ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

ĐỒ ÁN 3
Triển khai Neural Network bằng High Level Synthesis trên nền tảng Xilinx Vitis

LÊ VĂN KIỀU QUÝ
quy.lvk193070@sis.hust.edu.vn

Ngành Kỹ thuật Điện tử - Viễn thông
Chuyên ngành Điện tử

Giảng viên hướng dẫn: TS. Võ Lê Cường

Khoa: Điện tử
Trường: Điện – Điện tử.

Hà Nội, tháng 3/2023.

LỜI MỞ ĐẦU

Hiểu được hình ảnh là một nhiệm vụ khó khăn đối với máy tính. Tuy nhiên các hệ thống thị giác máy tính tiên tiến hiện nay có khả năng phân loại hình ảnh, giám sát, phát hiện, bắt bám đối tượng đã dần trở nên quan trọng trong nhiều lĩnh vực Đã nhiều sự tiến bộ đáng kể liên quan tới việc triển khai các hệ thống thị giác máy tính, trong số đó Neural Network (NN) hiện đang là một các tiếp cận phổ biến và đạt dược nhiều thành công nhất.

Các thuật toán để triển khai NN bao gồm nhiều lớp xử lý. Nó có thể đạt được độ chính xác rất cao. Để đạt được độ chính xác như vậy nó phải có một hệ thống tính toán lớn và phức tạp. Các thiết bị tại biên bị hạn chế về năng lượng, tài nguyên phần cứng,…Các ứng dụng yêu cầu độ chính xác cao, khả năng xử lý dữ liệu thời gian thực như trong quân sự (máy bay không người lái, tên lửa,..), trong giao thông (ô tô tự động, giám sát phương tiện,..), y tế,.. đặt ra nhiều thách thức tới việc triển khai NN cho các thiết bị tại biên.

Các nền tảng phần cứng để triển khai mạng NN có thể đến như CPUs, GPUs, FPGA,..Khi so sánh với CPUs, FPGAs có ưu điểm về tốc độ tính toán. Khi so với GPUs, FPGAs có ưu điểm về việc sử dụng năng lượng. Ngoài ra các thiết kế sử dụng FPGAs có thể tập trung một cách tối ưu vào logic phần cứng. Với một số ưu điểm như vậy, FPGAs thích hợp làm nền tảng phần cứng để xử lý NN.

Triển khai các ứng dụng trên FPGA thông qua ngôn ngữ mô tả phần cứng – HDL. Tuy nhiên việc triển khai các ứng dụng yêu cầu khối lượng tính toán lớn như NN thì HDL có nhiều hạn chế. Những tiến bộ trong các công cụ tổng hợp mức cao – HLS ngày nay cho phép triển khai thuật toán bằng ngôn ngữ lập trình bậc cao trước khi biên dịch sang Register Transfer Level – RTL.

Đồ án III này trình bày về cách triển khai NN sử dụng tổng hợp ở mức cao, và đưa ra một thiết kế NN đơn giản cho nhận dạng chữ số viết tay.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC HÌNH VẼ

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
HLS	High Level Synthesis
NN	Neural Network
CPUs	Central Processing Unit
GPUs	Graphics Processing Units
FGPAs	Field Programmable Gate Arrays
ASICs	Application-Specific Integrated Circuits
LUT	Look – up table
VHLS	Vivado High Level Synthesis
DSP	Digital Signal Processor
RTL	Register Transfer Level
IP	Intellectual Property

