BỘ GIÁO DỤC VÀ ĐÀO TẠO BỘ QUỐC PHÒNG

HỌC VIỆN KỸ THUẬT QUÂN SỰ

SÁI VĂN THUẬN

NGHIÊN CỨU THIẾT KẾ PHẦN CỨNG THỰC HIỆN CÁC HÀM TOÁN HỌC ĐIỂN HÌNH TRÊN CƠ SỞ PHƯƠNG PHÁP XẤP XỈ ÁP DỤNG TRONG XỬ LÝ TÍN HIỆU SỐ

LUẬN ÁN TIẾN SĨ KỸ THUẬT

HÀ NỘI - 2017

BỘ GIÁO DỤC VÀ ĐÀO TẠO BỘ QUỐC PHÒNG

HỌC VIỆN KỸ THUẬT QUÂN SỰ

SÁI VĂN THUẬN

NGHIÊN CỨU THIẾT KẾ PHẦN CỨNG THỰC HIỆN CÁC HÀM TOÁN HỌC ĐIỂN HÌNH TRÊN CƠ SỞ PHƯƠNG PHÁP XẤP XỈ ÁP DỤNG TRONG XỬ LÝ TÍN HIỆU SỐ

LUẬN ÁN TIẾN SĨ KỸ THUẬT

Chuyên ngành: KỸ THUẬT ĐIỆN TỬ
Mã số: 9 52 02 03

NGƯỜI HƯỚNG DẪN KHOA HỌC:

1. TS TRẦN VĂN KHẨN
2. PGS. TS HOÀNG VĂN PHÚC

HÀ NỘI - 2017

LỜI CAM ĐOAN

Tôi xin cam đoan các kết quả trình bày trong luận án là công trình nghiên cứu của tôi dưới sự hướng dẫn của các cán bộ hướng dẫn. Các số liệu, kết quả trình bày trong luận án là hoàn toàn trung thực và chưa được công bố trong bất kỳ công trình nào trước đây. Các kết quả sử dụng tham khảo đều đã được trích dẫn đầy đủ và theo đúng quy định.

Hà Nội, ngày 19 tháng 5 năm 2017

Tác giả

Sái Văn Thuận

LỜI CẢM ƠN

Trong quá trình nghiên cứu và hoàn thành luận án, nghiên cứu sinh đã nhận được nhiều sự giúp đỡ và đóng góp quý báu.

Trước tiên, nghiên cứu sinh xin bày tỏ lòng biết ơn sâu sắc đến các Thầy hướng dẫn PGS. TS **Hoàng Văn Phúc** và TS **Trần Văn Khẩn**. Các Thầy không những là người hướng dẫn, giúp đỡ nghiên cứu sinh hoàn thành luận án này mà còn là người truyền thụ động lực, quyết tâm cho nghiên cứu sinh vượt qua những khó khăn trong quá trình nghiên cứu.

Nghiên cứu sinh chân thành cảm ơn các Thầy trong Ban chủ nhiệm Khoa Vô tuyến điện tử và tập thể Thầy, Cô giáo trong Bộ môn Thông tin và Bộ môn Kỹ thuật vi xử lý đã luôn quan tâm, tạo điều kiện, giúp đỡ nghiên cứu sinh trong quá trình học tập tại Khoa và Bộ môn.

Tiếp theo, nghiên cứu sinh xin chân thành cảm ơn Phòng Sau đại học-Học viện Kỹ thuật Quân sự, Học viện Phòng không -Không quân đã luôn tạo điều kiện, giúp đỡ để nghiên cứu sinh hoàn thành được luận án này.

Cuối cùng, nghiên cứu sinh xin gửi lời cảm ơn sâu sắc tới bạn bè và các đồng nghiệp đã luôn động viên, giúp đỡ và chia sẻ những khó khăn trong quá trình làm luận án. Xin gửi lời cảm ơn đến những người thân trong gia đình đã động viên và là chỗ dựa vững chắc cho nghiên cứu sinh hoàn thành luận án.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Nghĩa Tiếng Anh	Nghĩa Tiếng Việt
ADP	Area-Delay product	Tích diện tích và độ giữ chậm
ASIC	Application Specific Integrated Circuit	Công nghệ ASIC
CORDIC	Coordinate Rotation Digital Computer	Thuật toán CORDIC
DAC	Digital to analog converter	Bộ chuyển đổi số thành tương tự
DDFS	Direct Digital Frequency Synthesizer	Bộ tổ hợp tần số trực tiếp
DSP	Digital Signal Processing	Xử lý tín hiệu số
FPGA	Field programmable gate array	Công nghệ FPGA
FSM	Finite State Machine	Phương pháp máy trạng thái hữu hạn
HNS	Hybrid Number System	Hệ thống số lai
LNS	Logarithmic Number System	Hệ thống số logarithm

LOD	Leading-One Detector	Bộ phát hiện bit '1' đầu tiên
LPF	Low-pass Filter	Lọc thông thấp
LSB	Least Significant Bit	Bit có trọng số nhỏ nhất
LUT	Look-up Table	Bảng tra
MAE	Maximum Absolute Error	Lỗi tuyệt đối cực đại
MAEP	Maximum Absolute Error Percentage	Phần trăm lỗi tuyệt đối cực đại
MSB	Most Significant Bit	Bit có trọng số lớn nhất
MTM	Multipartite Table Method	Phương pháp bảng nhiều phần
MUX	Multiplexer	Bộ ghép kênh
PA	Phase Accumulator	Bộ tích lũy pha
PAC	Phase to Amplitude Converter	Bộ chuyển đổi pha thành biên độ
ROM	Read-only Memory	Bộ nhớ chỉ đọc
SBTM	Symmetric Bipartite Table Method	Phương pháp bảng hai phần đối xứng
SC	Stochastic Computing	Tính toán ngẫu nhiên
SFDR	Spurious Free Dynamic Range	Khoảng động nhiễu tự do
SN	Stochastic Numbers	Các số ngẫu nhiên
SNG	Stochastic Number Generator	Bộ tạo số ngẫu nhiên

VHDL	Very High Speed Integrated Circuit Hardware Description Language	Ngôn ngữ mô tả phần cứng VHDL
VLSI	Very Large Scale Integration	Công nghệ tích hợp cỡ rất lớn

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG

DANH MỤC KÝ HIỆU TOÁN HỌC

Ký hiệu	**Ý nghĩa**
x	x là biến số
$\log_2(\bullet)$	lô-ga-rít cơ số 2
$\binom{n}{k}$	$\triangleq \frac{n!}{k!(n-k)!}$, với $0 \leq k \leq n$
$\lfloor \bullet \rfloor$	Lấy phần nguyên
$\lceil \bullet \rceil$	lấy số nguyên lớn hơn gần nhất
$sigmoid$	Hàm sigmoid, $sigmoid \triangleq \frac{1-e^{-x}}{1+e^{-x}}$
$tanh$	Hàm tang hyperbol, $tanh \triangleq \frac{1-e^{-2x}}{1+e^{-2x}}$

MỞ ĐẦU

1. Bối cảnh và động lực nghiên cứu

Hiện nay các hệ thống xử lý tín hiệu số (DSP: Digital Signal Processing) có mặt trong hầu hết các hệ thống điện tử và truyền thông. Cùng với các nhu cầu ngày càng tăng của các ứng dụng DSP, thị trường các thiết bị DSP đã có những tăng trưởng mạnh mẽ. Bên cạnh đó, sự phát triển nhanh chóng của các thiết bị và hạ tầng cho truyền thông không dây là một lý do quan trọng cho sự tăng trưởng của thị trường DSP toàn cầu.

Cùng với nhu cầu ngày càng lớn của các ứng dụng DSP có công suất thấp, tài nguyên nhỏ, hiệu năng cao và sự phát triển không ngừng của công nghệ mạch tích hợp, có thể thấy xu hướng của các DSP trong tương lai là:

- Các lõi DSP hiện đại sẽ hoạt động ở tần số rất cao. Ví dụ như bộ xử lý DSP C6000 của hãng Texas Instruments có tần số hoạt động là 1,2 GHz. Tuy nhiên tần số công tác vẫn là một hạn chế và cần có các giải pháp khác nhau để khắc phục hạn chế này.

- Các kiến trúc đa xử lý và đa lõi ngày các được ứng dụng nhiều trong các bộ xử lý để tăng hiệu năng của các hệ thống DSP và là một giải pháp cho bài toán hạn chế tần số công tác. Hơn nữa kiến trúc đa lõi trong các bộ xử lý cũng là một phương pháp đạt được hiệu năng cao.

- Các kiến trúc lai (hay trộn) giữa các lõi DSP, đơn vị vi điều khiển (MCU: microcontroler Unit) và bộ đồng xử lý cũng là một xu hướng cho các hệ

thống DSP hiện đại và tương lai để ứng dụng cho các bộ xử lý đa ứng dụng và các hệ thống đa phương tiện.

- Nhiều thành phần chuyên dụng sẽ được tính toán và thiết kế trước trong các hệ thống DSP để tăng tốc hệ thống và tạo ra khả năng xử lý tín hiệu thích nghi hiệu năng cao.

- Các hệ thống số lai (HNS: Hybrid Number System) và hệ thống số trộn (Mixed Radix Number System) hứa hẹn là một giải pháp tốt cho cân bằng giữa hiệu năng tính toán và công suất tiêu thụ.

Các hàm toán học như hàm lượng giác ($\sin x,\ \cos x$), hàm mũ (2^x), hàm logarithm ($\log_2(x)$), hàm nghịch đảo ($1/x$), hàm căn bậc hai ($\sqrt{x}$) (trong luận án này gọi chung là *các hàm cơ sở*), là rất phổ biến trong nhiều ứng dụng xử lý tín hiệu số và giữ vai trò quan trọng trong quyết định tốc độ xử lý và tài nguyên tiêu tốn của các bộ xử lý. Các bộ xử lý hiện đại với yêu cầu ngày càng cao về tốc độ, chiếm ít tài nguyên và công suất thấp đặt ra yêu cầu phải có các giải pháp thiết kế các đơn vị phần cứng tính toán các hàm toán học trên một cách hiệu quả.

Hơn nữa, cùng với xu hướng phát triển mạnh mẽ của thị trường DSP các hệ thống và thiết bị DSP sẽ được sử dụng ngày càng nhiều trong các ứng dụng. Cùng với đó là càng nhiều yêu cầu đặt ra đối với các lõi phần cứng chuyên dụng cho tính toán các hàm toán học cả về hiệu năng và ứng dụng, nhất là trong các hệ thống xử lý ảnh 3-D và hệ thống đa phương tiện tốc độ cao. Việc sử dụng nhiều các mạch nhân, đặc biệt là các kiến trúc song song có thể dẫn đến độ phức tạp hệ thống cao và công suất tiêu thụ lớn. Như đã chỉ ra trong [1], một ứng dụng dựng ảnh 3-D cần 90% các thao tác là thực

hiện các phép tính chia, nhân, khai căn. Theo nghiên cứu của B. G. Nam và các cộng sự trong [2] cho thấy một bộ xử lý ảnh 3-D điển hình sử dụng rất nhiều các phép tính như nhân, chia, bình phương và lũy thừa cùng với các thao tác cơ bản như cộng và trừ. Do đó, các thành phần tính toán các thao tác số học với độ phức tạp phần cứng thấp là một yêu cầu đặt ra cho các ứng dụng DSP.

Sự phát triển không ngừng của công nghệ tích hợp cho phép thiết kế và thực thi các bộ xử lý tinh vi với các thuật toán phức tạp có tốc độ tính toán cao. Ngoài ra, mật độ tích hợp cao cũng tạo ra các mạch DSP nhỏ hơn. Tuy nhiên, với nhu cầu ngày càng tăng nhanh của các thiết bị điện tử thông minh không dây, các thiết bị cầm tay, điện thoại di động, yêu cầu về hệ thống DSP công suất thấp và nhỏ gọn ngày càng đặt ra cấp thiết. Các ứng dụng này, không chỉ yêu cầu tốc độ xử lý cao mà còn đòi hỏi công suất tiêu thụ thấp, tài nguyên tiêu tốn ít (nhất là các thiết bị sử dụng pin). Trong khi đó, các bộ nhân song song như đã nói ở trên thường làm cho hệ thống có độ phức tạp cao và công suất tiêu thụ lớn.

Vì vậy, cần có các giải pháp khác nhau để đạt được kiến trúc tính toán hiệu quả mà không phải sử dụng nhiều mạch nhân phức tạp trong khi vẫn đạt được hiệu năng tính toán cần thiết. Do đó, nhiều nghiên cứu đã tập trung tìm kiếm các giải pháp khác nhau để đạt được các kiến trúc tính toán hiệu quả. Cùng với sự phát triển mạnh mẽ của công nghệ mạch tích hợp, thì một trong những giải pháp hứa hẹn đạt được hiệu quả đó là phương pháp tính toán dựa trên bộ nhớ hay tổng quát hơn là tính toán dựa trên bảng tra (LUT: Lookup Table).

Theo báo cáo của chỉ dẫn quốc tế về công nghệ bán dẫn (ITRS: Interna-

tional Technology Roadmap for Semiconductors) năm 2010 các bộ nhớ nhúng đã trở nên nhanh hơn, mật độ tích hợp cao hơn, công suất tiêu thụ nhỏ hơn và chiếm phần lớn trong các hệ thống trên chip (SoC: System on Chip). Theo báo cáo của ITRS xu thế sử dụng bộ nhớ trong SoC năm 2014 là hơn 92%. Vì vậy, tính toán dựa trên cơ sở bảng tra LUT kết hợp với một số mạch logic đơn giản là một giải pháp hứa hẹn đạt được hiệu quả cao. Bởi vì, tính toán dựa trên bảng LUT khác với tính toán trực tiếp bởi các mạch logic phức tạp. Nó tạo ra kết quả tính toán bằng cách truy cập vào bộ nhớ đã được lưu trữ các kết quả được tính toán trước. Vì vậy, nó sẽ tăng tốc độ tính toán vì thời gian chủ yếu là để truy nhập bộ nhớ. Phương pháp này cũng hứa hẹn làm giảm công suất tiêu thụ động bởi vì nó tối thiểu hóa tốc độ chuyển mạch. Nhược điểm của tính toán trên bảng LUT là khi toán hạng đầu vào có độ rộng bit lớn thì sẽ làm tăng kích thước của bảng LUT. Do đó, để giảm kích thước của bảng LUT cần phải có các giải pháp hiệu quả. Trong đó, phương pháp kết hợp bảng LUT với một vài mạch logic đơn giản dựa trên các thuật toán tối ưu là một giải pháp hứa hẹn đạt được các kiến trúc phần cứng đơn giản và hiệu quả về tốc độ, công suất thấp, tài nguyên tiêu tốn ít. Vì vậy, luận án sẽ tập trung nghiên cứu thực thi một vài hàm toán học ứng dụng trong DSP dựa trên phương pháp nêu trên.

Ngoài ra, trong một vài năm gần đây kỹ thuật tính toán ngẫu nhiên (SC: Stochastic computing) thu hút được sự quan tâm trong thiết kế phần cứng các mạch số học có độ phức tạp rất thấp. Đó cũng là cơ sở để tạo động lực cho nghiên cứu trong luận án này nhằm đạt được các kiến trúc tính toán hiệu quả dựa trên SC.

2. Đối tượng và phạm vi nghiên cứu

Luận án này tập trung nghiên cứu tìm phương pháp để đạt được kiến trúc phần cứng hiệu quả cho thực hiện các hàm toán học phổ biến ứng dụng trong xử lý tín hiệu số. Vì các ứng dụng DSP có đặc điểm riêng, nó khác với các ứng dụng tính toán nói chung nên phương pháp thực thi và kiến trúc cũng phải phù hợp. Một số đặc điểm quan trọng của các ứng dụng DSP được xem xét như sau:

- Cho phép lỗi: lỗi có thể được chấp nhận trong các ứng dụng DSP chẳng hạn như trong các bộ lọc số, xử lý ảnh và video. Vì vậy, độ phức tạp phần cứng có thể giảm đáng kể bằng việc sử dụng một số kỹ thuật như cắt bớt độ dài bit trong kết quả cuối cùng trong một số thao tác số học hay như rút gọn kích thước của bảng LUT trong một số trường hợp.
- Các ứng dụng DSP với độ phức tạp thấp là đòi hỏi ngày càng cao do nhu cầu lớn của các thiết bị điện tử cầm tay và di động. Các thiết bị điện tử ngày càng trở lên nhỏ gọn hơn. Mặc dù công nghệ mạch tích hợp đã phát triển mạnh nhưng vẫn có những đòi hỏi cao cho việc phát triển các phương pháp hiệu quả ở mức kiến trúc và mức hệ thống cho các ứng dụng DSP độ phức tạp thấp.
- Các ứng dụng DSP công suất thấp đã ngày càng phổ biến hơn do nhu cầu ngày càng tăng của các thiết bị dùng pin. Vì vậy, nghiên cứu các phương pháp thiết kế để đưa ra các kiến trúc nhằm giảm công suất tiêu thụ là một yêu cầu đặt ra cho các hệ thống DSP.

Vì vậy đối tượng và phạm vi nghiên cứu của luận án là tìm các phương pháp hiệu quả phù hợp với các ứng dụng xử lý tín hiệu số theo các đặc điểm như trên.

3. Đóng góp của luận án

Một số đóng góp chính của luận án có thể được tóm tắt như sau:

1. Đề xuất một phương pháp phương pháp xấp xỉ hàm logarithm nhị phân và kiến trúc phần cứng thực hiện chuyển đổi logarithm nhị phân. Tính toán trên miền logarithm sẽ đơn giản một số thao tác phức tạp từ đó giảm độ phức tạp phần cứng. Ngoài ra, tính toán hàm logarithm là một thao tác không thể thiếu trong nhiều ứng dụng DSP. Kiến trúc của bộ chuyển đổi logarithm nhị phân đề xuất đạt được độ phức tạp phần cứng thấp. Đóng góp của đề xuất này được trình bày trong các công trình số 1 và số 2.

2. Đề xuất một phương pháp cải tiến cho bộ chuyển đổi pha thành biên độ cho tính toán hàm sin ứng dụng trong bộ tổng hợp tần số số trực tiếp. Phương pháp đề xuất đã cải thiện độ phức tạp kiến trúc bộ chuyển đổi pha thành biên độ trong DDFS. Đóng góp của đề xuất này được trình bày trong công trình số 3.

3. Đề xuất một phương pháp xấp xỉ và các hàm toán học phổ biến trong DSP dựa trên xấp xỉ tuyến tính hai mức. Phương pháp đề xuất đã được áp dụng cho thực thi phần cứng 6 hàm toán học điển hình. Các kết quả thực thi cho thấy phương pháp đề xuất đạt được hiệu quả về tốc độ. Đóng góp của đề xuất này được trình bày trong công trình số 4.

4. Đề xuất phương pháp thực thi phần cứng tính toán các hàm toán học dựa trên tính toán ngẫu nhiên kết hợp với xấp xỉ các hàm toán học phức tạp bằng các hàm tuyến tính phân đoạn đều. Các kết quả thực thi theo

phương pháp đề xuất đã cải thiện được hiệu năng của các lõi tính toán. Đóng góp của đề xuất này được trình bày trong công trình số 5 và số 6.

4. Bố cục luận án

Luận án được tổ chức thành 5 chương, bố cục cụ thể như sau:

- Chương 1: CƠ SỞ VÀ PHƯƠNG PHÁP NGHIÊN CỨU

  Trong chương này trình bày những kiến thức cơ sở liên quan tới luận án bao gồm: khái quát về sử dụng các hàm toán học trong các ứng dụng DSP; các phương pháp thiết kế phần cứng cho tính toán hàm toán học. Chương này cũng trình bày phương pháp chung cho thực thi phần cứng các hàm toán học được áp dụng trong luận án.

- Chương 2: THIẾT KẾ PHẦN CỨNG TÍNH TOÁN LOGARITHM NHỊ PHÂN

  Trong chương này trình bày tổng hợp các phương pháp thực thi hàm logarithm nhị phân trên cơ sở phương pháp của Mitchell. Từ đó đề xuất một phương pháp xấp xỉ hàm logarithm dựa trên kỹ thuật xấp xỉ phân đoạn tuyến tính với các hệ số tối ưu có kết hợp bảng LUT sửa lỗi. Đề xuất một kiến trúc phần cứng tính toán hàm logarithm nhị phân, thực thi kiến trúc đề xuất trên FPGA và ASIC được thực hiện và so sánh với các nghiên cứu trước đó.

- Chương 3: THIẾT KẾ PHẦN CỨNG TÍNH TOÁN HÀM SIN

  Chương này trình bày một cải tiến trong phương pháp tính toán hàm *sin* ứng dụng cho bộ tổ hợp tần số trực tiếp. Phương pháp đề xuất dựa trên xấp xỉ tuyến tính đều có kết hợp với LUT sửa lỗi. Các phân tích và

tối ưu số được sử dụng để đạt được lỗi xấp xỉ tối thiểu và kiến trúc phần cứng có độ phức tạp thấp. Dựa trên phương pháp đề xuất, một kiến trúc bộ tổ hợp tần số trực tiếp được đề xuất và thực thi. Các kết quả thực thi được so sánh với các nghiên cứu tương tự cho thấy đạt được cải thiện về tài nguyên phần cứng.

- Chương 4: THIẾT KẾ PHẦN CỨNG TÍNH TOÁN HÀM TOÁN HỌC DỰA TRÊN XẤP XỈ TUYẾN TÍNH HAI MỨC

  Chương này sẽ trình bày một phương pháp mới để thực thi các hàm toán học phổ biến trong xử lý tín hiệu số. Phương pháp đề xuất dựa trên xấp xỉ tuyến tính hai mức trong đó có sử dụng nội suy đối xứng. Kiến trúc tổng quát của phương pháp đề xuất được đưa ra và áp dụng cho thực thi 6 hàm toán học điển hình. Các kết quả thực thi trên FPGA và ASIC cho thấy hiệu quả về tốc độ thực thi.

- Chương 5: THIẾT KẾ PHẦN CỨNG TÍNH TOÁN CÁC HÀM TOÁN HỌC SỬ DỤNG LOGIC NGẪU NHIÊN VÀ XẤP XỈ PHÂN ĐOẠN TUYẾN TÍNH ĐỀU

  Tính toán các hàm toán học như các hàm lượng giác, hàm logarithm, hàm mũ, hàm sigmoid, tanh là đòi hỏi đặt ra trong nhiều ứng dụng của DSP và mạng nơ-ron. Chương này trình bày một phương pháp mới để thực thi phần cứng tính toán các hàm toán học nói trên. Phương pháp đề xuất dựa trên xấp xỉ các hàm toán học bằng các các hàm tuyến tính phân đoạn đều. Sau đó, các hàm xấp xỉ tuyến tính được thực thi dựa trên logic ngẫu nhiên. Các kết quả thực thi cho thấy phương pháp đề xuất đã cải thiện hiệu năng của các mạch tính toán so với các phương pháp trước đó.

Chương 1
CƠ SỞ VÀ PHƯƠNG PHÁP NGHIÊN CỨU

1.1. Các hàm toán học trong xử lý tín hiệu số

Các hệ thống xử lý tín hiệu số thực hiện các chức năng khác nhau đều dựa trên một tập hợp các thao tác tính toán cơ bản được thực hiện trên phần cứng. Các thao tác tính toán cơ bản bao gồm phép cộng, phép nhân cho đến các hàm cơ sở như các hàm lượng giác, hàm mũ, khai căn, logarithm... Các hàm cơ sở đóng vai trò quan trọng trong nhiều ứng dụng bao gồm khoa học tính toán, đồ họa máy tính, các ứng dụng đồ họa 3D, xử lý tín hiệu số và thiết kế có hỗ trợ máy tính (CAD: Computer Aided Design) [3–6]. Thiết kế phần cứng hiệu quả cho tính toán các hàm toán học này có ý nghĩa quan trọng trong nâng cao hiệu năng các hệ thống xử lý tín hiệu số.

Trong ứng dụng DSP, một thao tác thường xuyên là phải tính toán hàm $\sin(\omega t)$ ở đây t là thời điểm lấy mẫu và $t = kt_s$ với $f_s = \frac{1}{t_s}$ là tần số lấy mẫu.

$$\sin(\omega t) = \sin(2\pi\, f\, kt_s) = \sin(2\pi \frac{f}{f_s} k) \tag{1.1}$$

Tần số số của sóng sin f/f_s có độ chính xác yêu cầu là $1/N$ nó là tần số được lượng tử từ $f = \frac{m}{N} f_s$, vậy tất cả hàm cần tính toán có dạng sau:

$$\sin(2\pi \frac{m}{N} k) = \sin(\frac{2\pi}{N} i) \qquad i = mk = 0, ..., N \tag{1.2}$$

Trong các ứng dụng xử lý âm thanh, tần số lấy mẫu f_s tùy thuộc vào các loại âm thanh, ví dụ như tín hiệu thoại sử dụng $f_s = 8KHz$, tần số lấy

mẫu tiếng nói con người là $f_s = 40KHz$, âm nhạc theo tiêu chuẩn MP3 có $f_s = 44,1KHz...$ Như vậy, chẳng hạn với tiếng nói con người với độ chính xác yêu cầu khoảng $0,1Hz$ có khoảng $N = 400000$ giá trị khác nhau được yêu cầu. Do đó, sử dụng bảng tra là không thực tế cho các ứng dụng này. Một phương pháp thường sử dụng để tính toán các hàm lượng giác là dùng xấp xỉ bằng chuỗi Taylor.

$$\begin{aligned} \sin(x) &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots \\ \cos(x) &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots \end{aligned} \tag{1.3}$$

Xấp xỉ này sử dụng một số số hạng đầu tiên của khai triển và điều chỉnh các hệ số của khai triển nhằm tăng độ chính xác. Tuy nhiên phương pháp này đòi hỏi các mạch tính toán tính toán có độ phức tạp lớn.

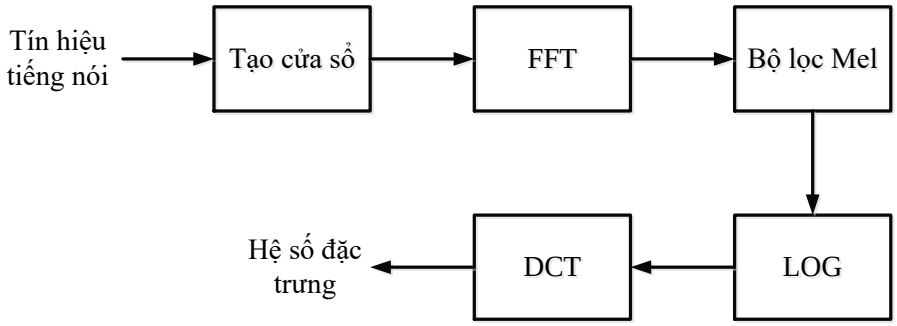
Ngoài ra, tính toán hàm sin có thể được sử dụng trong bộ tổ hợp tần số số, bộ trộn tần, bộ điều chế và giải điều chế và nhiều ứng dụng trong các hệ thống truyền thông số.

Hàm logarithm là thao tác chính trong các ứng dụng chuyển đổi tỷ số công suất và phổ công suất sang biểu diễn dạng đề-xi-bel, trong các ứng dụng xử lý âm thanh [7]. Tính toán hàm logarithm còn là thao tác không thể thiếu trong các hệ thống số logarithm (LNS: Logarithmic Number System) để chuyển đổi dữ liệu đầu vào sang miền logarithm.

Hàm logarithm cơ số hai được quan tâm hơn cả trong thực thi tính toán hàm logarithm. Bởi vì, không mất tính tổng quát, hàm logarithm của các cơ số khác có thể được tính thông qua hàm logarithm cơ số hai như công thức sau:

$$\log_a(x) = (\log_2 a)^{-1} \log_2(x) \tag{1.4}$$

Tính toán hàm logarithm còn được ứng dụng trong kỹ thuật nén ảnh và nhận dạng tiếng nói. Trong nhận dạng tiếng nói và tổng quát là xử lý tiếng nói, trích đặc trưng tiếng nói là một bước tính toán không thể thiếu. Trong đó, MFCC (Mel frequency cepstral coefficients) là một phương pháp hiệu quả cho trích đặc trưng tiếng nói [8]. Hình 1.1 biểu diễn quá trình trích đặc trưng tiếng nói theo phương pháp MFCC.



Hình 1.1: Quá trình trích đặc trưng tiếng nói theo phương pháp MFCC.

Đầu tiên, tín hiệu tiếng nói được tạo khung và tạo cửa sổ. Tạo khung tín hiệu là quá trình chặn tín hiệu tiếng nói thành các khung gồm N mẫu trên miền thời gian. Sau bước tạo khung mỗi một khung đơn lẻ sẽ được tạo cửa sổ bởi một hàm cửa sổ. Việc tạo cửa sổ nhằm tối thiểu hóa sự gián đoạn tại điểm đầu và điểm cuối của mỗi khung. Các khung và hàm cửa sổ sẽ được nhân với nhau. Giả sử hàm cửa sổ được định nghĩa là $W(n)$, $0 \leq n \leq N-1$, ở đây N là số mẫu trong một cửa sổ bất kỳ, thì tín hiệu đầu ra sau khi tạo của sổ sẽ là $Y(n) = X(n) \times \mathrm{W(n)},\ 0 \leq n \leq N-1$ với $X(n)$ là tín hiệu đầu vào khối tạo của sổ. Trong các hàm cửa sổ sử dụng phổ biến nhất là cửa sổ Hamming. Hàm cửa sổ Hamming được định nghĩa như sau

$$\mathrm{W}(n) = 0,54 - 0,46 \times cos(2\pi \mathrm{n}/(\mathrm{N}-1));\ 0 \leq \mathrm{n} \leq \mathrm{N}-1 \qquad (1.5)$$

Sau khi được tạo cửa sổ tín hiệu sẽ được thực hiện biến đổi Fourier nhanh để chuyển tín hiệu từ miền thời gian sang miền tần số. Tín hiệu sau khi thực hiện biến đổi Fourier sẽ được cho qua một bộ lọc Mel để tạo ra thang tần số Mel phù hợp với đặc tính tiếp nhận tần số của hệ thống thính giác con người. Sau bộ lọc Mel, phép tính logarithm sẽ được thực hiện như công thức (1.6). Cuối cùng một phép biến đổi Cosine rời rạc sẽ được thực hiện để tạo ra các hệ số MFCC

$$F_{Mel} = 781 \times \log_2(1 + f_{norm}), \tag{1.6}$$

ở đây F_{Mel} là chỉ đặc trưng MFCC và f_{norm} giá trị tần số chuẩn hóa nhận được bởi mô-đun FFT.

Trong kỹ thuật đồ họa 3D, hơn 80% thời gian xử lý của bộ xử lý đồ họa là để thực hiện dựng ảnh (render), hơn nữa hơn 90% thời gian xử lý dựng ảnh là thực hiện tính toán các hàm toán học cơ sở [1]. Vì vậy, các hàm toán học cơ sở được sử dụng thường xuyên trong ứng dụng xử lý đồ họa 3D và nó quyết định tốc độ xử lý của các đơn vị xử lý đồ họa.

Một thao tác thường xuyên trong tạo ảnh 3D là nội suy giá trị các thuộc tính phổ biến như màu sắc, tọa độ kết cấu từ một giá trị nguyên thủy. Để nhận được giá trị chính xác, cần nội suy một hàm của thuộc tính, kết quả nội suy sau đó được chuyển bằng một hàm nghịch đảo để tạo ra giá trị cuối cùng tại mỗi điểm ảnh. Các hàm khác cũng được thường xuyên sử dụng trong các chương trình dựng bóng điểm ảnh. Mạch tính toán hàm căn bậc hai là một hàm phổ biến trong tạo bóng độ sáng. Ngoài ra, các hàm như 2^x, $\log_2(x)$, $\sin x$ và $\cos x$ là thao tác quan trọng trong chương trình tạo bóng của kỹ thuật đồ họa [9].

Như vậy, tính toán các hàm toán học là yêu cầu đặt ra trong hầu hết các ứng dụng DSP. Với các yêu cầu ngày càng cao về tài nguyên và tốc độ của các bộ xử lý DSP, cần có các giải pháp thiết kế phần cứng hiệu quả cho tính toán các hàm toán học.

1.2. Các phương pháp thực thi phần cứng cho tính toán các hàm toán học

Tính toán các hàm cơ sở là yêu cầu đặt ra trong nhiều ứng dụng của xử lý tín hiệu số. Mặc dù các chương trình phần mềm có thể tính toán các hàm cơ sở một cách chính xác, nhưng chúng thường có độ trễ lớn và không phù hợp với các ứng dụng cường độ tính toán cao và đòi hỏi thời gian thực. Thực thi các hàm cơ sở bằng phần cứng có ưu điểm đáng kể về mặt tốc độ cũng như khả năng tăng thông lượng bằng việc sử dụng song song nhiều mô-đun phần cứng.

Các yêu cầu ngày càng lớn về tốc độ và thông lượng của nhiều ứng dụng là một động lực cho việc phát triển các phương pháp phần cứng để tính toán các hàm cơ sở với tốc độ cao. Hơn nữa, phần cứng tính toán các hàm cơ sở một cách hiệu quả và chính xác là cần thiết trong nhiều ứng dụng như đơn vị xử lý đồ họa (GPUs: Graphics Processing Units), các bộ xử lý tín hiệu số (DSPs: Digital Signal Processors), bộ đồng xử lý số dấu phẩy động của bộ xử lý đa năng và ASIC [10–12].

1.2.1. Phương pháp xấp xỉ bằng đa thức

Phương pháp xấp xỉ bằng đa thức là một giải pháp phổ biến trong xấp xỉ các hàm toán học. Trong đó, một hàm f sẽ được xấp xỉ bởi một đa thức p

có bậc d trong khoảng $[a, b]$. Đa thức xấp xỉ có dạng như sau:

$$p(x) = c_d x^d + c_{d-1} x^{d-1} + ... + c_1 x + c_0, \tag{1.7}$$

với qui tắc của Horner, đa thức trên có thể được viết lại dưới dạng:

$$p(x) = ((c_d x + c_{d-1})x + ...)x + c_0, \tag{1.8}$$

ở đây x là dữ liệu đầu vào. Để tối thiểu hóa sai số xấp xỉ $\varepsilon = \|p - f\|$, phương pháp xấp xỉ minimax thường được sử dụng. Trong đó, cực đại của trị tuyệt đối sai số sẽ được tối thiểu hóa. Sai số cho phương pháp xấp xỉ minimax sẽ là:

$$\|p - f\| = \max_{a \le x \le b} |f(x) - p(x)| \tag{1.9}$$

với $[a, b]$ là khoảng xấp xỉ. Nhiều nghiên cứu đã dựa vào các phương pháp khác nhau như phương pháp sử dụng khai triển Taylor, phương pháp Legendre, phương pháp Chebyshev, xấp xỉ Minimax để tối thiểu hóa lỗi cực đại. Bảng 1.1 biểu diễn sai số cực đại và trung bình sai số của xấp xỉ hàm e^x trên khoảng $[-1,\ 1]$ bằng xấp xỉ sử dụng đa thức bậc nhất với các phương pháp khác nhau. Có thể thấy rằng phương pháp minimax cho sai số cực đại là nhỏ nhất còn phương pháp Legrendre tạo ra trung bình sai số nhỏ nhất. Vì vậy, khi đòi hỏi sai số cực đại là nhỏ thì phương pháp minimax được sử dụng. Đa thức minimax thường được xác định dựa trên kiểu lặp sử dụng thuật toán chuyển đổi Remex [13], nó thường được sử dụng để xác định các hệ số tối ưu cho bộ lọc số.

Sidahao và các cộng sự trong [14] đã xấp xỉ các hàm trên toàn bộ khoảng giá trị của nó bằng các đa thức bậc cao. Phương pháp này có ưu điểm là yêu cầu bộ nhớ thấp, tuy nhiên nó có độ giữ chậm lớn. Hơn nữa, nhìn chung là

Bảng 1.1: Sai số cực đại và trung bình sai số trong xấp xỉ đa thức bậc 1 của hàm e^x trên khoảng $[-1, 1]$ với các phương pháp khác nhau.

Phương pháp	Taylor	Legendre	Chebyshev	Minimax
Lỗi cực đại	0,718	0,439	0,372	0,279
Trung bình lỗi	0,246	0,162	0,184	0,190

khó chấp nhận được với các hàm có tính phi tuyến lớn. Pineiro và các cộng sự trong [15] đã chia khoảng giá trị của hàm thành một số đoạn đều và xấp xỉ mỗi đoạn bằng các đa thức minimax bậc hai. Phương pháp này phù hợp với ước lượng các hàm với độ chính xác vừa phải (nhỏ hơn 24 bit).

1.2.2. Phương pháp bảng chia đôi và bảng nhiều phần

Phương pháp bảng chia đôi (BTM: Bipartite Method) phân chia đầu vào x có độ rộng n bit thành 3 phần x_0, x_1, x_2 với độ rộng bit tương ứng là n_0, n_1, n_2. Giả sử rằng $0 \leq x < 1$, thì đầu vào x có thể được biểu diễn như sau:

$$x = x_0 + x_1 \times 2^{-n_0} + x_2 \times 2^{-(n_0 + n_1)}. \tag{1.10}$$

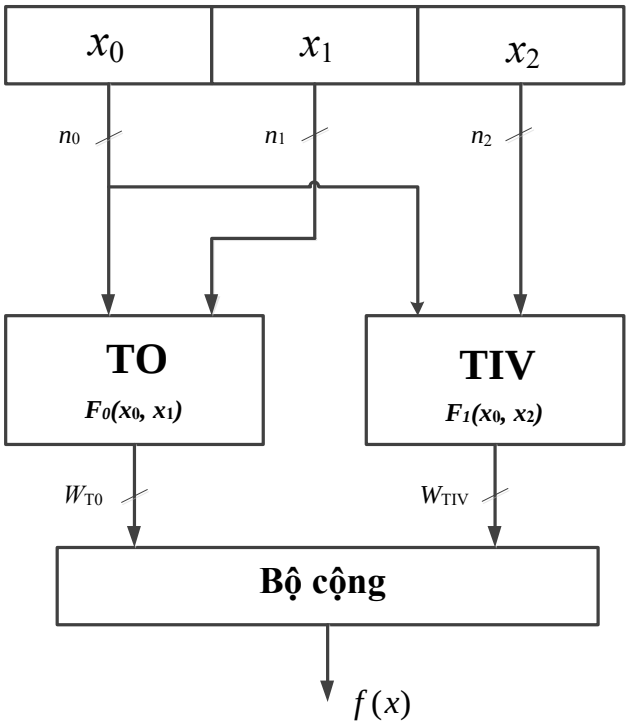
Sử dụng khai triển Taylor tại điểm $x_0 + x_1 \times 2^{-n_0}$, hàm $f(x)$ có thể được biểu diễn như sau:

$$f(x) = f(x_0 + x_1 \times 2^{-n_0}) + x_2 \times 2^{-(n_0+n_1)} f'(x_0 + x_1 \times 2^{-n_0}) + \varepsilon_{apx}, \tag{1.11}$$

ở đây, ε_{apx} biểu diễn lỗi xấp xỉ gây ra bởi việc loại bỏ các số hạng bậc cao trong khai triển. Sau khi xấp xỉ đạo hàm $f'(x_0 + x_1 \times 2^{-n_0})$ bằng $f'(x_0)$ sẽ phát sinh một lỗi khác là ε_{slp}, do đó công thức (1.11) sẽ được viết lại như

sau:

$$f(x) = \underbrace{f(x_0 + x_1 \times 2^{-n_0})}_{F_0(x_0,x_1)} + \underbrace{x_2 \times 2^{-(n_0+n_1)} f'(x_0)}_{F_1(x_0,x_2)} + \varepsilon_{apx} + \varepsilon_{slp}$$
$$\approx \widetilde{f}(x) = F_0(x_0, x_1) + F_1(x_0, x_2). \quad (1.12)$$



Hình 1.2: Phương pháp BTM.

