**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM**

**KHOA ĐIỆN – ĐIỆN TỬ**

**BỘ MÔN ĐIỆN TỬ**

---🙠🕮🙢---



**BÁO CÁO BÀI TẬP LỚN 2**

***Đề tài:***

**THIẾT KẾ VÀ HIỆN THỰC HOÁ BỘ XỬ LÝ FFT 8 ĐIỂM CẤU TRÚC DIT SỬ DỤNG CHUẨN SỐ THỰC DẤU CHẤM ĐỘNG 32-BIT (IEEE 754)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Giảng viên hướng dẫn:** | | | | Ths. Nguyễn Trung Hiếu | | |
| **Lớp - Nhóm:** | | | | L01 - 3 | | |
| **Mã nguồn dự án:** | | | | [Click](https://drive.google.com/drive/folders/1J_n6Ps7J6EZEhzRF5FXX1G4HLufGIYCZ) | | |
|  | | | |  | | |
| **Nhóm sinh viên thực hiện:** | | | | | | |
| **STT** | **Họ** | **Tên** | **MSSV** | | **Nhiệm vụ** | **Đánh giá** |
| 1 | Lý Quốc Hoàng | Minh | 2312077 | | Nghiên cứu lý thuyết  Thiết kế kiến trúc | 100% |
| 2 | Nguyễn Thanh | Phong | 2312626 | | Hiện thực hóa RTL  Kiểm tra và Mô phỏng  Tổng hợp  Hiện thực trên FPGA  Soạn thảo báo cáo | 100% |

*Thành phố Hồ Chí Minh, ngày 5 tháng 1 năm 2025*

**MỤC LỤC**

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 4](#_Toc218548052)

[1.1. Thuật toán Fast Fourier Transform (FFT) 4](#_Toc218548053)

[1.1.1. Biến đổi Fourier Rời rạc (DFT) 4](#_Toc218548054)

[1.1.2. Thuật toán Cooley-Tukey DIT (Decimation-In-Time) 4](#_Toc218548055)

[1.1.3. Cấu trúc Bướm (Butterfly) và Phân tầng cho FFT 8 điểm 4](#_Toc218548056)

[1.1.4. Công thức tính toán Bướm (Butterfly Unit) 5](#_Toc218548057)

[1.2. Chuẩn số thực dấu chấm động IEEE 754 (Single Precision) 6](#_Toc218548058)

[1.2.1. Cấu trúc dữ liệu 32-bit 6](#_Toc218548059)

[1.2.2. Lý do lựa chọn Floating Point thay vì Fixed Point 6](#_Toc218548060)

[1.3. Áp dụng vào tính toán một vector 6](#_Toc218548061)

[CHƯƠNG 2: THIẾT KẾ KIẾN TRÚC HỆ THỐNG 9](#_Toc218548062)

[2.1. Khối module top 9](#_Toc218548063)

[2.2. Khối điều khiển trung tâm fft\_control 9](#_Toc218548064)

[2.3. Khối idx\_gen\_8pt: 11](#_Toc218548065)

[2.4. Khối buf\_bank: 12](#_Toc218548066)

[2.5. Khối tính toán bướm (bf\_compute): 13](#_Toc218548067)

[2.6. Bộ nhân phức (complex\_mult\_pipe): 15](#_Toc218548068)

[2.7. Bộ delay: 17](#_Toc218548069)

[2.8. Khối chốt dữ liệu đầu vào (bf\_input\_latch): 18](#_Toc218548070)

[2.9. Khối bf\_top\_single: 20](#_Toc218548071)

[2.10. Khối hệ số quay (rom\_twiddle): 20](#_Toc218548072)

[CHƯƠNG 3: KIỂM CHỨNG VÀ MÔ PHỎNG 21](#_Toc218548073)

[3.1. Môi trường kiểm tra 21](#_Toc218548074)

[3.1.1. Bộ tạo mẫu ngẫu nhiên và Driver (Stimulus Generator) 21](#_Toc218548075)

[3.1.2. Mô hình tham chiếu (Golden Model) 21](#_Toc218548076)

[3.1.3. Bộ giám sát và Kiểm tra sai số (Monitor & Checker) 22](#_Toc218548077)

[3.1.4. Kiểm tra thuộc tính (SystemVerilog Assertions - SVA) 23](#_Toc218548078)

[3.2. Kiểm định bằng Assertion (Formal Verification) 23](#_Toc218548079)

[3.3. Kết quả mô phỏng 25](#_Toc218548080)

[3.3.1. Kết quả log pass 25](#_Toc218548081)

[3.3.2. Kết quả dạng sóng 29](#_Toc218548082)

[CHƯƠNG 4: TỔNG HỢP LOGIC 36](#_Toc218548083)

[4.1. Môi trường và Công cụ tổng hợp 36](#_Toc218548084)

[4.2. Báo cáo Diện tích 36](#_Toc218548085)

[4.3. Báo cáo Thời gian 37](#_Toc218548086)

[4.4. Báo cáo công suất 38](#_Toc218548087)

[CHƯƠNG 5: HIỆN THỰC HÓA TRÊN FPGA 40](#_Toc218548088)

[5.2. Môi trường và công cụ thực hiện 40](#_Toc218548089)

[5.3. Thiết kế hệ thống Top-level trên Kit DE2 40](#_Toc218548090)

[5.3.1. Sơ đồ khối hệ thống thực nghiệm 40](#_Toc218548091)

[5.3.2. Cơ chế hoạt động của Wrapper 41](#_Toc218548092)

[5.4. Phân bố chân và giao diện người dùng 41](#_Toc218548093)

[5.5. Kiểm tra thực nghiệm 42](#_Toc218548094)

[CHƯƠNG 6: KẾT LUẬN 43](#_Toc218548095)

[6.1. Kết quả đạt được 43](#_Toc218548096)

[6.2. Các mặt hạn chế 43](#_Toc218548097)

[6.3. Hướng phát triển 44](#_Toc218548098)

[PHỤ LỤC 45](#_Toc218548099)

[Testbench 45](#_Toc218548100)

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

## 1.1. Thuật toán Fast Fourier Transform (FFT)

### 1.1.1. Biến đổi Fourier Rời rạc (DFT)

Biến đổi Fourier rời rạc (Discrete Fourier Transform - DFT) là công cụ toán học nền tảng trong xử lý tín hiệu số, dùng để chuyển đổi một tín hiệu từ miền thời gian sang miền tần số. Đối với một chuỗi tín hiệu đầu vào rời rạc x[n] có độ dài N, công thức DFT được định nghĩa như sau:

Trong đó:

* x[n]: Mẫu tín hiệu đầu vào thứ  (miền thời gian).
* X[k]: Mẫu tín hiệu đầu ra thứ  (miền tần số).
* : Hệ số quay (Twiddle Factor).

Và khi Triển khai Euler ta có:

Phần thực =

Phần ảo =

Công thức biến đổi được sử dụng trong tính toán lý thuyết để đơn giản hoá trong quá trình so sánh với kết quả thực nghiệm.

### 1.1.2. Thuật toán Cooley-Tukey DIT (Decimation-In-Time)

Để khắc phục nhược điểm của DFT, thuật toán FFT (Fast Fourier Transform) ra đời, trong đó phổ biến nhất là thuật toán *Cooley-Tukey phân chia theo thời gian (Decimation-In-Time - DIT).* Nguyên lý cốt lõi của DIT là phương pháp "chia để trị" (divide and conquer). Thuật toán chia một bài toán DFT  điểm thành hai bài toán DFT  điểm:

* Một nhóm chứa các mẫu ở vị trí chẵn ().
* Một nhóm chứa các mẫu ở vị trí lẻ ().

Quá trình này được lặp lại đệ quy cho đến khi chỉ còn các phép tính DFT 2 điểm cơ bản (gọi là cấu trúc Bướm - Butterfly). Nhờ đó, độ phức tạp tính toán giảm xuống còn , giúp tăng tốc độ xử lý đáng kể.

### 1.1.3. Cấu trúc Bướm (Butterfly) và Phân tầng cho FFT 8 điểm

Trong bài tập lớn này, nhóm em thiết kế bộ xử lý FFT cho  điểm. Theo thuật toán Radix-2, quá trình tính toán được chia thành  tầng (Stage):

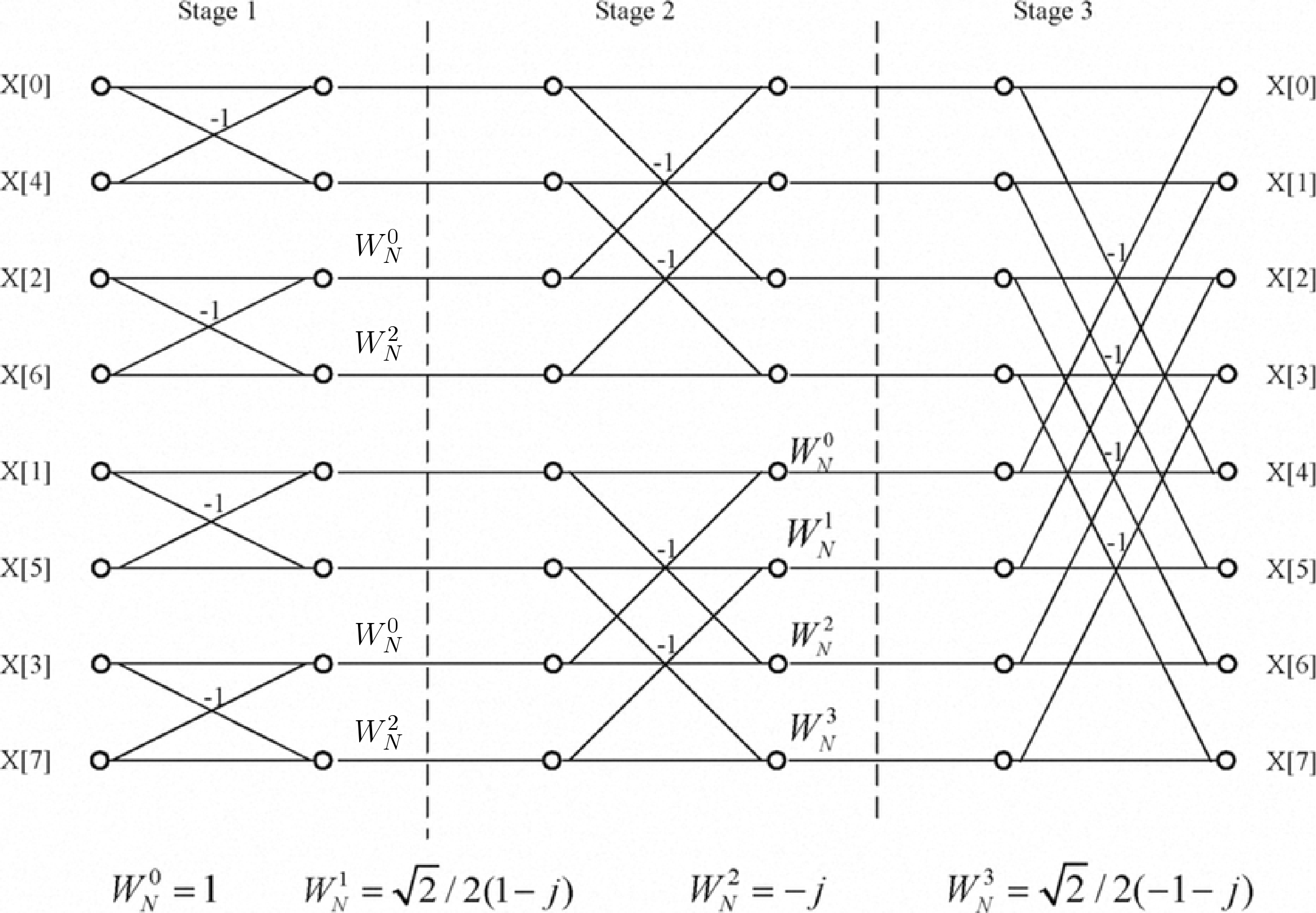
* **Đầu vào:** Dữ liệu được sắp xếp theo trật tự đảo bit (Bit-reversal order). Với , thứ tự chỉ số đầu vào là:



* **Đầu ra:** Dữ liệu tần số X[k] thu được theo thứ tự tự nhiên (0, 1, 2, 3, 4, 5, 6, 7).

Mô tả các tầng như sau:

1. **Ở Stage 0 (tầng 1):** Kết hợp các cặp mẫu đầu vào liền kề nhau. Hệ số quay sử dụng luôn là . Khoảng cách (span) giữa hai nhánh bướm là 1.
2. **Ở Stage 1 (Tầng 2):** Kết hợp kết quả từ Stage 0. Sử dụng các hệ số quay  và . Khoảng cách giữa hai nhánh bướm là 2.
3. **Ở Stage 2 (tầng 3):** Tầng cuối cùng tạo ra kết quả DFT. Sử dụng toàn bộ các hệ số quay . Khoảng cách giữa hai nhánh bướm là 4.



Hình 1. Sơ đồ bướm FFT 8 điểm DIT

### 1.1.4. Công thức tính toán Bướm (Butterfly Unit)

Phần tử cơ bản nhất của kiến trúc FFT là đơn vị tính toán Bướm (Butterfly Unit). Tại mỗi nhịp, nó nhận vào hai số phức đầu vào *A, B* và một hệ số quay , sau đó thực hiện phép tính:

* Đầu ra nhánh trên: 
* Đầu ra nhánh dưới: 
* Do hệ thống xử lý số phức (), các phép toán thực tế được triển khai phần cứng bao gồm:
* Nhân phức: 
* Cộng/Trừ phức: Cộng/trừ tương ứng phần thực với phần thực, phần ảo với phần ảo.

## 1.2. Chuẩn số thực dấu chấm động IEEE 754 (Single Precision)

### 1.2.1. Cấu trúc dữ liệu 32-bit

Để đảm bảo độ chính xác cao trong tính toán khoa học và kỹ thuật, bài tập lớn sử dụng định dạng số thực dấu chấm động (Floating Point) độ chính xác đơn theo chuẩn IEEE 754. Một từ mã 32-bit được chia thành 3 trường:

* *Bit dấu (Sign - S***):** 1 bit (Bit 31). *S=0* là số dương, *S=1* là số âm.
* *Phần mũ (Exponent - E):* 8 bit (Bit 30-23). Biểu diễn số mũ với giá trị Bias là 127.
* *Phần định trị (Mantissa/Fraction - M):* 23 bit (Bit 22-0). Biểu diễn phần lẻ của số học.