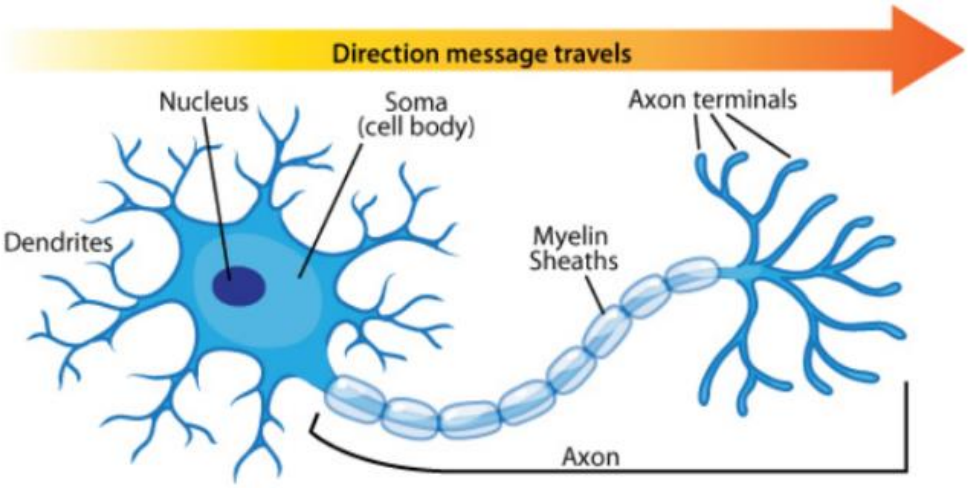
DANH MỤC BẢNG BIỂU

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Neural Networks

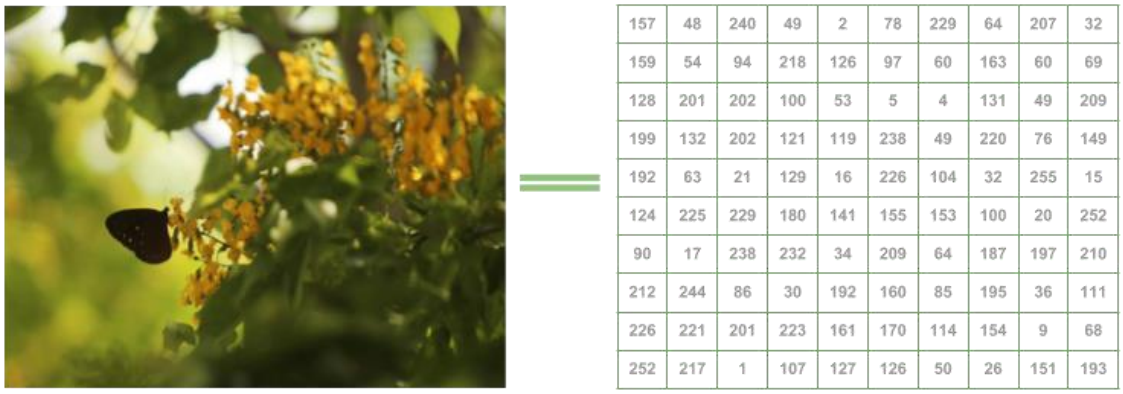
Neuron là đơn vị cơ bản cấu tạo nên hệ thống thần kinh của con người và là thành phần quan trong nhất của não bộ. Hình ảnh về một tế bào thần kinh được minh họa ở hình 1.1 bên dưới. Bộ não con người có khoảng 86 tỷ neurons, được kết nối bởi 10^14- 10^15 synapses. Mỗi neuron nhận tín hiệu đầu vào tại đuôi gai của nó và tạo ra các tín hiệu đầu ra dọc theo sợi trục, phân nhánh ra và kết nối với các sợ nhánh của neuron khác thông qua các synapses. Các synapses này ảnh hưởng đến việc truyền thông tin từ neuron này đến neuron khác bằng cách khuếch đại hoặc làm suy giảm tín hiệu điện, thậm chí là ức chế truyền tín hiệu ở các synapses khác.

Neural Networks lấy cảm hứng từ hệ thống thần kinh sinh học của con người và hoạt động của nó.



