ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN: THIẾT KẾ HỆ THỐNG SỐ HDL

THIẾT KẾ BỘ TĂNG TỐC BIẾN ĐỔI TÍN HIỆU DISCRETE FOURIER TRANSFORM (DFT) THEO CHUẨN GIAO TIẾP WISHBONE

Giảng viên hướng: ThS. Tạ Trí Đức

Mã môn học: CE213.P23

Nhóm Sinh viên thực hiện:

Nguyễn Hiền My – 22520899

Huỳnh Thanh Hà – 22520369

Lê Hồ Thanh Linh – 22520760

Nguyễn Ngọc Minh Thức – 22521452

TP. Hồ Chí Minh, tháng 5 năm 2025

LỜI CẢM ƠN

Để hoàn thành đồ án môn học, chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy Tạ Trí Đức – người đã tận tâm hướng dẫn, hỗ trợ và đồng hành cùng chúng em trong suốt quá trình thực hiện đề tài "Thiết kế bộ tăng tốc biến đổi tín hiệu Discrete Fourier Transform (DFT) theo chuẩn giao tiếp Wishbone".

Nhờ sự hướng dẫn nhiệt tình, tận tụy và kiến thức chuyên môn sâu rộng của thầy, chúng em đã vượt qua được nhiều khó khăn và đạt được những kết quả tích cực trong quá trình thực hiện đồ án. Không chỉ truyền đạt những kiến thức chuyên ngành quý báu, thầy còn giúp chúng em rèn luyện kỹ năng làm việc nhóm, tư duy logic, khả năng nghiên cứu khoa học và giải quyết vấn đề một cách sáng tạo, hiệu quả.

Chúng em đặc biệt trân trọng sự tận tâm của thầy khi đã dành thời gian góp ý, chỉnh sửa, định hướng báo cáo, giúp nhóm em hoàn thiện sản phẩm một cách tốt nhất. Những nhận xét và chỉ dẫn của thầy không chỉ giúp chúng em hoàn thành đồ án mà còn là hành trang quý giá cho chặng đường học tập và phát triển sau này.

Với giới hạn về kiến thức và kinh nghiệm thực tiễn, đồ án của chúng em chắc chắn vẫn còn những thiếu sót. Kính mong nhận được sự góp ý, nhận xét từ quý thầy để chúng em có cơ hội học hỏi và hoàn thiện hơn trong tương lai.

Một lần nữa, tập thể nhóm xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày 2 tháng 6 năm 2025

Nhóm sinh viên thực hiện

MỤC LỤC

DANH MỤC CÁC HÌNH ẢNH

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1. Lý do chọn đề tài

Biến đổi Fourier rời rạc (DFT) là một trong hai phương pháp phổ biến và mạnh mẽ nhất được sử dụng trong lĩnh vực xử lý tín hiệu số (phương pháp còn lại là lọc số – digital filtering). DFT cho phép chúng ta phân tích, biến đổi và tổng hợp tín hiệu theo những cách mà xử lý tín hiệu tương tự (analog) liên tục không thể thực hiện được.

Về cơ bản, DFT là một phép biến đổi toán học được dùng để xác định thành phần điều hòa (harmonic) hoặc thành phần tần số của một chuỗi tín hiệu rời rạc, chuỗi tín hiệu rời rạc được hiểu là tập các giá trị thu được bằng cách lấy mẫu định kỳ một tín hiệu liên tục theo thời gian. DFT rất hữu ích trong việc phân tích bất kỳ chuỗi rời rạc nào, bất kể chuỗi đó đại diện cho điều gì.

Biển đổi này là một công cụ nền tảng trong xử lý tín hiệu số, chuyển đổi tín hiệu từ miền thời gian sang miền tần số, được ứng dụng rộng rãi trong viễn thông, xử lý âm thanh, hình ảnh, phân tích tín hiệu y tế (ECG, EEG), và radar. Tuy nhiên, việc tính toán trực tiếp DFT có độ phức tạp $O(N^2)$, gây khó khăn cho các ứng dụng thời gian thực khi độ dài tín hiệu N lớn.

Để giải quyết vấn đề này, các thuật toán biến đổi Fourier nhanh (FFT) như Radix-2, Radix-4, và Radix-2^2 đã được phát triển, giảm độ phức tạp xuống $O(N\ log\ N)$. Trong đó, thuật toán Radix-2^2 là một lựa chọn tối ưu nhờ kết hợp cấu trúc bướm đơn giản của Radix-2 và số phép nhân phức thấp của Radix-4. Radix-2^2 sử dụng hai giai đoạn phân chia chẵn/lẻ liên tiếp, giảm số phép nhân không tầm thường và phù hợp với triển khai phần cứng.

Ngoài ra, chuẩn giao tiếp Wishbone được phát triển bởi OpenCores, là một giao thức mở, linh hoạt, và chuẩn hóa, được sử dụng rộng rãi trong thiết kế hệ thống trên chip (SoC). Wishbone hỗ trợ kết nối các khối IP (Intellectual Property) với độ trễ thấp, khả năng tái sử dụng cao, và tương thích với nhiều nền tảng phần cứng như FPGA hoặc ASIC. Việc tích hợp chuẩn Wishbone vào bộ tăng tốc DFT cho phép tích hợp dễ dàng vào các hệ thống SoC phức tạp, đáp ứng nhu cầu của các ứng dụng thời gian thực như 5G, IoT, và xử lý đa phương tiện.

Một cách tóm gọn, nhóm chọn đề tài bởi các lý do sau:

- Đáp ứng nhu cầu thực tiễn, bởi việc xử lý tín hiệu số chiếm một vai trò vô cùng quan trọng đối với các lĩnh vực công nghệ cao ngày nay như 5G, xử lý hình ảnh…
- Nhóm quyết định sử dụng thuật toán Radix-2^2 vì thuật toán này cung cấp hiệu suất cao, cũng như không quá phực tạp để triển khai trên FPGA.
- Chuẩn giao tiếp Wishbone giúp đảm bảo khả năng tích hợp hệ thống SoC.

1.2. Mục tiêu đề tài

Đề tài hướng đến thiết kế và triển khai một bộ tăng tốc phần cứng cho DFT, sử dụng thuật toán Radix-2^2 và tích hợp với chuẩn giao tiếp Wishbone. Các mục tiêu cụ thể bao gồm:

- Nghiên cứu và triển khai thuật toán DFT/FFT dựa trên Radix-2^2 để tối ưu hóa hiệu suất xử lý tín hiệu.
- Tích hợp chuẩn giao tiếp Wishbone để đảm bảo khả năng kết nối với các khối IP khác trong hệ thống SoC.
- Thiết kế mô tả phần cứng bằng Verilog HDL, bao gồm cả RTL (Register Transfer Level) và testbench để kiểm tra chức năng.
- Tổng hợp và triển khai thiết kế trên FPGA của Intel-Altera bằng công cụ Quartus, đảm bảo tính đúng đắn của thiết kế.
- Nạp kit FPGA để đánh giá tính đúng đắn thực tế.
- Tối ưu hóa thiết kế về độ trễ, tần số hoạt động tối đa, tài nguyên phần cứng.

1.3. Nội dung thực hiện

Nội dung thực hiện dự kiến của đề tài này bao gồm:

- Tìm hiểu về thuật toán biến đổi DFT, về thuật toán Radix-2^2 để triển khai xuống phần cứng.
- Tìm hiểu về giao thức wishbone
- Thiết kế bằng ngôn ngữ mô tả phần cứng verilog HDL, tích hợp thiết kế dựa trên thuật toán đã tìm hiểu với giao thức wishbone
- Sử dụng công cụ Quartus của Intel-Altera để tổng hợp thiết kế
- Thiết kế testbench và mô phỏng trên phần mềm modelSim
- Tối ưu hóa thiết kế bằng pipeline
- Báo cáo kết quả tổng hợp thiết kế về mặt tài nguyên sử dụng, độ trễ, tần số hoạt động tối đa.
- Nạp kit DE2

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Biến đổi Fourier rời rạc (DFT)

Biến đổi Fourier Rời rạc (DFT) chuyển đổi một dãy tín hiệu thời gian rời rạc $x[n]$ (với $n = 0,1, \dots, N-1$) sang miền tần số $X[k]$ (với $k = 0,1, \dots, N-1$) . Công thức DFT được định nghĩa như sau:

$$X[k] = \sum^{N-1} x\,[n] W_N^{nk}, \quad k = 0,1, \dots, N-1$$

$n=0$

Trong đó:

- $x[n]$: Giá trị tín hiệu tại mẫu n
- $X[k]$: Giá trị phổ tần số tại tần số k
- $W_N = e^{-j\frac{2\pi}{N}}$: Hệ số xoay (twiddle factor), là căn nguyên thủy bậc Ncủa đơn vị.
- N: Độ dài của dãy tín hiệu.

Việc tính trực tiếp DFT yêu cầu N^2 phép nhân và cộng phức, dẫn đến độ phức tạp tính toán N^2. Điều này trở nên không khả thi cho các dãy tín hiệu dài trong các ứng dụng như xử lý âm thanh, hình ảnh, hoặc viễn thông thời gian thực.

2.2. Biến đổi Fourier nhanh (FFT)

Biến đổi Fourier nhanh (FFT) được phát triển để giảm độ phức tạp tính toán của DFT xuống còn $O(N\ log\ N)$. FFT sử dụng nguyên tắc "chia để trị" (divide-andconquer), chia bài toán DFT lớn thành các bài toán con nhỏ hơn. Thuật toán Cooley-Tukey là một trong những phương pháp phổ biến nhất, yêu cầu N là lũy thừa của một số nguyên (thường là 2 hoặc 4).

Trong thuật toán Cooley-Tukey, dãy tín hiệu được chia thành các phần nhỏ hơn (chẳng hạn, chỉ số chẵn và lẻ trong Radix-2), sau đó tính DFT cho từng phần và kết hợp kết quả bằng hệ số xoay. Các biến thể phổ biến bao gồm Radix-2, Radix4, và Radix-2^2,...

2.3. Thuật toán Radix-2^2

Thuật toán Radix-2^2 là một biến thể của FFT, được thiết kế để kết hợp ưu điểm của Radix-2 (cấu trúc bướm đơn giản) và Radix-4 (số phép nhân phức thấp). Theo He và Torkelson (1996), Radix-2^2 đạt được độ phức tạp nhân tương đương với Radix-4, nhưng giữ cấu trúc bướm của Radix-2, giúp dễ dàng triển khai trên phần cứng, đặc biệt trong kiến trúc pipeline FFT.

2.3.1. Phân chia tín hiệu

Radix-2^2 thực hiện hai giai đoạn phân chia chẵn/lẻ liên tiếp, tương đương với việc chia dãy tín hiệu thành bốn nhóm. Để suy ra thuật toán, ta xem xét hai bước phân chia đầu tiên của FFT dạng Decimation-In-Frequency (DIF). Công thức DFT được viết lại bằng cách sử dụng ánh xạ chỉ số tuyến tính 3 chiều:

Trong đó $n_1, n_2 \in \{0,1\}, n_3 = \overline{0,1,\dots,N/4-1}$ và $k_1, k_2 \in \{0,1\}, k_3 = \overline{0,1,\dots,N/4-1}$. Thay vào công thức DFT, ta được:

$$X(k_1 + 2k_2 + 4k_3)$$

$$N/4-1 \quad 1 \quad 1$$

$$= \sum_{n3=0}\sum_{n2=0}\sum_{n1=0} x\left(\frac{N}{4}n_1 + \frac{N}{2}n_2 + n_3\right) W_N^{\left(\frac{N}{4}n_1+\frac{N}{2}n_2+n_3\right)(k_1+2k_2+4k_3)}$$

Tách biệt phần bướm (butterfly) cho n_1:

$$B_{\frac{N}{2}}(n_2, n_3) = x(n_2, n_3) + (-1)^{k1} x(n_2 + n_3 + \frac{N}{2})$$

Kết quả là:

$$X(k_1 + 2k_2 + 4k_3) = W_N^{\left(\frac{N}{4}n_2+n_3\right)(k_1+2k_2+4k_3)} \sum_{n3=0}^{N/4-1}\sum_{n2=0}^{1} B_{\frac{N}{2}}\left(\frac{N}{4}n_2 + n_3\right)$$

2.3.2. Phân tích hệ số xoay

Điểm mấu chốt của Radix-2^2 là phân tách hệ số xoay để giảm số phép nhân phức không tầm thường:

$$W_{N}^{(4n^2+n^3)(k^1+2k^2+4k^3)} = W_{\frac{N}{4n^2(k^1+2k^2)}} W_{\frac{N}{4n^2(4k^3)}} W_{N}^{n3(k1+2k2)} W_{N}^{n3(4k3)}$$

