Hiện thực bộ nhân số phức dấu chấm động cho tính toán FFT dựa trên thuật toán CORDIC xoay góc thích nghi

- **Võ Thị Phương Thảo**
- **Trương Thị Như Quỳnh**
- **Hoàng Trọng Thức**
- **Lê Đức Hùng**

Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM

(Bài nhận ngày 26 tháng 12 năm 2016, nhận đăng ngày 30 tháng 10 năm 2017)

TÓM TẮT

Trong bài báo này, bộ nhân số phức chuyển đổi nhanh Fourier dấu chấm động độ chính xác đơn được đề xuất. Kiến trúc bộ nhân được thiết kế dựa trên thuật toán CORDIC góc xoay thích nghi. Chip FPGA Stratix IV của Altera và tổng hợp SOTB công nghệ 65 nm được sử dụng để xây dựng và kiểm tra thiết kế. Trên chip FPGA, tần số hoạt động của bộ nhân là 103,9 MHz, tốc độ thực thi hệ thống là 16.966 triệu mẫu trên giây (Mega-Sample per second – MSps) và tài nguyên được sử dụng là 7,747 ALUTs và 625 thanh ghi. Mặt khác, tổng hợp ASIC sử dụng 16,858 standard cells trên 86,718µm^2 diện tích chip, đạt được tần số là 166 MHz và tốc độ hệ thống là 27,107 MSps. Độ chính xác của kết quả đạt được với sai số bình phương tối thiểu (Mean-Square-Error – MSE) là 1.133E-10 và khoảng 26 phần mỗi triệu (part-per-million – ppm) tỉ lệ lỗi tối đa.

Từ khóa: *bộ nhân số phức dấu chấm động, FFT, CORDIC xoay góc thích nghi*

MỞ ĐẦU

Biến đổi Fourier nhanh (Fast Fourier Transform–FFT) là thuật toán biến đổi nhanh của biến đổi Fourier rời rạc (Discrete Fourier Transform–DFT). Năm 1965, đồ thị tín hiệu (Signal Flow Graph–SFG) đầu tiên của FFT được phổ biến rộng rãi bởi Cooley và Tukey [1]. Từ đó, FFT được sử dụng hầu hết trong các ứng dụng khác nhau của xử lý tín hiệu số và truyền thông. Thuật toán FFT và biến đổi ngược của nó (Inverse Fast Fourier Transform – IFFT) là thành phần cốt lõi quan trọng trong các hệ thống truyền thông hiện đại, đặc biệt là trong mạng không dây và ứng dụng đa phương tiện như là WiMAX [2], 3GPP-LTE [3], MIMO [4] và CDMA [5]. Hơn nữa, FFT còn là đơn vị xử lý trung tâm trong thiết kế của hệ thống chia tần số trực giao (Orthogonal Frequency Division Multiplexing – OFDM) [6].

Điểm mạnh của kiến trúc FFT dấu chấm tĩnh là tốc độ nhanh và tài nguyên tối ưu. Tuy nhiên, việc bỏ đi các bit thấp (Least Significant Bits – LSBs) dẫn đến lỗi toán học trong dấu chấm tĩnh (Fixed-point Arithmetic Error – FAE) [7], đó là lỗi giữa hệ thống dấu chấm tĩnh và hệ thống dấu chấm động. Trong một số hệ thống không cần tính toán có độ chính xác cao thì FAE có thể không quan trọng. Thí dụ, có thể triển khai mạch FFT dấu chấm tĩnh trong các hệ thống âm thanh, các phương pháp nén lossy, các hệ thống truyền thông như DVB-T/DVB-H [8] và WLAN [9]. Tuy nhiên, ngày càng nhiều ứng dụng FFT không những yêu cầu độ chính xác cao mà còn cần miền dữ liệu hoạt động lớn. Vì vậy, thiết kế FFT dấu chấm tĩnh gặp khó khăn trong việc khắc phục những vấn đề đó. Vì thế, ngày nay các thiết kế FFT dấu chấm động có xu hướng phát triển phổ biến hơn, thí dụ, trong hệ thống ra-đa như ra-đa

khẩu độ mặt đất (Synthetic Aperture Radar – SAR) [10] và xử lý ảnh trong y khoa như hệ thống chụp cắt lớp quang Fourier (Fourier – Domain Optical Coherence Tomography – FD-OCT) [11]. Bên cạnh đó, dấu chấm động đã trở thành một yêu cầu không thể thiếu cho việc tính toán FFT số điểm lớn [12].

Thiết kế FFT dấu chấm động có thể được chia thành hai kiến trúc chính: FFT phân luồng liên tục (Continuous Flow FFT – CF-FFT) [6], [13] và bộ vi xử lý dựa trên bộ nhớ (Memory-Based Processor – Mem-FFT) [14, 15]. Các thiết kế CF-FFT có ưu thế về tốc độ nên được triển khai phổ biến hơn Mem-FFT. Tuy nhiên, chúng có nhược điểm là tốn nhiều tài nguyên, đặc biệt khi triển khai mạch CF-FFT với số điểm lớn dẫn đến khó thực hiện trong thiết kế VLSI. Ngược lại, các mẫu thiết kế Mem-FFT chủ yếu dựa trên việc sử dụng bộ nhớ nên cần các nguồn tài nguyên logic. Do đó, các thiết kế Mem-FFT rất có lợi thế trong thiết kế VLSI và rất dễ tương thích với các hệ thống có tích hợp sẵn CPU. Tuy nhiên, nhược điểm của Mem-FFT chính là độ trễ quá lớn dẫn đến tốc độ xử lý của hệ thống chậm. Một hệ thống Mem-FFT điển hình gồm 1 bộ cộng trừ và nhân chéo (Butterfly – BF) đơn vị, 1 đơn vị tạo địa chỉ, 1 bộ điều khiển đơn vị và 1 vài khối bộ nhớ. Thiết kế CF-FFT điển hình N điểm có $\log_2(N)$ tầng pipeline với mỗi tầng gồm có 1 BF đơn vị và 1 số thanh ghi dịch. Mặc dù kiến trúc là khác nhau nhưng hạn chế về tốc độ của các thiết kế BF đều ở bộ nhân (Twiddle Factor – TF). TF cũng là nhân tố chính dẫn đến sự chính xác của các hệ thống FFT.