Giá trị thực của số được tính theo công thức:



### 1.2.2. Lý do lựa chọn Floating Point thay vì Fixed Point

Mặc dù việc thiết kế phần cứng cho số dấu phẩy tĩnh (Fixed Point) đơn giản hơn và tốn ít tài nguyên hơn, bài tập lớn lần này lựa chọn số dấu chấm động (Floating Point) vì các ưu điểm vượt trội sau:

* *Dải động rộng:* Số chấm động 32-bit có thể biểu diễn các giá trị cực nhỏ (cỡ ) đến cực lớn (cỡ ). Điều này rất quan trọng trong xử lý tín hiệu FFT, nơi biên độ tín hiệu đầu ra có thể tăng vọt so với đầu vào, tránh hiện tượng tràn số (Overflow) thường gặp ở Fixed Point.
* *Độ chính xác cao:* Cơ chế tự động chuẩn hóa (Normalization) của số chấm động giúp duy trì độ chính xác tối đa cho cả tín hiệu biên độ nhỏ và lớn, giảm thiểu sai số lượng tử hóa (Quantization Error) qua nhiều tầng tính toán của FFT.
* *Tính chuẩn hóa:* Việc sử dụng chuẩn IEEE 754 giúp thiết kế dễ dàng tương thích và giao tiếp với các hệ thống vi xử lý, phần mềm mô phỏng (như MATLAB/Python) mà không cần các bước chuyển đổi định dạng phức tạp.

## 1.3. Áp dụng vào tính toán một vector

Để minh họa cụ thể quá trình xử lý của bộ FFT 8 điểm, nhóm thực hiện kiểm tra chi tiết *Vector số 3* từ log mô phỏng ở *chương 3*. Nhóm sẽ so sánh kết quả tính toán lý thuyết với kết quả thực tế thu được từ phần cứng sau khi mô phỏng.

Dữ liệu đầu vào lấy từ *Vector 3* (Time domain):

|  |  |  |
| --- | --- | --- |
| **Chỉ số n** | **Phần Thực (Real)** | **Phần Ảo (Imag)** |
| x[0] | 2.0093 | -3.2505 |
| x[1] | 8.3096 | -0.7749 |
| x[2] | 7.8467 | +7.7729 |
| x[3] | 3.6343 | +0.9048 |
| x[4] | -1.6396 | +6.5923 |
| x[5] | 1.3164 | -2.4985 |
| x[6] | 3.2041 | +4.9072 |
| x[7] | 1.4980 | +2.9404 |

Để kiểm tra nhanh tính đúng đắn mà không cần thực hiện toàn bộ các phép nhân số phức phức tạp, ta xét hai điểm tần số đặc biệt:

* Tại  (Thành phần một chiều - DC): .



* + *Tính tay:* *:* 2.0093 + 8.3096 + 7.8467 + 3.6343 + (-1.6396) + 1.3164 + 3.2041 + 1.4980 = 26.1718
  + *Mô phỏng:* 26.1787 (Khớp).
* Tại  (Tần số Nyquist): .



* + *Tính tay:* 2.0093 - 8.3096 + 7.8467 - 3.6343 + (-1.6396) - 1.3164 + 3.2041 - 1.4980 = -3.3379
  + *Mô phỏng:* -3.3379 (Khớp tuyệt đối).

Tương tự như thế với các hệ số k còn lại, ta có thể tính được đầy đủ 8 thành phần. Dưới đây là bảng đối chiếu toàn bộ 8 thành phần tần số đầu ra X[k]:

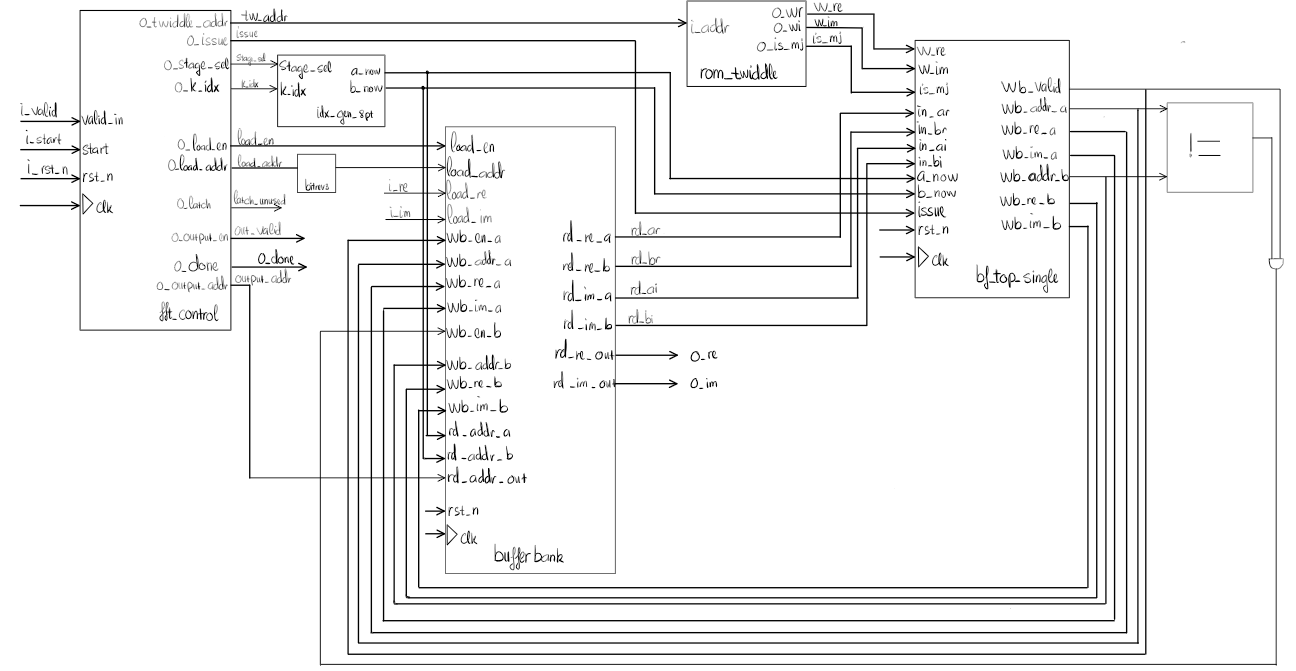
|  |  |  |  |
| --- | --- | --- | --- |
| **Chỉ số k** | **Ý nghĩa (Frequency)** | **Tính toán Lý thuyết (Re,Im)** | **Kết quả Mô phỏng (Re,Im)** |
| 0 | DC (0 Hz) | (26.1787, 16.5938) | (26.1787, 16.5938) |
| 1 | Cơ bản () | (9.7284, -18.2826) | (9.7284, -18.2826) |
| 2 |  | (-17.7998, -13.8320) | (-17.7998, -13.8320) |
| 3 |  | (-2.8718, -14.3139) | (-2.8718, -14.3139) |
| 4 | Nyquist () | (-3.3379, 15.4502) | (-3.3379, 15.4502) |
| 5 |  | (3.3009, -10.6881) | (3.3009, -10.6881) |
| 6 |  | (-3.5625, -4.8447) | (-3.5625, -4.8447) |
| 7 | Cơ bản âm | (4.4382, 3.9135) | (4.4382, 3.9135) |

Nhóm em cũng đã xây dựng 1 file Excel chuyên dùng để tính toán và chuyển đổi FFT cho số (vector 8 phần tử) nhập từ người dùng. Nhóm em xin đính kèm phía bên dưới.

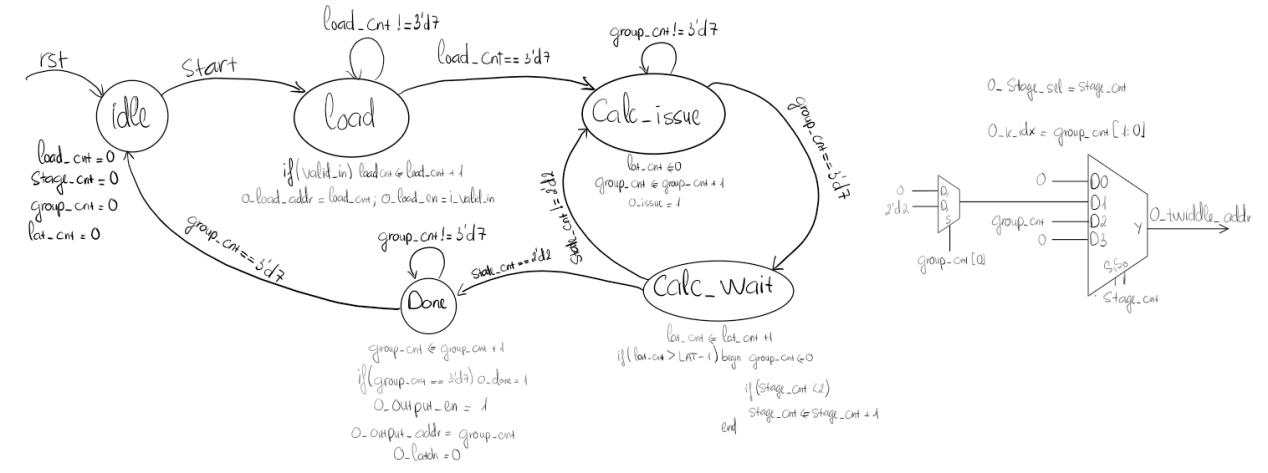


# CHƯƠNG 2: THIẾT KẾ KIẾN TRÚC HỆ THỐNG

## 2.1. Khối module top



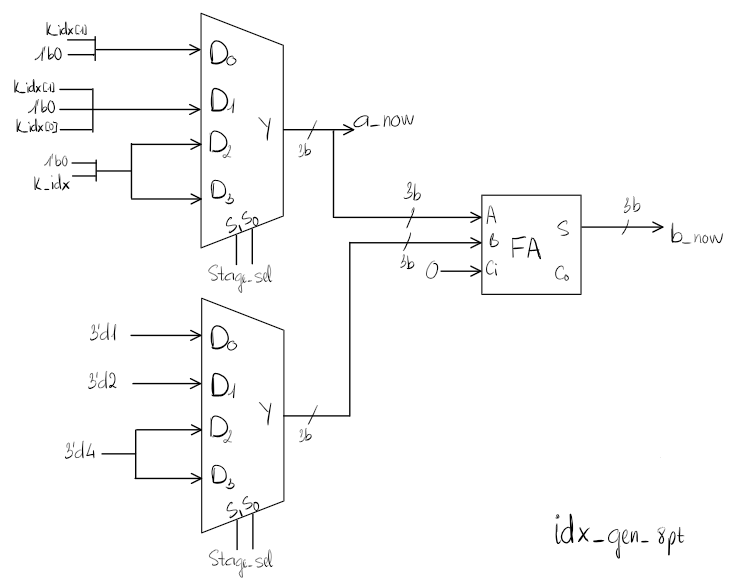
## 2.2. Khối điều khiển trung tâm fft\_control



- Khối này đóng vai trò là "bộ não" điều phối toàn bộ hoạt động của hệ thống. Nó hoạt động dựa trên một máy trạng thái hữu hạn (FSM) với 5 trạng thái chính:

* Trạng thái IDLE (Chờ):
* Đây là trạng thái mặc định khi hệ thống mới khởi động hoặc sau khi Reset (rst).
* Tại đây, toàn bộ các biến đếm nội bộ như load\_cnt (đếm mẫu nạp), stage\_cnt (đếm tầng FFT), group\_cnt (đếm nhóm bướm), và lat\_cnt (đếm độ trễ) đều được xóa về 0 để sẵn sàng cho chu kỳ mới.
* Hệ thống nằm chờ ở đây cho đến khi tín hiệu Start được kích hoạt.
* Trạng thái LOAD (Nạp mẫu):
* Khi có tín hiệu Start, FSM chuyển sang trạng thái **LOAD**.
* Nhiệm vụ: Điều khiển việc ghi 8 mẫu dữ liệu đầu vào vào bộ nhớ đệm (buffer\_bank).
* Hoạt động: FSM giám sát tín hiệu i\_valid\_in. Mỗi khi có một mẫu hợp lệ, biến load\_cnt tăng lên 1.
* Chuyển trạng thái: Quá trình lặp lại cho đến khi load\_cnt đạt giá trị 7 (tức là đã nạp đủ 8 mẫu), FSM tự động chuyển sang trạng thái tính toán (Calc\_issue).
* Trạng thái CALC\_ISSUE (Phát lệnh tính toán):
* Đây là pha đầu tiên của quy trình tính toán.
* Nhiệm vụ: Cấp phát tín hiệu o\_issue để đọc cặp dữ liệu (A, B) từ bộ nhớ và đưa vào đường ống xử lý (Pipeline).
* Tại trạng thái này, FSM cũng tính toán địa chỉ o\_twidlle\_addr để lấy hệ số quay Wn tương ứng từ ROM, dựa trên stage\_cnt và group\_cnt.
* Sau khi phát lệnh, biến đếm độ trễ lat\_cnt được reset về 0 và FSM chuyển ngay sang trạng thái chờ (Calc\_Wait).
* Trạng thái CALC\_WAIT (Chờ kết quả Pipeline):
* Đây là pha thứ hai, thiết kế đặc biệt để xử lý độ trễ của phép tính số thực dấu phẩy động.
* Nhiệm vụ: FSM tạm dừng việc cấp dữ liệu mới và chờ cho đến khi bộ tính toán (bf\_compute) hoàn thành phép tính hiện tại.
* Hoạt động: Biến lat\_cnt tăng dần theo mỗi chu kỳ clock. Khi lat\_cnt đạt đến ngưỡng quy định (LAT-1), nghĩa là dữ liệu đã đi hết đường ống và được ghi lại vào bộ nhớ, FSM sẽ kiểm tra điều kiện lặp:
* **Lặp nhóm (Inner Loop):** Nếu group\_cnt chưa đạt 7, FSM tăng group\_cnt và quay lại Calc\_issue để tính cặp mẫu tiếp theo trong cùng tầng.
* **Lặp tầng (Outer Loop):** Nếu đã hết nhóm (group\_cnt = 7) nhưng chưa hết tầng (stage\_cnt < 2), FSM tăng stage\_cnt, reset group\_cnt về 0 và quay lại Calc\_issue để bắt đầu tầng mới.
* **Kết thúc:** Nếu đã xong cả nhóm cuối cùng của tầng cuối cùng (Stage 2), FSM chuyển sang trạng thái Done.
* Trạng thái DONE (Hoàn tất):
* Thông báo quá trình tính toán FFT đã hoàn thành.
* FSM kích hoạt tín hiệu o\_done, đồng thời kích hoạt o\_output\_en và điều khiển địa chỉ o\_output\_addr để cho phép hệ thống bên ngoài đọc kết quả đã sắp xếp đúng thứ tự tần số.

## 2.3. Khối idx\_gen\_8pt:



- Khối này chịu trách nhiệm tính toán cặp địa chỉ đọc/ghi cho bộ nhớ trong quá trình thực thi thuật toán Butterfly.

