Thiết kế và thực hiện thuật toán FFT 64 điểm trên phần cứng bằng ngôn ngữ Verilog HDL

Nguyễn Minh Hiếu (20151336)
Điện tử 03 – K60
Đại học Bách khoa Hà Nội
hieu.nm151336@sis.hust.edu.vn

Nguyễn Minh Hiếu (20151337)
Điện tử 03 – K60
Đại học Bách khoa Hà Nội
hieu.nm151337@sis.hust.edu.vn

Ngô Văn Quyền (20153086)
Điện tử 08 – K60
Đại học Bách khoa Hà Nội
quyen.nv153086@sis.hust.edu.vn

***Tóm tắt*—Báo cáo này trình bày cở sở toán học về biến đổi Fourier rời rạc (DFT), biến đổi Fourier nhanh (FFT) và cụ thể là FFT 64 điểm, thiết kế, mô phỏng bộ FFT 64 điểm bằng ngôn ngữ Verilog HDL**

Từ khóa—FFT 64 điểm, Verilog HDL, CORDIC

I. GIỚI THIỆU CHUNG

Trong lĩnh vực Điện tử − Viễn thông, xử lí tín hiệu là một công việc rất phổ biến và quan trọng. Cơ sở lí thuyết của việc xử lí tín hiệu được xây dựng dựa trên cơ sở toán học là các phép biến đổi tín hiệu (rời rạc hoặc liên tục) trong các miền khác nhau, cơ bản nhất là miền thời gian và miền tần số

Biến đổi Fourier rời rạc (DFT) là một biến đổi rất thông dụng trong xử lý tin hiệu, trong đó biến đổi Fourier nhanh (FFT) là một thuật toán hiệu quả để tính DFT. Chính vì vậy, nhóm chúng em đã chọn đề tài " Thiết kế và thực hiện thuật toán FFT 64 điểm trên phần cứng bằng ngôn ngữ Verilog HDL" để làm bài tập lớn môn thiết kế hệ nhúng.

Trong báo cáo này, nhóm em sẽ trình bày cở sở toán học của DFT và FFT ở phần II, thiết kế bộ FFT 64 điểm ở phần III, trình bày kết quả mô phỏng ở phần IV và đưa ra kết luận ở phần V.

II. CƠ SỞ TOÁN HỌC CỦA DFT VÀ FFT

A. Biến đổi Fourier rời rạc (DFT)

Biến đội Fourier rời rạc (Discrete Fourier Transform – DFT) là một biến đổi trong giải tích Fourier cho các tín hiệu thời gian rời rạc. Đầu vào của DFT là một chuỗi rời rạc hữu hạn các số thực hoặc số phức, khiến DFT là một công cụ lý tưởng để xử lý thông tin trên các máy tính. Đặc biệt, DFT được sử dụng rộng rãi trong xử lý tín hiệu và các ngành liên quan đến phân tích tần số trong một tín hiệu, để giải phương trình đạo hàm riêng và làm các phép như tích chập [9].

Trong biến đổi DFT, tín hiệu đầu vào rời rạc x[n] N điểm (n=0,1,2,…,N−1) được chuyển thành tín hiệu đầu ra rời rạc X[k] N điểm (k=0,1,2,…,N−1). Dưới đây là công thức biến đổi thuận DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \quad (k = 0, 1, \dots, N-1) \qquad (1)$$

Đặt $e^{-j\frac{2\pi}{N}kn} = W_N^{kn}$, công thức trên có thể viết lại:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad (k = 0, 1, \dots, N-1) \qquad (2)$$

Công thức biến đổi ngược IDFT là:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad (n = 0, 1, \dots, N-1) \qquad (3)$$

Biến đổi DFT và IDFT có thể biểu diễn dưới dạng ma trận. Xét ma trận W_{nxn} như sau:

$$W = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ 1 & W_N^1 & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & & W_N^{2(N-1)} \\ & \vdots & & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix} \qquad (4)$$

Tín hiệu đầu vào và đầu ra được biểu diễn như sau:

$$x = [x[0], x[1], \dots, x[N-1]]^T \qquad (5)$$

$$X = [X[0], X[1], \dots, X[N-1]]^T \qquad (6)$$

Biến đổi DFT và IDFT có thể được biểu diễn dưới dạng:

$$X = Wx \qquad (7)$$

$$x = \frac{1}{N} W^{-1} X \qquad (8)$$

B. Thuật toán FFT Cooley − Tukey cơ số 2

Biến đổi Fourier nhanh (Fast Fourier Transform − FFT) là một thuật toán hiệu quả để biến đổi DFT và IDFT nhanh hơn nhiều so với tính toán theo công thức gốc mà vẫn cho ra cùng kết quả [6]. Có nhiều thuật toán FFT khác nhau sử dụng kiến thức từ nhiều mảng khác nhau của toán học, từ số phức tới lý thuyết nhóm và lý thuyết số, trong báo cáo này nhóm em chỉ nghiên cứu và thực hiện thuật toán FFT Cooley − Tukey cơ số 2 phân chia theo tần số (DIF).