Kiến trúc TF có thể được chia thành 2 thiết kế chính: thiết kế bảng tra (Look Up Table – LUT) và tính toán trực tiếp. Trong thiết kế TF dựa trên bảng tra LUT, các hằng số lượng giác đã tính toán trước và được lưu trữ trong bộ nhớ chỉ đọc (Read Only Memory – ROM), chúng được truy xuất ra để sử dụng khi thích hợp. Một vài thí dụ về bảng tra LUT được thực hiện trong [16], [17] và [18]. Ưu thế lớn nhất của kiến trúc LUT là tốc độ cao và độ trễ thấp trong khi yêu cầu về nguồn tài nguyên là tương đối nhỏ. Tuy nhiên, trong việc triển khai FFT số điểm lớn thì thiết kế LUT cần một ROM có dung lượng quá lớn dẫn đến việc thiết kế mạch tích hợp lớn (Very-Large-Scale Integration – VLSI) sẽ trở nên khó khăn hơn. Để khắc phục những vấn đề trên, nhiều công trình nghiên cứu đã được đề xuất. Thí dụ, H. Y. Lee và I. C. Park [19] đã đề xuất một thuật toán phân hủy cây nhị phân để giảm thiểu số lượng bộ nhân được lưu trữ trong ROM. R. Radhouane cùng các cộng sự [13] đưa ra một phương pháp để tối ưu bộ nhớ cho CF-FFT. Một 'bộ nhớ truy cập tối ưu hóa' được mô tả bởi X. Xiao cùng các cộng sự [20] để đọc dữ liệu và bộ nhân sử dụng kiến trúc bộ nhớ 4 bank. S. Mou và X. Yang [17] đã phân tích và tối ưu độ trễ đọc sau khi viết (Read-After-Write – RAW) trong hệ thống vi xử lý FFT. Tuy nhiên, các kĩ thuật này đã không đưa ra một giải pháp cuối cùng mà chỉ đánh đổi giữa dung lượng ROM và độ chính xác hoặc tốc độ thực thi. Vì những khó khăn đó, kiến trúc LUT chỉ phù hợp với các ứng dụng FFT ít điểm. Còn trong các ứng dụng cần tính FFT nhiều điểm thì giải pháp là tính toán bộ nhân trực tiếp.

Trong phương pháp tính toán bộ nhân trực tiếp, thuật toán xoay tọa độ véc-tơ trong hệ thống tính toán số (Coordinate Rotation DIgital Computer – CORDIC) đã sớm được đề xuất. Cho đến nay, bộ vi xử lý FFT dựa trên CORDIC vẫn được nghiên cứu và phát triển [14, 15]. Thuật toán CORDIC truyền thống có tác dụng lớn trong việc thực thi bộ nhân dấu chấm tĩnh vì nó làm giảm độ phức tạp của phép nhân thành phép dịch và phép cộng. Tuy nhiên, phương pháp này không thích hợp cho thiết kế bộ nhân dấu chấm động vì phép cộng của dấu chấm động không hề đơn giản. Mạch nhân dựa trên CORDIC truyền thống trong FFT dấu chấm động tốn nhiều tài nguyên và tốc độ thấp. Để khắc phục tình trạng trên, phương pháp CORDIC mới có độ chính xác

tương đương nhưng có độ hội tụ nhanh hơn được đề xuất. Hạn chế chính của phương pháp CORDIC truyền thống là số bước lặp. Có nhiều phương pháp được đề xuất với mục đích giảm số vòng lặp mà vẫn giữ được độ chính xác. Y. H. Hu cùng các cộng sự [21] đã đề xuất một phương pháp hiệu quả gọi là 'Angle Recoding CORDIC' (ARC). Thuật toán ARC cắt giảm tối thiểu 50 % tổng số vòng lặp. Ý tưởng của ARC là chỉ chọn một vài góc để xoay thay vì xoay tất cả các góc. Vì thế hệ thống có thể thực hiện nhanh hơn và thậm chí có độ chính xác cao hơn. Nhiều ứng dụng triển khai thuật toán ARC đã được trình bày trong [22, 23]. Hơn nữa, bài viết của Hong-Thu Nguyen cùng các cộng sự đã đề xuất một phương pháp thích nghi của ARC (Adaptive method of ARC – AARC) [24] dựa trên dữ liệu dấu chấm động có độ chính xác đơn. Thuật toán AARC đã được chứng minh thực nghiệm là ít tốn tài nguyên, độ trễ thấp và chính xác cao [24]. Dựa trên thuật toán AARC, thiết kế bộ nhân ROM-free đã được đề xuất trong bài báo này. Bộ nhân dựa trên AARC được xây dựng và chứng minh trên chip FPGA Stratix IV của Altera và tổng hợp SOTB công nghệ 65 nm. Kết quả thực nghiệm cho thấy kiến trúc đề xuất có tần số chạy trên chip FPGA là 103,9 MHz và phân tích trên ASIC là 166 MHz. Kết quả về diện tích chip là 7.747 ALUTs và 625 thanh ghi trên chip FPGA và 16,858 standard cells trên 86,718 µm^2 diện tích chip SOTB. Độ chính xác của mạch được kiểm tra bằng cách triển khai thuật toán sai số bình phương tối thiểu và phần mỗi triệu tỉ lệ lỗi tối đa. Kết quả độ chính xác là 1.133E−10 MSE và khoảng 26 phần mỗi triệu tỉ lệ lỗi tối đa. Tốc độ thực thi tương ứng trên chip FPGA và tổng hợp ASIC là 16,966 MSps và 27,107 MSps.

