

ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

**Thiết kế và triển khai bộ mã hóa tích chập và giải mã
Viterbi được tăng tốc bởi phần cứng dưới dạng server**

VŨ TUẤN MINH

minh.vt214015@sis.hust.edu.vn

**Ngành Điện tử - Viễn thông
Chuyên ngành Thiết kế vi mạch**

Giảng viên hướng dẫn: PGS. TS. Nguyễn Hữu Trung



Chữ kí GVHD

Khoa: Kỹ thuật truyền thông

Trường: Điện - Điện tử

HÀ NỘI, 07/2025

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

ĐÁNH GIÁ ĐỒ ÁN TỐT NGHIỆP
(DÀNH CHO CÁN BỘ HƯỚNG DẪN)

Tên đề tài: **Thiết kế và triển khai bộ mã hóa tích chập và giải mã Viterbi được tăng tốc bởi phần cứng dưới dạng server**

Họ tên SV: Vũ Tuấn Minh

MSSV: 20214015

Cán bộ hướng dẫn: PGS.TS Nguyễn Hữu Trung

STT	Tiêu chí (Điểm tối đa)	Hướng dẫn đánh giá tiêu chí	Điểm tiêu chí
1	Thái độ làm việc (2,5 điểm)	Nghiêm túc, tích cực và chủ động trong quá trình làm ĐATN	
		Hoàn thành đầy đủ và đúng tiến độ các nội dung được GVHD giao	
2	Kỹ năng viết quyển ĐATN (2 điểm)	Trình bày đúng mẫu quy định, bố cục các chương logic và hợp lý: Bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy, có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn, v.v.	
		Kỹ năng diễn đạt, phân tích, giải thích, lập luận: Cấu trúc câu rõ ràng, văn phong khoa học, lập luận logic và có cơ sở, thuật ngữ chuyên ngành phù hợp, v.v.	
3	Nội dung và kết quả đạt được (5 điểm)	Nêu rõ tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài, các vấn đề và các giả thuyết, phạm vi ứng dụng của đề tài. Thực hiện đầy đủ quy trình nghiên cứu: Đặt vấn đề, mục tiêu đề ra, phương pháp nghiên cứu/ giải quyết vấn đề, kết quả đạt được, đánh giá và kết luận.	
		Nội dung và kết quả được trình bày một cách logic và hợp lý, được phân tích và đánh giá thỏa đáng. Biện luận phân tích kết quả mô phỏng/ phần mềm/ thực nghiệm, so sánh kết quả đạt được với kết quả trước đó có liên quan.	
		Chỉ rõ phù hợp giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận đề đề xuất hướng giải quyết có thể thực hiện trong tương lai. Hàm lượng khoa học/ độ phức tạp cao, có tính mới/tính sáng tạo trong nội dung và kết quả đồ án.	
4	Điểm thành tích (1 điểm)	Có bài báo KH được đăng hoặc chấp nhận đăng/ đạt giải SV NCKH giải 3 cấp Trường trở lên/ Các giải thưởng khoa học trong nước, quốc tế từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế. (1 điểm)	
		Được báo cáo tại hội đồng cấp Trường trong hội nghị SV NCKH nhưng không đạt giải từ giải 3 trở lên/ Đạt giải khuyến khích trong cuộc thi khoa học trong nước, quốc tế/ Kết quả đồ án là sản phẩm ứng dụng có tính hoàn thiện cao, yêu cầu khối lượng thực hiện lớn. (0,5 điểm)	
		Điểm tổng các tiêu chí:	
		Điểm hướng dẫn:	

Cán bộ hướng dẫn
(Ký và ghi rõ họ tên)

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

ĐÁNH GIÁ ĐỒ ÁN TỐT NGHIỆP
(DÀNH CHO CÁN BỘ PHẢN BIỆN)

Tên đề tài: **Thiết kế và triển khai bộ mã hóa tích chập và giải mã Viterbi được tăng tốc bởi phần cứng dưới dạng server**

Họ tên SV: **Vũ Tuấn Minh**

MSSV: **20214015**

Cán bộ phản biện:

STT	Tiêu chí (Điểm tối đa)	Hướng dẫn đánh giá tiêu chí	Điểm tiêu chí
1	Trình bày quyển ĐATN (4 điểm)	<p>Đồ án trình bày đúng mẫu quy định, bố cục các chương logic và hợp lý: Bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy, có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn, v.v.</p> <p>Kỹ năng diễn đạt, phân tích, giải thích, lập luận: cấu trúc câu rõ ràng, văn phong khoa học, lập luận logic và có cơ sở, thuật ngữ chuyên ngành phù hợp, v.v.</p>	
2	Nội dung và kết quả đạt được (5,5 điểm)	<p>Nêu rõ tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài, các vấn đề và các giả thuyết, phạm vi ứng dụng của đề tài. Thực hiện đầy đủ quy trình nghiên cứu: Đặt vấn đề, mục tiêu đề ra, phương pháp nghiên cứu/ giải quyết vấn đề, kết quả đạt được, đánh giá và kết luận.</p> <p>Nội dung và kết quả được trình bày một cách logic và hợp lý, được phân tích và đánh giá thỏa đáng. Biện luận phân tích kết quả mô phỏng/ phần mềm/ thực nghiệm, so sánh kết quả đạt được với kết quả trước đó có liên quan.</p> <p>Chỉ rõ phù hợp giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai. Hàm lượng khoa học/ độ phức tạp cao, có tính mới/ tính sáng tạo trong nội dung và kết quả đồ án.</p>	
3	Điểm thành tích (1 điểm)	<p>Có bài báo KH được đăng hoặc chấp nhận đăng/ đạt giải SV NCKH giải 3 cấp Trường trở lên/ Các giải thưởng khoa học trong nước, quốc tế từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế. (1 điểm)</p> <p>Được báo cáo tại hội đồng cấp Trường trong hội nghị SV NCKH nhưng không đạt giải từ giải 3 trở lên/ Đạt giải khuyến khích trong cuộc thi khoa học trong nước, quốc tế/ Kết quả đồ án là sản phẩm ứng dụng có tính hoàn thiện cao, yêu cầu khối lượng thực hiện lớn. (0,5 điểm)</p>	
Điểm tổng các tiêu chí:			
Điểm phản biện:			

Cán bộ phản biện
(Ký và ghi rõ họ tên)

LỜI CAM KẾT

Tôi – *Vũ Tuấn Minh* – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *PGS. TS. Nguyễn Hữu Trung*. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

Hà Nội, ngày tháng năm

Tác giả ĐATN

LỜI CẢM ƠN

Đồ án tốt nghiệp cử nhân là một cột mốc đáng ghi nhớ trong cuộc đời mỗi người, nó đánh dấu sự chuyển hóa về chất bắt nguồn từ những thay đổi từ từ về lượng, nó là minh chứng cho thành quả của cả một quá trình dày công học tập nghiên cứu ở bậc đại học. Để đồ án này được hoàn thành một cách trọn vẹn đã có sự góp sức của vô số người. Tôi xin cảm ơn gia đình đã giúp tôi có cơ hội để học tập và phát triển bản thân. Tôi xin cảm ơn các thầy, cô đang công tác tại Đại học Bách Khoa Hà Nội nói chung và các thầy, cô thuộc Trường Điện – Điện tử nói riêng đã tận tình truyền đạt những kiến thức không chỉ là nền tảng quan trọng cho quá trình hình thành nên đồ án mà còn là hành trang quý giá trên con đường sự nghiệp sau này. Đặc biệt, tôi xin trân trọng gửi lời cảm ơn tới thầy Nguyễn Hữu Trung, giảng viên khoa Kỹ thuật truyền thông và cô Nguyễn Thúy Anh, giảng viên khoa Điện tử thuộc Trường Điện – Điện tử, Đại học Bách Khoa Hà Nội, những người đã hướng dẫn và sẵn sàng đưa ra những lời góp ý, trợ giúp khi tôi gặp khó khăn trong quá trình làm đồ án. Và cuối cùng, tôi xin cảm ơn chính bản thân mình đã có đủ bản lĩnh để đương đầu với thử thách và đủ thông thái để vượt qua những thử thách đó!

TÓM TẮT NỘI DUNG ĐỒ ÁN

Trong bối cảnh cách mạng công nghiệp 4.0, nhu cầu truyền dẫn dữ liệu tốc độ cao với độ tin cậy cao và tiết kiệm năng lượng đã trở thành yêu cầu cấp thiết cho các hệ thống viễn thông, IoT và an ninh quốc phòng. Mặc dù các thuật toán mã hóa tích chập và giải mã Viterbi đã được nghiên cứu rộng rãi để đảm bảo truyền dẫn tin cậy trong môi trường nhiễu, các giải pháp phần mềm truyền thống thường gặp hạn chế về tốc độ xử lý và độ trễ. Các nghiên cứu hiện tại tập trung chủ yếu vào tối ưu hiệu năng phần cứng trên FPGA nhưng chưa giải quyết được bài toán triển khai thực tế dưới dạng hệ thống server có khả năng truy cập từ xa, dẫn đến những hạn chế về tính linh hoạt và khả năng tích hợp vào các hệ thống lớn. Để khắc phục những hạn chế này, đồ án đề xuất hướng tiếp cận mới bằng cách kết hợp sức mạnh xử lý song song của FPGA với kiến trúc server hiện đại. Lựa chọn này xuất phát từ nhu cầu thực tế về một giải pháp vừa đảm bảo hiệu năng xử lý cao của phần cứng chuyên dụng, vừa cung cấp khả năng truy cập từ xa và dễ dàng sử dụng. Giải pháp được đề xuất bao gồm thiết kế hệ thống server FPGA tích hợp bộ mã hóa tích chập và giải mã Viterbi với khả năng truyền tải dữ liệu từ xa thông qua giao thức TCP và kiến trúc xử lý thích ứng cho phép cấu hình động các thông số mã hóa/giải mã. Đóng góp chính của đồ án là thiết kế và triển khai thành công hệ thống server FPGA hoàn chỉnh có khả năng cung cấp dịch vụ mã hóa/giải mã Viterbi với thông lượng đạt trên 66 Mbps, đồng thời cung cấp giải pháp ứng dụng thân thiện với người dùng không chuyên về phần cứng. Kết quả thực nghiệm cho thấy hệ thống không chỉ duy trì được ưu điểm về hiệu năng xử lý của FPGA mà còn khắc phục được các hạn chế về tính linh hoạt và khả năng tích hợp của các giải pháp phần cứng thuần túy trước đây, mở ra hướng ứng dụng mới trong các hệ thống truyền thông hiện đại yêu cầu cả hiệu suất cao và khả năng quản lý tập trung.

Sinh viên thực hiện
(Ký và ghi rõ họ tên)

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Đặt vấn đề.....	1
1.2 Khảo sát đề tài.....	1
1.3 Định hướng nghiên cứu	2
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG	3
2.1 Cơ sở lý thuyết	3
2.1.1 Mã hóa tích chập.....	3
2.1.2 Giải mã Viterbi	3
2.2 Lựa chọn thiết kế	4
2.3 Công nghệ sử dụng	6
2.3.1 SoC	6
2.3.2 AXI	7
2.3.3 DMA	7
2.3.4 BRAM.....	7
2.3.5 ILA.....	7
2.3.6 TCP	8
2.3.7 lwIP.....	8
2.3.8 Mesh Network	8
CHƯƠNG 3. THIẾT KẾ, TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG	9
3.1 Thiết kế kiến trúc.....	9
3.1.1 Mô hình hệ thống.....	9
3.1.2 Thiết kế tổng quan.....	9
3.2 Thiết kế PL	10
3.2.1 Thiết kế khối endec_interface.....	11

3.2.2 Thiết kế khối endec	13
3.2.3 Thiết kế khối control	14
3.2.4 Thiết kế khối slice	16
3.2.5 Thiết kế khối conv_encoder	17
3.2.6 Thiết kế khối branch_metric.....	18
3.2.7 Thiết kế khối add_compare_select	19
3.2.8 Thiết kế khối memory	20
3.2.9 Thiết kế khối traceback.....	21
3.3 Thiết kế PS	21
3.4 Triển khai và kiểm thử	23
3.4.1 Công cụ sử dụng	23
3.4.2 Kiểm thử bằng mô phỏng.....	24
3.4.3 Triển khai trên phần cứng và kiểm thử bằng ILA.....	25
3.4.4 Triển khai phần mềm nhúng và kiểm thử trực tiếp.....	28
3.4.5 Kiểm thử toàn diện hệ thống	31
3.5 Tổng kết.....	34
3.5.1 Đánh giá thông số hoạt động của Endec Server.....	34
3.5.2 So sánh với các hệ thống khác	36
CHƯƠNG 4. KẾT LUẬN	39
4.1 Đóng góp nổi bật	39
4.1.1 Tổng quát hóa thuật toán mã hóa tích chập và giải mã Viterbi.....	39
4.1.2 Kết hợp mã hóa tích chập và giải mã Viterbi.....	39
4.2 Thảo luận.....	40
4.2.1 Tiềm năng cho giải mã quyết định mềm	40
4.2.2 Khả năng tương thích giữa chiều dài ràng buộc 3 và Radix-4.....	40
4.2.3 Vấn đề về nghẽn tín hiệu trong triển khai thiết kế	41

4.2.4 Tối ưu khối add_compare_select.....	41
4.2.5 Tối ưu khối memory	42
4.3 Kết luận.....	42
4.4 Hướng phát triển.....	43
TÀI LIỆU THAM KHẢO.....	

DANH MỤC HÌNH VẼ

Hình 2.1	Quy trình trong phương pháp Đồng thiết kế cứng mềm	6
Hình 3.1	Mô hình hệ thống	9
Hình 3.2	Luồng dữ liệu trong Endec Server	10
Hình 3.3	Luồng dữ liệu trong khối endec_interface	11
Hình 3.4	Sơ đồ khối endec_interface	11
Hình 3.5	Sơ đồ tuần tự khối endec_interface	13
Hình 3.6	Sơ đồ khối endec	13
Hình 3.7	Sơ đồ khối control	14
Hình 3.8	Trạng thái máy khối control	15
Hình 3.9	Sơ đồ khối slice	16
Hình 3.10	Sơ đồ khối conv_encoder	17
Hình 3.11	Lưu đồ khối conv_encoder	18
Hình 3.12	Sơ đồ khối branch_metric	18
Hình 3.13	Sơ đồ khối add_compare_select	19
Hình 3.14	Lưu đồ khối add_compare_select	19
Hình 3.15	Sơ đồ khối memory	20
Hình 3.16	Lưu đồ khối memory	20
Hình 3.17	Sơ đồ khối traceback	21
Hình 3.18	Luồng dữ liệu trong khối PS	22
Hình 3.19	Lưu đồ khối PS	23
Hình 3.20	Sơ đồ khối của môi trường kiểm thử bằng mô phỏng	24
Hình 3.21	Quá trình kiểm thử bằng mô phỏng	25
Hình 3.22	Sơ đồ khối môi trường kiểm thử phần cứng	26
Hình 3.23	Thiết lập dữ liệu đầu vào trên VIO	26
Hình 3.24	Dữ liệu đọc được từ ILA	26
Hình 3.25	Phát hiện lỗi thông qua ILA	27
Hình 3.26	Waveform sau khi sửa lỗi	27
Hình 3.27	PS gửi một khung dữ liệu	28
Hình 3.28	Tìm khung dữ liệu TCP cho thông lượng tối ưu	29
Hình 3.29	Client gửi một nhóm dữ liệu	29
Hình 3.30	Kiểm thử kết nối với client	30
Hình 3.31	Kiểm thử client cùng mạng LAN	30
Hình 3.32	Kiểm thử client kết nối với Tailscale	31
Hình 3.33	Kết quả kiểm thử ban đầu	32

Hình 3.34	Kết quả kiểm thử sau khi điều chỉnh đa thức sinh	33
Hình 3.35	Kết quả kiểm thử thông lượng	33

DANH MỤC BẢNG BIỂU

Bảng 2.1	So sánh các giải pháp triển khai thiết kế	4
Bảng 2.2	Yêu cầu chức năng và phi chức năng của hệ thống	5
Bảng 3.1	Tín hiệu vào/ra khối endec_interface	12
Bảng 3.2	Tín hiệu vào/ra khối endec	14
Bảng 3.3	Tín hiệu vào/ra khối control	15
Bảng 3.4	Tín hiệu vào/ra khối slice	16
Bảng 3.5	Tín hiệu vào/ra khối conv_encoder	17
Bảng 3.6	Tín hiệu vào/ra khối branch_metric	18
Bảng 3.7	Tín hiệu vào/ra khối add_compare_select	19
Bảng 3.8	Tín hiệu vào/ra khối memory	20
Bảng 3.9	Tín hiệu vào/ra khối traceback	21
Bảng 3.10	So sánh các phương thức truyền tải dữ liệu giữa PS và PL . . .	22
Bảng 3.11	Danh sách phần mềm sử dụng	23
Bảng 3.12	Danh sách phần cứng sử dụng	23
Bảng 3.13	Khoảng giá trị của các tham số	31
Bảng 3.14	Đa thức sinh sau điều chỉnh	32
Bảng 3.15	Tài nguyên PL sử dụng	34
Bảng 3.16	Thông số hoạt động của hệ thống	35
Bảng 3.17	Thông lượng xử lý qua các giai đoạn của hệ thống	36
Bảng 3.18	So sánh MATLAB và Endec Server	37

DANH MỤC THUẬT NGỮ

STT	Thuật ngữ	Ý nghĩa Tiếng Anh	Ý nghĩa Tiếng Việt
1	AMP	Asymmetric Multi-Processing	Xử lý đa lõi không đối xứng
2	ASIC	Application-Specific Integrated Circuit	Mạch tích hợp chuyên dụng
3	AWGN	Additive White Gaussian Noise	Nhiều trắng cộng dạng Gauss
4	BUFG	Global Clock Buffer	Bộ đệm đồng hồ toàn cục
5	CPU	Central Processing Unit	Bộ xử lý trung tâm
6	DUT	Design Under Test	Thiết kế cần kiểm thử
7	Endec Server	Encoder-Decoder Server	Server mã hóa/giải mã
8	FF	Flip Flop	Mạch Flip Flop
9	FPGA	Field Programmable Gate Array	Cổng logic dạng mảng có thể tái lập trình
10	FSM	Finite State Machine	Trạng thái máy hữu hạn
11	GPIO	General-Purpose Input/Output	Chân tín hiệu đa mục đích
12	GPU	Graphics Processing Unit	Bộ xử lý đồ họa
13	HDL	Hardware Description Language	Ngôn ngữ mô tả phần cứng
14	IoT	Internet of Things	Mạng lưới vạn vật kết nối Internet
15	LAN	Local Area Network	Mạng máy tính nội bộ
16	LUT	Look-Up Table	Bảng tra cứu
17	LUTRAM	Look-Up Table Random Access Memory	Bộ nhớ đọc ghi ngẫu nhiên cấu tạo từ bảng tra cứu
18	RAM	Random Access Memory	Bộ nhớ đọc ghi ngẫu nhiên
19	SDR	Software-Defined Radio	Chuẩn vô tuyến định nghĩa bằng phần mềm
20	SNR	Signal-to-Noise Ratio	Tỷ lệ tín hiệu trên tạp âm
21	SoC	System on Chip	Hệ thống trên một vi mạch
22	UVM	Universal Verification Methodology	Phương pháp kiểm thử tổng quát

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong bối cảnh cách mạng công nghiệp 4.0, nhu cầu truyền dẫn dữ liệu tốc độ cao, ổn định và tiết kiệm năng lượng ngày càng trở nên cấp thiết. Các hệ thống viễn thông, IoT, hàng không vũ trụ hay an ninh quốc phòng đều đòi hỏi khả năng truyền tin chính xác ngay cả trong điều kiện nhiễu cao. Tuy nhiên, việc xử lý tín hiệu trên phần mềm truyền thống thường gặp hạn chế về tốc độ và độ trễ, đặc biệt với các thuật toán phức tạp như mã hóa tích chập và giải mã Viterbi.