Vậy phương pháp bảng chia đôi xấp xỉ hàm $f(x)$ bởi hàm $\widetilde{f}(x)$ sử dụng hai LUT. LUT thứ nhất được gọi là bảng các giá trị khởi tạo (TIV: Table of Initial Values) sẽ chứa $F_0(x_0, x_1)$ được định chỉ số bằng x_0, x_1 và LUT thứ hai gọi là bảng độ dịch (TO: Table of Offsets) chứa $F_1(x_0, x_2)$ được định chỉ số bởi x_0, x_2. Độ rộng bit của các phần chia nhỏ đầu vào n_0, n_1, n_2 (tương ứng với số đầu vào của LUT) và các độ rộng bit của từ nhớ trong LUT (ký hiệu là w_{TIV} và w_{TO}) sẽ được lựa chọn sao cho thỏa mãn độ chính xác yêu

cầu. Với các thiết kế yêu cầu độ chính xác trung thực thì lỗi tổng cộng thường nhỏ hơn 1 đơn vị có trọng số nhỏ nhất (*ulp*: unit in the least position). Các độ rộng bit w_{TIV}, w_{TO} bao gồm cả các bit yêu cầu đầu ra và các bit bảo vệ, các bit dư thừa sẽ không được xem xét. Như chỉ ra trong [16] lỗi làm tròn cuối cùng có thể giảm nhỏ hơn 0,5 *ulp* nếu chúng ta tạo ra bit 1 ở vị trí tiếp theo của bit bảo vệ cuối cùng trước khi các giá trị được lưu trữ vào trong TIV và các TO. Ngoài ra, trong [17] chỉ ra rằng kích thước của các TO có thể giảm đi một nửa nếu sử dụng tính đối xứng của các nội dung chứa trong TO.

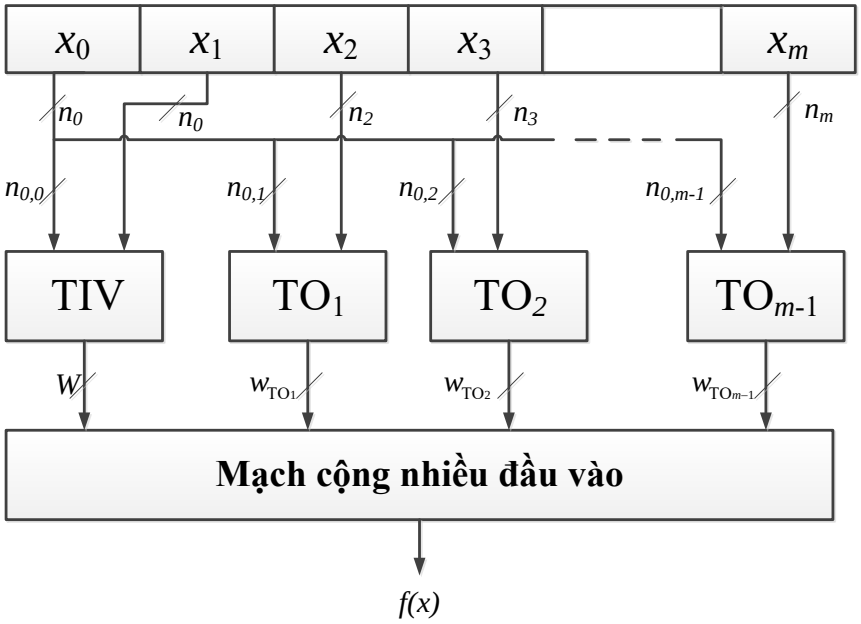
Trong phương pháp cộng bảng đối xứng (STAM: Symmetric Table Addition Method) [18] sử dụng tổng cộng m LUT, đầu vào được chia thành $m + 1$ phần.

$$x = x_0 + x_1 \times 2^{-n_0} + \cdots + x_m \times 2^{-(n_0+n_1+\cdots+n_{m-1})}, \qquad (1.13)$$

với cặp chỉ số (x_0, x_1) của $n_0 + n_1$ bit dùng để định địa chỉ cho TIV và $m - 1$ cặp chỉ số $(x_0, x_{i+1}),\ i = 1, 2, \ldots, m-1$ của $n_0 + n_{i+1}$ bit dùng để định địa chỉ cho $(m-1)$ TOi. So với phương pháp BTM thì khi thực thi phương pháp STAM sẽ đạt được kích thước tổng cộng của các LUT nhỏ hơn bởi vì TO ban đầu đã được chia thành một số TOi, thực ra BTM là một trường hợp đặc biệt của STAM với $m = 2$.

Trong phương pháp bảng nhiều phần [19] ngoài việc khai thác tính đối xứng của các TO như trong [17] [18] kích thước của các TO và kích thước tổng cộng của bảng sẽ giảm hơn nữa bằng việc cho phép phân chia chỉ số đầu tiên trong cặp chỉ số định địa chỉ cho các TO thành các phần khác nhau. $m - 1$ bảng $TO_i,\ i = 1, 2, \ldots, m-1$ sẽ được chỉ số bởi $(x_{0,i}, x_{i+1})$, $i =$

1, 2, ... , $m-1$ ở đây $x_{0,i}$ không nhất thiết phải bằng $x_{0,0}$, cặp chỉ số đầu tiên sẽ được sử dụng để định địa chỉ cho TIV. Vậy, STAM có thể được coi là một trường hợp đặc biệt của phương pháp bảng nhiều phần tổng quát hơn với $x_{0,i} = x_{0,0},\ i = 1, 2, \ldots, m-1$. Hình 1.3 biểu diễn kiến trúc tổng quát của phương pháp bảng nhiều phần. Kích thước của TIV và các TO tương ứng là $2^{n_{0,0}+n_1} \times \mathrm{w}_{TIV} = 2^{\alpha} \times \mathrm{w}$ bit và $2^{n_{0,i}+n_{i+1}} \times \mathrm{w}_{TO_i}$ bit. Ở đây w_{TIV} và w_{TO_i} là độ rộng từ nhớ trong TIV và các TO.



Hình 1.3: Phương pháp MTM.

1.2.3. Thuật toán CORDIC

CORDIC là viết tắt của COordinate Rotations DIgital Computer là phương pháp nổi tiếng để tính toán các hàm toán học một cách đơn giản và hiệu quả. Thuật toán CORDIC được đưa ra bởi Volder trong [20] cho tính toán các hàm lượng giác, nhân, chia và chuyển đổi các kiểu dữ liệu. Sau đó được tổng quát hóa để tính toán cho các hàm hyperbolic bởi Walther [21]. Thuật toán

CORDIC được ứng dụng rộng rãi trong nhiều lĩnh vực tính toán, nó là một phương pháp hiệu quả trong thực thi phần cứng các hàm cơ sở.

Thuật toán CORDIC dựa trên các công thức lặp đơn giản, trong đó việc thực thi chỉ bởi các phép cộng và dịch nhằm tránh việc sử dụng các thao tác nhân và chia phức tạp. Thuật toán CORDIC tổng quát bao gồm ba công thức lặp cơ bản sau:

$$\begin{aligned} x_{k+1} &= x_k - m\delta_k y_k 2^{-k} \\ y_{k+1} &= y_k + \delta_k x_k 2^{-k} \\ z_{k+1} &= z_k - \delta_k \sigma_k. \end{aligned} \tag{1.14}$$

Các hằng số m, δ_k và σ_k phụ thuộc vào tính toán cụ thể được thực hiện, cụ thể như sau:

- m sẽ nhận một trong các giá trị 0,1 hoặc -1. $m = 1$ sử dụng cho tính toán các hàm lượng giác và hàm lượng giác ngược. $m = -1$ sử dụng cho tính toán các hàm hyberbolic và hyberbolic ngược, các hàm mũ, hàm logarithm cũng như hàm căn bậc hai. $m = 1$ được dùng cho thực hiện các thao tác nhân và chia.

- δ_k là một trong hai hàm dấu sau:

$$\delta_k = sign(z_k) = \begin{cases} 1, & z_k \geq 0 \\ -1, & z_k < 0 \end{cases} \text{ hoặc } \delta_k = -sign(y_k) = \begin{cases} 1, & y_k < 0 \\ -1, & y_k \geq 0 \end{cases} \tag{1.15}$$

  Trường hợp đầu thường được gọi là chế độ quay, trong đó giá trị z được điều khiển tiến tới 0, còn trường hợp thứ hai gọi là chế độ véc-tơ, trong chế độ này giá trị của y được điều khiển về 0.

- Giá trị σ_k là các hằng số được lưu trữ trong bảng và nó tùy thuộc vào giá trị của m. Với $m = 1,\ \sigma_k = \tan^{-1}2^{-k}$ với $m = 0,\ \sigma_k = 2^{-k}$ và với $m = -1,\ \sigma_k = \tanh^{-1}2^{-k}$.

Để sử dụng các công thức này $x_1,\ y_1$ và z_1 cần được gán các giá trị ban đầu phù hợp. Một trong các đầu vào này, ví dụ như z_1, có thể là giá trị của sin hyperbol mà chúng ta cần xấp xỉ, $\sinh(z_1)$. Trong tất cả các trường hợp các giá trị bắt đầu cần phải được hạn chế tới một khoảng xác định xung quanh điểm bắt đầu để đảm bảo tính hội tụ. Khi các vòng lặp được thực hiện, một trong các biến có xu hướng tiến đến không trong khi đó các biến khác sẽ xấp xỉ đến giá trị cần tính toán.

Nhược điểm chính của thuật toán CORDIC là nó có tính hội tụ tuyến tính dẫn đến thời gian thực hiện cũng tỷ lệ tuyến tính với số bit của toán hạng. Nói một cách khác, thuật toán CORDIC dựa trên các kiến trúc lặp nên độ giữ chậm lớn, nhất là khi các toán hạng đầu vào có độ rộng bit lớn. Ngoài ra CORDIC cũng có một hạn chế nữa là nó chỉ tính toán được một số các hàm toán học nhất định.

1.2.4. Xấp xỉ hữu tỷ

Phương pháp xấp xỉ hữu tỷ là một phương pháp hiệu quả trong ước lượng các hàm giải tích. Phương pháp này thực hiện xấp xỉ một hàm $f(x)$ bởi một hàm hữu tỷ $R_{mn}(x)$. Hàm xấp xỉ $R_{mn}(x)$ là một hàm được biểu diễn bởi tỷ số của hai đa thức $P_m(x)$ và $Q_n(x)$ có bậc tương ứng là m và n như công thức (1.16).

$$f(x) \approx R_{mn}(x) = \frac{P_m(x)}{Q_n(x)} = \frac{p_m x^m + p_{m-1}x^{m-1} + \ ... \ + p_1 x + p_0}{q_n x^n + q_{n-1}x^{n-1} + ... + q_1 x + q_0}. \quad (1.16)$$

Nếu các đa thức $P_m(x)$ và $Q_n(x)$ là các đa thức có bậc tương ứng là m và n thì đa thức hữu tỷ $R_{mn}(x)$ được tạo bởi tỷ số giữa chúng có thể đạt được độ chính xác tương tự với trường hợp xấp xỉ bằng đa thức có bậc là $m + n$. Xấp xỉ đa thức là một trường hợp đặc biệt của xấp xỉ hữu tỷ với $P_m(x) = R_{m0}(x)$. Ưu điểm của xấp xỉ hữu tỷ so với xấp xỉ đa thức (với cùng độ chính xác yêu cầu) là bậc các đa thức sẽ nhỏ hơn và do đó đòi hỏi ít hơn các thao tác cộng và nhân. Hơn nữa, xấp xỉ hữu tỷ cho phép thực hiện các đa thức tử số và mẫu số đồng thời nên sẽ cải thiện được hiệu năng tính toán. Tuy nhiên nhược điểm của phương pháp xấp xỉ hữu tỷ là yêu cầu phải thực hiện các thao tác chia và do đó nó gây ra độ giữ chậm lớn.

Xấp xỉ hữu tỷ thường được sử dụng để thay thế cho phương pháp xấp xỉ đa thức trong các trường hợp các hàm cần xấp xỉ chứa các điểm cực như $\tan(x)$ hoặc các đường tiệm cận như $\tan^{-1}(x)$. Nhìn chung, xấp xỉ hữu tỷ là phương pháp hiệu quả để tính toán các hàm trong các bộ vi xử lý. Tuy nhiên nó không phù hợp trong thực thi các hàm trên FPGA do đòi hỏi các bộ chia. Thực thi phần cứng của xấp xỉ hữa tỷ đã được nghiên cứu trong [22].

1.3. Phương pháp thực thi các hàm toán học sử dụng trong luận án

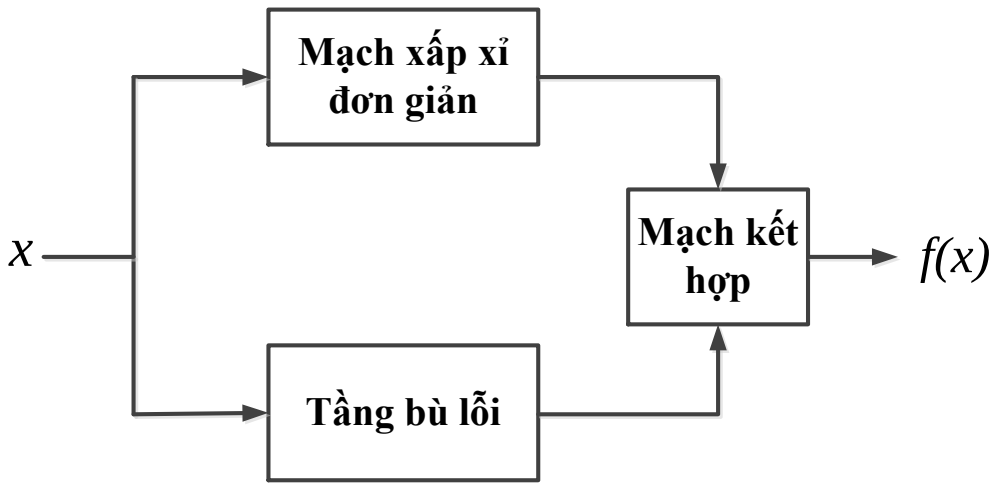
1.3.1. Phương pháp chung thiết kế các lõi phần cứng tính toán

Mục tiêu của luận án đặt ra là thiết kế các lõi phần cứng thực thi một số hàm toán học điển hình ứng dụng trong DSP. Để dung hòa giữa độ phức tạp của phần cứng và hiệu năng tính toán (độ chính xác, tốc độ thực hiện...) phương pháp đưa ra trong luận án là thực hiện tính toán các hàm toán học dựa trên xấp xỉ các hàm phức tạp bằng một hàm đơn giản hơn và dễ thực

thi bằng phần cứng đồng thời sử dụng một tầng bù sai số do xấp xỉ.

Về tổng quát, các hàm xấp xỉ thường là các hàm đa thức với số bậc đa thức xác định. Xấp xỉ đa thức bậc cao cho phép đạt được độ chính xác tốt. Tuy nhiên, khi sử dụng xấp xỉ bằng các đa thức bậc cao sẽ làm tăng độ phức tạp phần cứng cũng như độ trễ tính toán. Phương pháp xấp xỉ tuyến tính cho phép kiến trúc thực hiện đơn giản hơn, tuy nhiên sai số xấp xỉ sẽ lớn hơn so với việc sử dụng các hàm xấp xỉ bậc cao.

Với mục đích giảm độ phức tạp phần cứng của các lõi tính toán các hàm toán học và với mức sai số có thể chấp nhận của các ứng dụng DSP, trong luận án này sử dụng phương pháp xấp xỉ tuyến tính phân đoạn đều. Sau đó, việc bù sai số xấp xỉ có thể thực hiện bằng cách sử dụng một LUT có kích thước nhỏ hoặc tiếp tục sử dụng xấp xỉ tuyến tính các hàm sai số của xấp xỉ ban đầu có lợi dụng các đặc trưng của hàm sai số để giảm chi phí phần cứng và tăng tốc độ. Hình 1.4 mô tả phương pháp thực hiện trong luận án này.



Hình 1.4: Phương pháp độ chênh lệch.

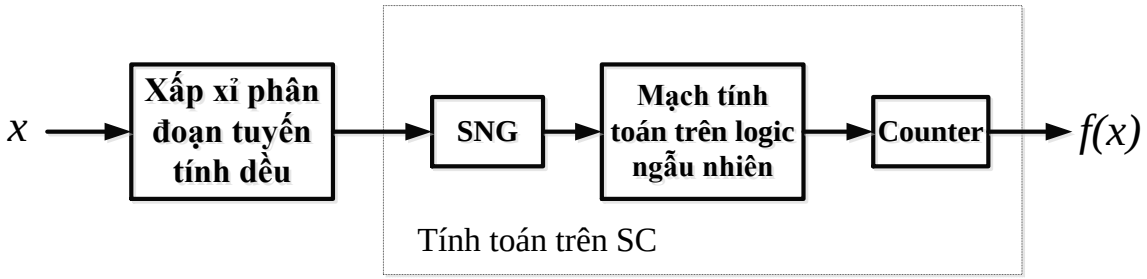
Từ đó có thể thấy, phương pháp đưa ra cần phải giải quyết các vấn đề cơ bản sau:

- Cần có các thuật toán để tìm các hàm xấp xỉ phân đoạn tuyến tính tối ưu theo tiêu chí đơn giản về thực thi nhằm giảm độ phức tạp phần cứng.
- Có các thuật toán để đảm bảo kích thước bảng LUT là tối ưu nhằm đơn giản phần cứng và tăng tốc độ tính toán, giảm công suất tiêu thụ.
- Khảo sát các đặc trưng của các hàm lỗi gây ra bởi xấp xỉ ban đầu để có thể lợi dụng cho tầng xấp xỉ bù lỗi nhằm đạt được kiến trúc hiệu quả.

Như vậy, phương pháp thực hiện của luận án là dựa trên kỹ thuật xấp xỉ các hàm toán học bằng các hàm tuyến tính phân đoạn. Yêu cầu đặt ra là khi xấp xỉ các hàm toán học bằng các hàm tuyến tính phân đoạn là tìm các hệ số của hàm xấp xỉ sao cho dung hòa về các tiêu chí lỗi xấp xỉ và chi phí phần cứng thực thi. Hơn nữa, do sử dụng bảng LUT để giảm giảm sai số, nên việc tính toán dựa trên LUT cũng được xem xét nghiên cứu.

Ngoài ra, lý thuyết về tính toán ngẫu nhiên (SC: Stochastic Computing) cho phép ứng dụng để thực thi các đơn vị số học với chi phí rất thấp và lỗi chấp nhận được. Đặc tính cơ bản của SC là một số được biểu diễn bởi một chuỗi bit và được xử lý bởi các mạch rất đơn giản, ví dụ như phép nhân trong SC được thực hiện chỉ bởi một cổng AND. Trong luận án này, còn sử dụng phương pháp thực thi các hàm toán học dựa trên xấp xỉ hàm phức tạp bằng các hàm tuyến tính phân đoạn đều. Sau đó, các hàm xấp xỉ sẽ được thực thi dựa trên logic ngẫu nhiên. Hình 1.5 mô tả phương pháp thực thi các hàm dựa trên kết hợp xấp xỉ phân đoạn tuyến tính đều và sử dụng SC. Trong đó, hàm cần tính toán được xấp xỉ bằng các hàm tuyến tính phân đoạn. Sau đó, được chuyển đổi sang các số ngẫu nhiên bằng mạch tạo số ngẫu nhiên (SNG: Stochastic Number Generator) để thực hiện các thao tác tính toán trên các

số ngẫu nhiên. Kết quả sau đó được chuyển đổi về các số nhị phân truyền thống bằng một bộ đếm để tạo ra kết quả cuối cùng.



Hình 1.5: Phương pháp tính toán sử dụng SC.

1.3.2. Phương pháp xấp xỉ tuyến tính phân đoạn đều

Trong phương pháp xấp xỉ phân đoạn tuyến tính đều, hàm cần tính toán $f(x)$ với $x \in (\alpha, \beta)$ được xấp xỉ phân đoạn tuyến tính đều. Trong đó, khoảng giá trị $x \in (\alpha, \beta)$ được chia thành s phân đoạn đều nhau và mỗi phân đoạn sẽ được xấp xỉ bởi một hàm tuyến tính. Để đơn giản về cấu trúc phần cứng thì s được lựa chọn là một số lũy thừa của 2 ($s = 1,\ 2,\ 4\ ...\ 2^k$). Trong phân đoạn thứ i hàm xấp xỉ có thể được viết như công thức (1.16).

$$f(x) \approx a_i x + b_i, \qquad \frac{i}{s}(\beta - \alpha) \leq x < \frac{i+1}{s}(\beta - \alpha), \ \ i = 0 \div s - 1. \tag{1.17}$$

Lỗi xấp xỉ trên phân đoạn thứ i là:

$$\varepsilon_i(x) = f(x) - (a_i x + b_i), \qquad \frac{i}{s}(\beta - \alpha) \leq x < \frac{i+1}{s}(\beta - \alpha); \ \ i = 0 \div s - 1. \tag{1.18}$$

Với mục đích đạt được hàm xấp xỉ với độ chính xác tốt nhất có thể, trong mỗi phân đoạn thứ i các hệ số a_i và b_i được tối ưu sao cho lỗi xấp xỉ $\varepsilon_i(x)$ là cực tiểu. Thuật toán tìm các hệ số tối ưu được mô tả như Bảng 1.2

Bảng 1.2: Thuật toán tìm hệ số tối ưu trong từng phân đoạn.

1:	*BEGIN* : Khởi tạo mảng giá trị a Khởi tạo mảng giá trị b Khởi tạo đoạn giá trị i
2:	for $p = 1 : length(a)$ for $q = 1 : length(b)$ for $k = 1 : length(x)$ tính $\varepsilon_i(x)$ theo công thức (1.17) end end end
3:	for $p = 1 : length(a)$ for $q = 1 : length(b)$ tính $\varepsilon_{\mathrm{i\,max}} = \max(\varepsilon_{\mathrm{i}}(x))$ end end
4:	Tính $\min(\varepsilon_{\mathrm{i\,max}})$
5:	*END* Xuất giá trị a, b tương ứng với $\min(\varepsilon_{\mathrm{i\,max}})$

Tùy thuộc vào hàm $f(x)$ cụ thể, cũng như các các yêu cầu cụ thể về độ chính xác và độ phức tạp của phần cứng mà các hệ số a_i và b_i sẽ được ràng buộc theo những điều kiện cụ thể. Thông thường để tránh việc sử dụng mạch nhân trong kiến trúc thì các hệ số a_i thường được ràng buộc dưới dạng một số lũy thừa của hai hoặc là tổng của các số lũy thừa của hai. Khi đó mạch nhân sẽ được thay thế bởi các thao tác dịch.

Ví dụ thực hiện thuật toán trên bằng một chương trình Matlab để xấp xỉ hàm $\sin x$ với x thuộc khoảng (0, 1) bằng 4 phân đoạn đều, trong đó các hệ số a_i và b_i được ràng buộc có dạng như công thức (1.18). Khi đó kết quả các hệ số cho từng phân đoạn như trên Bảng 1.3. Trong Bảng 1.3, các giá trị $\varepsilon_{i\,\max}$ là sai số xấp xỉ cực đại của phân đoạn thứ i. Giá trị $\varepsilon_{\max}$ là giá trị sai

số cực đại trên tất cả các phân đoạn, $\varepsilon_{\max} = \max_{\forall i}\{\varepsilon_{i\,\max}\}$.

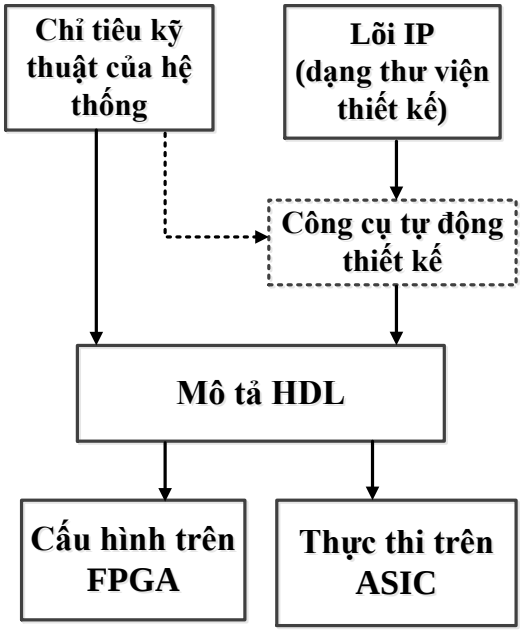
$$\begin{aligned} a_i &= \sum_{j=0}^{5} p.2^j, \qquad p = \{0,1\}; \\ \mathrm{b}_i &= K.2^{-8}, \qquad K \in N. \end{aligned} \tag{1.19}$$

Bảng 1.3: Các hệ số cho xấp xỉ tuyến tính hàm $\sin x$ bằng 4 phân đoạn.

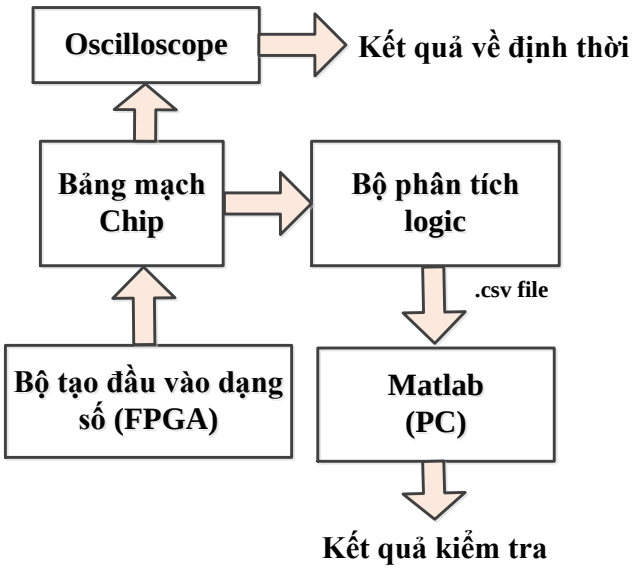
Phân đoạn	a_i	b_i	$\varepsilon_{i\,\max}$
$0 \le x < 0,25$	2^0	0	0,0026
$0,25 \le x < 0,5$	$2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$	3.2^{-8}	0,0031
$0,5 \le x < 0,75$	$2^{-1} + 2^{-2} + 2^{-4}$	19.2^{-8}	0,0031
$0,75 \le x < 1$	$2^{-1} + 2^{-3}$	56.2^{-8}	0,0059
$\varepsilon_{\max} = 0,0059$			

1.3.3. Qui trình thiết kế và kiểm chứng các lõi phần cứng tính toán

Hình 1.6 mô tả đề xuất một qui trình thiết kế tổng quát thực hiện xấp xỉ hóa các hàm phức tạp trong DSP sử dụng thư viện lõi IP cứng tính toán các hàm phức tạp cho các hệ thống DSP theo phương pháp và thuật toán tối ưu tham số thiết kế như đã trình bày ở trên. Bằng việc sử dụng một thư viện tạo ra các lõi IP tối ưu cùng với công cụ tự động sinh mã nguồn mô tả phần cứng (HDL) cho các hàm này, việc thực thi các phép tính toán này sẽ được tăng tốc với lượng tài nguyên phần cứng phát sinh là tối thiểu. Một công cụ phần mềm hỗ trợ thiết kế tự động được phát triển để tạo ra mô tả HDL của các khối tính toán các hàm phức tạp trên DSP, cùng với các chỉ tiêu kỹ thuật của hệ thống cho phép xuất ra mô tả HDL cho toàn hệ thống. Từ mô tả HDL này, lõi IP có thể được sử dụng để cấu hình cho thiết bị khả trình FPGA hay hiện thực hóa trên thư viện chuẩn ASIC.



Hình 1.6: Qui trình tạo lõi IP tính toán.



Hình 1.7: Qui trình kiểm chứng lõi IP tính toán.

Hình 1.7 trình bày qui trình kiểm chứng (verification) các lõi IP thực hiện tính toán hàm phức tạp ứng dụng trong DSP. Các mẫu hình (pattern) đầu vào được tạo ra trên FPGA, đưa tới mạch kiểm tra của chip ASIC. Kết quả thực hiện trên lõi IP sẽ được đưa tới máy hiện sóng (oscilloscope) để kiểm

tra một số tham số định thời (timing) như độ trễ tính toán chẳng hạn và bộ phân tích logic (logic analyzer) để kiểm tra chức năng, có thể kết hợp với phần mềm Matlab trên máy tính thông qua file đầu ra dạng .csv của bộ phân tích logic.

1.4. Kết luận chương 1

Chương này đã trình bày khái quát các vấn đề cơ bản liên quan đến các giải pháp đề xuất trong luận án bao gồm việc sử dụng các hàm toán học trong các ứng dụng xử lý tín hiệu số, các phương pháp điển hình trong thiết kế phần cứng tính toán các hàm toán học. Đồng thời cũng trình bày phương pháp chung sẽ được sử dụng trong các đề xuất để thiết kế phần cứng tính toán các hàm toán học trong luận án.

Chương 2
THIẾT KẾ PHẦN CỨNG TÍNH TOÁN HÀM LOGARITHM NHỊ PHÂN

2.1. Cơ sở và động lực nghiên cứu

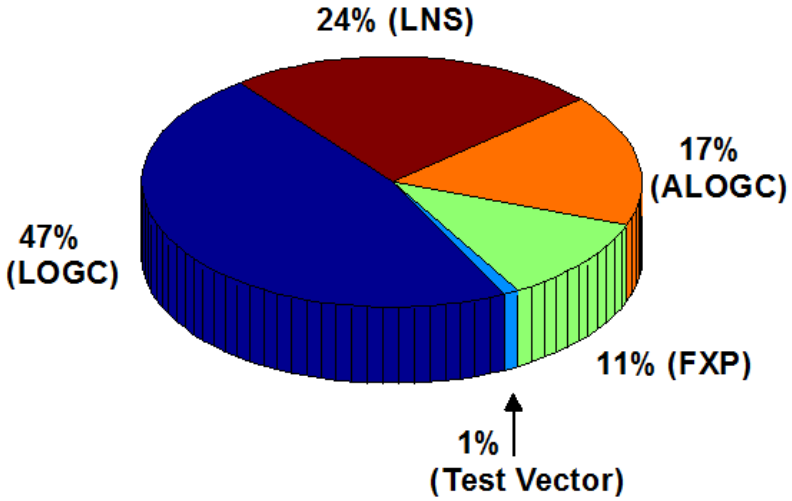
Sự phát triển của công nghệ mạch tích hợp đã trở thành động lực chính cho xử lý tín hiệu số thông qua các bộ xử lý tín hiệu số. Thông thường, quá trình xử lý tín hiệu số đòi hỏi một số lượng lớn các thao tác số học phức tạp như các phép nhân, chia, khai căn, lũy thừa. Ví dụ trong [1] chỉ ra là 86,5% thời gian xử lý trong hệ thống xử lý đồ họa ba chiều thời gian thực là để thực hiện các phép nhân và chia. Phần cứng thực thi các thao tác phức tạp này thường tiêu tốn lượng lớn tài nguyên và công suất, do đó dẫn đến độ giữ chậm lớn. Điều này là một thách thức đặt ra cho các bộ xử lý tín hiệu số, nhất là các ứng dụng trong các thiết bị di động và cầm tay. Các thiết bị này thường đòi hỏi xử lý thời gian thực, hạn chế về kích thước cũng như dung lượng bộ nhớ và công suất tiêu thụ.

Hệ thống số logarithm (LNS: Logarithmic number system) được đề xuất như là một sự thay thế cho các hệ thống số dấu phẩy tĩnh và dấu phẩy động để đơn giản các thao tác số học phức tạp [23]. Chẳng hạn, các phép nhân và chia có thể được đơn giản tương ứng thành các phép cộng và trừ, trong khi các phép tính căn bậc hai và lũy thừa được đơn giản tương ứng thành các phép dịch và nhân. Vì vậy, các mạch phần cứng cũng sẽ được đơn giản hơn.

Ngoài ra, LNS vốn đã có các đặc tính đơn giản của hệ thống số dấu phẩy tĩnh và đặc tính khoảng động lớn của hệ thống số dấu phẩy động. Tuy nhiên các phép cộng và trừ trong LNS là các hàm phi tuyến. Nhiều nghiên cứu đã được thực hiện để nhằm đơn giản phần cứng thực thi các hàm cộng/trừ logarithm [24–28]. Để đơn giản hơn, các phương pháp lai đã được đề xuất để kết hợp các thao tác phức tạp trong LNS với các phép cộng/trừ đơn giản trong hệ thống số dấu phẩy động/dấu phẩy tĩnh [2, 29–31].

Hệ thống LNS hứa hẹn là phù hợp với các ứng dụng công suất thấp. Tuy nhiên, các thao tác cộng và trừ trong LNS sẽ phức tạp hơn, nhược điểm này dẫn đến hạn chế khả năng sử dụng rộng rãi LNS. Vì vậy một giải pháp đặt ra là kết hợp LNS với hệ thống số nhị phân truyền thống gọi là hệ thống số lai (HNS: Hybrid Number System) trong đó đảm bảo được ưu điểm của hai hệ thống số này. Trong HNS các bộ chuyển đổi logarithm nhị phân (LOGC: Logarithm Converters) và các bộ chuyển đổi logarithm ngược (ALOGC: Anti-Logarithmic Converters) là thành phần không thể thiếu. Trong thực thi các bộ xử lý ảnh 3D tài nguyên chiếm của các LOGC và ALOGC chiếm khoảng 64% tài nguyên tổng cộng của bộ xử lý như thể hiện ở Hình 2.1 [31]. Vì vậy, việc giảm độ phức tạp của các bộ chuyển đổi LOGC và ALOGC sẽ dẫn tới giảm đáng kể tài nguyên chung của các bộ xử lý tín hiệu số sử dụng HNS. Hơn nữa, trong nhiều ứng dụng xử lý tín hiệu số thời gian thực, chẳng hạn như các ứng dụng trong các hệ thống truyền thông không dây thì các bộ chuyển đổi logarithm và logarithm ngược có hiệu năng cao, tiêu tốn ít tài nguyên và công suất thấp là rất cần thiết.

LNS yêu cầu phải chuyển đổi dữ liệu đầu vào sang miền logarithm, có nghĩa là phải tính toán logarithm của dữ liệu đầu vào, sử dụng các bộ chuyển



Hình 2.1: Phân bố tài nguyên phần cứng cho bộ xử lý HNS.

đổi logarithm đòi hỏi phí tổn thêm phần cứng. Về mặt lý thuyết, logarithm của một số có độ chính xác là vô hạn. Tuy nhiên, trong thực tế logarithm của dữ liệu đầu vào được tính toán với độ chính xác hữu hạn tùy thuộc vào ứng dụng. Nói chung, các chuyển đổi logarithm có độ chính xác cao đòi hỏi các bộ chuyển đổi phức tạp và làm tăng chi phí về phần cứng.

Một số phương thức khác nhau đã được để xuất cho việc thực thi phần cứng các bộ chuyển đổi logarithm. Các phương pháp này có thể phân thành ba loại chính: 1) các phương pháp dựa trên phương pháp Mitchell; 2) các phương pháp dựa trên LUT và 3) các phương pháp truy toán ký số (Digit Recurrence Methods).

Để thực thi một bộ chuyển đổi logarithm tốc độ cao và đơn giản về phần cứng, Mitchell trong [32] đã sử dụng xấp xỉ đường cong logarithm bởi một đoạn tuyến tính đơn giản. Phương pháp xấp xỉ của Mitchell có lỗi xấp xỉ lớn và do đó nhiều nghiên cứu sau này đã được đề xuất để cải thiện độ chính xác và dung hòa giữa tài nguyên, tốc độ và/hoặc công suất tiêu thụ. Các phương

pháp này nhìn chung là chia khoảng giá trị đầu vào thành một số khoảng nhỏ hơn (thường 2, 3, 4, 6 và 8 khoảng) và xấp xỉ mỗi khoảng giá trị đó bằng các hàm tuyến tính và do đó đạt được lỗi xấp xỉ nhỏ hơn [33–41].

Các phương pháp dựa trên phương pháp của Mitchell là rất phù hợp với các ứng dụng đòi hỏi độ phức tạp phần cứng thấp, công suất nhỏ và tốc độ cao mà không đòi hỏi độ chính xác cao. Đồ họa 3D [37], phân tích độ hội tụ của phương pháp lặp Jacobi [29], chuyển đổi cosin rời rạc và chuyển đổi cosin rời rạc ngược cho nén ảnh [30], là các ví dụ điển hình cho việc sử dụng các bộ tính toán logarithm mà trong đó không đòi hỏi cao về độ chính xác và cho phép đạt hiệu quả về công suất, tài nguyên và tốc độ.

Phương pháp truy toán ký số [42–46] có thể đạt được độ chính xác cao và hiệu quả về tài nguyên phần cứng. Tuy nhiên, phương pháp này dựa trên các kiến trúc lặp nên thường có độ trễ tính toán lớn và do đó không phù hợp với các ứng dụng đòi hỏi tốc độ tính toán cao. Phương pháp tính toán dựa trên LUT đạt được độ chính xác và tốc độ tính toán cao. Tuy nhiên nó đòi hỏi dung lượng bộ nhớ lớn nhất là khi độ rộng dữ liệu đầu vào có kích thước lớn, dẫn đến tiêu tốn về mặt tài nguyên và công suất tiêu thụ.

Với một độ chính xác đã cho, sẽ là lý tưởng nếu chuyển đổi logarithm được thực thi với tối thiểu về phần cứng và tốc độ cực đại. Điều này là rất quan trọng với nhiều ứng dụng nhất là các bộ xử lý đồ họa 3D (ví dụ như bộ xử lý đồ họa NVIDIA® Tesla® GPUs [47]) trong đó có hàng trăm thậm chí là hàng nghìn lõi, mỗi lõi chứa hơn một ALU thực hiện các thao tác trong LNS sử dụng chuyển đổi logarithm dựa trên phương pháp của Mitchell như chỉ ra trong [37]. Vì vậy, một cải thiện nhỏ về độ phức tạp phần cứng và độ giữ chậm cũng dẫn đến cải thiện đáng kể hiệu năng.

Các bộ chuyển đổi dựa trên phương pháp của Mitchell trong đó sử dụng phương pháp xấp xỉ tuyến tính với tối ưu lỗi xấp xỉ nhìn chung là đạt được độ chính xác không cao, nó phù hợp với các ứng dụng đòi hỏi độ phức tạp phần cứng thấp và tốc độ cao. Để đạt được tốc độ chuyển đổi nhanh và độ chính xác cao thì phương pháp sử dụng bảng là một giải pháp phù hợp. Tuy nhiên, phương pháp này có một hạn chế đáng kể là đòi hỏi dung lượng bộ nhớ lớn nhất là khi độ dài từ dữ liệu đầu vào lớn. Điều này dẫn đến làm tăng độ phức tạp phần cứng, công suất tiêu thụ lớn và khó khăn trong sử dụng các công cụ tổng hợp. Một giải pháp hứa hẹn cải thiện được độ chính xác mà vẫn đảm bảo được một chi phí phần cứng nhỏ là sử dụng các xấp xỉ tuyến tính dựa trên phương pháp của Mitchell kết hợp với một bảng LUT nhỏ để bù lỗi nhằm tăng độ chính xác.

2.2. Các nghiên cứu liên quan

Các phương pháp chuyển đổi logarithm nhị phân dựa trên phương pháp của Mitchell có thể đạt được kiến trúc phần cứng của các bộ chuyển đổi có độ phức tạp thấp và tốc độ chuyển đổi nhanh, nó phù hợp với các ứng dụng đòi hỏi hiệu quả về mặt tài nguyên, hiệu năng tính toán và không yêu cầu cao về độ chính xác. Mục này sẽ trình bày phương pháp chuyển đổi logarithm của Mitchell và các nghiên cứu sau đó dựa trên phương pháp đề xuất của Mitchell nhằm cải thiện độ chính xác.