THUẬT TOÁN CORDIC XOAY GÓC THÍCH NGHI (AARC)

Tính toán CORDIC dựa trên các vòng lặp của ba tham số *X, Y, Z* như trong công thức lặp 1.

$$x_{i+1} = x_i - sign(z_i) y_i 2^{-i}$$

$$y_{i+1} = y_i + sign(z_i) x_i 2^{-i} \quad (1)$$

$$z_{i+1} = z_i - sign(z_i)\alpha_i$$

Trong đó, x_{i+1} và y_{i+1} là giá trị tọa độ mới của véc-tơ khi véc-tơ xoay quanh góc mới z_{i+1}. Giá trị α_i trong phương trình được gọi là góc dư và có thể được tính theo công thức 2.

$$\alpha_i = tan^{-1}(2^{-i}) \quad (2)$$

Sau một vài bước lặp, kết quả cuối cùng sẽ bị tăng lên thêm một hằng số *K* gọi là hệ số chiều dài. Vì vậy, ngõ ra *X* và *Y* cần phải loại bỏ hệ số *K* để cho ra kết quả đúng. Trong thuật toán CORDIC truyền thống, góc dư là hằng số và góc z_i sẽ giảm gần một nửa giá trị sau mỗi lần lặp. Mặt khác, góc dư trong AARC không cố định và chúng được chọn dựa trên trạng thái trước đó của z_i. Như vậy, thuật toán AARC có thể cắt giảm số lần lặp trong khi kết quả ngõ ra vẫn có độ chính xác tương đương.

Hình 1 cho thấy mã giả của phương pháp AARC. Góc θ_i được chọn trong mỗi lần lặp dựa vào góc dư z_i sẽ nhanh chóng hội tụ về 0. Trong mỗi bước lặp, thuật toán sử dụng khái niệm *C*, là tích của các tham số c_i. Mỗi giá trị c_i thể hiện phạm vi của các góc còn lại xung quanh một góc không đổi như có thể được nhìn thấy trong các công thức 3.

$$c_i = \begin{cases} \frac{\theta_i + \theta_{i+1}}{2} & , 0 \le i \le (N-2) \\ \frac{\theta_i}{2} & , i < 0 \text{ và } i > (N-2) \end{cases} \quad (3)$$

$i = j = 0; x_0 = 1; y_0 = 0; K = 1;$
while $|z_j| > c_{15}$ and $i < N$ **do**
  if $|z_j| \in (c_{i+1}; c_i]$ **then**
    $z_{j+1} = z_j - sign(z_j)\theta_i;$
    $x_{j+1} = x_j - sign(z_j) y_j 2^{-i};$
    $y_{j+1} = y_j + sign(z_j) x_j 2^{-i};$
    $K = K \times k_i; j = j + 1;$
  end if
  $i = i + 1$
end while
$X = x_j \times K; Y = y_j \times K;$

Hình 1. Mã giả của kiến trúc AARC

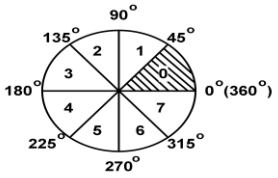
Do góc dư z_i luôn thay đổi nên K cũng sẽ thay đổi không phải là hằng số như trong phương

pháp truyền thống. Hệ số này là tích của các k_i như công thức 4. Giá trị k_i được tính theo công thức 5.

$$K = \prod_{i\in\theta} k_i \quad (4)$$

$$k_i = \cos(\theta_i) \quad (5)$$

Lưu ý rằng mã giả chỉ tính cho góc từ 0^0 đến 45^0. Những góc khác trong vòng tròn lượng giác phải được chuẩn hóa về khoảng giá trị trên như Hình 2.

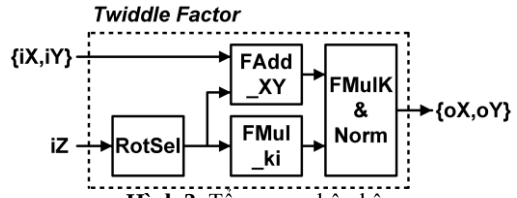


Segment	Correction	Segment	Correction
0	$\sin\theta_0$ $\cos\theta_0$	4	$\sin\theta_4 = -\sin\theta_0$ $\cos\theta_4 = -\cos\theta_0$
1	$\sin\theta_1 = \cos\theta_0$ $\cos\theta_1 = \sin\theta_0$	5	$\sin\theta_5 = -\cos\theta_0$ $\cos\theta_5 = -\sin\theta_0$
2	$\sin\theta_2 = \cos\theta_0$ $\cos\theta_2 = -\sin\theta_0$	6	$\sin\theta_6 = -\cos\theta_0$ $\cos\theta_6 = \sin\theta_0$
3	$\sin\theta_3 = \sin\theta_0$ $\cos\theta_3 = -\cos\theta_0$	7	$\sin\theta_7 = -\sin\theta_0$ $\cos\theta_7 = \cos\theta_0$

