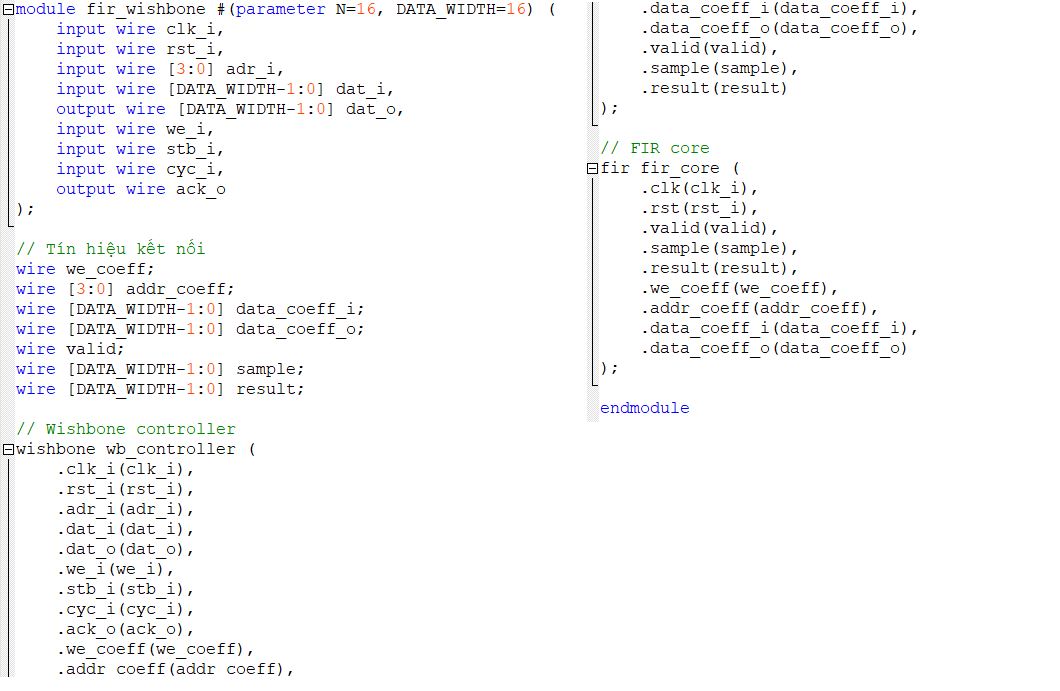
1. Mã nguồn thiết kế FIR (chưa tối ưu)