2.2.1. Phương pháp Mitchell

Mitchell trong [32] đã đề xuất một phương pháp xấp xỉ cho tính toán hàm logarithm của một số nhị phân như sau: Về mặt toán học một số nhị phân không dấu N trong khoảng $0 \leq N \leq (2^n - 1)2^j$, ở đây $j, k \in, \mathbb{Z}, n \in, \mathbb{N}$,

$j \leq k \leq n-1+j$ và $z_i \in \{0,1\}$ có thể biểu diễn như sau:

$$N = z_k ... z_2 z_3 z_0 . z_{-1} z_{-2} z_{-3} ... z_j = \sum_{i=j}^{k} 2^i z_i \tag{2.1}$$

Không mất tính tổng quát, có thể coi như $z_k = 1$ vì nó là bit '1' đầu tiên và công thức (2.1) có thể được viết lại như sau:

$$N = 2^k + \sum_{i=j}^{k-1} 2^i z_i = 2^k (1 + \sum_{i=j}^{k-1} 2^{i-k} z_i) = 2^k (1+x), \tag{2.2}$$

ở đây, $x = \sum_{i=j}^{k-1} 2^{i-k} z_i$ trong khoảng $[0,1)$. Như vậy, logarithm cơ số 2 của N sẽ là:

$$\log_2(N) = k + \log_2(1+x), \tag{2.3}$$

trong đó $k = \lfloor \log_2(N) \rfloor$ và $\log_2(1+x)$ tương ứng là phần nguyên và phần thập phân của $\log_2(N)$. Giá trị của k có thể nhận được bằng việc sử dụng một bộ phát hiện bit '1' đầu tiên (LOD: Leading-One Detector) trong dữ liệu đầu vào. Vì vậy, tính toán $\log_2(N)$ sẽ tùy thuộc vào tính toán $\log_2(1+x)$. Trong phương pháp của mình, Mitchell đã tính $\log_2(1+x)$ bằng xấp xỉ đơn giản như sau:

$$\log_2(N) \approx k + LA_{1.Mitchell}(1+x) = k + x \tag{2.4}$$

Phương pháp xấp xỉ của Mitchell cho phép thực thi đơn giản về phần cứng. Tuy nhiên, độ chính xác của xấp xỉ này là thấp và do vậy nhiều nghiên cứu đã tập trung để đề xuất các giải pháp nhằm tăng độ chính xác.

2.2.2. Các phương pháp tính toán hàm logarithm dựa trên phương pháp Mitchell

Để cải thiện độ chính xác, một số nghiên cứu đã đề xuất các giải pháp khác nhau cho xấp xỉ hàm $\log_2(1+x)$. Nhìn chung, các phương pháp này

thực hiện xấp xỉ hàm $\log_2(1+x)$ bằng cách chia khoảng $[0,1)$ của x thành một số đoạn, trên mỗi đoạn sử dụng một hàm xấp xỉ. Tổng quát, các hàm xấp xỉ có thể mô tả như sau:

$$\begin{aligned}\log_2(1+x) &\cong LA_S(1+x)\\ &= a_s \times x + b_s\\ &= \begin{cases} x + c_s \times x_{MSBh} + d_s, & a_s \geq 1\\ x + c_s \times \bar{x}_{MSBh} + d_s, & a_s < 1, \end{cases}\end{aligned} \tag{2.5}$$

ở đây, x_{MSBh} là ký hiệu của h bit có trọng số lớn nhất của x, $\bar{x}_{MSBh}$ là số bù 1 của x_{MSBh}. Các ký hiệu a_s, b_s, c_s và d_s là các hệ số của hàm xấp xỉ tại đoạn thứ s, S là số đoạn xấp xỉ. Trong các phương pháp trên, lỗi tuyệt đối cực đại (MAE: Maximum Absolute Error) và phần trăm lỗi tuyệt đối cực đại (MAPE: Maximum Absolute Percentage Error) được ký hiệu như công thức (2.6) và (2.7). Các giải pháp đưa ra chủ yếu nhằm tìm các hệ số của hàm xấp xỉ theo tiêu chí tối thiểu MAE và MAEP.

$$\text{MAE} = \max_{\forall x}\{|\log_2(1+x) - LA_S(1+x)|\} \tag{2.6}$$

$$\text{MAEP} = \max_{\forall N}\left\{\frac{|\log_2(1+x) - LA_S(1+x)|}{\log_2(N)} \times 100\%\right\} \tag{2.7}$$

Combet và các cộng sự trong [33] chia khoảng giá trị của x thành 4 phân đoạn đều và các hệ số của hàm xấp xỉ được xác định bằng phương pháp "thử và lỗi". Hall và các cộng sự trong [34] đề xuất một phương pháp xấp xỉ tuyến tính phân đoạn với 1, 2, 4 và 8 phân đoạn đều và việc tìm các hệ số hàm xấp xỉ theo phương pháp trung bình bình phương lỗi. Dựa trên phân tích sự khác biệt của giá trị chính xác hàm logarithm và xấp xỉ của Mitchell, SanGregory và các cộng sự trong [35] chia khoảng giá trị x thành 2 phân đoạn và đề xuất

chỉ sử dụng 4 MSBs của x cho công thức hàm xấp xỉ. Nhằm đạt được chi phí phần cứng thấp, Abed và Siferd trong [36] đề xuất xấp xỉ phân đoạn tuyến tính với 2, 3 và 6 phân đoạn. Việc xác định các hệ số thông qua mô phỏng phần mềm. Trong đó, 3, 4 và 6 MSBs của x được sử dụng tương ứng với các trường hợp 2, 3 và 6 phân đoạn. Kim và các cộng sự trong [37] chia khoảng của x thành 8 phân đoạn và sử dụng 25 MSBs trong công thức xấp xỉ. Tương tự như phương pháp ban đầu của Mitchell, Li và các cộng sự trong [38] chỉ xấp xỉ hàm $\log_2(1+x)$ bằng một đoạn thẳng nhưng đường xấp xỉ được dịch chuyển theo chiều thẳng đứng lên một giá trị so cho lỗi cực đại trong xấp xỉ của Mitchell giảm đi một nửa. Bằng các phân tích về hình học của mối liên hệ giữa đường cong $\log_2(1+x)$ với một hình bình hành, Juang và các cộng sự trong [39] đã đề xuất một xấp xỉ hai phân đoạn và chỉ sử dụng 4 MSBs cho phương trình sửa lỗi. Một xấp xỉ với 4 phân đoạn đã được đề xuất bởi Gutierrez và Valls trong [40], trong đó sử dụng 8 MSBs cho biểu diễn xấp xỉ. De Caro và các cộng sự trong [41] đã đề xuất một xấp xỉ 2,4 và 8 phân đoạn dựa trên tiêu chí tối thiểu phần trăm lỗi xấp xỉ. V.P. Hoang và C. K. Pham trong [48] đã thực hiện xấp xỉ hàm $\log_2(1+x)$ theo phương pháp cận đối xứng. Trong đó, xấp xỉ hàm lỗi của Mitchell bằng hai cặp đoạn đối xứng nhằm đạt được kiến trúc phần cứng có độ phức tạp thấp.

2.2.3. Khái quát về độ chính xác

Bảng 2.1 thể hiện độ chính xác của các phương pháp chuyển đổi logarithm nhị phân dựa trên phương pháp của Mitchell. Với mỗi phương pháp các tham số gồm: số phân đoạn; lỗi dương cực đại (MPE: Maximum Positive Error); lỗi âm cực đại (MNE: Maximum Negative Error); khoảng lỗi

Bảng 2.1: Độ chính xác của các phương pháp dựa trên phương thức Mitchell.

Phương pháp	Số đoạn	MPE ($\times 10^{-3}$)	MNE ($\times 10^{-3}$)	ER ($\times 10^{-3}$)	$MPEP$ (%)	$MNEP$ (%)	EPR (%)
Mitchell [32]	1	86,1	0	86,1	5,361	0	5,361
Combet [33]	4	8,0	-6,2	14,2	0,185	-0,267	0,452
Hall [34]	4	8,1	-2,3	10,4	0,126	-0,781	0,907
SanGregory [35]	4	29,3	-28,0	57,3	0,431	-1,540	1,917
Abed [36]	2	44.9	-18,3	63,2	0,930	-0,554	1,484
	3	29,3	-20,8	50,1	0,431	-0,268	0,699
	6	13,2	-13,0	26,2	0,153	-0,154	0,307
Kim [37]	8	2,0	-1,5	3,5	0,050	-0,044	0,094
Li [38]	1	43,0	-43,0	86,0	2,645	-4,304	6.949
Juang [39]	2	37,0	-9,8	46,8	1,079	-0,096	1,175
Gutierzet [40]	4	7,2	-7,9	15,1	0,181	-0,793	0,974
De Caro [41]	2	22,9	-28,6	51,5	0,582	-0,498	1,080
	4	9.8	-6,8	16,6	0,183	-0,431	0,603
	8	2,8	-2,5	5,3	0,032	-0,031	0,063
V. P. Hoang [48]	4	7,2	-7,9	15,1	0,181	-0,793	0,974

(ER: Error Range); phần trăm lỗi dương cực đại (MPEP: Maximum Positive Error Percentage) và phần trăm lỗi âm cực đại (MNEP: Maximum Negative Error Percentage). Trong các phương pháp kể trên, phương pháp của Kim và các cộng sự trong [37] đạt được MAE $= 0,002$ là tốt nhất (MAE $=$ max(|MPE|, |MNE|), trong khi phương pháp của De Caro và các cộng sự trong [41] đạt được MAEP $= 0,032\%$ là tốt nhất.

2.3. Đề xuất phương pháp xấp xỉ hàm logarithm

Với mục đích thiết kế tạo một bộ chuyển đổi logarithm tốc độ cao và đơn giản về phần cứng, luận án đề xuất một phương pháp xấp xỉ hàm logarithm trên cơ sở phương pháp của Mitchell. Trong đó, hàm $\log_2(1+x)$ được xấp xỉ bằng phương pháp phân đoạn tuyến tính. Để đơn giản về mặt cấu trúc phần cứng thì số đoạn được chia là một số lũy thừa của 2 và các phân đoạn là đều

nhau. Số phân đoạn lớn sẽ tăng độ chính xác tuy nhiên cũng làm tăng độ phức tạp phần cứng. Vì vậy, để dung hòa giữa độ phức tạp phần cứng và độ chính xác trong phương pháp đề xuất sẽ sử dụng xấp xỉ với bốn phân đoạn $(S = 4)$. Hàm xấp xỉ cho phân đoạn thứ i có thể viết như sau:

$$\log_2(1 + x) \approx a(x) = a_i x + b_i, \quad i = 0 \div 3. \tag{2.8}$$

Để đơn giản hơn nữa trong cấu trúc phần cứng, các hệ số a_i và b_i trong phương pháp đề xuất được ràng buộc theo dạng biểu diễn như công thức (2.9).

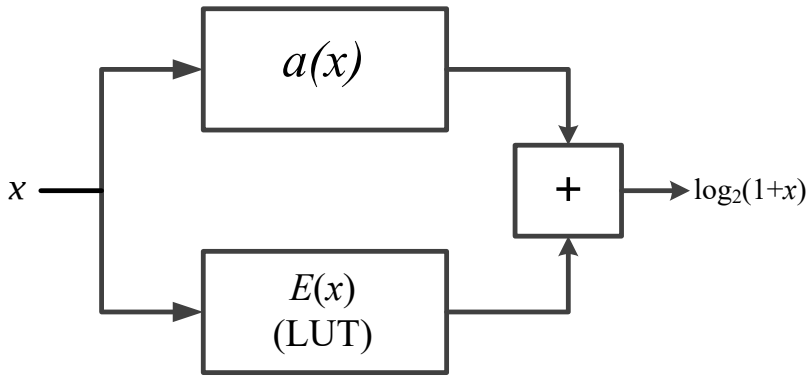
$$\begin{cases} a_i = \sum_{k=1}^{M} p_k 2^{-n}, \;\; p_k \in \{-1, 0, 1\} \\ b_i = B_i 2^{-K}, \;\; B_i, K \in Z \end{cases} \tag{2.9}$$

Sai số do xấp xỉ của phân đoạn thứ i sẽ là:

$$E(x) = \log_2(1 + x) - (a_i x + b_i) \tag{2.10}$$

Để giảm lỗi một bảng LUT được sử dụng để chứa các giá trị độ lệch để bù lại sai số xảy ra do xấp xỉ. Hình 2.2 mô tả phương pháp xấp xỉ đề xuất. Trong đó, các giá trị đầu vào x sẽ tương ứng cho ra các giá trị của hàm xấp xỉ $a(x)$ và các giá trị độ lệch giữa hàm xấp xỉ và giá trị chính xác của hàm $\log_2(1 + x)$ được lưu trong một LUT. Một bộ cộng sẽ được sử dụng để tạo ra các giá trị hàm $\log_2(1 + x)$ được xấp xỉ. Về mặt nguyên tắc, muốn tăng độ chính xác có thể tăng kích thước của bảng LUT, tuy nhiên việc tăng kích thước của LUT sẽ làm tăng độ phức tạp của phần cứng.

Nhằm đảm bảo lỗi xấp xỉ là nhỏ và đồng thời giảm kích thước của bảng LUT, thuật toán đề xuất nhằm tìm giá trị tối ưu của các hệ số a_i và b_i sao cho lỗi xấp xỉ là cực tiểu. Thuật toán đề xuất gồm có hai bước: bước thứ nhất là tìm các giá trị của các hệ số a_i và b_i theo phương pháp trung bình bình phương cực tiểu, bước thứ hai là gán các giá trị a_i và b_i có dạng như (2.9).



Hình 2.2: Phương pháp xấp xỉ đề xuất.

Bước 1: tìm các hệ số a_i, và b_i theo phương pháp trung bình bình phương cực tiểu.

Lấy trung bình bình phương lỗi trong khoảng $[x_1, x_2]$

$$\overline{E^2} = \frac{1}{x_2 - x_1} \int_{x_1}^{x_2} \{\log_2(1+x) - (ax+b)\}^2 dx, \ 0 \le x_1 < x_2 < 1 \quad (2.11)$$

Để tìm lỗi cực tiểu giải hệ sau để tìm a, b:

$$\begin{cases} \frac{\partial \overline{E^2}}{\partial a} = 0 \\ \frac{\partial \overline{E^2}}{\partial b} = 0 \end{cases} \Leftrightarrow \begin{cases} \int_{x_1}^{x_2} -2x\{\log_2(1+x) - (ax+b)\}dx = 0 \\ \int_{x_1}^{x_2} -2\{\log_2(1+x) - (ax+b)\}dx = 0 \end{cases} \quad (2.12)$$

Giải hệ (2.12) với bốn đoạn đều nhau của x trong khoảng $[0, 1)$ ta có các hệ số của bước thứ nhất (a_{step1}, b_{step1}) như Bảng 2.2.

Bước hai: Tìm các giá trị a_i và b_i có dạng như công thức (2.9), bước này được tiến hành như sau:

Trước tiên gán các giá trị a_i có dạng như trong công thức (2.9) có giá trị gần nhất với các giá trị trong bước 1 (a_{step1}). Nhằm đạt được sự dung hòa giữa độ phức tạp của phần cứng và độ chính xác cần đạt được, các hệ số a_i sẽ ấn định có dạng là tổng của hai số lũy thừa của 2. Ví dụ ở đoạn thứ nhất gán $a_1 = 2^0 + 2^{-2} \approx 1,2856$. Tiếp theo tìm lại giá trị b_i tối ưu thỏa mãn

Bảng 2.2: Các hệ số a_i, b_i của bước thứ nhất.

x	a_{step1}	b_{step1}
$0 \leq x < 0,25$	1,2856	0,0062
$0,25 \leq x < 0,5$	1,05104	0,0633
$0,5 \leq x < 0,75$	0,8890	0,1435
$0,75 \leq x < 1$	0,7701	0,2320

công thức (2.9) theo thuật toán mô tả ở Bảng 2.3. Thuật toán này sẽ tìm các hệ số K và B_i trong công thức (2.9) nhằm tìm ra hệ số b_i thỏa mãn lỗi xấp xỉ là cực tiểu.

Bảng 2.3: Thuật toán tìm hệ số b_i tối ưu trong từng phân đoạn.

1:	$BEGIN$: Khởi tạo mảng giá trị B Khởi tạo mảng giá trị K Khởi tạo đoạn giá trị i
2:	for $p = 1 : length(B)$ for $q = 1 : length(K)$ for $k = 1 : length(x)$ tính $E(x)$ theo công thức (2.10) end end end
3:	for $p = 1 : length(K)$ for $q = 1 : length(B)$ tính $E_{max} = max(E(x))$ end end
4:	Tính $min(E_{max})$
5:	END Xuất giá trị B, K tương ứng với min(E_{max})

Sau khi thực hiện gán các giá trị a_i và thực hiện tính toán các hệ số b_i theo lưu đồ thuật toán ở trên, nhận được các giá trị a_i, b_i sau bước thứ hai

(a_{step2}, b_{step2}) như trong Bảng 2.4.

Bảng 2.4: Các hệ số a_i, b_i của bước thứ hai.

x	a_{step2}	b_{step2}
$0 \leq x < 0,25$	$2^0 + 2^{-2}$	2^{-7}
$0,25 \leq x < 0,5$	$2^0 + 2^{-4}$	2^{-4}
$0,5 \leq x < 0,75$	$2^0 - 2^{-3}$	77×2^{-9}
$0,75 \leq x < 1$	$2^{-1} + 2^{-2}$	2^{-2}

Như vậy, hàm xấp xỉ được biểu diễn như công thức (2.13)

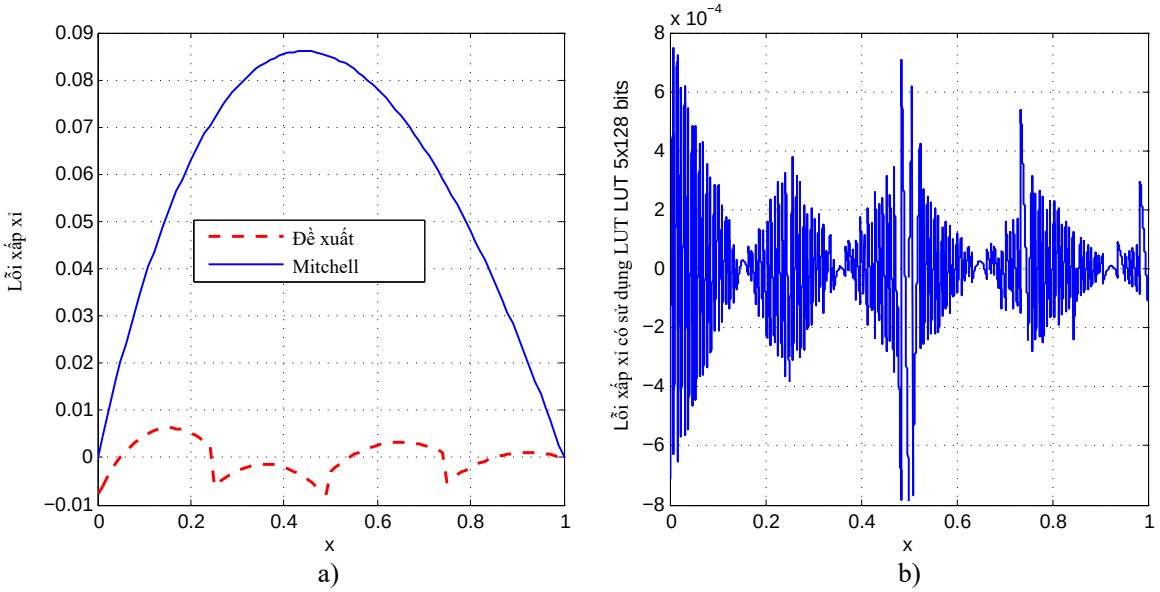
$$\log_2(1+x) \approx \begin{cases} (2^0 + 2^{-2})x + 2^{-7}, & 0 \leq x < 0,25 \\ (2^0 + 2^{-4})x + 2^{-4}, & 0,25 \leq x < 0,5 \\ (2^0 - 2^{-3})x + 77 \times 2^{-9}, & 0,5 \leq x < 0,75 \\ (2^{-1} + 2^{-2})x + 2^{-2}, & 0,75 \leq x < 1 \end{cases} \tag{2.13}$$

2.4. Phân tích sai số và so sánh

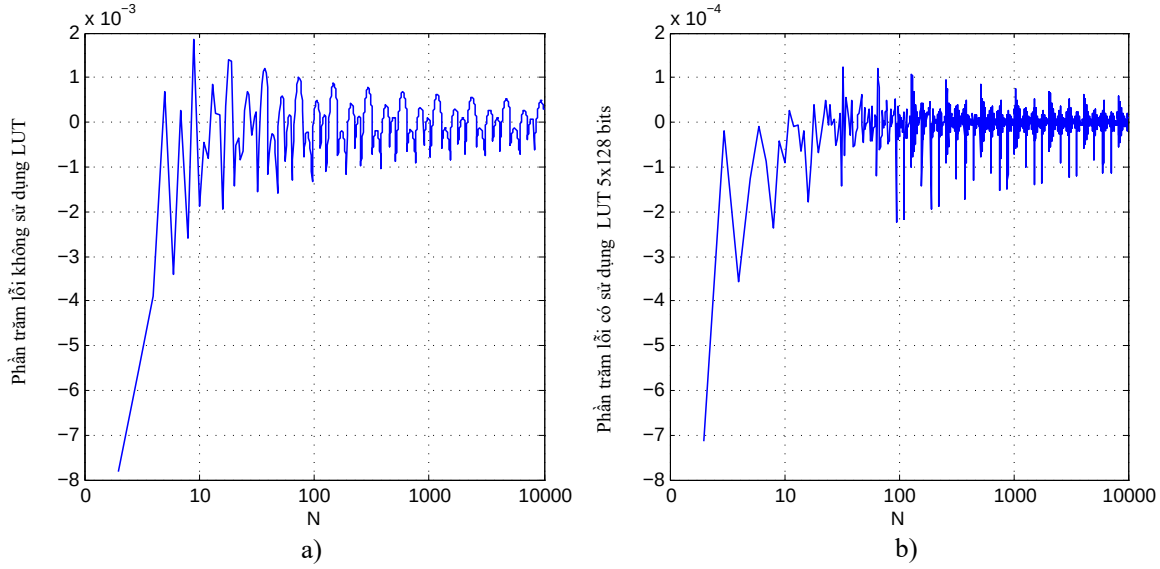
Phương pháp đề xuất được tiến hành phân tích sai số và so sánh với các kết quả trong [34], [41] và [48] (cũng xấp xỉ tuyến tính 4 đoạn). Các tham số được phân tích và so sánh lỗi gồm các tham số sai số (MAE, MPE, MNE), trung bình sai số, các tham số phần trăm sai số (MPEP, MNEP) và trung bình phần trăm sai số.

Hình 2.3 thể hiện sai số của phương pháp đề xuất trong trường hợp chưa sử dụng LUT (a), và sai số của phương pháp đề xuất khi có sử dụng bảng LUT kích thước 5×128 bits (b). Hình 2.4 thể hiện kết quả phần trăm sai số tương ứng với trường hợp không sử dụng LUT (a) và khi có sử dụng LUT

(b).



Hình 2.3: Sai số của phương pháp xấp xỉ đề xuất.



Hình 2.4: Phần trăm sai số của phương pháp xấp xỉ đề xuất.

Bảng 2.5 thể hiện các giá trị tham số đánh giá sai số của phương pháp đề xuất. Các giá trị về độ lệch cực đại, tỷ lệ sai số cực đại là tương tự với các kết quả trong [34], [41] và [48]. Đặc biệt các tham số về trung bình sai số và trung bình phần trăm sai số là giảm đáng kể so với các kết quả trong [34], [41]

và [48]

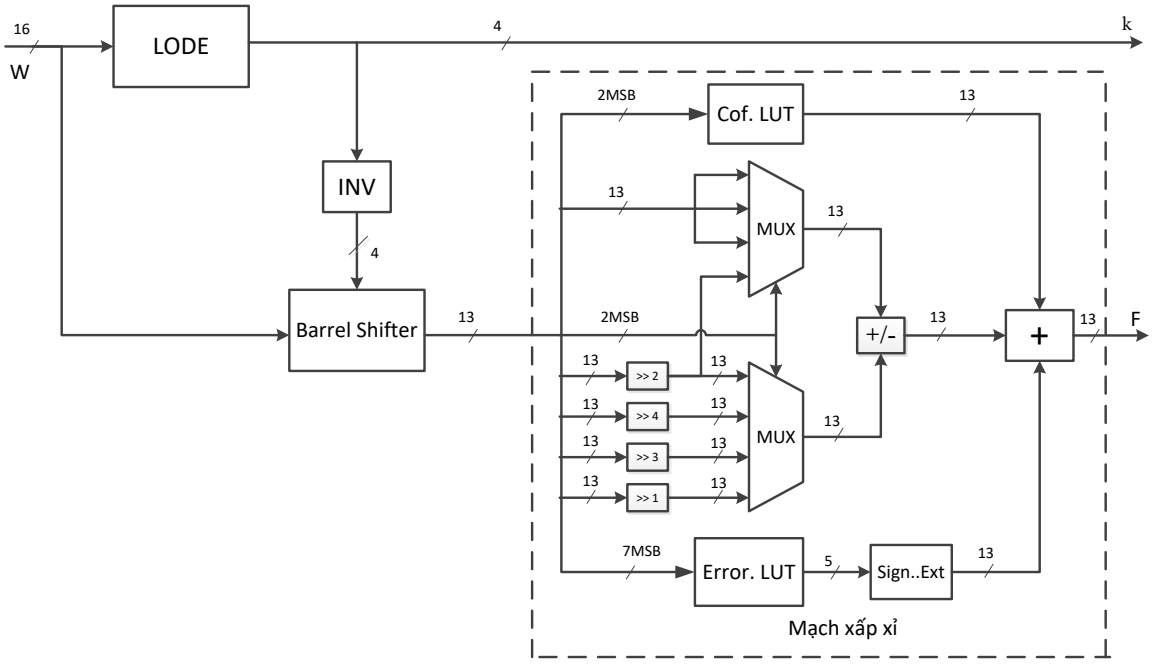
Bảng 2.5: Phân tích sai số và so sánh của phương pháp đề xuất.

Phương pháp	E.L.Hall [34]	De caro [41]	V.P. Hoang [48]	Đề xuất (chưa sử dụng LUT)	Đề xuất (có sử dụng LUT)
MPE	$8,1 \times 10^{-3}$	$9,8 \times 10^{-3}$	$10,1 \times 10^{-3}$	$\mathbf{6,3 \times 10^{-3}}$	$\mathbf{7,49 \times 10^{-4}}$
MNE	$-2,3 \times 10^{-3}$	$-6,8 \times 10^{-3}$	$-10,1 \times 10^{-3}$	$\mathbf{-8,8 \times 10^{-3}}$	$\mathbf{-7,88 \times 10^{-4}}$
MAE	$8,1 \times 10^{-3}$	$9,8 \times 10^{-3}$	$10,1 \times 10^{-3}$	$\mathbf{8,8 \times 10^{-3}}$	$\mathbf{7,88 \times 10^{-4}}$
Trung bình lỗi	$1,2 \times 10^{-3}$	$2,1 \times 10^{-3}$	$2,1 \times 10^{-3}$	$\mathbf{1,76 \times 10^{-3}}$	$\mathbf{1,43 \times 10^{-6}}$
MEP	$0,78$ %	$0,43$ %	$0,40$ %	$\mathbf{0,78}$ %	$\mathbf{0,07}$%
MPEP	$0,13$ %	$0,18$ %	$0,31$ %	$\mathbf{0,18}$ %	$\mathbf{0,01}$%
MNEP	$-0,78$ %	$-0,43$ %	$-0,40$ %	$\mathbf{-0,78}$ %	$\mathbf{0,07}$%
Trung bình phần trăm lỗi	$5,81 \times 10^{-5}$	$2,03 \times 10^{-4}$	$2,48 \times 10^{-4}$	$\mathbf{4,98 \times 10^{-5}}$	$\mathbf{2,42 \times 10^{-6}}$

2.5. Kiến trúc phần cứng và kết quả thực thi

Dựa trên phương pháp đề xuất, một kiến trúc phần cứng của bộ chuyển đổi logarithm cho số nguyên N có 16 bit đầu vào tương ứng với 4 bit phần nguyên và 13 bit phần thập phân đầu ra được thực hiện. Kiến trúc phần cứng đề xuất cho tính toán hàm logarithm cơ số hai của một số N thể hiện như Hình 2.5. Khối phát hiện bit 1 trọng số cao nhất và mã hóa (LODE) để tính toán phần nguyên k và mã hóa thành chuỗi nhị phân. Một khối đảo (INV) và mạch ghi dịch (Barrel Shifter) được sử dụng để tạo ra phần thập phân x. Hai bit MSB của x được sử dụng để lựa chọn một trong bốn đoạn xấp xỉ. Các bộ ghép kênh được sử dụng để lựa chọn các thao tác dịch phải của

chuỗi bit đầu vào nhằm tạo ra hệ số góc (a_i), một bảng LUT nhỏ chứa hệ số của bốn đoạn tuyến tính (Cof. LUT). Một bảng LUT chưa các giá trị bù lỗi (Error. LUT). Để tăng độ chính xác xấp xỉ có thể tăng kích thước của bảng LUT này, tuy nhiên để dung hòa giữa tài nguyên sử dụng và độ chính xác kích thước bảng LUT được lựa chọn là 5×128 bits. Đầu ra của khối xấp xỉ tạo ra phần thập phân F của $\log_2 N$.



Hình 2.5: Kiến trúc đề xuất cho bộ tính toán logarithm 16 bit.

Kiến trúc đề xuất đã được mô hình hóa trên ngôn ngữ VHDL và thực thi trên thiết bị Xilinx FPGA Spatan 3E. Tài nguyên tiêu tốn trong thực thi trên FPGA được tính thông qua số FPGA LUT được sử dụng. Kiến trúc đề xuất được thực thi và so sánh với các nghiên cứu tương tự thông qua tham số tích giữa chi phí thực thi và độ trễ (ADP: Area-Delay product). Bảng 2.6 thể hiện các kết quả thực thi trên FPGA của phương pháp đề xuất và so sánh với các kết quả trong [34], [41], [40]. Tham số ADP của kiến trúc đề xuất đã

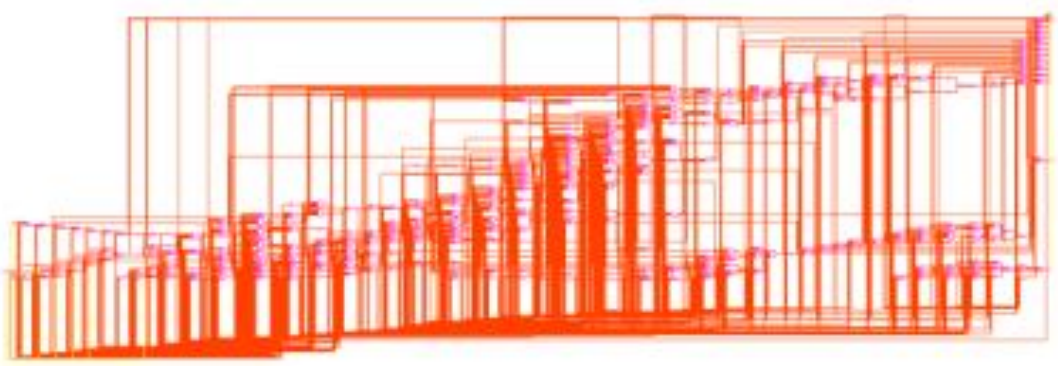
giảm được 36 % so với phương pháp trong [34], so với các kết quả trong [40] và [41] đã giảm tương ứng là 14 % và 13 %.

Bảng 2.6: Kết quả thực thi và so sánh trên FPGA.

Phương pháp	Slices	FPGA LUTs	Delay (ns)	ADP($\times 10^3$)
Hall [34]	98	188	28,574	5,371
Guitierz [40]	86	163	24,479	3,965
De Caro [41]	86	162	24,479	3,965
Đề xuất	**81**	**149**	**23,066**	**3,436**

Bảng 2.7: Kết quả thực thi và so sánh trên ASIC của phương pháp đề xuất.

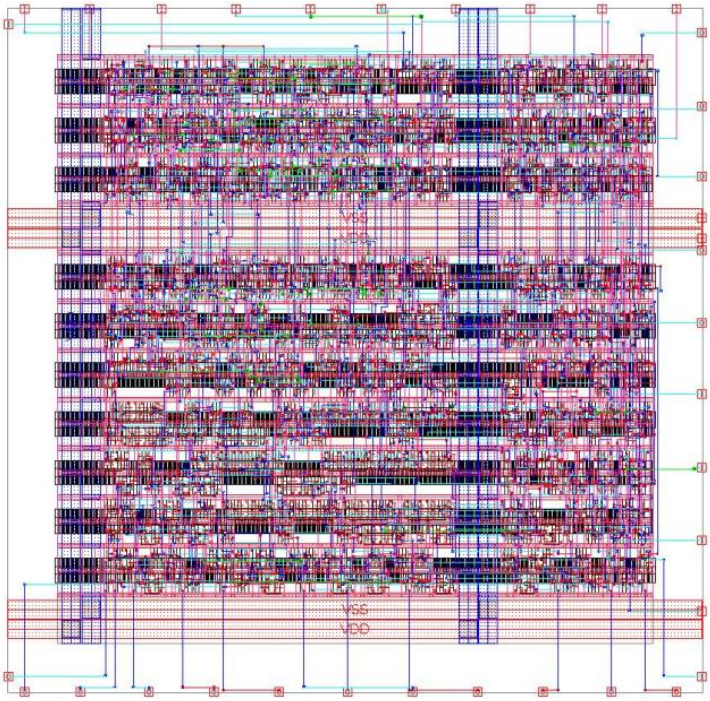
Phương pháp	Gutierez [40]	Hall [34]	De caro [41]	Đề xuất
Diện tích ($\times 10^3 \mu m^2$)	42,9	54,1	40,3	**27,2**
Độ giữ chậm (ns)	12,2	13,2	12,0	**11,5**
ADP($\times 10^3$)	523,4	714,1	483,6	**312,8**
Công suất (μW)	17,70	23,09	14,98	**9,40**



Hình 2.6: Netlist được tổng hợp của bộ chuyển đổi logarithm 16 bit sử dụng thư viện ASIC SOTB CMOS 65nm.

Ngoài ra, kết quả thực thi cho bộ chuyển đổi logarithm theo phương pháp

đề xuất trên công nghệ ASIC SOTB CMOS 65nm thể hiện trên Bảng 2.7. Có thể thấy rằng bộ chuyển đổi logarithm đề xuất đã cải thiện đáng kể tài nguyên, tốc độ tính toán và công xuất tiêu thụ. Công suất tiêu thụ được ước lượng dựa trên netlist được tổng hợp bằng công cụ Synosys Design Compiler. Hình 2.6 là netlist của bộ tạo logarithm đề xuất. Hình 2.7 thể hiện layout của bộ chuyển đổi logarithm đề xuất trên công nghệ ASIC CMOS 65nm.



Hình 2.7: Layout của bộ chuyển đổi logarithm 16-bit ($145 \times 145 \mu m$).

2.6. Kết luận chương 2

Trong chương này, nghiên cứu sinh đã trình bày một phương pháp xấp xỉ hàm logarithm trên cơ sở phương pháp của Mitchell. Trong đó sử dụng phương pháp xấp xỉ phân đoạn tuyến tính đều, với các hệ số góc của đoạn là có dạng là tổng của các số lũy thừa của hai. Để tăng độ chính xác xấp xỉ, một bảng LUT 5×128 bits được sử dụng. Trên cơ sở đó một kiến trúc thực

hiện cho bộ chuyển đổi một số nhị phân 16 bit nguyên cũng được đề xuất và thực thi trên FPGA và ASIC. Các kết quả thực thi và kiểm chứng cho thấy bộ chuyển đổi logarithm đề xuất cải thiện về tài nguyên phần cứng.

Chương 3
THIẾT KẾ PHẦN CỨNG TÍNH TOÁN HÀM SIN

3.1. Đặt vấn đề

Tính toán hàm *sin* là một yêu cầu đặt ra trong nhiều ứng dụng DSP chẳng hạn trong các bộ tổ hợp tần số trực tiếp (DDFS: Direct Digital Frequency Synthesizer), trong các mạch trộn tần, các mạch điều chế số. Tính toán hàm *sin* có thể thực hiện bằng phần mềm, tuy nhiên tốc độ tính toán là chậm và không phù hợp với các ứng dụng đòi hỏi tốc độ xử lý cao. Vì vậy, với các ứng dụng yêu cầu về tốc độ cao và các DSP thời gian thực, thực thi bằng phần cứng là cần thiết. Một vài phương pháp thực thi hàm *sin* đã được nghiên cứu, chẳng hạn như các kiến trúc tính toán hàm *sin* dựa trên thuật toán CORDIC [20]. Tuy nhiên, các kiến trúc này dựa trên giải pháp lặp do vậy không phù hợp với các ứng dụng yêu cầu tốc độ tính toán cao và các ứng dụng DSP thời gian thực. Ngoài ra, một số phương pháp xấp xỉ hàm *sin* như xấp xỉ bằng hàm bậc cao, xấp xỉ sử dụng chuỗi Taylor thường có độ phức tạp phần cứng cao. Vì vậy, với các ứng dụng yêu cầu về cao về tốc độ tính toán và đơn giản về phần cứng thực thi đòi hỏi phải có các giải pháp hiệu quả cho thực thi tính toán hàm *sin*.

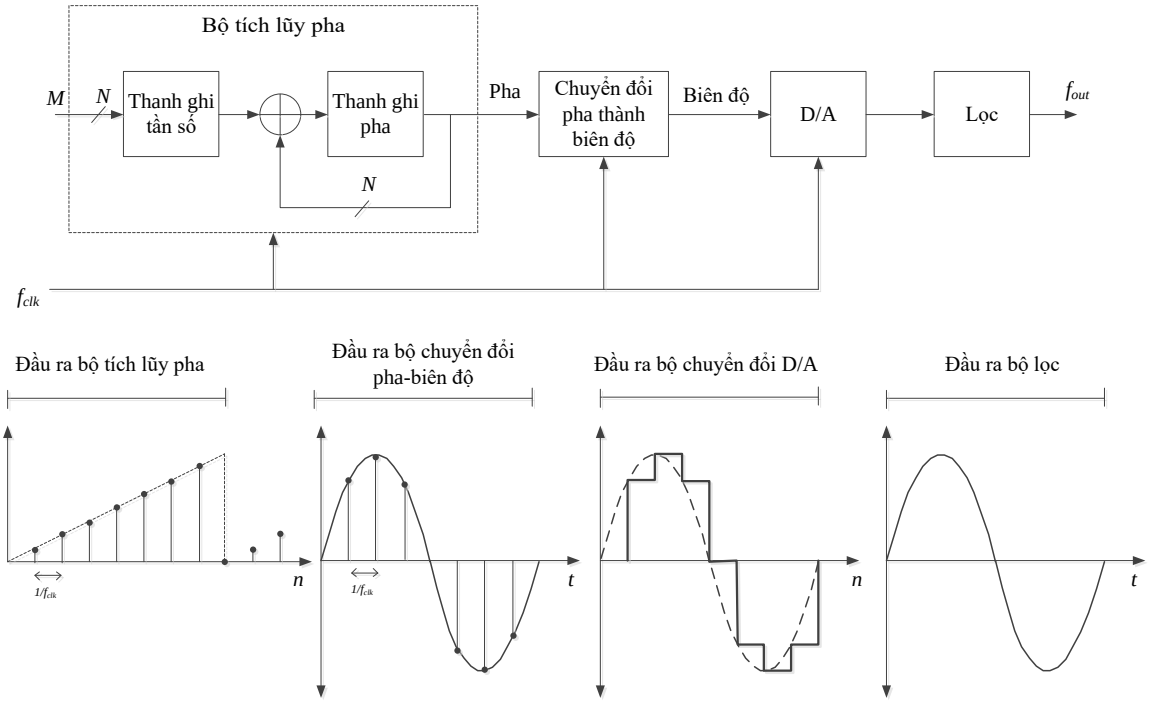
Các hệ thống truyền thông hiện đại với các dịch vụ tốc độ cao đòi hỏi các bộ tổ hợp tần số chất lượng tốt. Một bộ tổ hợp tần số sử dụng xử lý tín hiệu số để tạo ra các tín hiệu đầu ra có pha và tần số có thể điều chỉnh. Tần số

đầu ra là một giá trị của phép chia tần số đồng hồ tham chiếu. Hệ số chia sẽ được thiết lập bởi một từ điều chỉnh nhị phân. DDFS có các ưu điểm như độ chính xác tần số cao, tốc độ chuyển tần số nhanh, nhiễu pha thấp, có thể điều chế trực tiếp pha và tần số. DDFS đã được sử dụng trong nhiều ứng dụng của các hệ thống truyền thông số. Ví dụ có thể sử dụng DDFS để tạo ra bộ tạo tín hiệu đồng hồ với 2^{N-1} tần số đầu ra với N là độ phân dải của bộ tích lũy pha. Đặc điểm này là rất hữu dụng cho các hệ thống cần nhiều tần số tham chiếu và chuyển đổi đồng hồ tham chiếu đòi hỏi nhanh và thường xuyên. Trong các hệ thống truyền thông hiện đại, DDFS được dùng để thay thế cho các vòng khóa pha (PLL: Phase Locked Loops) do nó có nhiều ưu điểm hơn so với PLL. Chương này của luận án sẽ tập trung vào thiết kế phần cứng tính toán hàm *sin* ứng dụng cho DDFS.