Bài toán đặt ra là làm thế nào để tăng tốc quá trình mã hóa và giải mã dữ liệu một cách hiệu quả, đảm bảo độ tin cậy cao mà vẫn tiết kiệm tài nguyên phần cứng. Trong các giải pháp khả thi, FPGA nổi bật nhờ khả năng xử lý song song, tái cấu hình và tối ưu hiệu năng. Nếu triển khai thành công bộ mã hóa tích chập và giải mã Viterbi trên FPGA, hệ thống không chỉ đạt được tốc độ xử lý gần thời gian thực mà còn có thể ứng dụng rộng rãi trong các lĩnh vực như truyền thông không dây, hệ thống định vị, an toàn thông tin, xử lý tín hiệu số.

Việc nghiên cứu thiết kế một hệ thống phần cứng có khả năng hoạt động như một server thu/phát dữ liệu mã hóa sẽ mở ra hướng tiếp cận mới, góp phần nâng cao hiệu suất truyền thông và giảm thiểu lỗi trong các điều kiện kênh truyền không lý tưởng. Đây chính là vấn đề cấp thiết cần giải quyết, đáp ứng xu thế phát triển của công nghệ truyền thông hiện đại.

1.2 Khảo sát đề tài

Hiện nay đã có rất nhiều nghiên cứu tập trung vào tối ưu hiệu năng phần cứng của bộ mã hóa tích chập và giải mã Viterbi trên FPGA, nhưng các nghiên cứu này chưa giải quyết được bài toán triển khai dưới dạng server truy cập từ xa. Nghiên cứu [1] đạt thông lượng 80 Mbps với kiến trúc song song toàn phần cho bộ giải mã (2,1,7), sử dụng cơ chế lưu trữ dữ liệu tối ưu và RAM phân tán để giảm 10% tài nguyên logic, tiết kiệm 5% năng lượng. Tương tự như vậy, cả ba nghiên cứu [2], [3], [4] chỉ tập trung vào triển khai thuần phần cứng, không cung cấp cơ chế giao tiếp mạng để tích hợp vào hệ thống server.

Qua các khảo sát trên ta có thể xác định được ba hạn chế lớn trong các cách triển khai hiện tại: (i) các thiết kế tồn tại dưới dạng module phần cứng độc lập, thiếu giao diện mạng hoặc giao thức truyền dữ liệu để hoạt động như server; (ii) việc triển khai yêu cầu kiến thức chuyên sâu về FPGA để tái cấu hình module và công cụ thiết kế qua đó hạn chế ứng dụng thực tế; (iii) các bộ mã hóa cứng hóa

thông số, không cho phép cập nhật tốc độ mã và đa thức sinh từ xa – yêu cầu quan trọng khi triển khai server đa dịch vụ.

1.3 Định hướng nghiên cứu

Để khắc phục những hạn chế đã nêu ở mục 1.2, nghiên cứu này hướng đến phát triển bộ mã hóa tích chập và giải mã Viterbi được tăng tốc bởi FPGA và triển khai dưới dạng server với các chức năng chính sau: (i) tải lên dữ liệu cần mã hóa/giải mã từ xa; (ii) cấu hình thông số động (tốc độ mã, đa thức sinh, chiều dài ràng buộc) dễ dàng thay vì sửa đổi HDL; (iii) dùng kỹ thuật pipeline và song song hóa để tối ưu thông lượng xử lý. Từ đó nghiên cứu này sẽ kết hợp hiệu năng xử lý song song của FPGA với tính linh hoạt của mô hình server, cho phép người dùng không chuyên khai thác sức mạnh mã hóa tích chập và giải mã Viterbi thông qua kết nối từ xa. Hướng tiếp cận này mở rộng ứng dụng tiềm năng sang lĩnh vực IoT, hệ thống giám sát từ xa, và xử lý tín hiệu phân tán.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG

2.1 Cơ sở lý thuyết

2.1.1 Mã hóa tích chập

Mã hóa tích chập là một kỹ thuật mã hóa kênh quan trọng trong truyền thông số, được sử dụng rộng rãi trong các hệ thống như 4G/5G, vệ tinh, và truyền dẫn không dây. Khác với mã khối, mã tích chập xử lý dữ liệu thành một chuỗi liên tục, sử dụng các thanh ghi dịch và bộ cộng để tạo ra các bit mã hóa dựa trên cả bit hiện tại và các bit trước đó. Kỹ thuật này giúp phát hiện và sửa lỗi hiệu quả, đặc biệt trong môi trường nhiễu cao. Mã tích chập cân bằng giữa hiệu suất mã hóa và độ phức tạp tính toán, là nền tảng cho nhiều hệ thống truyền thông hiện đại [5]. Các tham số chính quyết định hiệu năng mã hóa tích chập bao gồm:

- Chiều dài ràng buộc: số bit ảnh hưởng đến đầu ra mã hóa. Khi chiều dài ràng buộc tăng, khả năng sửa lỗi được cải thiện nhưng độ phức tạp giải mã tăng theo hàm mũ do số lượng trạng thái cần xử lý tăng.
- Tốc độ mã: tỉ lệ bit thông tin/bit mã hóa. Tốc độ mã thấp cung cấp nhiều bit dư thừa giúp chống lỗi mạnh hơn trong kênh nhiễu, nhưng làm giảm thông lượng truyền dẫn. Kỹ thuật bỏ chọn bit dư có thể điều chỉnh linh hoạt tốc độ mã để cân bằng giữa độ tin cậy và hiệu suất.
- Đa thức sinh: các kết nối giữa thanh ghi dịch và bộ cộng, thường biểu diễn bằng số hệ 8. Chúng quyết định khoảng cách tự do – tham số then chốt đánh giá khả năng phát hiện/sửa lỗi.

2.1.2 Giải mã Viterbi

Thuật toán Viterbi là một phương pháp giải mã tối ưu cho mã hóa tích chập, được ứng dụng rộng rãi trong truyền thông số nhờ khả năng khôi phục dữ liệu chính xác ngay cả trong điều kiện nhiễu cao. Thuật toán hoạt động dựa trên nguyên lý quy hoạch động, duyệt qua tất cả các đường có thể trong lưới trạng thái để tìm ra chuỗi bit có khả năng cao nhất đã được phát đi. Nhờ hiệu quả và độ phức tạp vừa phải, Viterbi trở thành lựa chọn hàng đầu trong các hệ thống như 4G/5G, Wi-Fi, và thông tin vệ tinh. Thuật toán này kết hợp tính tối ưu và hiệu quả tính toán, đóng vai trò then chốt trong các hệ thống yêu cầu độ tin cậy cao [6]. Hiệu năng giải mã Viterbi phụ thuộc vào ba tham số chính:

- Chiều dài ràng buộc: quyết định số trạng thái trong sơ đồ lưới. Khi chiều dài ràng buộc tăng, khả năng sửa lỗi được cải thiện do không gian tìm kiếm lớn hơn, nhưng độ phức tạp tính toán sẽ tăng theo hàm mũ.

- Độ sâu truy ngược: xác định số bước để đưa ra quyết định giải mã cuối cùng. Độ sâu thường được chọn lớn hơn 5 lần chiều dài ràng buộc nhằm đảm bảo tỉ lệ lỗi bit dưới 10^{-5} trong kênh AWGN, nhưng tăng độ sâu truy ngược cũng dẫn đến tăng độ trễ xử lý và yêu cầu bộ nhớ lưu trữ đường đi lớn hơn.
- Độ rộng lượng tử hóa: biểu thị số bit dùng để biểu diễn khoảng cách sai khác của đường đi. Giá trị 3-5 bit là tối ưu cho giải mã mềm, cân bằng giữa độ chính xác (cải thiện 2 dB SNR so với giải mã cứng) và tài nguyên phần cứng.

2.2 Lựa chọn thiết kế

Hiện nay, việc triển khai bộ mã hóa tích chập và giải mã Viterbi có thể được thực hiện bằng nhiều giải pháp khác nhau, mỗi phương pháp mang lại những ưu/nhược điểm riêng. CPU là lựa chọn linh hoạt và dễ triển khai nhờ thư viện phần mềm phong phú, nhưng độ trễ cao và kém hiệu quả khi xử lý song song khối lượng lớn dữ liệu [7]. Trong khi đó, GPU cải thiện hiệu suất nhờ kiến trúc song song, nhưng tiêu thụ năng lượng lớn và đòi hỏi tối ưu hóa phức tạp [8]. Giải pháp ASIC cho hiệu năng tối ưu và tiết kiệm điện năng, nhưng chi phí thiết kế đắt đỏ và không linh hoạt khi thuật toán thay đổi [9]. FPGA cân bằng giữa hiệu suất và khả năng tùy biến, phù hợp cho các ứng dụng đòi hỏi độ trễ thấp, nhưng chưa tối ưu cho kiến trúc server truyền thống [2]. Cuối cùng, giải pháp SoC kết hợp CPU và FPGA khắc phục hạn chế của từng thành phần riêng lẻ bằng cách kết hợp xử lý phần mềm linh hoạt và tốc độ phần cứng, nhưng đòi hỏi thiết kế đồng bộ cả phần cứng lẫn phần mềm, dẫn đến độ phức tạp gia tăng [10].

Bảng 2.1: So sánh các giải pháp triển khai thiết kế

Giải pháp	Ưu điểm	Nhược điểm
CPU	Linh hoạt, dễ triển khai	Độ trễ cao, tốn tài nguyên khi xử lý song song
GPU	Dễ tiếp cận, có khả năng xử lý song song	Khó triển khai hơn CPU, tốn năng lượng
ASIC	Hiệu năng cao, tiết kiệm năng lượng	Chi phí thiết kế đắt, khó thay đổi thuật toán
FPGA	Cân bằng hiệu năng và linh hoạt	Chưa tối ưu cho kiến trúc server
SoC	Xử lý song song, độ trễ thấp, mở rộng được	Đòi hỏi thiết kế tối ưu phần cứng và phần mềm

Nhìn chung, tùy vào yêu cầu cụ thể về hiệu suất, độ trễ, chi phí, và khả năng mở rộng mà ta có thể lựa chọn giải pháp phù hợp. Trong bối cảnh các hệ thống server hiện đại hướng đến kiến trúc lai, SoC hoặc FPGA tích hợp sẵn trong trung tâm dữ

liệu đang nổi lên như xu hướng tối ưu cho bài toán mã hóa/giải mã quy mô lớn [11].

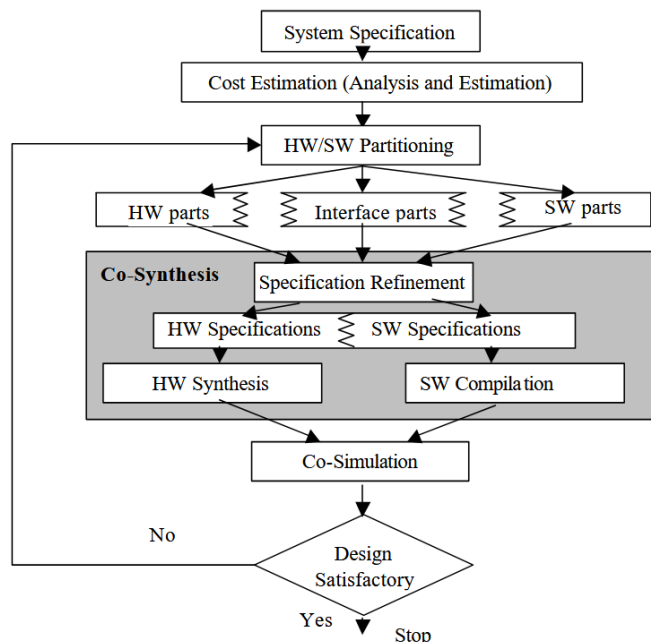
Để triển khai bộ mã hóa/giải mã dưới dạng server một cách hiệu quả, ta cần xây dựng yêu cầu chức năng và phi chức năng dựa vào những vấn đề cụ thể cần giải quyết:

Bảng 2.2: Yêu cầu chức năng và phi chức năng của hệ thống

Vấn đề giải quyết	Yêu cầu chức năng	Yêu cầu phi chức năng
Tính ứng dụng	Hỗ trợ tốc độ mã 1/2, 1/3, chiều dài ràng buộc 3-9 và tất cả các đa thức sinh	Hệ thống phải cung cấp giao diện trực quan để người dùng dễ dàng thay đổi cấu hình
Thông lượng xử lý	Thuật toán giải mã được thiết kế theo kiến trúc Radix-4 [12]	Hệ thống phải duy trì thông lượng xử lý ngay cả khi xử lý đồng thời nhiều luồng dữ liệu
Khả năng tái cấu hình	Có khả năng thay đổi cấu hình khi hệ thống đang hoạt động	Việc thay đổi cấu hình phải không ảnh hưởng đến hoạt động liên tục
Khả năng khử lỗi	Độ sâu truy ngược lớn hơn 7 lần chiều dài ràng buộc [6]	Tỷ lệ lỗi bit $< 10^{-6}$ trong kênh AWGN
Khả năng truy cập từ xa	Hệ thống hoạt động như một server	Luồng dữ liệu truyền tải phải được bảo mật
Độ tin cậy	Truyền tải dữ liệu qua giao thức TCP	Có khả năng phục hồi nếu xảy ra lỗi mạng
Độ linh hoạt	Truyền dữ liệu dưới dạng các tập tin	Các tập tin phải hoạt động độc lập với nhau

Dựa trên các yêu cầu chức năng và phi chức năng, giải pháp SoC được lựa chọn làm nền tảng để thiết kế và triển khai bộ mã hóa tích chập và giải mã Viterbi. Do tính chất phức tạp đòi hỏi sự kết hợp giữa phần cứng và phần mềm, phương pháp Đồng thiết kế cứng mềm (Hardware/Software Codesign) trở thành lựa chọn tối ưu. Quy trình của phương pháp này bắt đầu từ việc xác định thông số kỹ thuật của hệ thống dựa trên các yêu cầu thiết kế. Sau đó, quá trình ước lượng chi phí được thực hiện để phân tích và tối ưu hóa tài nguyên trước khi tiến hành phân chia chức năng giữa phần cứng và phần mềm. Phần cứng tập trung vào các khối xử lý chuyên dụng nhằm đạt hiệu suất cao, trong khi phần mềm đảm nhận các tác vụ linh hoạt và điều khiển. Giai đoạn tiếp theo là đồng tổng hợp, nơi các đặc tả phần cứng và phần mềm được phát triển song song, bao gồm tổng hợp phần cứng và biên dịch phần mềm. Kết quả được kiểm chứng thông qua mô phỏng đồng bộ để đảm bảo tính chính xác và hiệu quả của hệ thống. Nếu kết quả chưa đạt yêu cầu, quy trình quay lại bước

tính chỉnh đặc tả để điều chỉnh thiết kế, tạo thành một vòng lặp khép kín cho đến khi đạt được kết quả tối ưu. Cách tiếp cận này đặc biệt phù hợp với các hệ thống phức tạp như bộ mã hóa tích chập và giải mã Viterbi, nơi đòi hỏi sự cân bằng giữa tốc độ xử lý phần cứng và tính mềm dẻo của phần mềm. Quy trình này được minh họa rõ ràng trong Hình 2.1 [13].



Hình 2.1: Quy trình trong phương pháp Đồng thiết kế cứng mềm

2.3 Công nghệ sử dụng

2.3.1 SoC

Hệ thống trên chip (System-on-Chip - SoC) là một giải pháp tích hợp toàn bộ hệ thống máy tính vào một vi mạch duy nhất, bao gồm bộ xử lý, bộ nhớ, các khối xử lý tín hiệu số, và các giao diện ngoại vi. Trong đó, ZYNQ của Xilinx là một dòng SoC tiên tiến kết hợp lõi xử lý đa nhân với khả năng lập trình FPGA trên cùng một chip, mang lại sự linh hoạt và hiệu suất vượt trội. Kiến trúc ZYNQ chia thành hai phần chính: (i) phần xử lý (Processing System - PS) sử dụng lõi ARM Cortex, đảm nhiệm các tác vụ điều khiển phức tạp và giao tiếp với phần cứng bên ngoài; (ii) phần logic lập trình (Programmable Logic - PL) dựa trên công nghệ FPGA, cho phép thiết kế các khối xử lý tùy chỉnh để tăng tốc các thuật toán đòi hỏi tính toán song song như xử lý tín hiệu số, AI, hay mã hóa [14].

Liên kết giữa PS và PL trong ZYNQ được thực hiện thông qua các giao thức tốc độ cao như AXI, cho phép trao đổi dữ liệu hiệu quả giữa phần mềm chạy trên ARM và phần cứng tùy chỉnh trên FPGA. Nhờ đó, ZYNQ tối ưu hóa hiệu năng hệ thống bằng cách kết hợp sức mạnh xử lý tuần tự của CPU với khả năng xử lý song

song và thời gian thực của FPGA [15].

2.3.2 AXI

Giao thức AXI (Advanced eXtensible Interface) là một chuẩn giao tiếp quan trọng do ARM phát triển, được sử dụng rộng rãi trong các hệ thống SoC như Xilinx ZYNQ để kết nối hiệu quả giữa các khối xử lý (CPU, GPU), bộ nhớ và các module phần cứng tùy chỉnh trên FPGA. Trong đó, AXI-Stream là một biến thể không địa chỉ hóa của AXI, chuyên dụng cho các ứng dụng truyền dữ liệu tốc độ cao theo luồng liên tục, phù hợp với các tác vụ xử lý tín hiệu số hoặc truyền video. AXI cung cấp cơ chế truyền dữ liệu độc lập địa chỉ và dữ liệu, hỗ trợ burst transfers, và đảm bảo tính nhất quán trong hệ thống đa nhân [16].

2.3.3 DMA

DMA (Direct Memory Access) là cơ chế phần cứng cho phép các thiết bị ngoại vi truyền dữ liệu trực tiếp tới bộ nhớ hệ thống mà không cần sự can thiệp của CPU, giúp giảm tải xử lý và tăng hiệu suất hệ thống. Trong kiến trúc SoC như Xilinx ZYNQ, AXI DMA (AXI Direct Memory Access) là một module quan trọng, tận dụng giao thức AXI và AXI-Stream để kết nối hiệu quả giữa PS và PL. Với khả năng tối ưu hóa truyền dữ liệu giữa phần cứng và phần mềm, AXI DMA là thành phần không thể thiếu trong các thiết kế ZYNQ yêu cầu hiệu năng cao, từ IoT đến hệ thống nhúng phức tạp [17].

2.3.4 BRAM

BRAM (Block RAM) là một dạng bộ nhớ được tích hợp sẵn trong kiến trúc PL của dòng Xilinx ZYNQ, đóng vai trò quan trọng trong việc lưu trữ và truy xuất dữ liệu tốc độ cao. Khác với bộ nhớ RAM cần truy cập qua bus hệ thống, BRAM nằm trực tiếp trên PL của ZYNQ, giúp giảm độ trễ và tăng hiệu suất cho các ứng dụng đòi hỏi thời gian phản hồi nhanh [18].

2.3.5 ILA

Integrated Logic Analyzer (ILA) là một công cụ debug mạnh mẽ được tích hợp trong Xilinx Vivado Design Suite, cho phép người dùng quan sát và phân tích tín hiệu bên trong thiết kế PL một cách trực tiếp. ILA hoạt động như một bộ phân tích logic cứng, sử dụng các tài nguyên logic và bộ nhớ khả dụng trên PL để thu thập và lưu trữ dữ liệu tín hiệu theo thời gian thực. Đặc biệt, ILA có mối quan hệ chặt chẽ với BRAM, BRAM thường được sử dụng làm bộ đệm lưu trữ dữ liệu tín hiệu thu thập bởi ILA, với khả năng lưu trữ lên đến hàng nghìn mẫu dữ liệu. Độ sâu cửa sổ debug của ILA phụ thuộc trực tiếp vào số lượng BRAM được cấp phát, trong khi tốc độ lấy mẫu được xác định bởi clock domain của tín hiệu cần debug. Khi thiết kế các hệ thống xử lý tín hiệu số phức tạp, việc tối ưu hóa việc sử dụng BRAM

cho cả chức năng hệ thống lẫn nhu cầu debug bằng ILA trở thành một yếu tố quan trọng trong quá trình phát triển [19].