1. Mã nguồn thiết kế Wishbone (chưa tối ưu)

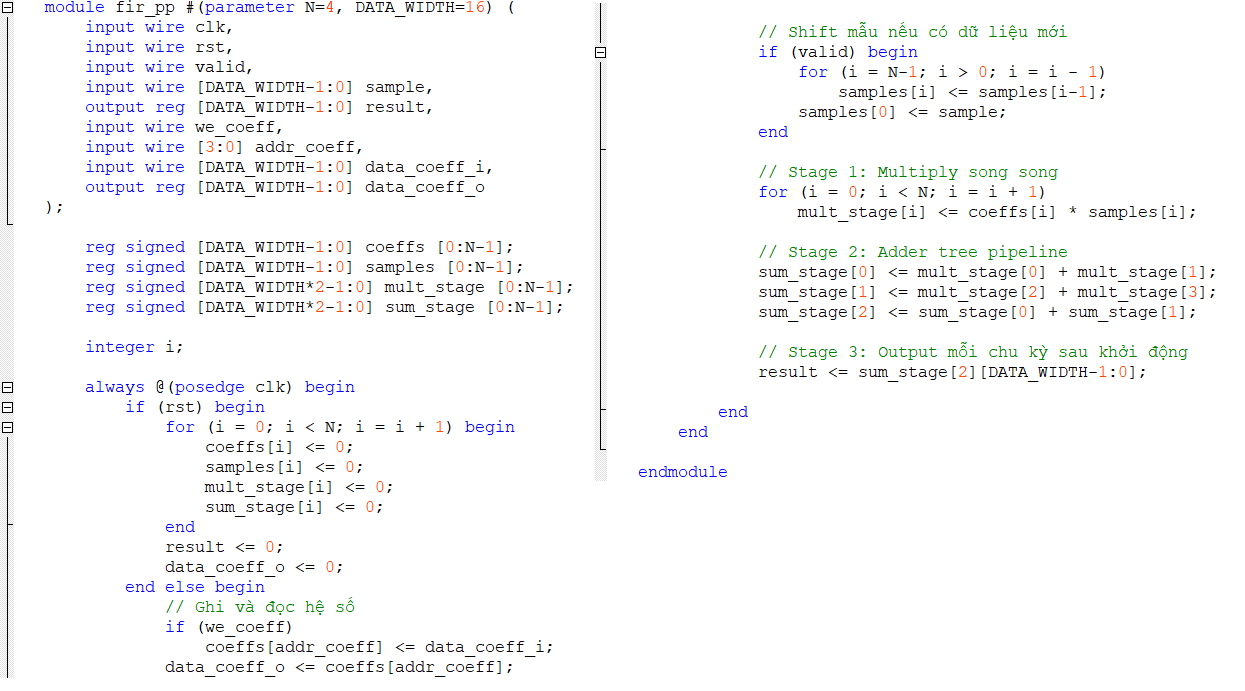


1. Mã nguồn thiết kế FIR\_Wishbone (chưa tối ưu)

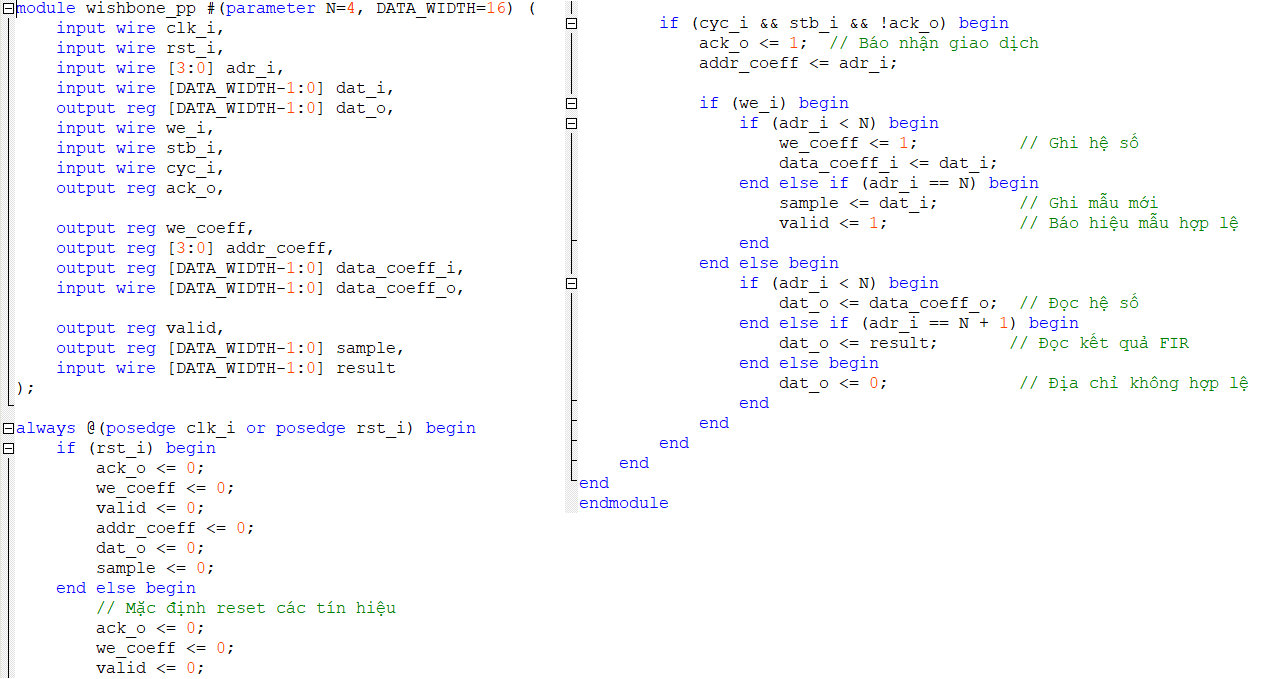


1. Mã nguồn thiết kế RTL đã tối ưu

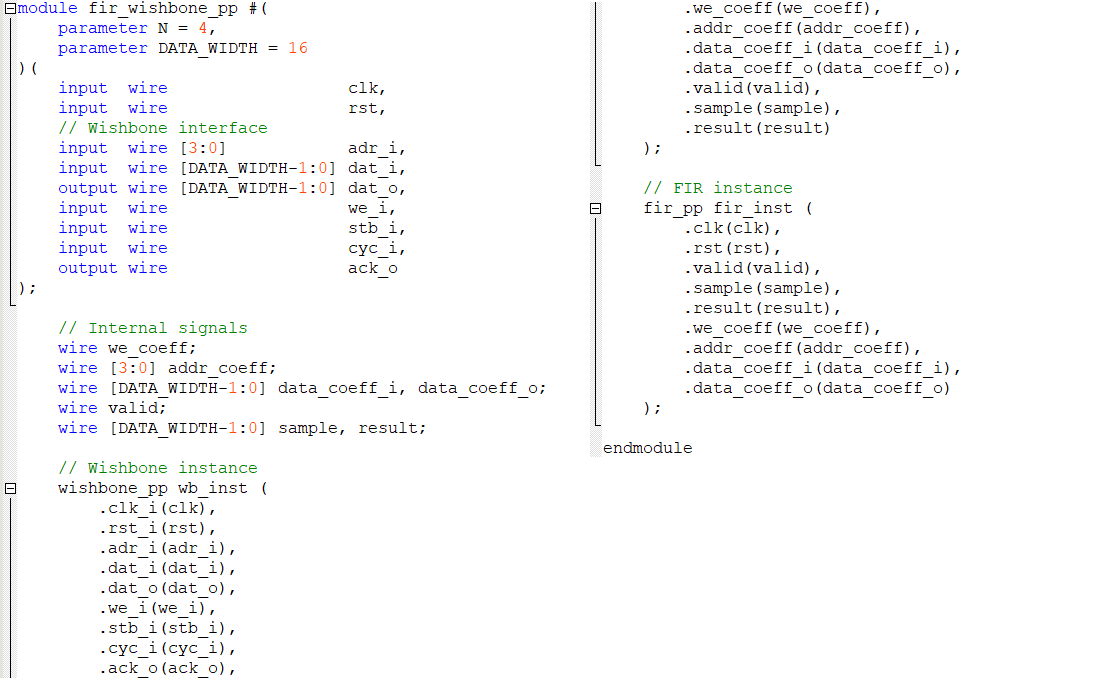
* Module FIR



* Module Wishbone



* Module FIR\_Wishbone (Slave)



* Module Master (Đảm nhận điều khiển các tín hiệu)



1. Link video kiểm thử trên FPGA

[Video\_NapKIT.mp4](https://drive.google.com/file/d/1WotjwGOF3KDNs5Fn3WzOfN5GVloHZ-1e/view)

1. Mã nguồn đồ án

[Ma\_nguon\_FIR\_Wishbone](https://github.com/pd23456789/Final-Project-HDL.git)

# TÀI LIỆU THAM KHẢO

[1] “DSP for FPGA: Simple FIR Filter in Verilog.” *Hackster.io*, [www.hackster.io/whitney-knitter/dsp-for-fpga-simple-fir-filter-in-verilog-91208d](http://www.hackster.io/whitney-knitter/dsp-for-fpga-simple-fir-filter-in-verilog-91208d).

[2] Distributed Arithmetic Architectures for FIR Filters-A Comparative review

<https://www.researchgate.net/publication/323351094_Distributed_arithmetic_architectures_for_FIR_filters-A_comparative_review>

[3] FIR Filter Design by Convex Optimization Using Directed Iterative Rank Refinement Algorithm

<https://ieeexplore.ieee.org/document/7373669>

[4] Design of FIR Filter Using Particle Swarm Optimization

(International Advanced Research Journal in Science, Engineering and Technology Vol.3, Issue 5, May 2016)

[5] Optimized FIR Filter using Genetic Algorithms: A Case Study of ECG Signals Filter Optimization

<https://www.researchgate.net/publication/373369317_Optimized_FIR_Filter_using_Genetic_Algorithms_A_Case_Study_of_ECG_Signals_Filter_Optimization>

[6] Design of Very High-Speed Pipeline FIR Filter Through Precise Critical Path Analysis

<https://ieeexplore.ieee.org/document/9361650>

[7] Pipelined Direct Form FIR Versus the Transposed Structure

<https://www.allaboutcircuits.com/technical-articles/pipelined-direct-form-fir-versus-the-transposed-structure/>