3.2. Bộ tổ hợp tần số số trực tiếp

Một DDFS có sơ đồ khối đơn giản như Hình 3.1 [49]. Nó bao gồm các khối cơ bản sau: một bộ tích lũy pha, một bộ chuyển đổi pha thành biên độ (PAC: Phase to Amplitude Converter), một bộ chuyển đổi số thành tương tự (DAC: Digital Analog Converter) và một bộ lọc. Một DDFS sẽ tạo ra tín hiệu sin với tần số cho trước. Tần số đầu ra sẽ phụ thuộc vào ba biến: tần số đồng hồ f_{clk}; số nhị phân được lập trình đưa tới thanh ghi pha (còn gọi là từ điều khiển tần số, M) và độ rộng N bit của bộ tích lũy pha. Từ điều khiển tần số là thành phần đầu vào chính của bộ tích lũy pha.

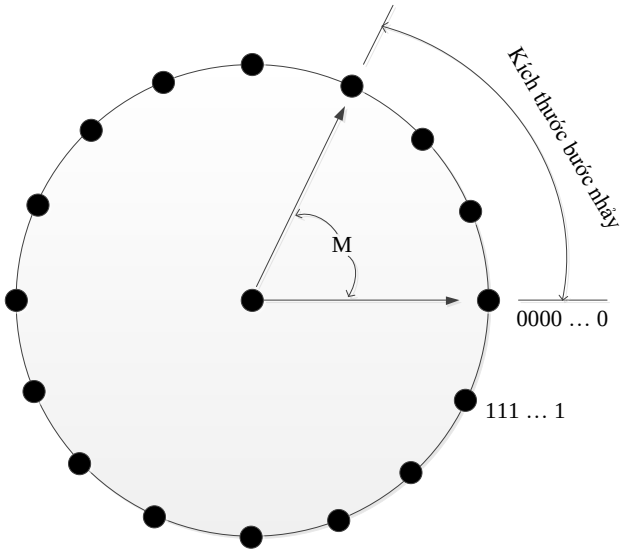
Bộ tích lũy pha bao gồm một thanh ghi tần số N bit dùng để chứa giá trị của từ điều khiển tần số, sau đó là một bộ cộng đủ N bit và một thanh ghi pha. Từ điều khiển tần số sẽ được đưa vào thanh ghi tần số. Tại mỗi xung



Hình 3.1: Sơ đồ khối đơn giản của DDFS và các tín hiệu trong DDFS.

đồng hồ giá trị của từ điều khiển tần số sẽ được cộng với dữ liệu trước đó và được giữ trong thanh ghi pha. Giá trị của từ điều khiển tần số biểu diễn góc pha tại mỗi bước và được cộng với giá trị trước đó tại $1/f_{clk}$ giây để tạo ra sự tăng pha tuyến tính. Giá trị pha được tạo ra từ bộ tích lũy pha tuân theo đặc tính modulo 2^N, nguyên lý của nó như một vòng quay pha số thể hiện trên Hình 3.2.

Bộ chuyển đổi pha thành biên độ sẽ chuyển đổi góc pha cung cấp từ bộ tích lũy pha thành biên độ *sin* tương ứng. Trong hầu hết các trường hợp thì PAC sẽ được thực thi dưới dạng số và theo sau nó là một bộ chuyển đổi số-tương tự và một bộ lọc thông thấp để tạo ra tín hiệu tương tự ở đầu ra. Cách thực thi PAC đơn giản nhất là sử dụng một bộ nhớ ROM đóng vai trò như là một bảng LUT, nó chuyển đổi giá trị pha thành biên độ sóng *sin* tương ứng.



Hình 3.2: Vòng pha số.

Công thức cơ bản của DDFS là:

$$f_{out} = \frac{M \times f_{clk}}{2^N}, \tag{3.1}$$

trong đó, f_{out} là tần số đầu ra của DDFS, M là từ điều khiển tần số nhị phân, f_{clk} là tần số đồng hồ của hệ thống và N là độ dài bit của bộ tích lũy pha.

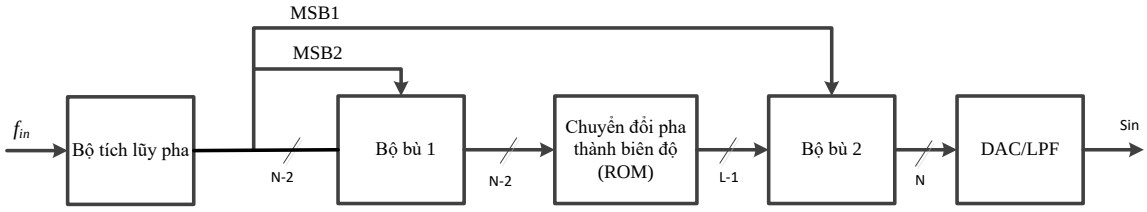
3.3. Bộ chuyển đổi pha-biên độ trong DDFS

Trong kiến trúc của DDFS, PAC được thực thi theo cách đơn giản nhất là sử dụng một ROM đóng vai trò như là một LUT, *trong luận án này gọi là sin-LUT*. Cơ sở của phương thức sử dụng sin-LUT là tính toán trước các giá trị biên độ sin rời rạc và lưu nó vào trong một cấu trúc dạng bảng tra, mỗi một giá trị biên độ chứa trong bộ nhớ này được định địa chỉ bởi giá trị pha tương ứng được cung cấp từ bộ tích lũy pha.

Nhược điểm chính của phương pháp này là độ chính xác của tần số đầu ra phụ thuộc vào kích thước của LUT. Khi đòi hỏi về độ chính xác tần số cao sẽ

dẫn đến yêu cầu kích thước LUT lớn và do đó tăng công suất tiêu thụ, giảm tốc độ, tăng độ phức tạp về phần cứng và chi phí.

Có thể giảm kích thước của sin-LUT mà không ảnh hưởng đến độ chính xác của tần số đầu ra bằng cách cắt bớt một vài bit đầu ra của bộ tích lũy pha, chỉ giữ lại một số bit pha quan trọng nhất. Tuy nhiên, điều này sẽ gây ra lỗi biên độ đầu ra và do đó gây ra nhiễu tại phổ đầu ra. Một kỹ thuật đơn giản nữa để giảm độ phức tạp của PAC là lợi dụng tính đối xứng 1/4 của hàm *sin*. Phương pháp này lợi dụng đặc tính của hàm *sin* là hàm lẻ trong một chu kỳ do đó giá trị biên độ ở hai nửa chu kỳ là đối xứng nhau, còn trong một nửa chu kỳ giá trị hàm *sin* là hàm chẵn. Do đó, để lưu giữ giá trị của một chu kỳ sóng chỉ cần lưu trữ 1/4 chu kỳ (giá trị biên độ ứng với pha từ 0 đến $\pi/2$). Phương pháp này cho phép nén kích thước LUT còn 1/4. Để tạo ra tín hiệu *sin* đầy đủ cần phải thêm vào một số mạch logic. Hình 3.3 mô tả phương pháp lợi dụng tính đối xứng của hàm *sin* [49]. Bộ bù thứ nhất sẽ sử dụng bit có trọng số lớn thứ hai của chuỗi bit đầu ra bộ tích lũy pha (MSB2) để điều khiển việc đảo pha còn bit có trọng số lớn nhất của đầu ra bộ tích lũy pha (MSB1) được sử dụng để đảo biên độ đầu ra của PAC



Hình 3.3: Kiến trúc DDFS lợi dụng tính đối xứng góc 1/4 của sóng *sin*.

Ở đây, luôn xuất hiện một lỗi giữa đầu ra của PAC và giá trị *sin* lý tưởng. Lỗi này gây ra do lỗi lượng tử biên độ do biểu diễn biên độ bằng một số hữu hạn các bit và lỗi gây ra bởi thuật toán tính toán biên độ trong PAC. Vì vậy,

phổ đầu ra của bộ tổ hợp sẽ gồm hai thành phần: phổ của sóng *sin* thuần túy và phổ do nhiễu gây bởi lỗi biên độ tại các mẫu đầu ra. Tỷ số công suất giữa thành phần tần số không mong muốn lớn nhất và công suất của hàm *sin* cần đạt được gọi là khoảng động nhiễu tự do (SFDR: Spurious Free Dynamic Range), nó là một tham số quyết định hiệu quả của bộ tổ hợp tần số [50].

3.4. Các kỹ thuật chuyển đổi pha-biên độ

Nhiều nghiên cứu đã được thực hiện để đạt được kỹ thuật chuyển đổi pha-biên độ hiệu quả. Các kỹ thuật chuyển đổi pha-biên độ có thể được phân chia thành các phương pháp sau: phương pháp phân chia góc, các kỹ thuật dựa trên quay góc, xấp xỉ đa thức và phương pháp nén biên độ *sin* [50].

3.4.1. Phân chia góc

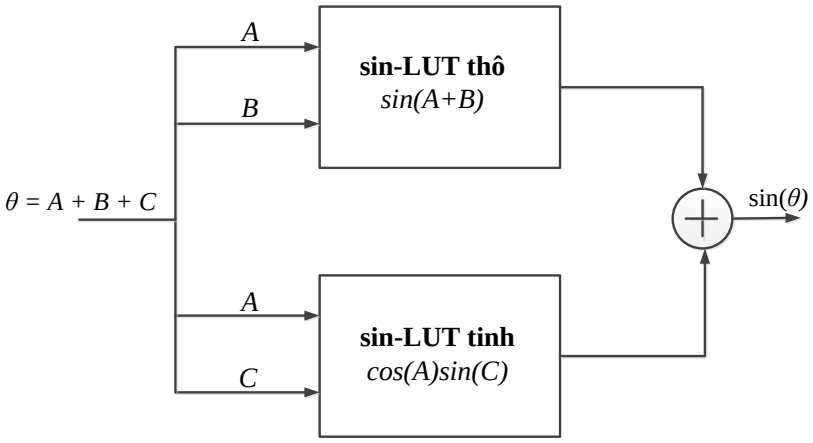
Trong kỹ thuật phân chia góc, các bit biểu diễn góc pha ở đầu ra bộ tích lũy pha sẽ được chia thành các phần khác nhau tương ứng với các chuỗi bit kề nhau, khi đó các góc pha được biểu diễn bằng tổng các góc thô và tinh. Sau đó sử dụng các sin-LUT để biểu diễn các góc này, đồng thời sử dụng các tính chất lượng giác và các kỹ thuật xấp xỉ để giảm dung lượng của sin-LUT so với việc sử dụng trực tiếp sin-LUT.

Tierney và các cộng sự trong [51]] là những người đầu tiên đề xuất kỹ thuật này, sau đó một số nghiên cứu để cải tiến hiệu quả của phương pháp này trong các công trình [52–56]. Điển hình của phương pháp này là nghiên cứu của Sunderland và các cộng sự trong [53]. Trong đó, từ pha đầu vào được chia thành 3 phần A, B và C, với A và C là các phần tương ứng với các bit

có trọng số lớn nhất và bé nhất của từ pha đầu vào và sử dụng xấp xỉ sau:

$$\begin{aligned}\sin\theta &= \sin(A+B+C) &&= \sin(A+B)\cos C + \cos(A+B)\sin C \\ &= \sin(A+B)\cos C &&+ \cos A\cos B\sin C - \sin A\sin B\sin C \\ &\approx \sin(A+B) &&+ \cos A\sin C \end{aligned} \tag{3.2}$$

Sau đó sin-LUT thứ nhất sẽ được sử dụng để chứa giá trị của $\sin(A+B)$ và sin-LUT thứ hai sẽ chứa giá trị $\cos A.\sin C$ được tính toán trước, đầu ra của các ROM sẽ được cộng với nhau để tạo ra giá trị xấp xỉ của sin như mô tả trên Hình 3.4



Hình 3.4: Kiến trúc Sunderland.

Nicholas và các cộng sự trong [54] đưa ra một phương pháp cải tiến hơn so với phương pháp của Sunderland. Phương pháp của Nicholas cũng áp dụng việc phân chia thành ROM thô và ROM tinh tương tự như trong phương pháp Sunderland. Tuy nhiên, các mẫu lưu trữ được tính toán bằng phương pháp tối ưu số để tối thiểu hóa lỗi hoặc trung bình bình phương lỗi của giá trị xấp xỉ và giá trị hàm sin tương ứng.

3.4.2. Các kỹ thuật dựa trên quay góc

Với mục đích loại bỏ hoàn toàn việc sử dụng các sin-LUT một số nghiên cứu đã sử dụng thuật toán CORDIC [20] và các thuật toán dựa trên quay góc khác để thực hiện PAC. Nhiều nghiên cứu đã đạt được hiệu quả với SFDR cao và độ phức tạp phần cứng phù hợp. Ưu diểm của phương pháp này là nó dễ dàng tổng hợp, không sử dụng mạch nhân phức tạp, tuy nhiên có nhược điểm đáng kể là phương pháp này dựa trên các kiến trúc lặp do đó độ giữ chậm lớn.

3.4.3. Xấp xỉ đa thức

Xấp xỉ đa thức là một trong những phương pháp hiệu quả để xấp xỉ hàm *sin* từ góc pha đầu vào, có thể biểu diễn xấp xỉ đa thức theo công thức tổng quát sau [50]:

$$\sin(\frac{\pi x}{2}) \approx \begin{cases} \sum_{i=0}^{r} c_{0i}(x - x_0)^i, & x_0 \leq x < x_1, \ (x_0 = 0) \\ \sum_{i=0}^{r} c_{1i}(x - x_1)^i, & x_1 \leq x < x_2 \\ \vdots & \\ \sum_{i=0}^{r} c_{ki}(x - x_k)^i, & x_k \leq x < x_{k+1} \\ \vdots & \\ \sum_{i=0}^{r} c_{(s-1)i}(x - x_{s-1})^i, & x_{s-1} \leq x < x_s, \ (x_s = 1), \end{cases} \tag{3.3}$$

ở đây, x là góc pha được chuẩn hóa trong khoảng $[0,\ 1)$, r là bậc của đa thức xấp xỉ, s là số phân đoạn xấp xỉ, c_{ki} là các hệ số của đa thức xấp xỉ và x_k là điểm đầu của đoạn thứ k.

Khi thực hiện xấp xỉ đa thức có bốn yếu tố chính là: bậc của đa thức xấp xỉ; số phân đoạn được dùng; vị trí của điểm đầu mỗi đoạn và cách thức tìm

các hệ số của đa thức xấp xỉ.

Nếu sử dụng xấp xỉ phân đoạn đều, tức là các đoạn có độ dài bằng nhau, thì điểm đầu của mỗi đoạn x_k sẽ xác định bởi công thức sau:

$$x_k = \frac{k}{s}, \qquad k \in \{0,\, 1,\, \ldots,\, s\}\,. \tag{3.4}$$

Hơn nữa số phân đoạn s là một số lũy thừa của hai thì giá trị của x trong đoạn thứ k, $x_k \leq x < x_{k+1}$ có thể nhận được bằng cách loại bỏ $\log_2 s$ bit có trọng số lớn nhất của giá trị đầu vào x.

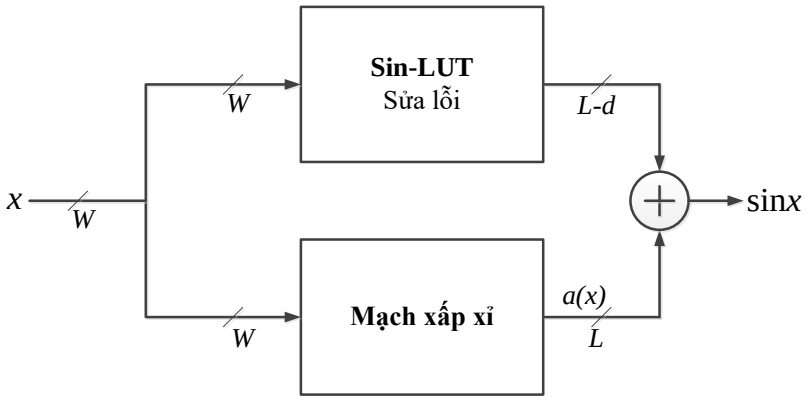
Các phương pháp sử dụng xấp xỉ đa thức bậc nhất được nghiên cứu trong các công trình [57–60]. Freeman trong [57] đã đề xuất một kiến trúc PAC dựa trên phân chia góc phần tư thứ nhất của hàm *sin* thành 16 đoạn tuyến tính, các đoạn có độ dài như nhau. Sau đó sử dụng hai LUT để chứa các hệ số và một mạch nhân. Bellaouar và các cộng sự trong [58] đã đề xuất một phương pháp nội suy tuyến tính cho thiết kế PAC. Phương pháp của Bellaouar dựa trên khai triển Taylor với chỉ hai số hạng đầu tiên trong khai triển Taylor được sử dụng. Ngoài ra, phương pháp của Bellaouar còn sử dụng một mạch nhân. Liu và các cộng sự trong [59] đề xuất một kiến trúc PAC sử dụng xấp xỉ đa thức bậc nhất với 12 đoạn có độ dài không bằng nhau. Do các đoạn có độ dài không bằng nhau nên trong thực thi phần cứng độ phức tạp sẽ tăng lên. Tuy nhiên, trong kiến trúc đề xuất của Liu không sử dụng các mạch nhân vì các hệ số có dạng là các số lũy thừa của 2 và chúng được lựa chọn với tiêu chí tối thiểu lỗi biên độ cực đại. Langlois và Al-Khalili trong [60] đề xuất phương án nội suy tuyến tính với các phân đoạn có độ dài bằng nhau tương tự như phương pháp của Freeman và Bellaouar nhưng trong đó không sử dụng mạch nhân. Các tác giả cũng đã đề xuất một số phương án lựa chọn

các hệ số dựa trên cực đại SFDR, một vài hệ số được thiết lập để đạt được các thiết kế với SFDR trong khoảng từ 60 tới 96 dBc.

Các phương pháp xấp xỉ hàm *sin* bằng đa thức bậc cao ứng dụng cho PAC được nghiên cứu trong các công trình [61–64]. Các nghiên cứu này thực hiện xấp xỉ các hàm *sin* bằng các hàm bậc 2 đến bậc 4. Weaver và Kerr trong [61] đề xuất một phương pháp xấp xỉ đa thức bậc hai dựa trên sử dụng ba số hạng đầu tiên của khai triển Taylor. Kiến trúc thực hiện phương pháp của Weaver và Kerr đòi hỏi hai sin-LUT để chứa các hệ số cùng với một mạch nhân và một mạch bình phương. Fanucci và các cộng sự trong [62] đã thực hiện xấp xỉ hàm *sin* trong góc phần tư đầu tiên, trong đó sử dụng bốn phân đoạn và mỗi phân đoạn được xấp xỉ bằng một đa thức bậc hai. Các hệ số của đa thức xấp xỉ được tìm để cực tiểu lỗi cực đại. Kiến trúc thực hiện phương pháp của Fanucci sử dụng 3 sin-LUT để chứa các hệ số và hai mạch nhân để tính toán hàm xấp xỉ. De Caro trong [63] đã đề xuất hai thiết kế cho kiến trúc cầu phương, trong đó xấp xỉ góc phần tám đầu tiên sử dụng đa thức bậc hai và bậc ba tương ứng cho các hàm *sin* và *cosin*. Trong mỗi trường hợp việc lựa chọn các hệ số được thực hiện nhằm tối ưu SFDR. Một số các phương pháp lựa chọn hệ số đã được các tác giả thực hiện bao gồm chuỗi Taylor, đa thức Chebysev và đa thức Lengendre. Q.K. Omran và các cộng sự trong [64] đã thực hiện một DDFS dựa trên xấp xỉ góc phần tư đầu tiên bởi xấp xỉ hai đoạn sử dụng xấp xỉ bậc 1 cho đoạn thứ nhất và bậc 4 cho đoạn thứ hai. Các tác giả cũng trình bày cách lựa chọn các hệ số tối ưu để đạt được sự đơn giản về phần cứng và SFDR cao. Trên cơ sở đó, kiến trúc bộ DDSF được đưa ra và thực thi trên ASIC. Theo các tác giả SFDR tốt nhất đạt được là 91,244 dBc.

3.4.4. Các phương pháp nén biên độ sin

Phương pháp nén biên độ *sin* là giải pháp sử dụng các mạch đơn giản để tính toán xấp xỉ hàm *sin* và các giá trị độ lệch giữa hàm xấp xỉ và hàm *sin* lý tưởng được chứa trong một sin-LUT, sau đó đầu ra của sin-LUT sẽ được cộng với giá trị xấp xỉ để tạo ra kết quả cuối cùng. Phương pháp này được mô tả như trên Hình 3.5. Hiệu quả của phương pháp này là giảm độ rộng từ lưu trữ trong sin-LUT và do đó giảm kích thước tổng của sin-LUT. Tùy vào độ chính xác của phép xấp xỉ mà độ rộng từ của sin-LUT sẽ giảm đi một số xác định d và kích thước của sin-LUT sẽ giảm đi một hệ số là $(L - d)/L$.



Hình 3.5: Phương pháp nén biên độ sin.

Không mất tính tổng quát cho góc pha $x \in [0,\ 1)$, sai số $\varepsilon(x)$ giữa giá trị sóng *sin* lý tưởng có biên độ là A và đầu ra của mạch xấp xỉ, $a(x)$, được cho bởi.

$$\varepsilon(x) = A\sin(\frac{\pi x}{2}) - a(x). \tag{3.5}$$

Giá trị cực đại của sai số $\varepsilon(x)$ sẽ quyết định số bit được giảm của độ rộng từ nhớ của sin-LUT. Với mục đích đơn giản thì các mạch xấp xỉ thường được thiết kế sao cho sai số $\varepsilon(x)$ là hoàn toàn âm hoặc hoàn toàn dương. Theo đó,

để tiết kiệm một bit của từ nhớ trong sin-LUT thì giá trị tuyệt đối của $\varepsilon(x)$ nhỏ hơn 0,5. Để tiết kiệm được 2 bit thì $|\varepsilon(x)| < 0,25$ và để tiết kiệm được d bit thì $|\varepsilon(x)| < 2^d$.

Một dạng đơn giản nhất của phương pháp này là thuật toán độ chênh lệch sin-pha [53]. Trong thuật toán này, hàm $a(x)$ trong công thức (3.5) là hàm đơn giản $a(x) = x$, lỗi tuyệt đối cực đại có thể dễ dàng tính được và nó nhỏ hơn 0,25 và vì vậy nó sẽ tiết kiệm được hai bit trong từ nhớ của sin-LUT.

Yamagishi và các cộng sự trong [65] đề xuất một cải tiến hiệu quả hơn so với thuật toán độ chênh lệch sin-pha gọi là "xấp xỉ lượng giác kép". Phương pháp này giảm được 3 bit trong từ nhớ sin-LUT. Mạch xấp xỉ thực hiện theo công thức (3.6)

$$a(x) = \begin{cases} \frac{5}{4}x\,, & 0 \leq x < \frac{1}{2} \\ \frac{3}{4}x + \frac{1}{4}\,, & \frac{1}{2} \leq x < 1 \end{cases} \tag{3.6}$$

Mạch logic thực hiện thao tác xấp xỉ này là đơn giản và chỉ gồm các thao tác dịch và cộng. Có thể dễ dàng thấy rằng sai số tuyệt đối cực đại là nhỏ hơn 0,125 và tiết kiệm được 3 bit độ dài từ nhớ trong sin-LUT.

Sodagar và Lahiji trong [66] đã đưa ra phương pháp xấp xỉ cho hai góc phần tư đầu tiên của hàm sin bằng hàm xấp xỉ bậc hai, các khoảng giá trị của x là chia đều và như là một nửa chu kỳ hàm sin. Phương pháp này đơn giản và tạo ra kết quả chính xác hơn so với thuật toán độ chênh lệch sin-pha. Với một giá trị góc pha $x \in [0,\ 2)$ biểu diễn cho một góc trong khoảng $[0,\ \pi)$, đầu ra xấp xỉ cho bởi công thức (3.7).

$$a(x) = x(2 - x), \qquad 0 \leq x < 2. \tag{3.7}$$

Ở đây có phép trừ trong thực tế là thực hiện số bù hai của x. Sai số tuyệt

đối cực đại là nhỏ hơn 0,0625 do vậy có thể tiết kiệm được 4 bit trong từ nhớ của sin-LUT. Tuy nhiên phương pháp này đòi hỏi phải thực hiện phép nhân do đó sẽ dẫn đến tăng độ phức tạp của phần cứng.

Langlois và Al-Khalili trong [67] đã đề xuất một phương pháp xấp xỉ cũng giảm được 4 bit trong từ nhớ của sin-LUT. Phương pháp này dựa trên việc phân chia góc phần tư đầu vào thành ba phần và sử dụng xấp xỉ tuyến tính trong mỗi đoạn. Hàm xấp xỉ có dạng như sau:

$$a(x) = \begin{cases} 0 + \frac{3}{2}x, & 0 \leq x < \frac{5}{16} \\ \frac{5}{32} + x, & \frac{5}{16} \leq x < \frac{3}{4} \\ \frac{1}{2} + \frac{x}{2}, & \frac{3}{4} \leq x < 1 \end{cases} \tag{3.8}$$

Việc thực thi mạch xấp xỉ là khá đơn giản chỉ bằng các phép dịch và các phép cộng. Việc lựa chọn 3 phân vùng của đầu vào x dựa trên 4 bit MSB. Sai số xấp xỉ cực đại là nhỏ hơn 0,0625 và do đó tiết kiệm được 4 bit trong từ nhớ của sin-LUT.

L. W. Hsu và D. C. Chang trong [68] đã thực hiện một hàm xấp xỉ $a(x)$ dựa trên ba phân đoạn với các đoạn không đều nhau theo công thức (3.9). Trong trường hợp này sai số cực đại là 0,029 do vậy tiết kiệm được 5 bit từ nhớ trong sin-LUT. Theo các tác giả thì SFDR trong thiết kế của họ đạt khoảng 75 dBc.

$$a(x) = \begin{cases} 1,4541x, & 0 \leq x < 0,433, \\ 0,9539x + 0,2151, & 0,433 \leq x < 0,73, \\ 0,3281x + 0,6719, & 0,73 \leq x < 1. \end{cases} \tag{3.9}$$

Trong một nghiên cứu khác, Hoang. V-P và P. Cong-Kha trong [69] đã thực hiện hàm xấp xỉ với 4 phân đoạn không đều nhau để tối ưu sai số giữa hàm

xấp xỉ và hàm *sin* lý tưởng. Hàm xấp xỉ với các hệ số tối ưu biểu diễn như công thức (3.10). Với hàm xấp xỉ này, sai số cực đại đạt được là 0,012 và do đó giảm được 6 bit từ nhớ của sin-LUT.

$$a(x) = \begin{cases} 1,4871x, & 0 \leq x < 0,363, \\ 1,1377x + 0,1277, & 0,363 \leq x < 0,598, \\ 0,7078x + 0,3839, & 0,598 \leq x < 0,804, \\ 0,2399x + 0,7601\,, & 0,804 \leq x < 1. \end{cases} \tag{3.10}$$

3.5. Đề xuất phương pháp cải tiến nén biên độ sin ứng dụng trong DDFS

3.5.1. Cơ sở và ý tưởng đề xuất

Các nghiên cứu gần đây nhằm đạt được một kiến trúc DDFS hiệu quả tập trung chủ yếu vào cải thiện hiệu năng của PAC. Khảo sát các kỹ thuật chuyển đổi pha thành biên độ ở trên có thể đưa ra một số đánh giá như sau:

Các giải pháp dựa trên kỹ thuật phân chia góc đầu vào cho phép giảm dung lượng sin-LUT tổng cộng. Tuy nhiên, kỹ thuật này vẫn chưa đạt được độ giảm lớn về dung lượng của sin-LUT. Vì vậy, khi đòi hỏi độ chính xác tần số cao thì phương pháp phân chia góc đầu vào vẫn cần một dung lượng sin-LUT khá lớn. Các phương pháp quay góc đạt được độ chính xác cao với chi phí phần cứng thấp tuy nhiên hạn chế về tốc độ do độ giữ chậm lớn và tốc độ chuyển tần số thấp. Trong khi đó, phương pháp xấp xỉ đa thức có thể đạt được độ chính xác và SFDR cao, nhưng quá trình lựa chọn các hệ số phức tạp và đòi hỏi các bộ nhân và bình phương trong xấp xỉ sử dụng các đa thức bậc cao do đó tăng chi phí phần cứng.

Các phương pháp nén biên độ *sin* là một giải pháp hiệu quả cho phép đạt

được một kiến trúc PAC có độ phức tạp thấp. Nó dựa trên việc kết hợp xấp xỉ đa thức với một sin-LUT sửa lỗi. Trong các kiến trúc đề xuất dựa trên giải pháp nén biên độ *sin* đã đề cập ở trên, các kết quả trong [68] và [69] đạt được tỷ số nén sin-LUT cao nhất. Tuy nhiên, có thể thấy kiến trúc của các đề xuất này vẫn sử dụng các mạch nhân do đó có độ phức tạp phần cứng cao. Ngoài ra, phương pháp đưa ra trong [68] và [69] dựa trên xấp xỉ phân đoạn tuyến tính không đều do đó làm cho kiến trúc phần cứng thực hiện sẽ phức tạp hơn. Từ những đánh giá đó có thể thấy các kiến trúc của PAC ở các nghiên cứu trong [68] và [69] có thể được cải thiện hơn nữa.

Ý tưởng đề xuất là tiếp tục lợi dụng ưu điểm tính đối xứng 1/4 của hàm *sin* và kết hợp với kỹ thuật phân chia góc. Đồng thời đề xuất một phương pháp nén biên độ *sin* mới để khắc phục những hạn chế trong [68] và [69]. Cụ thể, để đạt được tỷ số nén sin-LUT cao hơn và đồng thời kiến trúc phần cứng đơn giản hơn, trong chương này sẽ đề xuất một phương pháp xấp xỉ dựa trên xấp xỉ phân đoạn tuyến tính đều với số đoạn là một số lũy thừa của 2 và các hệ số góc là các số có dạng là tổng của các số lũy thừa của hai.

3.5.2. Đề xuất phương pháp nén biên độ sin

Để thuận tiện cho phân tích về mặt toán học, khoảng giá trị pha của tín hiệu đầu vào được qui ước là trong khoảng $[0, 1]$ thay vì là $[0, \pi/2]$ cho góc phần tư đầu tiên. Giả thiết là chỉ các giá trị pha của góc phần tư đầu tiên được lưu trữ trong sin-LUT. Công thức biểu diễn xấp xỉ như sau:

$$\sin(\frac{\pi}{2}x) \approx a(x) + \varepsilon(x) \tag{3.11}$$

Trong đó $a(x)$ là hàm xấp xỉ đề xuất và $\varepsilon(x)$ biểu diễn sai số xấp xỉ và nó được chứa trong một sin-LUT có kích thước nhỏ. Kích thước của sin-LUT

tùy thuộc vào sai số xấp xỉ. Để giảm kích thước của sin-LUT, hàm xấp xỉ cần đảm bảo sai số do xấp xỉ là nhỏ. Tuy nhiên, việc giảm kích thước sin-LUT phải dung hòa với độ phức tạp phần cứng cần đạt được. Hàm $a(x)$ có thể là một hàm phân đoạn tuyến tính hoặc một hàm bậc cao. Thông thường việc xấp xỉ bằng các hàm bậc cao sẽ đạt được độ chính xác cao hơn so với xấp xỉ bằng hàm phân đoạn tuyến tính. Tuy nhiên, khi thực hiện xấp xỉ bằng các hàm bậc cao sẽ đòi hỏi nhiều tài nguyên phần cứng hơn do cần phải thực hiện các phép nhân cho các hàm phi tuyến. Trong khi đó, các hệ thống truyền thông hiện đại đòi hỏi công suất thấp và đơn giản về phần cứng. Do vậy, xấp xỉ bằng các hàm tuyến tính phân đoạn cho thấy là một sự lựa chọn tốt đảm bảo công suất thấp và giảm độ phức tạp phần cứng.

Để đơn giản về phần cứng, phương pháp đề xuất sử dụng xấp xỉ tuyến tính phân đoạn đều, với số phân đoạn s là một số lũy thừa của 2. Hàm xấp xỉ $a(x)$ có dạng như công thức (3.12).

$$a(x) = \begin{cases} a_0 x + b_0, & 0 \leq x < \frac{1}{s}, \\ a_1 x + b_1, & \frac{1}{s} \leq x < \frac{2}{s}, \\ \quad \vdots & \\ a_{s-1} x + b_{s-1}, & \frac{s-1}{s} \leq x < 1. \end{cases} \tag{3.12}$$

Để đơn giản hơn nữa cấu trúc phần cứng thực thi, các hệ số $a_i (i = 0 \div s-1)$ có dạng là tổng của các số lũy thừa của 2 được biểu diễn như công thức (3.13), khi đó tránh việc sử dụng các mạch nhân trong cấu trúc phần cứng.

$$a_i = \sum_{k=0}^{M} p_k . 2^{-k}, \quad p_k = \{0, 1\} \tag{3.13}$$

Với mục đích đạt được tỷ số nén sin-LUT cao với mỗi một đoạn, trong phương pháp đề xuất sẽ thực hiện tối ưu về mặt toán học để tìm các hệ số

a_i và b_i tối ưu dựa trên tiêu chí là tối thiểu hóa giá trị cực đại của hàm sai số $\varepsilon(x) = \sin(\frac{\pi}{2}x) - a(x)$ vì giá trị cực đại này qui định số bit của từ nhớ trong sin-LUT. Đồng thời để tránh phức tạp trong thực thi phần cứng giá trị M trong công thức (3.13) cũng được giới hạn không quá 5 ($M \leq 5$). Ngoài ra, giá trị của hàm xấp xỉ luôn nhỏ hơn giá trị thật của hàm cần xấp xỉ ($\varepsilon(x) \geq 0$), điều này cho phép chỉ sử dụng các mạch cộng khi kết hợp mạch xấp xỉ với sin-LUT. Thực hiện một chương trình Matlab khảo sát cho các trường hợp với số phân đoạn khác nhau, giá trị sai số cực đại cũng như số bit giảm được trong từ nhớ sin-LUT của các trường hợp thể hiện trên Bảng 3.1. Trên Bảng 3.1, ký hiệu $\varepsilon_{\max}$ là sai số cực đại trên tất cả các phân đoạn và tương ứng với nó là số bit giảm trong từ nhớ của sin-LUT. Trong trường hợp sử dụng 2 và 4 phân đoạn thì số bit từ nhớ của sin-LUT chỉ giảm tương ứng là 3 và 5 bit. Khi tiếp tục tăng số phân đoạn lên 8 và 16 đoạn thì số bit giảm tương ứng là 7 và 8 bit. Tuy nhiên, sử dụng 16 phân đoạn sẽ tạo ra kiến trúc phần cứng phức tạp hơn khá nhiều so với sử dụng 8 phân đoạn trong khi chỉ giảm được 1 bit. Do vậy, để dung hòa giữa độ phức tạp phần cứng và số tỷ số nén của sin-LUT trường hợp sử dụng 8 phân đoạn là sự lựa chọn phù hợp để thực thi phần cứng.

Bảng 3.1: Lỗi cực đại và độ giảm từ nhớ sin-LUT với số phân đoạn khác nhau.

Số phân đoạn	$\varepsilon_{\max}$	Số bit giảm của từ nhớ trong LUT
2	0,0726(1/14)	3
4	0,0204 (1/49)	5
8	0,0076 (1/131)	7
16	0,0026 (1/385)	8

Các hệ số a_i, b_i tối ưu cho trường hợp xấp xỉ bằng 8 phân đoạn thể hiện trên Bảng 3.2. Độ chênh lệch cực đại ($\varepsilon_{\max}$) trong trường hợp này là 0,0076.

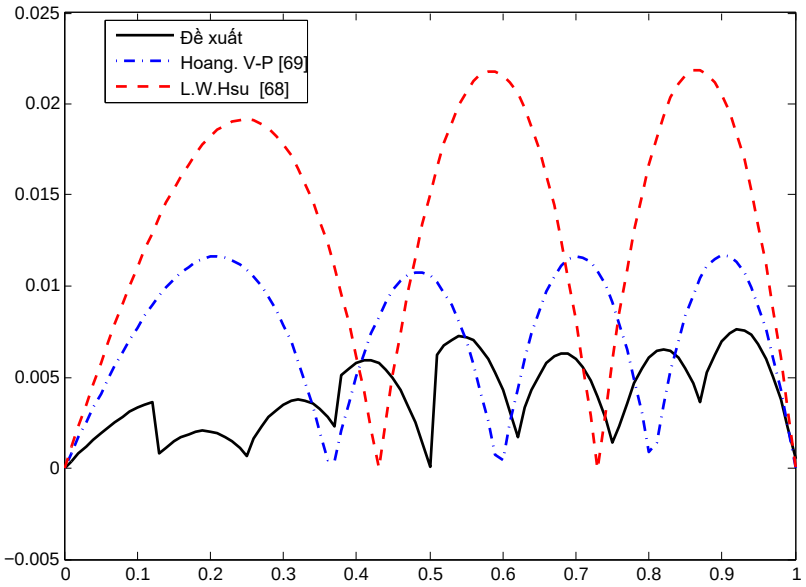
Bảng 3.2: Giá trị các hệ số tối ưu của hàm xấp xỉ tuyến tính 8 phân đoạn.

Đoạn	Khoảng giá trị đầu vào	a_i	b_i	$\varepsilon_{i\,\max}$
1	$0 < x \leq 0,125$	$2^0 + 2^{-1}$	0	0,0076
2	$0,125 < x \leq 0,25$	$2^0 + 2^{-1}$	0,007	0,0021
3	$0,25 < x \leq 0,375$	$2^0 + 2^{-2} + 2^{-3}$	0,038	0,0038
4	$0,375 < x \leq 0,5$	$2^0 + 2^{-2}$	0,082	0,0059
5	$0,5 < x \leq 0,625$	$2^0 + 2^{-5}$	0,186	0,0073
6	$0,625 < x \leq 0,75$	$2^{-1} + 2^{-2}$	0,360	0,0063
7	$0,75 < x \leq 0,875$	$2^{-2} + 2^{-3} + 2^{-4}$	0,595	0,0065
8	$0,875 < x \leq 1$	$2^{-3} + 2^{-4}$	0,812	0,0076

Hình 3.6 thể hiện lỗi xấp xỉ của phương pháp đề xuất so với các phương pháp trong [68] và [69]. Bảng 3.3 thể hiện sự so sánh độ chênh lệch cực đại và số bit có thể rút gọn trong độ dài từ nhớ của sin-LUT của phương pháp đề xuất so với các kết quả trong [68] và [69].

Bảng 3.3: So sánh độ chênh lệch cực đại và số bit giảm của từ nhớ trong sin-LUT.

Phương pháp	Maxdiff	Số bit giảm của từ nhớ trong LUT
L. W. Hsu [68]	0,0219 (1/46)	5
Hoang. V-P [69]	0,0120 (1/83)	6
Đề xuất	0,0076 (1/131)	7

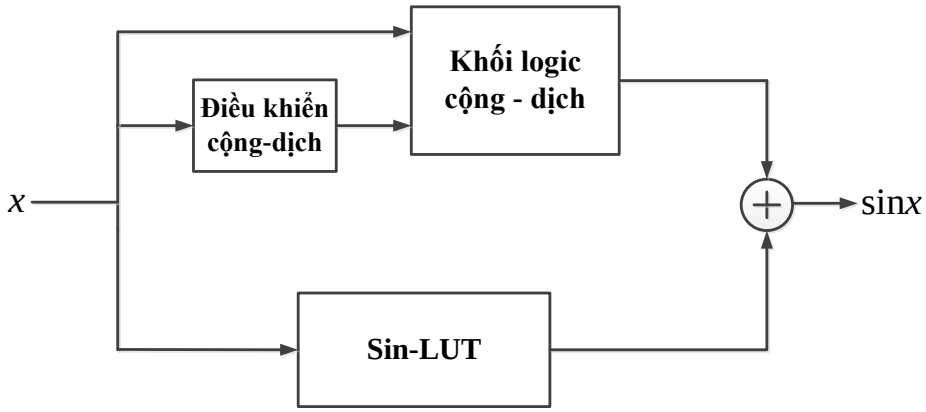


Hình 3.6: So sánh sai số của hàm xấp xỉ đề xuất với trong [68] và [69].