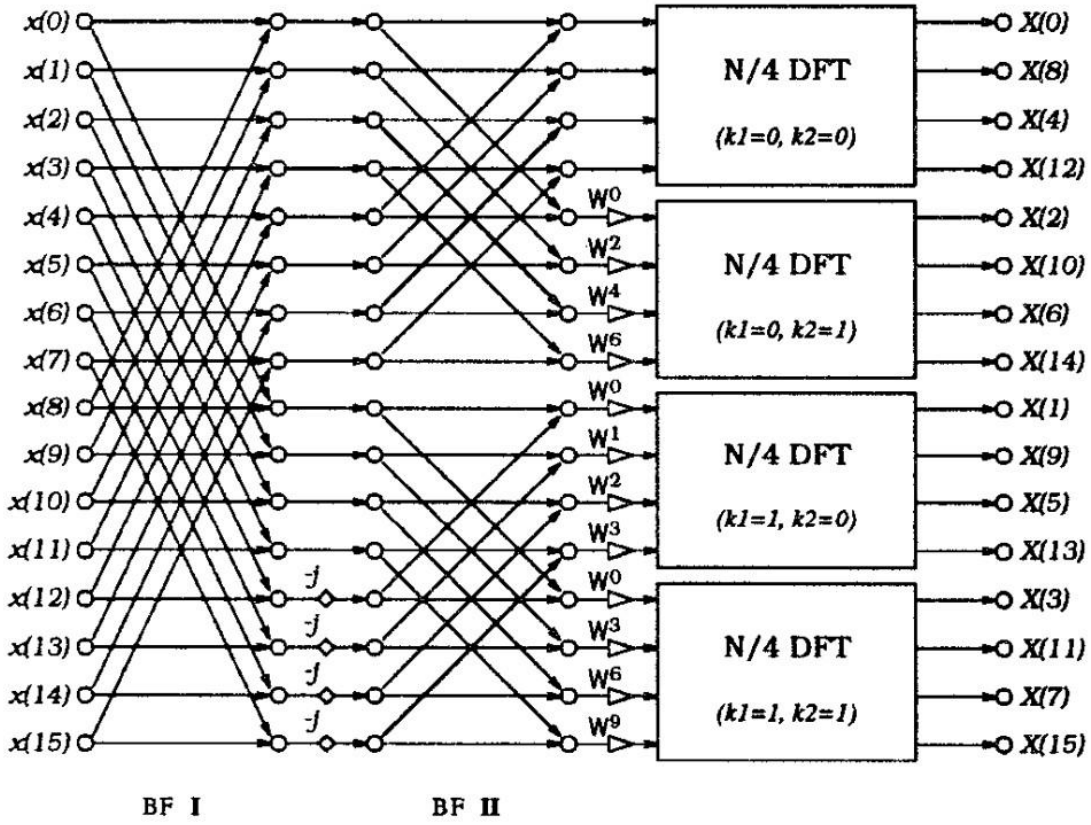
Sử dụng $W_N^4 = e^{-j\frac{2\pi}{N}\cdot\frac{N}{4}} = -j$, ta có:

$$W_{\frac{N}{4}}^{n2(k1+2k2)} = (-j)^{n2(k1+2k2)}$$

Điều này dẫn đến các phép nhân tầm thường (trivial multiplications), chỉ yêu cầu hoán đổi phần thực-phần ảo hoặc đổi dấu. Sau khi đơn giản hóa, công thức trở thành:

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{N/4-1}\left[H(k_1, k_2, n_3) W_N^{n_3(k_1+2k_2)}\right] W_{\frac{N}{4}}^{n_3 k_3} \quad (*)$$

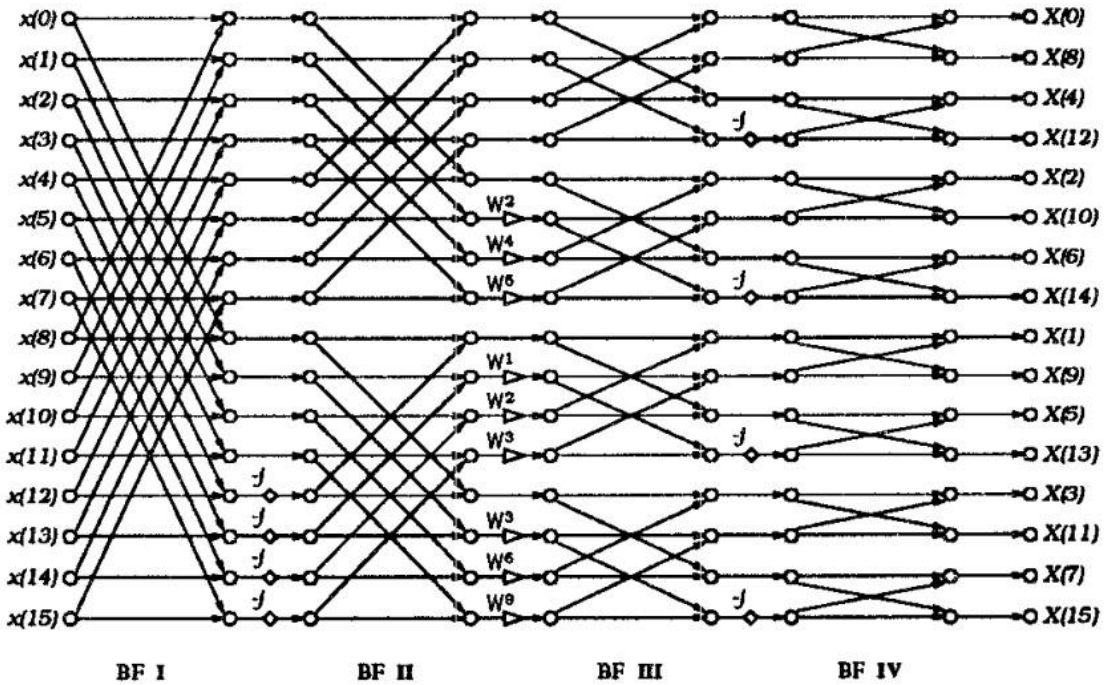
Trong đó $H(k_1, k_2, k_3)$ là kết quả của hai giai đoạn bướm đầu tiên, chỉ chứa các phép nhân tầm thường. Các phép nhân phức đầy đủ chỉ xuất hiện sau hai giai đoạn bướm, với hệ số $W_{Nn3(k1+2k2)}$.



Hình 2.1. *Bướm (butterfly) với các hệ số twiddle factors*

2.3.3. Áp dụng đệ quy

Quá trình trên được áp dụng đệ quy cho các DFT độ dài $N/4$ trong phương trình (*), tạo ra thuật toán Radix-2^2 hoàn chỉnh.



Hình 2.2. *Luồng tín hiệu Radix-2^2 FFT với N = 16*

Hình trên minh hoạt luồng tín hiệu cho cho N = 16, trong đó các phép nhân tầm thường (bằng $-j$) được biểu diễn bằng các hình thoi nhỏ, chỉ yêu cầu hoán đổi thực-ảo hoặc đổi dấu.

Ưu điểm của thuật toán Radix-2^2 bao gồm:

- Độ phức tạp của phép nhân tương đương với Radix-4, yêu cầu $log_4 N - 1$ phép nhân phức, thấp hơn so với $2(log_4 N - 1)$ của Radix-2.
- Cấu trúc bướm (butterfly) của thuật toán này giữ được sự đơn giản của Radix-2, điều này giúp dễ triển khai trên phần cứng.

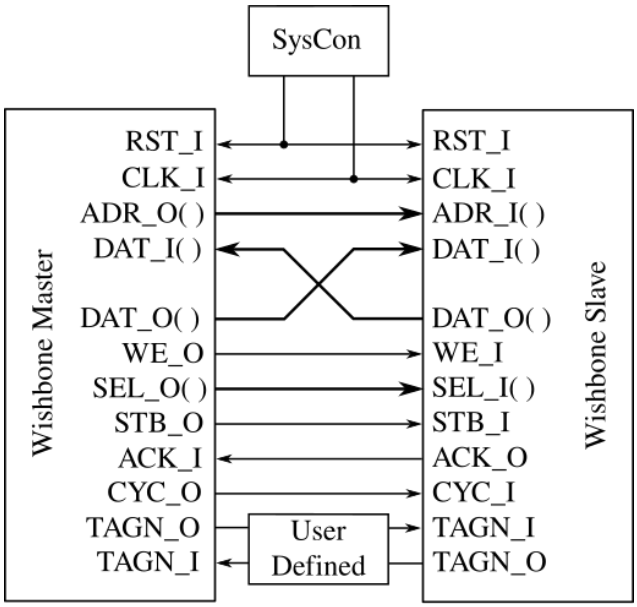
2.4. Chuẩn giao tiếp wishbone

2.4.1. Giới thiệu chung

Wishbone là một chuẩn giao tiếp bus mở được phát triển bởi OpenCores, hỗ trợ tích hợp IP core trong thiết kế hệ thống số, đặc biệt trong SoC và FPGA.

Mục tiêu của chuẩn giao tiếp này bao gồm: chuẩn hóa giao tiếp giữa các IP core, hỗ trợ tích hợp các module độc lập, tái sử dụng và giảm thời gian thiết kế, tăng tính tương thích giữa các khối chức năng.

Kiến trúc Wishbone sử dụng mô hình Master – Slave, trong đó các thiết bị đóng vai trò Master (chẳng hạn như CPU hoặc DMA) sẽ khởi tạo quá trình truyền thông, còn các thiết bị Slave có nhiệm vụ phản hồi các yêu cầu từ Master. Wishbone hỗ trợ nhiều cấu hình kết nối linh hoạt tùy thuộc vào yêu cầu của hệ thống. Các cấu hình này bao gồm: kết nối point-to-point là dạng đơn giản nhất, shared bus, bộ chuyển mạch chéo (crossbar switch) cho phép truyền thông song song giữa nhiều Master và Slave, và bus phân cấp (hierarchical bus) nhằm tổ chức kết nối theo từng tầng, giúp mở rộng và quản lý hệ thống dễ dàng hơn.



Hình 2.3. *Giao thức wishbone point-to-point*

Các tín hiệu chính của Wishbone (Slave) được mô tả dưới bảng dưới đây.

Bảng 2.1. *Các tín hiệu chính của Wishbone*

Tín hiệu	**Mô tả**
Các tín hiệu dùng chung giữa master và slaver	
DAT_I()	Bus dữ liệu đầu vào từ SLAVE đến MASTER; kích thước tối đa 64bit, phụ thuộc thiết kế IP
DAT_O()	Bus dữ liệu đầu ra từ MASTER đến SLAVE; hỗ trợ truyền dữ liệu song song theo độ rộng cổng
RST_I	Reset đầu vào cho từng giao diện Wishbone; không ảnh hưởng tới logic ngoại vi
TGD_I()	Bit bổ sung cho dữ liệu vào; có thể chứa thông tin như parity, error detection hoặc timestamp
TGD_O()	Bit bổ sung cho dữ liệu ra
Tín hiệu Master	
ACK_I	Phản hồi từ SLAVE xác nhận giao dịch hoàn tất thành công, kết thúc chu kỳ truyền
ADR_O()	Địa chỉ bus nhị phân do MASTER phát đến SLAVE; phụ thuộc vào độ rộng và kiểu truyền
CYC_O	Báo hiệu một chu kỳ truy cập đang diễn ra; duy trì mức cao trong toàn bộ thời gian giao dịch
ERR_I	Tín hiệu báo lỗi từ SLAVE cho biết giao dịch bị lỗi và không hoàn tất thành công
LOCK_O	MASTER yêu cầu quyền truy cập bus không bị ngắt giữa chừng để đảm bảo tính toàn vẹn giao dịch
RTY_I	Tín hiệu báo SLAVE hiện chưa sẵn sàng; yêu cầu MASTER thực hiện lại chu kỳ sau
SEL_O()	Chỉ định byte dữ liệu hợp lệ trên bus (ví dụ [SEL_O(3)] ứng với byte thứ 4 trong bus 32-bit)
STB_O	Báo hiệu dữ liệu đang được truyền; kích hoạt cùng các tín hiệu điều khiển khác như WE_O, ADR_O
TGA_O()	Định danh cho địa chỉ, chứa thông tin như độ dài địa chỉ hoặc vùng nhớ được bảo vệ
TGC_O()	Tag định danh kiểu chu kỳ như truy xuất cache, RMW, truy cập bộ nhớ,...

WE_O	Báo hiệu đây là chu kỳ ghi nếu mức cao; nếu không thì là chu kỳ đọc
Tín hiệu Slave	
ACK_O	Phản hồi của SLAVE xác nhận đã nhận và xử lý thành công yêu cầu từ MASTER
ADR_I()	Địa chỉ đầu vào từ MASTER, xác định vùng dữ liệu/thiết bị cần truy cập
CYC_I	Báo hiệu MASTER đang thực hiện một chu kỳ truy cập hợp lệ đến SLAVE
ERR_O	SLAVE báo lỗi trong quá trình giao tiếp, chu kỳ sẽ không được hoàn tất
LOCK_I	MASTER đang yêu cầu truy cập độc quyền; SLAVE chỉ phục vụ MASTER này cho đến khi LOCK bị hủy
RTY_O	SLAVE chưa sẵn sàng xử lý yêu cầu; báo cho MASTER lặp lại chu kỳ sau
SEL_I()	Định rõ byte dữ liệu nào đang có giá trị hợp lệ trong truyền ghi/đọc tương ứng
STB_I	Báo hiệu SLAVE đã được chọn để bắt đầu truyền dữ liệu; chỉ khi STB_I được kích hoạt SLAVE mới phản hồi
TGA_I()	Tag chứa thông tin về vùng địa chỉ như loại truy cập, độ dài hoặc cấp bảo vệ
TGC_I()	Tag nhận dạng loại chu kỳ (BLOCK, SINGLE, RMW) phục vụ xử lý nâng cao trong bus
WE_I	Báo hiệu MASTER đang yêu cầu ghi dữ liệu đến SLAVE; nếu không kích hoạt là chu kỳ đọc

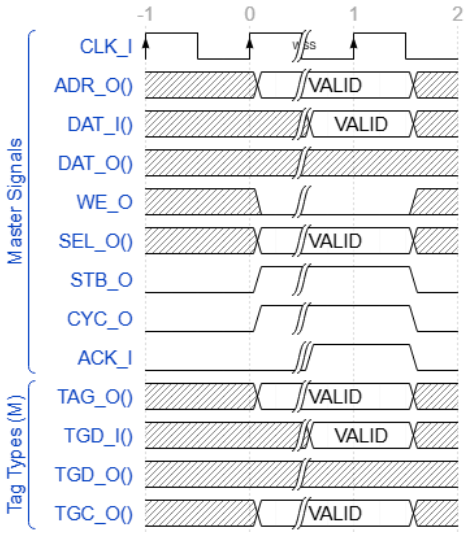
2.4.2. Đọc ghi đơn chu kỳ (single read/ write cycles)

Việc đọc ghi đơn chu kỳ trong giao thức wishbone cho phép truyền dữ liệu một lần tại một thời điểm. Phần dưới đây trình bày tổng quan về cơ chế hoạt động của cơ chế này.