Hình 2. Các góc chuẩn hóa trong vòng tròn lượng giác

HIỆN THỰC BỘ NHÂN

Tổng quan kiến trúc bộ nhân



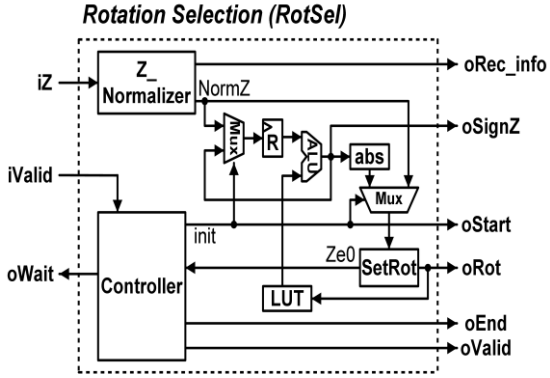
Hình 3. Tổng quan bộ nhân

Tổng quan bộ nhân TF được thể hiện như Hình 3. Kiến trúc TF gồm 4 mô-đun: Mô-đun chọn góc xoay θ_i (*RotSel*), Mô-đun tính tổng của *X* và *Y* (*FAdd_XY*), Mô-đun tính tích k_i (*FMul_k*$_i$) và Mô-đun tính hệ số *K* và chuẩn hóa ngõ ra (*FMulK_andNorm*). Mô-đun *RotSel* nhận giá trị góc *iZ* từ bên ngoài để tính ra các góc dư θ cần xoay. Tất cả các góc được chuyển vào Mô-đun *FAdd_XY* và *FMul_k*$_i$ theo từng clock. Sau đó, nhờ quá trình lặp, Mô-đun *FAdd_XY* và *FMul_k*$_i$ tính ra giá trị *X, Y* và *K* tương ứng. Mô-đun *FAdd_XY* nhận giá trị khởi tạo ban đầu *X, Y* tương ứng *iX, iY* từ bên ngoài. Sau đó, giá trị của *X* và *Y* được tính bởi Mô-đun *Add_XY* tại clock có góc dư θ_i truyền vào. Cùng lúc đó, hệ số chiều dài *K* được nhân với k_i bởi Mô-đun *FMul_k*$_i$ mỗi khi có góc θ_i. Quá trình lặp kết thúc khi Mô-đun RotSel hoàn thành việc tính và chuyển toàn bộ góc θ ra ngoài mô-đun, lúc này hai Mô-đun *FAdd_XY* và *FMul_k*$_i$ cũng hoàn thành quá trình lặp của mình. Đó cũng là lúc Mô-đun *FMulK_andNorm* hoạt động. Mô-đun *FMulK_andNorm* tiến hành nhân hai giá trị *X* và *Y* và giá trị *K* đến từ *FMul_k*$_i$ để cho ra kết quả cuối cùng. Giá trị cuối cùng của *X* và *Y* phải được chuẩn hóa theo định dạng của IEEE-754 trước khi truyền tương ứng ra *oX* và *oY* ở ngõ ra cuối cùng. Như đã đề cập ở trên, thuật toán AARC phải quy đổi tất cả góc ngõ vào trong vòng tròn lượng giác về đoạn 0^0 đến 45^0 trước quá trình tính toán. Mô-đun *RotSel* sẽ đảm nhiệm việc chuyển đổi này. Tuy nhiên, tín hiệu ngõ vào và ngõ ra phải được hoán đổi cùng với việc chuyển đổi góc. Về đặc điểm kĩ thuật, dữ liệu ngõ vào hoặc ngõ ra phải được hoán đổi với nhau và đổi dấu các tín hiệu *X* và *Y* ở ngõ ra để có kết quả chính xác cuối cùng. Việc chuyển đổi các giá trị ngõ vào và ngõ ra được thực hiện dựa trên Bảng 1.

Bảng 1. Bảng tra phục hồi dữ liệu dựa vào thông tin góc Z

Phân đoạn	Chuyển đổi góc	Chuyển đổi dữ liệu X-Y		Đảo tín hiệu ngõ ra	
		Ngõ vào	Ngõ ra	X	Y
0	Z	Không	không	Không	Không
1	$\pi/2 - Z$	Có	Không	Có	Không
2	$Z - \pi/2$	Không	Có	Có	Không
3	$\pi - Z$	Có	Có	Có	Có
4	$Z - \pi$	Không	Không	Có	Có
5	$3\pi/2 - Z$	Có	không	Không	Có
6	$Z - 3\pi/2$	Không	Có	Không	Có
7	$2\pi - Z$	Có	Có	Không	Không

Mô-đun chọn góc xoay θ_i (*RotSel*)