1) Giới hiệu thuật toán FFT Cooley − Tukey cơ số 2

Thuật toán FFT Cooley − Tukey, được đặt theo tên của hai nhà toán học J.W. Cooley và John Tukey. Có 2 dạng thuật toán FFT khác nhau là phân chia theo thời gian (Decimation in Time – DIT) và phân chia theo tần số (Decimation in Frequency − DIF) [6]. Cả hai dạng này được xây dựng dựa trên cơ chế phân chia đệ quy DFT N điểm thành DFT $\frac{N}{2}$ điểm. Hạn chế của thuật toán FFT cơ số 2 là chỉ áp dụng được khi N chia hết cho 2, tuy nhiên thuật toán này lại tỏ ra cực kì hiệu quả trong trường hợp N là lũy thừa của 2 vì khi đó có thể phân chia biến đổi N điểm lặp đi lặp lại đến biến đổi DFT 1 điểm.

2) Thuật toán FFT cơ số 2 phân chia theo tần số

Thuật toán FFT cơ số 2 phân chia theo tần số còn được gọi là thuật toán FFT Sande – Tukey. Thuật toán này là một dạng của thuật toán FFT Cooley – Tukey cơ số 2, sử dụng cơ chế DIF. Thuật toán DIF FFT được xây dựng như sau:

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn} \quad (k = 0, 1, \dots, N-1)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{kn} + \sum_{n=\frac{N}{2}}^{N-1} x[n]W_N^{kn}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{kn} + \sum_{n=0}^{\frac{N}{2}-1} x\left[n+\frac{N}{2}\right] W_N^{kn} W_N^{k\frac{N}{2}}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} (x[n] + (-1)^k x\left[n+\frac{N}{2}\right]) W_N^{kn} \qquad (9)$$

Như vậy X[k] có thể tách thành 2 dãy có chỉ số chẵn và chỉ số lẻ:

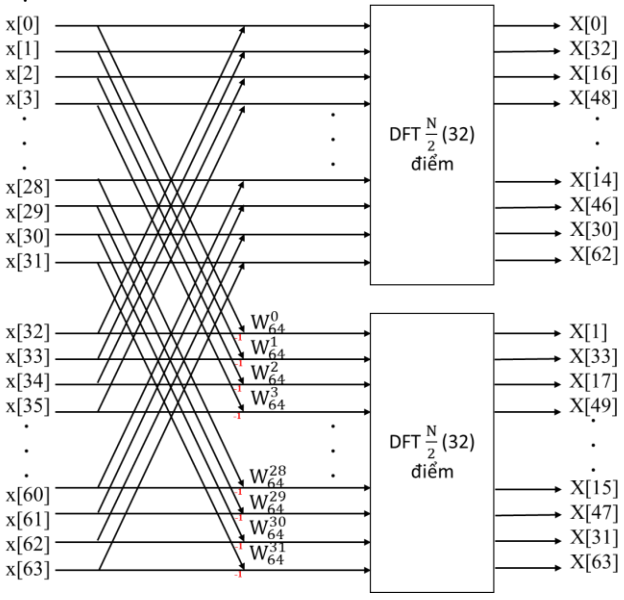
$$X[2k] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n+\frac{N}{2}\right]\right) W_{\frac{N}{2}}^{kn} \qquad (10)$$

$$\left(k = 0, 1, \dots, \frac{N}{2}-1\right)$$

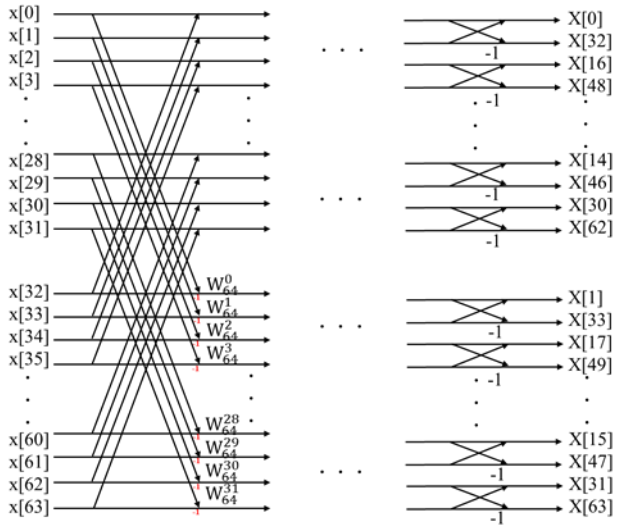
$$X[2k+1] = \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x\left[n+\frac{N}{2}\right]) W_{\frac{N}{2}}^{kn} W_N^n \qquad (11)$$

$$\left(k = 0, 1, \dots, \frac{N}{2}-1\right)$$

Từ đây một biến đổi DFT N điểm có thể tách thành 2 biến đổi DFT $\frac{N}{2}$ điểm, mỗi biến đổi DFT $\frac{N}{2}$ điểm lại có thể tách thành 2 biến đổi DFT $\frac{N}{4}$, tiếp tục phân chia đến DFT 1 điểm. Đây chính là cơ sở của thuật toán FFT cơ số 2. Trong báo cáo này nhóm em thực hiện FFT 64 điểm. Biến đổi DFT 64 điểm được tách thành 2 biến đổi DFT 32 điểm như trên hình 1.