3.6. Kết quả tổng hợp và thực thi

Hình 3.7 là kiến trúc của bộ chuyển đổi pha thành biên độ sử dụng phương pháp xấp xỉ 8 phân đoạn được đề xuất ở trên. Trong đó, khối điều khiển cộng-dịch sẽ căn cứ vào giá trị pha đầu vào để tạo tín hiệu điều khiển khối logic cộng-dịch ứng với một trong 8 đoạn tuyến tính. Khối logic cộng-dịch thực hiện các thao tác dịch và cộng để tính toán hàm tối ưu 8 đoạn $a(x)$. Khối sin-LUT dùng để chứa giá trị độ chênh lệch $\varepsilon(x)$. Một bộ cộng được sử dụng để tạo tín hiệu sin đầu ra.

Để đạt được tỷ số nén cao nhất có thể, trong phương pháp đề xuất có lợi dụng tính chất đối xứng $1/4$ của sóng sin và sử dụng phương thức phân chia LUT thô/tinh. Vì vậy, chỉ có giá trị pha của góc $1/4$ đầu tiên của hàm sin được lưu trữ và sin-LUT được phân chia thành sin-LUT thô và sin-LUT tinh. Kiến trúc đề xuất đã được thực thi trên thiết bị FPGA Xilinx Spartan-3E và được tổng hợp bởi công cụ Xilinx ISE 12.4 Suite.



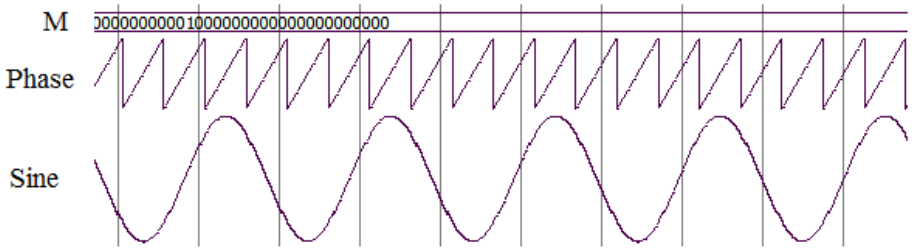
Hình 3.7: Kiến trúc bộ chuyển đổi pha-biên độ của phương pháp đề xuất.

Bộ DDFS theo phương pháp đề xuất được thiết kế với bộ tích lũy pha có độ dài $N = 32$ bit, 12 bit pha và 11 bit biên độ sin đầu ra (tương tự với độ chính xác của các thiết kế trong [68] và [69]). Bảng 3.4 thể hiện sự so sánh kết quả thực thi của bộ DDFS đề xuất với các nghiên cứu tương tự trước đó. Phương pháp đề xuất đã đạt được tỷ số nén LUT cao hơn và độ phức tạp phần cứng nhỏ hơn so với các phương pháp trước đó. Mặc dù số phân đoạn trong phương pháp đề xuất là lớn hơn so với trong [68] và [69] nhưng vẫn đạt được cấu trúc phần cứng đơn giản hơn là do các đoạn được chia đều và là một số lũy thừa của 2. Do đó, trong thực thi phần cứng chỉ sử dụng 3 bit trọng số lớn nhất của pha đầu vào để mã hóa, điều này cho phép không cần sử dụng các mạch so sánh. Hơn nữa, các hệ số của các đoạn có dạng là tổng các số lũy thừa của 2 nên trong cấu trúc phần cứng thực thi chỉ cần các mạch cộng- dịch mà không cần sử dụng mạch nhân. Phương pháp đề xuất có thể cho phép tăng số phân đoạn để đạt được độ chính xác và tỷ số nén sin-LUT cao hơn. Tuy nhiên, điều đó sẽ làm tăng độ phức tạp phần cứng. Để dung hòa giữa độ chính xác và tỷ số nén LUT với độ phức tạp của phần cứng, nghiên cứu sinh đề xuất lựa chọn giải pháp xấp xỉ với 8 phân đoạn.

Hình 3.8 là mô tả dạng sóng đầu ra với mô phỏng bằng Modelsim.

Bảng 3.4: So sánh phương pháp đề xuất với các trong phương pháp trước đó.

Phương pháp	Sunderland	Độ chênh lệch sin-pha	Độ chênh lệch truyền thống	Hsu [68]	Hoang [69]	Đề xuất
Kích thước LUT (bits)	4096	3854	2688	2560	1536	**1408**
Tỷ số nén Sin-LUT	44 : 1	50 : 1	67 : 1	70 : 1	117.3 : 1	**128:1**
Slices	144	146	133	176	136	**85**



Hình 3.8: Dạng sóng tín hiệu sin đầu ra mô phỏng trên Modelsim.

3.7. Kết luận chương 3

Chương này đã trình bày một phương pháp tính toán hàm *sin* ứng dụng cho DDFS dựa trên phương thức độ chênh lệch tuyến tính với hàm tuyến tính phân đoạn đều có các hệ số tối ưu nhằm đạt được tỷ số nén LUT cao và cấu trúc phần cứng đơn giản. Một kiến trúc DDFS dựa trên phương pháp đề xuất đã được tổng hợp và thực thi. Kiến trúc đề xuất đã đạt được tỷ số nén LUT là 128:1 với độ phức tạp phần cứng nhỏ. Vì vậy, kiến trúc đề xuất là phù hợp với các ứng dụng đòi hỏi độ phức tạp phần cứng thấp.

Chương 4
THIẾT KẾ PHẦN CỨNG TÍNH TOÁN CÁC HÀM TOÁN HỌC DỰA TRÊN XẤP XỈ TUYẾN TÍNH HAI MỨC

4.1. Giới thiệu chung

Các hàm toán học như hàm sin, logarithm, hàm mũ, hàm nghịch đảo ... được sử dụng rộng rãi trong nhiều lĩnh vực như truyền thông, đồ họa máy tính, khoa học tính toán và xử lý tín hiệu số. Thực thi các hàm toán học nói trên có thể thực hiện bằng các chương trình phần mềm. Tuy nhiên, tính toán các hàm toán học bằng phần mềm sẽ có tốc độ tính toán chậm. Vì vậy, nhiều nghiên cứu đã tập trung thực hiện tính toán các hàm toán học bằng các phần cứng chuyên dụng.

Một số các phương pháp khác nhau đã được nghiên cứu và đề xuất để thực thi phần cứng tính toán các hàm toán học. Các phương pháp này bao gồm: thuật toán CORDIC [20], xấp xỉ đa thức [70–72], xấp xỉ hữu tỷ [22] và các phương pháp dựa trên bảng [9, 73–75]. Thuật toán CORDIC dựa trên kiến trúc lặp do đó có độ giữ chậm lớn nên không phù hợp với các ứng dụng thời gian thực. Phương pháp xấp xỉ hữu tỷ có độ chính xác khá cao tuy nhiên đòi hỏi độ phức tạp phần cứng cao. Ngày nay, với sự phát triển của công nghệ mạch tích hợp cho phép dung lượng bộ nhớ lớn thì các phương pháp dựa trên bảng được sử dụng khá phổ biến. Tuy nhiên, các phương pháp dựa trên bảng có một nhược điểm là khi độ rộng toán hạng đầu vào lớn đòi hỏi dung lượng

lớn, điều đó đòi hỏi nhiều tài nguyên phần cứng cũng như khó khăn trong thực hiện của các công cụ tổng hợp.

Mặt khác, với nhu cầu ngày càng tăng nhanh của các thiết bị điện tử thông minh không dây, các thiết bị cầm tay, điện thoại di động, yêu cầu về hệ thống xử lý tín hiệu số công suất thấp và nhỏ gọn ngày càng đặt ra cấp thiết. Với các ứng dụng này thì không chỉ yêu cầu về tốc độ cao mà còn đòi hỏi cao về công suất thấp, tài nguyên tiêu tốn ít. Tính toán các hàm toán học là những thao tác quan trọng trong các ứng dụng xử lý tín hiệu số, chúng chiếm một phần lớn tài nguyên phần cứng cũng như quyết định đến tốc độ xử lý của các bộ xử lý. Vì vậy đòi hỏi có các lõi tính toán hiệu quả về tài nguyên, tốc độ và công suất tiêu thụ.

Chương này đề xuất một phương pháp thực thi phần cứng cho tính toán các hàm toán học được sử dụng phổ biến trong các ứng dụng xử lý tín hiệu số. Phương pháp đề xuất dựa trên xấp xỉ các hàm toán học bằng hai mức theo phương pháp xấp xỉ tuyến tính phân đoạn. Các kết quả thực thi cho thấy phương pháp đề xuất đạt được hiệu quả về tốc độ thực thi.

4.2. Các nghiên cứu liên quan

Hầu hết các phương pháp thiết kế phần cứng tính toán các hàm toán học hiện nay dựa trên việc sử dụng bảng kết hợp với sử dụng các mạch số học đơn giản như các mạch cộng và nhân. Các phương pháp dựa trên bảng có thể được chia thành 3 dạng [76]: các phương pháp thiên về tính toán (Compute-bound Methods), các phương pháp thiên về bảng (Table bound Methods) và phương pháp trung gian (in-between Methods) được coi là dung hòa giữa hai phương pháp trên.

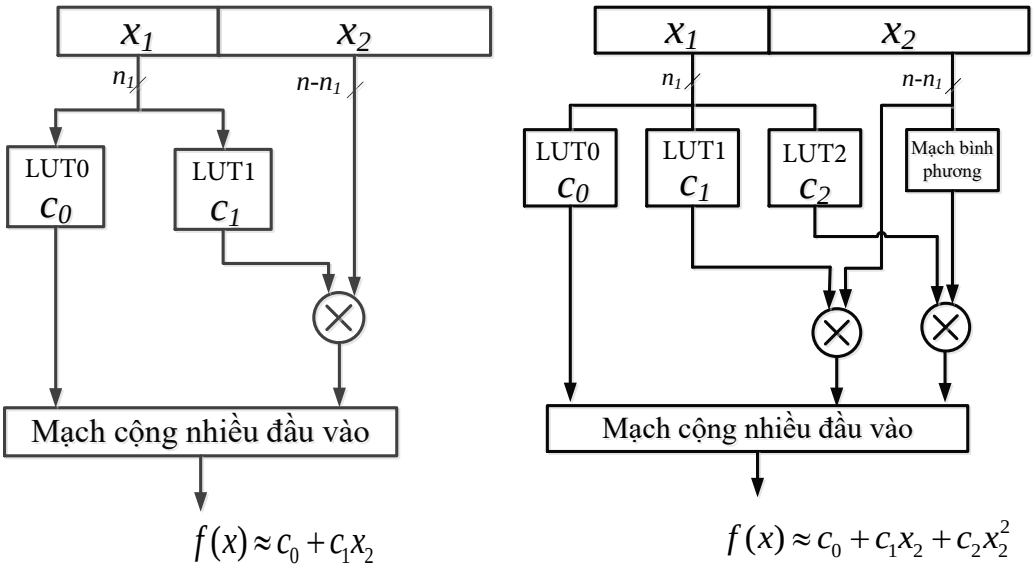
Các phương pháp thiên về tính toán sử dụng một bảng có kích thước nhỏ để chứa các tham số sau đó được sử dụng cho xấp xỉ đa thức bậc 3 hoặc bậc cao hơn [73]. Các phương pháp này có thể đạt được độ chính xác tốt, tuy nhiên nó có độ phức tạp phần cứng cao do sử dụng xấp xỉ bằng các đa thức bậc cao.

Các phương pháp thiên về sử dụng bảng sử dụng bảng có kích thước lớn kết hợp với một hoặc một vài mạch cộng, điển hình cho dạng này là các phương pháp BTM [16], SBTM [17] và MTM [19]. Các phương pháp này nhìn chung có tốc độ cao nhưng nhược điểm là khi kích thước toán hạng tăng thì kích thước của bảng sẽ rất lớn.

Các phương pháp trung gian sử dụng kích thước bảng trung bình và giảm đáng kể số lượng tính toán. Trong phương pháp này, hàm cần tính toán sẽ được xấp xỉ bằng các đa thức phân đoạn. Trong đó, khoảng giá trị của biến đầu vào sẽ dược chia thành các đoạn đều nhau và sử dụng xấp xỉ đa thức trên từng phân đoạn.

Điển hình cho các phương pháp trên là các nghiên cứu trong [9] và [74,75]. Trong đó đầu vào x có n bit được chia thành hai phần x_1 có n_1 bit trọng số lớn nhất và x_2 có $n - n_1$ bit có trọng số bé nhất. Trong đó x_1 được sử dụng để lựa chọn các hệ số của đa thức xấp xỉ từ các bảng chứa các hệ số. Số các phân đoạn là 2^{n_1} tương ứng với số đầu vào của các bảng hệ số. Hình 4.1 mô tả kiến trúc cho xấp xỉ sử dụng đa thức bậc 1 và bậc 2 bao gồm các LUT chứa hệ số và các mạch số học cần thiết.

Trong [74], E. George Walters và M. J. Schulte đã trình bày cách xấp xỉ các hàm bằng hàm bậc nhất và bậc hai. Trong đó sử dụng các bộ nhân và bộ bình phương rút gọn. Các hệ số của đa thức xấp xỉ được tính toán dựa trên



Hình 4.1: Nội suy bậc 1 và bậc 2 các hàm.

tối ưu các tham số như độ rộng bit của bộ nhân rút gọn và bộ bình phương rút gọn, số phân đoạn sử dụng với một sai số xác định.

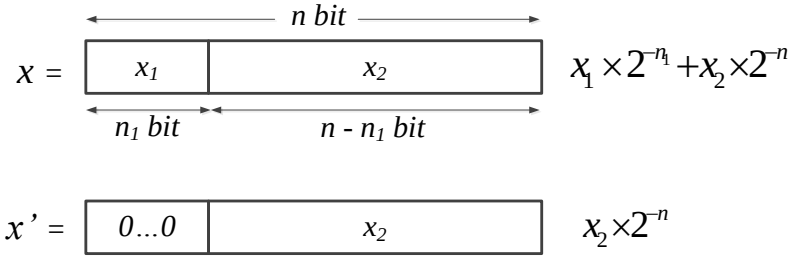
J.A. Pineiro và các cộng sự trong [9] đã thực hiện tính toán các hàm dựa trên nội suy bậc hai. Phương pháp tìm các hệ số của hàm xấp xỉ bậc hai của Pineiro dựa trên một thuật toán lặp gồm 3 bước nhằm tối thiểu hóa sai số đồng thời tối ưu số phân đoạn và độ rộng bit của các từ nhớ trong LUT chứa các hệ số cũng như độ rộng toán hạng của các mạch số học.

S. F. Hsiao và các cộng sự trong [75] đã thực thi phần cứng tính toán các hàm toán học dựa trên hai bước. Trong bước xấp xỉ đầu tiên hàm được xấp xỉ bằng phân đoạn tuyến tính đều. Sau đó các hàm sai số của bước đầu tiên được tính toán thông qua một hàm sai số chuẩn hóa trong bước xấp xỉ thứ hai. Các phân tích sai số và thuật toán nhằm tối ưu các tham số được áp dụng để tìm hệ số tối ưu cho xấp xỉ hàm.

4.3. Đề xuất phương pháp xấp xỉ

Trong phương pháp đề xuất hàm được xấp xỉ bởi hai mức. Trong mức đầu tiên sử dụng xấp xỉ phân đoạn tuyến tính đều và mức thứ hai thực hiện xấp xỉ hàm sai số do xấp xỉ của mức đầu tiên sử dụng phương pháp xấp xỉ phân đoạn tuyến tính đối xứng.

4.3.1. Xấp xỉ mức 1

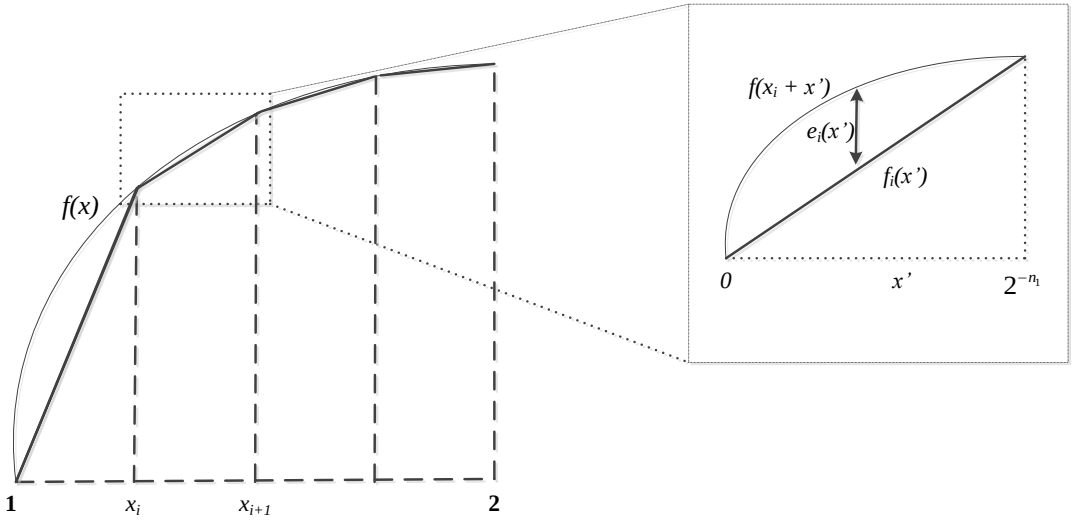


Hình 4.2: Phân chia biến đầu vào thành hai phần.

Trong mức đầu tiên đầu vào x có n bit phần thập phân được chia thành hai phần x_1 và x_2 với độ dài bit tương ứng là n_1 và $n - n_1$ như biểu diễn trên Hình 4.2. Trong bước xấp xỉ đầu tiên khoảng giá trị của x được chia thành 2^{n_1} đoạn dựa trên x_1 như thấy trên Hình 4.3 cho trường hợp $n_1 = 2$ và khoảng giá trị đầu vào là $[1,\ 2)$. Với đoạn thứ i, $x \in [x_i,\ x_{i+1})$ hai giá trị $f(x_i)$ và $f(x_{i+1})$ được sử dụng trong xấp xỉ ở mức thứ nhất để xấp xỉ hàm bằng một hàm $f_i(x')$, ở đây x' nhận giá trị trong khoảng $[0,\ 2^{-n_1})$.

$$f_i(x') = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \cdot x' + f(x_i) \tag{4.1}$$

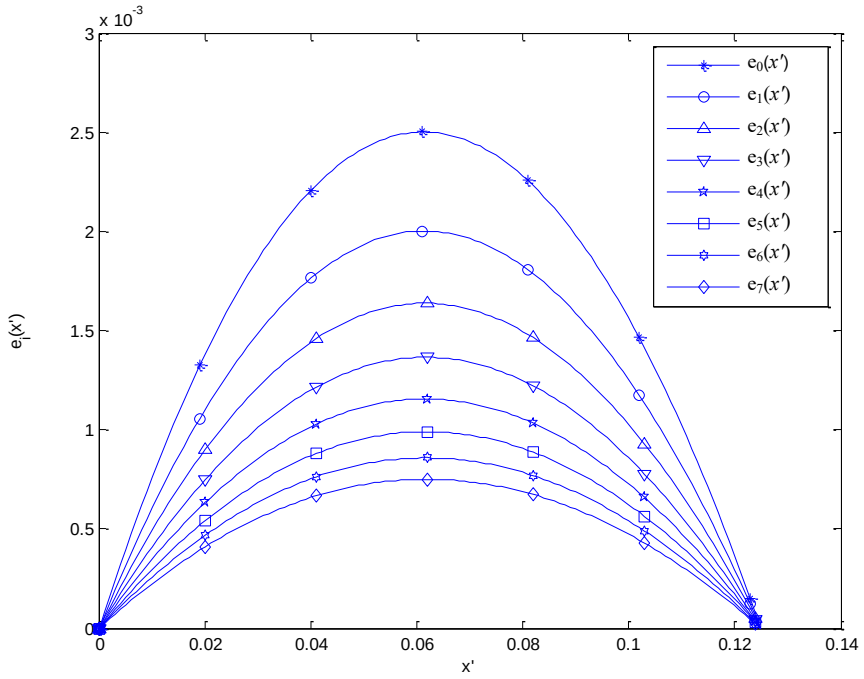
Như mô tả trên Hình 4.3, xấp xỉ ở mức 1 sẽ tạo ra lỗi xấp xỉ là độ chênh lệch giữa hàm xấp xỉ ban đầu và hàm thực. Độ chênh lệch này là một hàm



Hình 4.3: Xấp xỉ mức thứ nhất của hàm $f(x)$ trong đoạn thứ i ($n_1 = 2$).

lỗi $e_i(x')$, $i = 0, 1, \ldots, 2^{n_1}$ có biểu diễn như công thức (4.2).

$$e_i(x') = f(x_i + x') - f_i(x'), \qquad 0 \leq x' < 2^{-n_1}. \tag{4.2}$$



Hình 4.4: Hàm lỗi do xấp xỉ mức 1 đối với hàm $\log_2(x)$ với $n_1 = 3$.

4.3.2. Xấp xỉ mức 2

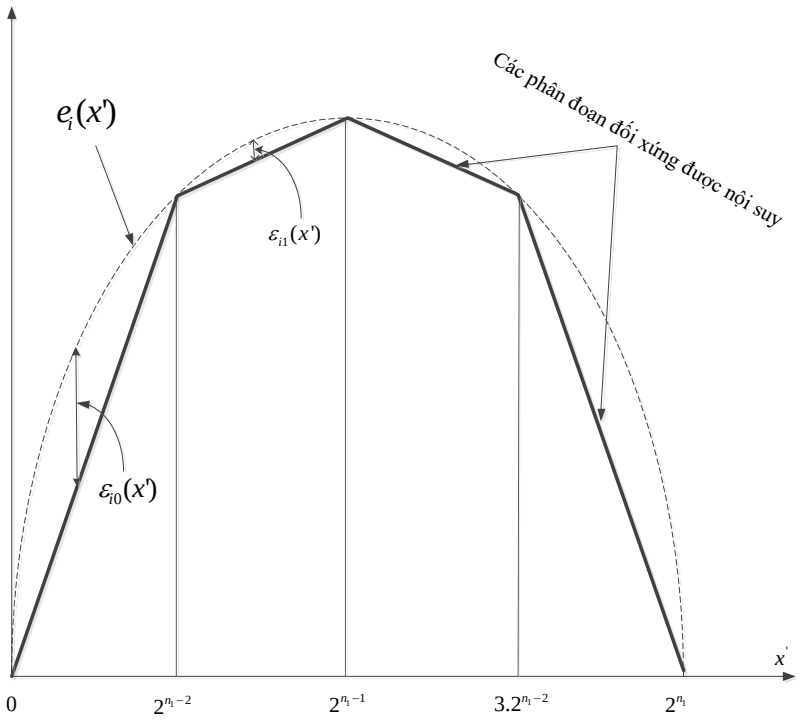
Trong tầng xấp xỉ thứ hai các hàm chênh lệch $e_i(x')$ được xấp xỉ bằng phương pháp xấp xỉ phân đoạn tuyến tính sử dụng nội suy đối xứng. Có thể thấy rằng các hàm $e_i(x')$ có dạng như các hàm parabol. Ví dụ như, các hàm chênh lệch của hàm $\log_2(x)$ có dạng như trên Hình 4.4 với trường hợp xấp xỉ ban đầu của tám đoạn, tức là $n_1 = 3$. Các hàm này có dạng parabol vì nó được tạo ra bởi phép trừ của hàm ban đầu với hàm xấp xỉ tuyến tính vì vậy nó sẽ là biểu diễn của các số hạng bậc cao còn lại. Vì vậy, các hàm chênh lệch này có dạng đối xứng qua trục $x' = 2^{-n_1-1}$. Do đó, nếu ta chỉ xấp xỉ nửa khoảng giá trị của $x' \in [0,\, 2^{-n_1-1})$ bằng các đoạn tuyến tính, nửa còn lại là các đoạn được nội suy đối xứng qua trục $x' = 2^{-n_1-1}$ thì sai số xấp xỉ là không thay đổi. Điều này sẽ cho phép thực thi phần cứng sẽ đơn giản hơn vì khi đó chỉ sử dụng một hoặc một số bit có trọng số lớn nhất (MSB) của dữ liệu đầu vào để nội suy ra các đoạn đối xứng ở nửa còn lại trong khoảng giá trị của x'. Hình 4.5 mô tả xấp xỉ ở mức thứ hai cho hàm $e_i(x')$ trong trường hợp sử dụng hai cặp đoạn đối xứng.

Trong phương pháp đề xuất, nửa khoảng giá trị của $x', x' \in [0,2^{-n_1-1})$, được chia thành s đoạn con, trong mỗi đoạn con thứ j, $j = 0,\, 1,\, \ldots,\, s-1$, hàm lỗi $e_i(x')$ được xấp xỉ bằng một đoạn tuyến tính có các hệ số là a_{ij} và b_{ij}, khi đó lỗi xấp xỉ có dạng như công thức (4.3).

$$\varepsilon_{\mathrm{ij}}(x') \;=\; e_i(x') \;-\; (a_{\mathrm{ij}} \times x' \;+\; b_{\mathrm{ij}}) \tag{4.3}$$

Lỗi cực đại cho xấp xỉ trên đoạn j ký hiệu là $\varepsilon_{ij\,\max}$

$$\varepsilon_{ij\,\max} = \max\{\varepsilon_{ij}(x')\} \tag{4.4}$$



Hình 4.5: Xấp xỉ hàm hàm $e_i(x')$ bằng hai cặp đoạn đối xứng.

Hơn nữa, để đơn giản phần cứng thực thi các hệ số a_{ij} và b_{ij} được ấn định có dạng như công thức (4.5), trong đó các hệ số a_{ij} là tổng các số lũy thừa của 2 khi đó có thể thực thi bằng các phép dịch và cộng tránh việc sử dụng mạch nhân phức tạp.

$$\begin{aligned} a_{ij} &= \sum 2^{-N}, \qquad N \in \mathbb{N} \\ b_{\mathrm{ij}} &= B \times 2^{-(n-n_1)}, \;\; B \in \mathbb{N} \end{aligned} \tag{4.5}$$

Thuật toán đề xuất tìm các giá trị hệ số a_{ij} và b_{ij} tối ưu sao cho $\varepsilon_{ij\,\max}$ nhận giá trị cực tiểu trong đoạn thứ j được mô tả như Bảng 4.1. Sử dụng một chương trình Matlab thực hiện thuật toán trên để tính toán các hệ số a_{ij} và b_{ij} tối ưu cho các hàm $\log_2(x)$, $\sin x$, $1/x$, $\sqrt{x}$, $1/\sqrt{x}$ và 2^x với các khoảng giá trị đầu vào được chuẩn hóa như thể hiện trên Bảng 4.2 và tương ứng với xấp xỉ mức 1 gồm 8 đoạn ($n_1 = 3$) và ở xấp xỉ mức 2 sử dụng xấp

xỉ phân đoạn nội suy đối xứng trong đó mỗi hàm $e_i(x')$ được xấp xỉ bởi hai cặp đoạn đối xứng. Bảng 4.3 thể hiện kết quả các hệ số a_{ij} và b_{ij} tối ưu cho hàm $\log_2(x)$.

Bảng 4.1: Thuật toán tìm hệ số tối ưu của xấp xỉ mức 2

1:	*BEGIN* : Khởi tạo mảng giá trị a_{ij} Khởi tạo mảng giá trị b_{ij} Khởi tạo đoạn giá trị j
2:	for $p = 1 : length(a_{ij})$ for $q = 1 : length(b_{ij})$ for $k = 1 : length(x')$ tính $\varepsilon_{ij}(x')$ theo công thức (4.3) end end end
3:	for $p = 1 : length(a_{ij})$ for $q = 1 : length(b_{ij})$ tính $\varepsilon_{\mathrm{ij\,max}} = \max(\varepsilon_{\mathrm{ij}}(x'))$ end end
4:	Tính $\min(\varepsilon_{\mathrm{ij\,max}})$
5:	*END* Xuất giá trị a, b tương ứng với $\min(\varepsilon_{\mathrm{ij\,max}})$

Bảng 4.2: Khoảng giá trị đầu vào chuẩn hóa của các hàm

Hàm	$\log_2(x)$	$\sin(x)$	$1/x$	$\sqrt{x}$	$1/\sqrt{x}$	2^x
Khoảng giá trị đầu vào chuẩn hóa	[1, 2)	[0, 1)	[1, 2)	[1, 2)	[1, 2)	[0, 1)

4.4. Kiến trúc phần cứng và kết quả thực thi

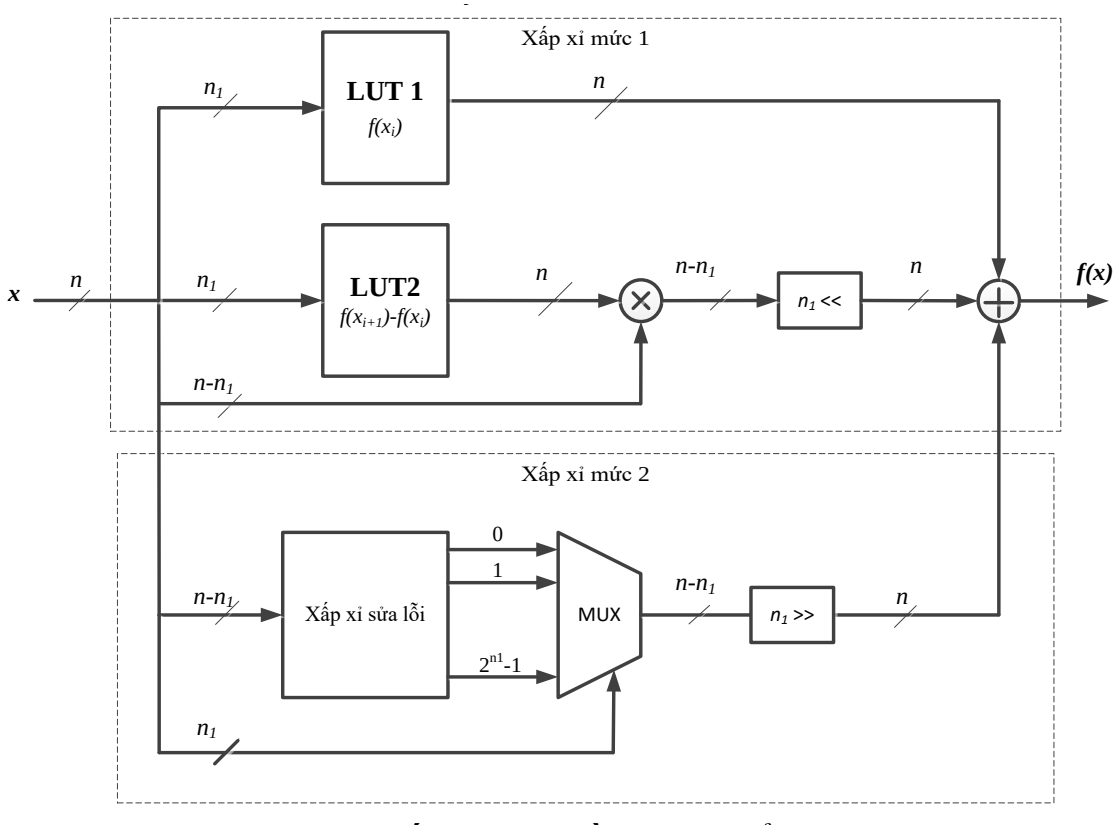
Kiến trúc tổng quát thực thi các hàm toán học của phương pháp đề xuất có dạng như Hình 4.6. Trong mức xấp xỉ thứ nhất thực hiện tính toán hàm $f_i(x')$ như công thức (4.1). Trong đó các giá trị hàm $f(x_i)$ và $f(x_{i+1}) - f(x_i)$

Bảng 4.3: Các hệ số a_{ij} và b_{ij} tối ưu ở mức xấp xỉ thứ 2 của hàm $\log_2(1+x)$.

$e_0(x')$	a_{00}	b_{00}	a_{01}	b_{01}
	2^{-4}	58.2^{-20}	2^{-5}	791.2^{-20}
$e_1(x')$	a_{10}	b_{10}	a_{11}	b_{11}
	$2^{-4}+2^{-6}+2^{-9}$	72.2^{-20}	$2^{-6}+2^{-9}$	1024.2^{-20}
$e_2(x')$	a_{20}	b_{20}	a_{21}	b_{21}
	$2^{-5}+2^{-7}$	65.2^{-20}	$2^{-7}+2^{-8}$	986.2^{-20}
$e_3(x')$	a_{30}	b_{30}	a_{31}	b_{31}
	$2^{-5}+2^{-9}$	48.2^{-20}	$2^{-7}+2^{-9}+2^{-10}$	775.2^{-20}
$e_4(x')$	a_{40}	b_{40}	a_{41}	b_{41}
	2^{-5}	0	$2^{-7}+2^{-10}$	672.2^{-20}
$e_5(x')$	a_{50}	b_{50}	a_{51}	b_{51}
	$2^{-6}+2^{-7}$	40.2^{-20}	2^{-7}	561.2^{-20}
$e_6(x')$	a_{60}	b_{60}	a_{61}	b_{61}
	2^{-6}	88.2^{-20}	2^{-7}	427.2^{-20}
$e_7(x')$	a_{70}	b_{70}	a_{71}	b_{71}
	2^{-6}	51.2^{-20}	2^{-8}	512.2^{-20}

được lưu vào trong bảng LUT sử dụng để tính toán hàm $f_i(x')$. Phép chia cho $x_{i+1}-x_i$ trong công thức (4.1) được thay thế bởi một phép dịch bởi vì nó là một số lũy thừa của 2. Ở mức xấp xỉ thứ 2, thực hiện xấp xỉ các hàm lỗi bằng phương pháp xấp xỉ phân đoạn tuyến tính có đối xứng. Khối xấp xỉ sửa lỗi sẽ bao gồm 2^{n_1} khối xấp xỉ tương ứng với 2^{n_1} hàm $e_i(x')$.

Dựa trên kiến trúc tổng quát được đề xuất, các thực thi phần cứng để tính toán các hàm $\log_2(x)$, $\sin x$, $1/x$, $\sqrt{x}$, $1/\sqrt{x}$ và 2^x với định dạng 23 bit thập



Hình 4.6: Kiến trúc phần cứng tổng quát.

phân đầu vào và đầu ra 23 bit đã được thực hiện. Về nguyên tắc khi tăng số phân đoạn ở mức xấp xỉ thứ nhất (n_1 lớn) thì sẽ đạt được độ chính xác cao hơn, tuy nhiên điều đó sẽ dẫn đến tăng độ phức tạp phần cứng. Vì vậy, để dung hòa giữa độ phức tạp phần cứng và độ chính xác nhận được, kiến trúc thực hiện với xấp xỉ mức 1 bằng 8 phân đoạn ($n_1 = 3$). Ngoài ra trong mức xấp xỉ thứ hai, mỗi hàm lỗi $e_i(x')$ được xấp xỉ bởi phân đoạn tuyến tính của hai cặp đoạn đối xứng. Các kiến trúc đề xuất đã được mô hình trên ngôn ngữ VHDL và thực thi trên thiết bị FPGA Xilinx Virtex 6 với công cụ tổng hợp Xilinx ISE 12.4 Suite. Kết quả thực thi trên FPGA của các kiến trúc đề xuất thể hiện trên Bảng 4.4.

Bảng 4.5 thể hiện kết quả tổng hợp bằng Synopsys Design Compiler cho

Bảng 4.4: Kết quả thực thi của các kiến trúc đề xuất trên FPGA Xilinx Virtex 6

Hàm	$\log_2(x)$	$\sin(x)$	$1/x$	$\sqrt{x}$	$1/\sqrt{x}$	2^x
Slices	789	755	1005	897	894	885
Delay(ns)	6,79	6,76	7,28	6,78	7,33	7,73
ADP($\times 10^3$)	5,42	5,10	7,32	6,08	6,55	6,84

Bảng 4.5: Kết quả tổng hợp và so sánh trên ASIC

Hàm	Phương pháp	Diện tích (μm^2)	Diện tích chuẩn hóa	Độ trễ (ns)	Độ trễ chuẩn hóa
$\log_2(x)$	Đề xuất	26695	1	3,98	1
	Trong [75]	24023	0,90	12,69	3,19
$sin(x)$	Đề xuất	24452	1	3,86	1
	Trong [75]	22890	0,94	13,08	3,39
$1/x$	Đề xuất	32726	1	3,99	1
	Trong [75]	24137	0,74	12,63	3,17
$\sqrt{x}$	Đề xuất	27213	1	3,83	1
	Trong [75]	16043	0,59	12,03	3,14
$1/\sqrt{x}$	Đề xuất	28752	1	4,00	1
	Trong [75]	22317	0,78	12,83	3,21
2^x	Đề xuất	30887	1	3,86	1
	Trong [75]	18913	0,61	12,16	3,15

các hàm khác nhau của phương pháp đề xuất với mục đích tối thiểu tài nguyên trên thư viện công nghệ CMOS 90 nm. Bảng 4.5 đồng thời cũng đưa ra sự so sánh của phương pháp đề xuất với các kết quả trong [75]. Có thể thấy rằng, phương pháp đề xuất đã đạt được tốc độ tính toán nhanh hơn đáng kể so với kết quả trong [75]. Cụ thể, độ trễ giảm hơn 3 lần ở tất cả các hàm. Tuy nhiên, tài nguyên tiêu chiếm của phương pháp đề xuất cũng lớn hơn 1,1 đến 1,6 lần so với [75]. Hơn nữa, độ chính xác trong [75] là 23 bit,

tức là sai số khoảng $1,2.10^{-7}$, trong khi đó độ chính xác của phương pháp đề xuất là thấp hơn và sẽ được phân tích trong mục sau.

4.5. Phân tích sai số

Để đánh giá độ chính xác của phương pháp đề xuất, các sai số gây ra bởi các thành phần phần cứng trong sơ đồ thực thi và tác động của chúng tới đầu ra được phân tích. Các sai số này bao gồm sai số lượng tử do độ rộng bit hạn chế của các từ nhớ trong ROM (ký hiệu là ε_{ROM}), sai số gây ra bởi bộ nhân rút gọn (ε_{MUL}), sai số xấp xỉ do xấp xỉ mức hai gây ra (ε_{apx2}) và sai số do làm tròn của bộ cộng cuối cùng (ε_{ADD}). Vì vậy, lỗi tổng cộng sẽ là:

$$\varepsilon_{total} = \varepsilon_{ROM} + \varepsilon_{MUl} + \varepsilon_{ADD} + \varepsilon_{apx2} \tag{4.6}$$

Ký hiệu B_F, B_M và B_A là độ rộng bit tương ứng của các bộ nhớ ROM, bộ nhân rút gọn và bộ cộng, chúng ta có thể biểu diễn lỗi do các thành phần phần cứng này gây ra như trên Bảng 4.6.

Bảng 4.6: Sai số của các module phần cứng.

Module	ROM	Bộ nhân rút gọn	Bộ cộng
Độ rộng bit	B_F	B_M	B_A
ε_{Modul}	2^{-B_F-1}	2^{-B_M+1}	2^{-B_A-1}
ε_{ROM}	2^{-B_F-1}		
ε_{MUL}	$2^{-B_F} + 2^{-B_M+1}$		
ε_{ADD}			2^{-B_A-1}

Lỗi xấp xỉ ở mức 2, ε_{apx2}, được xác định là lỗi tuyệt đối lớn nhất do xấp xỉ

ở mức 2 của tất cả các hàm $e_i(x')$. Trong phương pháp đề xuất, phần cứng được thực thi với xấp xỉ mức 1 với 8 phân đoạn ($n_1 = 3$) tương ứng tạo ra 8 hàm lỗi và mỗi hàm lỗi này được xấp xỉ bởi phân đoạn tuyến tính của hai cặp đoạn đối xứng. Độ rộng bit của ROM và bộ cộng là 23 bit ($B_F = B_A = 23$) và độ rộng bit của bộ nhân rút gọn là 20 bit ($B_M = 20$). Khi đó, các thành phần lỗi và lỗi tổng cộng tương ứng với hàm khác nhau được biểu diễn như trên Bảng 4.7.