Về quá trình single read cycles (đọc đơn chu kỳ), tức là quá trình cho phép master yêu cầu data từ slave, bao gồm các bước cơ bản sau:

- Khởi đầu (Clock edge 0): Master tiến hành cung cấp địa chỉ hợp lệ [Add_O()] và [Tga_O()], đồng thời đặt We_O về mức thấp để đánh dấu đang trong chu kỳ đọc. Master cũng sẽ kích hoạt [Sel_O()] để chỉ định vị trí mong nhận data, cùng với đó là bật [Stb_O] để bắt đầu giai đoạn đọc.

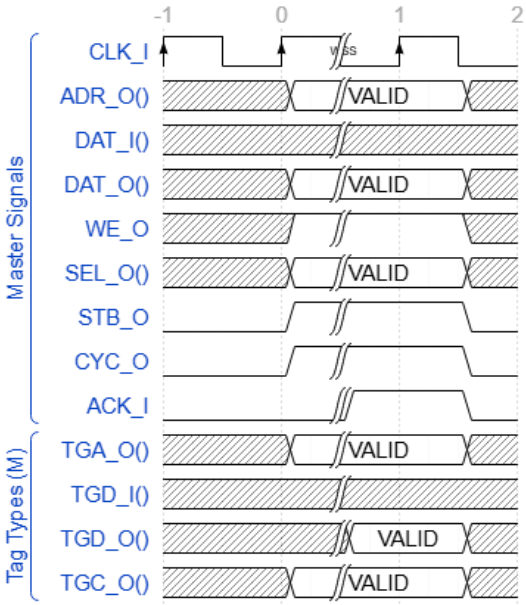
- Thiết lập (Setup, edge 1): Lúc này, slave sẽ tiến hành giải mã input, phản hồi bằng cách gửi lại cho master tín hiệu [Ack_I] và cung cấp dữ liệu hợp lệ trên [Dat_I()], [Tgd_I()].
- Kết thúc (Clock, edge 1): master lưu lại data từ [Dat_I()], [Tgd_I()], sau đó tắt Stb_P để kết thúc chu kỳ đọc. Slave xác nhận điều này bằng cách tắt [Ack_I].



Hình 2.4. *Đọc đơn chu kỳ*

Đối với quá trình single wirte (ghi đơn chu kỳ), tức là quá trình cho phép master truyền data đến slave, cũng bao gồm 3 bước cơ bản như sau:

- Khởi đầu (Clock Edge 0): Master tiến hành cung cấp địa chỉ hợp lệ trên [Adr_O()] và [Tga_O()], đồng thời cũng đưa data hợp lệ vào [Dat_(O)] và [Tgd_O()]. [We_O] lúc này cũng được bật lên để đánh dấu bắt đầu chu kỳ ghi, [Sel_O()] sẽ chỉ định vị trí gửi data, cùng với đó, [Stb_O] được kích hoạt để bắt đầu giai đoạn ghi.
- Thiết lập (settup, edge 1): Lúc này, slave tiến hành giải mã đầu vào, bật [ACK_I] để xác nhận sẵn sàng lưu data.
- Kết thúc (Clock, edge 1): Slave lưu dữ liệu từ [Dat_O()] và [Tgd_O()]. Lúc này master sẽ vô hiệu hóa tín hiệu [Stb_O] để đánh dấu kết thúc chu kỳ ghi, còn slave phản hồi tín hiệu này bằng cách tắt [Ack_I].



Hình 2.5. *Ghi đơn chu kỳ*

CHƯƠNG 3. THIẾT KẾ HỆ THỐNG

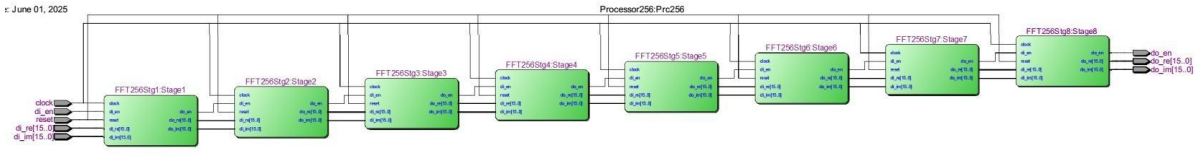
3.1. Tổng quan thiết kế

Hệ thống được xây dựng với mục tiêu:

- Xử lý FFT 256 điểm theo chuẩn radix-2^2 hiệu quả về mặt tài nguyên và tốc độ với pipeline 8 tầng.
- Giao tiếp với hệ thống bên ngoài thông qua chuẩn Wishbone.

Kiến trúc thiết kế này do nhóm xây dựng bao gồm các module chính là:

- Các module từ stage 1 đến stage 8 đại diện cho 8 tầng pipeline
- Module bướm (butterfly) để thực hiện phép cộng trừ giữ 2 số phức
- Module nhân (multiplier) để thực hiện phép nhân giữa 2 số phức
- Module Twiddle256 để lưu trữ các hệ số Twiddle được sử dụng trong quá trình xử lý
- Module delay buffer để lưu trữ tạm thời dữ liệu giữa các giai đoạn trong quá trình xử lý

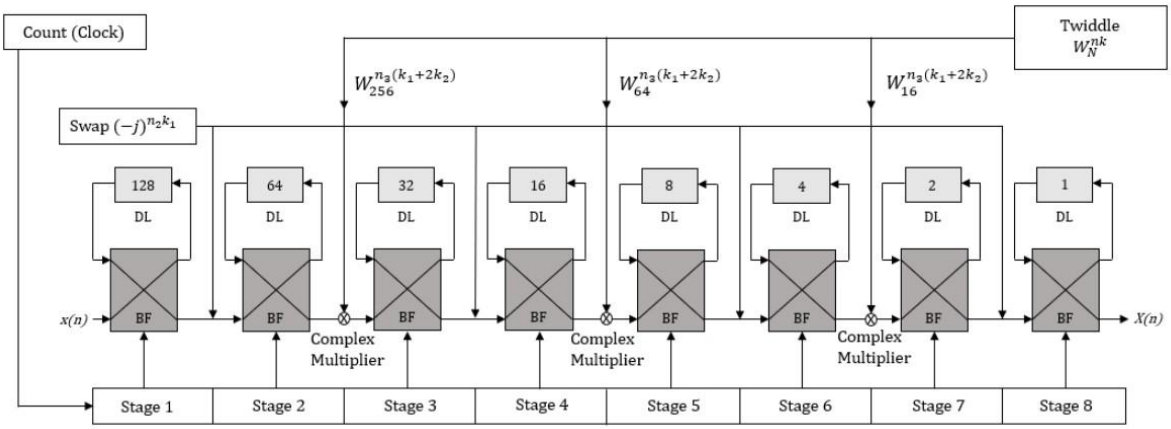


Hình 3.1. *Tổng quan thiết kế FFT 256 với 8 tầng pipeline*

3.2. Dataflow của hệ thống

Trong FFT 256 điểm sử dụng thuật toán Radix-2^2, quá trình tính toán được chia thành 8 giai đoạn, mỗi giai đoạn thực hiện các phép toán cụ thể.

Ở giai đoạn 1, thực hiện phép hoán đổi với biểu thức $(-j)^{n2k1}$. Giai đoạn 2 tính toán với các hệ số Twiddle dựa trên biểu thức $W_{256}{}^{n3(k2+k1)}$, tương ứng với FFT 256 điểm. Sang giai đoạn 3, tiếp tục thực hiện phép hoán đổi $(-j)^{n2k1}$. Giai đoạn 4 sử dụng hệ số Twiddle $W_{64}{}^{n3(k2+k1)}$, tương ứng với FFT 64 điểm. Giai đoạn 5 lại thực hiện phép hoán đổi $(-j)^{n2k1}$, và giai đoạn 6 sử dụng hệ số Twiddle $W_{16}{}^{n3(k2+k1)}$, tương ứng với FFT 16 điểm. Tại giai đoạn 7, tiếp tục hoán đổi với $(-j)^{n2k1}$, và cuối cùng, giai đoạn 8 thực hiện các phép toán dựa trên cấu trúc FFT 4 điểm.



Hình 3.2. *Dataflow cho bộ xử lý FFT Radix-2^2 với 256 điểm*

Trong quá trình tính toán FFT 256 điểm sử dụng thuật toán Radix-2^2, dữ liệu được xử lý tuần tự qua 8 giai đoạn. Cụ thể như sau:

Trong 128 chu kỳ xung clock đầu tiên, các điểm đầu vào từ x_0 đến x_{127} được lần lượt lưu vào khối delay buffer của giai đoạn 1. Trong giai đoạn này chưa có phép tính nào được thực hiện vì khối delay buffer đang trong quá trình nạp dữ liệu.

Tại chu kỳ clock thứ 129, khi điểm x_{128} được đưa vào, khối butterfly ở giai đoạn 1 bắt đầu thực hiện phép cộng và phép trừ giữa x_0 (đã lưu trong khối delay buffer của giai đoạn 1 từ 128 chu kỳ trước) và x_{128}. Kết quả phép cộng được gửi đến khối delay buffer của giai đoạn 2, còn kết quả phép trừ được lưu vào khối delay buffer của giai đoạn 1.

Quy trình này tiếp tục với mỗi điểm đầu vào mới. Ví dụ, tại chu kỳ clock thứ 130, khi x_{129} được đưa vào, khối butterfly của giai đoạn 1 sẽ xử lý x_1 và x_{129}, kết quả phép cộng được lưu vào khối delay buffer của giai đoạn 2, còn kết quả phép trừ được lưu lại vào khối delay buffer trong giai đoạn 1. Quá trình này tiếp diễn đến chu kỳ clock thứ 192.

Đến chu kỳ clock thứ 193, khi x_{192} được đưa vào, khối butterfly của giai đoạn 1 thực hiện phép cộng và trừ giữa x_{64} và x_{192}. Tại thời điểm này, không chỉ giai đoạn 1 mà giai đoạn 2 cũng bắt đầu thực hiện phép tính. Giai đoạn 2 xử lý các kết quả

cộng do butterfly ở giai đoạn 1 tạo ra và đã được lưu trữ trong khối delay buffer của giai đoạn 2. Giai đoạn này tiếp tục thực hiện phép cộng và trừ, trong đó kết quả cộng được chuyển đến khối delay buffer của giai đoạn 3 và kết quả trừ được lưu trong khối delay buffer của giai đoạn 2.

Từ thời điểm này trở đi, mỗi điểm đầu vào mới sẽ kích hoạt tính toán ở nhiều giai đoạn cùng lúc. Quá trình tiếp tục cho đến chu kỳ clock thứ 255. Tại chu kỳ thứ 256, khi điểm đầu vào cuối cùng x_{256} được đưa vào, giai đoạn 1 thực hiện phép cộng và trừ giữa x_{127} và x_{255}. Kết quả cộng được đưa đến khối delay buffer của giai đoạn 2, còn kết quả trừ được lưu lại trong khối delay buffer của giai đoạn 1.