Hình 1. Lưu đồ phân chia DFT 64 điểm thành 2 DFT 32 điểm



Hình 2. Lưu đồ tín hiệu hoàn chỉnh thuật toán FFT 64 điểm

Tiếp tục phân chia 2 DFT 32 điểm thành 4 DFT 16 điểm, chia tiếp thành 8 DFT 8 điểm, 16 DFT 4 điểm, 32 DFT 2 điểm, cuối cùng ta được lưu đồ tín hiệu hoàn chỉnh như trên hình 2. Các X[k] đầu ra bị đảo thứ tự, được chia thành 2 nửa, nửa trên là 32 X[k] chỉ số chẵn, nửa dưới là 32 X[k] chỉ số lẻ. Thứ tự đảo đầu ra được xác định bằng cách đảo bit, ví dụ x[61] (111101) tương ứng đầu ra X[47] (101111).

3) So sánh độ phức tạp tính toán của DFT và DIF FFT

Từ công thức (1) của biến đổi DFT, ta nhận thấy để tính trực tiếp mỗi giá trị của X[k] cần đến N phép nhân 2 số phức và N−1 phép cộng 2 số phức. Để tính một phép nhân 2 số phức lại cần 4 phép nhân 2 số thực và 2 phép cộng 2 số thực. Như vậy để tính N phép nhân 2 số phức cần 4N phép nhân 2 số thực và 2N phép cộng 2 số thực, để tính N−1 phép nhân 2 số phức cần 2N−2 phép cộng 2 số thực. Ngoài ra để tính mỗi giá trị W_N^{kn} $(k = 0, 1, \dots, N-1)$ cần 2 phép tính giá trị các hàm sin, cos, vậy ta cần 2N phép tính giá trị các hàm sin, cos để tính một giá trị X[k] [1]. Tóm lại để tính một giá trị X[k] từ công thức gốc cần số các phép tính như sau:

- N phép nhân phức (4N phép nhân thực và 2N phép cộng thực)
- N−1 phép cộng phức (2N−2 phép cộng thực)
- 2N phép tính giá trị các hàm sin, cos

Đối với thuật toán DIF FFT, từ cơ sở lí thuyết phân chia của thuật toán, tổng số phép nhân 2 số phức là $\frac{N}{2}\log_2 N$ và tổng số phép cộng 2 số phức là $N\log_2 N$, ít hơn rất nhiều so với tính DFT theo công thức gốc. Như vậy FFT có độ phức tạp tính toán thấp hơn nhiều lần tính trực tiếp DFT, dẫn đến tốc độ thực hiện rất nhanh mà vẫn cho ra kết quả đúng

III. Thiết Kế Bộ FFT 64 Điểm

A. Thuật toán CORDIC

1) Giới hiệu thuật toán CORDIC

Thuật toán CORDIC (COordinate Rotation DIgital Computer), còn được gọi là thuật toán Volder, là một thuật toán để tính toán các hàm lượng giác và hyperbolic. Thuật toán này cực kì phù hợp để thực hiện trên phần cứng (ví dụ các kit FPGA) vì nó không yêu cầu bất cứ bộ nhân nào. Các thao tác của thuật toán chỉ gồm cộng, trừ, dịch bit, tra bảng Lookup Table (LUT), vì vậy CORDIC cũng được gọi là thuật toán dịch và cộng. CORDIC có thể tính toán các hàm lượng giác và

hyperbolic với bất cứ độ chính xác mong muốn nào, tùy vào số bit để biểu diễn trong LUT, số bit biểu diễn càng nhiều thì độ chính xác càng cao [7].

Trong triển khai FFT trên phần cứng, CORDIC được sử dụng để thực hiện các phép nhân phức với W_N^n, trong đó yêu cầu tính toán các hàm lượng giác sin, cos, độ lớn và pha. CORDIC xoay quanh ý tưởng về việc xoay pha số phức bằng cách nhân nó với các giá trị không đổi nối tiếp nhau. Tuy nhiên, vì các bội số được sử dụng đều là lũy thừa của 2, nên chỉ cần sử dụng các phép dịch bit và cộng để tính toán, vì vậy không cần sử dụng bất cứ bộ nhân nào.

2) Nguyên lí hoạt động của CORDIC

CORDIC dựa trên một tính chất của phép nhân 2 số phức: argument của tích 2 số phức bằng tổng 2 argument mỗi số phức. Việc nhân phức với W_N^n thực chất là một phép quay argument của số phức đi một góc nào đó mà không làm thay đổi module của số phức. Nguyên lí hoạt động của CORDIC như sau: Giả sử có số phức z=x+yj, nếu muốn cộng thêm argument của z (quay z theo chiều dương, ngược chiều quay kim đồng hồ), ta nhân z với (1+Kj) hoặc nếu muốn trừ đi argument của z (quay z theo chiều âm, cùng chiều quay kim đồng hồ), ta nhân z với (1−Kj), trong đó K= 2^{-i}, với i=0,1,2,… và góc quay được φ=arctan(K) hoặc φ=−arctan(K) tương ứng với hai trường hợp quay z . Số phức nhận được sau khi nhân trong mỗi trường là:

- Nếu quay z theo chiều dương:
  z'=z(1+Kj)=(x+yj)(1+Kj)=(x−yK) + (y+xK)j (12)
- Nếu quay z theo chiều âm:
  z'=z(1−Kj)=(x+yj)(1−Kj)=(x+yK) + (y−xK)j (13)

Vì K=2^{-i} nên việc nhân một số với K thực chất là chia số đó cho 2^i, điều này có thể thực hiện qua phép dịch phải (>>) i bit, vì vậy nên ta không cần sử dụng bộ nhân. Trên thực tế ta sử dụng phép dịch phải số học (>>>) i bit để giữ dấu của số bị dịch bit. Các phương trình (12) và (13) có thể viết lại [3], [4]:

- Nếu quay z theo chiều dương:
  z'=(x−y>>>i) + (y+x>>>i)j (14)
- Nếu quay z theo chiều âm:
  z'=(x+y>>>i) + (y−x>>>i)j (15)

Ta nhận thấy K= 2^{-i} luôn nhỏ hơn 1 nên góc φ luôn nhỏ hơn 45°. Khi cần quay một góc bất kì trong khoảng −90° đến 90°, ta quay số phức lần lượt theo các góc φ theo chiều dương hoặc âm đến khi đạt độ chính xác mong muốn, số phức cuối cùng nhận được là:

$$u=z(1\pm j)(1\pm 2^{-1}j)(1\pm 2^{-2}j)\dots \quad (16)$$

Tuy nhiên, CORDIC sẽ làm module của z tăng lên $\sqrt{(2(1+(2^{-1})^2)(1+(2^{-2}))^2}$ ... lần, số lần tăng lần được gọi là CORDIC Gain và nó phục thuộc vào số lần quay z để đạt được độ chính xác mong muốn. Công thức tổng quát tính CORDIC Gain nếu thực hiện quay z n lần là [2]:

$$A_n = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \quad (17)$$

Bảng 1 là các giá trị góc quay φ (tính theo độ) và CORDIC Gain tương ứng với i=0:7 và n=1:8. Vì module tăng lên nên trong khi tính toán ta phải chia số cần tính cho CORDIC Gain, điều này sẽ được thực hiện nhờ phép dịch phải logic bit sao cho kết quả gần đúng nhất. Thuật toán CORDIC cho FFT 64 điểm sẽ được trình bày kĩ hơn ở phần B.

Bảng 1. Các giá trị φ và n với i=0:7 và n=1:8

i	n	φ (arctan(2^{-i}))	CORDIC Gain
0	1	45°	1.41421356
1	2	26.56505°	1.58113883
2	3	14.03624°	1.62980060
3	4	7.12502°	1.64248407
4	5	3.57633°	1.64568892
5	6	1.78991°	1.64649228
6	7	0.89517°	1.64669325
7	8	0.44761°	1.64674351

B. Kiến trúc bộ FFT 64 điểm

1) Khối CORDIC

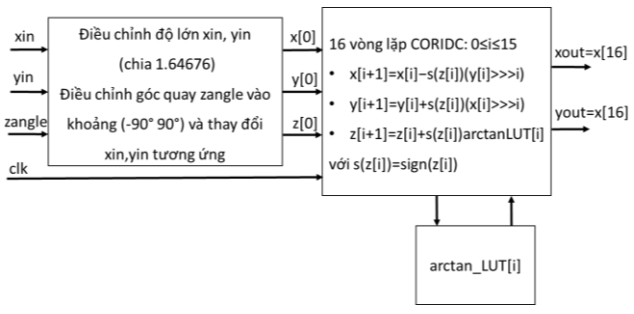
Khối CORDIC được sử dụng tính toán gần đúng nhất phép nhân phức với W_N^n. Ở bài tập lớn này, nhóm em sử dụng 32 bit biểu diễn trong LUT, trong đó bit [31] có trọng số −180°, bit [30] có trọng số 90°, các bit[29] đến bit[0] có trọng số $(\frac{90}{2^m})$ °, với m=1,2,…,30 tương ứng. Ví dụ chuỗi 00010010111001000000010100011101 sẽ biểu diễn gần đúng góc arctan(2^{-1})=26.56505°, bởi vì:

$$26.56505 \approx \frac{90}{2^2}+\frac{90}{2^5}+\frac{90}{2^7}+\frac{90}{2^8}+\frac{90}{2^9}+\frac{90}{2^{12}}+\frac{90}{2^{20}}+\frac{90}{2^{22}}+\frac{90}{2^{26}}+\frac{90}{2^{27}}+\frac{90}{2^{28}}+\frac{90}{2^{30}} \quad (18)$$