* **Nguyên lý hoạt động:** Dựa trên cấu trúc hình học của thuật toán FFT DIT (Decimation-In-Time), khoảng cách (stride) giữa hai điểm tham gia vào một con bướm thay đổi theo từng tầng (Stage 0: cách 1, Stage 1: cách 2, Stage 2: cách 4).
* Hoạt động chi tiết:
* Khối sử dụng các bộ dồn kênh (**Multiplexers - MUX**) được điều khiển bởi tín hiệu Stage\_sel (tương ứng với stage\_cnt từ FSM).
* Các bit của địa chỉ cơ sở k\_idx (từ group\_cnt) được hoán vị hoặc kết hợp với các hằng số (0, 1) thông qua MUX để tạo ra địa chỉ a\_now.
* Địa chỉ b\_now sau đó được tính toán dựa trên a\_now cộng thêm khoảng cách tương ứng của tầng đó (sử dụng bộ cộng FA - Full Adder).
* Kết quả là khối này luôn cung cấp đúng cặp chỉ số cho 2 mẫu cần tính toán tại bất kỳ thời điểm nào của thuật toán.

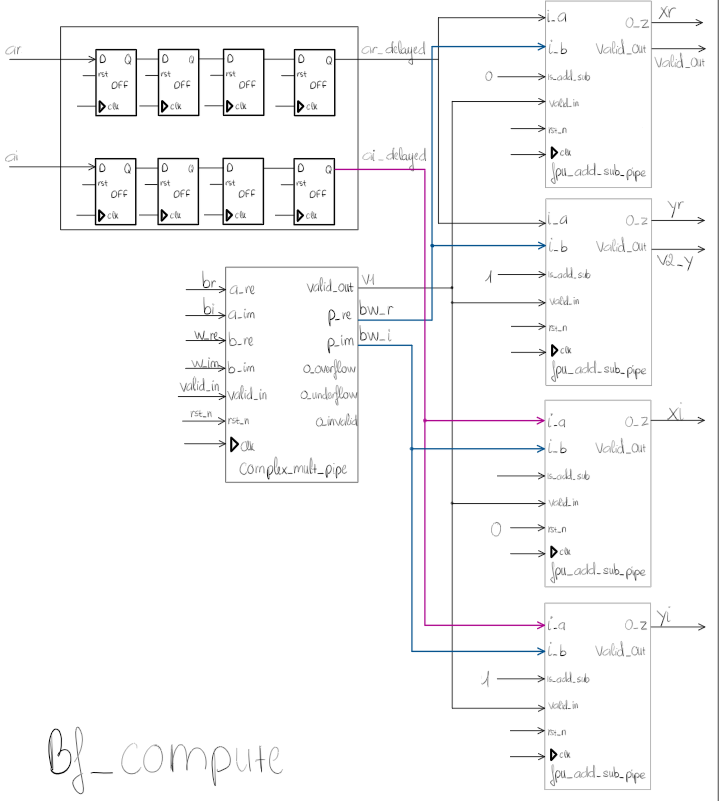
## 2.4. Khối buf\_bank:



- Thay vì dùng RAM khối thông thường, thiết kế này sử dụng kiến trúc thanh ghi (Register-based) để tối ưu hóa tốc độ truy xuất song song.

* Cấu trúc lưu trữ: Gồm 2 mảng thanh ghi riêng biệt: 8 thanh ghi 32-bit cho phần Thực (Real) và 8 thanh ghi 32-bit cho phần Ảo (Imag).
* Cơ chế Đọc/Ghi đa cổng (Multi-port access):
* **Cổng Nạp (Load Port):** Khi tín hiệu load\_en tích cực, dữ liệu từ bên ngoài (load\_re, load\_im) được ghi vào địa chỉ load\_addr.
* **Cổng Ghi hồi tiếp (Write-back Port):** Khi bộ tính toán trả về kết quả (wb\_en\_a/b tích cực), dữ liệu mới được ghi đè vào các thanh ghi tại địa chỉ wb\_addr\_a/b.
* **Cổng Đọc (Read Port):** Hệ thống các bộ MUX lớn ở đầu ra cho phép đọc đồng thời 2 giá trị bất kỳ (A và B) dựa trên địa chỉ rd\_addr\_a và rd\_addr\_b do khối idx\_gen cung cấp.
* **Ưu điểm:** Kiến trúc này loại bỏ hoàn toàn xung đột đọc/ghi (read/write collision) thường gặp ở RAM đơn cổng, cho phép FSM đọc dữ liệu cũ và ghi dữ liệu mới một cách linh hoạt.

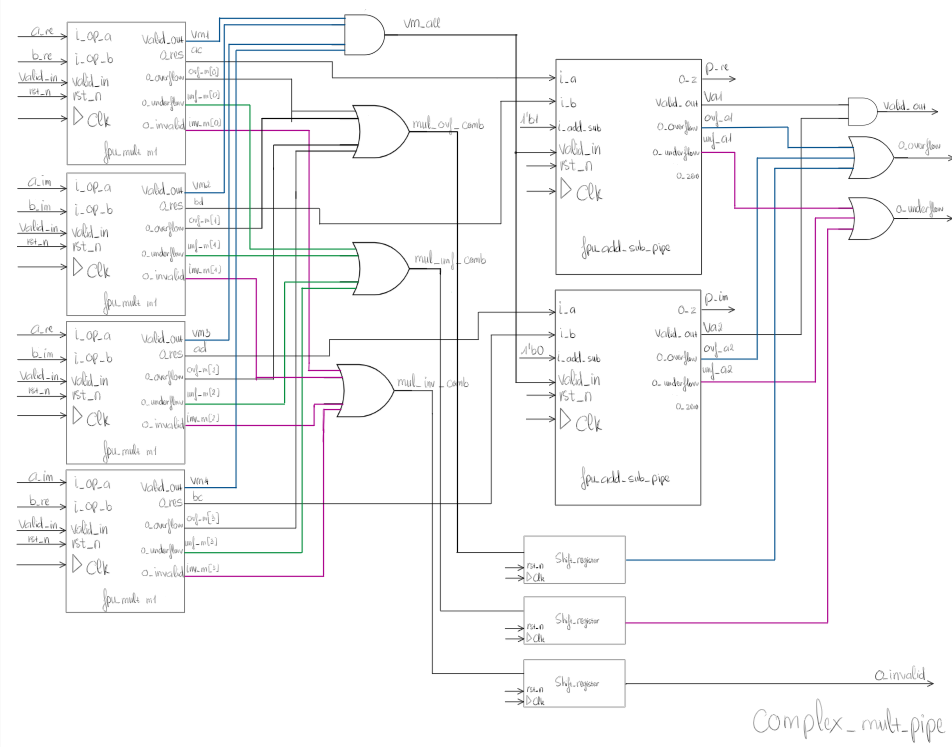
## 2.5. Khối tính toán bướm (bf\_compute):



- Đây là trái tim xử lý số học của hệ thống, thực hiện phép tính: A' = A + W.B và B' = A - W.B.

* ***Pipeline Tầng 1 - Nhân Phức (Complex Multiplier):***
* Module complex\_mult\_pipr nhận dữ liệu mẫu từ B và hệ số quay W
* Nó thực hiện 4 phép nhân thực và 2 phép cộng/trừ thực để tạo ra phần thực và ảo của tích W.B. Các cờ báo tràn (overflow) cũng được xử lý tại đây.
* ***Pipeline Tầng 2 - Căn chỉnh thời gian (Delay Logic):***
* Trong khi B đang được nhân với W (mất nhiều chu kỳ), mẫu A không tham gia phép nhân.
* Để đảm bảo A và (W.B) đến bộ cộng cuối cùng cùng một lúc, mẫu A được đưa qua một chuỗi các Flip-Flop (Shift Register) để tạo độ trễ tương đương.
* ***Pipeline Tầng 3 - Cộng/Trừ Bướm (Butterfly Add/Sub):***
* Kết quả của phép nhân và mẫu A (đã làm trễ) được đưa vào các khối cộng/trừ số thực dấu phẩy động (add\_xr, sub\_yr, v.v.).
* Bốn bộ tính toán hoạt động song song để tạo ra 4 giá trị đầu ra: Re(A'), Im(A'), Re(B'), Im(B').
* **Tín hiệu Valid:** Một tín hiệu valid được truyền song song dọc theo đường ống (pipeline). Chỉ khi tín hiệu này ở mức cao tại đầu ra cuối cùng, kết quả mới được coi là đúng và cho phép ghi vào bộ nhớ (wb\_valid).

## 2.6. Bộ nhân phức (complex\_mult\_pipe):



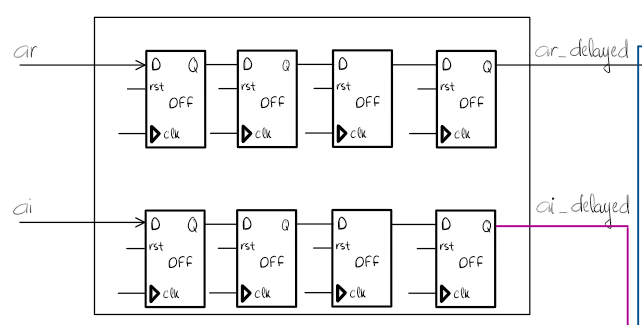
- Đây là khối tiêu tốn tài nguyên nhất, thực hiện phép nhân giữa mẫu dữ liệu đầu vào B (Br + jBi) với hệ số quay W (Wr + jWi).

* ***Tầng 1: 4 Bộ Nhân Thực (FPU Multipliers):***
* Sơ đồ mạch sử dụng 4 bộ nhân số thực dấu chấm động (fpu\_mult\_m1) hoạt động song song để tính các thành phần trung gian:
* vm1 = Br x Wr
* vm2 = Bi x Wi
* vm3 = Br x Wi
* vm4 = Bi x Wr
* Các tín hiệu cờ báo như overflow (tràn), underflow (tràn dưới), invalid (không hợp lệ) từ 4 bộ nhân này được gom lại (OR logic) để cảnh báo lỗi.
* ***Tầng 2: 2 Bộ Cộng/Trừ Thực (FPU Add/Sub):***

Kết quả từ tầng nhân được đưa vào 2 bộ cộng/trừ (fpu\_add\_sub\_pipe):

* **Kênh Thực:** Thực hiện phép trừ Pre = vm1 - vm2. Khối này được cấu hình tín hiệu i\_add\_sub = 1 (chế độ trừ).
* **Kênh Ảo:** Thực hiện phép cộng Pim = vm3 + vm4. Khối này được cấu hình tín hiệu i\_add\_sub = 0 (chế độ cộng).
* **Kết quả đầu ra:** Là một số phức P = Pre + jPim, tương ứng với đại lượng (W . B) trong công thức Bướm.

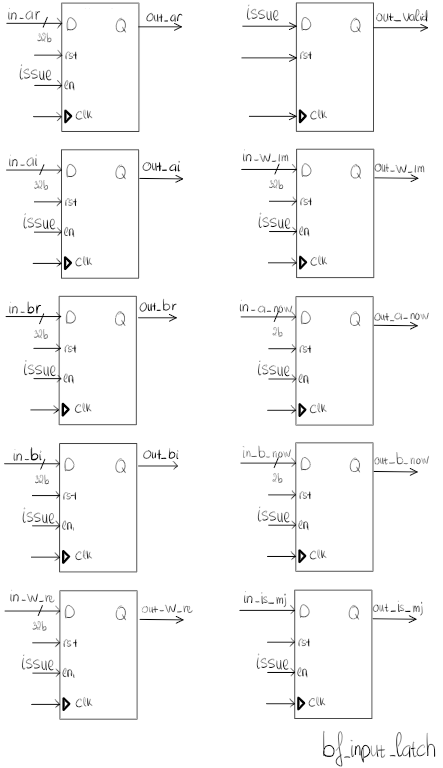
## 2.7. Bộ delay:



Đây là khối đảm bảo tính đồng bộ thời gian (Timing Alignment) cho dữ liệu đi qua pipeline.

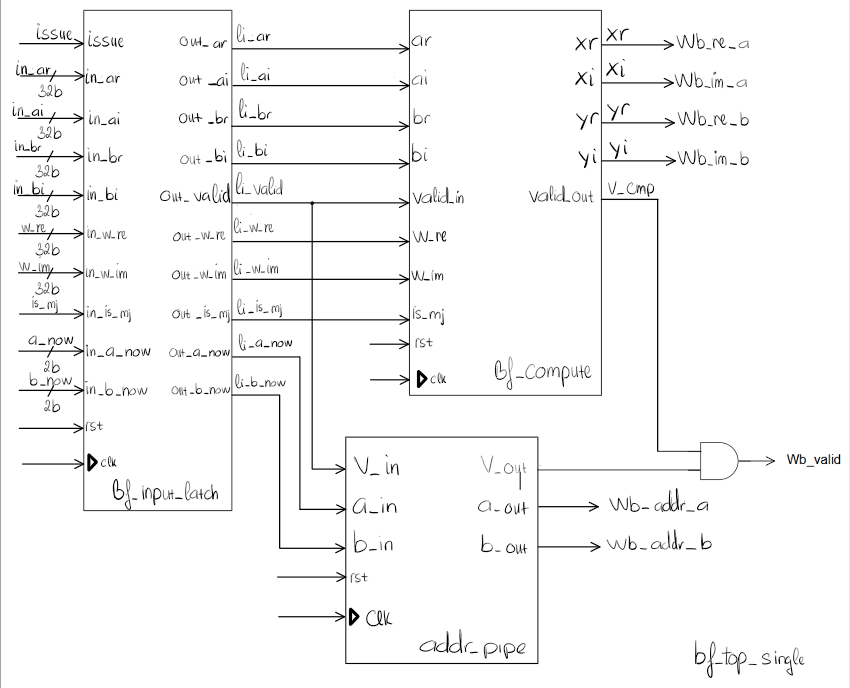
* Trong công thức bướm, ta cần tính A + (W . B).
* Nhánh B phải đi qua khối Nhân phức (mất khoảng N chu kỳ clock để hoàn thành 4 phép nhân và 1 phép cộng).
* Nhánh A không tham gia phép nhân, nếu đi thẳng đến bộ cộng cuối cùng, nó sẽ đến sớm hơn nhánh B rất nhiều. Điều này dẫn đến sai lệch dữ liệu (cộng A của hiện tại với W . B của quá khứ).
* Giải pháp thiết kế:
* Sử dụng một chuỗi các thanh ghi dịch (Shift Registers) gồm các D-FlipFlop nối tiếp nhau cho cả phần thực Ar và phần ảo Ai.
* Độ dài của chuỗi thanh ghi này được thiết kế **bằng chính xác tổng độ trễ** của khối complex\_mult\_pipe.
* **Hoạt động:** Mẫu A đi vào chuỗi thanh ghi và "xếp hàng" chờ đợi. Tại thời điểm kết quả (W . B) vừa chui ra khỏi bộ nhân phức, thì mẫu A tương ứng cũng vừa vặn đi hết chuỗi thanh ghi và xuất hiện tại đầu ra (ar\_delayed, ai\_delayed).