2.3.6 TCP

TCP (Transmission Control Protocol) là một trong những giao thức cốt lõi của bộ giao thức TCP/IP, đóng vai trò quan trọng trong việc truyền dữ liệu đáng tin cậy và theo thứ tự giữa các thiết bị mạng. TCP cung cấp cơ chế kiểm soát lỗi, điều khiển luồng và đảm bảo dữ liệu được giao nhận chính xác giữa máy nguồn và máy đích. TCP là giao thức không thể thay thế cho các ứng dụng yêu cầu độ chính xác tuyệt đối [20].

2.3.7 lwIP

lwIP (lightweight IP) là một thư viện mã nguồn mở, được thiết kế tối ưu cho các hệ thống nhúng có tài nguyên hạn chế (như vi điều khiển, FPGA, hoặc SoC). lwIP server là một triển khai của giao thức mạng dựa trên thư viện này, cho phép các thiết bị nhúng hoạt động như một máy chủ web, máy chủ TCP/UDP với bộ nhớ RAM chỉ từ vài chục KB và CPU tốc độ thấp [21].

2.3.8 Mesh Network

Việc mở cổng mạng trên tường lửa router từ lâu đã là giải pháp phổ biến để truy cập thiết bị trong mạng nội bộ (LAN) từ bên ngoài Internet, nhưng nó đi kèm nhiều hạn chế: (i) quá trình cấu hình phức tạp, (ii) rủi ro bảo mật do mở cổng tường lửa, (iii) cần sự hỗ trợ từ nhà cung cấp mạng. Để khắc phục những vấn đề này, Mesh Network ra đời như một giải pháp hiện đại, cho phép các thiết bị kết nối trực tiếp với nhau qua Internet một cách an toàn và linh hoạt. [22].

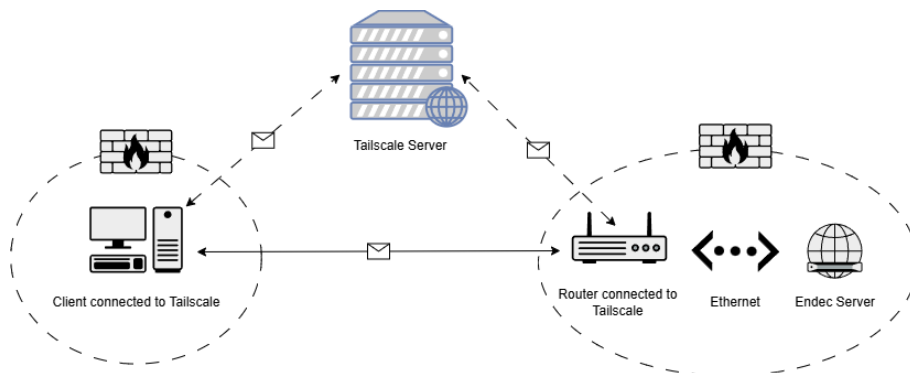
Mesh Network là kiến trúc mạng phi tập trung, trong đó mỗi node (thiết bị) có thể giao tiếp trực tiếp với các node khác thông qua nhiều đường dẫn, tự động định tuyến lại nếu một liên kết bị lỗi. Một ví dụ điển hình của dịch vụ cung cấp Mesh Network là Tailscale, dịch vụ này xây dựng Mesh Network dựa trên giao thức WireGuard, giúp kết nối các thiết bị qua Internet như thể chúng cùng một mạng LAN mà không cần Port Forwarding hoặc cấu hình tường lửa phức tạp [23].

CHƯƠNG 3. THIẾT KẾ, TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG

3.1 Thiết kế kiến trúc

3.1.1 Mô hình hệ thống

Như đã giới thiệu và giải thích ở Chương 2, hệ thống server sẽ được triển khai trên kiến trúc SoC với khả năng truy cập từ xa, trên cơ sở này người viết mặc định client và server sẽ hoạt động ở hai mạng LAN khác nhau. Thông thường việc giao tiếp giữa hai mạng LAN sẽ phụ thuộc vào việc mở cổng mạng trên tường lửa router. Tuy nhiên, việc cấu hình này sẽ cần sự can thiệp của nhà cung cấp mạng và điều này sẽ tạo thành sự bất tiện trong trải nghiệm người dùng. Chính vì vậy, một giải pháp tối ưu và linh hoạt hơn đó chính là client và server sẽ kết nối với một server trung gian trước để khởi tạo và tạo thành Mesh Network. Sau khi kết nối vào Mesh Network, các thiết bị có thể tiến hành gửi và nhận dữ liệu như khi cùng một mạng LAN và không bị rào cản bởi tường lửa, Hình 3.1 minh họa về mô hình hệ thống khi sử dụng dịch vụ Tailscale làm kết nối trung gian giữa client và server.



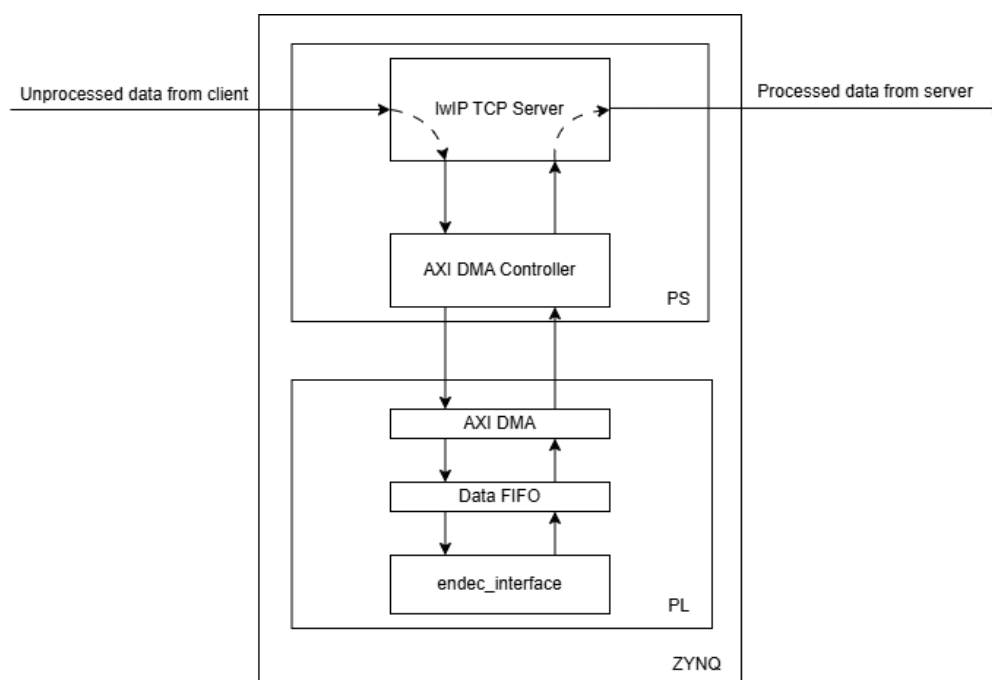
Hình 3.1: Mô hình hệ thống

Trên các thiết bị sử dụng hệ điều hành phổ biến thì có thể kết nối với server của Tailscale một cách trực tiếp, nhưng do Endec Server hướng tới việc tối ưu về mặt hiệu năng và độ trễ nên sẽ được thiết kế chạy trên baremetal. Việc chạy trên baremetal tuy cung cấp lợi thế rất lớn về hiệu năng và độ trễ nhưng cũng đồng thời gây ra trở ngại trong việc cấu hình kết nối với server Tailscale. Một giải pháp cho việc này đó chính là dùng một router làm cầu nối điều hướng luồng dữ liệu từ Endec Server đến server trung gian của Tailscale.

3.1.2 Thiết kế tổng quan

Việc sử dụng PL để tăng tốc khả năng mã hóa/giải mã và PS để điều khiển giao tiếp bên ngoài đòi hỏi sự chặt chẽ và tối ưu trong thiết kế giao tiếp giữa PS và PL bên cạnh việc tối ưu tác vụ xử lý tín hiệu trong nội bộ hai khối này. Hình 3.2 minh

họa cho luồng di chuyển của dữ liệu sau khi Endec Server nhận thành công dữ liệu từ client thông qua mạng lưới kết nối Internet đã được đề cập ở mục 3.1.1.



Hình 3.2: Luồng dữ liệu trong Endec Server

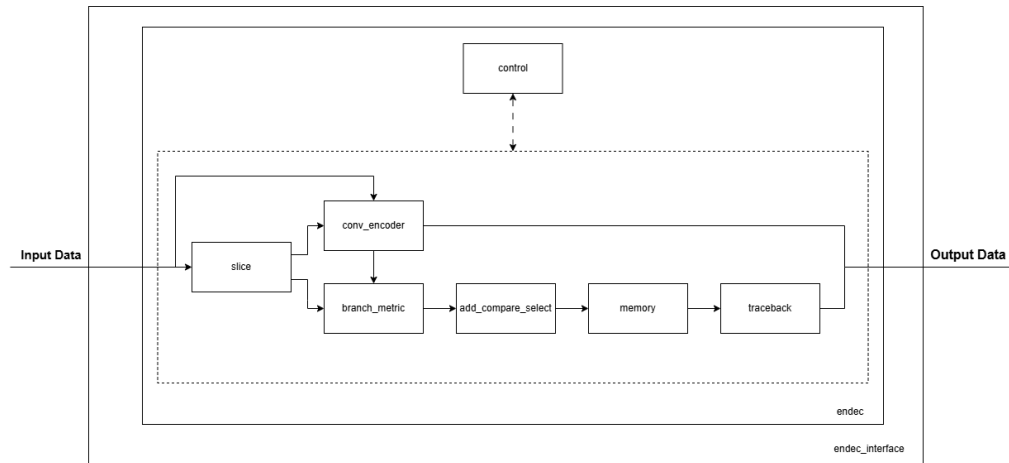
Các phần tiếp theo sẽ giải thích chi tiết về thiết kế và chức năng của từng khối chức năng trong đường đi của luồng dữ liệu.

3.2 Thiết kế PL

Thiết kế phần cứng cho bộ giải mã Viterbi đã trải qua nhiều thập kỷ phát triển, từ những kiến trúc đầu tiên dựa trên Radix-2 với các khối cơ bản như BMU (Branch Metric Unit), ACS (Add-Compare-Select) và bộ nhớ truy ngược (Traceback Memory) [24][25], đến các cải tiến hiệu suất bằng cách tăng bậc xử lý (radix). Trong môi trường FPGA, nơi tài nguyên phần cứng có thể được tùy chỉnh linh hoạt, việc chuyển từ Radix-2 sang Radix-4 trở thành xu hướng tối ưu nhờ khả năng xử lý 2 bit/chu kỳ, giảm độ trễ và tăng gấp đôi thông lượng so với Radix-2 truyền thống. Các nghiên cứu từ thập niên 90 [12] đã chứng minh Radix-4 phù hợp hơn với kiến trúc song song của FPGA, đồng thời tiết kiệm tài nguyên bộ nhớ và năng lượng nhờ giảm số chu kỳ tính toán.

Lợi thế lớn nhất của FPGA chính là khả năng xử lý đa luồng và thông lượng dữ liệu cao nhờ kiến trúc song song mềm dẻo. Trong nghiên cứu này, thiết kế PL tận dụng triệt để ưu thế trên thông qua kiến trúc giải mã Viterbi Radix-4, nơi mỗi chu kỳ xử lý đồng thời 2 bit dữ liệu. Để đảm bảo hiệu suất tổng thể, thiết kế còn tích hợp cơ chế pipeline tối ưu giữa các tầng xử lý và giao tiếp hiệu quả với PS thông qua AXI DMA cùng cơ chế điều phối luồng dữ liệu thông minh, tránh nghẽn cổ

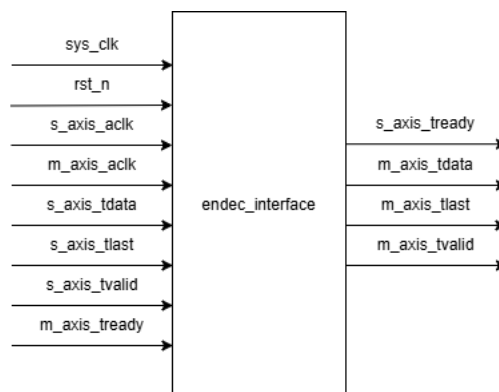
chai khi trao đổi dữ liệu tốc độ cao.



Hình 3.3: Luồng dữ liệu trong khối endec_interface

Hình 3.3 minh họa chi tiết luồng dữ liệu trong khối PL được thiết kế một cách có chủ đích và tỉ mỉ với đặc tính tuyến tính thể hiện rõ rệt. Đặc điểm thiết kế nổi bật này không chỉ mang lại hai lợi ích quan trọng hàng đầu mà còn đảm bảo hiệu suất ổn định: (i) thời gian xử lý trở nên hoàn toàn có thể dự đoán được (deterministic), từ đó loại bỏ hoàn toàn nguy cơ xảy ra hiện tượng treo hệ thống ngoài ý muốn trong điều kiện vận hành thực tế - đây chính là một yếu tố then chốt và không thể thiếu đối với các hệ thống thời gian thực yêu cầu độ tin cậy cao; (ii) kiến trúc pipeline được áp dụng nhất quán và xuyên suốt giữa các khối chức năng khác nhau, nhờ đó giúp tối đa hóa thông lượng xử lý của PL, đồng thời phát huy tối đa hiệu năng phần cứng sẵn có.

3.2.1 Thiết kế khối endec_interface



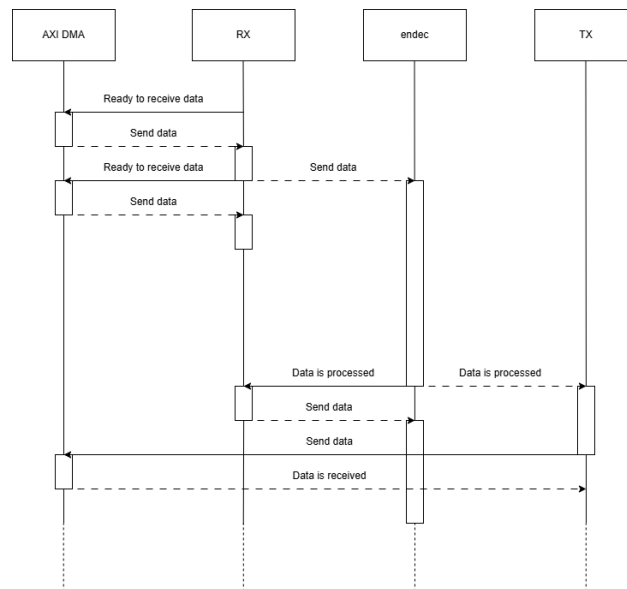
Hình 3.4: Sơ đồ khối endec_interface

Bảng 3.1: Tín hiệu vào/ra khối endec_interface

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
sys_clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst_n	1	1	I	thiết lập lại trạng thái ban đầu
s_axis_aclk	1	1	I	tín hiệu đồng bộ theo sườn dương, dùng cho s_axis
m_axis_aclk	1	1	I	tín hiệu đồng bộ theo sườn dương, dùng cho m_axis
s_axis_tdata	64	1	I	dữ liệu dùng cho s_axis
s_axis_tlast	1	1	I	báo hiệu nhịp dữ liệu cuối cùng của s_axis
s_axis_tvalid	1	1	I	báo hiệu dữ liệu hợp lệ cho s_axis
m_axis_tready	1	1	I	báo hiệu sẵn sàng nhận dữ liệu cho m_axis
s_axis_tready	1	1	O	báo hiệu sẵn sàng nhận dữ liệu cho s_axis
m_axis_tdata	64	1	O	dữ liệu dùng cho m_axis
m_axis_tlast	1	1	O	báo hiệu nhịp dữ liệu cuối cùng của m_axis
m_axis_tvalid	1	1	O	báo hiệu dữ liệu hợp lệ cho m_axis

Khối endec_interface đóng vai trò quan trọng như một giao diện trung gian kết nối giữa khối AXI DMA và khối endec, có nhiệm vụ chính là tiếp nhận dữ liệu được truyền theo giao thức AXI từ AXI DMA, sau đó thực hiện chuyển đổi sang định dạng phù hợp để khối endec có thể xử lý một cách hiệu quả.

Về mặt cấu trúc, khối này được chia thành ba module chức năng chính bao gồm RX, endec và TX, trong đó mỗi module đều được thiết kế thành các tiến trình hoạt động độc lập nhằm đáp ứng tính chất hoạt động tách biệt và chuyên biệt của từng thành phần. Cơ chế hoạt động chi tiết của các tiến trình này, đặc biệt khi khối endec_interface bắt đầu thiết lập quá trình giao tiếp với AXI DMA, được mô tả và thể hiện một cách rõ ràng, cụ thể trong Hình 3.5.

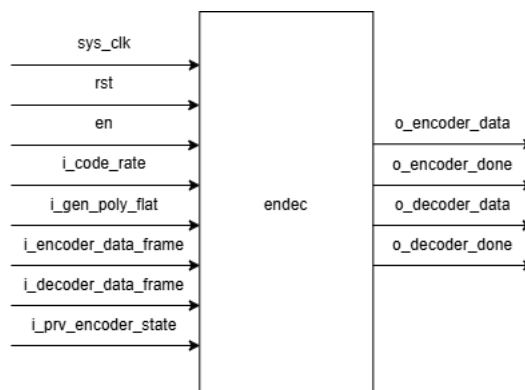


Hình 3.5: Sơ đồ tuần tự khối endec_interface

Hình 3.5 minh họa rõ ưu điểm của kiến trúc đa luồng độc lập trong thiết kế: ngay khi khối endec bắt đầu xử lý dữ liệu hiện tại, khối RX đã có thể đồng thời tiếp nhận gói dữ liệu mới từ AXI DMA. Nhờ cơ chế hoạt động song song này, kể từ chu kỳ xử lý thứ hai trở đi, endec luôn có sẵn dữ liệu trong bộ đệm RX để xử lý ngay lập tức mà không phải chờ đợi thao tác truyền dữ liệu từ DMA, qua đó tối ưu hóa thông lượng hệ thống.

Để đảm bảo độ tin cậy, sau mỗi chu kỳ nhận dữ liệu từ RX và truyền dữ liệu đến TX, khối endec sẽ tự động thiết lập lại trạng thái ban đầu theo thiết kế. Cơ chế reset tuần hoàn này giúp hệ thống tránh được các lỗi tích lũy và duy trì tính ổn định trong quá trình vận hành liên tục.

3.2.2 Thiết kế khối endec



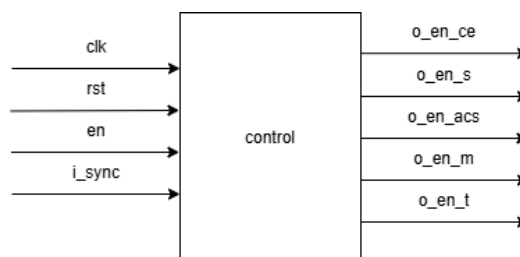
Hình 3.6: Sơ đồ khối endec

Bảng 3.2: Tín hiệu vào/ra khối endec

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
sys_clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst_n	1	1	I	thiết lập lại trạng thái ban đầu
en	1	1	I	tín hiệu cho phép hoạt động
i_code_rate	1	1	I	cấu hình tốc độ mã
i_gen_poly_flat	27	1	I	cấu hình đa thức sinh
i_encoder_data_frame	192	1	I	khung dữ liệu cần mã hóa
i_decoder_data_frame	384	1	I	khung dữ liệu cần giải mã
i_prv_encoder_state	8	1	I	trạng thái cuối của thanh ghi dịch trong chu kỳ trước
o_encoder_data	576	1	O	dữ liệu đã được mã hóa
o_encoder_done	1	1	O	báo hiệu mã hóa hoàn thành
o_decoder_data	128	1	O	dữ liệu đã được giải mã
o_decoder_done	1	1	O	báo hiệu giải mã hoàn thành

Khối endec đóng vai trò như là khối bao quát có vai trò liên kết tín hiệu giữa các khối con phía dưới và kết nối tín hiệu đầu vào đầu ra. Số lượng bit cho mỗi tín hiệu kết nối đã được người thiết kế tính toán sao cho tối ưu nhất với 64 bit dữ liệu trong giao thức truyền dữ liệu giữa endec_interface và AXI DMA. Việc tổng số bit tín hiệu đầu vào/ra là bội số của 64 sẽ giúp toàn bộ chỗ chứa dữ liệu trong mỗi nhịp thời gian được tận dụng tối đa.