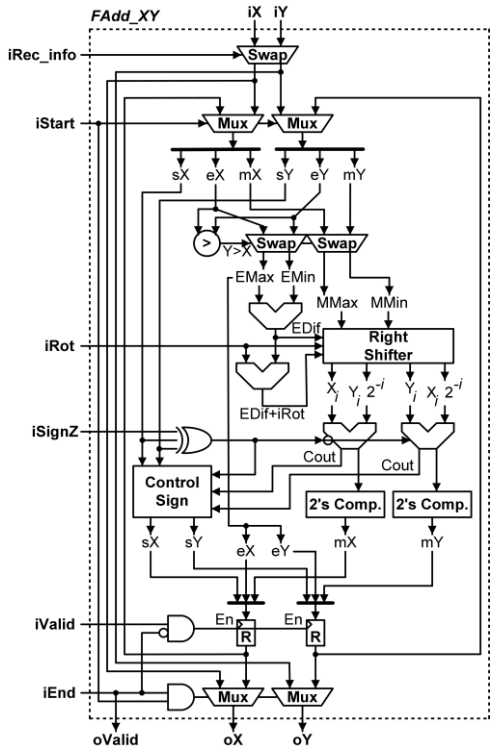


Hình 4. Sơ đồ khối của Mô-đun RotSel

Hình 4 mô tả sơ đồ khối của Mô-đun *RotSel*. Trong đó, Mô-đun *Z_Normalizer* chuyển đổi góc ngõ vào *iZ* về dạng góc chuẩn hóa *NormZ* có giá trị trong đoạn từ 0^0 đến 45^0. Tín hiệu *oRec_info* có 3-bit sẽ chứa thông tin miền giá trị mà *iZ* phụ thuộc vào. Tín hiệu *oRec_info* sẽ giúp phục hồi lại dữ liệu sau này dựa vào Bảng 1. Lúc bắt đầu quá trình lặp, mạch đa hợp (multiplexer – *Mux*) sẽ đưa giá trị *NormZ* cất vào thanh ghi. Còn trong quá trình lặp, kết quả của bộ cộng trừ (*ALU*) sẽ được *Mux* chọn để cất vào thanh ghi. Trong ALU, giá trị của góc *Z* tiếp theo được tính bằng cách cộng hoặc trừ giá trị *Z* hiện tại với góc dư θ hiện tại được chọn ra từ bảng tra. Mà góc θ hiện tại thì được tính bởi lần xoay trước thông qua LUT như có thể thấy trong Hình 4. Kết quả *ALU* được lặp lại và lưu trữ trong thanh ghi trong mỗi lần lặp và dấu của nó sẽ được đưa thẳng ra ngõ ra bằng tín hiệu *oSignZ*. Mô-đun *SetRot* là Mô-đun ra quyết định lựa chọn góc cho lần lặp tiếp theo. Dựa trên giá trị của góc *Z* hiện tại mà nó nhận được, Mô-đun *SetRot* sẽ chọn góc tiếp theo để xoay. Khi bắt đầu quá trình lặp, Mô-đun *SetRot* nhận giá trị góc *Z* trong Mô-đun *Z_Normalizer* thông qua tín hiệu *NormZ*. Ngoài ra thì nó nhận giá trị tuyệt đối của góc *Z* từ *ALU* khi đang trong quá trình lặp. Toàn bộ quá trình lặp sẽ kết thúc khi tín hiệu cho biết *Z* bằng không (*Ze0*) được bật lên. Và cuối cùng, Mô-đun *Controller* điều khiển hoạt động lặp của *RotSel* và quản lý các tín hiệu điều khiển khác như là *oValid*, *oStart*, *oEnd* và *oWait*. Giả sử rằng góc *Z* cần xoay nhiều lần để tính toán. Khi đó, tín hiệu *oWait* sẽ được Mô-đun *RotSel* bật lên để yêu cầu bên ngoài chờ, còn hai tín hiệu ngõ vào là *iValid* và *iZ* sẽ được giữ nguyên trạng thái. Tín hiệu *oValid* bật lên 1 để đánh dấu một phiên làm việc của bộ nhân TF. Tín hiệu *oStart* và *oEnd* bật lên trong một clock sẽ lần lượt đánh dấu sự bắt đầu và kết thúc của phiên tính toán. Tín hiệu *oWait* ở mức 0 khi Mô-đun kết thúc quá trình chuyển đổi góc xoay. Sau đó, trong chu kỳ kế tiếp, tín hiệu ngõ vào *iValid* và *iZ* được tắt khi chúng thấy tín hiệu *oWait* tắt và phiên tính *Z* hoàn tất. Có một trường hợp đặc biệt ngoại lệ là khi góc ngõ vào bằng 0. Trong trường hợp này, giá trị *oX* và *oY* ở ngõ ra bằng giá trị *X* và *Y* ở ngõ vào. Vì thế, phiên tính toán khi *Z* bằng 0 chỉ kéo dài trong một clock.

Mô-đun tính tổng của *X* và *Y* (*FAdd_XY*)

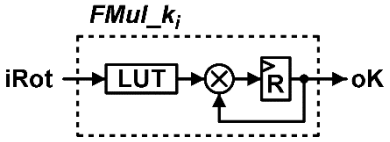
Mô-đun FAdd_XY cộng tích lũy hai số dấu chấm động X và Y . Số dấu chấm động 32-bit theo chuẩn IEEE-754 bao gồm 3 thành phần: thành phần dấu (signed) 1-bit, thành phần số mũ (exponent) 8-bit và thành phần giá trị (mantissa) 23-bit. Có 3 bước cơ bản để thực thi bộ cộng hai số dấu chấm động: cân bằng số mũ, cộng hoặc trừ hai phần mantissa, chuẩn hóa kết quả ngõ ra. Đầu tiên, số mũ của cả hai số hạng phải được so sánh với nhau. Số có số mũ nhỏ hơn được dịch phải để có số mũ giống với số mũ của số còn lại. Từ đó dẫn đến hai phần mantissa của hai số hạng có thể được cộng trực tiếp với nhau ở bước thứ hai. Trong bước cuối cùng, kết quả của bộ cộng được chuẩn hóa về định dạng dấu chấm động IEEE-754. Trong kiến trúc này, Mô-đun FAdd_XY sẽ bỏ qua ước chuẩn hóa cuối cùng. Lý do là vì Mô-đun FAdd_XY cần tính cộng tích lũy dữ liệu. Do đó, việc chuẩn hóa và giải chuẩn hóa dữ liệu liên tục là không hợp lý. Vì thế, phần chuẩn hóa ngõ ra được bỏ qua trong Mô-đun FAdd_XY để đơn giản hóa cho việc thiết kế.



Hình 5. Sơ đồ khối của Mô-đun FAdd_XY

Sơ đồ khối của Mô-đun *FAdd_XY* được thể hiện trong Hình 5. Thông tin *iRec_info* cho biết góc hiện tại đang ở góc phần tám nào trong vòng tròn lượng giác như Hình 2. Giá trị vào *iX* và *iY* có thể được hoán đổi dựa vào giá trị *iRec_info* theo Bảng 1. Sau đó, *Mux* chọn dữ liệu từ ngõ vào hoặc dữ liệu tương ứng từ thanh ghi. Dữ liệu được chọn được chia thành 3 phần độc lập: phần dấu (*sX* và *sY*), phần exponent (*eX* và *eY*) và phần mantissa (*mX* và *mY*) như đã mô tả trong hình. Phần exponent được so sánh với nhau để chọn ra số có exponent lớn hơn. *EMax* và *MMax* lần lượt là giá trị exponent và manissa của số có số mũ lớn hơn. Tương tự, *EMin* và *MMin* tương ứng là exponent và mantissa của số có số mũ nhỏ hơn. Giá trị *Edif* là hiệu của *EMax* và *EMin*. Chức năng của Mô-đun *Right_Shifter* là dịch phải phần mantissa để cân bằng giá trị số mũ của hai số. Mô-đun *Right_Shifter* cần thông tin của *Edif*, *iRot* và *Edif +iRot* cùng với hai mantissa *MMax* và *MMin* để tính các giá trị của X_i, Y_i2^{-i}, Y_i và X_i2^{-i} như trong Hình 5. Theo công thức 1 thì cặp X_i, Y_i2^{-i} cũng như cặp Y_i, X_i2^{-i} sẽ được cộng hoặc trừ với nhau. Việc cộng hay trừ sẽ phụ thuộc vào các phần dấu *sX* và *sY* , cùng với dấu của *Z* đến từ ngõ vào thông qua tín hiệu *iSignZ*. Kết quả các phép tính sẽ được đi qua các Mô-đun *2's complement* để lấy trị tuyệt đối. Các cờ tràn *Cout* của các *ALU* cùng với các phần dấu được đưa đến Mô-đun *Control Sign* để sinh ra các dấu của kết quả. Phần exponent của kết quả chính là giá trị *EMax*. Hai kết quả cuối cùng của X và Y được lưu trữ trong thanh ghi cho lần lặp tiếp theo hoặc đưa ra bên ngoài thông qua hai tín hiệu oX và oY. Ngoài ra, đối với trường hợp đặc biệt, khi góc vào bằng 0 được biết khi tín hiệu *iStart* và *iEnd* bật lên đồng thời trong một clock. Khi đó, *oX* và *oY* sẽ được lấy trực tiếp dữ liệu từ tín hiệu ngõ vào thay vì lấy từ hai thanh ghi kết quả.