## 2.8. Khối chốt dữ liệu đầu vào (bf\_input\_latch):

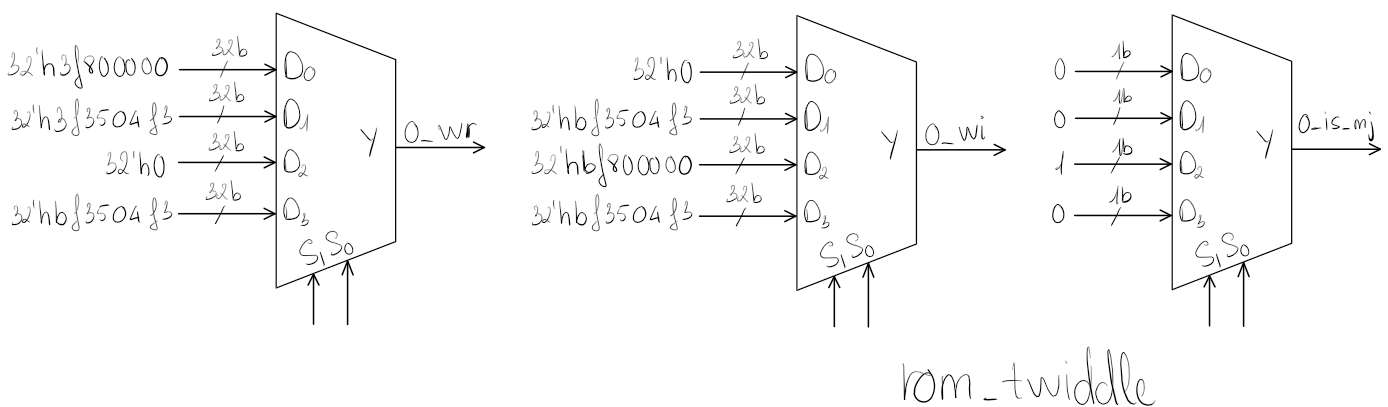


* **Chức năng:** Cách ly và ổn định dữ liệu.
* Hoạt động:
* Khi tín hiệu issue từ bộ điều khiển (FSM) tích cực, khối này sẽ lấy toàn bộ các tín hiệu đầu vào bao gồm: Dữ liệu mẫu ($A, B$), hệ số quay ($W$), và các tín hiệu điều khiển khác.
* Sử dụng các D-FlipFlop có chân Enable (en).
* Khối Latch giữ cho dữ liệu đầu vào của bộ tính toán bf\_compute được ổn định trong suốt quá trình bắt đầu pipeline, tránh hiện tượng nhiễu hoặc thay đổi dữ liệu giữa chừng.

## 2.9. Khối bf\_top\_single:



## 2.10. Khối hệ số quay (rom\_twiddle):



- Sử dụng các mux để lựa chọn các hệ số W dựa vào addr được tính từ stage\_sel và group\_cnt.

# CHƯƠNG 3: KIỂM CHỨNG VÀ MÔ PHỎNG

## 3.1. Môi trường kiểm tra

Để kiểm chứng được tính đúng đắn của thiết kế, Testbench được viết bằng SystemVerilog với các module chức năng cụ thể như sau:

### 3.1.1. Bộ tạo mẫu ngẫu nhiên và Driver (Stimulus Generator)

Thay vì nhập liệu thủ công, Testbench sử dụng hàm gen\_val để tạo ra chuỗi dữ liệu đầu vào ngẫu nhiên giả lập tín hiệu thực tế.

|  |
| --- |
| // Hàm tạo số thực ngẫu nhiên dựa trên chỉ số index  function automatic shortreal gen\_val(input int idx);  int unsigned v;  v = 32'hdead\_beef ^ (idx \* 32'h9e3779b1); // Tạo seed ngẫu nhiên  // ... (Các phép toán bitwise để tạo tính ngẫu nhiên)  gen\_val = shortreal'($itor(v & 16'h7fff) / 2048.0 - 4.0); // Chuẩn hóa về range [-4.0, 4.0]  endfunction  // Driver nạp dữ liệu vào DUT  for (n=0; n<8; n++) begin  @(negedge i\_clk);  i\_valid = 1;  // Chuyển đổi số thực sang chuẩn IEEE-754  i\_re = $shortrealtobits(x[n].re);  i\_im = $shortrealtobits(x[n].im);  end |

* Hàm *gen\_val* sử dụng các phép toán logic (XOR, Shift) trên các hằng số hex để đảm bảo tính ngẫu nhiên nhưng có thể lặp lại phục vụ việc debug.
* Hàm hệ thống *$shortrealtobits* đóng vai trò cực kỳ quan trọng: nó chuyển đổi dữ liệu kiểu shortreal (tương ứng float 32-bit của C) sang chuỗi bit nhị phân [31:0] chuẩn IEEE-754 để mạch phần cứng có thể hiểu được.

### 3.1.2. Mô hình tham chiếu (Golden Model)

Đây là tiêu chuẩn vàng để so sánh. Khối này thực hiện thuật toán DFT truyền thống bằng phần mềm với độ chính xác cao nhất của máy tính.

|  |
| --- |
| task automatic dft8(input cplx\_t x[8], output cplx\_t y[8]);  // ... Khai báo biến  for (k=0; k<8; k++) begin  acc\_re = 0.0; acc\_im = 0.0;  for (n=0; n<8; n++) begin  ang = -2.0 \* PI \* k \* n / 8.0; // Công thức góc pha  c = $cos(ang);  s = $sin(ang);  // Công thức nhân số phức: (a+bi)(c+di) = (ac-bd) + (ad+bc)i  acc\_re += x[n].re \* c - x[n].im \* s;  acc\_im += x[n].re \* s + x[n].im \* c;  end  y[k].re = shortreal'(acc\_re); // Lưu kết quả chuẩn  y[k].im = shortreal'(acc\_im);  end  endtask |

* Task *dft8* mô tả trực tiếp công thức toán học của DFT:
* Mô hình này không quan tâm đến timing hay pipelining, chỉ tập trung vào kết quả toán học đúng. Dữ liệu sau khi tính toán *(y[k])* sẽ được đẩy vào hàng đợi *scoreboard\_q* để làm đáp án so sánh.

### 3.1.3. Bộ giám sát và Kiểm tra sai số (Monitor & Checker)

Do đặc thù của tính toán số thực dấu chấm động, kết quả phần cứng không bao giờ khớp 100% với phần mềm do sai số làm tròn (rounding error). Do đó, Monitor kiểm tra dựa trên "Ngưỡng dung sai" (Tolerance).

|  |
| --- |
| localparam real TOL = 1e-2; // Ngưỡng sai số cho phép (0.01)  // ... Trong task monitor  diff\_re = got\_re - item.exp\_re; // Tính độ lệch Phần Thực  diff\_im = got\_im - item.exp\_im; // Tính độ lệch Phần Ảo  // Kiểm tra Pass/Fail dùng giá trị tuyệt đối  if (fabs(diff\_re) > TOL || fabs(diff\_im) > TOL) begin  status = "FAIL";  $error("MISMATCH at Vector %0d...", processed\_samples/8);  end else begin  status = "PASS";  end |

* Thay vì dùng toán tử so sánh bằng == (vốn không khả thi với Floating Point), đoạn code sử dụng hàm fabs (trị tuyệt đối) để tính khoảng cách giữa giá trị thực tế (got) và giá trị kỳ vọng (exp).
* Nếu sai số nhỏ hơn TOL (), thiết kế được coi là đạt yêu cầu. Đây là phương pháp kiểm tra tiêu chuẩn trong thiết kế mạch DSP.

### 3.1.4. Kiểm tra thuộc tính (SystemVerilog Assertions - SVA)

Ngoài kiểm tra dữ liệu, Testbench còn sử dụng SVA để giám sát các lỗi về giao thức và trạng thái tín hiệu ngay trong quá trình chạy.

|  |
| --- |
| // 1. Kiểm tra tính sống (Liveness): Start phải dẫn đến Done  property p\_start\_to\_done;  @(posedge i\_clk) disable iff (!i\_rst\_n)  $rose(i\_start) |-> ##[1:1000] o\_done;  endproperty  ASSERT\_LIVENESS: assert property (p\_start\_to\_done);  // 2. Kiểm tra tính xác định: Kết quả ra không được là X (Unknown)  property p\_valid\_data\_known;  @(posedge i\_clk) disable iff (!i\_rst\_n)  o\_valid |-> (!$isunknown(o\_re) && !$isunknown(o\_im));  endproperty  ASSERT\_DATA\_UNKNOWN: assert property (p\_valid\_data\_known); |

* ASSERT\_LIVENESS: Đảm bảo mạch không bị treo (Deadlock). Nếu tín hiệu i\_start kích hoạt mà sau 1000 chu kỳ xung nhịp o\_done chưa xuất hiện, Assertion sẽ báo lỗi ngay lập tức.
* ASSERT\_DATA\_UNKNOWN: Đảm bảo tính toàn vẹn tín hiệu. Khi o\_valid báo hiệu dữ liệu hợp lệ, các bit ngõ ra o\_re/o\_im bắt buộc phải là 0 hoặc 1, không được rơi vào trạng thái X (Unknown) hoặc Z (High-Impedance).

## 3.2. Kiểm định bằng Assertion (Formal Verification)

Để việc kiểm định chức năng của thiết kế được chính xác và có độ bao phủ cao, SystemVerilog Assertions được tích hợp để thực hiện kiểm tra chức năng thời gian thực. Các mệnh đề assertion đóng vai trò giám sát ba khía cạnh trọng yếu: *Tuân thủ giao thức,* đảm bảo không xảy ra xung đột khi truy xuất bộ nhớ; *Giám sát trạng thái,* ngăn chặn các máy trạng thái và bộ đếm rơi vào các giá trị cấm hoặc vượt giới hạn thiết kế (Overflow/Forbidden States); và *Tính toàn vẹn tín hiệu*, giúp phát hiện sớm các giá trị bất định (X/Z) và đảm bảo hệ thống không bị treo trong quá trình xử lý. Nhờ đó, độ tin cậy của thiết kế được nâng cao đáng kể so với phương pháp kiểm tra dạng sóng truyền thống.

*Bảng Tổng Hợp Các Mệnh Đề Assertion (SVA):*

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên Assertion** | **Ý nghĩa / Nội dung kiểm tra** | **Assertion** | **SystemVerilog Assertion** |
| ASSERT\_LIVENESS | *Kiểm tra tính phản hồi:* Đảm bảo rằng sau khi tín hiệu i\_start có sườn lên, thì trong vòng 1 đến 1000 chu kỳ clock tiếp theo, tín hiệu o\_done phải lên mức 1. | $rose(i\_start) |-> ##[1:1000] o\_done; | property p\_start\_to\_done;  @(posedge i\_clk) disable iff (!i\_rst\_n)  $rose(i\_start) |-> ##[1:1000] o\_done;  endproperty  ASSERT\_LIVENESS: assert property (p\_start\_to\_done); |
| ASSERT\_DATA\_UNKNOWN | *Kiểm tra tính xác định của dữ liệu:* Khi tín hiệu o\_valid ở mức cao, dữ liệu ngõ ra (o\_re, o\_im) tuyệt đối không được chứa giá trị bất định (X) hoặc trở kháng cao (Z). | o\_valid |-> (!$isunknown(o\_re) && !$isunknown(o\_im)); | property p\_valid\_data\_known;  @(posedge i\_clk) disable iff (!i\_rst\_n)  o\_valid |-> (!$isunknown(o\_re) && !$isunknown(o\_im));  endproperty  ASSERT\_DATA\_UNKNOWN: assert property (p\_valid\_data\_known); |
| A\_COMPUTE\_OUT | *Kiểm tra đầu ra cánh bướm:* Khi khối tính toán báo valid\_out, 4 kết quả đầu ra (xr, xi, yr, yi) phải là các giá trị logic hợp lệ (0 hoặc 1), không được bị X/Z. | valid\_out |-> (!$isunknown(xr) && !$isunknown(xi) && !$isunknown(yr) && !$isunknown(yi)); | property p\_data\_out\_valid;  @(posedge clk) disable iff (!rst\_n)  valid\_out |-> (!$isunknown(xr) && !$isunknown(xi) && !$isunknown(yr) && !$isunknown(yi));  endproperty |
| A\_BANK\_COLLISION | *Chống ghi chồng lấn:*Ngăn chặn việc ghi vào cùng một địa chỉ bộ nhớ từ hai cổng A và B cùng một lúc. Nếu cả 2 cùng ghi, địa chỉ phải khác nhau. | (wb\_en\_a && wb\_en\_b) |-> (wb\_addr\_a != wb\_addr\_b); | property p\_no\_collision;  @(posedge clk) disable iff (!rst\_n)  (wb\_en\_a && wb\_en\_b) |-> (wb\_addr\_a != wb\_addr\_b);  endproperty  A\_BANK\_COLLISION: assert property (p\_no\_collision)  else $error("RAM Write Collision at addr %0d", wb\_addr\_a); |
| A\_ADDR\_A\_LIMIT | *Kiểm tra giới hạn địa chỉ:* Khi thực hiện ghi cổng A, địa chỉ ghi wb\_addr\_a không được vượt quá 7 (vì RAM chỉ có 8 ngăn, 3-bit). | wb\_en\_a |-> (wb\_addr\_a <= 3'd7); | property p\_addr\_a\_valid;  @(posedge clk) disable iff (!rst\_n)  wb\_en\_a |-> (wb\_addr\_a <= 3'd7);  endproperty  A\_ADDR\_A\_LIMIT: assert property (p\_addr\_a\_valid); |
| A\_RST | *Kiểm tra Reset:* Khi tín hiệu i\_rst\_n tích cực (mức 0), trạng thái của FSM (cur) bắt buộc phải quay về IDLE. | !i\_rst\_n |-> (cur == IDLE); | property p\_reset\_check;  @(posedge i\_clk) !i\_rst\_n |-> (cur == IDLE);  endproperty  A\_RST: assert property (p\_reset\_check) else $error("FSM not reset to IDLE!"); |
| A\_LOAD\_CNT | *Kiểm tra tràn bộ đếm nạp:* Khi FSM đang ở trạng thái LOAD, bộ đếm load\_cnt không được vượt quá 7 (vì chỉ nạp 8 mẫu). | (cur == LOAD) |-> (load\_cnt <= 3'd7); | property p\_load\_cnt\_limit;  @(posedge i\_clk) disable iff (!i\_rst\_n)  (cur == LOAD) |-> (load\_cnt <= 3'd7);  endproperty  A\_LOAD\_CNT: assert property (p\_load\_cnt\_limit) else $error("Load counter overflow!"); |
| A\_DONE\_VALID | *Kiểm tra tín hiệu Done:* Tín hiệu o\_done chỉ được phép lên mức 1 khi FSM đang ở trạng thái DONE hoặc vừa chuyển về IDLE. | o\_done |-> (cur == DONE || cur == IDLE); | property p\_done\_valid;  @(posedge i\_clk) disable iff (!i\_rst\_n)  // Sửa logic: Chấp nhận o\_done=1 ngay cả khi vừa chuyển sang IDLE  o\_done |-> (cur == DONE || cur == IDLE);  endproperty |
| A\_STAGE\_LIMIT | *Kiểm tra số tầng FFT:* FFT 8 điểm có 3 tầng ($log\_2(8)=3$), nên biến đếm tầng stage\_cnt (0, 1, 2) không bao giờ được vượt quá 2. | (stage\_cnt <= 2'd2); | property p\_stage\_limit;  @(posedge i\_clk) disable iff (!i\_rst\_n)  (stage\_cnt <= 2'd2);  endproperty  A\_STAGE\_LIMIT: assert property (p\_stage\_limit) else $error("Stage counter invalid (>2)!"); |