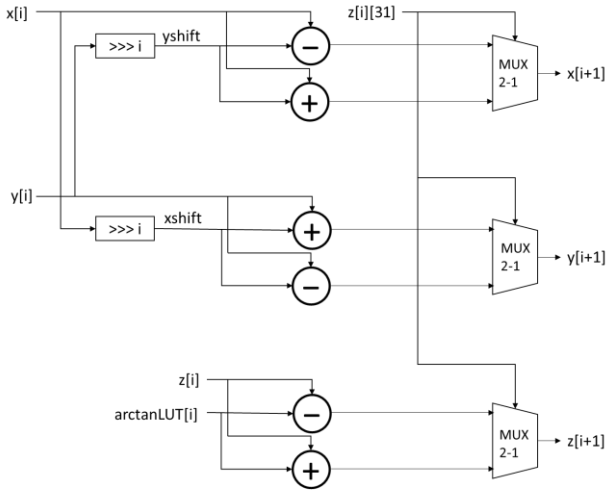
Nhóm em biểu diễn đầu vào và đầu ra đều là các số có dấu 16 bit, biểu diễn số âm theo kiểu mã bù 2. Vì vậy, khi thực hiện CORDIC sẽ quay 16 lần, giá trị CORDIC Gain là:

$$A_{64} = \prod_{i=0}^{63} \sqrt{1+2^{-2i}} = 1.64676 \quad (19)$$

Do đó, sau bộ CORDIC ta phải chia kết quả cho A_{64}, hay nhân với $\frac{1}{A_{64}} = 0.60725 \approx 2^{-1}+2^{-4}+2^{-5}+2^{-7}+2^{-8}+2^{-10}+2^{-11}+2^{-12}+2^{-14}$, việc này được thực hiện nhờ phép dịch phải số học bit nên không cần bộ nhân nào. Hình 3 thể hiện thuật toán khối CORDIC, với xin, yin, zangle, clk, xout, yout lần lượt là phần thực, phần ảo số đầu vào, góc quay, xung clock, phần, phần ảo số đầu ra. Trong 1 vòng lặp, nếu z[i]>0 thì thực hiện nhân (x+y)(1+2^{-i}j) (quay theo chiều dương) và trừ z[i] cho arctanLUT[i], nếu z[i]<0 thì thực hiện nhân (x+y)(1−2^{-i}j) (quay theo chiều âm) và cộng z[i] với arctanLUT[i], từ đó tính x[i+1], y[i+1], z[i+1], đầu ra x[16], y[16] sau 16 vòng lặp chính là đầu ra khối CORDIC [4].



Hình 3. Thuật toán khối CORDIC



Hình 4. Kiến trúc một vòng lặp CORDIC

Hình 4 là kiến trúc một vòng lặp trong CORDIC. Trong một vòng lặp sử dụng 3 bộ MUX 2−1 để cho ra các đầu ra đưa vào vòng lặp tiếp theo. Việc tính toán các giá trị cũng chỉ có các bộ cộng trừ và dịch phải số học bit làm giảm độ phức tạp phần cứng rất nhiều so với dùng các bộ nhân

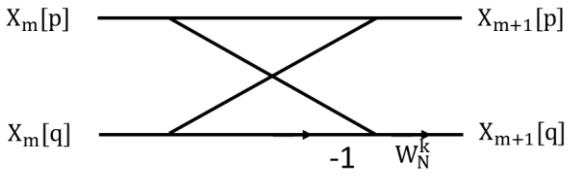
2) Khối Butterfly

Khối Butterfly được xây dựng từ sơ đồ cánh bướm cho DIF FFT cơ số 2 (hình 5) [1]. 2 số phức đầu vào $X_m[p]$ và $X_m[q]$ của stage m được dùn để tính toán 2 đầu ra $X_{m+1}[p]$ và $X_{m+1}[q]$ của stage m+1, với:

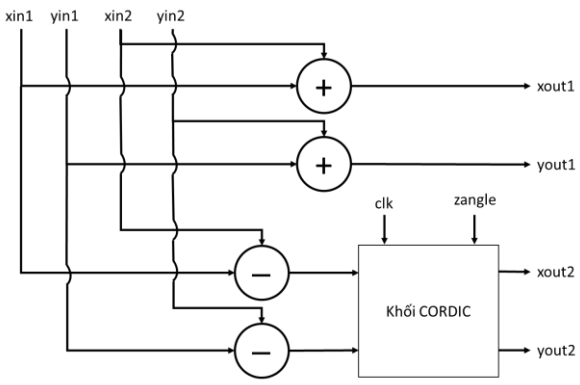
$$X_{m+1}[p] = X_m[p] + X_m[q] \quad (20)$$

$$X_{m+1}[q] = (X_m[p] - X_m[q])W_N^k \quad (21)$$

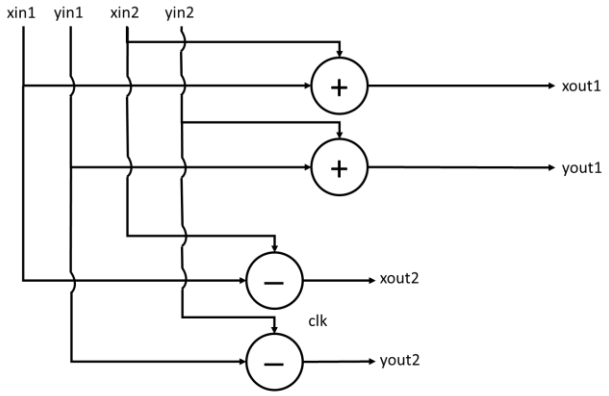
Như vậy trong khối Butterfly chỉ cần dùng 1 bộ CORDIC để tính toán đầu ra thứ hai. Sơ đồ khối Butterfly được thể hiện trên hình 5. Khối Butterfly có 2 đầu vào có phần thực và phần ảo lần lượt là xin1, yin1, xin2, yin2, 2 đầu ra có phần thực và phần ảo lần lượt là xout1, yout1, xout2, yout2 [4].