Mô-đun tính tích k_i (*FMul_k$_i$*)

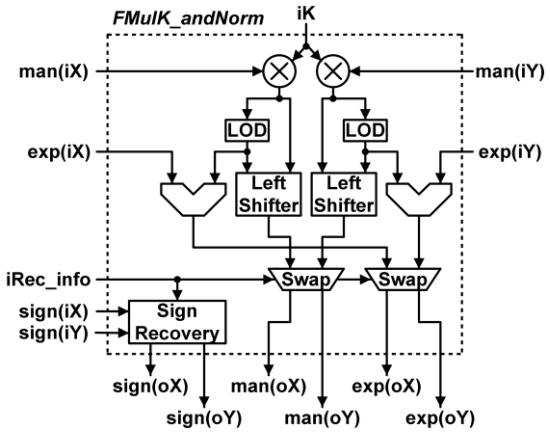


Hình 6. Sơ đồ khối của Mô-đun FMul_k$_i$

Kết quả ngõ ra của Mô-đun *RotSel*, *iRot* được truyền vào Mô-đun *FMul_k$_i$* để tính giá trị *K*. Mô-đun *FMul_k$_i$* thực hiện việc nhân tích lũy giá trị *K* theo công thức 4 và nó được thực thi song song với Mô-đun *FAdd_XY*. Hình 6 mô tả sơ đồ khối của Mô-đun *FMul_k$_i$*. Khi nhận 4-bit *iRot*, giá trị k$_i$ mới được chọn từ bảng tra LUT để đưa vào bộ nhân. Khi đó, giá trị k_i mới sẽ được nhân với giá trị *K* hiện tại để tạo ra hệ số *K* mới. Hệ số *K* này được lưu trữ bằng thanh ghi và được truyền ra ngõ ra bằng tín hiệu *oK* như có thể thấy trong Hình 6. Khi bắt đầu, thanh ghi lưu hệ số *K* được khởi tạo với giá trị là 1. Bởi vì các giá trị k_i đều là số dương nên sẽ sử dụng thiết kế bộ nhân không dấu tốc độ cao kế thừa từ công trình trước đó [25]. Hệ số *K* có giá trị từ 0,60725 đến 1. Hệ

số này đạt được giá trị nhỏ nhất là 0,60725 khi tất cả 16 giá trị nhân với nhau. Và nó bằng 1 khi trường hợp đặc biệt *Z* vào bằng 0^0. Bởi vì giá trị hệ số *K* chỉ dao động trong một khoảng cố định biết trước, nên Mô-đun *FMul_k*$_i$ chỉ xử lý phần giá trị của *K* mà bỏ qua phần dấu và phần mũ của nó.

Mô-đun tính hệ số *K* và chuẩn hóa ngõ ra (*FMulK_andNorm*)



Hình 7. Sơ đồ khối của Mô-đun FMulK_andNorm

Mục đích của Mô-đun *FMulK_andNorm* là để nhân hai giá trị của *X* và *Y* đến từ Mô-đun *FAdd_XY* với hệ số *K* đến từ Mô-đun *FMul_k*$_i$, sau đó chuẩn hóa kết quả ngõ ra dạng 32-bit dấu chấm động IEEE-754. Hình 7 cho thấy sơ đồ khối của Mô-đun này. Bởi vì phép nhân không cần quy đồng hệ số mũ nên giá trị mantissa của *K* được nhân trực tiếp với phần mantissa của hai giá trị vào là *X* và *Y* . Việc thực thi bộ nhân được dựa vào bộ nhân không dấu tốc độ cao kế thừa từ công trình trước đó của nhóm [25]. Kết quả của bộ nhân sẽ được chuyển đến Mô-đun phát hiện số 1 đầu (Lead-One-Detector – *LOD*) để tìm ra vị trí của bit '1' đầu tiên đang nằm ở vị trí nào trong chuỗi số. Dựa trên thông tin từ Mô-đun *LOD*, Mô-đun *Left_Shifter* dịch kết quả đó sang bên trái nhằm bỏ tất cả các bit '0' vô nghĩa ở đằng trước trong chuỗi bit. Đồng thời, phần số mũ của kết quả được giảm bằng lượng bit mà nó dịch trái. Cuối cùng, phần exponent và phần mantissa của kết quả được hoán đổi cho nhau tương ứng như trong Bảng 1 dựa vào tín hiệu *iRec_info*, với *iRec_info* cho biết vị trí của góc *Z* thuộc góc phần tám nào trong vòng tròn lượng giác. Và cuối cùng, Mô-đun *Sign Recovery* cũng căn cứ vào dấu của hai tín hiệu ngõ vào cùng với thông tin đến từ *iRec_info* để quyết định dấu cho ngõ ra.

KẾT QUẢ THỰC NGHIỆM

Thiết kế bộ nhân được xây dựng và kiểm tra trên chip FPGA Stratix IV của Altera và phân tích ASIC trên công nghệ SOTB 65nm. Tài nguyên sử dụng trên FPGA là 7747 LUTs và 625 thanh ghi. Tần số đạt được trên FPGA là 103,9 MHz. Kết quả thực nghiệm của phân tích ASIC được so sánh với các thiết kế khác được trình bày trong Bảng 2. Dựa trên Bảng 2, thiết kế AARC-TF sử dụng 16.858 standard cells chiếm diện tích 86,718 μm^2. Rõ ràng, có thể thấy rằng thiết kế được đề xuất có thời gian trễ là 6,024 ns, tức tương đương với tần số tối đa đạt được là 166 MHz khi so sánh với những thiết kế khác sử dụng cùng một diện tích tương đương. Tuy nhiên, lưu ý rằng thiết kế AARC-TF là một kiến trúc dựa trên nền tảng CORDIC, do đó AARC-TF tính trực tiếp bộ nhân trong khi các thiết kế khác thì sử dụng kiến trúc bảng tra LUT làm cơ sở. Mà kiến trúc bảng tra LUT thì tiêu tốn thêm tài nguyên bộ nhớ để lưu trữ các giá trị của phép tính lượng giác trước đó, dẫn đến các vấn đề bất lợi trong tính toán FFT với số điểm lớn. Vì vậy, kiến trúc AARC-TF hoàn toàn có thể thay thế được cho các phương pháp truyền thống và giải quyết được tối ưu bộ nhớ cho tính toán trong hệ thống FFT có số điểm lớn.