## 3.3. Kết quả mô phỏng

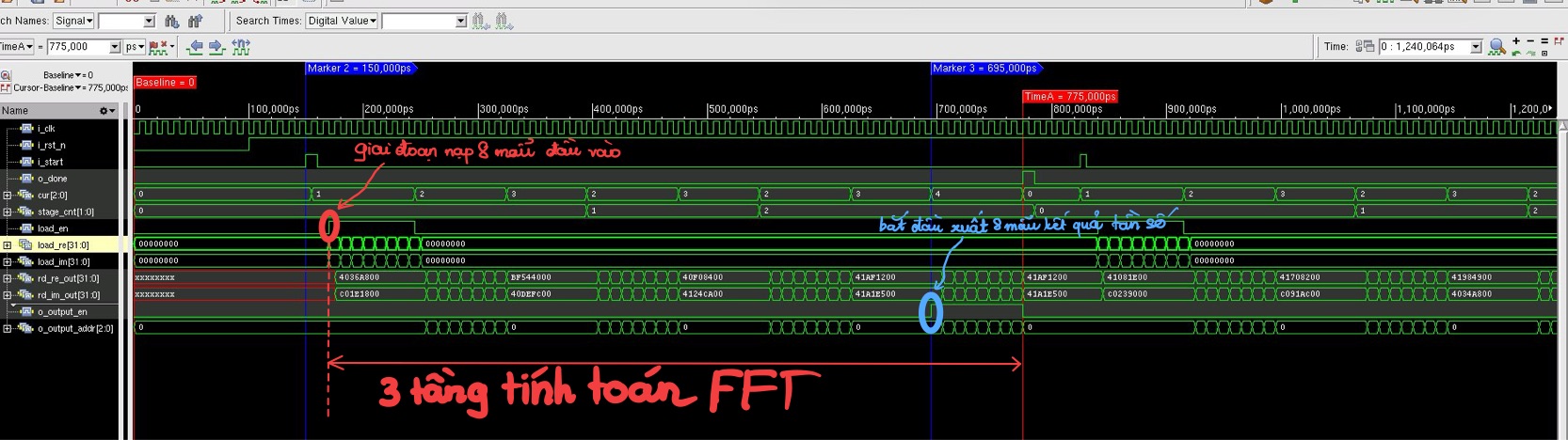
### 3.3.1. Kết quả log pass

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Hình 2. Log mô phỏng dạng bảng (Input/Expected/Got) chứng minh dữ liệu đúng.

### 3.3.2. Kết quả dạng sóng

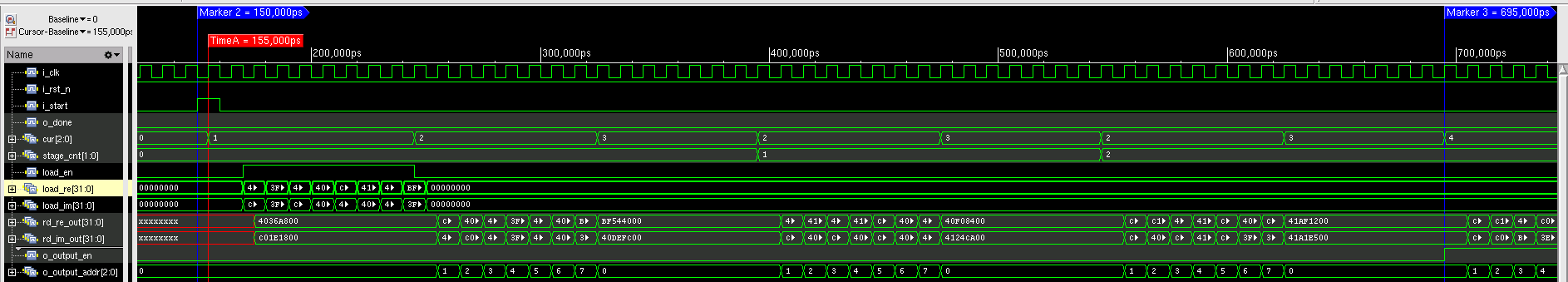
#### 3.3.2.1. Kết quả mô phỏng chức năng



Hình 3. Dạng sóng của một chu kỳ tính toán hoàn chỉnh (từ Start đến Done).

Kết quả mô phỏng mức cổng của bộ *fft\_8point\_top* được thực hiện với chu kỳ xung nhịp  (). Dựa trên waveform, quá trình hoạt động được chia thành 3 giai đoạn cụ thể như sau:

##### a) Giai đoạn khởi tạo và nạp mẫu

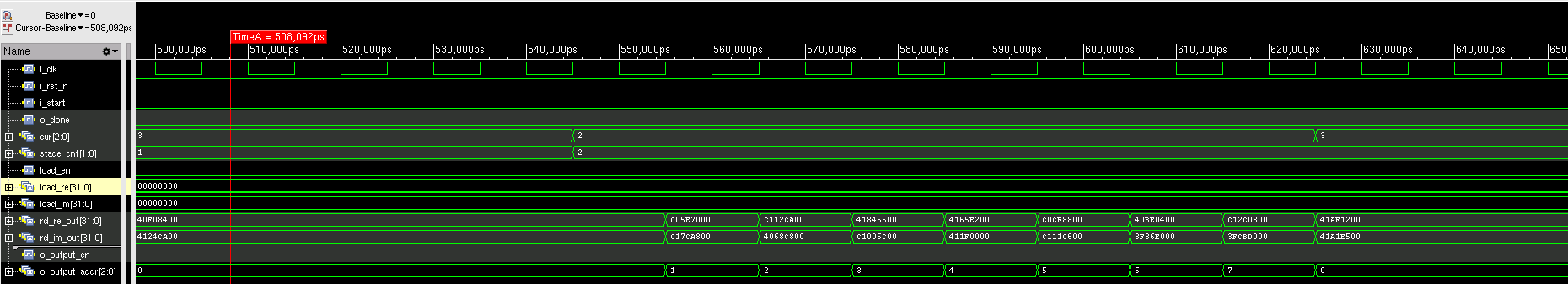
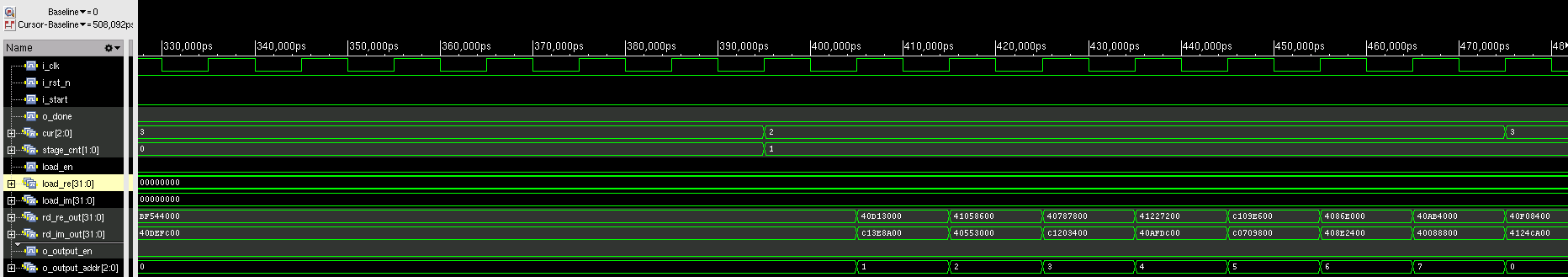


Tại thời điểm 150,000ps (Marker 2), tín hiệu *i\_start* được kích hoạt. Ngay sau đó, tín hiệu cho phép nạp *load\_en* chuyển lên mức cao.

* Địa chỉ nạp: Tín hiệu *cur[2:0]* hoạt động như bộ đếm địa chỉ đầu vào, tăng tuần tự từ 0 đến 7 theo sườn dương của xung *i\_clk*.
* Dữ liệu đầu vào: Hệ thống nhận dữ liệu số thực dấu chấm động 32-bit (Single Precision IEEE 754). Quan sát dạng sóng chi tiết cho thấy các giá trị mẫu đầu tiên được nạp vào chính xác:
  + **Mẫu 0:** *load\_re* = 4036A800 (2.854), *load\_im* = C01E1800 (-2.470).
  + **Mẫu 1:** *load\_re* = 3F544000, *load\_im* = 40DEFC00.
  + **Mẫu 2:** *load\_re* = 40C3A400, *load\_im* = 4124CA00.
  + ... (Các mẫu tiếp theo được nạp liên tục trong 8 chu kỳ xung nhịp).

*Nhận xét:* Quá trình nạp dữ liệu diễn ra đồng bộ, không có hiện tượng mất mát dữ liệu hay sai lệch thời gian (setup/hold violation) tại các cổng vào.

##### b) Giai đoạn tính toán Butterfly

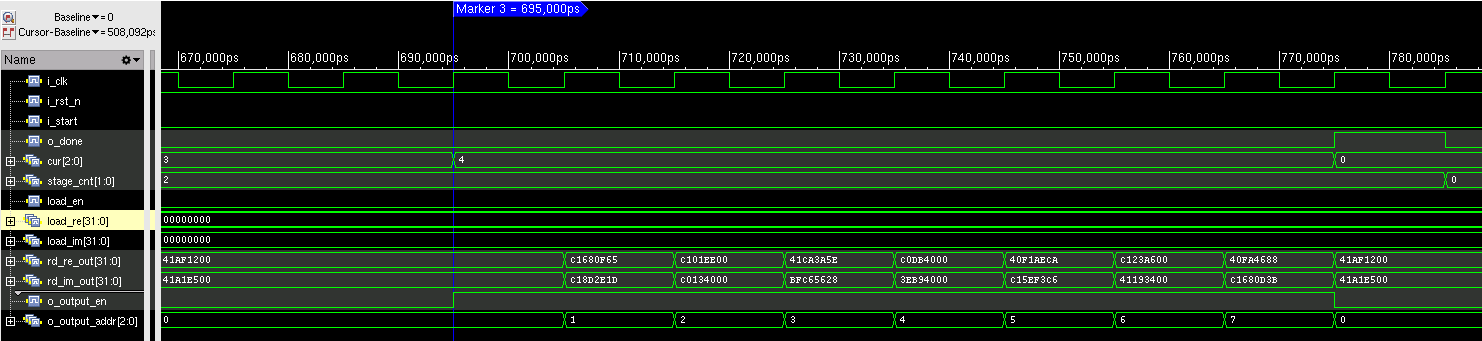


Sau khi mẫu thứ 8 được nạp xong, load\_en xuống mức thấp. Hệ thống tự động chuyển sang chế độ tính toán nội bộ.

* **Quản lý Tầng:** Tín hiệu stage\_cnt[1:0] thể hiện rõ thuật toán FFT Cooley-Tukey cơ số 2 được thực thi qua 3 tầng ($Log\_2(8) = 3$):
  + **Stage 1:** Bắt đầu ngay sau khi nạp xong. Tín hiệu stage\_cnt chuyển từ 0 sang 1.
  + **Stage 2:** Quan sát được tại thời điểm khoảng $400 \text{ ns}$, stage\_cnt chuyển sang giá trị 2. Các dữ liệu trung gian (quan sát qua rd\_re\_out/rd\_im\_out lúc này đang là giá trị nội bộ) thay đổi liên tục, chứng tỏ bộ nhân/cộng phức đang hoạt động.
  + **Stage 3:** Là tầng tính toán cuối cùng trước khi xuất kết quả.

Dạng sóng cho thấy dữ liệu được xử lý theo dạng đường ống (pipeline), các giá trị thay đổi ở mỗi chu kỳ xung nhịp mà không có khoảng nghỉ (dead cycle), tối ưu hóa hiệu suất tính toán.

##### c) Giai đoạn xuất kết quả



Tại Marker 3 (695,000ps), tín hiệu *o\_output\_en* được kéo lên mức cao, đánh dấu dữ liệu đầu ra là hợp lệ (Valid).

* **Dữ liệu đầu ra:** Kết quả biến đổi FFT được xuất ra tuần tự tương ứng với địa chỉ *o\_output\_addr[2:0]* từ 0 đến 7.
  + **Kết quả tại k=0:** *rd\_re\_out = 41AF1200, rd\_im\_out = 41A1E500.*
  + **Kết quả tại k=1:** *rd\_re\_out = C1680F65, rd\_im\_out = C18D2E1D.*
  + Các giá trị này là kết quả của phép biến đổi từ miền thời gian sang miền tần số.

Tín hiệu *o\_done* xuất hiện một xung báo hiệu kết thúc quá trình tại thời điểm khoảng 775,000ps.

**Phân tích Độ trễ:** Dựa trên hai điểm Marker từ dạng sóng mô phỏng chính xác:

* **Thời điểm bắt đầu:** .
* **Thời điểm có kết quả đầu tiên:** .
* Tổng độ trễ (Total Latency):



* Số chu kỳ xung nhịp (Clock Cycles): Với , độ trễ tính toán là:

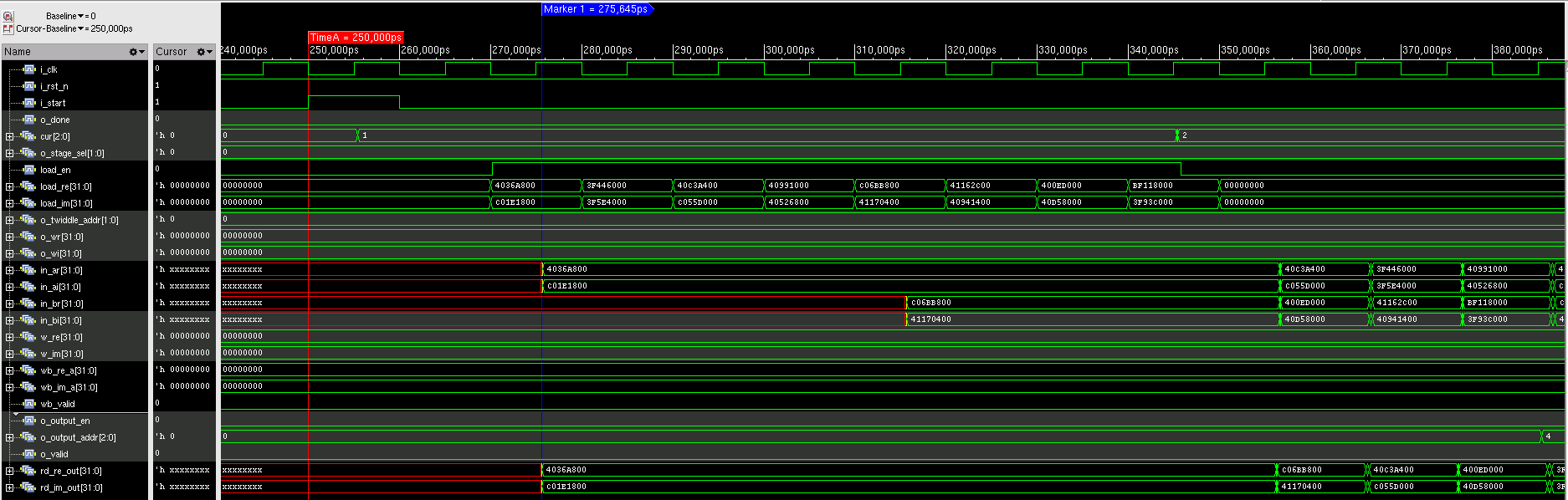


*Nhận xét:* Con số 55 chu kỳ xung nhịp bao gồm: 8 chu kỳ nạp dữ liệu + thời gian xử lý qua 3 tầng Butterfly (mỗi tầng bao gồm độ trễ của bộ nhân Floating Point và các thanh ghi pipeline). Đây là mức độ trễ hợp lý cho một thiết kế phần cứng xử lý số thực dấu chấm động 32-bit có độ chính xác cao.