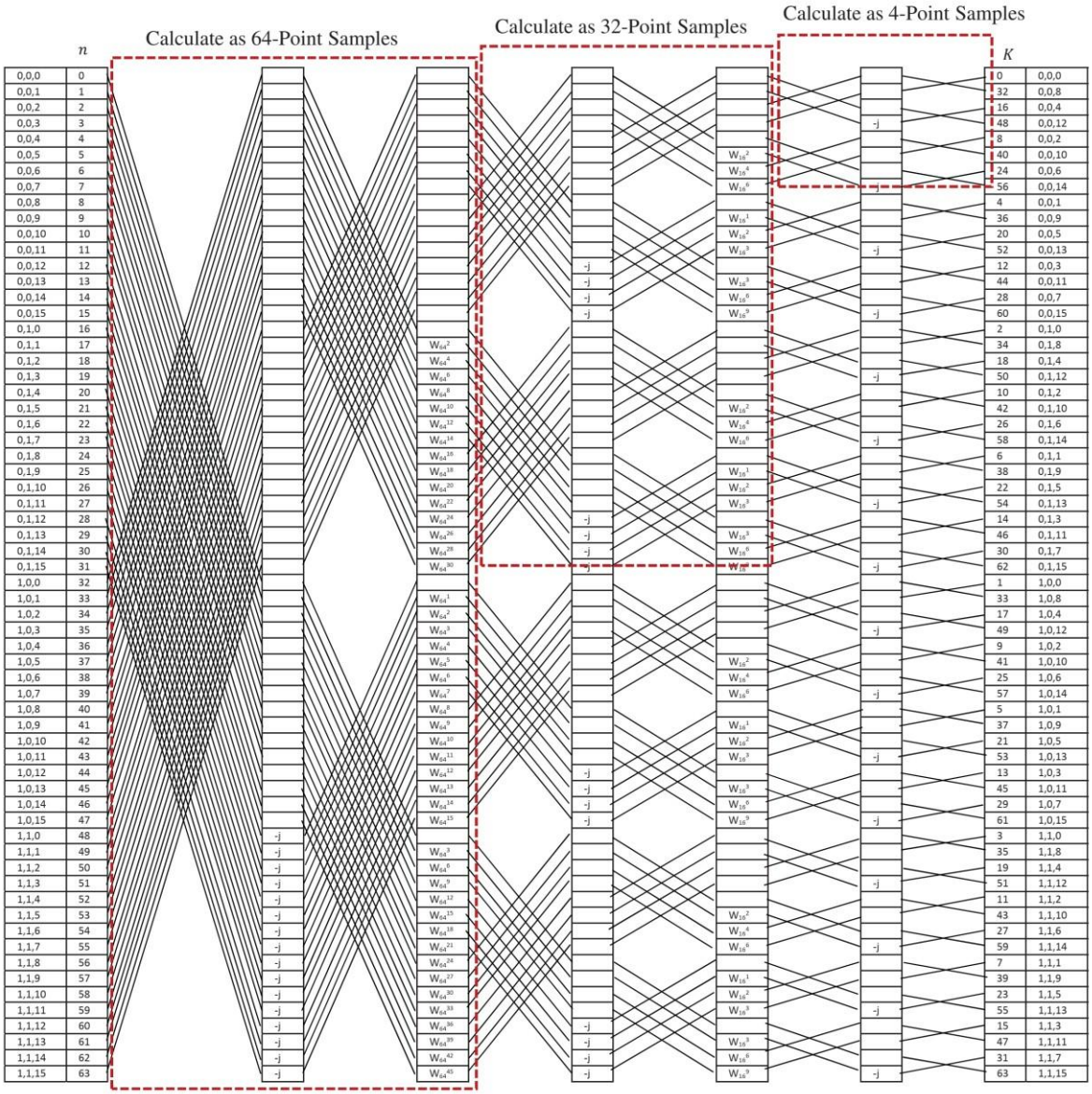
Đồng thời, các giai đoạn còn lại từ giai đoạn 2 đến giai đoạn 8 cũng đồng loạt thực hiện các phép cộng và trừ. Tại mỗi giai đoạn, kết quả cộng được chuyển tiếp sang khối delay buffer của giai đoạn kế tiếp, còn kết quả trừ được lưu lại trong khối delay buffer của giai đoạn hiện tại.

Cuối cùng, kết quả đầu ra từ giai đoạn 8 chính là kết quả FFT hoàn chỉnh của toàn bộ 256 điểm đầu vào.

Bảng 3.1. *Tìm giá trị của các phần tử xử lý tại mỗi vị trí cho bộ xử lý FFT Radix-2²*

PE / N	1^{st} Stage $(-j)^{n_2k_1}$	2^{nd} Stage $W_N^{n_3(k_1+2k_2)}$	3^{rd} Stage $(-j)^{n_2k_1}$	4^{th} Stage $W_N^{n_3(k_1+2k_2)}$	5^{th} Stage $(-j)^{n_2k_1}$	6^{th} Stage $W_N^{n_3(k_1+2k_2)}$	7^{th} Stage $(-j)^{n_2k_1}$	8^{th} Stage $W_N^{n_3(k_1+2k_2)}$
256	$n_1, n_2, k_1, k_2 = 0, 1$ $n_3, k_3 = 0, 1, 2, \ldots, 63$ & N = 256		$n_1, n_2, k_1, k_2 = 0, 1$ $n_3, k_3 = 0, 1, 2, \ldots, 15$ & N = 64 repeating for every next 64 points		$n_1, n_2, k_1, k_2 = 0, 1$ $n_3, k_3 = 0, 1, 2, 3$ & N = 16 repeating for every next 16 points		$n_1, n_2, k_1, k_2 = 0, 1$ $n_3, k_3 = 0$ & N = 4 repeating for every next 4 point	

Các phần tử xử lý (processing elements), phép hoán đổi (swapping) và giá trị hệ số Twiddle được xác định dựa trên các giá trị n_1, n_2, n_3, k_1, k_2, k_3 như trình bày trong bảng trên. Quá trình tính toán tại từng điểm của các phần tử xử lý diễn ra theo sơ đồ trong Hình 3.2 (ở đây, nhóm chỉ để hình minh họa với FFT 64 điểm, tức là từ stage 3 đến stage 8, 2 stage đầu tiên có quá trình tính toán tương tự). Giá trị của các phần tử xử lý, cũng như các phép toán giữa khối butterfly và các phần tử xử lý tại từng vị trí, đều được điều khiển bằng tín hiệu điều khiển theo xung clock.



Hình 3.3. *Đồ thị đơn luồng cho bộ xử lý FFT 64 điểm sử dụng radix-2^2*

3.3. Các module verilog chính

3.3.1. Delay buffer

Module Delay Buffer dùng để lưu trữ tạm thời dữ liệu giữa các giai đoạn trong quá trình xử lý FFT, module này hoạt động đồng bộ với xung clock và sử dụng cơ chế lưu trữ tương tự như hàng đợi FIFO. Trong thiết kế FFT 256 điểm, số lượng giá trị mà Delay Buffer lưu trữ ở từng giai đoạn giảm dần theo thứ tự: 128, 64, 32, 16, 8, 4, 2 và 1.

Hình 3.4. *Module delay buffer*

Trong module này, nhóm sử dụng hai shift registers riêng biệt để lưu phần thực (di_re) và phần ảo (di_im) của tín hiệu đầu vào phức. Tại mỗi chu kỳ xung nhịp (clock), dữ liệu trong bộ đệm được dịch lên một vị trí, và giá trị mới được ghi vào vị trí đầu tiên. Sau 32 chu kỳ, dữ liệu tại đầu vào sẽ xuất hiện tại đầu ra (do_re, do_im), đảm bảo độ trễ chính xác.

3.3.2. Butterfly

Module Butterfly thực hiện phép toán cộng, trừ giữa 2 số phức đầu vào x_0 và x_1. Kết quả đầu ra của module bao gồm y_0 là tổng của x_0 và x_1, y_1 là hiệu của $x_0$và x_1.

Hình 3.5. *Module butterfly*

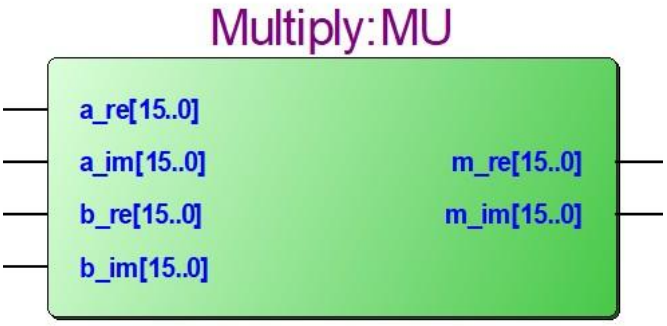
3.3.3. Twiddle256

Module Twiddle256 dùng để lưu trữ các hệ số Twiddle được sử dụng trong quá trình xử lý FFT. Đầu vào của module là địa chỉ 8 bit (giá trị từ 0 đến 255), và đầu ra là hệ số Twiddle tương ứng với địa chỉ đó. Ở đây, các hệ số Twiddle được tính trước và lưu trong ROM, đọc đồng bộ với xung clock dựa trên bộ đếm nhị phân 8-bit.

Hình 3.6. *Module twiddle 256*

3.3.4. Multiplier

Module Multiply thực hiện phép nhân giữa hai số phức đầu vào a và b. Đầu ra của module là tích của hai số phức này.



Hình 3.7. *Module Multiply*

Cụ thể, với hai số phức đầu vào có dạng $a + bi$ và $c + di$, module này tính toán kết quả nhân phức theo công thức:

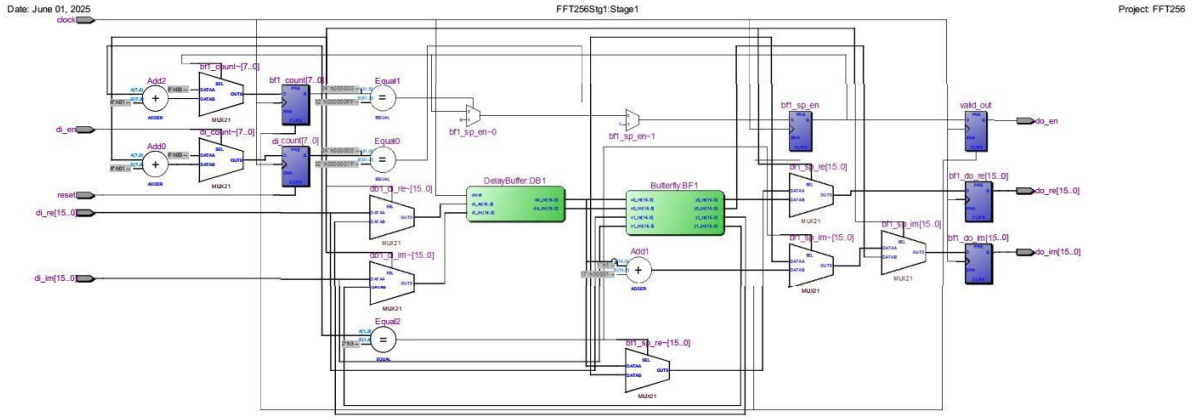
$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$

Trong quá trình xử lý, bốn phép nhân có dấu được thực hiện độc lập: $a \cdot c$, $a \cdot d$, $b \cdot c$ và $b \cdot d$. Sau đó, các kết quả trung gian này được chia tỷ lệ bằng phép dịch phải để giữ nguyên độ rộng dữ liệu và giới hạn sai số số học trong hệ thống fixedpoint.

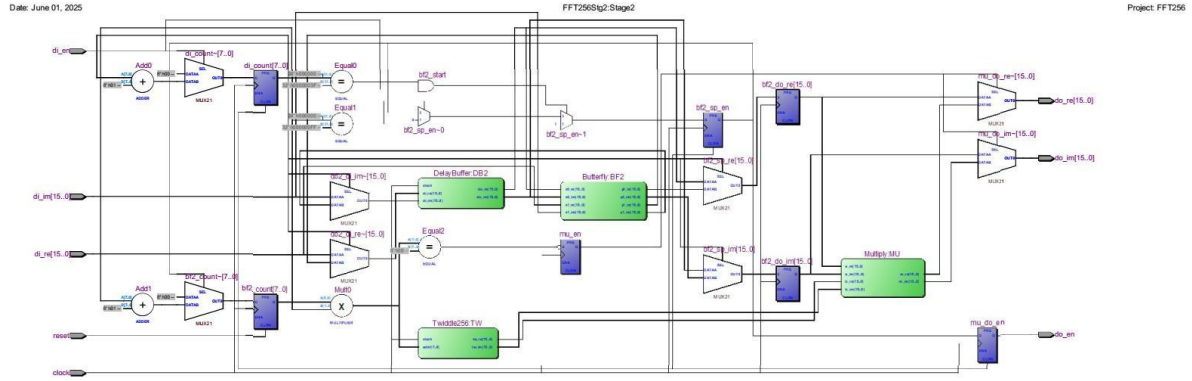
Phần thực của kết quả (m_re) được tính bằng hiệu của $a.\ c$ và $b.\ d$ trong khi phần ảo (m_im) là tổng của $a.\ d$ và $b.\ c$

3.3.5. Các module của từng stage

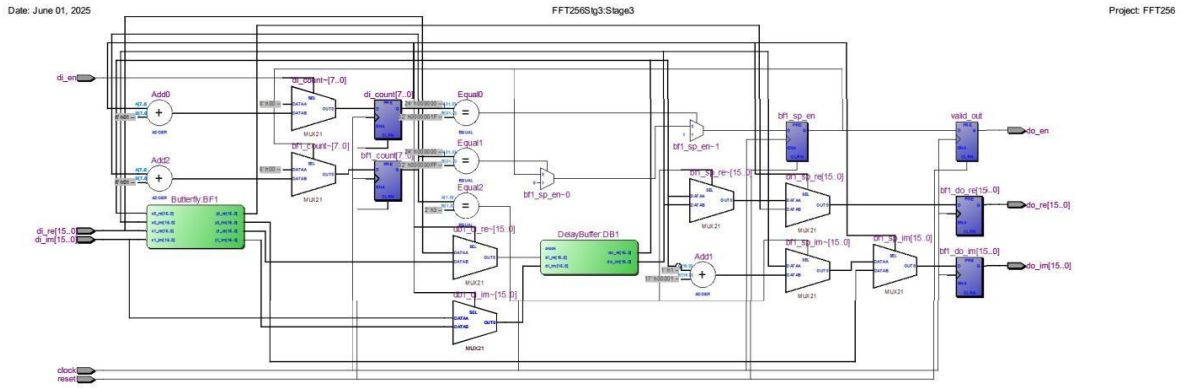
Dựa trên dataflow đã được mô tả chi tiết ở phần trước, nhóm đã tiến hành thiết kế và mô tả hành vi cho các module. Trong mục này, nhóm trình bày kết quả của quá trình thiết kế mô tả hành vi dưới dạng sơ đồ RTL netlist, thể hiện cấu trúc và mối liên kết giữa các thành phần của hệ thống.