3.2.3 Thiết kế khối control

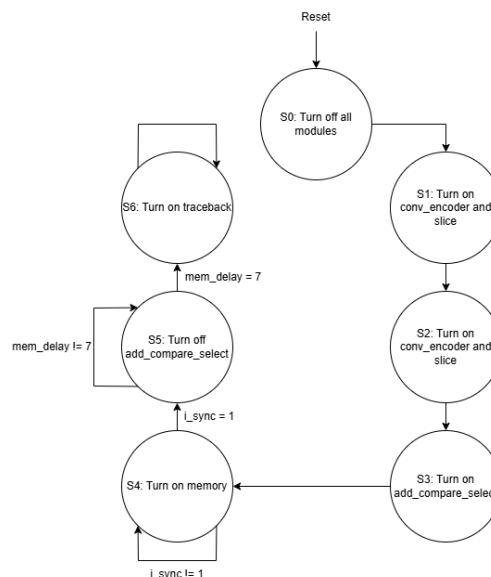


Hình 3.7: Sơ đồ khối control

Bảng 3.3: Tín hiệu vào/ra khối control

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en	1	1	I	tín hiệu cho phép hoạt động
i_sync	1	1	I	tín hiệu đồng bộ trạng thái
o_en_ce	1	1	O	tín hiệu hoạt động cho khối conv_encoder
o_en_s	1	1	O	tín hiệu hoạt động cho khối slice
o_en_acs	1	1	O	tín hiệu hoạt động cho khối add_compare_select
o_en_m	1	1	O	tín hiệu hoạt động cho khối memory
o_en_t	1	1	O	tín hiệu hoạt động cho khối traceback

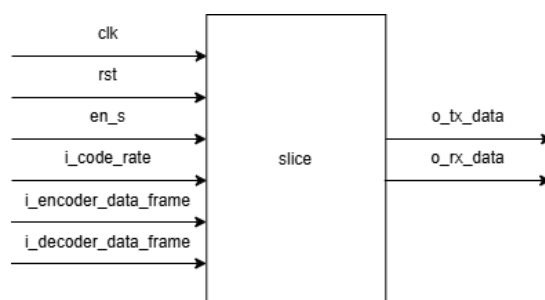
Khối control đóng vai trò trung tâm trong việc điều phối hoạt động của toàn hệ thống. Với đặc điểm phần lớn các khối chức năng đều sử dụng phần tử nhớ, việc điều khiển chính xác thời điểm kích hoạt/khoá các tín hiệu hoạt động là vô cùng quan trọng để đảm bảo quá trình nhận/gửi dữ liệu được đồng bộ hoá hoàn toàn. Để đạt được hiệu quả tối ưu, giải pháp triển khai thông qua FSM được lựa chọn. Hình 3.8 mô tả chi tiết các trạng thái hoạt động cũng như cơ chế chuyển đổi trạng thái của hệ thống, giúp đảm bảo tính ổn định và hiệu suất cao trong mọi tình huống vận hành.



Hình 3.8: Trạng thái máy khối control

Trong hệ thống PL, hoạt động chuyển mạch (switching activity) của các cổng logic giữa trạng thái 0 và 1 là nguồn tiêu thụ năng lượng chủ yếu. Do đó, cơ chế tắt động (clock gating/power gating) các khối chức năng khi không hoạt động không chỉ giảm đáng kể công suất tiêu thụ của PL mà còn nâng cao hiệu quả năng lượng cho toàn bộ hệ thống Endec Server.

3.2.4 Thiết kế khối slice



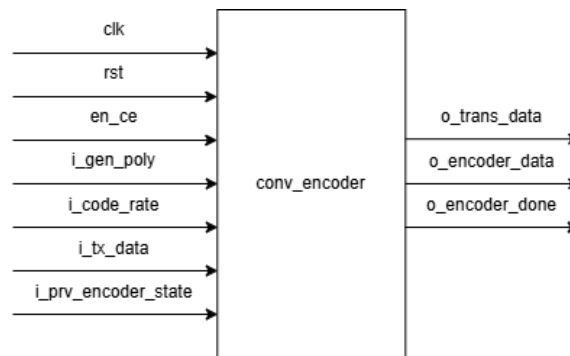
Hình 3.9: Sơ đồ khối slice

Bảng 3.4: Tín hiệu vào/ra khối slice

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en_s	1	1	I	tín hiệu cho phép hoạt động
i_code_rate	1	1	I	báo hiệu thay đổi trạng thái
i_encoder_data_frame	192	1	I	khung dữ liệu cần mã hóa
i_decoder_data_frame	384	1	I	khung dữ liệu cần giải mã
o_tx_data	1	1	O	nhóm bit cần mã hóa trong một bước thời gian
o_rx_data	6	1	O	nhóm bit cần giải mã trong một bước thời gian

Khối slice đảm nhận chức năng phân mảnh dữ liệu đầu vào, thực hiện chia tách các khung dữ liệu cần mã hóa/giải mã thành các nhóm bit con theo từng chu kỳ xử lý. Cơ chế phân chia này được thiết kế đặc biệt để đảm bảo sự tương thích tối ưu với các thuật toán xử lý tín hiệu ở các khối phía sau, đồng thời duy trì tính toàn vẹn của dòng dữ liệu trong suốt quá trình xử lý.

3.2.5 Thiết kế khối conv_encoder

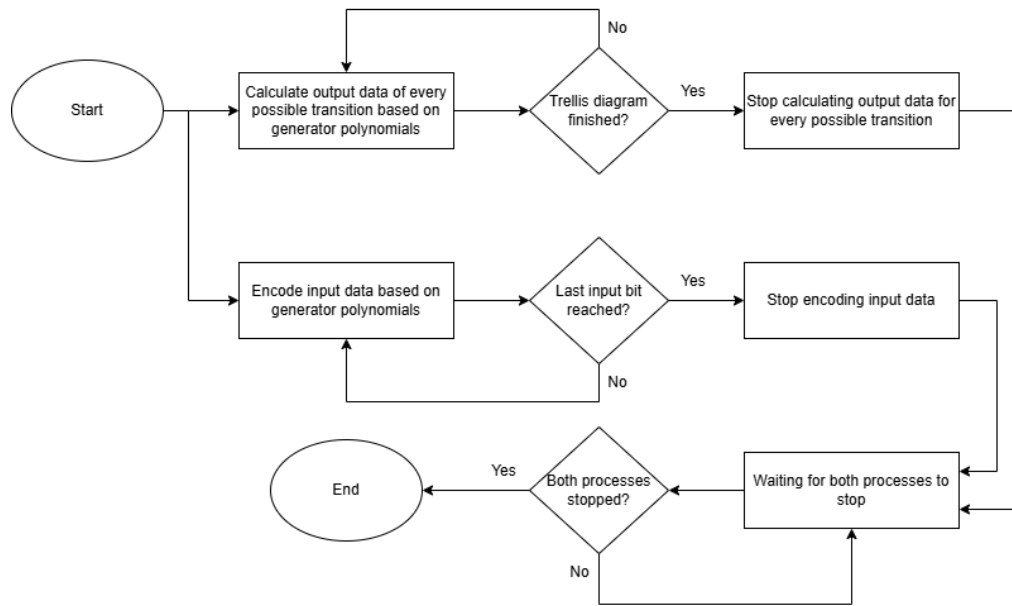


Hình 3.10: Sơ đồ khối conv_encoder

Bảng 3.5: Tín hiệu vào/ra khối conv_encoder

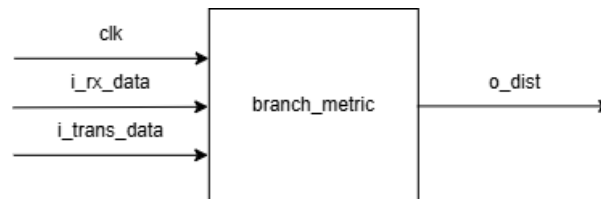
Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en_ce	1	1	I	tín hiệu cho phép hoạt động
i_gen_poly	9	3	I	cấu hình đa thức sinh
i_code_rate	1	1	I	cấu hình tốc độ mã
i_tx_data	1	1	I	dữ liệu cần mã hóa trong một bước thời gian
i_prv_encoder_state	8	1	I	trạng thái cuối của thanh ghi dịch trong chu kỳ trước
o_trans_data	6	256x4	O	đầu ra tương ứng với một dịch chuyển
o_encoder_data	576	1	O	dữ liệu đã được mã hóa
o_encoder_done	1	1	O	báo hiệu mã hóa hoàn thành

Khối conv_encoder có nhiệm vụ chính là tìm đầu ra cho mỗi dịch chuyển giữa các trạng thái đối với cấu hình đa thức sinh tương ứng. Tuy nhiên, thuật toán tìm đầu ra này cũng chính là thuật toán mã hóa tích chập, chính vì vậy, ta có thể dễ dàng ứng dụng thêm một luồng mã hóa dữ liệu song song với luồng giải mã dữ liệu trong thiết kế. Việc xử lý song song này được mô hình hóa ở Hình 3.3 và Hình 3.11. Giải thích chi tiết cho việc áp dụng này sẽ nằm ở mục 4.1.2 trong Chương 4.



Hình 3.11: Lưu đồ khối conv_encoder

3.2.6 Thiết kế khối branch_metric



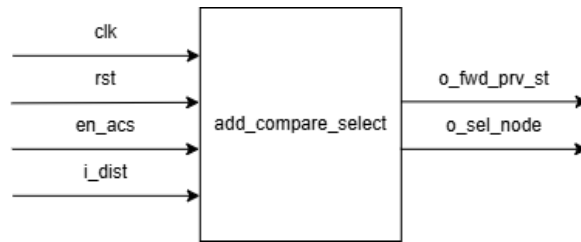
Hình 3.12: Sơ đồ khối branch_metric

Bảng 3.6: Tín hiệu vào/ra khối branch_metric

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
i_rx_data	6	1	I	đầu ra cần giải mã
i_trans_data	6	256x4	I	đầu ra của một dịch chuyển
o_dist	3	256x4	O	khoảng cách Hamming của một dịch chuyển

Khối branch_metric có nhiệm vụ so sánh đầu ra cần giải mã và đầu ra của dịch chuyển giữa các trạng thái để từ đó tính được khoảng cách Hamming.

3.2.7 Thiết kế khối add_compare_select



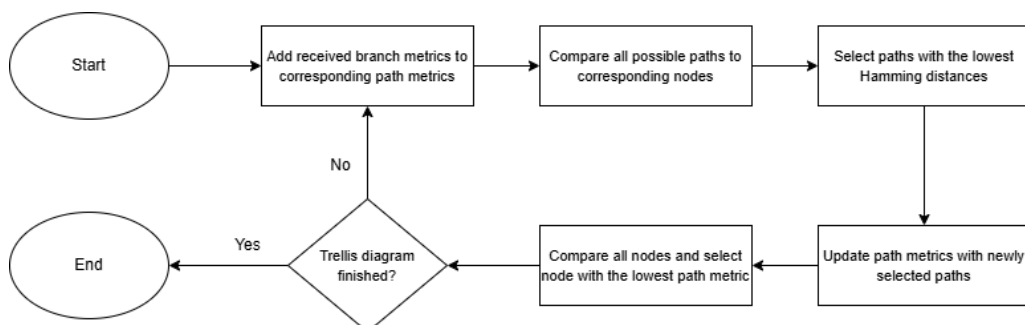
Hình 3.13: Sơ đồ khối add_compare_select

Bảng 3.7: Tín hiệu vào/ra khối add_compare_select

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en_acs	1	1	I	tín hiệu cho phép hoạt động
i_dist	3	256x4	I	khoảng cách Hamming của một dịch chuyển
o_fwd_prv_st	8	256	O	nút được chọn trong bốn dịch chuyển khả thi đến một nút
o_sel_node	8	1	O	nút có khoảng cách Hamming nhỏ nhất

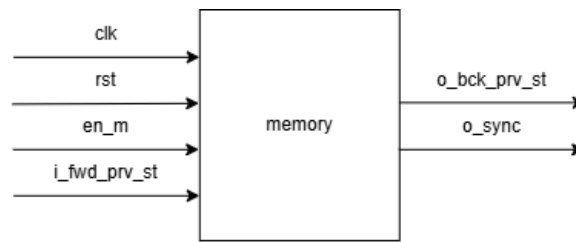
Khối add_compare_select có nhiệm vụ tìm ra dịch chuyển và nút có khoảng cách Hamming nhỏ nhất để từ đó xây dựng nên sơ đồ lưới.

Đây là khối phức tạp nhất trong thiết kế PL, tuy có thuật toán khá đơn giản như minh họa ở Hình 3.14 nhưng với 1024 dịch chuyển và 256 nút cần xử lý và tính toán song song trong một bước thời gian thì việc tối ưu để triển khai thành công trên PL là một thách thức lớn. Vấn đề và cách giải quyết trong việc tối ưu và triển khai khối chức năng này sẽ được bàn chi tiết ở mục 4.2.4 trong Chương 4.



Hình 3.14: Lưu đồ khối add_compare_select

3.2.8 Thiết kế khối memory

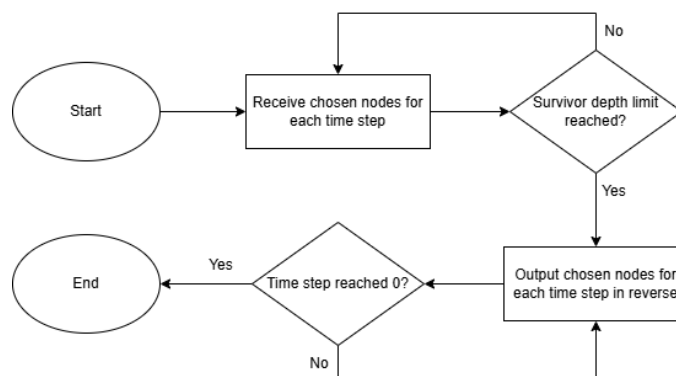


Hình 3.15: Sơ đồ khối memory

Bảng 3.8: Tín hiệu vào/ra khối memory

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en_m	1	1	I	tín hiệu cho phép hoạt động
i_fwd_prv_st	8	256	I	nút được chọn trong bốn dịch chuyển khả thi đến một nút chiều thuận
o_bck_prv_st	8	256	O	nút được chọn trong bốn dịch chuyển khả thi đến một nút chiều nghịch
o_sync	1	1	O	tín hiệu đồng bộ trạng thái

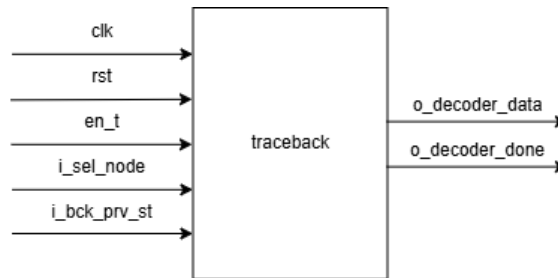
Khối memory có nhiệm vụ lưu lại giá trị của các nút theo từng bước thời gian để xây dựng nên sơ đồ lưới. Sau khi lưu lại đủ số bước thời gian, khối memory sẽ gửi dữ liệu ở đầu ra để tiến hành truy ngược nhằm tìm đường đi có khoảng cách Hamming bé nhất cũng chính là đầu ra cần giải mã. Thuật toán này được minh họa chi tiết ở Hình 3.16.



Hình 3.16: Lưu đồ khối memory

Mặc dù thuật toán của khối này có nguyên lý hoạt động đơn giản, thách thức chính nằm ở bài toán thiết kế kiến trúc bộ nhớ. Với yêu cầu lưu trữ đồng thời dữ liệu của 256 nút cho mỗi bước thời gian và duy trì liên tục 64 bước thời gian, phương án triển khai thông thường sử dụng LUT và FF trên PL trở nên bất khả thi do giới hạn tài nguyên phần cứng [26]. Giải pháp tối ưu hóa kiến trúc bộ nhớ nhằm cân bằng giữa hiệu năng xử lý và mức độ sử dụng tài nguyên sẽ được phân tích sâu tại mục 4.2.5 trong Chương 4.

3.2.9 Thiết kế khối traceback



Hình 3.17: Sơ đồ khối traceback

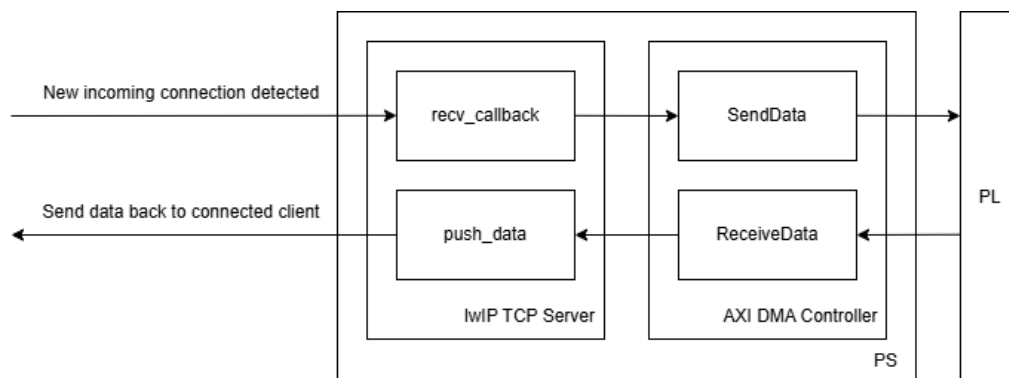
Bảng 3.9: Tín hiệu vào/ra khối traceback

Tên tín hiệu	Số bit	Mảng	I/O	Chức năng
clk	1	1	I	tín hiệu đồng bộ theo sườn dương
rst	1	1	I	thiết lập lại trạng thái ban đầu
en_t	1	1	I	tín hiệu cho phép hoạt động
i_sel_node	8	1	I	nút có khoảng cách Hamming nhỏ nhất
i_bck_prv_st	8	256	I	nút được chọn trong bốn dịch chuyển khả thi đến một nút chiều nghịch
o_decoder_data	128	1	O	dữ liệu đã được giải mã
o_decoder_done	1	1	O	báo hiệu giải mã hoàn thành

Từ dữ liệu của nút có khoảng cách Hamming nhỏ nhất từ khối add_compare_select và sơ đồ lưới lưu trữ dịch chuyển các nút từ khối memory, khối traceback sẽ truy ngược theo bước thời gian để tìm đầu vào cho dịch chuyển các nút để từ đó tìm được đầu ra cần giải mã.

3.3 Thiết kế PS

Khối PS có nhiệm vụ điều phối luồng dữ liệu giữa PL và client cần sử dụng dịch vụ mã hóa/giải mã dữ liệu. Luồng dữ liệu này được minh họa ở Hình 3.18.