Bảng 4.7: Các sai số thành phần và sai số tổng cộng của thực thi các hàm.

Hàm	$\log_2(x)$	$\sin x$	$1/x$	$\sqrt{x}$	$1/\sqrt{x}$	2^x
ε_{ROM}	2^{-24}	2^{-24}	2^{-24}	2^{-24}	2^{-24}	2^{-24}
ε_{MUL}	$2^{-23}+2^{-19}$	$2^{-23}+2^{-19}$	$2^{-23}+2^{-19}$	$2^{-23}+2^{-19}$	$2^{-23}+2^{-19}$	$2^{-23}+2^{-19}$
ε_{ADD}	2^{-24}	2^{-24}	2^{-24}	2^{-24}	2^{-24}	2^{-24}
ε_{apx2}	$9,85.10^{-5}$	$7,95.10^{-5}$	$7,93.10^{-5}$	$2,04.10^{-5}$	$6,17.10^{-5}$	$8,79.10^{-5}$
ε_{total}	$1,01.10^{-4}$	$8,16.10^{-5}$	$8,14.10^{-5}$	$2,25.10^{-5}$	$6,38.10^{-5}$	$9,01.10^{-5}$

4.6. Kết luận chương 4

Chương này đã trình bày một phương pháp tính toán các hàm toán học được ứng dụng phổ biến trong xử lý tín hiệu số dựa trên xấp xỉ hai mức, mức xấp xỉ thứ nhất dựa trên phương pháp xấp xỉ phân đoạn tuyến tính đều và mức xấp xỉ thứ hai là bước xấp xỉ hàm sai số gây ra bởi mức đầu tiên theo phương pháp xấp xỉ phân đoạn tuyến tính có đối xứng. Các thực thi một số hàm toán học điển hình theo phương pháp đề xuất đã cho thấy hiệu quả về tốc độ thực thi. Vì vậy, kiến trúc đề xuất là phù hợp với các ứng dụng đòi hỏi tốc độ tính toán cao.

Chương 5

THIẾT KẾ PHẦN CỨNG TÍNH TOÁN CÁC HÀM TOÁN HỌC SỬ DỤNG LOGIC NGẪU NHIÊN VÀ XẤP XỈ PHÂN ĐOẠN TUYẾN TÍNH ĐỀU

5.1. Khái quát về tính toán ngẫu nhiên

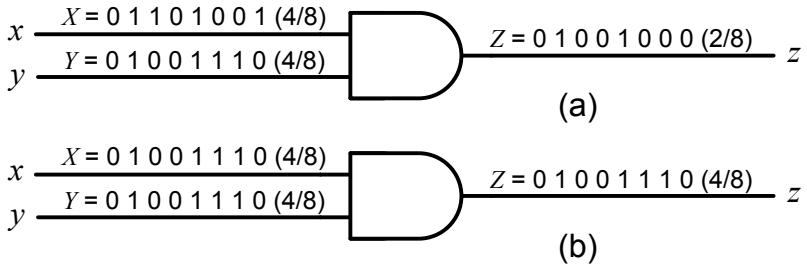
5.1.1. Giới thiệu

Tính toán ngẫu nhiên (SC: Stochastic computing), được đề xuất đầu tiên từ những năm 1960 bởi Gaines trong [77] và gần đây được quan tâm trở lại do việc ứng dụng vào thực thi các đơn vị số học với chi phí rất thấp và lỗi chấp nhận được. Đặc tính cơ bản của SC là một số được biểu diễn bởi một chuỗi bit và được xử lý bởi các mạch rất đơn giản. Một số thực x được biểu diễn bởi một số ngẫu nhiên (SN: Stochastic Numbers) là một chuỗi bit X chứa N_1 bit 1 và N_0 bit 0, biểu diễn số ngẫu nhiên. Gains đề xuất biểu diễn số ngẫu nhiên theo hai định dạng, đơn cực và lưỡng cực. Trong định dạng đơn cực $x = p = N_1/(N_1 + N_2)$, p luôn nằm trong khoảng $[0, 1]$ và nó được xem như là xác suất của vị trí bất kỳ trong chuỗi bit X đầu ra là bit 1. Trong định dạng lưỡng cực biểu diễn số x bằng cách chuyển giá trị p sang khoảng $[-1, 1]$, khi đó $x = 2p - 1$. Ví dụ một chuỗi bit X có 25% bit 1 và 75% bit 0 được ký hiệu là một số $p = 0,25$, nó chính là xác suất xuất hiện bit 1 tại vị trí bất kỳ trong chuỗi bit. Như vậy phụ thuộc vào tỷ số của tổng số bit 1 trên độ dài chuỗi chứ không phụ thuộc vào vị trí của chúng, điều đó có nghĩa nó là giá trị ngẫu nhiên.

Mục đích chính trong đề xuất SC là để thực thi các thao tác số học với chi phí thấp. Ví dụ như phép nhân trong SC được thực hiện rất đơn giản chỉ với một cổng AND. Cho hai chuỗi bit thực hiện phép AND logic với nhau, nếu xác suất xuất hiện bit 1 ở các đầu vào lần lượt là p_1 và p_2 thì xác suất xuất hiện bit 1 ở các vị trí của đầu ra cổng AND sẽ là $p_1 \times p_2$. Hình 5.1 biểu diễn phép nhân của hai số ngẫu nhiên này. Trên Hình 5.1a cả hai đầu vào của cổng AND là biểu diễn của số 4/8, chuỗi bit đầu ra là $4/8 \times 4/8 = 2/8$. Hình 5.1b biểu diễn khả năng khác của hai chuỗi bit, trong trường hợp này các đầu vào là các chuỗi bit có độ tương quan cao dẫn tới chuỗi bit đầu ra là 4/8 nó là một giá trị không chính xác. Ví dụ này là một mô tả điển hình cho vấn đề trong xử lý các số ngẫu nhiên.

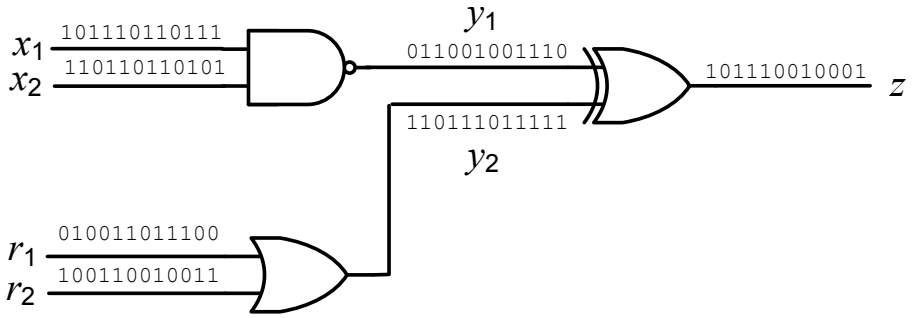
Ngoài tính đơn giản, các mạch SC còn có khả năng chịu lỗi tốt. Một số ngẫu nhiên X được xác định bằng số bit 1 trong chuỗi bit chứ không phải là vị trí của nó. Điều này có nghĩa là biểu diễn X dưới dạng một hệ thống số dư và do đó có khả năng chịu đựng lỗi. Ví dụ một chuỗi bit 00110010 là một số ngẫu nhiên của 3/8. Trong môi trường nhiễu, các bit có thể bị sai lệch. Nếu xảy ra lật bit tại một vị trí do nhiễu thì giá trị của số ngẫu nhiên sẽ thay đổi nhỏ thành 2/8 hoặc 4/8. Trong khi đó, trong biểu diễn nhị phân truyền thống nếu các bit sai lệch tại vị trí trọng số lớn sẽ xảy ra sai số rất lớn. Ví dụ, một số nhị phân truyền thống 0,011 có giá trị là 3/8, nếu lỗi xảy ra tại bit có trong số cao và chuyển thành 0,111 thì kết quả là một số 3/8 sẽ chuyển thành một số 7/8. Các số ngẫu nhiên là không có bit bậc cao do các bit được coi là có trọng số như nhau. Ưu điểm này của SC cho phép các mạch SC chịu đựng nhiễu tốt hơn.

Tính toán các hàm đơn giản trong SC là rất đơn giản bằng việc sử dụng



Hình 5.1: Mạch nhân trong tính toán ngẫu nhiên.

các cổng logic cơ bản. Ví dụ tính toán hàm $Z = \frac{1}{4} + \frac{1}{2}X_1X_2$ trên logic ngẫu nhiên thể hiện trên Hình 5.2 [78]. Trong ví dụ này các số ngẫu nhiên X_1, X_2 và Z biểu diễn của các đầu vào x_1, x_2 và đầu ra z. Mạch có hai đầu vào chính là x_1 và x_2 và hai đầu vào phụ r_1 và r_2 là các hằng số có giá trị là $1/2$. Cổng NAND trên Hình 5.2 sẽ thực hiện hàm $Y_1 = 1 - X_1X_2$, nó tương ứng với một mạch nhân và một mạch trừ. Cổng OR thực hiện $Y_2 = R_1 + R_2 - R_1R_2$ vì $R_1 = R_2 = 1/2$ nên $Y_2 = 3/4$. Cuối cùng cổng XOR thực hiện hàm $Z = Y_1 + Y_2 - 2Y_1Y_2 = \frac{1}{4} + \frac{1}{2}X_1X_2$.



Hình 5.2: Mạch SC thực thi hàm $Z = \frac{1}{4} + \frac{1}{2}X_1X_2$.

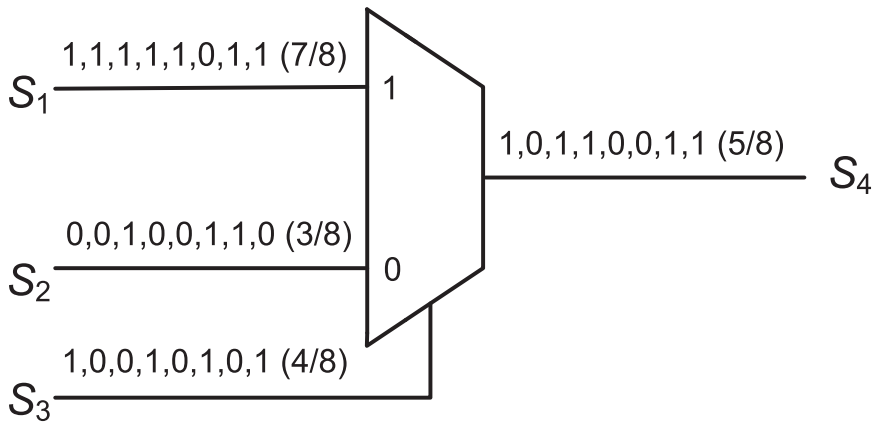
SC được coi là một loại kỹ thuật xấp xỉ. Các kỹ thuật xấp xỉ hiện nay được quan tâm bởi nó là một giải pháp hứa hẹn để khắc phục rào cản năng lượng và công suất trong các ứng dụng IC hiện đại. Các ứng dụng này thường chịu các nguồn lỗi tự nhiên, ví dụ như các ứng dụng xử lý ảnh và học máy

(Machine Learning) là các ứng dụng chịu các nguồn lỗi tự nhiên. Tương tự các hệ thống xử lý các dữ liệu cảm giác cũng rất phù hợp khi được xử lý bằng SC.

5.1.2. Các thành phần cơ bản của SC

Vì các số ngẫu nhiên được xem như là xác suất nên nó trong khoảng $[0, 1]$. Điều này sẽ gây ra bất tiện trong thao tác cộng bởi vì tổng của hai số trong khoảng $[0, 1]$ sẽ là một số trong khoảng $[0, 2]$. Vì lý do này các thao tác cộng tỷ lệ được sử dụng trong SC để ánh xạ giá khoảng $[0, 2]$ về khoảng $[0, 1]$. Bộ cộng trong SC được mô tả như trên Hình 5.3, một bộ ghép kênh 2 đầu vào được sử dụng để tính toán tổng của hai số ngẫu nhiên $p(S_1)$ và $p(S_2)$ tương ứng với hai đầu vào S_1 và S_2. Một đầu vào thứ ba là giá trị hằng số $p(S_3) = 1/2$ được sử dụng làm tín hiệu chọn kênh của MUX. Xác suất xuất hiện bit 1 ở đầu ra S_4 sẽ được xác định như công thức (5.1). Với các số ngẫu nhiên như ở trên Hình 5.3 ta có $p(S_4) = (7/8 + 3/8)/2 = 5/8$.

$$p(S_4) = p(S_3)p(S_1) + (1 - p(S_3))p(S_2) = (p(S_1) + p(S_2))/2. \qquad (5.1)$$



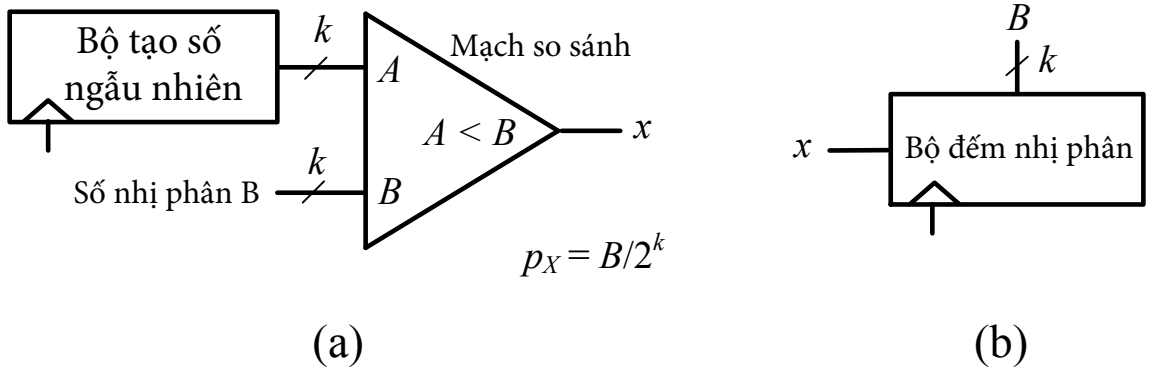
Hình 5.3: Mạch cộng trong tính toán ngẫu nhiên.

Mạch chuyển một số nhị phân sang một số ngẫu nhiên và ngược lại là các thành phần không thể thiếu trong SC. Mạch chuyển đổi số nhị phân sang số ngẫu nhiên được gọi là bộ tạo số ngẫu nhiên (SNG: Stochastic Number Generator) được mô tả như trên Hình 5.4a. Quá trình chuyển đổi sẽ được thực hiện bằng cách là trong mỗi chu kỳ đồng hồ bộ tạo số ngẫu nhiên sẽ được tạo ra một số nhị phân ngẫu nhiên m bit và so sánh với số nhị phân đầu vào. Nếu số ngẫu nhiên nhỏ hơn số nhị phân đầu vào thì đầu ra bộ so sánh sẽ tạo ra bit 1, ngược lại sẽ tạo ra bit 0. Mạch chuyển một số ngẫu nhiên sang số nhị phân truyền thống sẽ thực hiện bởi một bộ đếm như Hình 5.4b.

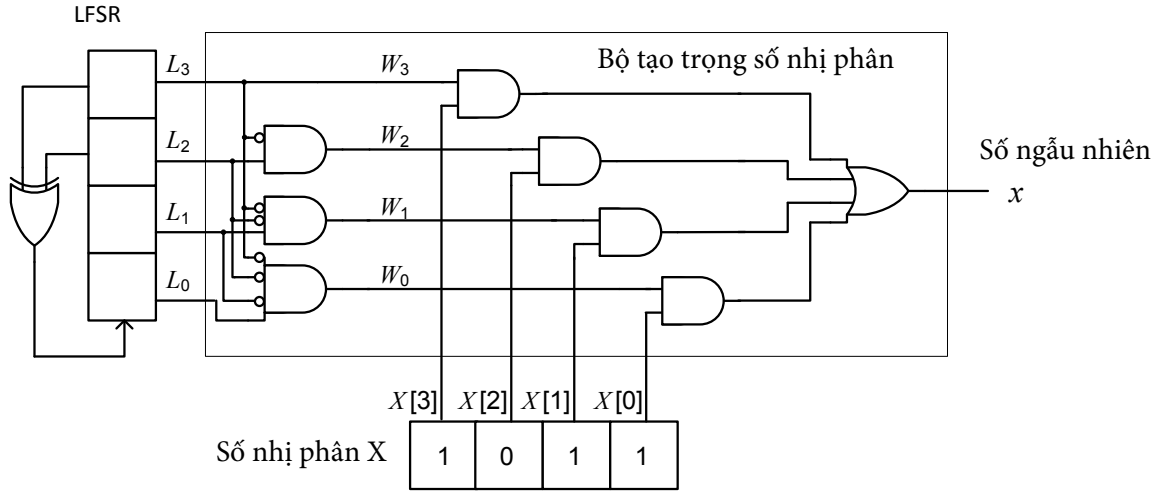
Để tạo ra các số ngẫu nhiên các SNG thường sử dụng các thanh ghi dịch hồi tiếp tuyến tính (LFSR: Linear Feedback Shift Registers), các LFSR thường là thanh ghi gồm có m flip-flop và một chu kỳ có $n = 2^m - 1$ trạng thái (trạng thái toàn các bit 0 bị loại trừ). Để tăng độ chính xác các LFSR phải tạo ra các số nhị phân có tính tương quan thấp. Nhiều nghiên cứu đã đề xuất các kiểu LFSR khác nhau để đạt được việc tạo ra các chuỗi bit có tính tương quan thấp. Trong đó điển hình là LFSR được đề xuất bởi Kumaresan trong [79]. Hình 5.5 mô tả một LFSR 4 bit theo đề xuất trong [79], nó chuyển một số nhị phân 4 bit x thành một số ngẫu nhiên có độ dài là 16 bit.

Các thao tác số học cơ bản trong SC tùy thuộc vào định dạng của số ngẫu nhiên là đơn cực (UP:Unipolar) hay lưỡng cực (BP: Bipolar). Phép nhân trong SC có thể thực thi bằng một cổng AND ở dạng đơn cực (Hình 5.6a) còn ở dạng lưỡng cực nó được thực hiện bằng một cổng XNOR (Hình 5.6b). Cổng đảo (INV) sẽ thực hiện phép tính $1 - x$ ở dạng đơn cực và $-x$ ở dạng lưỡng cực (Hình 5.6c). Các thao tác phức tạp hơn đòi hỏi sử dụng các phương pháp thực thi cụ thể như sử dụng đa thức Bernstein trong [80] hoặc

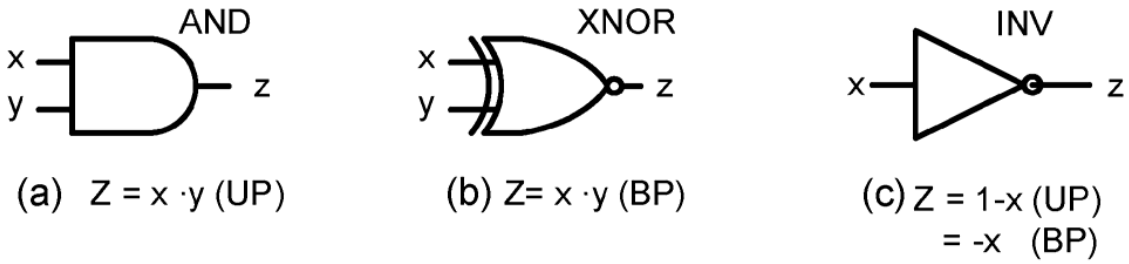
máy trạng thái hữu hạn [81]. Phép chia và khai căn trên các logic ngẫu nhiên đã được nghiên cứu trong [82].



Hình 5.4: Các bộ chuyển đổi SC: (a) nhị phân sang ngẫu nhiên; (b) ngẫu nhiên sang nhị phân.



Hình 5.5: SNG được đề xuất bởi Gupta và Kumaresan [79].



Hình 5.6: Một số thành phần cơ bản của SC.

5.1.3. Các ứng dụng

Tính toán ngẫu nhiên là một phương pháp mới đầy hứa hẹn cho việc giải mã LDPC. Ứng dụng SC trong giải mã LDPC đã đạt được những kết quả quan trọng. Các bộ giải mã LDPC ngẫu nhiên đã được nghiên cứu trong [83], [84]. Ngoài ra, các bộ giải mã dựa trên SC cho mã phân cực [85], bộ giải mã Viterbi [86] cũng đã được thiết kế. Đặc tính ngẫu nhiên và khả năng chịu đựng lỗi của SC là phù hợp với các ứng dụng này.

Thực thi mạng trí tuệ nhân tạo là ứng dụng chính của SC [87], [88], [89]. Hiện nay mạng nơ-ron thần kinh đang thu hút được quan tâm nên việc nghiên cứu về mạng nơ-ron ngẫu nhiên cũng được thúc đẩy mạnh mẽ bởi vì có sự giống nhau giữa mã hóa ngẫu nhiên với các "đột biến" thần kinh [78]. Các mạng nơ-ron dựa trên SC có độ phức tạp thấp cho phép thiết kế được mạng nơ-ron có độ phức tạp thấp.

Khai phá và nhận dạng dữ liệu [90], [91], học máy [92] cũng là những ứng dụng mới của SC. Các ứng dụng này vốn có khả năng chấp nhận lỗi, tạo ra các kỹ thuật tính toán xấp xỉ trong đó có SC là phù hợp với thực thi hiệu quả các ứng dụng này. SC cũng đã được ứng dụng rất tốt trong các mạch xử lý ảnh, nhiều mạch xử lý ảnh mới đã được nghiên cứu dựa trên SC trong [93], [94], các chip xử lý ảnh dựa trên SC cũng đã được chế tạo và cho thấy chất lượng tốt hơn so với các thiết kế truyền thống tương tự [95].

Gần đây một số nghiên cứu sử dụng SC để thực hiện các hàm toán học ứng dụng trong các lĩnh vực xử lý tín hiệu số và mạng nơ-ron. Các nghiên cứu của Qian và các cộng sự trong [80], Li trong [96], Parhi trong [97] đã tập trung sử dụng SC để thực thi nhiều hàm toán học khác nhau. Ý tưởng chính

của các nghiên cứu này là dựa trên xấp xỉ các hàm toán học bằng đa thức Bernstein, khai triển Maclaurin hoặc phương pháp máy trạng thái hữu hạn (FSM: Finite State Machine) để tính toán các hàm toán học.

5.2. Các nghiên cứu liên quan

Như đã nói ở trên, tính toán các hàm toán học dựa trên SC đã được nghiên cứu trong [80], [96] và [97]. Các nghiên cứu này sử dụng đa thức Bernstein, máy trạng thái hữu hạn và khai triển Maclaurin để thực thi các hàm toán học khác nhau trên logic ngẫu nhiên.

5.2.1. Thực thi sử dụng đa thức Bernstien

Trong [80] các tác giả đã thực hiện tính toán các hàm toán học sử dụng logic ngẫu nhiên trên cơ sở xấp xỉ các hàm toán học bằng đa thức Bernstien. Một hàm đa thức Bernstein bậc n có dạng như sau:

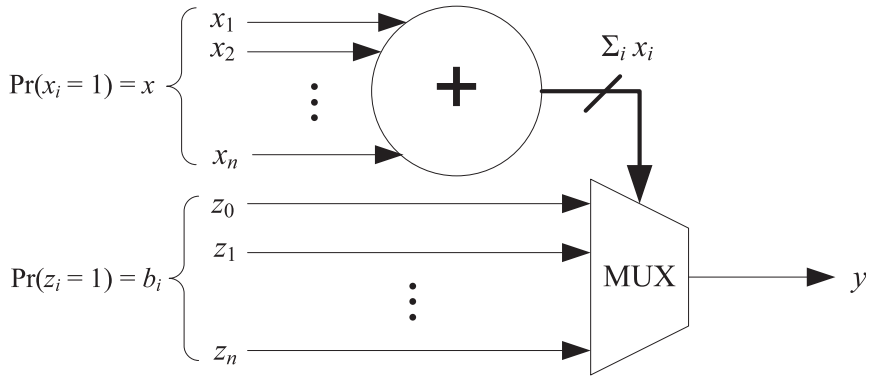
$$B(x) = \sum_{i=0}^{n} b_i B_{i,n}(x). \tag{5.2}$$

Trong đó b_i là các số thực được gọi là các hệ số Bernstein và các $B_{i,n}(x)$ được gọi là các đa thức Bernstein cơ sở và được tính như công thức (5.3).

$$B_{i,n}(x) = \begin{pmatrix} n \\ i \end{pmatrix} x^i (1-x)^{n-i}. \tag{5.3}$$

Một đa thức bậc n có thể xấp xỉ bằng một đa thức Bernstein có bậc không nhỏ hơn n. Hơn nữa nếu các khoảng giá trị của đa thức bậc n trong khoảng $[0, 1]$ thì chúng ta có thể chuyển nó thành một đa thức Bernstein với tất cả các hệ số cũng nằm trong khoảng $[0, 1]$ và từ đó có thể thực thi nó bằng các logic ngẫu nhiên chỉ với sử dụng mạch MUX như thể hiện trên Hình 5.7. Sơ đồ thực hiện gồm có một bộ cộng và một mạch MUX. Các đầu vào bộ cộng

sẽ là tập các giá trị đầu vào $(x_1 \ ... \ x_n)$. Các dữ liệu đầu vào mạch MUX sẽ là $(z_0 \ ... \ z_n)$. Đầu ra bộ cộng sẽ lựa chọn các đầu vào của MUX.



Hình 5.7: Thực thi hàm dựa trên SC sử dụng đa thức Bernstien.

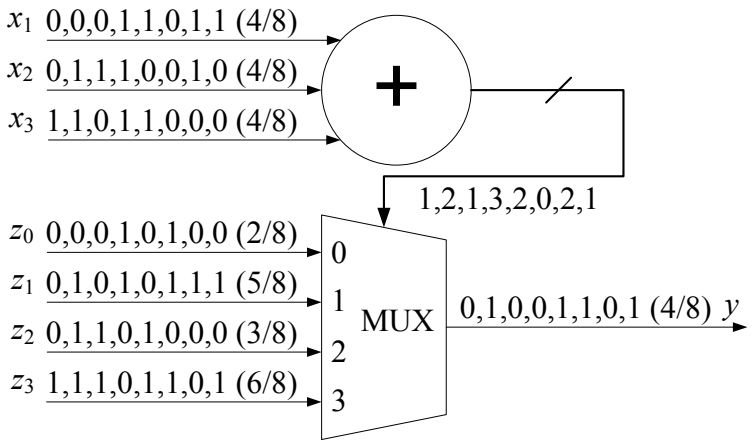
Ví dụ da thức $f(x) = \frac{1}{4} + \frac{9}{8}x - \frac{15}{8}x^2 + \frac{5}{4}x^3$ có thể chuyển thành đa thức Bernstien bậc 3 như sau:

$$f(x) = \frac{2}{8}B_{0,3}(x) + \frac{5}{8}B_{1,3}(x) - \frac{3}{8}B_{2,3}(x) + \frac{6}{8}B_{3,3}(x) \tag{5.4}$$

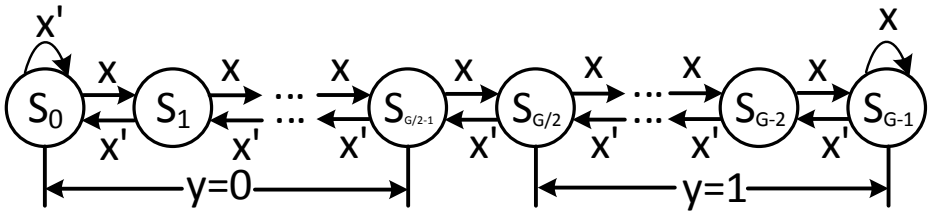
Giả sử rằng đa thức ban đầu cần ước lượng tại giá trị $x = 0,5$. Chuỗi bit ngẫu nhiên đầu vào x_1, x_2 và x_3 là độc lập và biểu diễn của giá trị $x = 0,5$. Chuỗi bit ngẫu nhiên của các đầu vào $z_1, ..., z_3$ biểu diễn các giá trị hệ số $b_0 = \frac{2}{8}$, $b_1 = \frac{5}{8}, b_2 = \frac{3}{8}$ và $b_3 = \frac{6}{8}$. Thực thi trên logic ngẫu nhiên của đa thức Bernstien này như Hình 5.8.

5.2.2. Thực thi sử dụng máy trạng thái hữu hạn

Phương pháp máy trạng thái hữu hạn để thực thi các hàm toán học đã được đề xuất bởi Brown và Card trong [81]. Một sơ đồ chuyển đổi trạng thái điển hình thể hiện như Hình 5.9 để thực thi hàm $\tanh(\frac{G}{2}x)$. Trên Hình 5.9 X là đầu vào của số ngẫu nhiên và Y là đầu ra của chuỗi bit ngẫu nhiên. Sơ đồ FSM như vậy được thực thi bởi một bộ đếm ngược và đếm thuận.



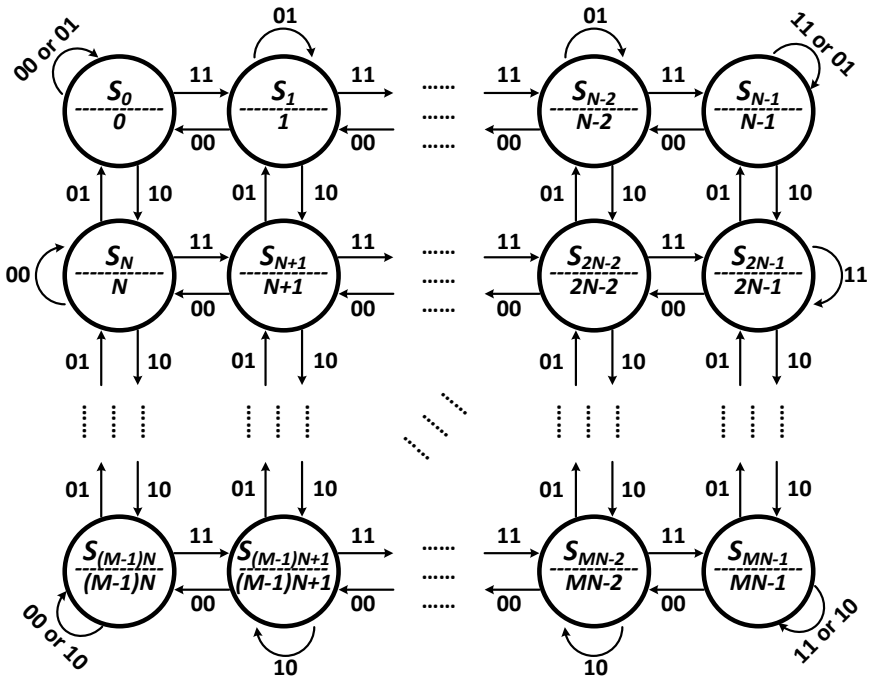
Hình 5.8: Thực thi hàm $f(x) = \frac{1}{4} + \frac{9}{8}x - \frac{15}{8}x^2 + \frac{5}{4}x^3$ trên SC sử dụng đa thức Bernstien.



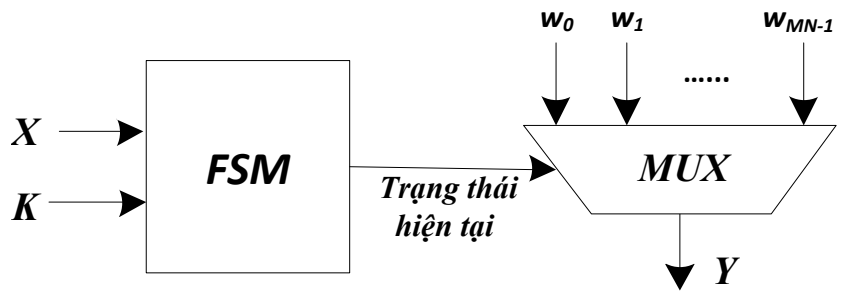
Hình 5.9: Thực thi hàm $\tanh(\frac{G}{2}x)$ trên SC sử dụng phương pháp FSM.

Tuy nhiên trong [96] các tác giả phát biểu rằng "các sơ đồ đề xuất bởi Brown và Card không thể sử dụng để tổng hợp các hàm phức tạp hơn chẳng hạn như các hàm đa thức bậc cao hay các hàm phi đa thức khác" vì vậy sơ đồ FSM như Hình 5.10 đã được đề xuất trong [96] để thực thi các hàm bất kỳ. Trên Hình 5.10 có hai đầu vào X và K, các số trên các đường mũi tên thể hiện điều kiện chuyển trạng thái, số đầu tiên tương ứng với đầu vào X và số thứ hai tương ứng với đầu vào K. Đầu ra nhị phân của FSM sẽ được mã hóa sử dụng $\log_2 \lceil MN \rceil$ bit. Số bên dưới mỗi trạng thái biểu diễn giá trị mã hóa ở đầu vào của FSM ở trạng thái S_t $(0 \leq t \leq MN - 1)$. Vì đầu ra của FSM chưa phải là chuỗi ngẫu nhiên nên kiến trúc hoàn chỉnh của thực thi các hàm trên số ngẫu nhiên được cho ở Hình 5.11, trong đó khối FSM tương ứng như Hình 5.10, với X biểu diễn đầu vào của chuỗi ngẫu nhiên và

đầu ra của FSM sẽ được sử dụng làm tín hiệu lựa chọn cho khối MUX. Các tham số K và w_i được lựa chọn để tối thiểu độ chênh lệch đầu ra Y và hàm thực. Các tác giả trong [96] đã chỉ ra rằng việc thêm các đầu vào K và w_i cho phép thực thi các hàm bất kỳ. Tuy nhiên, các đầu vào này phải được biểu diễn trong khoảng $[0, 1]$. Điều này cũng làm tăng độ phức tạp phần cứng do phải thêm các SNG.



Hình 5.10: Sơ đồ chuyển trạng thái của phương pháp FSM đề xuất trong [96].



Hình 5.11: Kiến trúc hoàn chỉnh cho thực thi các hàm sử dụng FSM [96].

5.2.3. Thực thi sử dụng khai triển Maclaurin

Trong [97] các tác giả đã thực thi các hàm toán học bằng các phần tử logic ngẫu nhiên dựa trên khai triển Maclaurin kết hợp với qui tắc Horner của các hàm. Dựa trên đặc tính của khai triển Maclaurin có các hệ số của các số hạng là thay đổi luân phiên giữa dương và âm do vậy kết hợp với qui tắc Horner có thể thực thi các hàm theo khai triển Maclaurin bởi các cổng NAND trong logic ngẫu nhiên. Khai triển Maclaurin của một số hàm điển hình như sau:

- Các hàm lượng giác

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} \cdots \tag{5.5}$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} \cdots \tag{5.6}$$

- Hàm mũ

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} \cdots \tag{5.7}$$

- Hàm logarithm tự nhiên

$$\ln(1+x) = \sum_{n=0}^{\infty} (-1)^{n+1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} \cdots \tag{5.8}$$

- Hàm tang hyberbol

$$\tanh x = \tfrac{1-e^{-2x}}{1+e^{-2x}}$$

$$\tanh x = \sum_{n=1}^{\infty} \frac{B_{2n} 4^n (4^n - 1)}{2n!} x^{2n-1} = x - \frac{1}{3}x^3 + \frac{2}{5}x^5 - \frac{17}{315}x^7 \cdots \tag{5.9}$$

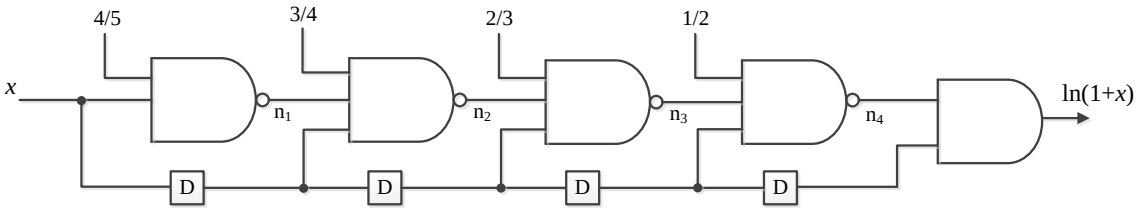
ở đây B_i là các số Bernoulli.

Trong [97] các tác giả đã thực hiện các hàm bằng việc sử dụng một số các số hạng đầu tiên trong khai triển Maclaurin. Kiến trúc phần cứng thực hiện các hàm trong [97] như sau:

- Hàm $\ln(1+x)$ được xấp xỉ bằng khai triển Maclaurin bậc 5 và kết hợp với qui tắc Horner theo công thức sau:

$$\begin{aligned}\ln(1+x) &\approx x - \tfrac{x^2}{2} + \tfrac{x^3}{3} - \tfrac{x^4}{4} + \tfrac{x^5}{5} \\ &= x(1 - \tfrac{1}{2}x(1 - \tfrac{2}{3}x(1 - \tfrac{3}{4}x(1 - \tfrac{4}{5}x)))).\end{aligned} \tag{5.10}$$

  Kiến trúc thực thi của hàm $\ln(1+x)$ dựa trên khai triển Maclaurin bậc 5 trên logic ngẫu nhiên như Hình 5.12.



Hình 5.12: Kiến trúc phần cứng thực thi hàm $\ln(1+x)$ sử dụng khai triển Maclaurin bậc 5.

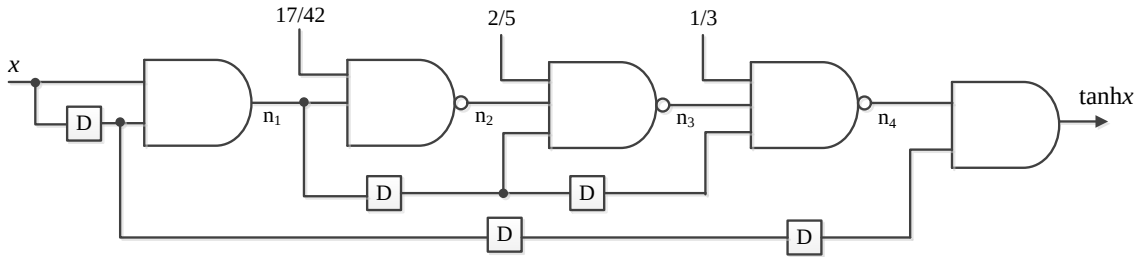
Trên Hình 5.12 các hệ số n_1, n_2, n_3 và n_4 được xác định như sau:

$$\begin{aligned} &n_1 = 1 - \tfrac{4}{5}x^2; \quad n_2 = 1 - \tfrac{3}{4}n_1; \quad n_3 = 1 - \tfrac{2}{3}n_2, \\ &n_4 = 1 - \tfrac{1}{2}n_3; \quad \ln(1+x) = n_4 \cdot x. \end{aligned} \tag{5.11}$$

- Khai triển bậc 7 của hàm $\tanh x$ như công thức sau:

$$\tanh x \approx x(1 - \frac{1}{3}x^2(1 - \frac{2}{5}x^2(1 - \frac{17}{42}x^2))). \tag{5.12}$$

  Kiến trúc thực thi của hàm $\tanh x$ dựa trên khai triển Maclaurin bậc 7 trên logic ngẫu nhiên như Hình 5.13.



Hình 5.13: Kiến trúc thực thi hàm $\tanh x$ sử dụng khai triển Maclaurin bậc 7.

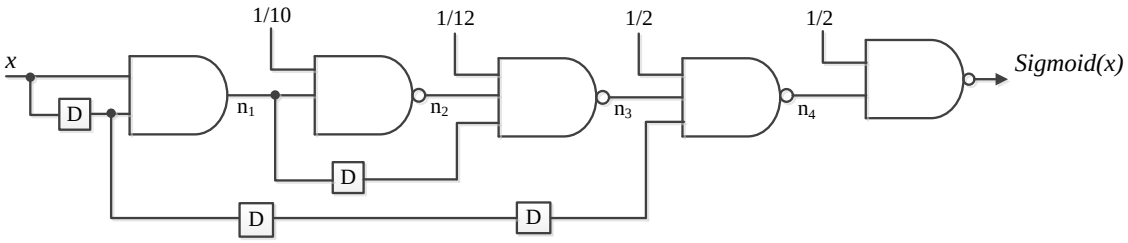
Các hệ số n_1, n_2, n_3 và n_4 trên Hình 5.13 được xác định như sau:

$$\begin{aligned} &n_1 = x^2\,; \quad n_2 = 1 - \tfrac{17}{42} n_1 \cdot x^2\,; \quad n_3 = 1 - \tfrac{2}{5} n_2 \cdot x^2\,; \\ &n_4 = 1 - \tfrac{1}{3} n_3 \cdot x^2\,; \quad \tanh x = n_4 \cdot x. \end{aligned} \tag{5.13}$$

- Khai triển bậc 5 của hàm $\text{sigmoid}(x)$ như sau:

$$\begin{aligned} \text{sigmoid}(x) &\approx \tfrac{1}{2} + \tfrac{x}{4} - \tfrac{x^3}{48} + \tfrac{x^5}{480} \\ &= 1 - \tfrac{1}{2} + \tfrac{x}{4} - \tfrac{x^2}{48} + \tfrac{x^3}{480} \\ &= 1 - \tfrac{1}{2}(1 - \tfrac{x}{2}(1 - \tfrac{x^2}{12}(1 - \tfrac{x^2}{10}))). \end{aligned} \tag{5.14}$$

Kiến trúc thực thi hàm $\text{sigmoid}(x)$ thể hiện như Hình 5.14.