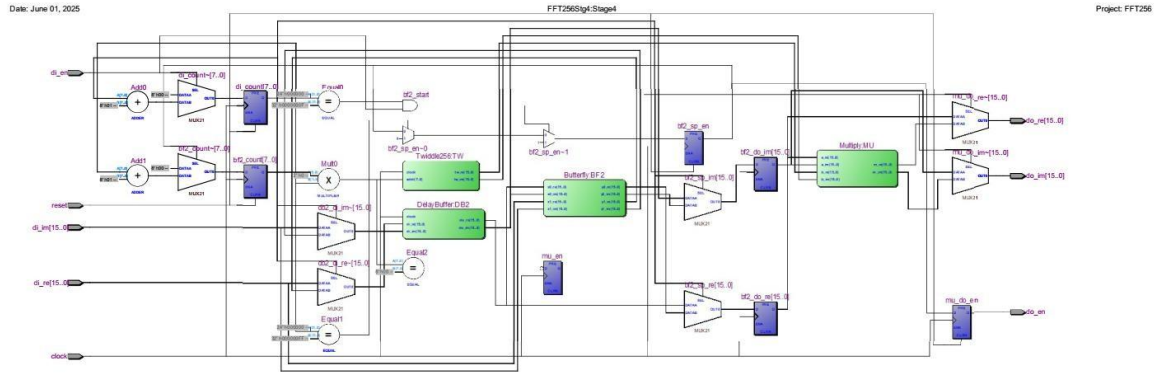
Hình 3.8. *RTL netlist của stage 1*



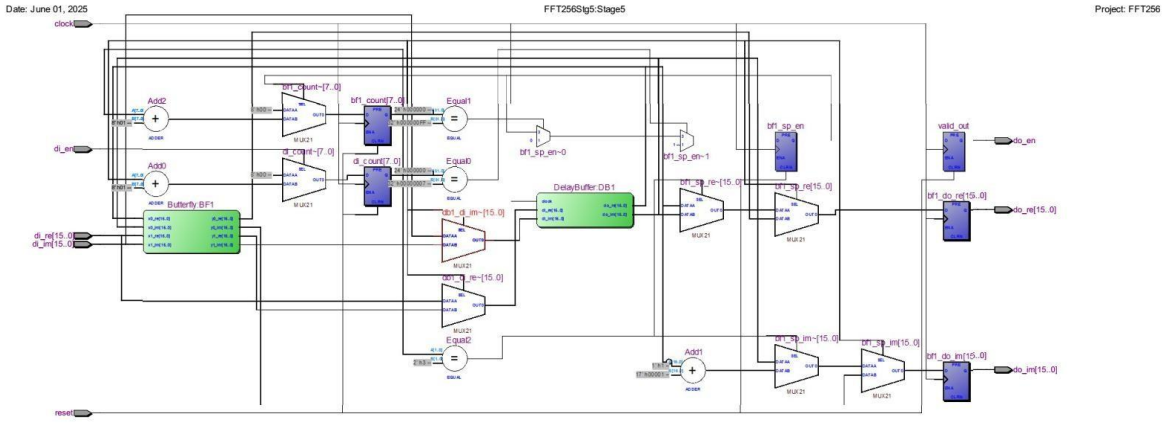
Hình 3.9. *RTL netlist của stage 2*



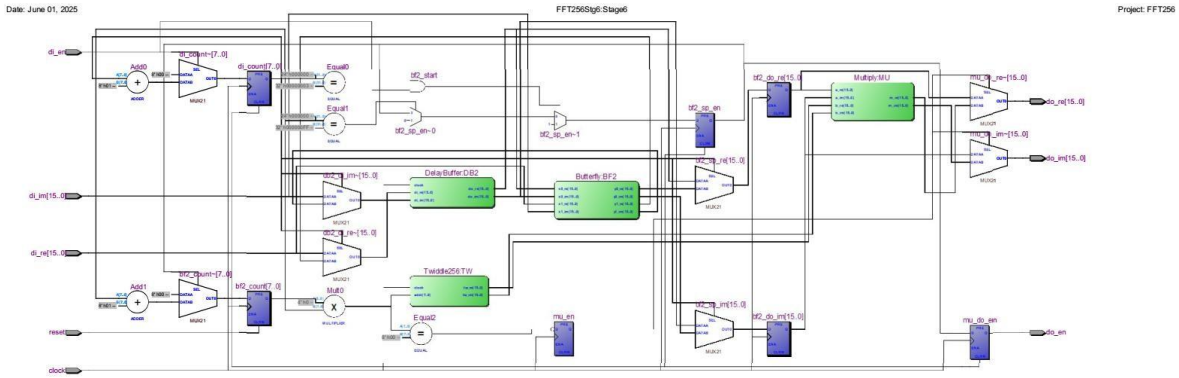
Hình 3.10. *RTL netlist của stage 3*



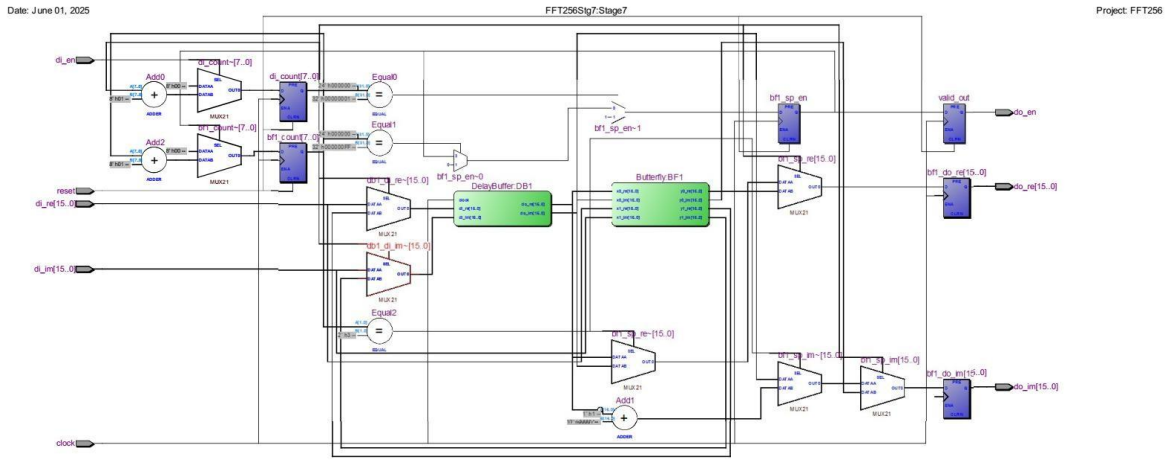
Hình 3.11. *RTL netlist của stage 4*



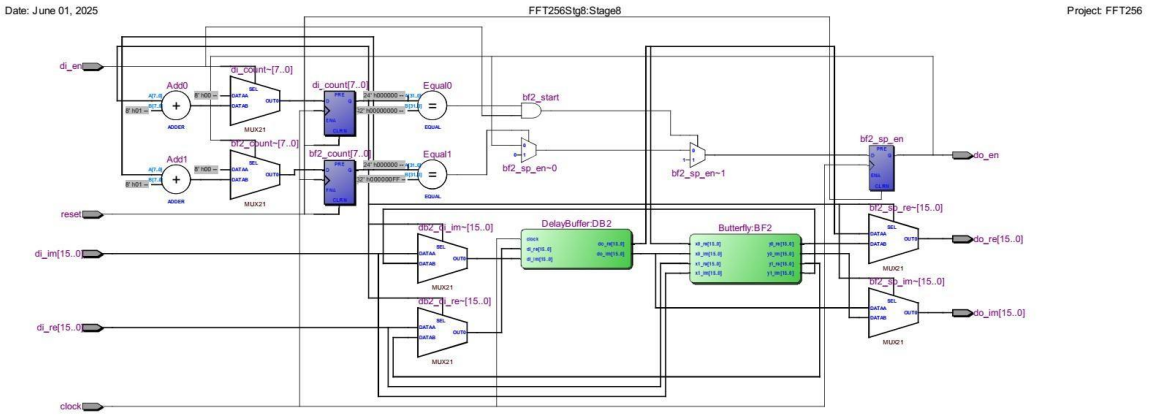
Hình 3.12. *RTL netlist của stage 5*



Hình 3.13. *RTL netlist của stage 6*



Hình 3.14. *RTL netlist của stage 7*



Hình 3.15. *RTL netlist của stage 8*

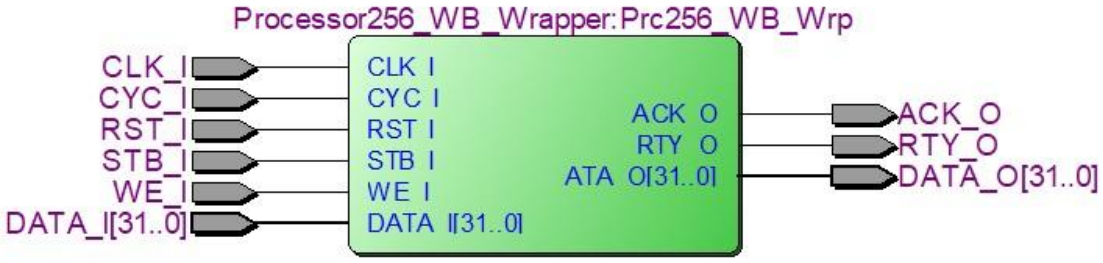
3.4. Xây dựng theo chuẩn giao tiếp Wishbone

Với cách thiết kế ngõ vào và ngõ ra của top module xử lý DFT 256 điểm, module wrapper được xây dựng để kết nối top module này với hệ thống thông qua chuẩn giao tiếp Wishbone. Mục tiêu của module bọc là hỗ trợ hai thao tác ghi dữ liệu đọc dữ liệu theo chuẩn giao tiếp Wishbone.

Module wrapper sử dụng các tín hiệu chuẩn trong giao thức Wishbone như sau:

- CLK_I: Tín hiệu xung clock.
- RST_I: Tín hiệu reset tích cực cao, dùng để đưa hệ thống về trạng thái khởi tạo.
- DATA_I: Bus dữ liệu đầu vào có độ rộng 32 bit, được sử dụng trong các thao tác ghi dữ liệu từ master.
- DATA_O: Bus dữ liệu đầu ra có độ rộng 32 bit, chứa dữ liệu phản hồi từ module khi thực hiện thao tác đọc.

- CYC_I: Tín hiệu cho biết master đang khởi tạo một giao dịch hợp lệ. Khi được kích hoạt, module sẽ sẵn sàng tiếp nhận hoặc trả dữ liệu.
- STB_I: Tín hiệu cho biết module đang được chọn để bắt đầu truyền dữ liệu.
- WE_I: Tín hiệu điều khiển thao tác đọc/ghi. Khi WE_I ở mức cao, master yêu cầu ghi dữ liệu vào module. Ngược lại, khi ở mức thấp, master thực hiện thao tác đọc dữ liệu.
- ACK_O: Tín hiệu phản hồi từ slave, cho biết thao tác ghi hoặc đọc đã được xử lý thành công.
- RTY_O: Tín hiệu phản hồi yêu cầu master thử lại trong các chu kỳ sau, được sử dụng khi master yêu cầu đọc dữ liệu chưa được tính toán xong.



Hình 3.16. *Top module được bọc theo chuẩn Wishbone*

Bên trong module wrapper, ngoài việc khai báo và sử dụng các thanh ghi và tín hiệu phục vụ cho việc giao tiếp với top module, module wrapper còn sử dụng thêm ba thanh ghi trung gian là out_reg, ack_reg và rty_reg. Các thanh ghi này lần lượt được dùng để điều khiển giá trị của các ngõ ra tương ứng là DATA_O, ACK_O và RTY_O.

```
reg     [31:0]   out_reg;
reg              ack_reg;
reg              rty_reg;

reg              di_en;
reg     [15:0]   di_re;
reg     [15:0]   di_im;

wire             do_en;
wire    [15:0]   do_re;
wire    [15:0]   do_im;

assign DATA_O = out_reg;
assign ACK_O  = ack_reg;
assign RTY_O  = rty_reg;
```

Hình 3.17. *Các thanh ghi nội bộ và tín hiệu được sử dụng trong module wrapper*

```
Processor256 #(.WIDTH(16))
    Prc256_u (
        .clock(CLK_I),
        .reset(RST_I),
        .di_en(di_en),
        .di_re(di_re),
        .di_im(di_im),
        .do_en(do_en),
        .do_re(do_re),
        .do_im(do_im)
    );
```

Hình 3.18. *Top module được khởi tạo trong module wrapper*

Khi master thực hiện thao tác đọc hoặc ghi dữ liệu thông qua giao thức Wishbone, hai tín hiệu điều khiển CYC_I và STB_I cần phải được duy trì ở mức cao để xác định rằng một giao dịch hợp lệ đang diễn ra.