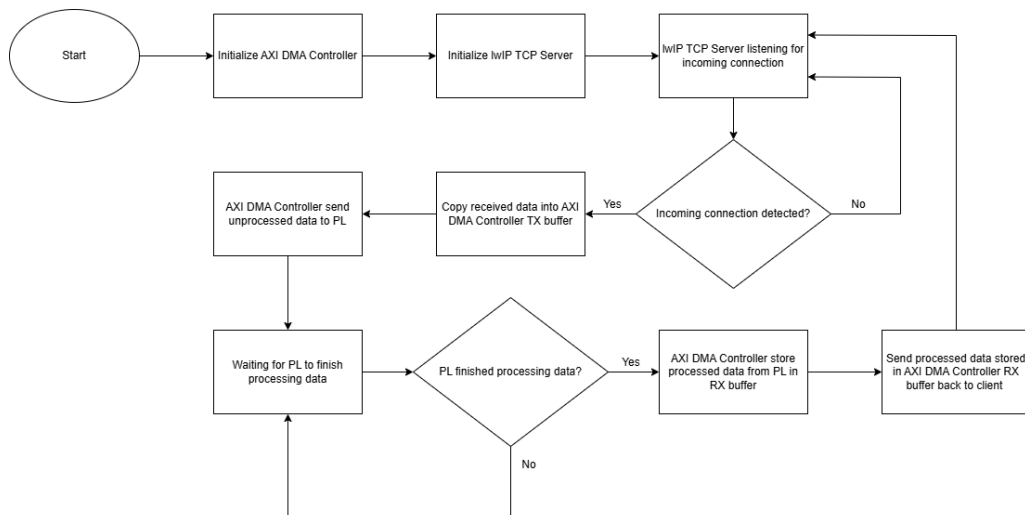
Hình 3.18: Luồng dữ liệu trong khối PS

Dữ liệu thô nhận từ client trải qua quá trình tái định dạng để tuân thủ nghiêm ngặt chuẩn giao tiếp AXI trước khi truyền tới PL. Việc áp dụng chuẩn giao tiếp được tích hợp sẵn trong kiến trúc SoC mang lại ba lợi ích nổi bật: (i) tối ưu hóa thông lượng truyền dữ liệu và giảm thiểu độ trễ; (ii) đảm bảo tuyệt đối tính toàn vẹn dữ liệu thông qua cơ chế kiểm tra lỗi phần cứng; (iii) bảo vệ tín hiệu khỏi nhiễu điện từ nhờ các đường truyền được thiết kế tối ưu bởi nhà sản xuất - yếu tố then chốt trong các hệ thống giao tiếp tốc độ cao.

Bảng 3.10: So sánh các phương thức truyền tải dữ liệu giữa PS và PL

Phương thức	Thông lượng	Khối lượng dữ liệu	Tài nguyên sử dụng
AXI GPIO	10-100 MB/s	Thấp, PS phải kết nối trực tiếp với từng chân tín hiệu	Cao, PS phải điều khiển trực tiếp mọi khung truyền dữ liệu
AXI BRAM	300-600 MB/s	Thấp, bị giới hạn bởi cấu trúc của BRAM	Cao, PS phải liên tục lấy mẫu và cập nhật dữ liệu cho BRAM
AXI DMA	1.2 GB/s	Cao, có thể lên tới 1024 bit trong một xung đồng hồ	Thấp, PS chỉ phải khởi tạo và cấu hình DMA

Có rất nhiều cách để truyền dữ liệu giữa PS và PL như thể hiện ở Bảng 3.10 nhưng AXI DMA nổi bật lên nhờ sự tối ưu trong thông lượng và độ trễ khi truyền tải dữ liệu với khối lượng lớn. Bên cạnh đó, việc sử dụng AXI DMA còn giúp giải phóng tài nguyên PS khỏi việc di chuyển các khối dữ liệu để dành tài nguyên cho việc chạy lwIP TCP server. Đối với tài nguyên phần cứng tương đối hạn hẹp của PS trên PYNQ-Z2 [26] thì việc ứng dụng AXI DMA cho giao tiếp giữa PS và PL trở thành điều bắt buộc. Hình 3.19 minh họa một cách trực quan thuật toán xử lý khi PS trong quá trình hoạt động.



Hình 3.19: Lưu đồ khối PS

3.4 Triển khai và kiểm thử

3.4.1 Công cụ sử dụng

Bảng 3.11: Danh sách phần mềm sử dụng

Mục đích	Công cụ	Ngôn ngữ
Mô phỏng và kiểm thử	Questa Sim	SystemVerilog, Verilog
Tạo bộ dữ liệu chuẩn cho quá trình kiểm thử	MATLAB	MATLAB
Tổng hợp và triển khai thiết kế trên phần cứng; Đọc dữ liệu từ ILA	Vivado	Tcl
Thiết kế và triển khai phần mềm nhúng	Vitis Classic	C
Đọc dữ liệu debug console qua USB	PuTTY	
Stress Test thông lượng TCP	iPerf2	
Chạy script truyền dữ liệu qua giao thức TCP và stress test thông lượng thực tế	Powershell	Python
Thiết lập Mesh Network	Tailscale	

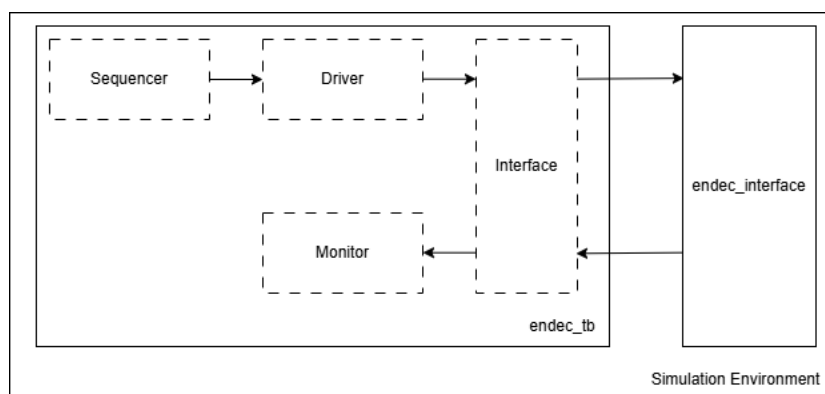
Bảng 3.12: Danh sách phần cứng sử dụng

Mục đích	Công cụ	Nhà sản xuất
Chạy Endec Server	PYNQ-Z2	TUL
Router kết nối với Endec Server	Beryl AX	GL.iNet
Cung cấp kết nối mạng	Modem E8372	Huawei
Thử nghiệm thông lượng thuật toán MATLAB; Dùng làm client cho kết nối TCP	Laptop Legion 5	Lenovo

Với độ phức tạp của đề tài này, quá trình triển khai được chia thành ba giai đoạn chính: (i) thiết kế phần cứng trên PL, (ii) triển khai thiết kế PL lên phần cứng thực tế và (iii) lập trình nhúng cho PS. Nhằm đảm bảo độ tin cậy của hệ thống, sau khi hoàn thành mỗi giai đoạn, người viết sẽ tiến hành kiểm thử toàn diện các chức năng đã triển khai. Cách tiếp cận này giúp hạn chế tối đa các lỗi tiềm ẩn đồng thời cho phép phát hiện và xử lý sớm các vấn đề phát sinh trong quá trình phát triển hệ thống.

3.4.2 Kiểm thử bằng mô phỏng

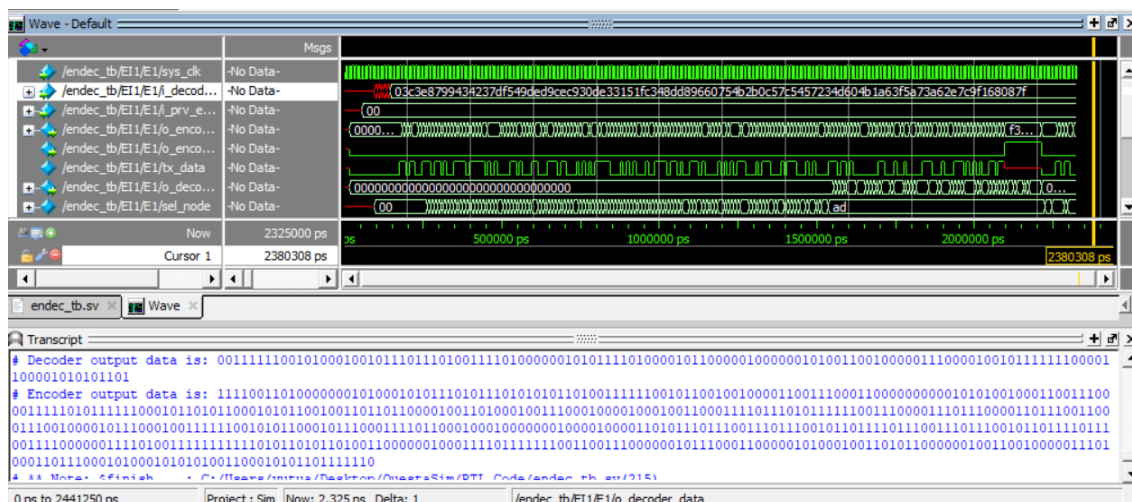
Môi trường kiểm thử cho thiết kế cần kiểm tra (DUT) được dựa trên kiến trúc kiểm thử tổng quát (UVM) [27]. Tuy nhiên, vì nhiệm vụ chính của đề tài này là thiết kế và triển khai mà không phải kiểm thử nên môi trường kiểm thử đã được tối giản hóa và chỉ giữ lại những gì cần thiết nhất. Bên cạnh đó, việc kiểm thử cũng sẽ tập trung vào khả năng vận hành đúng theo thiết kế mà không đề cập tới khả năng chống lỗi của hệ thống nếu như bộ dữ liệu cố tình sai khác với thiết kế ban đầu.



Hình 3.20: Sơ đồ khối của môi trường kiểm thử bằng mô phỏng

Hình 3.20 mô tả kiến trúc tổng quan của môi trường kiểm thử. Cụ thể, hệ thống hoạt động theo cơ chế sau: Sequencer có nhiệm vụ chuyển đổi các dữ liệu mẫu (được tạo bằng MATLAB) sang định dạng tương thích với endec_interface. Dữ liệu sau khi được xử lý sẽ được Driver truyền tới endec_interface thông qua Interface. Song song đó, Monitor sẽ giám sát quá trình xử lý của endec_interface và thu nhận kết quả phản hồi thông qua Interface. Quy trình kiểm thử được thực hiện bằng cách: (i) nạp dữ liệu đầu vào vào endec_interface, (ii) so sánh kết quả đầu ra với bộ dữ liệu chuẩn. Phương pháp này cho phép đánh giá chính xác hiệu năng và độ tin cậy của endec_interface.

Thiết kế này sau đó được triển khai trên phần mềm Questa Sim, kết quả của việc kiểm thử bằng mô phỏng được thể hiện ở Hình 3.21.



Hình 3.21: Quá trình kiểm thử bằng mô phỏng

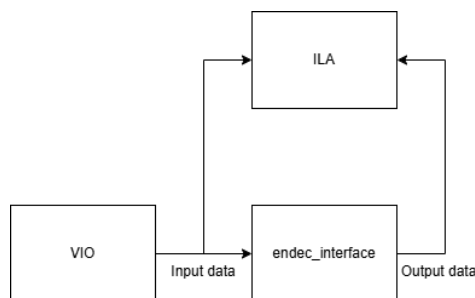
Ví dụ minh họa cụ thể về quá trình kiểm thử bằng phương pháp mô phỏng được trình bày trong Hình 3.21. Trong thử nghiệm này, hệ thống đang được chạy với cấu hình thông số kỹ thuật bao gồm tốc độ mã hóa đặt ở mức 1/3, chiều dài ràng buộc được thiết lập là 9, đồng thời sử dụng bộ đa thức sinh tương ứng cho ba đầu ra ở dạng hệ 8 lần lượt là 557, 663 và 711. Kết quả mô phỏng thu được cho thấy tín hiệu đầu ra hoàn toàn khớp 100% so với bộ dữ liệu mẫu tham chiếu được tạo ra từ MATLAB. Mặc dù đây mới chỉ là kết quả thử nghiệm trên một bộ dữ liệu cụ thể, nhưng với thiết kế đã được tổng quát hóa để có thể áp dụng cho nhiều trường hợp khác nhau, cùng với việc toàn bộ 704/704 bit dữ liệu đều khớp chính xác, kết quả này đã chứng minh một cách thuyết phục rằng thiết kế hiện tại hoạt động đúng như mong đợi và đạt được độ tin cậy cao.

3.4.3 Triển khai trên phần cứng và kiểm thử bằng ILA

a, Triển khai thiết kế PL

Sau khi kiểm thử mô phỏng về mặt chức năng thành công, thiết kế PL sẽ được nạp vào phần mềm Vivado để tiến hành tổng hợp thành các phần tử LUT và FF và triển khai các phần tử này trên phần cứng của PYNQ-Z2. Cũng tương tự như kiểm thử trên mô phỏng, việc kiểm thử trên phần cứng cũng bao gồm hai bước là: (i) nạp dữ liệu vào thiết kế cần kiểm thử và (ii) so sánh dữ liệu đầu ra với bộ dữ liệu mẫu.

Tuy nhiên, cấu trúc của thiết kế trên phần cứng sẽ khác với mô phỏng nên môi trường kiểm thử phần cứng cũng phải được thay đổi để phù hợp với sự khác biệt này. Hình 3.22 mô tả môi trường kiểm thử thiết kế được triển khai trên phần cứng. Trong đó, khối giao tiếp ảo đầu vào/ra (Virtual Input/Output - VIO) có tác dụng gửi dữ liệu đầu vào tới endec_interface (Hình 3.23) và dữ liệu đầu ra sẽ được ILA đọc và hiển thị dưới dạng tín hiệu sóng trong Vivado (Hình 3.24).



Hình 3.22: Sơ đồ khối môi trường kiểm thử phần cứng

Hardware Manager - localhost:xilinx_tcf/Xilinx/1234-tuLA

control.sv

hw_ila_1

hw_vios

Hardware

Debug Probe Properties

Dashboard Options

hw_vio_1

🔍

⏮

⏪

⏩

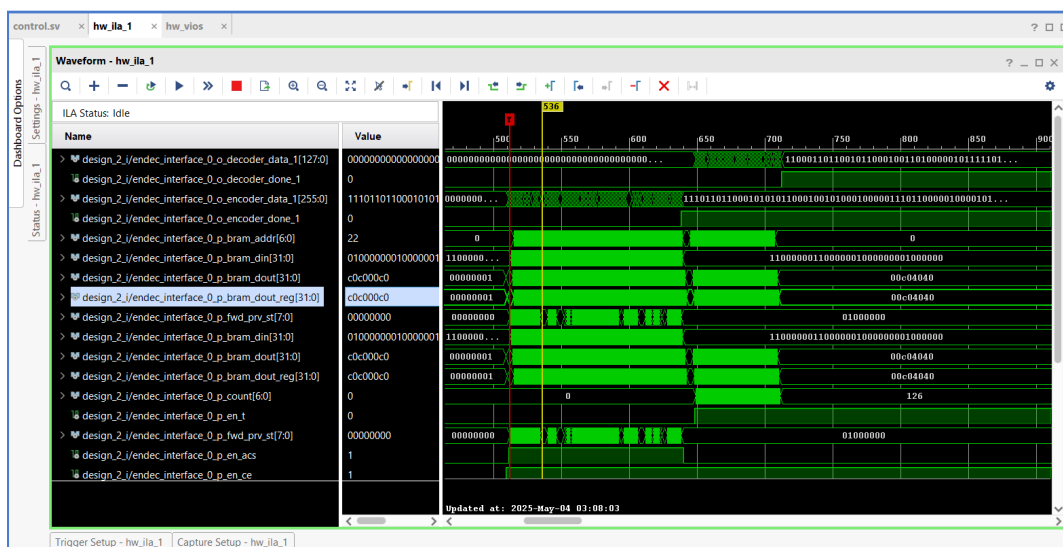
⏭

+

-

Name	Value	Activity	Direction	VIO
> 📡 design_2_i/vio_0_probe_out5[255:0]	[B] 1110_1101_1000_1010_1011_0001_0010_1000_1000_0011_1011_0000_0100	▼	Output	hw_vio_1
> 📡 design_2_i/vio_0_probe_out3[26:0]	[B] 000_0000_0011_0011_0111_1110_1101	▼	Output	hw_vio_1
📡 design_2_i/vio_0_probe_out0	[B] 1	▼	Output	hw_vio_1
> 📡 design_2_i/vio_0_probe_out6[7:0]	[B] 0000_0000	▼	Output	hw_vio_1
> 📡 design_2_i/vio_0_probe_out4[127:0]	[B] 1100_0110_1100_1011_0001_0011_0100_0001_0111_1101_1011_1011_0011	▼	Output	hw_vio_1
📡 design_2_i/vio_0_probe_out2	[B] 0	▼	Output	hw_vio_1
📡 design_2_i/vio_0_probe_out1	[B] 1	▼	Output	hw_vio_1
📡 design_2_i/encdec_interface_0_o_encoder_done	[B] 1		Input	hw_vio_1
> 📡 design_2_i/encdec_interface_0_o_encoder_data[255:0]	[B] 1110_1101_1000_1010_1011_0001_0010_1000_1000_0011_1011_0000_0100_0010		Input	hw_vio_1
📡 design_2_i/encdec_interface_0_o_decoder_done	[B] 1		Input	hw_vio_1
> 📡 design_2_i/encdec_interface_0_o_decoder_data[127:0]	[B] 1100_0110_1100_1011_0001_0011_0100_0001_0111_1101_1011_1011_0011_0001		Input	hw_vio_1

Hình 3.23: Thiết lập dữ liệu đầu vào trên VIO



Hình 3.24: Dữ liệu đọc được từ ILA

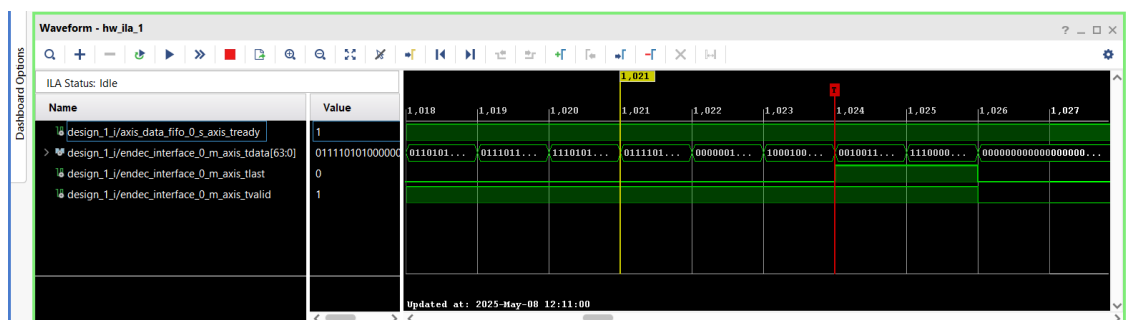
Để sử dụng ILA như một Logic Analyzer đúng nghĩa, ta có thể tạo thêm đầu ra mới cho các khối chức năng và nối trực tiếp tín hiệu nội bộ trong các khối chức năng với đầu ra đó. Việc này tương đương với dùng que đo của Logic Analyzer để tiến hành đo đặc đường dây tín hiệu trong mạch thực tế. Các tín hiệu có tiền tố p_ trong Hình 3.24 đều đã được ILA đọc dữ liệu thông qua kỹ thuật này.

Thiết kế của PL trong mục này vẫn đang là dạng dữ liệu thô mà chưa được chuyển đổi cho tương thích với giao tiếp AXI. Tuy vậy nhưng các tín hiệu thu được bởi ILA trong Hình 3.24 cũng hoàn toàn khớp với bộ dữ liệu mẫu từ MATLAB. Từ đó, ta có thể kết luận việc triển khai thiết kế PL trên phần cứng đã thành công và tiếp tục triển khai giai đoạn tiếp theo.

b, Triển khai giao tiếp giữa PS và PL

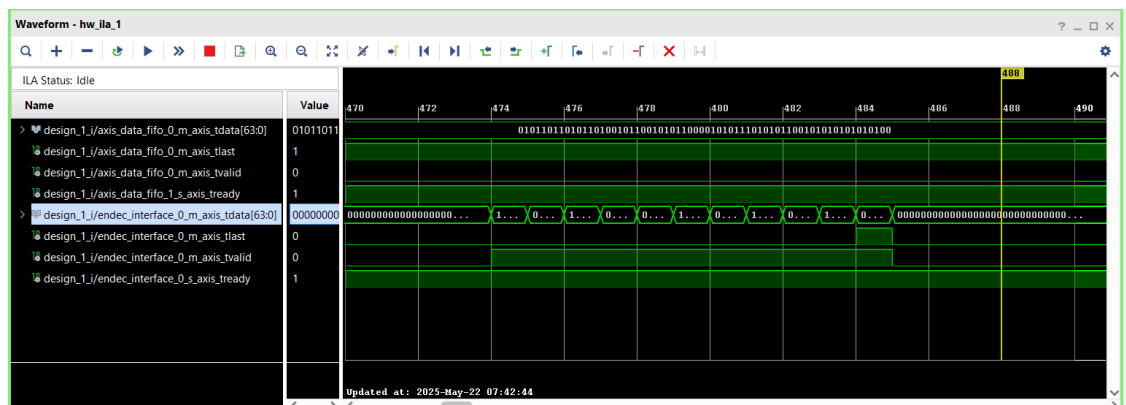
Phần này sẽ tiếp tục triển khai giao tiếp truyền dữ liệu giữa PS và PL trên cơ sở kiểm thử thành công trong việc sử dụng dữ liệu thô để truyền vào PL của phần a. Để việc truyền nhận dữ liệu giữa PL và PS diễn ra hiệu quả thì việc thiết kế giao thức kết nối là vô cùng quan trọng. Như đã trình bày ở mục 3.2.1, endec_interface và khối AXI DMA sẽ giao tiếp thông qua giao thức AXI, quá trình kiểm thử chức năng của hệ thống cho đến bước hiện tại cũng đã diễn ra thành công.