Hình 5.14: Kiến trúc thực thi hàm $\text{sigmoid}(x)$ sử dụng khai triển Maclaurin bậc 5.

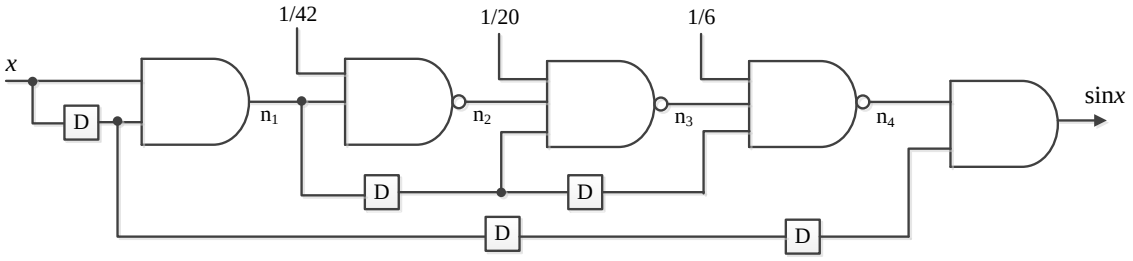
Trên Hình 5.14 các hệ số n_1, n_2, n_3 và n_4 được xác định như sau:

$$\begin{aligned} &n_1 = x^2\,; \quad n_2 = 1 - \tfrac{1}{10} n_1\,; \quad n_3 = 1 - \tfrac{1}{12} n_2\,; \\ &n_4 = 1 - \tfrac{1}{2} n_3\,; \quad \text{sigmoid}(x) = 1 - \tfrac{1}{2} n_4. \end{aligned} \tag{5.15}$$

- Khai triển bậc 7 của hàm sin$x$ như công thức sau

$$\sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} = x(1 - \frac{x^2}{6}(1 - \frac{x^2}{20}(1 - \frac{x^2}{42}))). \quad (5.16)$$

Hình 5.15 là kiến trúc thực thi hàm sin$x$ sử dụng khai triển Maclaurin bậc 7 trong [97].



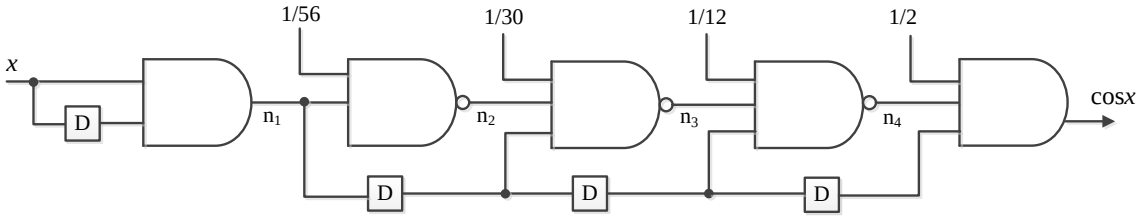
Hình 5.15: Kiến trúc thực thi hàm sin$x$ sử dụng khai triển Maclaurin bậc 7.

Trên Hình 5.15 các hệ số n_1, n_2, n_3 và n_4 được xác định như sau:

$$\begin{aligned} &n_1 = x^2\,; \quad n_2 = 1 - \tfrac{1}{42}n_1 \cdot x^2\,; \quad n_3 = 1 - \tfrac{1}{20}n_2 \cdot x^2\,; \\ &n_4 = 1 - \tfrac{1}{6}n_3 \cdot x^2\,; \quad \sin x = n_4 \cdot x. \end{aligned} \quad (5.17)$$

- Khai triển bậc 8 của hàm cos$x$ và kiến trúc thực thi hàm cos$x$ như sau:

$$\begin{aligned} \cos x &\approx 1 - \tfrac{x^2}{2!} + \tfrac{x^4}{4!} - \tfrac{x^6}{6!} + \tfrac{x^8}{8!} \\ &= 1 - \tfrac{x^2}{2}(1 - \tfrac{x^2}{12}(1 - \tfrac{x^2}{2}(1 - \tfrac{x^2}{56}))). \end{aligned} \quad (5.18)$$



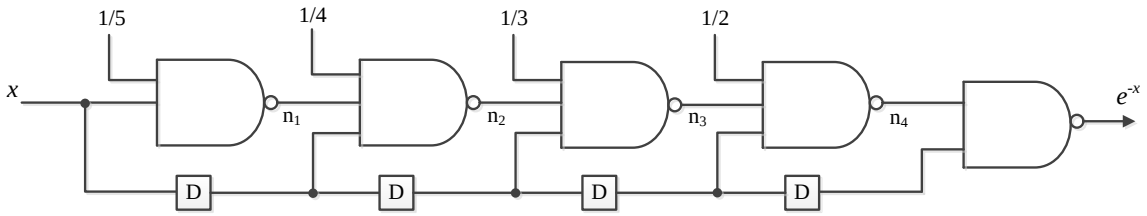
Hình 5.16: Kiến trúc thực thi hàm cos$x$ sử dụng khai triển Maclaurin bậc 8.

Trên Hình 5.16 các hệ số n_1, n_2, n_3 và n_4 được xác định như sau:

$$\begin{aligned} &n_1 = x^2\,; \quad n_2 = 1 - \tfrac{1}{56} n_1 \cdot x^2\,; \quad n_3 = 1 - \tfrac{1}{30} n_2 \cdot x^2\,; \\ &n_4 = 1 - \tfrac{1}{12} n_3 \cdot x^2\,; \quad \cos x = \tfrac{1}{2} n_4 \cdot x. \end{aligned} \tag{5.19}$$

- Khai triển bậc 5 của hàm e^{-x} và kiến trúc thực thi hàm e^{-x} như sau:

$$\begin{aligned} e^{-x} &\approx 1 - x + \tfrac{x^2}{2!} - \tfrac{x^3}{3!} + \tfrac{x^4}{4!} - \tfrac{x^5}{5!} \\ &= \; 1 - x(1 - \tfrac{x}{2}(1 - \tfrac{x}{3}(1 - \tfrac{x}{4}(1 - \tfrac{x}{5})))). \end{aligned} \tag{5.20}$$



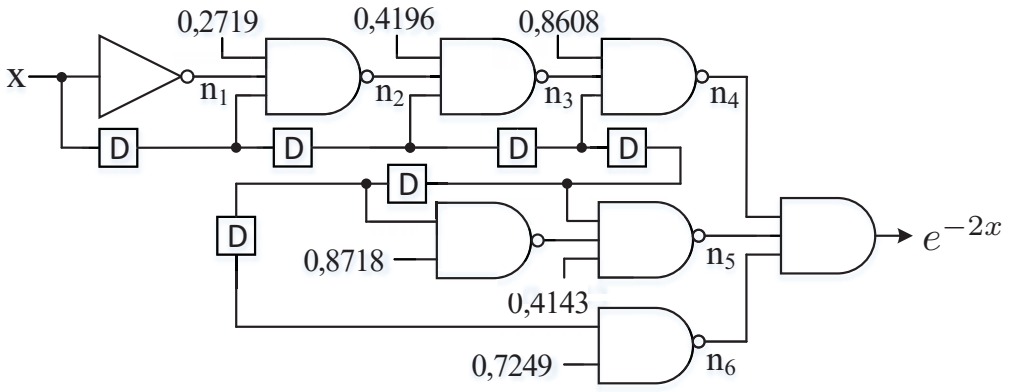
Hình 5.17: Kiến trúc thực thi hàm e^{-x} sử dụng khai triển Maclaurin bậc 5.

Trên Hình 5.17 các hệ số n_1, n_2, n_3 và n_4 được xác định như sau:

$$\begin{aligned} &n_1 = 1 - \tfrac{1}{5} \cdot x\,; \quad n_2 = 1 - \tfrac{1}{4} n_1 \cdot x\,; \quad n_3 = 1 - \tfrac{1}{3} n_2 \cdot x\,; \\ &n_4 = 1 - \tfrac{1}{2} n_3 \cdot x\,; \quad e^{-x} = 1 - n_4 \cdot x. \end{aligned} \tag{5.21}$$

- Hàm e^{-2x} có khai triển Maclourin bậc 7 như sau:

$$\begin{aligned} e^{-2x} &\approx 1 - 2x + \tfrac{4x^2}{2!} - \tfrac{8x^3}{3!} + \tfrac{16x^4}{4!} - \tfrac{32x^5}{5!} + \tfrac{64x^6}{6!} - \tfrac{128x^7}{7!} \\ &= (1 - 0,7249x) \times (1 - 0,4143x + 0,3612x^2) \times \\ &\quad (1 - 1,1445x + 0,4810x^2) \times (1 + 0,2837x + 0,205x^2) \\ &= \; (1 - 0,7249x) \times (1 - 0,4143x + 0,3612x^2) \times \\ &\quad (1 - 0,8608x + 0,3612x^2 - 0,0982x^3 + 0,0986x^4) \\ &= \; (1 - 0,7249x) \times (1 - 0,4143x(1 - 0,8718x) \times \\ &\quad (1 - 0,8608(1 - 0,4196x(1 - 0,2719x(1 - x)))). \end{aligned} \tag{5.22}$$



Hình 5.18: Kiến trúc thực thi hàm e^{-2x} sử dụng khai triển Maclaurin bậc 7.

Các hệ số $n_i, i = 1 \div 6$ trên Hình 5.18 như sau:

$$\begin{aligned} &n_1 = 1 - x\,; \;\; n_2 = 1 - 0,2719 n_1 x\,; \;\; n_3 = 1 - 0,4196 n_2 x\,; \\ &n_4 = 1 - 0,8608 n_3 x\,; \;\; n_5 = 1 - 0,4143 x(1 - 0,8718x)\,; \\ &n_6 = 1 - 0,7249x\,; \;\; e^{-2x} = n_4 n_5 n_6. \end{aligned} \tag{5.23}$$

- Khai triển bậc 9 của hàm $\frac{\sin \pi x}{\pi}$ như công thức sau:

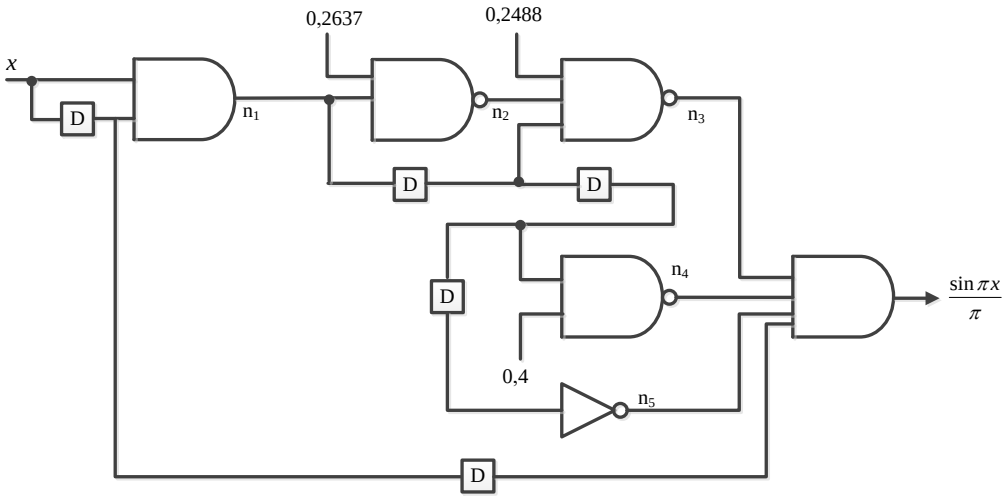
$$\begin{aligned} \frac{\sin \pi x}{\pi} &\approx x - \frac{\pi^2 x^3}{3!} + \frac{\pi^4 x^5}{5!} - \frac{\pi^6 x^7}{7!} + \frac{\pi^8 x^9}{9!} \\ &= x(1 - x^2)(1 - 0,4x^2)(1 - 0,2488x^2 + 0,0656x^4) \\ &= x(1 - x^2)(1 - 0,4x^2)(1 - 0,2488x^2(1 - 0,2637x^2)). \end{aligned} \tag{5.24}$$

Kiến trúc phần cứng thực thi hàm $\frac{\sin \pi x}{\pi}$ theo khai triển bậc Maclaurin bậc 9. trong [97] như Hình 5.19

Trên Hình 5.19 các hệ số n_1, n_2, n_3, n_4 và n_5 được xác định như sau:

$$\begin{aligned} &n_1 = x^2; \;\; n_2 = 1 - 0,2637 n_1; \;\; n_3 = 1 - 0,2488 n_2 \cdot x^2\,; \\ &n_4 = 1 - 0,4x^2\,; \;\; n_5 = 1 - x^2\,; \;\; \frac{\sin \pi x}{x} = n_3 n_4 n_5 \cdot x. \end{aligned} \tag{5.25}$$

Trong [97] các tác giả đã thực thi các hàm toán học theo phương pháp khai triển Maclaurin như trên và so sánh với phương pháp sử dụng đa thức



Hình 5.19: Kiến trúc thực thi hàm $\frac{\sin \pi x}{\pi}$ sử dụng khai triển Maclaurin bậc 9.

Bernstien và FSM. Theo các kết quả đưa ra trong [97] cho thấy phương pháp sử dụng khai triển Maclaurin đạt được hiệu quả cao hơn so với phương pháp sử dụng đa thức Bernstien và FSM trong [80], [96]. Trong chương này của luận án, sẽ đề xuất sử dụng một phương pháp mới để thực thi các hàm toán học trên logic ngẫu nhiên. Phương pháp đề xuất sử dụng xấp xỉ phân đoạn tuyến tính đều các hàm toán học. Các kiến trúc tính toán các hàm toán học theo phương pháp Maclaurin trình bày ở trên sẽ được thực thi lại và so sánh với thực thi của phương pháp đề xuất.

5.3. Phương pháp xấp xỉ tuyến tính phân đoạn đều các hàm cho tính toán trên SC

Để thực thi trên phần cứng các hàm toán học phức tạp thường được xấp xỉ về các hàm đơn giản hơn có thể thực thi bằng các thao tác cơ bản như cộng, trừ, nhân, chia. Độ phức tạp của phần cứng sẽ phụ thuộc vào hàm xấp xỉ cụ thể. Trong chương này của luận án, sử dụng tính toán ngẫu nhiên để tính toán các hàm sau khi được xấp xỉ. Các hàm ban đầu được xấp xỉ bằng

phương pháp xấp xỉ phân đoạn tuyến tính đều. Để phù hợp với tính toán trên SC, các khoảng giá trị của biến đầu vào của các hàm được xác định trong khoảng [0, 1]. Nhằm đơn giản về kiến trúc phần cứng thì số phân đoạn được lựa chọn là một số lũy thừa của 2. Về nguyên tắc khi số phân đoạn lớn thì độ chính xác xấp xỉ tăng, tuy nhiên khi số phân đoạn lớn sẽ làm tăng độ phức tạp phần cứng, do vậy việc lựa chọn số phân đoạn xấp xỉ cần dung hòa giữa độ phức tạp phần cứng và yêu cầu về độ chính xác. Trong chương này, các hàm được lựa chọn xấp xỉ với 8 phân đoạn đều. Trong phân đoạn thứ i hàm xấp xỉ được viết như công thức (5.26).

$$f(x) \approx a_i x + b_i, \qquad \frac{i}{8} \leq x < \frac{i+1}{8}, \;\; i = 0 \div 7. \tag{5.26}$$

Lỗi xấp xỉ trên phân đoạn thứ i là:

$$\varepsilon_i(x) = f(x) - (a_i x + b_i), \; \frac{i}{8} \leq x < \frac{i+1}{8}, \;\; i = 0 \div 7. \tag{5.27}$$

Ngoài ra, để chính xác trong biểu diễn các hệ số trên phần cứng, thì các hệ số a_i và b_i có dạng như công thức (5.28), trong đó B là biểu diễn độ rộng bit của phần cứng chứa các hệ số.

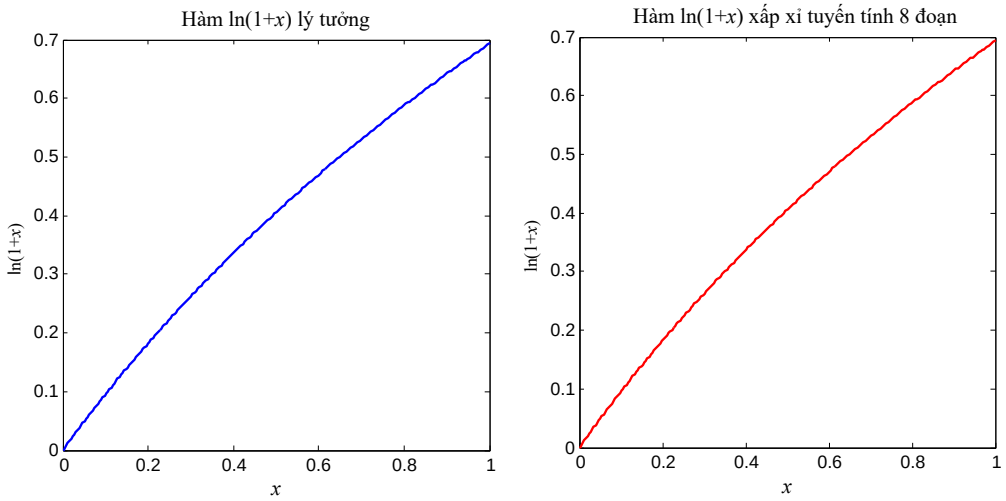
$$a_i, b_i = N \times 2^{-B}, \quad N, B \in \mathbb{Z}. \tag{5.28}$$

Thực hiện tìm các hệ số a_i và b_i tối ưu heo tiêu chí tối thiểu hóa lỗi cực đại trên từng phân đoạn theo thuật toán như đã trình bày trong mục 1.3.2. Trong đó, các hệ số a_i và b_i được ràng buộc theo công thức (5.28) và độ rộng bit các thành phần phần cứng thực thi là 8 bit ($B = 8$). Các hệ số cho xấp xỉ 9 hàm toán học điển hình trong các ứng dụng của xử lý tín hiệu số và mạng nơ-ron thể hiện trên các Bảng 5.1, Bảng 5.2 và Bảng 5.3. Trên các bảng này, tham số $\varepsilon_{\max}$ là lỗi cực đại xảy ra do xấp xỉ. Hình 5.20 thể hiện trường hợp

hàm $\ln(1+x)$ ở cả hai trường hợp lý tưởng và xấp xỉ phân đoạn tuyến tính 8 đoạn với các hệ số tìm được.

Bảng 5.1: Hệ số của các hàm $\ln(1+x)$, $\tanh(x)$ và $\text{sigmoid}(x)$.

Hàm $\ln(1+x)$			Hàm $\tanh x$			Hàm $\text{sigmoid}(x)$		
Đoạn	a_i	b_i	Đoạn	a_i	b_i	Đoạn	a_i	b_i
0	243×2^{-8}	0	0	255×2^{-8}	0	0	64×2^{-8}	128×2^{-8}
1	217×2^{-8}	3×2^{-8}	1	247×2^{-8}	2×2^{-8}	1	64×2^{-8}	128×2^{-8}
2	193×2^{-8}	9×2^{-8}	2	232×2^{-8}	5×2^{-8}	2	63×2^{-8}	128×2^{-8}
3	180×2^{-8}	14×2^{-8}	3	213×2^{-8}	12×2^{-8}	3	63×2^{-8}	128×2^{-8}
4	162×2^{-8}	23×2^{-8}	4	189×2^{-8}	24×2^{-8}	4	57×2^{-8}	131×2^{-8}
5	151×2^{-8}	30×2^{-8}	5	165×2^{-8}	39×2^{-8}	5	57×2^{-8}	131×2^{-8}
6	143×2^{-8}	36×2^{-8}	6	141×2^{-8}	57×2^{-8}	6	52×2^{-8}	135×2^{-8}
7	126×2^{-8}	51×2^{-8}	7	117×2^{-8}	78×2^{-8}	7	50×2^{-8}	137×2^{-8}
$\varepsilon_{\max}=0,0018$			$\varepsilon_{\max}=0,0012$			$\varepsilon_{\max}=7,1\times10^{-4}$		



Hình 5.20: Hàm $\ln(1+x)$ lý tưởng và xấp xỉ tuyến tính 8 đoạn.

Bảng 5.2: Hệ số của các hàm $\sin x$, $\cos x$ và e^{-x}.

Hàm $\sin x$			Hàm $\cos x$			Hàm e^{-x}		
Đoạn	a_i	b_i	Đoạn	a_i	b_i	Đoạn	a_i	b_i
0	255×2^{-8}	0	0	-13×2^{-8}	1	0	-234×2^{-8}	256×2^{-8}
1	254×2^{-8}	0	1	-47×2^{-8}	132×2^{-8}	1	-211×2^{-8}	252×2^{-8}
2	245×2^{-8}	2×2^{-8}	2	-79×2^{-8}	140×2^{-8}	2	-187×2^{-8}	246×2^{-8}
3	232×2^{-8}	7×2^{-8}	3	-106×2^{-8}	160×2^{-8}	3	-166×2^{-8}	238×2^{-8}
4	214×2^{-8}	16×2^{-8}	4	-138×2^{-8}	166×2^{-8}	4	-144×2^{-8}	227×2^{-8}
5	201×2^{-8}	24×2^{-8}	5	-162×2^{-8}	181×2^{-8}	5	-131×2^{-8}	219×2^{-8}
6	178×2^{-8}	41×2^{-8}	6	-186×2^{-8}	199×2^{-8}	6	-116×2^{-8}	208×2^{-8}
7	144×2^{-8}	71×2^{-8}	7	-211×2^{-8}	221×2^{-8}	7	-101×2^{-8}	195×2^{-8}
$\varepsilon_{\max}=0,002$			$\varepsilon_{\max}=0,0015$			$\varepsilon_{\max}=0,0013$		

Bảng 5.3: Hệ số của các hàm e^{-2x}, $\frac{\sin(\pi x)}{\pi}$ và $\log_2(1+x)$.

Hàm e^{-2x}			Hàm $\frac{\sin(\pi x)}{\pi}$			Hàm $\log_2(1+x)$		
Đoạn	a_i	b_i	Đoạn	a_i	b_i	Đoạn	a_i	b_i
0	-450×2^{-8}	255×2^{-8}	0	255×2^{-8}	0	0	351×2^{-8}	0
1	-355×2^{-8}	243×2^{-8}	1	254×2^{-8}	0	1	310×2^{-8}	5×2^{-8}
2	-273×2^{-8}	233×2^{-8}	2	245×2^{-8}	2×2^{-8}	2	282×2^{-8}	12×2^{-8}
3	-213×2^{-8}	200×2^{-8}	3	-232×2^{-8}	7×2^{-8}	3	258×2^{-8}	21×2^{-8}
4	-168×2^{-8}	178×2^{-8}	4	-214×2^{-8}	16×2^{-8}	4	234×2^{-8}	33×2^{-8}
5	-218×2^{-8}	153×2^{-8}	5	-201×2^{-8}	24×2^{-8}	5	217×2^{-8}	44×2^{-8}
6	-100×2^{-8}	132×2^{-8}	6	-178×2^{-8}	41×2^{-8}	6	205×2^{-8}	53×2^{-8}
7	-73×2^{-8}	108×2^{-8}	7	-144×2^{-8}	71×2^{-8}	7	190×2^{-8}	66×2^{-8}
$\varepsilon_{\max}=0,0039$			$\varepsilon_{\max}=0,002$			$\varepsilon_{\max}=0,0018$		

5.4. Kiến trúc phần cứng tính toán các hàm toán học sử dụng logic ngẫu nhiên kết hợp với xấp xỉ phân đoạn tuyến tính đều

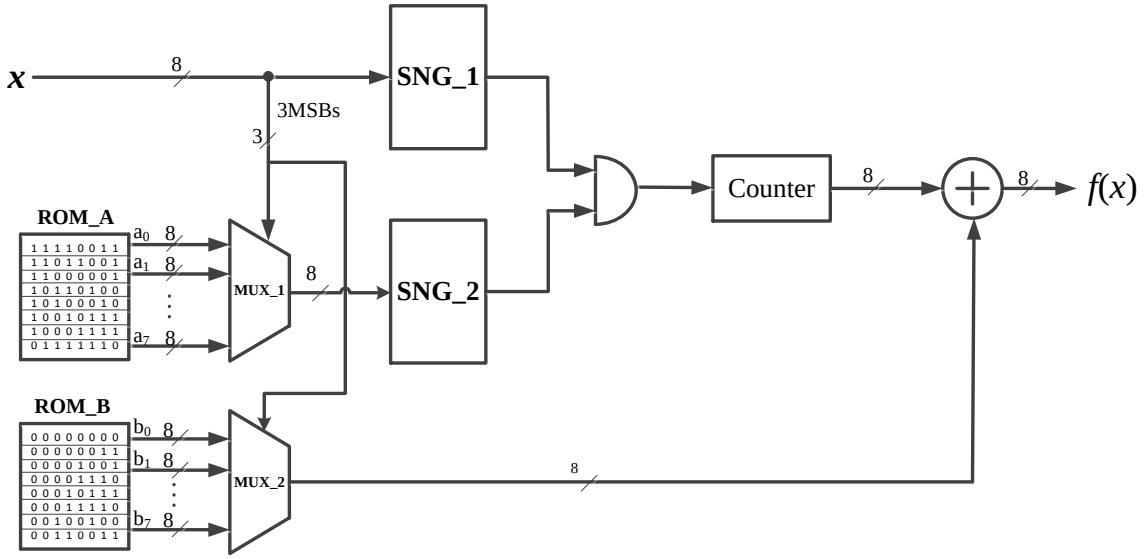
Trong phần này, các kiến trúc phần cứng tính toán một số hàm toán học phổ biến dựa trên các logic ngẫu nhiên và xấp xỉ phân đoạn tuyến tính sẽ được đề xuất. Kiến trúc phần cứng cho các hàm riêng lẻ sẽ được đưa ra, đồng thời một kiến trúc chung cho tính toán nhiều hàm toán học dựa trên phương pháp đề xuất cũng được thiết kế và thực thi.

5.4.1. Kiến trúc phần cứng tính toán các hàm $\ln(1+x)$, $\tanh x$, *sigmoid*(x) *và* $\sin x$

Theo phương pháp đề xuất thì các hàm được xấp xỉ bằng phân đoạn tuyến tính đều và trong mỗi phân đoạn hàm được xấp xỉ bởi một đoạn tuyến tính. Do vậy khi thực thi đòi hỏi một mạch nhân và một mạch cộng. Việc thực thi phép nhân trong các kiến trúc truyền thống sẽ yêu cầu khá nhiều chi phí về phần cứng. Trong logic ngẫu nhiên phép nhân này chỉ cần thực thi bởi một cổng AND, do đó cho phép giảm chi phí phần cứng và tăng tốc độ. Tuy nhiên đòi hỏi các bộ tạo số ngẫu nhiên SNG. Theo Bảng 5.1 và Bảng 5.2 các hệ số cho xấp xỉ các hàm $\ln(1+x)$, $\tanh x$, $\text{sigmoid}(x)$ và $\sin x$ trên tất cả các phân đoạn đều thuộc khoảng [0,1]. Từ đó có thể xây dựng một kiến trúc chung cho tính toán các hàm này như thể hiện trên Hình 5.21.

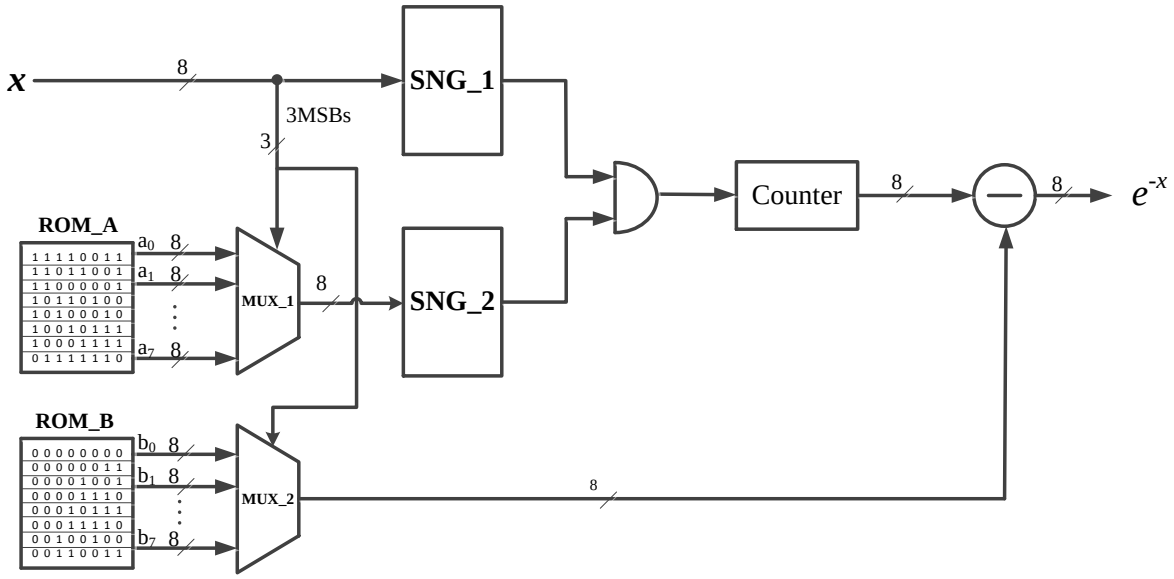
5.4.2. Kiến trúc phần cứng tính toán hàm e^{-x}

Các hệ số cho xấp xỉ hàm e^{-x} có giá trị tuyệt đối cũng đều thuộc khoảng $[0, 1]$. Tuy nhiên các hệ số a_i trong xấp xỉ hàm này có giá trị âm, do đó vậy kiến trúc thực hiện chúng tương tự như kiến trúc thực hiện các hàm



Hình 5.21: Kiến trúc phần cứng thực thi hàm $\ln(1+x)$, $\tanh x$ *sigmoid*, *sinx*.

$\ln(1+x)$, $\tanh x$, $\text{sigmoid}(x)$ và $\sin x$, chỉ khác thay thế bộ cộng trong kiến trúc Hình 5.21 thành một bộ trừ. Kiến trúc phần cứng thực hiện hàm e^{-x} thể hiện trên Hình 5.22.



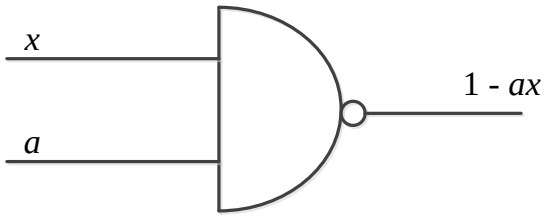
Hình 5.22: Kiến trúc phần cứng thực thi hàm e^{-x}.

5.4.3. Kiến trúc phần cứng tính toán hàm cosx

Các hệ số cho xấp xỉ hàm $\cos x$ có đặc điểm là các hệ số a_i đều âm và có trị tuyệt đối nhỏ hơn 1, còn các hệ số b_i lại thuộc khoảng [1,2). Để thực thi phần cứng cần biểu diễn các hệ số có trị tuyệt đối trong khoảng [0,1]. Do đó, cần biến đổi các hệ số về khoảng $[0, 1]$. Ví dụ, với phân đoạn thứ hai ($x \in [0, 125, 0, 25)$), hàm được xấp xỉ như sau:

$$\cos x \approx -47.2^{-8}.x + 260.2^{-8} = 1 - 47.2^{-8}.x + 14.2^{-8} = 1 - ax + b, \tag{5.29}$$

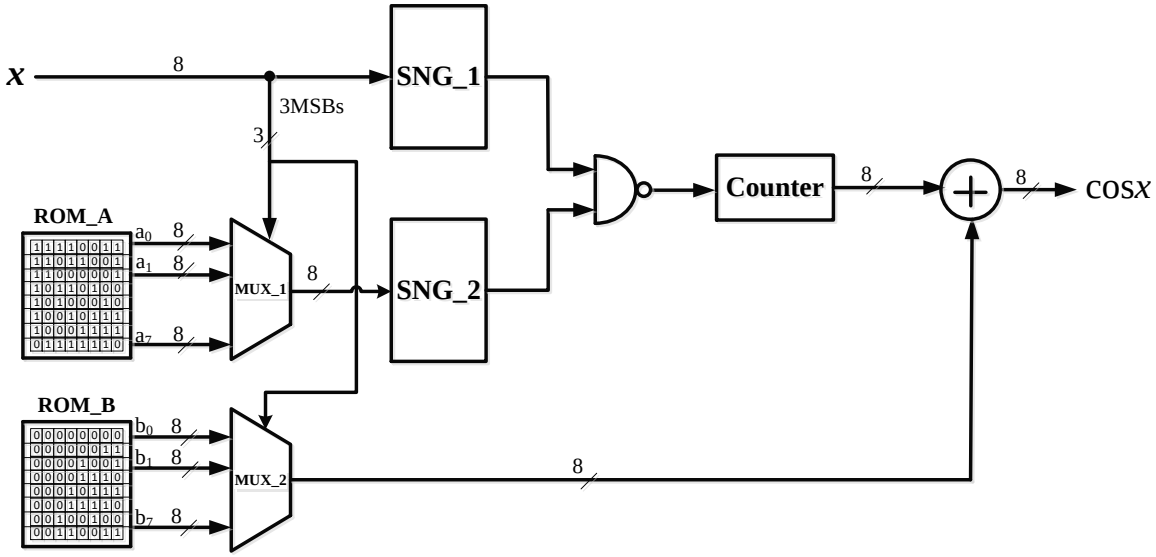
với $a = 47.2^{-8}$ và $b = 14.2^{-8}$, với các phân đoạn còn lại biến đổi hoàn toàn tương tự ta cũng nhận được dạng biểu diễn như công thức (5.29). Trong SC với các đầu vào định dạng đơn cực, phép tính $1 - ax$ được thực hiện bởi một cổng NAND như mô tả ở Hình 5.23. Do đó, kiến trúc phần cứng thực hiện hàm $\cos x$ sẽ có dạng như biểu diễn trên Hình 5.24.



Hình 5.23: Thao tác tính 1-ax trong SC.

5.4.4. Kiến trúc phần cứng tính toán hàm e^{-2x}

Các hệ số a_i cho xấp xỉ hàm e^{-2x} có đặc điểm là đều là các số âm và ba hệ số đầu tiên (a_0, a_1, a_2) có trị tuyệt đối lớn hơn 1. Điều này sẽ dẫn tới kiến trúc thực thi thay đổi để phù hợp với đặc điểm của các hệ số. Với ba phân

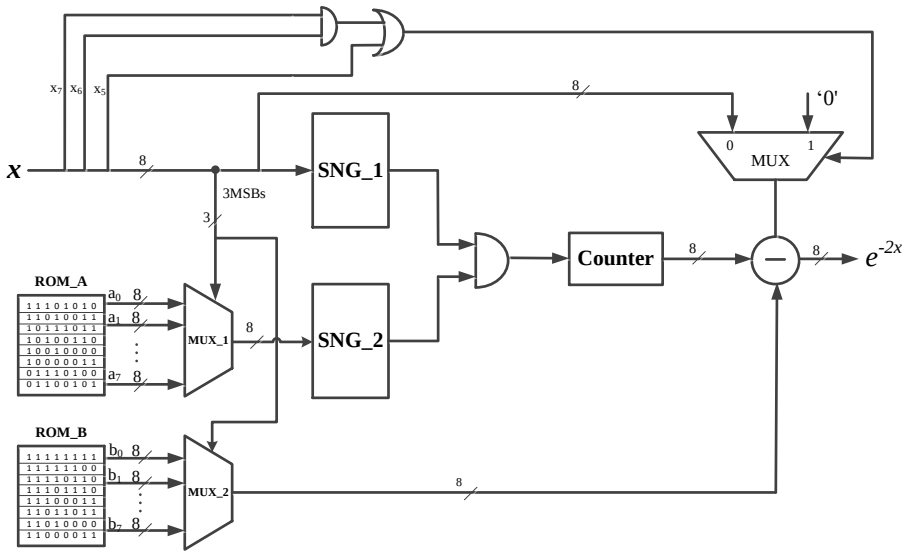


Hình 5.24: Kiến trúc phần cứng thực thi hàm $cosx$.

đoạn đầu tiên, biểu thức xấp xỉ có thể được viết lại như sau:

$$f(x) = -a_i x + b_i = -(1 + a_i^{'})x + b_i = -a_i^{'}x - x + b_i, \;\; i = 0, 1, 2. \quad (5.30)$$

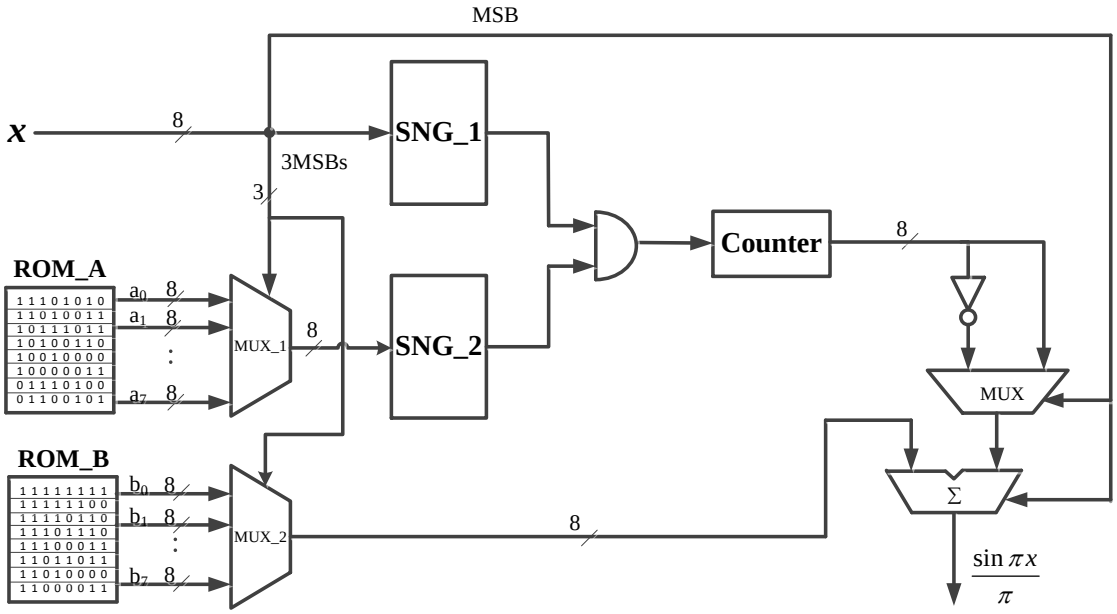
Khi đó các giá trị $a_i^{'}$ sẽ thuộc khoảng $[0, 1]$ và sẽ được chứa trong ROM_A. ngoài ra cần tạo ra một tín hiệu điều khiển để để đưa giá trị x vào bộ trừ ở đầu ra. Kiến trúc phần cứng thực thi hàm thể hiện trên Hình 5.25.



Hình 5.25: Kiến trúc phần cứng thực thi hàm e^{-2x}.

5.4.5. Kiến trúc phần cứng tính toán hàm $\frac{\sin(\pi x)}{\pi}$

Các hệ số xấp xỉ hàm $\frac{\sin(\pi x)}{\pi}$ có đặc điểm đều có trị tuyệt đối thuộc khoảng $[0, 1]$, trong đó bốn hệ số a_i đầu tiên có giá trị dương còn 4 hệ số sau có giá trị âm, do vậy kiến trúc phần cứng sẽ có cả mạch cộng và mạch trừ và tín hiệu điều khiển để sử dụng mạch cộng và mạch trừ sử dụng bit MSB của dữ liệu đầu vào. Kiến trúc của mạch tính toán hàm $\frac{\sin(\pi x)}{\pi}$ như trên Hình 5.26.