Bảng 2. So sánh kết quả thực thi trên ASIC với những thiết kế khác

	Công nghệ	Kiến trúc TF	Độ trễ (ns)	Standard cells	Diện tích (µm2)
AARC – TF	65 nm SOTB	CORDIC-based	6,024	16.858	86.718
Fused Butterfly với bộ nhân Wallace [18]	45 nm	Bulk LUT-based	4,65	N/A	116.886
Fused Butterfly với MCM [18]	45 nm	Bulk LUT- based	4,08	N/A	97.302
Fused Butterfly với MMCM [18]	45 nm	Bulk LUT- based	4,34	N/A	101.312

Bảng 3 cho biết thời gian và độ chính xác của bộ nhân số phức được thể hiện qua cả hai thiết kế trên FPGA và phân tích trên ASIC. Bởi vì độ trễ của kiến trúc chỉ dựa trên giá trị của góc ngõ vào, nên để kiểm tra tốc độ của thiết kế, 3600 góc ngõ vào trong khoảng từ 0^0 đến $359,9^0$, với $0,1^0$ bước nhảy được sử dụng để kiểm nghiệm. Kết quả thực nghiệm cho thấy kiến trúc cần 22.046 clock để tính cho tất cả 3.600 góc. Vì vậy, nó mất trung bình 6.124 clock cho mỗi góc. Như vậy, ứng với tần số tối đa của thiết kế trên chip FPGA và tổng hợp trên ASIC lần lượt là 103,9 MHz và 166 MHz, ta có tốc độ tương ứng là 16,966 MSps và 27,107 MSps. Về độ chính xác, giá trị MSE là một thông số luôn luôn cần thiết cho các hệ thống xử lý tín hiệu số. Tuy nhiên, trong thực thi dấu chấm động, tỉ lệ lỗi lớn nhất là cần thiết bên cạnh những giá trị MSE. Thiết kế đề xuất có độ chính xác là 1,133E-10 MSE và 25,894 ppm tỉ lệ lỗi tối đa như có thể thấy trong Bảng 3.

Bảng 3. Độ chính xác và thời gian trễ

	Độ trễ trung bình	Tốc độ trung bình (MSps)	Độ chính xác	Tỉ lệ lỗi tối đa (ppm)
Stratix IV	6,124	16,966	1,133E – 10	25,894
SOTB 65nm		27,107		

KẾT LUẬN

Bài báo đề xuất một kiến trúc tính trực tiếp bộ nhân số phức dấu chấm động sử dụng thuật toán AARC. Thiết kế thực thi trên chip FPGA Stratix IV của Altera và tổng hợp ASIC trên công nghệ SOTB 65 nm với dữ liệu dấu chấm động có độ chính xác đơn. Kết quả thực nghiệm cho thấy, bộ nhân có độ chính xác cao với 1,133E-10 MSE và 25,894 ppm tỉ lệ lỗi tối đa. Tần số đạt được là 103,9 MHz trên chip FPGA và 166 MHz trên tổng hợp ASIC. Với độ trễ trung bình là 6.124 clock trên mỗi góc xoay, tốc độ thực thi đạt được trên chip FPGA và tổng hợp ASIC tuần tự là 16,966 MSps và 27,107 MSps. Tài nguyên sử dụng trong thiết kế này là tương đối nhỏ. Cụ thể, chip FPGA sử dụng 7.747 LUTs và 625 thanh ghi trong khi tổng hợp ASIC sử dụng 16.858 standard cells trên diện tích 86,718 μm^2. Kết luận, thiết kế bộ nhân trực tiếp sử dụng kiến trúc AARC-TF có thể khắc phục vấn đề về bộ nhớ trong hệ thống FFT có số điểm lớn. Vì thế, đề xuất TF có thể được sử dụng cho thực thi FFT số điểm lớn dấu chấm động tốc độ cao.

Lời cảm ơn: *Bài báo là kết quả hợp tác nghiên cứu giữa Phòng Thí Nghiệm DESLAB, Khoa Điện tử–Viễn thông, Trường Đại học Khoa học Tự nhiên–ĐHQG-HCM và Phòng Thí Nghiệm VLSI, Trường Đại Học Điện tử - Truyền thông (University of Electro-Communications - UEC), Tokyo, Nhật Bản. Nghiên cứu được tài trợ bởi Đại học Quốc gia Thành phố Hồ Chí Minh (ĐHQG - HCM) trong khuôn khổ Đề tài mã số B2017-18-05.*

An efficient floating-point FFT twiddle factor implementation based on adaptive angle recoding CORDIC algorithm

- **Vo Thi Phuong Thao**
- **Truong Thi Nhu Quynh**
- **Hoang Trong Thuc**
- **Le Duc Hung**

University of Science, VNU-HCM

ABSTRACT

In this paper, a single-precision floating-point FFT twiddle factor (TF) implementation is proposed. The architecture is based on the Adaptive Angle Recoding CORDIC (AARC) algorithm. The TF design was built and verified on Altera Stratix IV FPGA chip and 65nm SOTB synthesis. The FPGA implementation had 103.9 MHz maximum frequency, throughput result of 16.966 Mega-Sample per second (MSps), and resources utilization of 7.747 ALUTs and 625 registers. On the other hand, the SOTB synthesis has 16.858 standard cells on an area of 298x291 μm^2, 166 MHz maximum frequency, and the speed of 27.107 MSps. The accuracy results were 1.133E-10 Mean-Square-Error (MSE) and about 26 part-per-million (ppm) maximum error.