Nếu đây là thao tác ghi (được xác định khi tín hiệu WE_I ở mức cao), module sẽ kích hoạt di_en và trích dữ liệu từ DATA_I: 16 bit thấp được gán vào di_re, 16 bit cao vào di_im. Đồng thời, ack_reg được đặt lên 1 để phát tín hiệu xác nhận ACK_O, báo hiệu thao tác ghi đã hoàn tất thành công.

```
if (CYC_I && STB_I) begin
    if (WE_I) begin
            di_en <= 1'b1;
            di_re <= DATA_I[15:0];
            di_im <= DATA_I[31:16];
            ack_reg <= 1'b1;
    end
```

Hình 3.19. *Logic thực hiện thao tác ghi trong module wrapper*

Ngược lại, nếu là thao tác đọc (WE_I = 0), module sẽ kiểm tra tín hiệu do_en để xác định liệu top module đã có kết quả đầu ra. Nếu do_en = 1, dữ liệu từ do_im và do_re sẽ được ghép lại và gán vào thanh ghi out_reg, đồng thời ack_reg được đặt lên 1 để xác nhận thao tác đọc thành công. Ngược lại, nếu do_en = 0, tức dữ liệu chưa sẵn sàng, module sẽ gán rty_reg = 1 để báo hiệu master cần thử lại ở chu kỳ sau.

```
else begin
    if (do_en) begin
        out_reg <= {do_im, do_re};
        ack_reg <= 1'b1;
    end else begin
        rty_reg <= 1'b1;
    end
end
```

Hình 3.20. *Logic thực hiện thao tác đọc trong module wrapper*

3.5. Xây dựng hệ thống SoC để nạp kit DE2

Đối với đề tài này, nhóm đã tiến hành xây dựng bộ biến đổi tín hiệu DFT theo chuẩn giao tiếp wishbone như mục tiêu ban đầu đề ra nhằm đảm bảo khả năng tương thích và giao tiếp hiệu quả với các thành phần khác trong hệ thống. Tuy nhiên, để tích hợp và triển khai trên kit FPGA, tận dụng công cụ Qsys của Quartus, nhóm đã mở rộng thiết kế bằng cách bọc thêm giao thức Avalon, do Qsys hỗ trợ chuẩn này một cách trực tiếp và thuận tiện.

Việc xây dựng hệ thống SoC là điều hợp lý vì với thiết kế DFT 256 điểm này, việc sử dụng top module FFT256point với cơ chế điều khiển clock bằng switch vật lý không phải là giải pháp tối ưu. Cụ thể, nhập thủ công 256 giá trị đầu vào thông qua switch và quan sát đầu ra bằng cách thay đổi trạng thái switch không chỉ tốn thời gian mà còn khó khăn trong việc đánh giá và so sánh kết quả. Thay vào đó, nhóm tận dụng Qsys để tích hợp lõi DFT vào một kiến trúc SoC, sau đó viết software tích hợp để điều khiển. Giải pháp này hỗ trợ xuất kết quả đầu ra trực tiếp qua console, tạo điều kiện thuận lợi cho việc so sánh và xác minh tính chính xác của kết quả DFT. Cách tiếp cận này không chỉ nâng cao hiệu quả triển khai mà còn đảm bảo tính linh hoạt và độ tin cậy trong quá trình kiểm tra và đánh giá hệ thống.

3.5.1. Bọc thiết kế DFT bằng chuẩn avalon

Để bọc thiết kế bằng chuẩn avalon, nhóm đã tiến hành triển khai các module bộ đệm đầu vào và đầu ra (InBur256 và OutBur256) theo chuẩn này. Sau đó tiến hành hiện thực logic của top module DFT 256 điểm (Processor256) này cùng với các bộ đệm vừa nêu trên theo đúng chuẩn giao tiếp avalon. Dưới dây là mô tả các module dùng để bọc.

Đầu tiên, đối với module bộ đệm đầu vào (InBuf256), module này được thiết kế để lưu trữ 256 điểm data đầu vào, với các tín hiệu chính bao gồm:

- Clk: tín hiệu xung clk
- addr [7:0]: Địa chỉ 8 bit, xác định vị trí dữ liệu trong bộ nhớ.
- read_en: Tín hiệu kích hoạt đọc, cho phép truy xuất dữ liệu.
- dout [31:0]: Dữ liệu đầu ra 32 bit, kết hợp phần thực (15:0) và phần ảo (31:16).

Module này sử dụng mảng bộ nhớ mem [0:255] với mỗi phần tử 32 bit để lưu trữ dữ liệu mẫu. Trong giai đoạn khởi tạo, bộ nhớ được nạp sẵn các giá trị từ 32'h00000080 đến 32'h00007FFF, đại diện cho tín hiệu đầu vào để kiểm tra và mô phỏng. Khi read_en được kích hoạt, module xuất dữ liệu tại vị trí addr qua dout, cung cấp luồng dữ liệu liên tục cho lõi DFT.

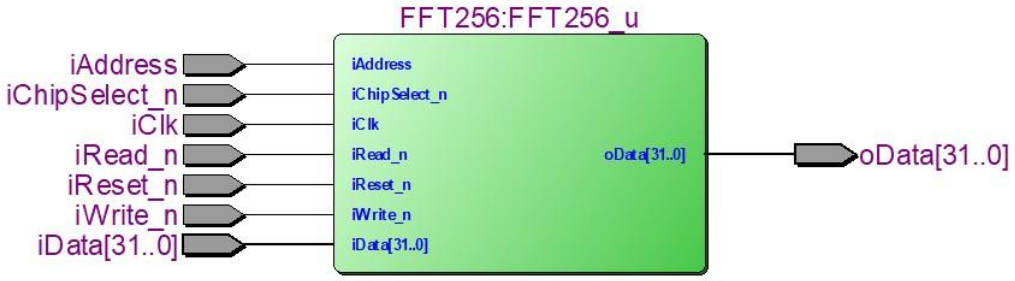
Tiếp theo, đối với module bộ đệm đầu ra (OutBuf256), module này chịu trách nhiệm lưu trữ 256 điểm kết quả tính toán từ Processor256. Các tín hiệu chính bao gồm:

- clk, reset_n: Clock và reset để đồng bộ và khởi tạo module.
- write_en: Tín hiệu từ Processor256 cho phép ghi dữ liệu đầu ra.
- re_in [15:0], im_in [15:0]: Dữ liệu phần thực và phần ảo từ Processor256 .
- done: Tín hiệu báo hoàn thành việc ghi 256 điểm
- dout [31:0]: Dữ liệu đầu ra 32 bit, kết hợp phần ảo (31:16) và phần thực (15:0).
- addr [7:0]: Địa chỉ 8 bit để software có truy cập kết quả.

Module này cũng sử dụng mảng bộ nhớ mem [0:255] để lưu trữ kết quả. Khi write_en được kích hoạt, dữ liệu từ re_in và im_in được ghi vào mem tại vị trí write_ptr, với con trỏ tăng dần. Khi write_ptr đạt 255 (tức là đã ghi đủ 256 điểm), tín hiệu done được khẳng định, báo hiệu hoàn tất quá trình ghi. Software sau đó có thể đọc dữ liệu qua dout bằng cách chỉ định addr.

Cuối cùng, module chính được bọc là FFT256, đóng vai trò trung tâm. Module này bao gồm các tín hiệu chuẩn của avalon như sau:

- iClk: Tín hiệu xung clock của hệ thống.
- iReset_n: Tín hiệu reset mức thấp, khởi tạo lại các thanh ghi và trạng thái khi được kích hoạt.
- iChipSelect_n, iWrite_n, iRead_n: Tín hiệu điều khiển chuẩn Avalon, lần lượt xác định lựa chọn module, thao tác ghi và thao tác đọc.
- iAddress [8:0]: Địa chỉ 9 bit, xác định vị trí truy cập dữ liệu.
- iData[31:0] và oData[31:0] là dữ liệu vào ra với 32 bit.



Hình 3.21. *Top module được bọc theo chuẩn Avalon*

Bên trong, module này sử dụng các thanh ghi và tín hiệu như input_wr_ptr, output_rd_ptr, fft_ready và fft_count để quản lý luồng dữ liệu. Quá trình hoạt động bao gồm:

(1) Khi tín hiệu iData = 1 với iChipSelect_n và iWrite_n ở mức thấp, module kích hoạt trạng thái fft_ready và bật read_en để bắt đầu nạp data từ bộ đệm đầu vào.

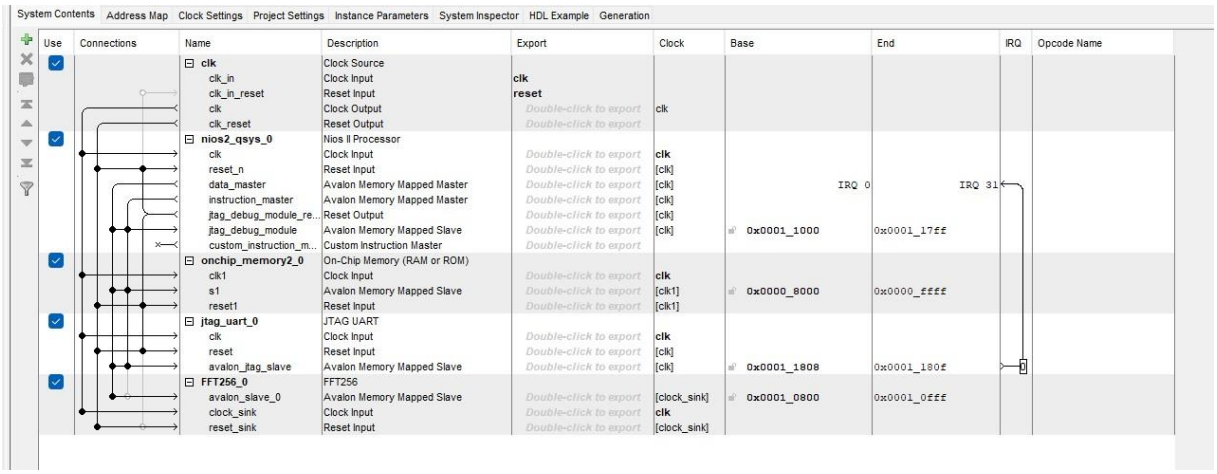
(2) Data được nạp một cách tuần tự vào Processor256 thông qua các tín hiệu di_re (phần thực) và di_im (phần ảo), với con trỏ input_wr_ptr tăng dần để truy cập 256 điểm.

(3) Sau khi xử lý 256 điểm (fft_count = 256), module ngừng nạp dữ liệu và chờ kết quả từ bộ đệm đầu ra.

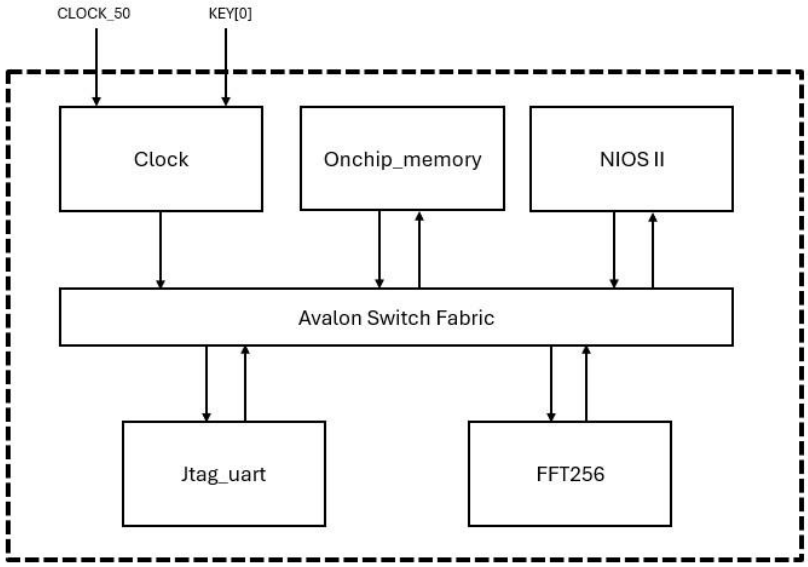
(4) Khi iChipSelect_n và iRead_n ở mức thấp, module trả về dữ liệu từ bộ đệm đầu ra hoặc trạng thái fft_done qua oData.

3.5.2. Xây dựng hệ thống SoC với IP FFT256 tự thiết kế

Sau khi tiến hành bọc top module theo chuẩn giao tiếp avalon, nhóm tiến hành cấu hình IP tự thiết kế này và tiến hành tích hợp vào hệ thống SoC với công cụ Qsys của Quatus.



Hình 3.22. *Kết nối hệ thống trên công cụ Qsys*



Hình 3.23. *Hệ thống SoC tính DFT 256 điểm*

3.5.3. Viết code software để điều khiển

Để tích hợp việc điều khiển, nhóm đã tiến hành xây dựng software trên công cụ Nios II Software Build Tools for Eclipse của Quartus. Điều này giúp kích hoạt việc tính toán DFT, và trực quan kết quả trên console để có thể kiểm tra tính chính xác một cách dễ dàng.

Trong phần này, nhóm đã xây dựng thêm một hàm để đảo ngược bit của chỉ số (bit_reverse), điều này là cần thiết vì cần phải sawpx xếp lại kết quả đầu ra của thuật toán DFT.

```c
#include "stdio.h"
#include "stdint.h"
#include "io.h"
#include "system.h"

unsigned int bit_reverse(unsigned int x, int bits) {
    unsigned int r = 0;
    int i;
    for (i = 0; i < bits; i++) {
        r = (r << 1) | (x & 1);
        x >>= 1;
    }
    return r;
}

int main() {
    uint32_t begin, done;
    uint32_t raw_output[256];
    uint32_t reordered_output[256];

    begin = 0x00000001;
    IOWR(FFT256_0_BASE, 0, begin);

    do {
        done = IORD(FFT256_0_BASE, 256);
    } while (!(done & 0x1));

    int i;
    for (i = 0; i < 256; i++) {
        raw_output[i] = IORD(FFT256_0_BASE, i);
    }

    for (i = 0; i < 256; i++) {
        unsigned int rev_idx = bit_reverse(i, 8);
        reordered_output[rev_idx] = raw_output[i];
    }

    printf("Processed FFT Output (Q1.15 Hex Format):\n");
    for (i = 0; i < 256; i++) {
        uint32_t raw = reordered_output[i];
        uint16_t real_q15 = raw & 0xFFFF;         // lower 16 bits
        uint16_t imag_q15 = (raw >> 16) & 0xFFFF; // upper 16 bits

        printf("X(%3d) =>  Real: 0x%04X, Imag: 0x%04X\n", i, real_q15, imag_q15);
    }

    return 0;
}
```

Hình 3.24. *Code sofeware cho hệ thống SoC DFT 256 điểm*

Đầu tiên, tiến hành gán giá trị địa chỉ cơ sở cho module FFT256, hay nói cách khác, thao tác này kích hoạt việc tính toán DFT, bắt đầu xử lý 256 điểm dữ liệu từ bộ đệm đầu vào. Cùng với đó, liên tục đọc tín hiệu hoàn tất done từ offset 256, chờ bit thấp nhất được bật lên để xác nhận quá trình tính toán DFT đã kết thúc.