Hình 5.26: Kiến trúc phần cứng thực thi hàm $\frac{\sin(\pi x)}{\pi}$.

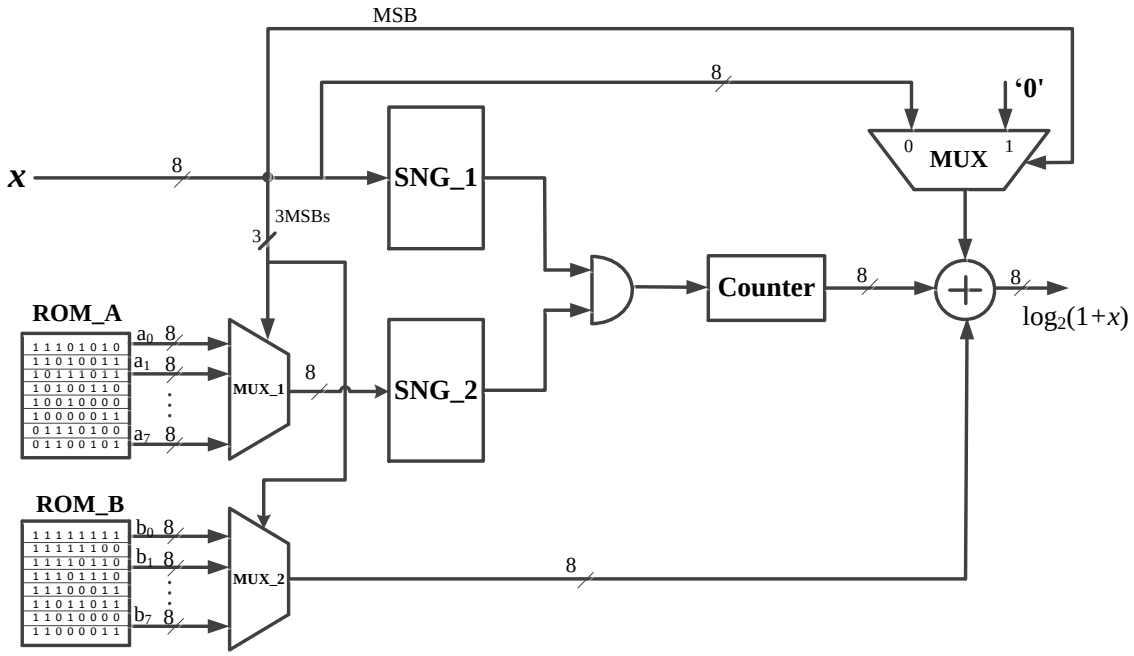
5.4.6. Kiến trúc phần cứng tính toán hàm $\log_2(1+x)$

Theo Bảng 5.3. các hệ số cho xấp xỉ hàm có 4 giá trị đầu của hệ số a_i thuộc khoảng $[1, 2)$. Do vậy với các hệ số a_i lớn hơn 1. Có thể phân tích như sau:

$$f(x) = a_i x + b_i = (1 + a_i^{'})x + b_i = x + a_i^{'} x + b_i, \quad i = 0, 1, 2, 3. \qquad (5.31)$$

Khi đó các giá trị $a_i^{'}$ sẽ thuộc khoảng [0,1] và sẽ được chứa trong ROM_A. Tín hiệu điều khiển để đưa giá trị x vào bộ cộng sau cùng chỉ sử dụng bit

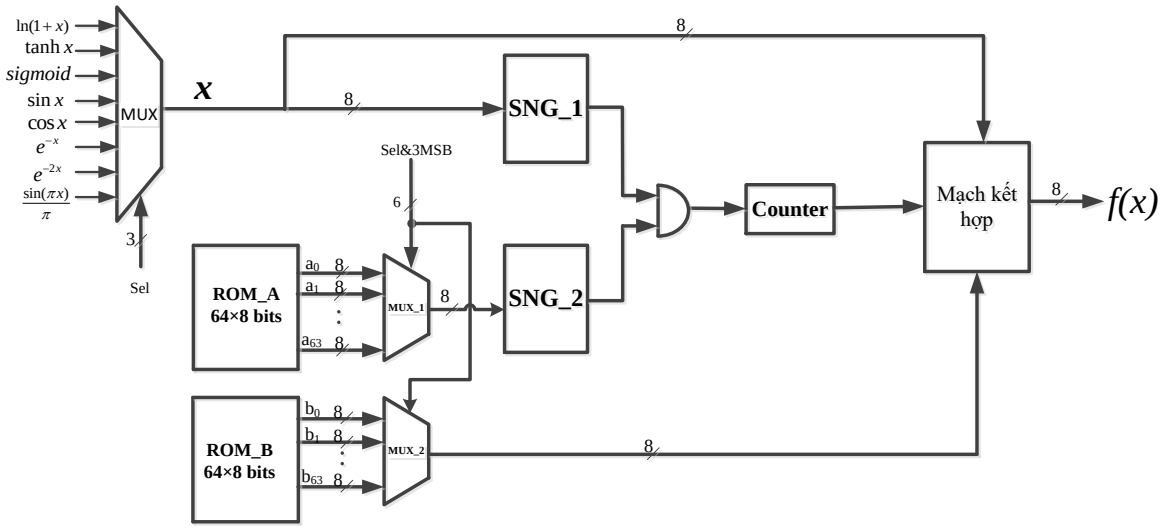
MSB của dữ liệu vào. Kiến trúc phần cứng cho tính toán hàm $\log_2(1+x)$ thể hiện trên Hình 5.27.



Hình 5.27: Kiến trúc phần cứng thực thi hàm $\log_2(1+x)$.

5.4.7. Kiến trúc phần cứng cho tính toán nhiều hàm toán học.

Phương pháp đề xuất cũng có thể cho phép tạo ra một kiến trúc chung cho tính toán nhiều hàm toán học. Bằng việc tăng dung lượng của các bộ nhớ ROM chứa các hệ số và thêm một số mạch logic điều khiển đơn giản. Kiến trúc phần cứng cho tính toán nhiều hàm được thể hiện trên Hình 5.28. Kiến trúc này cho phép tính toán 8 hàm khác nhau. Các bộ nhớ ROM_A và ROM_B có kích thước đều bằng 64×8 bit tương ứng chứa các hệ số xấp xỉ a_i và b_i cho 8 hàm. Ngoài ra cần một tín hiệu điều khiển (Sel) để lựa chọn các đầu vào tương ứng 8 hàm. Mạch kết hợp sẽ gồm một số mạch logic đơn giản để tạo ra tín hiệu đầu ra phù hợp với từng hàm.



Hình 5.28: Kiến trúc phần cứng chung thực thi nhiều hàm toán học.

5.5. Kết quả thực thi và so sánh

Kiến trúc đề xuất tính toán các hàm toán học khác nhau trình bày trong mục 5.4 được tổng hợp trên FPGA và ASIC. Tất cả các hàm được thực thi đều có đầu vào và đầu ra đều có định dạng đơn cực và được biểu diễn bằng 8 bit. Các thanh ghi LFSR sử dụng trong kiến trúc phần cứng là các thanh ghi 8 bit. Kiến trúc cho tính toán mỗi hàm sử dụng hai SNG và hai ROM có kích thước nhỏ để chứa các hệ số. Các kiến trúc được đề xuất trong [97] như trình bày ở mục *5.2.3* cũng được thực thi lại để so sánh. Các kiến trúc so sánh cũng được thực hiện với đầu vào và đầu ra biểu diễn bằng 8 bit, các thanh ghi LFSR là thanh ghi 8 bit. Bảng 5.4 thể hiện kết quả thực thi trên FPGA Virtex6 Xc6Wx75t và được tổng hợp bởi ISE 12.4 Suite. Bảng 5.5 thể hiện kết quả tổng hợp bằng Synopsys Design Compiler cho các hàm khác nhau của phương pháp đề xuất và các kiến trúc so sánh trên thư viện SAED công nghệ CMOS 90 nm. Các kết quả thực thi cho thấy phương pháp đề xuất đạt được hiệu quả cao hơn so với các kiến trúc theo phương pháp trong [97] cả

về tài nguyên, tốc độ và công suất tiêu thụ.

Bảng 5.4: Kết quả thực thi trên FPGA.

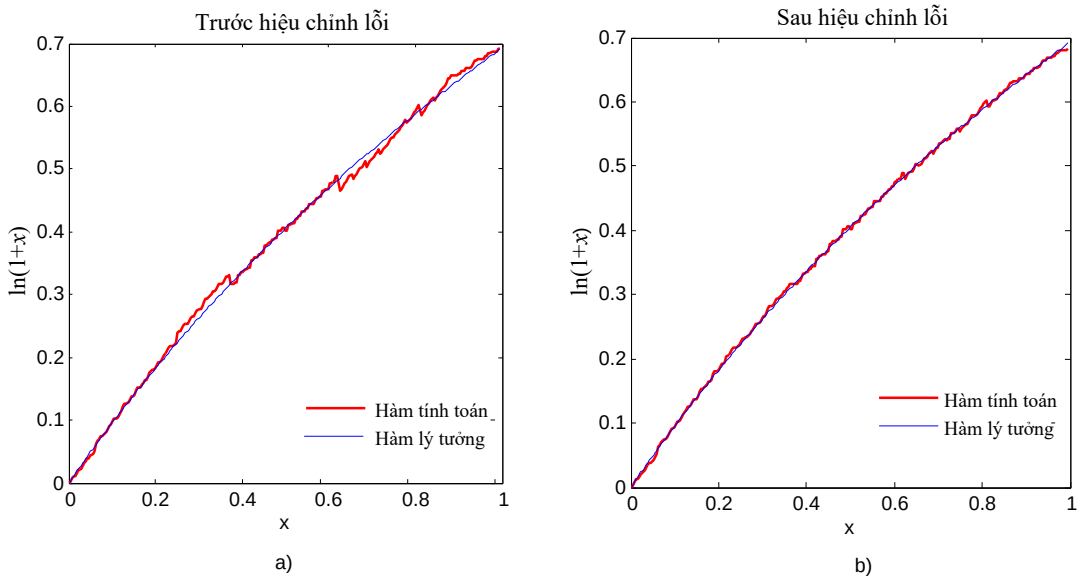
Hàm	Đề xuất			Phương pháp dùng khai triển Maclaurin [97]			
	Slice LUT	Độ trễ (ns)	f_{max} (MHz)	Bậc	slice LUT	Độ trễ (ns)	f_{max} (MHz)
$\ln(1+x)$	79	1,885	530,504	5	86	2,287	437,192
$\tanh x$	78	1,885	530,804	7	87	2,290	436,624
sigmoid(x)	75	2,288	437,006	5	96	2,287	437,192
$\sin x$	77	2,288	437,006	7	88	2,290	436,624
$\cos x$	78	2,288	437,006	8	96	2,287	437,192
e^{-x}	77	1,885	530,504	5	87	2,287	437,192
e^{-2x}	94	2,288	437,006	7	99	2,290	436,624
$\frac{\sin(\pi x)}{\pi}$	81	2,288	437,006	9	64	2,288	437,192
$\log_2(1+x)$	87	2,288	437,006	-	-	-	-
Đa hàm	133	2,288	37,006	-	-	-	-

Bảng 5.5: Kết quả thực thi trên ASIC.

Hàm	Đề xuất			Phương pháp dùng khai triển Maclaurin [97]			
	Diện tích (μm^2)	Độ trễ (ns)	Công suất (μW)	Bậc	Diện tích (μm^2)	Độ trễ (ns)	Công suất (μW)
$\ln(1+x)$	3459	1,69	80,95	5	5381	1,74	101,24
$\tanh x$	3449	1,69	80,34	7	4933	1,74	83,51
sigmoid	3589	1,70	97,89	5	5350	1,74	189,17
$\sin x$	3741	1,7	60,67	7	4933	1,74	83,51
$\cos x$	3746	1,7	65,87	8	5387	1,74	121,73
e^{-x}	3469	1,69	78,43	5	5870	1,74	121,73
e^{-2x}	4227	1,70	135,86	7	6430	1,74	256,55
$\frac{\sin(\pi x)}{\pi}$	4188	1,7	70,30	9	4881	1,74	82,17
$\log_2(1+x)$	4072	1,70	132,32	-	-	-	-
Đa hàm	6949	1,77	298,65	-	-	-	-

5.6. Khảo sát và hiệu chỉnh sai số

Để đánh giá sai số trong tính toán các hàm trên phần cứng đề xuất, một chương trình kiểm tra được viết để nhận được dữ liệu đầu ra cho từng hàm. Với các kiến trúc đề xuất có dữ liệu đầu vào và đầu ra có định dạng 8 bit, dữ liệu đầu ra gồm 256 giá trị tương ứng với 256 giá trị đầu vào.



Hình 5.29: Lỗi trước và sau hiệu chỉnh hàm $\ln(1+x)$.

Hình 5.30a mô tả lỗi của hàm $\ln(1+x)$, giá trị trung bình sai số tuyệt đối (MAE: Mean Absolute Error) của hàm $\ln(1+x)$ là 0,0064. Với các kiến trúc đề xuất cho phép hiệu chỉnh sai số đầu ra. Có thể minh họa kỹ thuật hiệu chỉnh sai số đầu ra thông qua ví dụ trong trường hợp hàm $\ln(1+x)$. Quan sát sai số đầu ra trên Hình 5.29a có thể thấy sai số lớn xảy ra ở các khoảng $x \in (2,5;\ 3,75)$ và x $\in (0,625;\ 1)$, các khoảng giá trị này thuộc các phân đoạn số 2, 5, 6 và 7. Bằng cách điều chỉnh các hệ số a_i và b_i tương ứng với các phân đoạn này cho phép giảm sai số xuống các giá trị nhỏ hơn. Hình 5.29b mô tả sai số của hàm $\ln(1+x)$ sau khi hiệu chỉnh. Giá trị MAE sau

Bảng 5.6: So sánh lỗi của phương pháp đề xuất với các phương pháp khác.

Hàm		Đề xuất	Phương pháp Maclaurin			Đa thức Bernstien			FSM
$\sin x$	Bậc	-	3	5	7	3	5	7	8 trạng thái
	MAE	**0,0026**	0,0016	0,0033	0,0034	0,0136	0,0088	0,0066	0,0025
$\cos x$	Bậc	-	2	4	6	2	4	6	8 trạng thái
	MAE	**0,0035**	0,0082	0,0025	0,0023	0,0356	0,0178	0,0120	0,0053
$\ln(1+x)$	Bậc	-	5	6	7	5	6	7	8 trạng thái
	MAE	**0,0026**	0,0141	0,0109	0,0081	0,0090	0,0076	0,0066	0,0186
$\tanh x$	Bậc	-	3	5	7	3	5	7	8 trạng thái
	MAE	**0,0029**	0,0178	0,0175	0,0140	0,0182	0,0110	0,0082	0,0351
e^{-x}	Bậc	-	4	5	6	4	5	6	8 trạng thái
	MAE	**0,0027**	0,0018	0,0008	0,0008	0,0130	0,0103	0,0086	0,0154
e^{-2x}	Bậc	-	5	6	7	6	7	8	8 trạng thái
	MAE	**0,0027**	0,0019	0,0011	0,0009	0,0195	0,0170	0,0875	0,0423
sigmoid	Bậc	-	5			-			8 trạng thái
	MAE	**0,0024**	0,0046			-			0,0198
$\sin(\pi x)/\pi$	MAE	**0,0027**	-			-			-
$\log_2(1+x)$	MAE	**0,0025**	-			-			-

hiệu chỉnh đạt 0,0026. Thực hiện tương tự với các hàm còn lại, lỗi đạt được sau hiệu chỉnh của các hàm thể hiện trên Bảng 5.6. Trên Bảng 5.6 tham số lỗi của các hàm được đưa ra trong [97] cũng được trình bày để so sánh với phương pháp đề xuất trong luận án này. Lỗi của phương pháp đề xuất đã được cải thiện so với các phương pháp trước đó đối với hầu hết các hàm. Hơn nữa, phương pháp đề xuất được thực thi trên dữ liệu 8 bit (so với 10 bit

trong [97]), khi tăng độ rộng bit của dữ liệu có thể cải thiện hơn về độ chính xác.

5.7. Kết luận chương 5

Chương này này trình bày một phương pháp thiết kế các lõi phần cứng tính toán các hàm toán học dựa trên các kỹ thuật xấp xỉ phân đoạn tuyến tính đều và sử dụng các logic ngẫu nhiên. Kiến trúc phần cứng tính toán các hàm toán học phổ biến đã được thiết kế dựa trên phương pháp đề xuất. Các kết quả thực thi cho thấy, các kiến trúc đề xuất đã cải thiện hiệu năng so với các phương pháp trước đó

KẾT LUẬN VÀ HƯỚNG NGHIÊN CỨU TƯƠNG LAI

Trong luận án này, nghiên cứu sinh đã tiến hành nghiên cứu những kiến thức cơ bản về các phương pháp thực thi phần cứng tính toán các hàm toán học cơ sở. Đồng thời xem xét đặc tính của các ứng dụng xử lý tín hiệu số. Trên cơ sở đó luận án này tập trung vào nghiên cứu các lõi phần cứng hiệu quả cho tính toán các hàm toán học điển hình ứng dụng trong xử lý tín hiệu số. Một số kết quả đạt được của luận án có thể tóm tắt như sau:

A. Một số kết quả đạt được của luận án

1. Đề xuất một phương pháp tính toán hàm logarithm nhị phân dựa trên xấp xỉ phân đoạn tuyến tính đều với các hệ số tối ưu nhằm đạt được lỗi xấp xỉ nhỏ. Ngoài ra, còn sử dụng một bảng LUT để bù sai số nhằm tăng độ chính xác. Trên cơ sở phương pháp đề xuất một kiến trúc tính toán logarithm nhị phân của một số 16 bit đầu vào đã được đưa ra và thực thi trên FPGA và ASIC. Các kết quả thực thi cho thấy bộ chuyển đổi loarithm nhị phân đạt được hiệu quả về tài nguyên và tốc độ;

2. Đề xuất một phương pháp cải tiến cho tính toán hàm *sin* ứng dụng cho bộ tổ hợp tần số trực tiếp dựa trên kỹ thuật xấp xỉ phân đoạn tuyến tính đều kết hợp với bảng LUT. Phương pháp đề xuất tập trung vào cải thiện tỷ lệ nén sin-LUT của bộ chuyển đổi pha-biên độ trong DDFS. Các kết quả thực thi và kiểm chứng đã cho thấy kiến trúc đề xuất đạt được hiệu quả về mặt tài nguyên;

3. Đề xuất một phương pháp tính toán các hàm toán học được ứng dụng phổ biến trong xử lý tín hiệu số dựa trên xấp xỉ hai mức, mức xấp xỉ thứ nhất dựa trên phương pháp xấp xỉ phân đoạn tuyến tính đều và mức xấp xỉ thứ hai là bước xấp xỉ hàm sai số gây ra bởi mức đầu tiên theo phương pháp xấp xỉ phân đoạn tuyến tính có đối xứng. Các thực thi một số hàm toán học điển hình theo phương pháp đề xuất đã cho thấy hiệu quả về tốc độ . Vì vậy, kiến trúc đề xuất là phù hợp với các ứng dụng đòi hỏi tốc độ thực thi cao.

4. Đề xuất một phương pháp tính toán các hàm toán học dựa trên tính toán ngẫu nhiên kết hợp với xấp xỉ phân đoạn tuyến tính đều. Dựa trên phương pháp này 9 hàm toán học sử dụng phổ biến trong ứng dụng xử lý tín hiệu số và mạng nơ-ron đã được thực thi. Đồng thời một kiến trúc chung cho tính toán nhiều hàm toán học dựa trên phương pháp đề xuất cũng được thiết kế và thực thi. Các kết quả tổng hợp trên FPGA và ASIC cho thấy hiệu quả của phương pháp đề xuất so với các nghiên cứu tương tự.

B. Hướng phát triển tiếp theo

Dựa trên các kết quả nghiên cứu trong luận án này, một số hướng nghiên cứu có thể phát triển trong tương lai là:

- Các hệ thống DSP hiện đại yêu cầu các thành phần tính toán có tài nguyên thấp và hiệu năng cao vì vậy các kiến trúc hiệu quả cho các hàm toán học khác cần tiếp tục nghiên cứu và nâng cao hiệu năng cho các ứng dụng chuyên biệt.

- Phương pháp kết hợp xấp xỉ các hàm bằng phân đoạn tuyến tính với

một LUT có kích thước nhỏ để bù sai số cho thấy hiệu quả với các ứng dụng đòi hỏi chi phí phần cứng thấp. Có thể phát triển phương pháp bằng cách tìm một thuật toán chung có tính hệ thống cho thực thi các hàm toán học. Trong đó, các tham số như độ rộng bit dữ liệu đầu vào, kích thước của LUT bù sai số, số phân đoạn xấp xỉ sẽ được tối ưu theo độ chính xác yêu cầu.

- Nghiên cứu ứng dụng các lõi tính toán đề xuất trong luận án vào các ứng dụng như: xử lý tiếng nói, xử lý ảnh, đồ họa...

- Các lõi tính toán được thiết kế trong luận án này dựa trên SC được thực hiện trên các hàm có đầu vào và đầu ra dạng đơn cực. Do vậy có thể tiếp tục nghiên cứu thực thi với các hàm có đầu vào và đầu ra dạng lưỡng cực. Hơn nữa, sử dụng SC là giải pháp hứa hẹn trong nhiều ứng dụng như mạng nơ-ron, xử lý ảnh... Do vậy, nghiên cứu thiết kế các lõi tính toán dựa trên SC cho các ứng dụng này là một hướng có thể tiếp tục nghiên cứu phát triển. Đặc biệt các kiến trúc song song dựa trên SC là một hướng nghiên cứu mới có thể phát triển cho nhiều ứng dụng.

DANH MỤC CÁC CÔNG TRÌNH ĐÃ CÔNG BỐ

A. Các công trình liên quan trực tiếp đến luận án

1. **Sai Van Thuan**, Hoang Van phuc, Tran Van Khan "An Efficient Hardware Logarithm Generator With An Optimized Difference Method for Digital Signal Processing" *Tạp chí Khoa học và Kỹ thuật, Học viện Kỹ thuật quận sự,*Trang 59-68, số 174, tháng 02 năm 2016
2. **V. T. Sai** and V. P. Hoang, "An optimized implementation of logarithm hardware generator for digital signal processing," *in 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, 2016, pp. 153-157.
3. **Sái Văn Thuận**, Hoàng Văn Phúc, Trần Văn Khẩn, Dương Quang Mạnh, "Thiết kế bộ tổ hợp tần số trực tiếp sử dụng phương pháp độ chênh lệch tuyến tính với xấp xỉ phân đoạn đều và các hệ số tối ưu," *Tạp chí Nghiên cứu khoa học và Công nghệ quân sự,*, số 42,Trang 103-110, tháng 4 năm 2016.
4. **Sái Văn Thuận**, Hoàng Văn Phúc, Trần Văn Khẩn, "Thiết kế phần cứng tính toán các hàm toán học dựa trên xấp xỉ tuyến tính hai mức," *Tạp chí Nghiên cứu khoa học và Công nghệ quân sự,*, số 49,Trang 53-60, tháng 6 năm 2017.
5. **Sái Văn Thuận**, Hoàng Văn Phúc, Trần Văn Khẩn và Nguyễn Minh Tú "Thực thi phần cứng tính toán các hàm toán học sử dụng logic ngẫu

nhiên và xấp xỉ phân đoạn tuyến tính đều" *Tạp chí Khoa học và Kỹ thuật, Học viện Kỹ thuật quận sự*, Trang 180-190, số 187, tháng 12 năm 2017.

6. Van-Tinh Nguyen, Van-Phuc Hoang, **Van-Thuan Sai**, Tieu-Khanh Luong, Minh-Tu Nguyen and Han Le Duc, "A New Approach of Stochastic Computing for Arithmetic Functions in Wideband RF Transceivers" *60th IEEE International Midwest Symposium on Circuits and Systems (MWCAS)*, Boston, USA, 2017, pp. 1525-1528.

B. Các công trình sử dụng tham khảo trong luận án

1. Hoang Van Phuc, **Sai Van Thuan**, Pham Cong Kha, "Low Area Asic Implementation of Logarithm Approximation for Digital Signal Processing," *Chuyên san công nghệ thông tin và truyền thông, Học viện kỹ thuật quân sự*, trang 42-57, tháng 10 năm 2015.

2. **Sái Văn Thuận**, Hoàng Văn Phúc, Trần văn Khẩn, "Phương Pháp Chênh Lệch Trong Hiện Thực Hóa Các Hàm Phức Tạp Trên ASIC Cho Các Hệ Thống DSP," *Hội thảo Quốc gia 2015 về Điện tử, Truyền thông và Công nghệ thông tin(ECIT 2015)*, Thành phố hồ Chí minh, Việt Nam, Trang 267-272, tháng 12 năm 2015.

TÀI LIỆU THAM KHẢO

[1] K. Yoshida, T. Sakamoto, and T. Hase, "A 3D graphics library for 32-bit microprocessors for embedded systems," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 1107–1114, 1998.

[2] B. G. Nam, H. Kim, and H. J. Yoo, "Power and area-efficient unified computation of vector and elementary functions for handheld 3D graphics systems," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 490–504, 2008.

[3] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scale, "Altivec extension to powerpc accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–95, 2000.

[4] D. Harris, "An exponentiation unit for an opengl lighting engine," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 251–258, 2004.

[5] I. Nobuhiro, M. Hirano, E. Yukio, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki, "2.44 GFLOPS 300MHz floating-point vector processing unit for high performance 3D graphics computing," in *Proceedings of the 25th European Solid-State Circuits Conference*, Conference Proceedings, pp. 106–109.

[6] S. Oberman, G. Favor, and F. Weber, "AMD 3DNow technology: architecture and implementations," *IEEE Micro*, vol. 19, no. 2, pp. 37–48, 1999.

[7] J. Y. Stein, *Digital signal processing: a computer science perspective*. John Wiley & Sons, Inc., 2000.

[8] J.-C. Wang, J.-F. Wang, and Y.-S. Weng, "Chip design of mfcc extraction for speech recognition," *INTEGRATION, the VLSI journal*, vol. 32, no. 1, pp. 111–131, 2002.

[9] J. A. Pineiro, S. F. Oberman, J. M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 304–318, 2005.

[10] A. Kunimatsu, N. Ide, T. Sato, Y. Endo, H. Murakami, T. Kamei, M. Hirano, F. Ishihara, H. Tago, M. Oka, A. Ohba, T. Yutaka, T. Okada, and M. Suzuoki, "Vector unit architecture for emotion synthesis," *IEEE Micro*, vol. 20, no. 2, pp. 40–47, 2000.

[11] D. M. Lewis, "114 MFLOPS logarithmic number system arithmetic unit for DSP applications," in *Solid-State Circuits Conference, 1995. Digest of Technical Papers. 41st ISSCC, 1995 IEEE International*, Conference Proceedings, pp. 86–87.

[12] S. Hyun-Chul, L. Jin-Aeon, and K. Lee-Sup, "A minimized hardware architecture of fast phong shader using taylor series approximation in 3d graphics," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, Conference Proceedings, pp. 286–291.

[13] E. Remes, "Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation," *CR Acad. Sci. Paris*, vol. 198, pp. 2063–2065, 1934.

[14] N. Sidahoao, G. A. Constantinides, and P. Y. Cheung, "Architectures for function evaluation on FPGAs," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, Conference Proceedings, pp. II–804–II–807 vol.2.

[15] J. A. Pineiro, J. D. Bruguera, and J. M. Muller, "Faithful powering computation using table look-up and a fused accumulation tree," in *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, Conference Proceedings, pp. 40–47.

[16] D. D. Sarma and D. W. Matula, "Faithful interpolation in reciprocal tables," in *Proceedings 13th IEEE Sympsoium on Computer Arithmetic*, Conference Proceedings, pp. 82–91.

[17] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 842–847, 1999.

[18] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 21, no. 2, pp. 167–177, 1999.

[19] F. d. Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 319–330, 2005.

[20] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959.

[21] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference.* ACM, Conference Proceedings, pp. 379–385.

[22] I. Koren and O. Zinaty, "Evaluating elementary functions in a numerical coprocessor based on rational approximations," *IEEE Transactions on Computers*, vol. 39, no. 8, pp. 1030–1037, 1990.

[23] T. Stouraitis and V. Paliouras, "Considering the alternatives in low-power design," *IEEE Circuits and Devices Magazine*, vol. 17, no. 4, pp. 22–29, 2001.

[24] M. G. Arnold, T. A. Bailey, J. R. Cowles, and J. J. Cupal, "Redundant logarithmic arithmetic," *IEEE Transactions on Computers*, vol. 39, no. 8, pp. 1077–1086, 1990.

[25] M. G. Arnold and P. D. Vouzis, "A serial logarithmic number system alu," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, Conference Proceedings, pp. 151–156.

[26] J. M. Muller, A. Scherbyna, and A. Tisserand, "Semi-logarithmic number systems," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 145–151, 1998.

[27] E. E. Swartzlander and A. G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Transactions on Computers*, vol. C-24, no. 12, pp. 1238–1242, 1975.

[28] E. E. J. Swartzlander, D. V. S. Chandra, H. T. J. Nagle, and S. A. Starks, "Sign/logarithm arithmetic for FFT implementation," *IEEE Transactions on Computers*, vol. C-32, no. 6, pp. 526–534, 1983.

[29] P. Huang, D. H. Y. Teng, K. Wahid, and S. B. Ko, "Convergence analysis of jacobi iterative method using logarithmic number system," in *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, Conference Proceedings, pp. 27–32.

[30] P. Lee, "An evaluation of a hybrid-logarithmic number system DCT/IDCT algorithm [image compression applications]," in *2005 IEEE International Symposium on Circuits and Systems*, Conference Proceedings, pp. 4863–4866 Vol. 5.

[31] B. G. Nam, H. Kim, and H. J. Yoo, "A low-power unified arithmetic unit for programmable handheld 3-D graphics systems," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 8, pp. 1767–1778, 2007.

[32] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[33] M. Combet, H. V. Zonneveld, and L. Verbeek, "Computation of the base two logarithm of binary numbers," *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 6, pp. 863–867, 1965.

[34] E. L. Hall, D. D. Lynch, and S. J. Dwyer, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Transactions on Computers*, vol. C-19, no. 2, pp. 97–105, 1970.

[35] S. L. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, "A fast, low-power logarithm approximation with CMOS VLSI implementa-

tion," in *42nd Midwest Symposium on Circuits and Systems (Cat. No.99CH36356)*, vol. 1, Conference Proceedings, pp. 388–391 vol. 1.

[36] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421–1433, 2003.

[37] H. Kim, B. G. Nam, J. H. Sohn, J. H. Woo, and H. J. Yoo, "A 231-MHz, 2.18-mw 32-bit logarithmic arithmetic unit for fixed-point 3-D graphics system," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2373–2381, 2006.

[38] Z. Li, J. An, M. Yang, and Y. Jing, "FPGA design and implementation of an improved 32-bit binary logarithm converter," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, Conference Proceedings, pp. 1–4.

[39] T. B. Juang, S. H. Chen, and H. J. Cheng, "A lower error and ROM-free logarithmic converter for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 12, pp. 931–935, 2009.

[40] R. Gutierrez and J. Valls, "Low cost hardware implementation of logarithm approximation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2326–2330, 2011.

[41] D. D. Caro, N. Petra, and A. G. M. Strollo, "Efficient logarithmic converters for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 10, pp. 667–671, 2011.

[42] T. Lang and E. Antelo, "High-throughput CORDIC-based geometry operations for 3D computer graphics," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 347–361, 2005.

[43] P. K. Meher, J. Valls, T. B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

[44] J. A. Pineiro, M. D. Ercegovac, and J. D. Bruguera, "Algorithm and architecture for logarithm, exponential, and powering computation," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1085–1096, 2004.

[45] T. K. Rodrigues and J. E. E. Swartzlander, "Adaptive CORDIC: Using parallel angle recoding to accelerate rotations," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 522–531, 2010.

[46] M. G. B. Sumanasena, "A scale factor correction scheme for the CORDIC algorithm," *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1148–1152, 2008.

[47] NVIDIA, "Tesla C2075 companion processor tesla C2075 datasheet," 2001.

[48] V. P. Hoang and C. K. Pham, "Novel quasi-symmetrical approach for efficient logarithmic and anti-logarithmic converters," in *PRIME 2012; 8th Conference on Ph.D. Research in Microelectronics & Electronics*, Conference Proceedings, pp. 1–4.

[49] J. Vankka, "Methods of mapping from phase to sine amplitude in direct digital synthesis," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 44, no. 2, pp. 526–534, 1997.

[50] J. M. P. Langlois and D. Al-Khalili, "Phase to sinusoid amplitude conversion techniques for direct digital frequency synthesis," *IEE Proceedings - Circuits, Devices and Systems*, vol. 151, no. 6, pp. 519–528, 2004.

[51] J. Tierney, C. Rader, and B. Gold, "A digital frequency synthesizer," *IEEE Transactions on Audio and Electroacoustics*, vol. 19, no. 1, pp. 48–57, 1971.

[52] B. J. Hutchinson, *Frequency Synthesis and Applications.* New York:: IEEE Press, 1975.

[53] D. A. Sunderland, R. A. Strauch, S. S. Wharfield, H. T. Peterson, and C. R. Cole, "CMOS/SOS frequency synthesizer LSI circuit for spread spectrum communications," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 4, pp. 497–506, 1984.

[54] H. T. Nicholas, H. Samueli, and B. Kim, "The optimization of direct digital frequency synthesizer performance in the presence of finite word length effects," in *Proceedings of the 42nd Annual Frequency Control Symposium, 1988.*, Conference Proceedings, pp. 357–363.

[55] F. Curticapean, K. I. Palomaki, and J. Niittylahti, "Direct digital frequency synthesiser with high memory compression ratio," *Electronics Letters*, vol. 37, no. 21, pp. 1275–1277, 2001.

[56] P. R. Symons, "DDFS phase mapping technique," *Electronics Letters*, vol. 38, no. 21, pp. 1291–1292, 2002.

[57] R. Freeman, "'digital sine conversion circuit for use in direct digital synthesizers," *U.S. Patent 4,809,205,28 February 1989.*

[58] A. Bellaouar, M. S. O. brecht, A. M. Fahim, and M. I. Elmasry, "Low-power direct digital frequency synthesis for wireless communications," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, pp. 385–390, 2000.

[59] S. I. Liu, T. B. Yu, and H. W. Tsao, "Pipeline direct digital frequency synthesiser using decomposition method," *IEE Proceedings - Circuits, Devices and Systems*, vol. 148, no. 3, pp. 141–144, 2001.

[60] J. M. P. Langlois and D. Al-Khalili, "Novel approach to the design of direct digital frequency synthesizers based on linear interpolation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 9, pp. 567–578, 2003.

[61] L. Weaver and R. Kerr, "'high resolution phase to sine amplitude conversion', u.s. patent 4 905 177, 27 february 1990."

[62] L. Fanucci, R. Roncella, and R. Saletti, "A sine wave digital synthesizer based on a quadratic approximation," in *Proceedings of the 2001 IEEE International Frequncy Control Symposium and PDA Exhibition (Cat. No.01CH37218)*, Conference Proceedings, pp. 806–810.

[63] D. De Caro, E. Napoli, and A. Strollo, "Rom-less direct digital frequency synthesizers exploiting polynomial approximation," in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 2. IEEE, 2002, pp. 481–484.

[64] Q. K. Omran, M. T. Islam, N. Misran, and M. R. I. Faruque, "A ROM-less direct digital frequency synthesizer based on hybrid polynomial approximation," *The Scientific World Journal*, vol. 2014, 2014.

[65] A. Yamagishi, M. Ishikawa, T. Tsukahara, and S. Date, "A 2-V, 2-GHz low-power direct digital frequency synthesizer chip-set for wireless communication," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 2, pp. 210–217, 1998.

[66] A. M. Sodagar and G. R. Lahiji, "Mapping from phase to sine-amplitude in direct digital frequency synthesizers using parabolic approximation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 12, pp. 1452–1457, 2000.

[67] J. M. P. Langlois and D. Al-Khalili, "ROM size reduction with low processing cost for direct digital frequency synthesis," in *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (IEEE Cat. No.01CH37233)*, vol. 1, Conference Proceedings, pp. 287–290 vol.1.

[68] L. W. Hsu and D. C. Chang, "Design of direct digital frequency synthesizer with high rom compression ratio," in *2005 12th IEEE International Conference on Electronics, Circuits and Systems*, Conference Proceedings, pp. 1–4.

[69] V.-P. Hoang and P. Cong-Kha, "An improved linear difference method with high ROM compression ratio in direct digital frequency synthesizer," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 94, no. 3, pp. 995–998, 2011.

[70] P. T. P. Tang, "Table-lookup algorithms for elementary functions and their error analysis," in *[1991] Proceedings 10th IEEE Symposium on Computer Arithmetic*, Conference Proceedings, pp. 232–236.

[71] A. S. Noetzel, "An interpolating memory unit for function evaluation: analysis and design," *IEEE Transactions on Computers*, vol. 38, no. 3, pp. 377–384, 1989.

[72] G. H. Garcia and W. J. Kubitz, "Minimum mean running time function generation using read-only memory," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 147–156, 1983.

[73] M. J. Schulte and E. E. Swartzlander, "Hardware designs for exactly rounded elementary functions," *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 964–973, 1994.

[74] E. G. Walters and M. J. Schulte, "Efficient function approximation using truncated multipliers and squarers," in *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, Conference Proceedings, pp. 232–239.

[75] S. F. Hsiao, H. J. Ko, and C. S. Wen, "Two-level hardware function evaluation based on correction of normalized piecewise difference functions," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 5, pp. 292–296, 2012.

[76] J.-M. Muller, "A few results on table-based methods," in *Developments in Reliable Computing*. Springer, 1999, pp. 279–288.

[77] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 149–156.

[78] A. Alaghi, "The logic of random pulses: Stochastic computing," Ph.D. dissertation, University of Michigan, 2015.

[79] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[80] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

[81] B. D. Brown and H. C. Card, "Stochastic neural computation. i. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[82] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 1. IEEE, 2000, pp. 599–602.

[83] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5617–5626, 2011.

[84] X.-R. Lee, C.-L. Chen, H.-C. Chang, and C.-Y. Lee, "A 7.92 Gb/s 437.2 mW stochastic LDPC decoder chip for IEEE 802.15. 3c applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 2, pp. 507–516, 2015.

[85] B. Yuan and K. K. Parhi, "Reduced-latency LLR-based SC list decoder for polar codes," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 107–110.

[86] T.-H. Chen and J. P. Hayes, "Design of stochastic Viterbi decoders for convolutional codes," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2013 IEEE 16th International Symposium on*. IEEE, 2013, pp. 66–71.

[87] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, "A new stochastic computing methodology for efficient neural network implementation," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 3, pp. 551–564, 2016.

[88] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "FPGA implementation of a deep belief network architecture for character recognition using stochastic computation," in *Information Sciences and Systems (CISS), 2015 49th Annual Conference on*. IEEE, 2015, pp. 1–5.

[89] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 880–883.

[90] V. K. Chippa, S. Venkataramani, K. Roy, and A. Raghunathan, "Storm: a stochastic recognition and mining processor," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 39–44.

[91] A. Morro, V. Canals, A. Oliver, M. L. Alomar, and J. L. Rossello, "Ultra-fast data-mining hardware architecture based on stochastic computing," *PloS one*, vol. 10, no. 5, p. e0124176, 2015.

[92] S. Gupta and G. Kailash, "Revisiting stochastic computation: approximate estimation of machine learning kernels," in *Workshop on Approximate Computing Across the System Stack*, 2014.

[93] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. IEEE, 2011, pp. 154–161.

[94] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[95] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester, "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering," in *Custom Integrated Circuits Conference (CICC), 2014 IEEE Proceedings of the*. IEEE, 2014, pp. 1–4.

[96] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2012, pp. 480–487.

[97] K. Parhi and Y. Liu, "Computing arithmetic functions using stochastic logic by series expansion," *IEEE Transactions on Emerging Topics in Computing*, 2016.