Hình 1.1 Tế bào thần kinh

Xét bài toàn phân loại hình ảnh: Dữ liệu đầu vào của bài toán là một bức ảnh. Một ảnh được biểu ảnh bằng ma trận các giá trị. Mô hình phân lớp sẽ phải dự đoán được lớp của ảnh từ ma trận điểm ảnh này, ví dụ như ảnh đó là con mèo, chó, hay là chim.



157	48	240	49	2	78	229	64	207	32
159	54	94	218	126	97	60	163	60	69
128	201	202	100	53	5	4	131	49	209
199	132	202	121	119	238	49	220	76	149
192	63	21	129	16	226	104	32	255	15
124	225	229	180	141	155	153	100	20	252
90	17	238	232	34	209	64	187	197	210
212	244	86	30	192	160	85	195	36	111
226	221	201	223	161	170	114	154	9	68
252	217	1	107	127	126	50	26	151	193

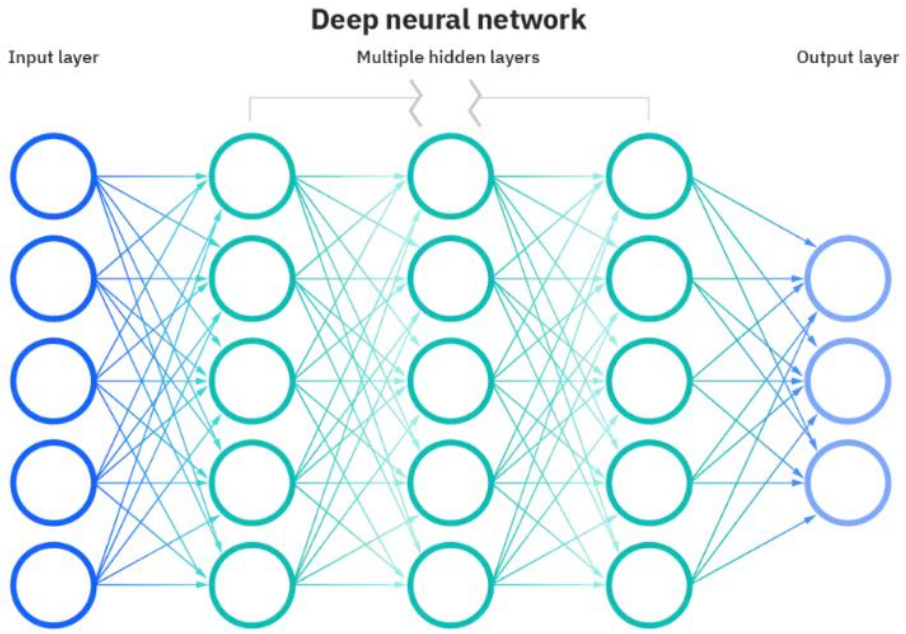
Chúng ta nhìn thấy

Máy tính thấy

Hình 1.2. Biểu diễn hình ảnh

Để biểu diễn một bức ảnh 256x256 pixel trong máy tính thì ta cần ma trận sẽ có kính thước 256x256 chiều, và tùy thuộc vào bức ảnh là có màu hay ảnh xám thì ma trận này sẽ có số kênh tương ứng, ví dụ với ảnh màu 256x256 RGB, chúng ta sẽ có ma trận 256x256x3 để biểu diễn ảnh này.

Kiến trúc của NN được minh họa ở hình 1.3 bên dưới bao gồm một tập các lớp cơ bản như: input layer, hidden layer, output layer.



Hình 1.3 Kiến trúc Neural Network

Input layer: Đây là lớp mà hình ảnh đầu vào sẽ được đưa vào mô hình NN, tùy thuộc vào yêu cầu, mà hình ảnh sẽ được định hình lại thành các kích thước khác nhau.

Tầng ẩn (hidden layer): Đây là tầng nằm ở giữa, thể hiện cho quá trình xử lý thông tin và suy luận của mạng. Nó sẽ nhận các thông tin đầu vào ở đầu vào và trả kết quả ở đầu ra thông qua chức năng kích hoạt.