Keywords: *floating-point FFT Twiddle Factor , FFT, AARC CORDIC*

TÀI LIỆU THAM KHẢO

[1]. J.W. Cooley, J.W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*, 19, 297–301 (1965).

[2]. J.G. Andrews, A. Ghosh, R. Muhamed, *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Prentice Hall Communications Engineering and Emerging Technologies Series (2007).

[3]. E. Dahlman, *3G Evolution: HSPA and LTE for Mobile Broadband*, Academic Press (2008).

[4]. A.F. Molisch, X. Zhang, FFT-based hybrid antenna selection schemes for spatially correlated MIMO channels, *IEEE Communications Letters*, 8, 1, 36–38 (2004).

[5]. Y. Tang, B. Vucetic, The FFT-based multiuser detection for DSCDMA ultra - wideband communication systems, in *Ultra Wideband Systems Joint with Conf. on Ultrawideband Systems and Technologies*, Kyoto, Japan, 111–115 (2004).

[6]. V. Arunachalam, A.N.J. Raj, Efficient VLSI Implementation of FFT for orthogonal frequency division multiplexing applications, *IET Circuits, Devices & Systems*, 8, 6, 526 – 531 (2014).

[7]. C.F. Gerald, P.O. Wheatley, *Applied Numerical Analysis*, 7th ed. Addison Wesley (2003).

[8]. A. Cortes, I. Velez, I. Zalbide, A. Irizar, J. Sevillano, An FFT Core for DVB-T/DVB-H Receivers, in *IEEE 13th Int. Conf. on Electronics, Circuits and Systems*, Nice, France, 102–105 (2006).

[9]. C. Lin, Y. Yu, L. Van, A Low-power 64-point FFT/IFFT Design for IEEE 802.11g WLAN Application, *in Proc. of Int. Symp. on Circuits and Systems*, 4523–4526 (2006).

[10]. C. Le, S. Chan, F. Cheng, W. Fang, M. Frischman, S. Hensley, R. Johnson, M. Jourdan, M. Marina, B. Parham, F. Rogez, P. Rosen, B. Shah, S. Taft, Onboard FPGA-based SAR Processing for Future Spaceborne Systems, *in Proc. of the IEEE Radar Conf.*, 15–20 (2004).

[11]. J. Li, M.V. Sarunic, L. Shannon, Scalable, High Performance Fourier Domain Optical Coherence Tomography: Why FPGAs and not GPGPUs, in *IEEE 19th Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 49–56 (2011).

[12]. E.E. Swartzlander, Systolic FFT Processor: Past, Present and Future, *in Proc. of 2006 Int. Conf. on Application Specific Systems, Steamboat Springs*, 153–158 (2006).

[13]. R. Radhouane, P. Liu, C. Modlin, Minimizing the Memory Requirement for Continuous Flow FFT Implementation: Continuous Flow Mixed Mode FFT (CFMM – FFT), in *IEEE Int. Symp. on Circuits and Systems* (ISCAS 2000), I–116 I–119 (2000).

[14]. J. Zhou, Y. Dong, Y. Dou, Y. Lei, Dynamic Configurable FloatingPoint FFT Pipelines and Hybrid-Mode CORDIC on FPGA, in *2008 Int. Conf. on Embedded Software and Systems (ICESS'08),* 616–620 (2008).

[15]. E. Oruklu, X. Xiao, J. Saniie, Reduced memory and low power architecture for CORDIC-based FFT processors, *Journal of Signal Processing Systems*, 2, 66, 129–134 (2011).

[16]. V. Montano, M. Jiménez, Design and Implementation of a Scalable ˜ Floating-point FFT IP Core for Xilinx FPGAs, in *53rd IEEE Int. Midwest Symp. on Circuits and Systems*, 533–536 (2010).

[17]. S. Mou, X. Yang, Research on the RAW Dependency in Floatingpoint FFT Processors, *in 8th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel Distributed Computing*, Qingdao, China, pp. 88 – 92 (2007).

[18]. J.H. Min, S.W. Kim, E.E. Swartzlander Jr., A FloatingPoint Fused FFT Butterfly Arithmetic Unit with Merged MultipleConstant Multipliers, in *2011 Conf. Record of the 45th Asilomar Conf. on Signals, Systems and Computers (ASILOMAR)*, 520 – 524 (2011).

[19]. H.Y. Lee, I.C. Park, Balanced binary-tree decomposition for areaefficient pipelined FFT Processing, *IEEE Trans. on Circuits and Systems I: Regular Papers*, 54, 4, 889 – 900 (2007).

[20]. X. Xiao, E. Oruklu, J. Saniie, An Efficient FFT Engine with Reduced Addressing Logic, IEEE Trans. on Circuits and Systems II: Express Briefs, 55, 11, 1149–1153 (2008).

[21]. Y.H. Hu, S. Naganathan, An angle recoding method for CORDIC algorithm implementation, *IEEE Trans. on Computer*, 42, 1, 99–102 (1993).

[22]. P.K. Meher, S.Y. Park, CORDIC designs for fixed angle of rotation, *IEEE Trans. on VLSI System*, 21, 2, 217–228 (2013).

[23]. S. Aggarwal, P.K. Meher, K. Khare, Scale free hyperbolic CORDIC processor and its application to waveform generation, *IEEE Trans. on Circuits and Systems-I: Regular Papers*, 60, 2, 314–326 (2013).

[24]. N.H. Thu, N.X. Thuan, H.T. Thuc, L.Đ. Hung, P.C. Kha, A low-resource low-latency hybrid adaptive CORDIC with floating-point precision, *IEICE Electronics Express (ELEX)*, 12, 9, 20150258 (2015).

[25]. L.X. Vy, H.T. Thuc, B.T. Tu, D.D.A. Vu, A High-speed Unsigned 32-bit Multiplier Based on Booth-encoder and Wallace-tree Modifications, in *The 2014 IEEE Int. Conf. on Advanced Technologies for Communications (ATC'14)*, 739–744 (2011).