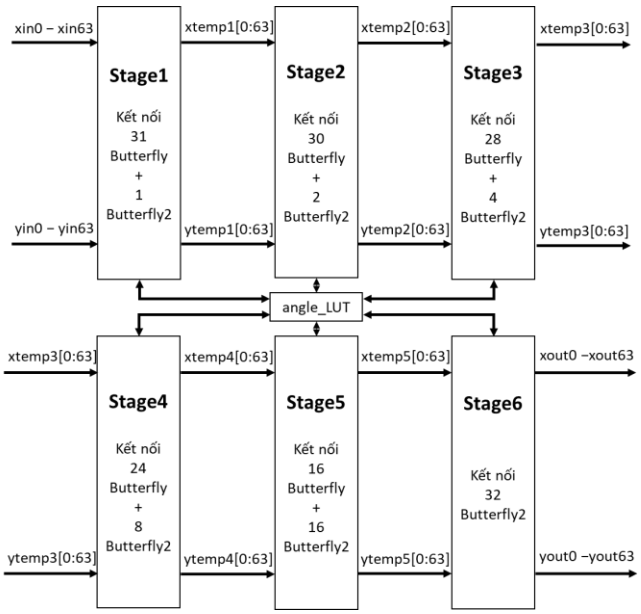
Hình 5. Sơ đồ cánh bướm cho DIF FFT cơ số 2



Hình 6. Sơ đồ khối Butterfly



Hình 7. Sơ đồ khối Butterfly2



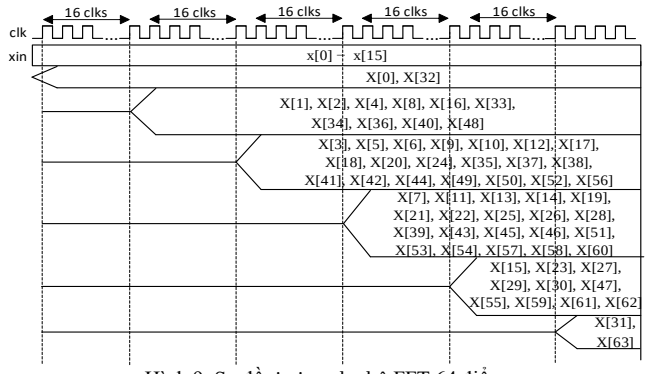
Hình 8. Sơ đồ khối FFT chính

3) Khối FFT chính

Trước khi xây dựng khối FFT chính, nhóm em xây dựng khối Butterfly2 dành để tính sơ đồ cánh bướm mà góc quay của CORDIC là 0. Khối này giống hệt khối Butterfly chỉ khác là không có bộ CORDIC ở đầu ra xout2, yout2 (hình 7), từ đó giảm được sai số do bộ CORDIC gây ra. Khối FFT 64 điểm gồm 6 stages, các stage từ 1 đến 6 có số Butterfly và Butterfly2 lần lượt là (31, 1), (30, 2), (28, 4), (24, 8), (16, 16) và (0, 32). Stage 6 (đầu ra) không sử dụng khối Butterfly nào vì chỉ toàn các DFT 2 điểm nên chỉ dùng khối Butterfly2. Sơ đồ khối FFT chính được thể hiện trên hình 8. 64 đầu vào với phần thực xin0−xin63 và phần ảo yin0−yin63 sẽ cho ra 16 đầu ra với phần thực xout0−xout63 và phần ảo yin0−yin63. Sau mỗi stage 1−5, 64 thanh ghi 16 bit xtemp1 (xtemp2,…, xtemp5) và 64 thanh ghi 16 bit ytemp1(ytemp2,…, ytemp5) dùng để lưu trữ giá trị trung gian tính toán được sau mỗi stage. Bảng Angle_LUT là bảng biểu diễn các góc quay trong FFT 64 điểm để đưa vào bộ CORDIC trong các khối Butterfly, gồm 31 góc $e^{-j\frac{\pi}{8}n}$, với n=1,…,31, tương ứng 31 góc −5.625°, −11.25,…, −174.375°.

Bảng 2. Số clock cycles cần cho các đầu ra

Đầu ra	Số clock cycle
X[0], X[32]	0
X[1], X[2], X[4], X[8], X[16], X[33], X[34], X[36], X[40], X[48]	16
X[3], X[5], X[6], X[9], X[10], X[12], X[17], X[18], X[20], X[24], X[35], X[37], X[38], X[41], X[42], X[44], X[49], X[50], X[52], X[56]	32
X[7], X[11], X[13], X[14], X[19], X[21], X[22], X[25], X[26], X[28], X[39], X[43], X[45], X[46], X[51], X[53], X[54], X[57], X[58], X[60]	48
X[15], X[23], X[27], X[29], X[30], X[47], X[55], X[59], X[61], X[62]	64
X[31], X[63]	80