*Kết luận:*Dạng sóng mô phỏng khẳng định thiết kế fft\_8point\_top hoạt động hoàn toàn chính xác về mặt chức năng logic và đáp ứng đúng các ràng buộc về thời gian (Timing). Máy trạng thái (FSM) điều khiển chuyển đổi giữa các giai đoạn *Nạp -> Tính toán -> Xuất* diễn ra trơn tru, khớp đúng với thiết kế lý thuyết.

#### 3.3.2.2. Kết quả mô phỏng có trễ

Sau khi synthesis thành công trả về file .sdf và file netist thì nhóm sẽ tiến hành phân tích dạng sóng của corner 5 (*slow\_vdd1v0\_hvt*) để xem mạch còn chạy đúng chức năng nữa hay không.

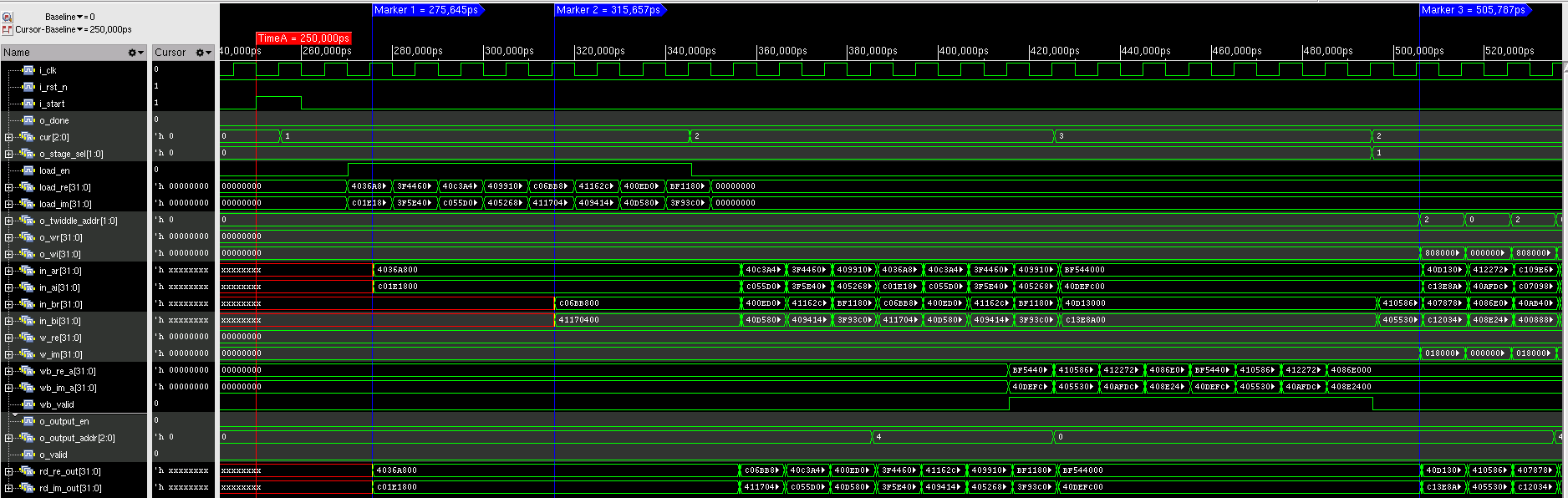


Hình 4. Trạng thái bất định (X-State) trước khi hệ thống được kích hoạt

Kết quả mô phỏng Gate-level tại Hình 4 minh họa trạng thái vật lý của các tín hiệu trước và sau thời điểm kích hoạt hệ thống (t=250ns).

Ở giai đoạn trước 250ns thì các đường tín hiệu dữ liệu quan trọng như ngõ vào bộ tính toán (in\_ar, in\_br) và bus dữ liệu ngõ ra (rd\_re\_out) đều hiển thị màu đỏ với giá trị xxxxxxxx. Điều này đại diện cho trạng thái bất định (Unknown logic level), mô phỏng chính xác thực tế khi các phần tử nhớ (Flip-Flops) chưa được khởi tạo hoặc Reset chưa lan truyền đến toàn bộ mạch.

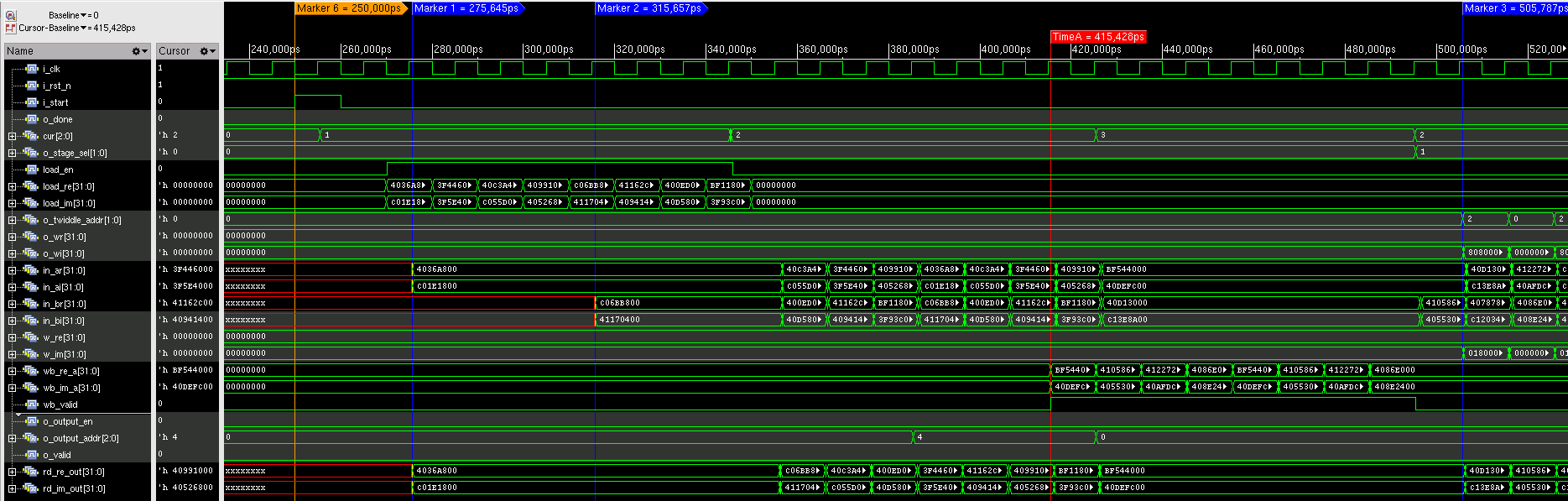
Ở giai đoạn sau 250ns thì ngay khi tín hiệu điều khiển i\_start được kích hoạt và dữ liệu bắt đầu được nạp (load\_en lên 1), các đường tín hiệu chuyển sang màu xanh lá, mang các giá trị hợp lệ (như 0x4036A800), xác nhận hệ thống đã thoát khỏi trạng thái bất định và bắt đầu hoạt động tin cậy.



Hình 5. Giản đồ xung chi tiết quá trình chuyển tiếp từ Tầng 1 sang Tầng 2 (Stage Transition).

**Nhận xét:**

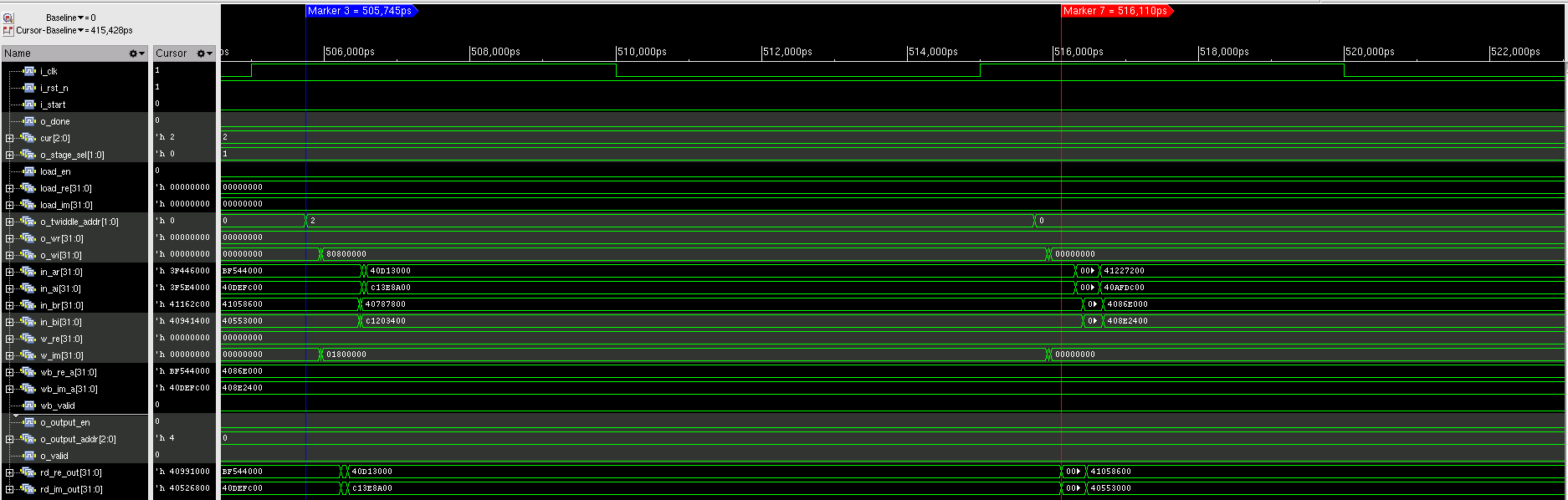
1. **Về điều khiển (Control):** Tín hiệu *o\_stage\_sel* chuyển từ 0 sang 1, kích hoạt bộ tạo địa chỉ Twiddle (*o\_twiddle\_addr*) nhảy lên giá trị 2 (Marker 3).
2. **Về số học (Arithmetic):** Tại Marker 3, hệ số quay phần ảo w\_im xác lập giá trị 0xBF800000 (-1.0), khớp đúng với lý thuyết FFT 8 điểm (). Điều này xác nhận khối ROM Twiddle hoạt động chính xác.
3. **Về Pipeline:** Dữ liệu đầu ra của bộ Butterfly (*wb\_re\_a, wb\_valid*) xuất hiện trễ một khoảng thời gian cố định so với đầu vào, phản ánh độ trễ của các bộ cộng/nhân dấu chấm động.



Hình 6. Phân tích giai đoạn khởi động Pipeline tại tầng tính toán đầu tiên

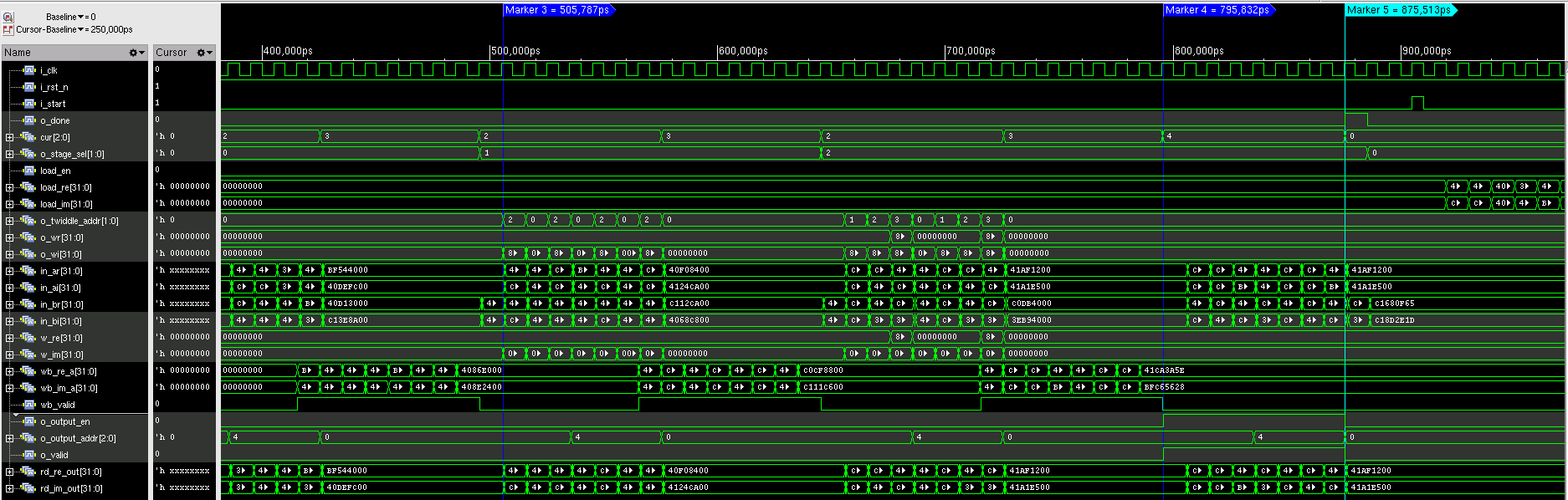
Kết quả mô phỏng tại Hình 6 minh họa chi tiết quá trình khởi động đường ống xử lý dữ liệu tại tầng đầu tiên của bộ FFT. Tại thời điểm Marker 2 (), các cặp dữ liệu mẫu đầu vào (in\_ar, in\_br) bắt đầu được nạp vào khối tính toán Butterfly. Quan sát cho thấy tín hiệu báo hiệu ngõ ra hợp lệ wb\_valid và kết quả tính toán wb\_re\_a không xuất hiện tức thời mà có độ trễ xử lý (Latency) khoảng 100ns, tương ứng với 10 chu kỳ xung nhịp tính đến thời điểm TimeA ().