Ví dụ minh họa ở Hình 3.25 và Hình 3.26 sẽ chứng minh rõ nét nhất giá trị của việc ứng dụng ILA trong quá trình kiểm thử và sửa lỗi cho hệ thống.



Hình 3.25: Phát hiện lỗi thông qua ILA

Từ Hình 3.25 ta có thể thấy được lỗi trong chức năng TX của khối endec_interface. Tín hiệu t_last đã được bật lên ở hai thay vì một nhịp dữ liệu cuối. Điều này vi phạm giao thức AXI và lỗi này sẽ làm ảnh hưởng đến luồng dữ liệu giao tiếp từ PL đến PS và PS đến client.

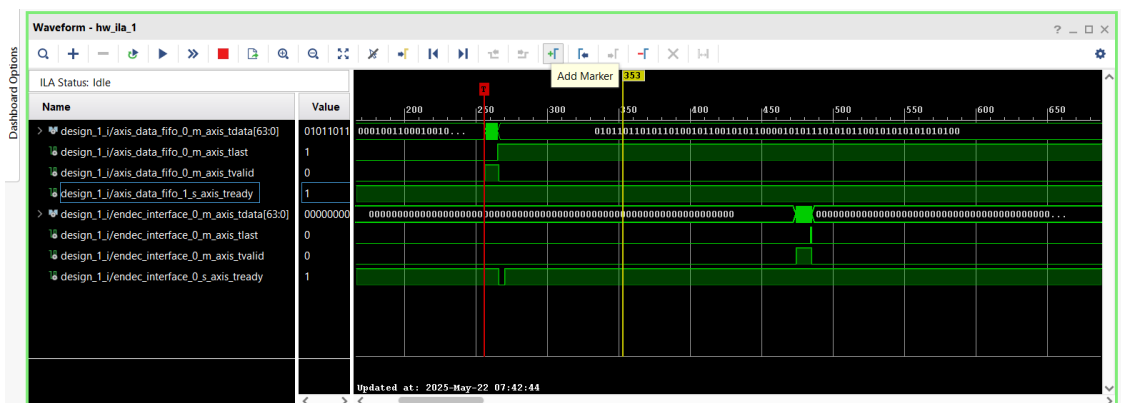


Hình 3.26: Waveform sau khi sửa lỗi

Sau khi sửa lỗi, tín hiệu `t_last` đã hoạt động đúng như mong đợi và có dạng như ở Hình 3.26. Điều này cũng chỉ ra một điểm yếu rất lớn của việc kiểm thử bằng mô phỏng: môi trường mô phỏng mang tính chủ quan của người thiết kế và từ đó tính toàn diện của các trường hợp kiểm thử là không được đảm bảo. Chính vì vậy, việc sử dụng ILA trong quá trình phát triển và kiểm thử hệ thống là điều bắt buộc.

3.4.4 Triển khai phần mềm nhúng và kiểm thử trực tiếp

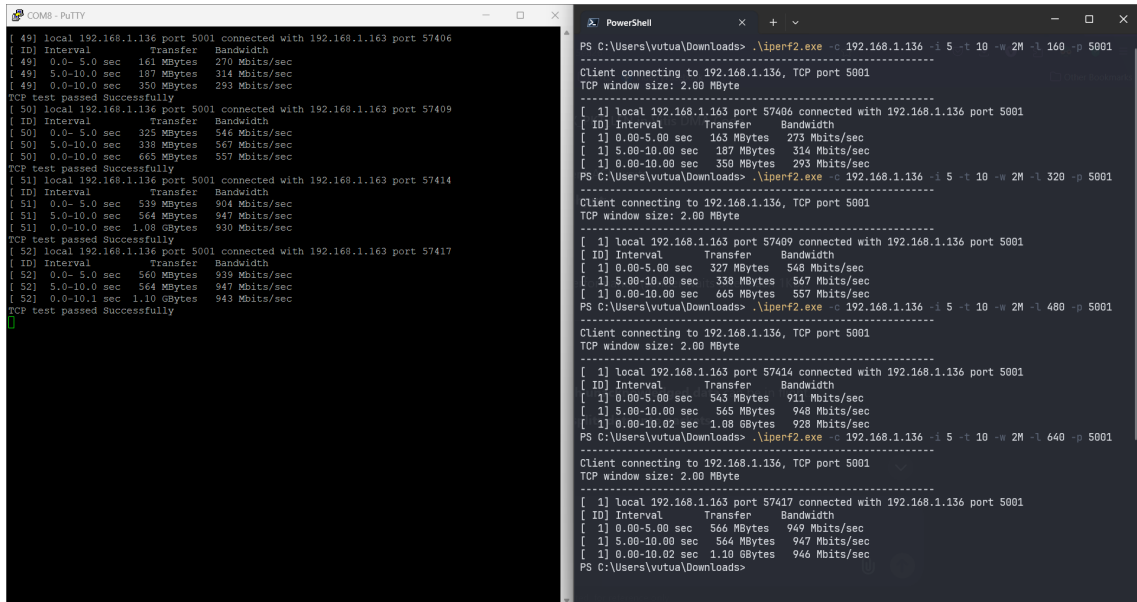
Hình 3.18 và Hình 3.19 minh họa toàn bộ quy trình xử lý dữ liệu trong hệ thống. Sau khi hoàn thành các giai đoạn triển khai, kiểm thử và khắc phục lỗi trên PL, công đoạn cuối cùng là phát triển phần mềm nhúng cho PS. Phần mềm này có hai nhiệm vụ chính: (i) điều khiển khối AXI DMA thông qua AXI DMA Controller để quản lý luồng dữ liệu tốc độ cao giữa PS và PL, đồng thời (ii) thiết lập giao tiếp mạng với client thông qua lwIP TCP Server, tạo thành một hệ thống hoàn chỉnh có khả năng xử lý và truyền tải dữ liệu hiệu quả.



Hình 3.27: PS gửi một khung dữ liệu

Hình 3.27 minh họa chi tiết các tín hiệu giao tiếp được ghi lại bằng công cụ ILA trên bus dữ liệu kết nối giữa khối xử lý phần cứng PL và module AXI DMA. Quá trình này diễn ra sau khi bộ xử lý hệ thống PS hoàn tất việc cấu hình các thanh ghi của AXI DMA và thực hiện gửi thành công một khung dữ liệu đầy đủ từ bộ nhớ hệ thống sang phía PL.

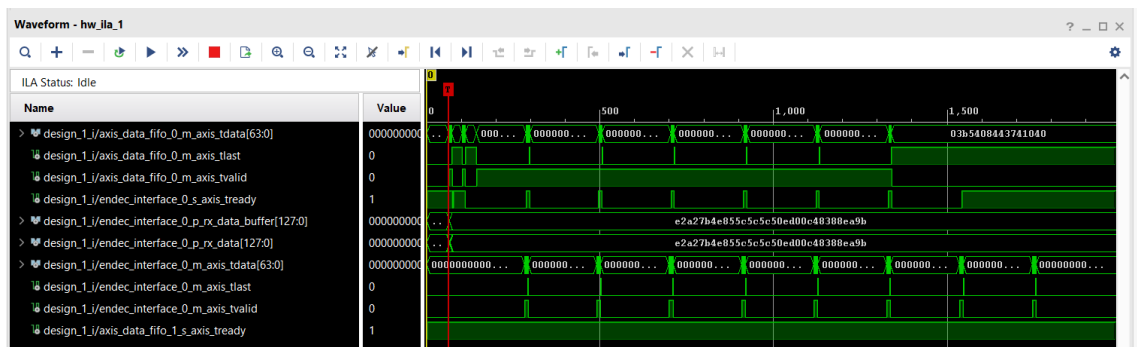
Từ kết quả phân tích tín hiệu ILA, có thể quan sát rõ ràng rằng phía PL đã nhận dữ liệu chính xác và đưa ra tín hiệu phản hồi sau khoảng thời gian xử lý tương đương với 209 chu kỳ xung nhịp. Khoảng thời gian đáp ứng này bao gồm cả độ trễ xử lý tín hiệu và thời gian truyền dẫn dữ liệu qua bus kết nối, cho thấy quá trình truyền nhận dữ liệu giữa hai khối đã diễn ra đúng như thiết kế.



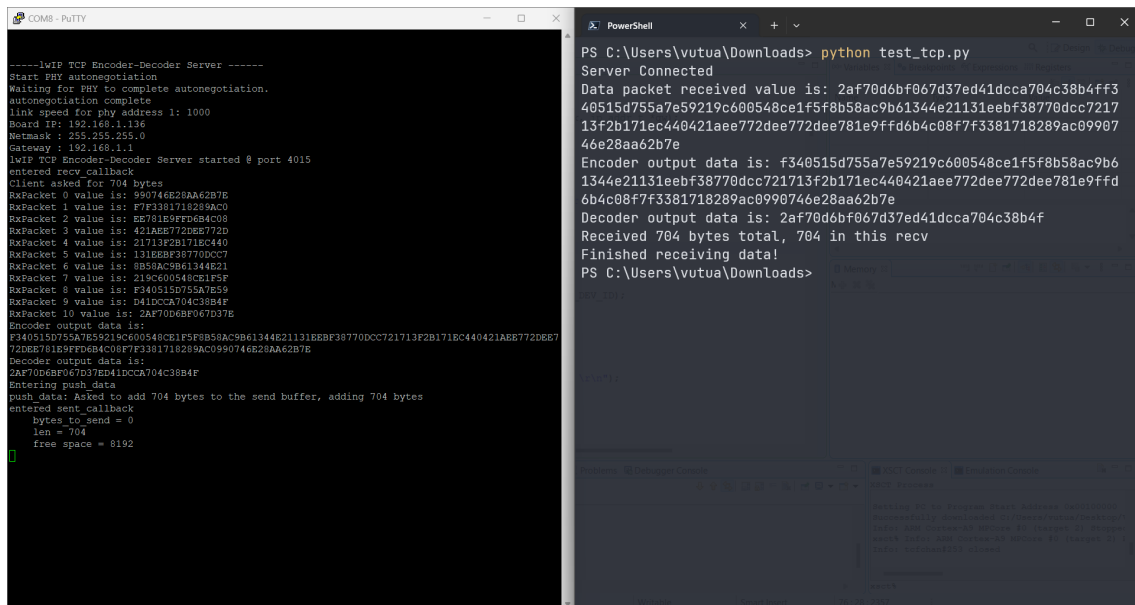
Hình 3.28: Tìm khung dữ liệu TCP cho thông lượng tối ưu

Mặc dù TCP được ưa chuộng nhờ khả năng đảm bảo tính toàn vẹn dữ liệu, giao thức này lại không tối ưu về mặt thông lượng do phải chịu chi phí xử lý (overhead) cho các cơ chế kiểm soát lỗi. Điều này dẫn đến yêu cầu quan trọng trong việc tối ưu hóa kích thước khung dữ liệu giữa client và Endec Server. Kết quả thử nghiệm hệ thống với kết nối Ethernet 1Gbps trong Hình 3.28 cho thấy: thông lượng đạt giá trị bão hòa 950 Mbps tại kích thước khung dữ liệu khoảng 500 byte. Để đạt được hiệu suất tối ưu, kích thước khung dữ liệu cần được chọn lớn hơn ngưỡng 500 byte này.

Sau khi tìm được cấu hình tối ưu cho kết nối giữa client và Endec Server, ta tiến hành thử nghiệm gửi tập tin dữ liệu từ client qua Endec Server. Hình 3.29 cho thấy tín hiệu đã được PS tiếp nhận thành công từ client và truyền tới PL để xử lý. Tương tự như vậy, Hình 3.30 đã chứng tỏ được tín hiệu từ PL đã được PS tiếp nhận thành công và chuyển tiếp đến client. Kết quả cuối cùng mà client nhận được so sánh với bộ dữ liệu mẫu từ MATLAB và cho ra kết quả đúng 100%.

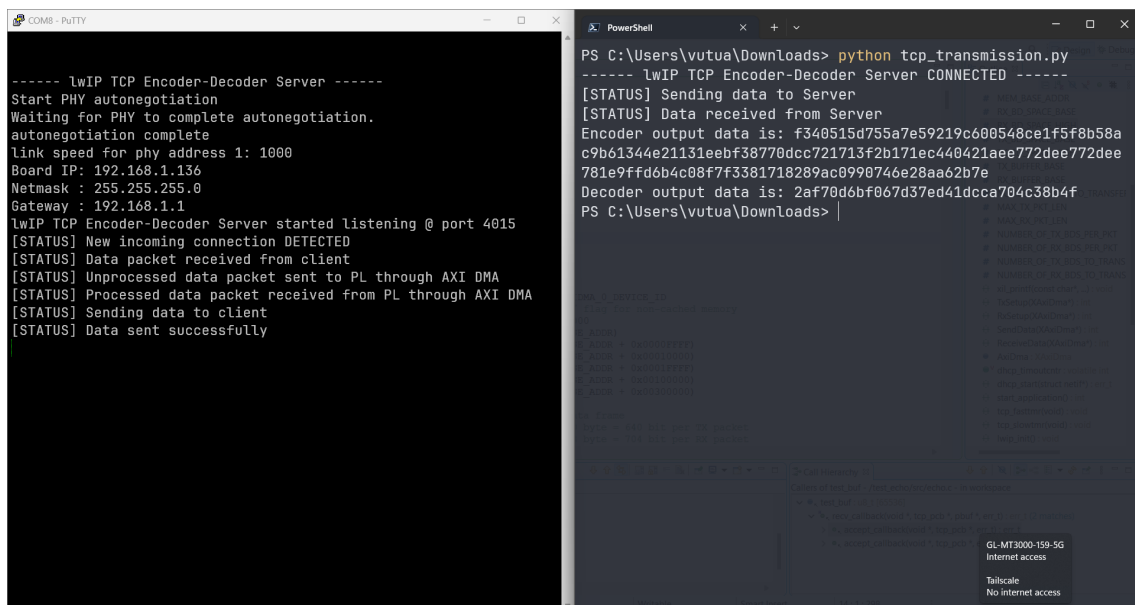


Hình 3.29: Client gửi một nhóm dữ liệu

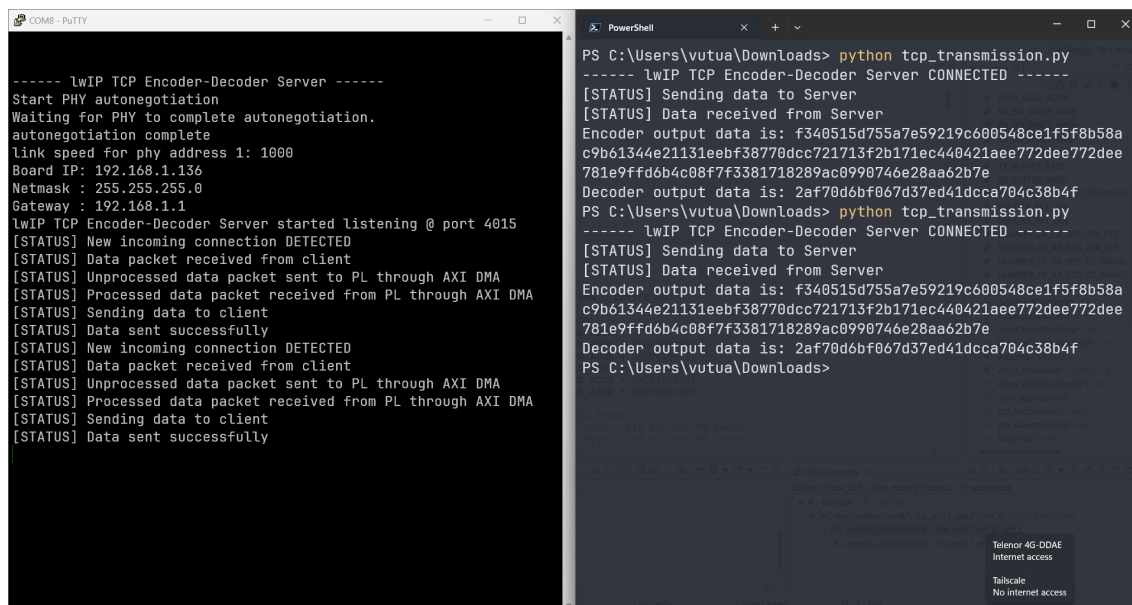


Hình 3.30: Kiểm thử kết nối với client

Tiếp theo, ta tiến hành kiểm thử kết nối Tailscale. Endec Server được kết nối trực tiếp với router Beryl AX qua cổng Ethernet. Trong Hình 3.31, client kết nối vào mạng GL-MT3000-159-5G (mạng Wi-Fi do router Beryl AX phát ra) và giao tiếp thành công với Endec Server – đây là trường hợp kết nối chung mạng LAN. Ở trường hợp thứ hai (Hình 3.32), dù client thuộc một mạng LAN khác, nhưng nhờ cả router Beryl AX và client cùng tham gia vào Mesh Network của Tailscale, việc truyền nhận dữ liệu giữa client và Endec Server vẫn diễn ra bình thường mà không cần mở cổng mạng trên router.



Hình 3.31: Kiểm thử client cùng mạng LAN



Hình 3.32: Kiểm thử client kết nối với Tailscale

3.4.5 Kiểm thử toàn diện hệ thống

Sau khi hoàn thành việc kiểm thử các chức năng cơ bản, quá trình kiểm thử toàn diện toàn hệ thống sẽ được triển khai. Trong giai đoạn này, các bộ dữ liệu mẫu sẽ được tự động sinh ngẫu nhiên bằng phần mềm MATLAB, với các tham số đầu vào tuân thủ theo khoảng giá trị được quy định chi tiết trong Bảng 3.13.