Hình 9. Sơ đồ timing cho bộ FFT 64 điểm

C. *Timing cho bộ DIF FFT 64 điểm*

Trong khối CORDIC, mỗi vòng quay số phức sẽ tốn 1 clock cycles, vì có 16 vòng lặp nên một khối CORDIC sẽ tốn 16 clock cycle để thực hiện. Ta đã biết bộ FFT 64 điểm gồm 6 stages, các stage 1−5 gồm các khối Butterfly, mỗi khốiButterfly lại có 1 khối CORDIC nên để thực hiện một stage sẽ tốn 16 clock cycles, vì các khối Butterfly được thực hiện song song với nhau. Các đầu ra của bộ FFT sẽ tốn số lượng clock cycle khác nhau, tùy thuộc vào nó được tính toán qua bao nhiêu stage cần sử dụng bộ CORDIC. Dựa vào lưu đồ tín hiệu ở hình 2, bảng 2 thể hiện số clock cycle mà mỗi đầu ra cần để tính toán. Sơ đồ timing với các clock cycle lí tưởng được thể hiện trên hình 9.

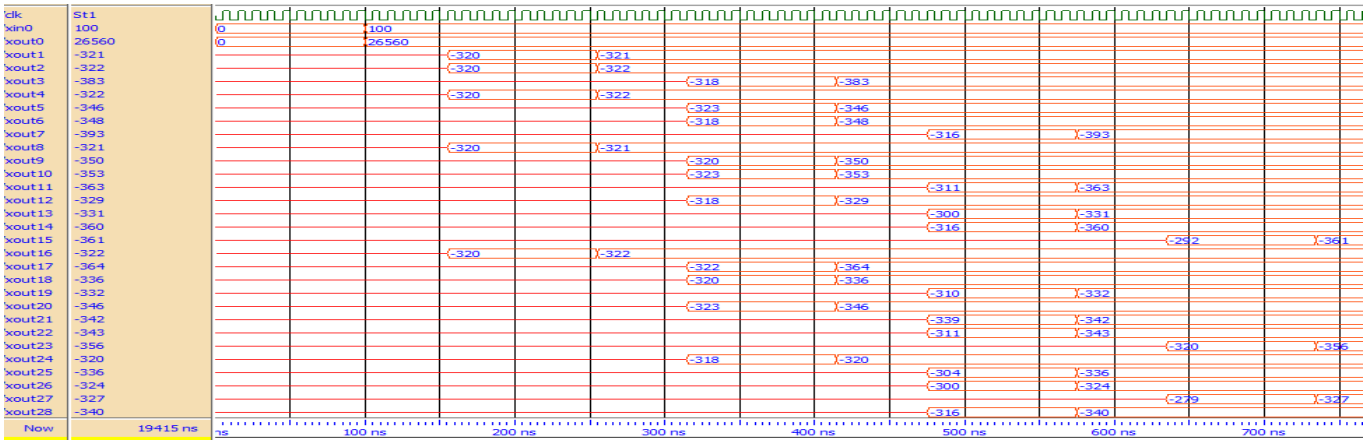
IV. KẾT QUẢ MÔ PHỎNG

A. *Kết quả mô phỏng trên Modelsim*

Nhóm em mô phỏng bộ DIF FFT 64 điểm trên Modelsim để so sánh timing với lí thuyết.. Hình 10 là kết quả mô phỏng trên Modelsim, với x là phần thực, y là phần ảo, đơn vị thời gian là ns. Mỗi clock cycle kéo dài 10ns, các đầu vào được đưa vào ở thời điểm 100ns. Sườn lên cung clock đầu tiên tính từ thời điểm 105ns, trên hình chỉ thể hiện được đầu vào x[0] và các đầu ra từ X[0] đến X[28]. Đầu ra X[0] có kết quả ngay thời điểm đưa các đầu vào vào bộ FFT 64 điểm, vì không cần bộ CORDIC nào (không cần sử dụng xung clock), các đầu ra còn lại nhóm em nhận thấy có kết quả tại thời điểm đúng như timing trong lí thuyết. Ngoài ra các đầu ra từ X[29] đến X[63] cũng đã được nhóm em kiểm tra và xác nhận là có timing đúng theo lí thuyết.

B. *So sánh kết quả mô phỏng với kết quả trên MATLAB*

Bảng 3 so sánh kết quả bộ DIF FFT 64 điểm trên mô phỏng và kết quả trên MATLAB trong một trường hợp các đầu vào từ x[0] đến x[31] toàn số thực, còn lại các đầu vào bằng 0. Các đầu ra có sai số khác nhau tùy số lượng bộ CORDIC cần sử dụng để tính toán. Nhóm em nhận thấy là X[0] và X[32] chính xác tuyệt đối vì không cần dùng bộ CORDIC nào (nếu dựa theo công thức DFT (1) thì X[0] chính là tổng các đầu vào); X[63] có sai số khá cao vì cần nhiều bộ CORDIC nhất. Tính toán cho thấy sai số trong trường hợp này là 2.69%. Vì đầu vào là các số có giá trị nhỏ hơn 1000, tức là chỉ sử dụng tối đa 10/16 bit nên có sai số thấy được, do vòng lặp CORDIC lặp đến 16 lần, nên nếu thay đổi đầu vào thành các số có giá trị vài nghìn (sử dụng nhiều bit hơn) thì chắc chắn sai số sẽ giảm. Tuy nhiên một hạn chế của bộ FFT này là chỉ sử dụng 16 bit biểu diễn số có dấu, nên khoảng biểu diễn là $[-2^{16} \quad 2^{16}-1]$, nếu đầu vào có giá trị quá lớn có thể xảy ra tràn số ở đầu ra dẫn đến kết quả không chính xác. Việc này có thể khắc phục bằng cách tăng số bit biểu diễn, tuy nhiên sẽ làm tăng độ phức tạp của khối CORDIC, đồng nghĩa với việc cần sử dụng nhiều phần cứng hơn.