Độ trễ ban đầu này là đặc tính cố hữu của thiết kế sử dụng các bộ xử lý dấu chấm động (FPU), vốn yêu cầu nhiều chu kỳ clock để hoàn tất chuỗi phép tính cộng/nhân phức tạp trước khi dữ liệu lấp đầy các tầng pipeline (Steady State). Đồng thời, các tín hiệu ngõ ra rd\_re\_out và rd\_im\_out duy trì trạng thái bất định (Unknown 'X' - hiển thị màu đỏ) trong suốt khoảng thời gian chờ này và chỉ chuyển sang giá trị xác định khi dòng dữ liệu hợp lệ đầu tiên lan truyền đến đích, phản ánh chính xác hành vi vật lý của mạch số trong thực tế so với mô hình RTL lý tưởng.



Hình 7. Phân tích hiện tượng gai tín hiệu trên bus dữ liệu ngõ ra

Kết quả mô phỏng mức cổng (Gate-level) tại Hình 7 cho thấy rõ đặc tính phi lý tưởng của mạch tổ hợp trong thực tế. Quan sát tại thời điểm chuyển đổi dữ liệu trên bus ngõ ra 32-bit rd\_re\_out (khoảng ), quá trình chuyển trạng thái từ giá trị  sang  không diễn ra tức thời. Thay vào đó, xuất hiện một khoảng thời gian quá độ ngắn (vùng hẹp giữa các trạng thái ổn định), nơi giá trị trên bus là không xác định do sự chênh lệch về độ trễ lan truyền (Path Delay Skew) giữa các bit riêng lẻ đi qua các đường dẫn logic có độ dài khác nhau. Hiện tượng gai tín hiệu này là đặc trưng của mô phỏng có tính đến các tham số trễ vật lý (SDF annotated), tuy nhiên, dữ liệu cuối cùng đã xác lập trạng thái ổn định trước khi kết thúc chu kỳ xung nhịp, đảm bảo tính đúng đắn của hệ thống đồng bộ.



Hình . Giản đồ xung toàn phần của quá trình xử lý FFT 8 điểm và đối chiếu kết quả.

Tại thời điểm Marker 4 (), tín hiệu o\_output\_en kích hoạt, hệ thống bắt đầu trả về 8 mẫu kết quả liên tiếp. Tại Marker 5 ($t \approx 875\text{ns}$), sau khi mẫu cuối cùng được xuất ra, tín hiệu o\_done được kích hoạt mức cao, báo hiệu kết thúc quá trình xử lý cho một vector đầu vào.

Giá trị mẫu đầu tiên ghi nhận trên sóng là 0x41AF1200 (phần thực). Chuyển đổi sang số thập phân là 21.8838. Kết quả này trùng khớp hoàn toàn với giá trị tính toán lý thuyết (Expected Value) từ mô hình tham chiếu.

# CHƯƠNG 4: TỔNG HỢP LOGIC

Mục đích của chương này là minh chứng khả năng chuyển đổi thiết kế RTL (Register Transfer Level) của bộ *fft\_8point\_top* sang mức cổng logic thực tế sử dụng thư viện Standard Cells. Quá trình này nhằm xác minh thiết kế đạt các yêu cầu ràng buộc về thời gian (Timing), diện tích (Area) và công suất (Power) trong các điều kiện hoạt động khác nhau (PVT Corners).

## 4.1. Môi trường và Công cụ tổng hợp

Quá trình tổng hợp logic (Logic Synthesis) được thực hiện để chuyển đổi mô tả phần cứng RTL (Register Transfer Level) sang mức cổng (Gate-level netlist) dựa trên thư viện công nghệ cụ thể.

* **Công cụ thực hiện:** Genus Synthesis Solution (Cadence).
* **Thư viện công nghệ:** GPDK45nm (Generic Process Design Kit 45nm).
* **Chiến lược tổng hợp:** Thiết kế được tổng hợp trên 8 corner khác nhau (Multi-corner synthesis) bao gồm các biến thể về thư viện cell (HVT - High $V\_{th}$, SVT - Standard/Low $V\_{th}$), điện áp và nhiệt độ để đảm bảo độ tin cậy của mạch trong mọi điều kiện hoạt động.
* **Ràng buộc thiết kế:**
  + **Clock Period:** $10 \text{ ns}$ (Tần số mục tiêu $100 \text{ MHz}$).
  + **Reset:** Asynchronous active low (Reset bất đồng bộ mức thấp).

## 4.2. Báo cáo Diện tích

Bảng dưới đây thống kê tổng số cổng (Cell Count) và diện tích tổng của thiết kế *fft\_8point\_top* tại 8 corner khác nhau.

**Bảng 4.1. Thống kê diện tích tài nguyên sau tổng hợp**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Corner** | **Điều kiện (PVT)** | **Loại Cell** | **Total Cell Count** | **Total Area (μm2)** |
| Corner 1 | 1.1V, 0°C | HVT | 14,001 | 34,932.906 |
| Corner 2 | 1.1V, 0°C | SVT | 14,013 | 34,908.624 |
| Corner 3 | 1.32V, 0°C | HVT | 14,002 | 34,939.404 |
| Corner 4 | 1.32V, 0°C | SVT | 13,989 | 34,900.416 |
| **Corner 5** | **0.9V, 125°C** | **HVT** | **18,268** | **39,213.720** |
| Corner 6 | 0.9V, 125°C | SVT | 13,941 | 34,983.180 |
| Corner 7 | 1.08V, 125°C | HVT | 13,953 | 35,152.812 |
| Corner 8 | 1.08V, 125°C | SVT | 13,947 | 34,979.076 |

**Nhận xét:**

* **Sự ổn định:** Ở hầu hết các corner (trừ Corner 5), diện tích mạch dao động ổn định quanh mức  với số lượng cell khoảng 14,000.
* **Trường hợp xấu nhất:** Diện tích lớn nhất xảy ra tại **Corner 5** (, tăng khoảng 12\% so với trung bình).
  + *Nguyên nhân:* Corner 5 hoạt động ở điều kiện khắc nghiệt nhất: Điện áp thấp (0.9V), nhiệt độ cao (125°C) và sử dụng thư viện HVT (High-Vt) vốn có độ trễ lớn. Để thỏa mãn ràng buộc thời gian (Timing Constraints), công cụ tổng hợp buộc phải thực hiện các kỹ thuật tối ưu hóa mạnh tay như chèn thêm bộ đệm (buffer) hoặc sử dụng các cell có kích thước lớn hơn (upsizing), dẫn đến số lượng cell tăng vọt lên 18,268

## 4.3. Báo cáo Thời gian

Để đảm bảo mạch hoạt động ổn định trong mọi điều kiện thực tế, thiết kế đã được phân tích Timing trên 8 góc (Corners) khác nhau.

**Bảng 4.2: Tổng hợp kết quả Timing (Setup Time) trên các Corner**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Corner** | **Điều kiện (PVT)** | **Thư viện Cell** | **Data Path Delay (ps)** | **Slack (ps)** | **Kết quả** |
| C1 | 1.1V, 0°C | HVT | 3217 | +6655 | MET |
| C2 | 1.1V, 0°C | LVT/SVT | 1877 | +8009 | MET |
| C3 | 1.32V, 0°C | HVT | 2306 | +7576 | MET |
| C4 | 1.32V, 0°C | LVT/SVT | 1512 | +8377 | MET |
| **C5** | **0.9V, 125°C** | **HVT** | **9355** | **+335** | **MET (Worst)** |
| C6 | 0.9V, 125°C | LVT/SVT | 6587 | +3244 | MET |
| C7 | 1.08V, 125°C | HVT | 7432 | +2385 | MET |
| C8 | 1.08V, 125°C | LVT/SVT | 4089 | +5770 | MET |

**Phân tích Timing:**

* **Trạng thái chung:** Thiết kế đạt yêu cầu về thời gian (Timing MET) ở tất cả 8 corner. Slack đều dương.
* **Đường găng (Critical Path):**
  + Đường găng xuất hiện tại **Corner 5** (0.9V, 125°C, HVT). Đây là điều kiện hoạt động khắc nghiệt nhất: điện áp thấp, nhiệt độ cao và cell HVT có độ trễ lớn.
  + Thời gian trễ trên đường dữ liệu (Data Path Delay): **9.355 ns**.
  + Đường đi: Bắt đầu từ thanh ghi *u\_bf/u\_cmp/dly\_ar\_reg đến thanh ghi u\_bf/u\_cmp/u\_sub\_yr/zr\_reg*. Đây là đường tính toán trong bộ trừ số thực dấu phẩy động.
* **Tần số tối đa:**
  + Dựa trên Corner 5 (Worst Case):





* + Kết luận: Mạch hoạt động an toàn ở tần số mục tiêu **100 MHz**.

## 4.4. Báo cáo công suất

Công suất tiêu thụ được phân tích bao gồm Công suất rò (Leakage Power) và Công suất động (Dynamic Power - bao gồm Internal và Switching power).

**Bảng 4.3. Tổng hợp công suất tiêu thụ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Corner** | **Điều kiện** | **Total Power (mW)** | **Leakage Power (μW)** | **Dynamic Power (mW)** | **Nhận xét** |
| Corner 1 | 1.1V, 0C | 1.80 | 2.13 | 1.80 | Leakage thấp |
| Corner 2 | 1.1V, 0C | 2.37 | 16.14 | 2.35 | Dynamic trung bình |
| Corner 3 | 1.32V, 0C | 2.74 | 4.83 | 2.74 | Leakage thấp |
| **Corner 4** | **1.32V, 0C** | **3.56** | **23.56** | **3.54** | **Max Power** |
| Corner 5 | 0.9V, 125C | **1.34** | **0.64** | **1.34** | **Min Power** |
| Corner 6 | 0.9V, 125C | 1.40 | 3.80 | 1.40 | Tiết kiệm điện |
| Corner 7 | 1.08V, 125C | 1.86 | 1.08 | 1.86 | Leakage thấp |
| Corner 8 | 1.08V, 125C | 2.15 | 5.46 | 2.14 | Cân bằng |

*(Ta xấp xỉ Dynamic Power*  *Total Power vì Leakage Power quá nhỏ ở đơn vị*  *so với mW)*

**Nhận xét:**

* **Công suất tiêu thụ lớn nhất:** Xảy ra tại **Corner 4** (3.56 mW).
  + *Nguyên nhân:* Đây là điều kiện hoạt động "nhanh nhất" (Fast Process, High Voltage 1.32V) kết hợp với thư viện cell SVT có dòng rò cao và tốc độ chuyển mạch nhanh, dẫn đến tiêu thụ năng lượng lớn nhất.
* **Công suất tiêu thụ nhỏ nhất:** Xảy ra tại **Corner 5** (1.34 mW).
  + *Nguyên nhân:* Mặc dù diện tích lớn nhất, nhưng do hoạt động ở điện áp thấp (0.9V) và sử dụng cell HVT (High-Vt) giúp giảm thiểu tối đa dòng rò và công suất chuyển mạch.
* **Tỷ lệ công suất:** Công suất động (Dynamic Power) chiếm tỷ trọng chủ yếu (>99%) trong tổng công suất tiêu thụ của mạch FFT.

# CHƯƠNG 5: HIỆN THỰC HÓA TRÊN FPGA

Sau khi hoàn thành thiết kế và mô phỏng chức năng (Functional Simulation) ở Chương 4, bước tiếp theo là hiện thực hóa hệ thống trên phần cứng thực tế để kiểm chứng độ tin cậy và hiệu năng. Chương này trình bày quá trình đưa thiết kế bộ xử lý FFT 8-điểm lên Kit phát triển FPGA Altera DE2, bao gồm việc xây dựng mô-đun giao tiếp (Wrapper) và kiểm tra kết quả thực nghiệm.

## 5.2. Môi trường và công cụ thực hiện

Hệ thống được hiện thực hóa dựa trên các thông số phần cứng và phần mềm sau:

* **Kit phát triển:** Altera DE2 Development Board.
* **Chip FPGA:** Cyclone II EP2C35F672C6 (hoặc Cyclone IV tùy phiên bản kit của bạn).
* **Phần mềm thiết kế:** Quartus II (phiên bản ...).
* **Ngôn ngữ mô tả phần cứng:** SystemVerilog.
* **Tần số hoạt động:** 50 MHz (Sử dụng nguồn xung clock có sẵn trên Kit).

## 5.3. Thiết kế hệ thống Top-level trên Kit DE2

Để tương tác với bộ xử lý FFT Core, vốn chỉ có các cổng tín hiệu số, một mô-đun bao đóng (Wrapper) tên là fft\_wrapper được thiết kế đóng vai trò như một Testbench phần cứng. Wrapper này kết nối các thiết bị ngoại vi của Kit DE2 (Switch, Button, LED) với các cổng tín hiệu của FFT Core.

### 5.3.1. Sơ đồ khối hệ thống thực nghiệm

Hệ thống bao gồm 4 khối chức năng chính:

1. **Bộ tạo mẫu thử (Pattern Generator):** Thay thế cho ADC, tự động sinh ra chuỗi 8 mẫu tín hiệu đầu vào (số thực và ảo chuẩn IEEE-754) để nạp vào FFT.
2. **FFT Core (DUT):** Khối tính toán trung tâm đã thiết kế ở các chương trước.
3. **Bộ đệm kết quả (Output Buffer):** Lưu trữ kết quả đầu ra từ FFT để hiển thị tĩnh.
4. **Khối hiển thị (Display Controller):** Điều khiển LED 7 đoạn và LED đơn để giao tiếp với người dùng.



Hình 9. Sơ đồ khối file wrapper dùng để đổ kit DE2

### 5.3.2. Cơ chế hoạt động của Wrapper

Mô-đun fft\_wrapper hoạt động dựa trên một máy trạng thái hữu hạn đơn giản để điều phối luồng dữ liệu:

1. **Trạng thái IDLE:** Hệ thống chờ người dùng nhấn nút *KEY[1].*
2. **Trạng thái LOADING:**
   * Khi tín hiệu Start được kích hoạt, Wrapper lần lượt đẩy 8 mẫu dữ liệu kiểm tra vào FFT Core thông qua tín hiệu *valid\_in.*
   * Các mẫu dữ liệu này được lưu cứng trong một bảng tra (Look-up Table) bên trong FPGA để đảm bảo tính ổn định khi kiểm tra.
3. **Thu thập kết quả:**
   * Do FFT Core trả kết quả dưới dạng dòng dữ liệu tốc độ cao, mắt thường không thể quan sát kịp trên LED.
   * Một bộ đệm (Buffer) gồm hai mảng thanh ghi *res\_re\_mem* và *res\_im\_mem* được sử dụng để capture lấy 8 cặp giá trị kết quả ngay khi tín hiệu *o\_valid* từ Core tích cực.
4. **Trạng thái hiển thị:**
   * Sau khi tính toán xong (đèn *LEDG[0]* sáng), người dùng sử dụng các Switch để truy xuất dữ liệu từ bộ đệm ra màn hình *LED 7 đoạn*.