Sau khi tính toán DFT hoàn tất, đọc 256 giá trị đầu ra 32 bit từ phần cứng, lưu vào mảng raw_output. Mỗi giá trị chứa phần thực (16 bit thấp) và phần ảo (16 bit cao) của kết quả. Để đáp ứng yêu cầu của thuật toán FFT, các chỉ số được đảo ngược bit bằng hàm bit_reverse, sau đó dữ liệu được lưu vào mảng reordered_output theo thứ tự mới. Cuối cùng, chương trình trích xuất phần thực và phần ảo từ mỗi giá trị, hiển thị qua console ở định dạng hex Q1.15, giúp dễ dàng quan sát và so sánh kết quả với giá trị mong đợi.

CHƯƠNG 4. MÔ PHỎNG THIẾT KẾ

4.1. Thiết kế testbench

Với số lượng điểm đầu vào lên đến 256, việc cung cấp giá trị dữ liệu đầu vào và quan sát kết quả đầu ra qua waveform sẽ trở nên rất bất tiện và dễ sai sót. Ngoài ra, do thiết kế DFT sử dụng thuật toán radix-2^2, thứ tự các điểm đầu ra sẽ bị đảo thứ tự. Điều này gây khó khăn đáng kể trong việc đánh giá và đối chiếu kết quả đầu ra.

Để khắc phục những hạn chế trên, testbench được thiết kế nhằm giúp người dùng dễ dàng đưa dữ liệu vào và lấy dữ liệu ra từ thiết kế thông qua các file .txt. Cách tiếp cận này khá thuận tiện, vì dữ liệu đầu vào có thể dễ dàng tạo bằng các ngôn ngữ lập trình cấp cao, và dữ liệu đầu ra cũng có thể được xử lý, đối chiếu dễ dàng hơn.

4.1.1. Testbench của thiết kế FFT256 điểm.

```
reg         clock;
reg         reset;
reg         di_en;
reg [15:0]  di_re;
reg [15:0]  di_im;

wire        do_en;
wire[15:0]  do_re;
wire[15:0]  do_im;

reg [15:0]  imem[0:511];
reg [15:0]  omem[0:511];
```

Ngoài các tín hiệu kết nối trực tiếp với top module như *clock*, *reset*, *di_en*, *di_re*, *di_im*, *do_en*, *do_re* và *do_im*, *testbench* còn sử dụng thêm hai bộ nhớ tạm là *imem* và *omem*. *imem* được dùng để lưu trữ dữ liệu đầu vào được đọc từ file, còn *omem* lưu lại dữ liệu đầu ra từ thiết kế trước khi ghi ra file.

```
task LoadInputData;
    input[80*8:1]   filename;
begin
    $readmemh(filename, imem);
end
endtask
```

Vì dữ liệu đầu vào của thiết kế được lưu trong một file .txt, testbench sử dụng tác vụ LoadInputData để đọc nội dung từ file và nạp vào bộ nhớ imem.

```
task GenerateInputWave;
    integer n;
begin
    di_en <= 1;
    for (n = 0; n < 256; n = n + 1) begin
        di_re <= imem[2*n];
        di_im <= imem[2*n+1];
        @(posedge clock);
    end
    di_en <= 0;
    di_re <= 'bx;
    di_im <= 'bx;
end
endtask
```

Sau khi dữ liệu được nạp từ file .txt vào bộ nhớ tạm imem, testbench sẽ sử dụng tác vụ GenerateInputWave để tuần tự đưa toàn bộ dữ liệu đầu vào vào top module. Trong tác vụ này, mỗi cặp giá trị thực (di_re) và ảo (di_im) sẽ được lấy từ imem và phát theo từng chu kỳ xung clock, đồng thời kích hoạt tín hiệu di_en để báo hiệu dữ liệu đầu vào hợp lệ.

```
task SaveOutputData;
    input[80*8:1]   filename;
    integer         fp, n, m;
begin
    fp = $fopen(filename);
    m = 0;
    for (n = 0; n < 256; n = n + 1) begin
        m[7] = n[0];
        m[6] = n[1];
        m[5] = n[2];
        m[4] = n[3];
        m[3] = n[4];
        m[2] = n[5];
        m[1] = n[6];
        m[0] = n[7];
        $fdisplay(fp, "%h  %h  // %d", omem[2*m], omem[2*m+1], n[7:0]);
    end
    $fclose(fp);
end
endtask
```

Bên cạnh việc đọc dữ liệu đầu vào, testbench còn sử dụng tác vụ SaveOutputData để ghi kết quả đầu ra vào file .txt. Do thuật toán radix-2^2 được dùng trong quá trình tính toán DFT nên thứ tự các điểm đầu ra bị đảo theo quy tắc bit-reversal. Vì vậy, trong tác vụ này, testbench sẽ thực hiện việc ánh xạ lại thứ tự các điểm từ dạng bit-reversed về thứ tự tuyến tính ban đầu trước khi ghi ra file.

```
Processor256 DUT (
    .clock  (clock  ),  //  i
    .reset  (reset  ),  //  i
    .di_en  (di_en  ),  //  i
    .di_re  (di_re  ),  //  i
    .di_im  (di_im  ),  //  i
    .do_en  (do_en  ),  //  o
    .do_re  (do_re  ),  //  o
    .do_im  (do_im  )   //  o
);
```

Trong testbench, module Processor256 được khởi tạo dưới tên DUT và kết nối trực tiếp với các tín hiệu điều khiển và dữ liệu đã được khai báo ở phần đầu.

```
initial begin : OCAP
    integer     n;
    forever begin
        n = 0;
        while (do_en !== 1) @(negedge clock);
        while ((do_en == 1) && (n < 256)) begin
            omem[2*n  ] = do_re;
            omem[2*n+1] = do_im;
            n = n + 1;
            @(negedge clock);
        end
    end
end
```

Khối initial OCAP trong testbench có chức năng thu nhận dữ liệu từ ngõ ra của thiết kế và lưu vào bộ nhớ omem. Quá trình capture dữ liệu bắt đầu khi tín hiệu do_en từ mức 0 chuyển sang 1, tức là thời điểm thiết kế bắt đầu trả giá trị của các điểm ngõ ra. Từ thời điểm đó, testbench sẽ ghi lần lượt các giá trị phần thực và phần ảo của đầu ra vào omem tại mỗi cạnh xuống của xung clock , cho đến khi đủ 256 điểm.

```
initial begin : STIM
    wait (reset == 1);
    wait (reset == 0);
    repeat(10) @(posedge clock);

    fork
        begin
            LoadInputData("input256.txt");
            GenerateInputWave;
        end

        begin
            wait (do_en == 1);
            repeat(256) @(posedge clock);
            SaveOutputData("output256.txt");
            @(negedge clock);
        end
    join

    repeat(10) @(posedge clock);
    $finish;
end
initial begin : TIMEOUT
    repeat(10000) #20;  //  1000 Clock Cycle Time
    $display("[FAILED] Simulation timed out.");
    $finish;
end
```

Khối initial STIM đóng vai trò là phần điều khiển chính cho quá trình mô phỏng trong testbench. Đầu tiên, nó thực hiện thao tác reset để đưa hệ thống về trạng thái khởi tạo mặc định. Sau khi reset hoàn tất, testbench tiến hành nạp dữ liệu đầu vào bằng cách gọi tác vụ LoadInputData với file "input256.txt", chứa dữ liệu của 256 điểm đầu vào cho thiết kế.

Khi tín hiệu do_en được kích lên mức cao, điều đó cho thấy thiết kế đã bắt đầu xuất kết quả. Lúc này, testbench gọi tác vụ SaveOutputData để thu nhận và lưu lại

256 giá trị đầu ra vào file "output256.txt". Sau khi toàn bộ dữ liệu đầu ra đã được ghi lại, testbench chờ thêm một vài chu kỳ đồng hồ để đảm bảo quá trình xử lý hoàn tất rồi kết thúc mô phỏng.

4.1.2. Testbench cho module đã bọc theo chuẩn Wishbone

Testbench này về cơ bản vẫn giữ nguyên phần lớn các tác vụ và logic kiểm tra giống như thiết kế testbench đã trình bày ở mục trước. Điểm khác biệt chính nằm ở việc DUT trong testbench lần này là top module đã được bọc theo chuẩn giao tiếp Wishbone. Do đó, thay vì sử dụng trực tiếp các tín hiệu điều khiển gốc của top module như trong testbench trước, testbench trong mục này sử dụng các tín hiệu giao tiếp theo chuẩn Wishbone để tương tác với DUT.

```
reg         CLK_I;
reg         RST_I;
reg  [31:0] DATA_I;
wire [31:0] DATA_O;

reg         CYC_I;
reg         STB_I;
reg         WE_I;
wire        ACK_O;
wire        RTY_O;

reg [15:0]  imem[0:511];
reg [15:0]  omem[0:511];
```

Tương tự như testbench đã được thiết kế ở mục trước, testbench trong phần này ngoài việc sử dụng các tín hiệu giao tiếp theo chuẩn Wishbone của DUT, còn sử dụng hai bộ nhớ tạm là imem và omem để lưu trữ dữ liệu đầu vào và đầu ra.

```
task GenerateInputWave;
    integer n;
begin
    for (n = 0; n < 256; n = n + 1) begin
        DATA_I[15:0]  <= imem[2*n];
        DATA_I[31:16] <= imem[2*n+1];
        @(posedge CLK_I)
        WE_I = 1;
        CYC_I = 1;
        STB_I = 1;
    end
    CYC_I = 0;
    STB_I = 0;
    WE_I = 0;
end
endtask
```

So với testbench trước đó, tác vụ GenerateInputWave trong testbench này có thêm bước điều khiển các tín hiệu WE_I, CYC_I và STB_I để thực hiện thao tác ghi dữ liệu từ imem vào top module theo chuẩn Wishbone. Trong suốt quá trình ghi 256 điểm đầu vào, ba tín hiệu này được giữ ở mức cao và sẽ được đưa về mức thấp sau khi ghi hoàn tất.

```
Processor256_WB_Wrapper DUT (
    .CLK_I  (CLK_I),    //  i
    .RST_I  (RST_I),    //  i
    .DATA_I (DATA_I),   //  i
    .DATA_O (DATA_O),   //  i
    .CYC_I  (CYC_I),    //  i
    .STB_I  (STB_I),    //  o
    .WE_I   (WE_I),     //  o
    .ACK_O  (ACK_O),    //  o
    .RTY_O  (RTY_O)
);
```

Tương tự như testbench trước, module Processor256 sau khi được bọc theo chuẩn Wishbone được khởi tạo với tên DUT và được kết nối trực tiếp với các tín hiệu điều khiển và dữ liệu đã khai báo ở phần đầu testbench.

```
//  Output Data Capture
initial begin : OCAP
    integer     n;
    forever begin
        n = 0;
        while ((RTY_O != 1)) @(negedge CLK_I);
        while ((RTY_O != 0)) @(negedge CLK_I);
        while ((n < 256)) begin
            omem[2*n  ] = DATA_O[15:0];
            omem[2*n+1] = DATA_O[31:16];
            n = n + 1;
            @(negedge CLK_I);
        end
    end
end
```

Khối initial OCAP trong testbench này vẫn đảm nhiệm chức năng thu nhận dữ liệu đầu ra như ở testbench trước. Tuy nhiên, thay vì dựa vào tín hiệu di_en để xác định thời điểm DUT bắt đầu xuất kết quả, testbench này sử dụng tín hiệu RTY_O. Cụ thể, vòng lặp while đầu tiên chờ đến khi RTY_O chuyển sang mức cao, báo hiệu quá trình ghi dữ liệu đầu vào đã kết thúc. Vòng lặp tiếp theo chờ RTY_O trở về mức thấp, cho biết DUT đã sẵn sàng cung cấp dữ liệu đầu ra hợp lệ.

```
initial begin : STIM
    wait (RST_I == 1);
    wait (RST_I == 0);
    repeat(10) @(posedge CLK_I);

    fork
        begin
            LoadInputData("input256.txt");
            GenerateInputWave;
        end
        begin
            wait (ACK_O == 1);
            wait (ACK_O == 0);
            CYC_I = 1;
            STB_I = 1;
            wait (RTY_O == 1);
            wait (RTY_O == 0);
            repeat(256) @(posedge CLK_I);
            SaveOutputData("output256.txt");
            @(negedge CLK_I);
        end
    join
    CYC_I = 0;
    STB_I = 0;
    repeat(10) @(posedge CLK_I);
    $finish;
end
```

Khối initial STIM trong testbench này về cơ bản vẫn tương tự như testbench ở mục trước, bắt đầu bằng thao tác reset để đưa hệ thống về trạng thái khởi đầu.