Hình 10. Kết quả mô phỏng trên Modelsim

Bảng 3. So sánh KQ mô phỏng và KQ MATLAB

TT	Đầu vào	KQ mô phỏng	KQ MATLAB
0	1000	3200	3200
1	−800	1000−1846i	1000−1991i
2	1000	0	0
3	−800	935−401i	1000−541i
4	1000	0	0
5	−800	975−117i	1000−174i
6	1000	0	0
7	−800	943+99i	1000+43i
8	1000	0	0
9	−800	966+227i	1000+214i
10	1000	0	0
11	−800	956+397i	1000+373i
12	1000	0	0
13	−800	998+580	1000+533i
14	1000	0	0
15	−800	973+738i	1000+705i
16	1000	0	0
17	−800	952+891i	1000+902i
18	1000	0	0
19	−800	985+1161i	1000+1139i
20	1000	0	0
21	−800	961+1464i	1000+1442i
22	1000	0	0
23	−800	958+1853i	1000+1856i
24	1000	0	0
25	−800	992+2492i	1000+2480i
26	1000	0	0
27	−800	995+3594i	1000+3568i
28	1000	0	0
29	−800	1026+6128i	1000+6052i
30	1000	0	0
31	−800	992+18371i	1000+18315i
32	0	28800	28800
33	0	1000−18316i	1000−18315i
34	0	0	0
35	0	1009−6041i	1000−6052i
36	0	0	0
37	0	957−3609i	1000−3568i
38	0	0	0
39	0	991−2481i	1000−2480i
40	0	0	0
41	0	1002−1855i	1000−1856i
42	0	0	0
43	0	1002−1449i	1000−1442i
44	0	0	0
45	0	1000−1162i	1000−1139i
46	0	0	0
47	0	1007−928i	1000−902i
48	0	0	0
49	0	1048−701i	1000−705i
50	0	0	0
51	0	1035−579i	1000−533i
52	0	0	0
53	0	995−422i	1000−373i
54	0	0	0
55	0	1032−251i	1000−214i
56	0	0	0
57	0	1048−84i	1000−43i
58	0	0	0
59	0	1067+94i	1000+174i
60	0	0	0
61	0	1064+426i	1000+541i
62	0	0	0
63	0	1136+1727i	1000+1991i

V. KẾT LUẬN

Như vậy nhóm em đã hoàn thành thực hiện FFT 64 điểm bằng ngôn ngữ Verilog HDL, kết quả được so sánh với kết quả trên MATLAB để kiểm tra độ chinh xác. Với các giá trị đầu vào các phức tạp hoặc có giá trị càng nhỏ thì sai số đầu ra càng lớn, các sai số này là không thể tránh khỏi khi thực hiện trên phần cứng. Các sai số có thể giảm nếu biểu diễn dữ liệu theo kiểu dấu phảy động và sử dụng các bộ nhân, tuy nhiên sẽ cần dùng nhiều tài nguyên phần cứng hơn. Với việc sử dụng CORDIC để tiết kiệm phần cứng thì sai số 2.69% như ở phần IV đã trình bày là chấp nhận được.

TÀI LIỆU THAM KHẢO

[1] Dang Quang Hieu, “Digital Signal Processing Lecture”, 2015

[2] Abhishek Kesh, Chintan S.Thakkar, Rachit Gupta, Siddharth S. Seth, T. Anish, “Implementation of Fast Fourier Transform (FFT) on FPGA using Verilog HDL”, Indian Institute of Technology Kharagpur, 2004

[3] Ray Andraka, “A survey of CORDIC algorithms for FPGA based computers”, Proceedings of the 1998 ACM/SIGDA sixth International Symposium on Field Programmable Gate Array

[4] Joshua Peter Ebenezer, Kaustav Brahma, “Implementation Of The Fast Fourier Transform Using Cordic”, Indian Institute of Technology Kharagpur, 2018

[5] Pong P.Chu, “FPGA Prototyping By Verilog Examples”, John Wiley & Sons Inc Publication, 2008

[6] https://en.wikipedia.org/wiki/Cooley_Tukey_FFT_algorithm, truy cập lần cuối ngày 24/5/2019

[7] https://en.wikipedia.org/wiki/CORDIC, truy cập lần cuối ngày 24/5/2019

[8] http://dspguru.com/dsp/faqs/cordic/, truy cập lần cuối ngày 24/5/2019

[9] https://en.wikipedia.org/wiki/Fast_Fourier_transform, truy cập lần cuối ngày 24/5/2019