Bảng 3.13: Khoảng giá trị của các tham số

Tham số	Khoảng giá trị
Chiều dài ràng buộc	4 : 9
Tốc độ mã	1/2 : 1/3
Đa thức sinh cho từng đầu ra	0 : $2^9 - 1$
Trạng thái trước của thanh ghi dịch	0 : $2^8 - 1$
Dữ liệu cần mã hóa	0 : $2^{192} - 1$
Dữ liệu cần giải mã	0 : $2^{384} - 1$

MATLAB sẽ được sử dụng để sinh ra tổng cộng 15000 khung dữ liệu ngẫu nhiên, với các tham số đầu vào được giới hạn trong khoảng giá trị đã xác định trước. Toàn bộ tập dữ liệu này sau đó sẽ được chuyển đến Endec Server thông qua giao thức truyền thông đã được thiết lập để tiến hành quá trình kiểm thử chức năng. Dưới đây là các kết quả chi tiết thu được từ quá trình kiểm thử này

```

1  === SCOREBOARD ===
2  Total samples compared: 15000
3  Matching samples: 13765
4  Mismatched samples: 1235
5
6  Mismatch details:
7
8  Sample 2:
9  Decoder output mismatch!
10 First mismatch at bit position 1
11
12 Sample 3:
13 Decoder output mismatch!
14 First mismatch at bit position 0
15
16 Sample 30:
17 Decoder output mismatch!
18 First mismatch at bit position 0
19
20 Sample 44:
21 Decoder output mismatch!
22 First mismatch at bit position 0
23
24 Sample 77:
25 Decoder output mismatch!
26 First mismatch at bit position 0
27
28 Sample 84:
29 Decoder output mismatch!

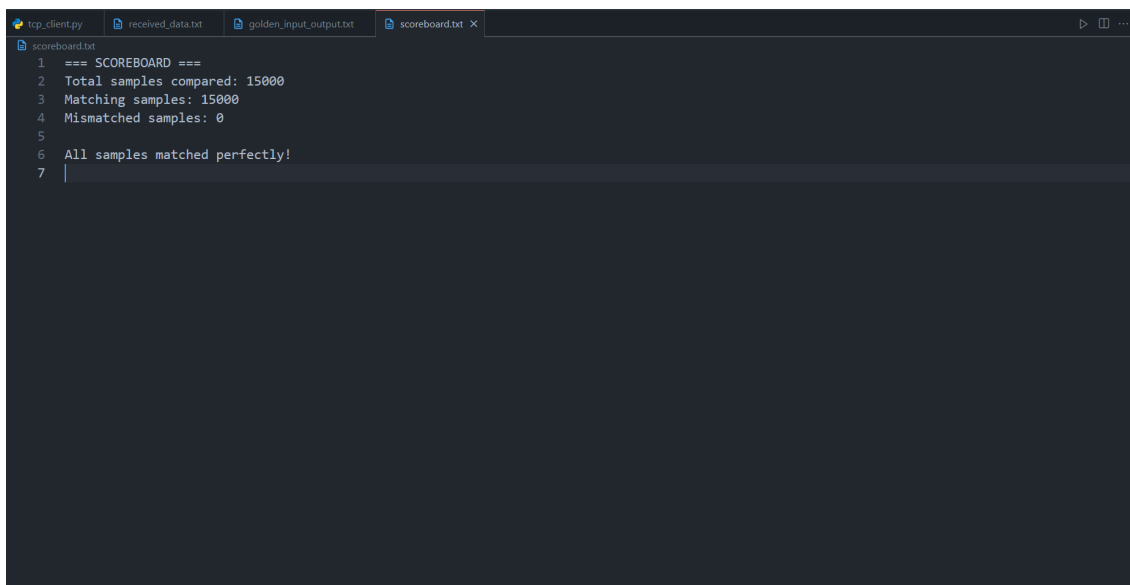
```

Hình 3.33: Kết quả kiểm thử ban đầu

Hình 3.33 cho thấy một hiện tượng đáng chú ý: dù dữ liệu mã hóa luôn chính xác, quá trình giải mã lại cho kết quả sai. Tỷ lệ lỗi 8.23% quan sát được trong hệ thống mã hóa/giải mã có thể bắt nguồn từ việc sử dụng các đa thức sinh không tối ưu. Khi áp dụng phương pháp ngẫu nhiên hóa toàn bộ đa thức sinh, hệ thống có thể tạo ra những đa thức không đạt chuẩn về khoảng cách tự do, thậm chí bao gồm cả các đa thức có khả năng gây lỗi thảm họa (catastrophic) [28][29]. Để khắc phục vấn đề này, thuật toán sinh ngẫu nhiên trong MATLAB sẽ được điều chỉnh, chỉ sử dụng các đa thức sinh đã được kiểm chứng và phổ biến trong các hệ thống thông tin thực tế:

Bảng 3.14: Đa thức sinh sau điều chỉnh

Chiều dài ràng buộc	Tốc độ mã 1/2	Tốc độ mã 1/3
4	15 – 17	15 – 17 – 13
5	23 – 35	25 – 33 – 37
6	53 – 75	55 – 64 – 71
7	133 – 171	133 – 145 – 171
8	247 – 371	247 – 371 – 357
9	561 – 753	557 – 663 – 711



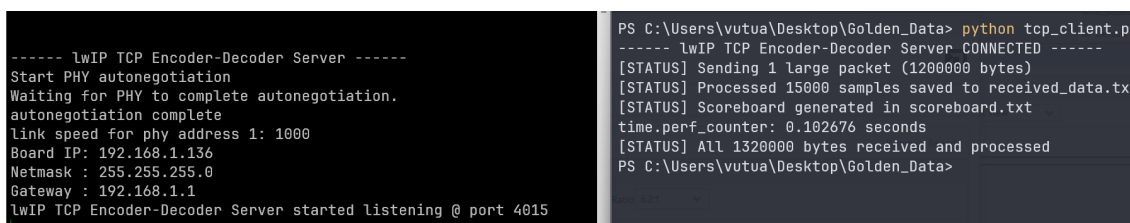
```

1 === SCOREBOARD ===
2 Total samples compared: 15000
3 Matching samples: 15000
4 Mismatched samples: 0
5
6 All samples matched perfectly!
7

```

Hình 3.34: Kết quả kiểm thử sau khi điều chỉnh đa thức sinh

Hình 3.34 trình bày kết quả kiểm thử hệ thống sau khi đã tối ưu hóa bộ đa thức sinh. Kết quả đạt được độ chính xác 100% đã khẳng định rằng nguyên nhân gây sai số trong các thử nghiệm trước đó xuất phát từ đặc tính của bộ đa thức sinh mà không phải do lỗi của hệ thống mã hóa/giải mã. Trên cơ sở này, nghiên cứu tiếp tục tiến hành đánh giá thông lượng thực tế của toàn hệ thống.



```

----- lwIP TCP Encoder-Decoder Server -----
Start PHY autonegotiation
Waiting for PHY to complete autonegotiation.
autonegotiation complete
link speed for phy address 1: 1000
Board IP: 192.168.1.136
Netmask : 255.255.255.0
Gateway : 192.168.1.1
lwIP TCP Encoder-Decoder Server started listening @ port 4015

PS C:\Users\vutua\Desktop\Golden_Data> python tcp_client.py
----- lwIP TCP Encoder-Decoder Server CONNECTED -----
[STATUS] Sending 1 large packet (1200000 bytes)
[STATUS] Processed 15000 samples saved to received_data.txt
[STATUS] Scoreboard generated in scoreboard.txt
time.perf_counter: 0.102676 seconds
[STATUS] ALL 1320000 bytes received and processed
PS C:\Users\vutua\Desktop\Golden_Data>

```

Hình 3.35: Kết quả kiểm thử thông lượng

Theo kết quả trong Hình 3.35, hệ thống ghi nhận độ trễ xử lý trung bình là 0.1 giây cho mỗi chu kỳ từ khi client gửi dữ liệu đến khi nhận được dữ liệu đã xử lý. Trên cơ sở 15000 khung dữ liệu thử nghiệm, với thông số kỹ thuật mỗi khung bao gồm 192 bit dữ liệu đầu vào cần mã hóa và 128 bit dữ liệu đầu ra đã được giải mã, ta có thể tính toán được thông lượng của Endec Server thông qua công thức sau:

$$\text{Thông lượng mã hóa} = \frac{\text{Số bit mã hóa đầu vào}}{\text{Thời gian xử lý}} \quad (3.1)$$

$$\text{Thông lượng giải mã} = \frac{\text{Số bit giải mã đầu ra}}{\text{Thời gian xử lý}} \quad (3.2)$$

Công thức tính thông lượng (3.1) và (3.2) dựa vào lượng thông tin hữu dụng mà bên phát/bên thu cần truyền/ nhận. Áp dụng công thức trên, ta có thể tính được thông lượng mã hóa/giải mã của Endec Server lần lượt là 28.8Mbps và 19.2Mbps

3.5 Tổng kết

3.5.1 Đánh giá thông số hoạt động của Endec Server

Trong quá trình phát triển, hệ thống đã trải qua nhiều lần điều chỉnh và tối ưu hóa nhằm cải thiện thông lượng, hiệu suất, khắc phục lỗi và tối ưu tài nguyên. Do đó, việc đánh giá khả năng hoạt động của hệ thống không thể thực hiện trong giai đoạn triển khai giữa chừng mà chỉ có thể tiến hành sau khi đã hoàn thành kiểm thử chức năng và tối ưu hóa hiệu năng. Dưới đây là bảng tổng hợp các thông số của hệ thống sau khi đã vượt qua kiểm thử chức năng và được tối ưu hóa hiệu suất:

Bảng 3.15: Tài nguyên PL sử dụng

Tài nguyên	Giá trị sử dụng tuyệt đối	Giá trị sử dụng tương đối
LUT	43095	81%
LUTRAM	734	4%
FF	33739	32%
BRAM	125	89%
BUFG	2	6%

Kết quả tổng hợp trong Bảng 3.15 cho thấy thiết kế hiện đã tối ưu hóa việc sử dụng tài nguyên PL để đạt hiệu năng xử lý cao. Đáng chú ý, tài nguyên LUT đạt mức sử dụng 81% (43,095 LUT) do yêu cầu xử lý phức tạp của khối add_compare_select - nơi cần thực hiện mạng lưới tính toán đồng thời 1024 dịch chuyển cho mỗi bước thời gian. Tuy nhiên, việc vượt ngưỡng 80% này tiềm ẩn nguy cơ nghẽn tín hiệu khi triển khai phần cứng, vấn đề sẽ được phân tích sâu tại mục 4.2.3 trong Chương 4.

Mặt khác, tài nguyên FF chỉ đạt 32% (33,739 FF) do chủ yếu đóng vai trò trung chuyển tín hiệu và không ảnh hưởng đáng kể đến hiệu năng hệ thống. Các tài nguyên khác thể hiện mức sử dụng đa dạng: BRAM đạt 89% (125 khối) cho thấy việc tận dụng tối đa bộ nhớ, trong khi LUTRAM (4%) và BUFG (6%) được sử dụng ở mức khiêm tốn.

Bảng 3.16: Thông số hoạt động của hệ thống

Thông số	Giá trị
Chiều dài ràng buộc hỗ trợ	4 : 9
Tốc độ mã hỗ trợ	1/2 : 1/3
Đa thức sinh hỗ trợ	0 : $2^9 - 1$
Độ sâu truy ngược	64
Số bit mã hóa đầu vào trong một khung dữ liệu	192 bit
Số bit giải mã đầu ra trong một khung dữ liệu	128 bit
Xung nhịp hoạt động	81.25 MHz
Công suất tiêu thụ	1.64 W
Nhiệt độ khi hoạt động	43.9°C

Như kết quả trình bày trong Bảng 3.16, hệ thống hiện không hỗ trợ chiều dài ràng buộc 3 - một thông số khá phổ biến trong các hệ thống tương tự. Nguyên nhân cụ thể và giải pháp khắc phục cho vấn đề này sẽ được phân tích chi tiết tại mục 4.2.2 trong Chương 4.

Về hiệu năng hoạt động, hệ thống đạt được xung nhịp 81.25 MHz - con số tuy khiêm tốn nhưng cần được đánh giá trong bối cảnh kiến trúc Radix-4 đặc thù của bộ giải mã Viterbi. Nguyên nhân chính giới hạn tốc độ xử lý nằm ở đường truyền tới hạn (Critical Path) trong khối add_compare_select, nơi phải thực hiện phép so sánh đồng thời 1024 dịch chuyển trạng thái. Tuy nhiên, cần lưu ý rằng với cơ chế xử lý 2 bit/chu kỳ của kiến trúc Radix-4, hiệu năng thực tế của hệ thống hiện tại ở 81.25 MHz sẽ tương đương với hệ thống với kiến trúc Radix-2 truyền thống hoạt động ở 162.5 MHz khi các điều kiện khác là tương đồng.

Kết quả kiểm thử hệ thống tại mục 3.4.4 cho thấy quá trình xử lý mỗi khung dữ liệu cần 209 chu kỳ xung nhịp. Dựa trên thông số xung nhịp hoạt động được trình bày trong Bảng 3.16 kết hợp với việc biến đổi các công thức (3.1) và (3.2), ta có thể xác định thông lượng hệ thống khi giao tiếp với AXI DMA thông qua các công thức tính toán sau:

$$\text{Thông lượng mã hóa} = \frac{\text{Số bit mã hóa đầu vào trong một khung dữ liệu}}{\text{Chu kỳ xử lý} \times \frac{1}{\text{Xung nhịp hoạt động}}} \quad (3.3)$$

$$\text{Thông lượng giải mã} = \frac{\text{Số bit giải mã đầu ra trong một khung dữ liệu}}{\text{Chu kỳ xử lý} \times \frac{1}{\text{Xung nhịp hoạt động}}} \quad (3.4)$$

Với kiến trúc xử lý song song độc lập giữa các tác vụ TX và RX đã được mô tả trong mục 3.2.1, hệ thống đạt được độ ổn định cao khi xử lý mỗi khung dữ liệu trong 209 chu kỳ xung nhịp. Điều này khẳng định tính chính xác của các công thức (3.3) và (3.4) trong việc đánh giá thông lượng hệ thống khi tích hợp với AXI DMA. Dựa trên kết quả kiểm thử thực tế thu được tại mục 3.4.5, thông lượng hệ thống qua các giai đoạn kiểm thử được tổng hợp thành bảng sau:

Bảng 3.17: Thông lượng xử lý qua các giai đoạn của hệ thống

Giai đoạn	Thông lượng mã hóa	Thông lượng giải mã
Qua AXI DMA	74.6 Mbps	49.8 Mbps
Qua lwIP TCP Server	28.8 Mbps	19.2 Mbps

Kết quả so sánh từ Bảng 3.17 cho thấy thông lượng hệ thống giảm xuống chỉ còn 39% khi xử lý qua lwIP TCP Server so với khi sử dụng trực tiếp AXI DMA. Sự sụt giảm đáng kể này bắt nguồn từ ba yếu tố chính. Thứ nhất, chi phí điều khiển (overhead) cho mỗi tác vụ xử lý đã làm giảm hiệu suất tổng thể. Thứ hai, hệ thống hiện mới chỉ khai thác một nhân xử lý trong khi board PYNQ-Z2 được trang bị bộ xử lý lõi kép, dẫn đến việc chưa tận dụng hết tiềm năng phần cứng. Thứ ba, việc AXI DMA Controller hoạt động ở chế độ polling thay vì interrupt khiến PS phải chờ DMA hoàn thành tác vụ trước khi chuyển sang xử lý lwIP TCP Server, tạo ra các khoảng thời gian chết không cần thiết. Những hạn chế này kết hợp lại đã ảnh hưởng đáng kể đến thông lượng xử lý của toàn hệ thống.

3.5.2 So sánh với các hệ thống khác

Hệ thống Endec Server được phát triển với mục tiêu hướng đến người dùng không chuyên, do đó cần có sự so sánh khả năng hoạt động với các giải pháp tương tự trên thị trường. Hiện nay, các bộ mã hóa tích chập và giải mã Viterbi thường được triển khai dưới dạng module tích hợp trong vi mạch, thiết kế FPGA hoặc chương trình chạy trên CPU/GPU. Tuy nhiên, các giải pháp này đều có những hạn chế chung như không phổ biến, không hỗ trợ đầy đủ các cấu hình thông dụng, không có thông số chuẩn hóa và đặc biệt là khó tiếp cận đối với người dùng phổ thông, khiến chúng trở thành tiêu chí tham chiếu không lý tưởng để đánh giá hệ thống hiện tại.

Trong bối cảnh đó, MATLAB nổi lên như một lựa chọn tham chiếu phù hợp mặc dù có những nhược điểm về hiệu năng và tối ưu tài nguyên. Ưu điểm nổi bật của MATLAB là tính phổ biến cao và dễ tiếp cận đối với đa số người dùng không chuyên. Mặc dù thông lượng xử lý phụ thuộc nhiều vào cấu hình phần cứng nền tảng, nhưng với cùng một thuật toán, kết quả từ MATLAB vẫn có thể cung cấp một

chuẩn đánh giá tương đối khách quan. Chính những yếu tố này khiến MATLAB trở thành điểm tham chiếu hợp lý để đánh giá hiệu năng của hệ thống Endec Server trong nghiên cứu hiện tại. Sau đây là bảng so sánh giữa MATLAB và Endec Server dưới góc nhìn của người dùng với các tiêu chí khác nhau:

Bảng 3.18: So sánh MATLAB và Endec Server

Tiêu chí	MATLAB	Endec Server
Quá trình cài đặt	Cần có license và tải về MATLAB	Kết nối với Mesh Network qua Tailscale
Tài nguyên yêu cầu	Lớn, không phù hợp với các hệ thống nhúng	Có kết nối mạng
Tài nguyên sử dụng	Tiêu tốn tài nguyên khi thực hiện mã hóa/giải mã	Gần như không tiêu tốn tài nguyên
Công suất tiêu thụ	45 W (CPU)*	1.64 W
Độ ổn định	Dễ gặp hiện tượng treo, nghẽn dữ liệu	Luôn hoạt động ổn định
Thông lượng xử lý	Mã hóa: 1.31 Mbps* Giải mã: 0.23 Mbps*	Mã hóa: 28.8 Mbps Giải mã: 19.2 Mbps

* Được thực hiện trên CPU Ryzen 5 5600H

Qua bảng so sánh giữa MATLAB và Endec Server, có thể thấy rõ sự khác biệt đáng kể về hiệu suất và hiệu quả năng lượng giữa hai giải pháp. MATLAB đòi hỏi phải có bản quyền và cài đặt phần mềm công kênh, trong khi Endec Server chỉ cần kết nối mạng đơn giản qua Tailscale. Về mặt tài nguyên, MATLAB tiêu tốn nhiều năng lượng với mức tiêu thụ trung bình là 45W cho các CPU tầm trung phổ biến, ngược lại Endec Server hoạt động ổn định với mức tiêu thụ năng lượng cực thấp chỉ 1.64W.

Khả năng hoạt động ổn định của Endec Server đến từ việc kết hợp hài hòa giữa kiến trúc phần cứng xác định và phần mềm được tối ưu hóa cao. Trên khối xử lý lập trình được PL, hệ thống đạt được tính xác định thời gian thực nhờ kiến trúc xử lý song song cứng, cho phép thực hiện các tác vụ xử lý tín hiệu số một cách độc lập và ổn định mà không phụ thuộc vào hệ điều hành. Điều này giải thích tại sao Endec Server có thể duy trì hoạt động liên tục trong khi MATLAB dễ gặp phải tình trạng treo hệ thống hay nghẽn dữ liệu. Ở khối xử lý ứng dụng PS, giải pháp sử dụng phương pháp lập trình baremetal giúp hệ thống đạt được sự ổn định tối đa. Bằng cách loại bỏ hoàn toàn hệ điều hành, hệ thống tránh được các vấn đề về tranh chấp tài nguyên và giảm thiểu độ trễ xử lý.

Ưu điểm về tính ổn định này càng được khẳng định thông qua hiệu suất xử lý vượt trội của Endec Server. Trong khi MATLAB chỉ đạt được thông lượng mã hóa

1.31 Mbps và giải mã 0.23 Mbps, thì Endec Server có thể xử lý lên tới 28.8 Mbps cho mã hóa và 19.2 Mbps cho giải mã. Sự khác biệt này đến từ việc tối ưu hóa toàn diện từ phần cứng đến phần mềm, đặc biệt là khả năng xử lý tín hiệu số hiệu quả với nhiễu được kiểm soát chặt chẽ. Mặc dù có sự suy giảm đáng kể về thông lượng do quá trình truyền tải TCP qua lwIP TCP Server, Endec Server vẫn duy trì được hiệu suất ấn tượng.

Kết quả so sánh đã chứng minh rằng Endec Server không chỉ đáp ứng được các yêu cầu kỹ thuật đặt ra ban đầu mà còn thể hiện rõ những ưu điểm vượt trội về hiệu suất và tiết kiệm năng lượng so với phương pháp truyền thống sử dụng MATLAB. Điều này khẳng định tính khả thi và giá trị ứng dụng thực tiễn của giải pháp Endec Server trong các hệ thống thực tế.

CHƯƠNG 4. KẾT LUẬN

4.1 Đóng góp nổi bật

4.1.1 Tổng quát hóa thuật toán mã hóa tích chập và giải mã Viterbi

Tính tổng quát hóa là một yêu cầu quan trọng của Endec Server. Do thiết kế PL không thể thay đổi trong lúc hoạt động, việc xây dựng một kiến trúc phần cứng có thể xử lý nhiều cấu hình khác nhau trở thành một thách thức cần phải giải quyết.

Đầu ra của mã hóa tích chập có thể biểu diễn bằng công thức:

$$\sum_{k=0}^{K-1} x^k \cdot y^k$$

Trong đó:

- K : Chiều dài ràng buộc của hệ thống.
- x^k : Giá trị bit thứ k trong thanh ghi dịch.
- y^k : Giá trị bit thứ k trong đa thức sinh.