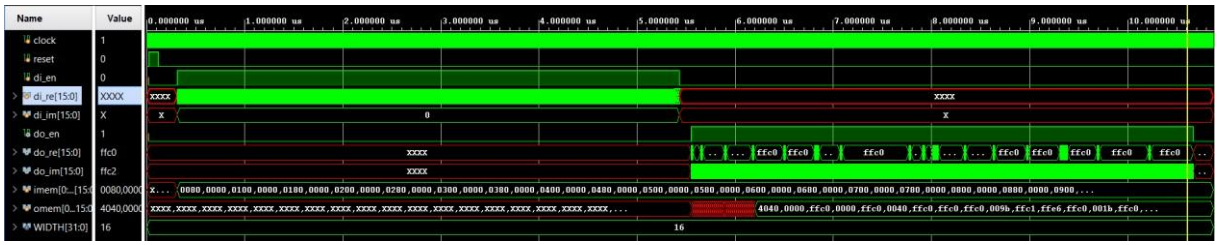
Sau đó, tác vụ GenerateInputWave được gọi để đưa dữ liệu từ imem vào DUT. Tuy nhiên, điểm khác biệt là thay vì sử dụng tín hiệu di_en để xác định thời điểm ghi nhận đầu ra như trước, testbench này sử dụng hai tín hiệu theo chuẩn Wishbone là ACK_O và RTY_O. Trong đó, ACK_O được dùng để xác định thời điểm kết thúc quá trình ghi dữ liệu đầu vào, còn RTY_O đóng vai trò thông báo khi DUT bắt đầu xuất ra kết quả, từ đó kích hoạt tác vụ SaveOutputData.

4.2. Kết quả mô phỏng dạng sóng

Để thực hiện mô phỏng chức năng cho thiết kế DFT 256 điểm, nhóm đã lưu trữ dữ liệu đầu vào trong file "input256.txt". Nhằm đơn giản hóa quá trình kiểm thử và so sánh kết quả, dãy dữ liệu đầu vào được lựa chọn là các giá trị liên tiếp từ 1 đến 256.

Tuy nhiên, vì thiết kế sử dụng định dạng fixed-point Q1.15 để biểu diễn và xử lý dữ liệu, nên các giá trị này cần được chuẩn hóa về khoảng [0, 1] bằng cách chia cho 256. Sau khi chuẩn hóa, các giá trị sẽ được chuyển đổi sang định dạng Q1.15 và biểu diễn ở dạng hexa, phù hợp với yêu cầu đầu vào của thiết kế phần cứng.

4.2.1. Kết quả mô phỏng dạng sóng của thiết kế FFT256 điểm



Quan sát waveform mô phỏng, ta thấy quá trình bắt đầu bằng thao tác reset, nhằm đưa toàn bộ hệ thống về trạng thái khởi tạo. Sau một vài chu kỳ, các giá trị đầu vào được lưu trong bộ nhớ imem lần lượt được gán vào các ngõ vào di_re và

di_im. Đồng thời, tín hiệu di_en được kích lên mức cao để bắt đầu quá trình nhận dữ liệu đầu vào.

Module nhận mỗi điểm đầu vào trong một chu kỳ, do đó sau 256 chu kỳ (từ chu kỳ 16 đến 271) toàn bộ 256 điểm đầu vào đã được nạp hoàn tất. Ngay sau đó, tín hiệu di_en được đưa xuống mức thấp để kết thúc quá trình nhận dữ liệu.

Sau một khoảng trễ xử lý ngắn, cụ thể từ chu kỳ 272 đến 277, hệ thống bắt đầu xuất kết quả đầu ra từ chu kỳ 278 thông qua các ngõ ra do_re và do_im. Tín hiệu do_en cũng được đưa lên mức cao để báo hiệu dữ liệu đầu ra hợp lệ. Tương tự quá trình nhập liệu, mỗi chu kỳ hệ thống xuất ra một điểm kết quả, nên cần 256 chu kỳ (từ là đến chu kỳ 533) để xuất toàn bộ dữ liệu đầu ra.

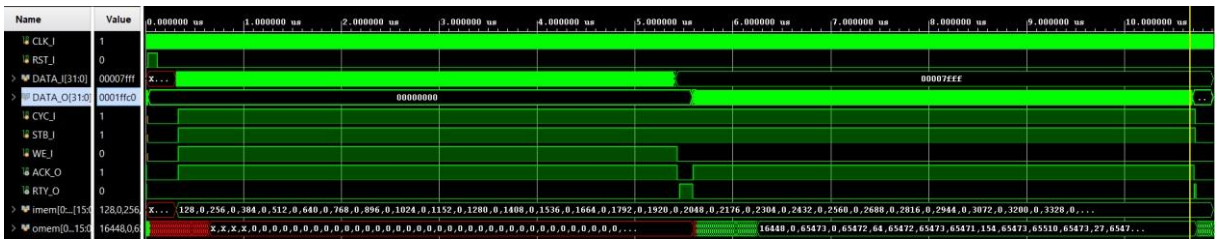
Từ đó có thể xác định độ trễ xử lý của hệ thống như sau:

- Độ trễ tuyệt đối tính từ lúc bắt đầu nhập đến lúc có điểm đầu ra đầu tiên là 278 - 16 = 262 chu kỳ.
- Độ trễ xử lý sau khi nhập xong toàn bộ dữ liệu (tức tính từ chu kỳ 271 đến chu kỳ 278) là khoảng 6 chu kỳ.
- Độ trễ đầu-cuối (end-to-end latency) là 533 - 16 = 518 chu kỳ cho toàn bộ 256 điểm.

Ngoài ra, hệ thống có thể xử lý liên tục nhiều khối dữ liệu mà không cần reset. Cụ thể, khi ngay sau chu kỳ 271, ta tiếp tục nhập thêm 256 điểm mới (từ chu kỳ 272 đến 527), thì kết quả đầu ra của dãy mới bắt đầu xuất hiện ngay tại chu kỳ 534.

Như mô tả thiết kế testbench ở mục 4.1.1, các điểm kết quả đầu ra này sẽ được testbench sắp xếp lại thứ tự và lưu lại trong file “output256.txt”.

4.2.2. Kết quả mô phỏng dạng sóng của module đã bọc theo chuẩn wishbone



Tương tự như waveform của thiết kế gốc, quá trình mô phỏng bắt đầu bằng thao tác reset để đưa hệ thống về trạng thái khởi tạo ban đầu. Sau một vài chu kỳ, quá trình nhận dữ liệu đầu vào.

Tuy nhiên, quá trình này, thay vì được gán giá trị trực tiếp thông qua các ngõ vào như trong testbench của thiết kế gốc, nay được thực hiện thông qua các thao tác ghi dữ liệu tuân theo chuẩn giao tiếp Wishbone. Trong suốt quá trình truyền dữ liệu đầu vào, các tín hiệu CYC_I, STB_I, WE_I được duy trì ở mức cao để thực hiện thao tác ghi dữ liệu. Việc ghi được thực hiện tại mỗi cạnh lên của xung clock, đồng thời tín hiệu ACK_O được đưa lên mức cao để xác nhận rằng thao tác ghi dữ liệu vào module đã được hoàn tất thành công.

Sau khi hoàn tất quá trình ghi, testbench hạ tín hiệu WE_I xuống mức thấp để chuyển sang chế độ đọc dữ liệu đầu ra. Do thiết kế cần một khoảng thời gian xử lý nhất định để hoàn thành phép biến đổi DFT, kết quả đầu ra chưa thể sẵn sàng ngay lập tức. Trong khoảng trễ này, nếu testbench cố gắng thực hiện thao tác đọc, thiết kế sẽ phản hồi bằng cách kích hoạt tín hiệu RTY_O lên mức cao để báo hiệu rằng dữ liệu chưa sẵn sàng và thao tác đọc hiện tại cần được hoãn lại.

Chỉ khi dữ liệu đầu ra đã được tính toán xong, tín hiệu RTY_O mới được đưa xuống mức thấp, cho phép testbench bắt đầu quá trình đọc dữ liệu đầu ra. Tương tự như ở thiết kế gốc, các kết quả được trả về theo từng chu kỳ, mỗi chu kỳ một điểm dữ liệu, cho đến khi hoàn tất toàn bộ 256 điểm kết quả đầu ra.

4.3. Đánh giá độ chính xác của thiết kế

Để kiểm tra độ chính xác của thiết kế FFT 256 điểm, nhóm đã tiến hành so sánh kết quả đầu ra thu được từ mô phỏng (output256.txt) với kết quả chuẩn bằng python. Các chỉ số nhóm sử dụng để đánh giá bao gồm:

- Mean Absolute Error (MAE) để đánh giá sai số trung bình giữa 2 dãy:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{ireal} - y_{isim}|$$

- Mean Squared Error (MSE):

$$N$$

$$MSE = \frac{1}{N}\sum_{i=1}(y_{ireal} - y_{isim})2$$

- Root Mean Squared Error (RMSE): $RMSE = \sqrt{\overline{MSE}}$
- Signal-to-Noise Ratio (SNR) để đánh giá độ "sạch" của kết quả mô phỏng:

$$SNR^{(dB)} = 10.log_{10}\left(\frac{\sum|y_{real}|2}{\sum|y^{real} - y^{sim}|^2}\right)$$

Kết quả đánh giá nhóm tính được các chỉ số sai số được tính như sau:

- Mean Absolute Error (MAE): 0.0045
- Mean Squared Error (MSE): 0.000031
- Root Mean Squared Error (RMSE): 0.0056
- Signal-to-Noise Ratio (SNR): 64.42 dB

Kết quả cho thấy sai số tuyệt đối trung bình (MAE) rất nhỏ, chỉ khoảng 0.0045, và sai số bình phương trung bình (MSE) ở mức 0.000031 – cho thấy sự khác biệt giữa kết quả mô phỏng và kết quả chuẩn là rất nhỏ. Đặc biệt, giá trị SNR đạt 64.42 dB, đây là một chỉ số rất tốt trong các hệ thống xử lý tín hiệu số sử dụng số cố định (fixed-point), phản ánh độ chính xác cao và độ nhiễu thấp của thiết kế.

Với các chỉ số trên, có thể kết luận rằng hệ thống hoạt động chính xác và tin cậy, đáp ứng tốt yêu cầu của một bộ tăng tốc FFT 256 điểm trong các ứng dụng xử lý tín hiệu cơ bản.

CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC

5.1. Kết quả tổng hợp trên quartus

5.1.1. Kết quả tổng hợp thiết kế

Kiểm tra tài nguyên

Resouce	Usage
Total logic elements	1,740
Total combinational functions	1,701
Total registers	515
Total pins	68
Embedded Multiplier 9-bit elements	24
Maximum fan-out	771
Total fan-out	8763
Average fan-out	3.42

Kiểm tra định thời

Fmax	Restricted Fmax
65.1 MHz	65.1 MHz

Kiểm tra công suất

Total Thermal Power Dissipation	117.90 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.95 mW
I/O Thermal Power Dissipation	37.95 mW

5.1.2. Kết quả tổng hợp khi đã tích hợp chuẩn wishbone bus

Kiểm tra tài nguyên

Resouce	Usage

Total logic elements	1,745
Total combinational functions	1674
Total registers	582
Total pins	68
Embedded Multiplier 9-bit elements	71
Maximum fan-out	838
Total fan-out	8984
Average fan-out	3.45

Kiểm tra định thời

Fmax	Restricted Fmax
67.82 MHz	67.82 MHz

Kiểm tra công suất

Total Thermal Power Dissipation	118.31 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.95 mW
I/O Thermal Power Dissipation	38.36 mW

5.2. Nạp kit DE2

Nhóm đã tiến hành nạp kit thực tế trên kit DE2 để kiểm tra tính chính xác của thiết kế. Trong video nạp kit đã bao gồm các mô tả liên quan.

Link video nạp kit: Link video

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

Nhóm đã thực hiện thành công mục tiêu của đề tài, triển khai thành công thuật toán DFT với phương pháp Radix-2^2, thực hiện tối ưu hóa thiết kế bằng pipeline. Nhóm cũng đã thành công thích hợp thiết kế của thuật toán với chuẩn giao tiếp

wishbone, đảm bảo khả năng kết nối linh hoạt với các IP khác trong hệ thống. Tính đúng đắn của thiết kế thuật toán cũng như thiết kế sau khi bọc theo chuẩn giao tiếp wishbone đã được kiểm tra kĩ lưỡng bằng testbench và mô phỏng dưới dạng waveform.

Nhóm đã tổng hợp và báo cáo tài nguyên đầy đủ đối với thiết kế thuật toán cũng như thiết kế sau khi bọc theo chuẩn giao tiếp wishbone. Nhóm đã thành công triển khai thiết kế lên trên kit FPGA DE2 thực tế, kết quả nạp kit cho thấy chức năng hoàn toàn chính xác.

6.2. Hướng phát triển

Trong tương lai, đề tài có thể được phát triển thêm một số hướng như:

- Phát triển bộ tăng tốc cho DFT với số điểm lớn hơn (1024 điểm, 4096 điểm) để đáp ứng các ứng dụng xử lý tín hiệu phức tạp hơn.
- Tiếp tục cải thiện thiết kế bằng cách áp dụng các kỹ thuật tiên tiến như pipeline đa mức hoặc tối ưu hóa tài nguyên để giảm thêm mức tiêu thụ năng lượng và tăng hiệu suất.
- Chạy mô phỏng và kiểm tra chức năng của thiết kế bằng Autotest/ UVM+
- Tổng hợp và thiết kế vật lý dạng khối – block design.
- Kiểm tra thiết kế vật lý và rút trích thông số.