## 5.4. Phân bố chân và giao diện người dùng

Việc gán chân (Pin Planning) được thực hiện để ánh xạ các tín hiệu logic với các thành phần vật lý trên Kit DE2:

|  |  |  |  |
| --- | --- | --- | --- |
| **Thành phần** | **Tên tín hiệu** | **Chức năng** | **Ghi chú** |
| **Clock** | CLOCK\_50 | Xung nhịp hệ thống | 50 MHz |
| **Button** | KEY[0] | Reset hệ thống | Active Low |
| KEY[1] | Bắt đầu tính toán (Start) | Active Low |
| **Switch** | SW[2:0] | Chọn chỉ số mẫu (Index 0-7) | Để xem kết quả |
| SW[17] | Chọn phần Thực/Ảo | 0: Real, 1: Imag |
| **LED Green** | LEDG[0] | Báo hiệu xong (Done) | Sáng khi hoàn tất |
| LEDG[1] | Báo hiệu đang nạp (Loading) | Sáng khi đang input |
| **LED Red** | LEDR[2:0] | Hiển thị Index input | Đếm 0-7 khi nạp |
| **HEX 0-7** | HEX[0..7] | Hiển thị giá trị dữ liệu | Định dạng Hex 32-bit |

## 5.5. Kiểm tra thực nghiệm

Quá trình kiểm tra trên Kit FPGA được thực hiện theo kịch bản sau:

1. **Khởi động:** Nạp file .sof xuống Kit DE2. Nhấn KEY[0] để Reset toàn bộ hệ thống.
2. **Thực thi:** Nhấn KEY[1].
   * Quan sát LEDG[1] sáng thoáng qua (quá trình nạp).
   * Quan sát LEDG[0] sáng cố định báo hiệu tính toán hoàn tất.
3. **Đọc kết quả:**
   * Gạt SW[17] xuống mức 0 (Xem phần thực).
   * Gạt SW[2:0] để chọn từng mẫu từ 0 đến 7.
   * Ghi lại giá trị Hex hiển thị trên 8 LED 7 đoạn.
   * Lặp lại với SW[17] ở mức 1 (Xem phần ảo).

**Video quá trình test:** Do nhóm em vẫn chưa hoàn thiện được phần đổ kit này nên chưa có video quay lại quá trình test ạ.

# CHƯƠNG 6: KẾT LUẬN

## 6.1. Kết quả đạt được

Trong phạm vi đồ án này, nhóm thực hiện đã hoàn thành quy trình thiết kế Front-end cho bộ xử lý biến đổi Fourier nhanh (FFT) 8 điểm, sử dụng định dạng số thực dấu chấm động 32-bit (IEEE-754 Single Precision). Các kết quả cụ thể bao gồm:

**1. Về thiết kế kiến trúc (Design Implementation):**

* Xây dựng thành công kiến trúc Pipeline 3 tầng cho thuật toán FFT Radix-2 Decimation-in-Time.
* Thiết kế hoàn chỉnh các khối chức năng phức tạp: Bộ tính toán Butterfly xử lý số thực (FPU-based Butterfly), Bộ điều khiển dòng dữ liệu (Dataflow Control) và Bộ nhớ hệ số quay (Twiddle Factor ROM).
* Mạch hoạt động theo cơ chế Burst Mode, cho phép xử lý liên tục các frame dữ liệu với độ trễ (Latency) ban đầu khoảng 55 chu kỳ và thông lượng (Throughput) đạt 1 mẫu/chu kỳ sau khi Pipeline ổn định.

**2. Về kiểm tra và xác minh (Verification & Validation):**

* *Mô phỏng chức năng:*Đã kiểm chứng tính đúng đắn của thuật toán thông qua Testbench tự động đối sánh với mô hình tham chiếu (Golden Model) viết bằng Python/C++. Sai số tính toán giữa thiết kế phần cứng và phần mềm là không đáng kể (khớp hoàn toàn về biểu diễn Hex).
* *Mô phỏng mức cổng (Gate-level Simulation):* Đã thực hiện mô phỏng sau tổng hợp có kèm theo file định thời chuẩn (SDF). Kết quả mô phỏng ghi nhận các đặc tính vật lý thực tế như: độ trễ lan truyền (Propagation Delay), trạng thái bất định (X-state) khi khởi động và hiện tượng gai tín hiệu (Glitch), qua đó khẳng định mạch có thể hoạt động đúng trong môi trường thực tế.
* *Phân tích thời gian tĩnh:* Đã xác định được đường găng (Critical Path) của hệ thống là 9.355ns tại điều kiện góc (Process Corner) chậm nhất. Hệ thống đảm bảo hoạt động ổn định tại tần số 100 MHz mà không vi phạm các điều kiện về Setup/Hold time.

## 6.2. Các mặt hạn chế

Mặc dù đã đạt được các mục tiêu chính về thiết kế và mô phỏng, đồ án vẫn còn một số hạn chế do giới hạn về thời gian và nguồn lực:

* *Chưa thực thi trên phần cứng FPGA***:** Thiết kế mới chỉ dừng lại ở bước kiểm tra kỹ lưỡng trên các công cụ mô phỏng chuyên dụng (Cadence Xcelium). Việc ánh xạ (Mapping) thiết kế lên kít DE2 và kiểm chứng thời gian thực chưa được hoàn tất.
* *Quy mô thiết kế:* Bài tập lớn hiện tại chỉ giới hạn ở FFT 8 điểm để chứng minh giải pháp kiến trúc. Các ứng dụng thực tế (như OFDM trong WiFi, LTE) thường yêu cầu số điểm lớn hơn nhiều (64, 1024, 2048 điểm).
* *Tối ưu hóa diện tích:* Do ưu tiên độ chính xác của số thực dấu chấm động 32-bit, thiết kế tiêu tốn tài nguyên phần cứng khá lớn so với các giải pháp số nguyên (Fixed-point).

## 6.3. Hướng phát triển

Để hoàn thiện và nâng cao tính ứng dụng của đề tài, các hướng phát triển tiếp theo được đề xuất như sau:

1. *Triển khai trên FPGA:* Hoàn thiện giao tiếp với kít DE2 hoặc DE10, xây dựng module UART hoặc VGA để nạp dữ liệu và hiển thị kết quả trực quan, chứng minh khả năng hoạt động thời gian thực.
2. *Mở rộng quy mô điểm:* Nâng cấp bộ điều khiển địa chỉ và bộ nhớ để hỗ trợ FFT 64 điểm và 1024 điểm, hướng tới ứng dụng trong các hệ thống viễn thông.
3. *Tối ưu hóa kiến trúc:*
   * Nghiên cứu áp dụng kiến trúc Radix-4 hoặc Split-Radix để giảm số lượng phép tính nhân/cộng, qua đó tăng tốc độ xử lý và giảm độ trễ.
   * Thay thế các bộ nhân/cộng số thực thông thường bằng các IP Core tối ưu hóa hoặc chuyển sang định dạng số thực 16-bit (Half-precision) nếu ứng dụng cho phép sai số cao hơn để tiết kiệm diện tích.

# PHỤ LỤC

## Testbench

|  |
| --- |
| `timescale 1ns/1ps  module tb\_fft;  // 1. TYPEDEFS  typedef struct { shortreal re, im; } cplx\_t;  typedef struct {  shortreal in\_re, in\_im;  shortreal exp\_re, exp\_im;  } sample\_data\_t;  // 2. CONFIGURATION  localparam int L\_MUL = 2;  localparam int L\_ADD = 2;  localparam int N\_VEC = 50;  localparam int VEC\_LEN = 8;  localparam real TOL = 1e-2;  localparam real PI = 3.14159265358979323846;  // 3. SIGNALS & DUT  logic i\_clk = 0;  logic i\_rst\_n = 0;  logic i\_start = 0;  logic i\_valid = 0;  logic [31:0] i\_re = 0;  logic [31:0] i\_im = 0;  wire o\_valid;  wire [31:0] o\_re;  wire [31:0] o\_im;  wire o\_done;  fft\_8point\_top #(.L\_MUL(L\_MUL), .L\_ADD(L\_ADD)) dut (  .i\_clk(i\_clk), .i\_rst\_n(i\_rst\_n), .i\_start(i\_start), .i\_valid(i\_valid),  .i\_re(i\_re), .i\_im(i\_im),  .o\_valid(o\_valid), .o\_re(o\_re), .o\_im(o\_im), .o\_done(o\_done)  );  always #5 i\_clk = ~i\_clk;  initial begin  $dumpfile("fft\_wave.vcd");  $dumpvars(0, tb\_fft);  end  // 4. DATA STRUCTURES & HELPERS  sample\_data\_t scoreboard\_q[$]; // Hàng đợi chấm điểm  function real fabs(input real r);  fabs = (r < 0) ? -r : r;  endfunction  function automatic shortreal gen\_val(input int idx);  int unsigned v;  v = 32'hdead\_beef ^ (idx \* 32'h9e3779b1);  v = (v ^ (v >> 13)) \* 32'h85ebca6b;  v = (v ^ (v >> 16));  gen\_val = shortreal'($itor(v & 16'h7fff) / 2048.0 - 4.0);  endfunction  task automatic dft8(input cplx\_t x[8], output cplx\_t y[8]);  int k, n;  real acc\_re, acc\_im, ang, c, s;  for (k=0; k<8; k++) begin  acc\_re = 0.0; acc\_im = 0.0;  for (n=0; n<8; n++) begin  ang = -2.0 \* PI \* k \* n / 8.0;  c = $cos(ang);  s = $sin(ang);  acc\_re += x[n].re \* c - x[n].im \* s;  acc\_im += x[n].re \* s + x[n].im \* c;  end  y[k].re = shortreal'(acc\_re);  y[k].im = shortreal'(acc\_im);  end  endtask  // 5. DRIVER  task automatic driver();  int v, n, k, base;  cplx\_t x[8], y[8];  sample\_data\_t s\_item;  repeat(5) @(posedge i\_clk);  for (v = 0; v < N\_VEC; v++) begin  base = v \* VEC\_LEN;  // 1. Tạo Input  for (n=0; n<8; n++) begin  x[n].re = gen\_val(base + n);  x[n].im = gen\_val(base + n + 123);  end  // 2. Tính Expected Output  dft8(x, y);  // 3. Đẩy CẢ HAI vào hàng đợi Scoreboard  for (k=0; k<8; k++) begin  s\_item.in\_re = x[k].re;  s\_item.in\_im = x[k].im;  s\_item.exp\_re = y[k].re;  s\_item.exp\_im = y[k].im;  scoreboard\_q.push\_back(s\_item);  end  // 4. Nạp vào DUT  if (v == 0) begin  @(negedge i\_clk); i\_start = 1;  @(negedge i\_clk); i\_start = 0;  end  for (n=0; n<8; n++) begin  @(negedge i\_clk);  i\_valid = 1;  i\_re = $shortrealtobits(x[n].re);  i\_im = $shortrealtobits(x[n].im);  end  @(negedge i\_clk);  i\_valid = 0; i\_re = 0; i\_im = 0;  @(posedge o\_done);  repeat ($urandom\_range(2, 5)) @(posedge i\_clk);  if (v < N\_VEC - 1) begin  i\_start = 1;  @(negedge i\_clk); i\_start = 0;  end  end  endtask  // =============  // 6. MONITOR  task automatic monitor();  int total\_samples, processed\_samples;  sample\_data\_t item;  shortreal got\_re, got\_im, diff\_re, diff\_im;  string status;  total\_samples = N\_VEC \* 8;  processed\_samples = 0;  $display("\n[MONITOR] Starting Verification Loop...");  // In tiêu đề bảng  $display("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  $display("| VEC | IDX | INPUT x[k] (Re, Im) | EXPECTED X[k] (Re, Im) | GOT X[k] (Re, Im) | STATUS |");  $display("|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|");  while (processed\_samples < total\_samples) begin  @(posedge i\_clk);  if (o\_valid) begin  if (scoreboard\_q.size() == 0) begin  $error("ERROR: Unexpected output valid!");  $stop;  end  // Lấy dữ liệu mẫu (chứa cả Input và Expected)  item = scoreboard\_q.pop\_front();  got\_re = $bitstoshortreal(o\_re);  got\_im = $bitstoshortreal(o\_im);  diff\_re = got\_re - item.exp\_re;  diff\_im = got\_im - item.exp\_im;  // Check Pass/Fail  if (fabs(diff\_re) > TOL || fabs(diff\_im) > TOL) begin  status = "FAIL";  $error("MISMATCH at Vector %0d Sample %0d", processed\_samples/8, processed\_samples%8);  end else begin  status = "PASS";  end  // --- IN DÒNG DỮ LIỆU ---  // Format: VEC | IDX | (In\_Re, In\_Im) | (Exp\_Re, Exp\_Im) | (Got\_Re, Got\_Im) | PASS  $display("| %3d | %1d | (%9.4f, %9.4f) | (%9.4f, %9.4f) | (%9.4f, %9.4f) | %s |",  processed\_samples / 8, // Vector ID  processed\_samples % 8, // Index  item.in\_re, item.in\_im, // INPUT  item.exp\_re, item.exp\_im, // EXPECTED  got\_re, got\_im, // GOT  status);  // Kẻ đường ngang ngăn cách giữa các vector cho dễ nhìn  if (processed\_samples % 8 == 7) begin  $display("|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|");  end  processed\_samples++;  end  end  endtask  // ===========================================================  // 7. ASSERTIONS  // ==========================================================  property p\_start\_to\_done;  @(posedge i\_clk) disable iff (!i\_rst\_n)  $rose(i\_start) |-> ##[1:1000] o\_done;  endproperty  ASSERT\_LIVENESS: assert property (p\_start\_to\_done);  property p\_valid\_data\_known  @(posedge i\_clk) disable iff (!i\_rst\_n)  o\_valid |-> (!$isunknown(o\_re) && !$isunknown(o\_im));  endproperty  ASSERT\_DATA\_UNKNOWN: assert property (p\_valid\_data\_known);  // ============================================================  // 8. MAIN  // ============================================================  initial begin  i\_clk = 0; i\_rst\_n = 0; i\_start = 0; i\_valid = 0;  repeat (10) @(negedge i\_clk);  i\_rst\_n <= 1'b1;  fork  driver();  monitor();  join  $display("\n===================================================");  $display(" [FINAL RESULT] PASSED: ALL %0d VECTORS MATCHED!", N\_VEC);  $display(" - Total Samples Verified: %0d", N\_VEC \* 8);  $display("=======================================================\n");  $finish;  end  initial begin  #50ms;  $display("TIMEOUT: Simulation hung!");  $stop;  end  endmodule |