Ta có thể nhận thấy đa thức sinh hoạt động như một bit mask: nếu $y^k = 1$, giá trị x^k được giữ lại, nếu $y^k = 0$, giá trị x^k bị loại bỏ. Nhận xét trên cho thấy cấu trúc phần cứng cho chiều dài ràng buộc lớn có thể tái sử dụng cho chiều dài ràng buộc nhỏ. Khi dùng cấu trúc K lớn cho K' nhỏ, chỉ cần đặt $y^k = 0$ với mọi $k > K' - 1$. Như vậy, hệ thống có thể linh hoạt mở rộng mà không cần thay đổi kiến trúc phần cứng cơ bản.

4.1.2 Kết hợp mã hóa tích chập và giải mã Viterbi

Thiết kế ban đầu của đề tài chỉ tập trung vào triển khai bộ giải mã Viterbi. Tuy nhiên, trong quá trình thực hiện, người thiết kế nhận thấy rằng việc hỗ trợ tất cả các đa thức sinh khả thi với thông số tốc độ mã 1/3, chiều dài ràng buộc 9 và 1024 dịch chuyển mỗi chu kỳ sẽ đòi hỏi một dung lượng bộ nhớ rất lớn để lưu trữ tất cả giá trị. Nhận thức này đã dẫn đến ý tưởng mới: tính toán giá trị đầu ra dựa trên đa thức sinh đầu vào thay vì lưu trữ trước các kết quả.

Để thực hiện phương pháp này, cần tích hợp thêm bộ mã hóa tích chập vào trong bộ giải mã Viterbi. Mặc dù vậy, do thuật toán mã hóa tích chập có độ phức tạp không cao, việc xử lý song song hai luồng dữ liệu (vừa giải mã vừa mã hóa) không phải là thách thức đáng kể. Từ đó, khái niệm về bộ xử lý kết hợp giải mã/mã hóa đã được hình thành.

4.2 Thảo luận

4.2.1 Tiềm năng cho giải mã quyết định mềm

Một hạn chế đáng kể của bộ giải mã hiện tại là chưa hỗ trợ quyết định mềm. Về mặt kiến trúc, việc tích hợp quyết định mềm vào thiết kế hiện có tương đối đơn giản, chỉ cần điều chỉnh khối `branch_metric`. Tuy nhiên, thách thức chính nằm ở các phép toán đặc thù của quyết định mềm, đòi hỏi thực hiện các phép lũy thừa và khai căn - những toán tử mà chỉ có khối DSP mới được tối ưu để xử lý hiệu quả.

Với yêu cầu xử lý 1024 dịch chuyển trong một bước thời gian cùng giới hạn 220 khối DSP trên PYNQ-Z2 [26], giải pháp khả thi duy nhất để triển khai quyết định mềm là tăng chu kỳ tính toán trong khối `branch_metric`. Tuy nhiên, do luồng dữ liệu được xử lý tuyến tính, bất kỳ sự tắc nghẽn nào ở khối đầu nguồn như `branch_metric` sẽ ảnh hưởng nghiêm trọng đến thông lượng tổng thể của hệ thống.

Xuất phát từ những phân tích trên, người thiết kế đã quyết định không triển khai quyết định mềm cho hệ thống trong phạm vi đề tài này.

4.2.2 Khả năng tương thích giữa chiều dài ràng buộc 3 và Radix-4

Kiến trúc Radix-2 truyền thống có đặc điểm là tất cả các chiều dài ràng buộc thông dụng (từ 3 đến 9) đều sử dụng cùng một thuật toán tương đồng, do số lượng thanh ghi dịch luôn nhỏ hơn một so với chiều dài ràng buộc và lớn hơn số bit xử lý trong mỗi chu kỳ (1 bit). Điều này dẫn đến việc trạng thái trước của thanh ghi dịch luôn ảnh hưởng đến trạng thái sau đối với mọi chiều dài ràng buộc phổ biến.

Tuy nhiên, kiến trúc Radix-4 xử lý 2 bit/chu kỳ tạo ra sự khác biệt đáng kể: với chiều dài ràng buộc 3, số thanh ghi dịch giảm xuống còn 2 (3-1), khiến toàn bộ bit của chu kỳ trước bị đẩy ra khỏi thanh ghi trong mỗi chu kỳ mới. Kết quả là mỗi bước thời gian trong quá trình giải mã trở nên độc lập hoàn toàn, đòi hỏi phải có thuật toán riêng biệt cho các chiều dài ràng buộc lớn hơn 3.

Trong quá trình phát triển, người thiết kế đã thành công trong việc xây dựng thuật toán giải mã cho chiều dài ràng buộc 3 phù hợp với kiến trúc phần cứng hiện có và đã được kiểm thử chức năng đầy đủ. Tuy nhiên, vấn đề nảy sinh khi triển khai trên phần cứng thực tế: việc bổ sung thuật toán riêng này làm tăng gấp đôi kích thước và tài nguyên sử dụng cho khối `add_compare_select` - vốn đã phải xử lý song song 1024 dịch chuyển trong mỗi chu kỳ. Xét đến hạn chế về tài nguyên của board PYNQ-Z2 [26], cùng với thực tế là chiều dài ràng buộc 3 không mang lại hiệu quả khử nhiễu đáng kể cho các hệ thống thực tế và ít được sử dụng trong các ứng dụng hiện nay, người thiết kế đã quyết định không hỗ trợ chiều dài ràng buộc 3 trong thiết kế cuối cùng.

4.2.3 Vấn đề về nghẽn tín hiệu trong triển khai thiết kế

Thiết kế hiện tại sử dụng khoảng 81% tài nguyên LUT trong PL, một con số tưởng chừng khiêm tốn nhưng thực tế đã đạt ngưỡng gần giới hạn. Nguyên nhân chính xuất phát từ việc vị trí các LUT và FF trên PL là cố định, đòi hỏi phải tối ưu hóa cực kỳ cẩn thận trong việc bố trí tín hiệu kết nối và sắp xếp các khối chức năng.

Quá trình triển khai thiết kế trên Vivado thường gặp phải nhiều khó khăn đáng kể, đặc biệt khi mức độ sử dụng tài nguyên vượt quá ngưỡng 85%. Một trong những thách thức chính là hiện tượng tắc nghẽn tín hiệu (congestion), thường trở thành nguyên nhân trực tiếp dẫn đến thất bại trong giai đoạn tổng hợp. Hơn nữa, việc sử dụng tài nguyên ở mức độ cao không chỉ làm giảm hiệu suất thiết kế mà còn kéo dài đáng kể thời gian cần thiết cho quá trình tổng hợp và triển khai. Một điểm hạn chế quan trọng khác nằm ở cách Vivado báo cáo thông tin: công cụ chỉ cung cấp dữ liệu về mức độ sử dụng tài nguyên sau khi toàn bộ quy trình hoàn tất. Điều này buộc người thiết kế phải chờ đợi đến giai đoạn cuối cùng mới có thể xác định được tính khả thi của việc triển khai, gây ra sự chậm trễ đáng kể và cản trở tiến độ phát triển tổng thể.

4.2.4 Tối ưu khối `add_compare_select`

Khối `add_compare_select` là thành phần sử dụng nhiều tài nguyên nhất trong hệ thống do phải xử lý hai tác vụ phức tạp: tìm dịch chuyển có path metric nhỏ nhất trong bốn khả năng và xác định nút nhỏ nhất trong 256 nút. Việc tối ưu hóa khối này đặt ra những thách thức đáng kể về mặt kiến trúc. Nguyên nhân chính khiến khối này tiêu tốn nhiều tài nguyên xuất phát từ đặc tính của LUT - vốn là phần tử tổ hợp không có khả năng lưu trữ. Khi các phép toán được nối tiếp mà không có FF trung gian, số lượng LUT cần thiết sẽ tăng theo hàm mũ. Hơn nữa, bản thân việc triển khai LUT trên PL cũng tốn kém tài nguyên do yêu cầu diện tích lớn, khiến chúng trở thành thành phần quý giá cần được sử dụng hợp lý.

Đối với bài toán tìm nút nhỏ nhất trong 256 nút, giải pháp tương đối đơn giản nhờ tính chất tuyến tính của thuật toán. Bằng cách áp dụng phương pháp chia đôi (tree-based comparison), độ trễ tính toán tăng từ 1 lên 8 chu kỳ (tương ứng $\log_2 256$) nhưng giúp tiết kiệm đáng kể số lượng LUT sử dụng. Bên cạnh đó, độ trễ này chỉ ảnh hưởng ở giai đoạn đầu của quá trình xử lý, và hoàn toàn chấp nhận được trong hệ thống có 209 chu kỳ cho mỗi khung dữ liệu.

Tuy nhiên, việc tối ưu tính toán path metric lại phức tạp hơn nhiều do yêu cầu đặc thù của thuật toán. Khác với việc tìm nút nhỏ nhất, path metric đòi hỏi tính toán theo kiểu feed-forward, nghĩa là mỗi kết quả đầu ra phải được xác định trước

khi nhận đầu vào tiếp theo. Điều này khiến mọi độ trễ đều có tính chất tích lũy qua các chu kỳ xử lý. Trong bối cảnh tài nguyên hạn chế, người thiết kế buộc phải đưa ra giải pháp cân bằng bằng cách chèn thêm FF trung gian, làm tăng độ trễ từ 1 lên 2 chu kỳ cho mỗi bước tính toán. Với độ sâu dò ngược 64 bước, hệ thống phải chịu thêm 64 chu kỳ trễ tổng cộng. Bên cạnh đó, thông lượng hệ thống còn bị giảm 44% do bị tăng thêm 64 chu kỳ xử lý cho mỗi khung dữ liệu. Đây là minh chứng rõ ràng cho sự đánh đổi giữa hiệu năng và tài nguyên sử dụng trong thiết kế FPGA.

4.2.5 Tối ưu khối memory

Việc lưu trữ dữ liệu với yêu cầu 8 bit cho mỗi nút, 256 nút mỗi bước thời gian và tổng cộng 64 bước thời gian đòi hỏi một dung lượng bộ nhớ cực lớn. Thực tế triển khai cho thấy, phương án sử dụng LUT và FF truyền thống vượt xa khả năng đáp ứng của board PYNQ-Z2 [26]. Cụ thể, cách tiếp cận này đã làm mức sử dụng LUT và FF tăng vọt lên lần lượt là 600% và 500%, đồng thời khiến quá trình tổng hợp và triển khai trên Vivado kéo dài hơn 5 giờ do tình trạng nghẽn tín hiệu như đã phân tích trong mục 4.2.3. Những hạn chế này cho thấy giải pháp dựa trên LUT và FF là không khả thi trong thực tế.

Để khắc phục vấn đề trên, giải pháp sử dụng BRAM trong PL được đề xuất nhằm tận dụng tài nguyên phần cứng chuyên dụng cho lưu trữ dữ liệu. Với khả năng truy cập một thanh ghi 36 bit mỗi chu kỳ ở mỗi cổng và hỗ trợ chế độ cổng đôi (True Dual Port), BRAM cho phép đọc/ghi đồng thời qua hai cổng độc lập trong cùng một chu kỳ xung nhịp. Nhờ đó, trạng thái của 4 nút có thể được kết hợp vào một cổng, và với hai cổng hoạt động song song, mỗi khối BRAM có thể xử lý dữ liệu cho 8 nút cùng lúc. Tính toán cho thấy chỉ cần $\frac{256}{8} = 32$ khối BRAM là đủ để đọc/ghi dữ liệu song song cho toàn bộ 256 nút trong một chu kỳ, vừa tối ưu tài nguyên vừa đảm bảo hiệu năng hệ thống.

4.3 Kết luận

Đồ án đã hoàn thành mục tiêu thiết kế và triển khai thành công hệ thống Endec Server tích hợp bộ mã hóa tích chập và giải mã Viterbi trên nền tảng SoC, đáp ứng đầy đủ các yêu cầu về hiệu năng xử lý, tính linh hoạt trong cấu hình và khả năng truy cập từ xa. Thông qua việc áp dụng kiến trúc Radix-4 được tối ưu hóa, hệ thống đạt được thông lượng ấn tượng 28.8 Mbps cho mã hóa và 19.2 Mbps cho giải mã, vượt trội so với các giải pháp phần mềm truyền thống như MATLAB. Những đóng góp nổi bật của nghiên cứu bao gồm việc tổng quát hóa thuật toán để hỗ trợ đa dạng cấu hình từ chiều dài ràng buộc 4-9 và tốc độ mã 1/2-1/3, tích hợp song song cả chức năng mã hóa và giải mã trên cùng kiến trúc phần cứng để tận dụng tối đa tài nguyên FPGA, cũng như triển khai thành công mô hình server với giao thức TCP

hỗ trợ truy cập từ xa thông qua Mesh Network sử dụng Tailscale, mở ra khả năng ứng dụng rộng rãi trong các hệ thống IoT và truyền thông phân tán.

Hệ thống đã khắc phục hiệu quả những hạn chế của các nghiên cứu trước đây về tính khả dụng và khả năng tích hợp, đồng thời giải quyết thành công các thách thức kỹ thuật như vấn đề nghẽn tín hiệu và tối ưu tài nguyên PL. Cụ thể, giải pháp sử dụng kiến trúc BRAM thay thế cho LUT/FF trong khối memory, kết hợp với phương pháp tối ưu hóa khối add_compare_select thông qua chia nhỏ các phép toán phức tạp và triển khai pipeline hóa luồng dữ liệu đã giúp duy trì thông lượng ổn định cho toàn hệ thống. Những kết quả này không chỉ khẳng định tính khả thi của giải pháp tích hợp phần cứng-phần mềm mà còn mở ra nhiều hướng phát triển tiềm năng trong tương lai.

4.4 Hướng phát triển

Để tiếp tục nâng cao hiệu năng và mở rộng phạm vi ứng dụng, các hướng nghiên cứu tiếp theo có thể tập trung vào việc tích hợp giải mã quyết định mềm bằng cách tận dụng khối DSP cho các phép toán phức tạp, tối ưu hóa hiệu suất thông qua việc khai thác song song hai lõi xử lý PS theo chế độ AMP, nghiên cứu thuật toán lai Radix-2/Radix-4 để hỗ trợ đầy đủ chiều dài ràng buộc 3, cũng như tích hợp hệ thống vào các giải pháp SDR hay trung tâm dữ liệu yêu cầu xử lý tín hiệu tốc độ cao. Những phát triển này sẽ tiếp tục củng cố vị thế của Endec Server như một giải pháp toàn diện, cân bằng giữa hiệu suất xử lý và tính linh hoạt, đáp ứng nhu cầu ngày càng cao của các hệ thống truyền thông hiện đại.

TÀI LIỆU THAM KHẢO

- [1] Y. Sun and Z. Ding, “FPGA Design and Implementation of a Convolutional Encoder and a Viterbi Decoder Based on 802.11a for OFDM,” en, vol. 2012, Jul. 2012, Publisher: Scientific Research Publishing. DOI: 10.4236/wet.2012.33019.
- [2] R. Li, Y. Dou, Y. Lei, S. Ni, and S. Guo, “Design and Implementation of the Parameterized Multi-Standard High-Throughput Radix-4 Viterbi Decoder on FPGA,” *IEICE Transactions on Communications*, vol. E95.B, pp. 1602–1611, May 2012. DOI: 10.1587/transcom.E95.B.1602.
- [3] H. P. Basavaraj and P. M., “FPGA Implementation of High Speed Convolutional Encoder and Viterbi Decoder for Software Defined Radio,” in *2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Feb. 2023, pp. 1–4. DOI: 10.1109/ICECCT56650.2023.10179840.
- [4] A. J. Mandwale and A. O. Mulani, “Different Approaches For Implementation of Viterbi decoder on reconfigurable platform,” in *2015 International Conference on Pervasive Computing (ICPC)*, Jan. 2015, pp. 1–4. DOI: 10.1109/PERVASIVE.2015.7086976.
- [5] “Fundamentals of Convolutional Coding,” en, in *Fundamentals of Convolutional Coding*, John Wiley & Sons, Ltd, 2015, pp. i–xv, ISBN: 978-1-119-09879-9. DOI: 10.1002/9781119098799.fmatter.
- [6] A. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*.
- [7] *Viterbi Decoder - Decode convolutionally encoded data using Viterbi algorithm - Simulink*, en. [Online]. Available: <https://www.mathworks.com/help/comm/ref/viterbidecoder.html>.
- [8] Z. Liu, C. Liu, H. Zhang, and L. Zhao, “76.5-Gb/s Viterbi Decoder for Convolutional Codes on GPU,” *IEEE Embedded Systems Letters*, vol. 17, no. 1, pp. 22–25, Feb. 2025, ISSN: 1943-0671. DOI: 10.1109/LES.2024.3416401.
- [9] *TMS320C6416 data sheet, product information and support | TI.com*. [Online]. Available: <https://www.ti.com/product/TMS320C6416>.
- [10] R. Mamarde and S. Khoje, “Viterbi Decoder Using Zynq-7000 AP-SoC,” in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, Jun. 2018, pp. 941–944. DOI: 10.1109/ICCONS.2018.8663202.

- [11] *Acceleration in the AWS Cloud*, en. [Online]. Available: <https://www.amd.com/en/where-to-buy/accelerators/alveo/cloud-solutions/aws.html>.
- [12] P. Black and T. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992, ISSN: 1558-173X. DOI: 10.1109/4.173118.
- [13] R. Gallery, "Hardware/Software Codesign," en, 2003, Publisher: Dublin Institute of Technology. DOI: 10.21427/D7NB29.
- [14] *AMD Zynq™ 7000 SoCs*, en. [Online]. Available: <https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html>.
- [15] *Zynq®-7000 SoC First Generation Architecture*, en-in, May 2019. [Online]. Available: <https://www.mouser.in/xilinx-zynq-7000-socs>.
- [16] *AMBA® AXI4 Interface Protocol*, en. [Online]. Available: <https://shorturl.at/qNCns>.
- [17] *Introduction • AXI DMA LogiCORE IP Product Guide (PG021) • Reader • AMD Technical Information Portal*. [Online]. Available: https://docs.amd.com/r/en-US/pg021_axi_dma/Introduction.
- [18] Russell, *What is a Block RAM in an FPGA? For Beginners*. en-US, Jun. 2022. [Online]. Available: <https://nandland.com/lesson-15-what-is-a-block-ram-bram/>.
- [19] *Integrated Logic Analyzer (ILA)*, en. [Online]. Available: <https://shorturl.at/Giu1Q>.
- [20] *What is TCP (Transmission Control Protocol)?* en-US, Section: Computer Subject. [Online]. Available: <https://shorturl.at/g3RTU>.
- [21] *lwIP - A Lightweight TCP/IP stack - Summary [Savannah]*. [Online]. Available: <https://savannah.nongnu.org/projects/lwip/>.
- [22] *What is Mesh Network?* en-US, Section: Computer Networks. [Online]. Available: <https://www.geeksforgeeks.org/computer-networks/what-is-mesh-network/>.
- [23] *Tailscale: How it works*. [Online]. Available: <https://tailscale.com/blog/how-tailscale-works>.
- [24] J. Heller and I. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 835–848, Oct. 1971, ISSN: 2162-2175. DOI: 10.1109/TCOM.1971.1090711.

- [25] G. Feygin and P. Gulak, "Survivor sequence memory management in Viterbi decoders," Jul. 1991, 2967–2970 vol.5. DOI: 10.1109/ISCAS.1991.176168.
- [26] *AUP PYNQ-Z2*, en. [Online]. Available: <https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html>.
- [27] Admin, *UVM Tutorial*, en-GB. [Online]. Available: <https://shorturl.at/5mxJY>.
- [28] G. Forney, "Convolutional codes I: Algebraic structure," *IEEE Transactions on Information Theory*, vol. 16, no. 6, pp. 720–738, Nov. 1970, ISSN: 1557-9654. DOI: 10.1109/TIT.1970.1054541.
- [29] J. MasseyY and M. Sain, "Inverses of Linear Sequential Circuits," *IEEE Transactions on Computers*, vol. C-17, no. 4, pp. 330–337, Apr. 1968, ISSN: 1557-9956. DOI: 10.1109/TC.1968.229392.