Nonlinear Layer: Activation function: Được sử dụng để xác định đầu ra của mạng neural. Một số activation function phổ biến như Sigmoid, ReLU, Leaky ReLU, TanH và Softmax. ReLU (Rectified Linear Units, f = max(0, x)) hiện là hàm kích hoạt phổ biến nhất cho NN. Hàm ReLU được ưa chuộng vì tính toán đơn giản, giúp hạn chế tình trạng vanishing gradient, và cũng cho kết quả tốt hơn. ReLU cũng như những hàm kích hoạt khác, được đặt ngay sau tầng convolution, ReLU sẽ gán những giá trị âm bằng 0 và giữ nguyên giá trị của đầu vào khi lớn hơn 0.

Output Layer: Cuối cùng là lớp đầu ra bao gồm nhãn được mã hóa bằng cách sử dụng phương pháp one-hot encoding.

1.2 Các nền tảng phần cứng

Central Processing Units (CPUs) là lõi của bộ xử lý được tìm thấy trong hầu hết thiết bị điện tử ngày nay. Hầu hết các CPU này đều có mục đích chung, có thể lập trình linh hoạt và được chế tạo để có hiệu suất tốt trên phạm vi rộng. Có nhiều loại bộ xử lý khác nhau phù hợp với các hệ thống nhúng, với sự đánh đổi khác nhau về tốc độ và sức mạnh yêu cầu. Tuy nhiên, CPU tính toán kết quả tuần tự và do đó không lý tưởng phù hợp với vấn đề song song cao được đưa ra bởi các mạng thần kinh tích chập.

Digital Signal Processors (DSPs) là những bộ vi xử lý chuyên dụng cao. Chúng được tối ưu hóa để xử lý tín hiệu dấu phẩy động nhanh và hiệu quả . Tuy nhiên, DSP chủ yếu vẫn là bộ xử lý "ít lõi" được tối ưu hóa để có tốc độ xử lý nhanh. hoạt động tuần tự và do đó không thể khai thác triệt để tính song song có trong NN.

Graphics Processing Units (GPUs) là bộ xử lý nhiều lõi ban đầu được thiết kế cho khối lượng công việc đồ họa song song cao. GPU rất phù hợp cho khối lượng công việc song song do NN trình bày và được hỗ trợ đầy đủ bởi hầu hết học sâu khuôn khổ. Chúng tạo thành nền tảng chính cho nghiên cứu trong lĩnh vực NN. Tuy nhiên GPUs tiêu tốn rất nhiều năng lượng, điều này là không phù hợp với các thiết bị nhúng xử lý dữ liệu tại biên.

Field-Programmable Gate Arrays (FGPAs) với các thiết kế trên FPGA hoạt động tốt nhất cho các tính toán rất thường xuyên có thể bị phân biệt nặng nề bằng cách xây dựng các công cụ xử lý tùy chỉnh bằng cách sử dụng các khối logic lập trình được. Các thuật toán yêu cầu phân nhánh và quyết định phụ thuộc vào dữ liệu ít phù hợp hơn cho loại song song hóa này và dẫn đến việc sử dụng kém sức mạnh tính toán. Hiệu suất của các thiết kế FPGA có thể được tăng thêm bằng cách sử dụng điểm cố định hoặc định dạng dữ liệu dấu chấm động nửa chính xác. FPGA có thể tiết kiệm năng lượng hơn và có thể mở rộng so với GPU, trong khi vẫn duy trì mức độ linh hoạt hợp lý.

Application-Specific Integrated Circuits (ASICs) là giải pháp lý tưởng khi nói đến hiệu suất tối đa và hiệu quả năng lượng tối đa. Tuy nhiên, ASIC thậm chí còn ít phù hợp với tính toán bất thường hơn so với FPGA và chúng còn yêu cầu nhiều thuật toán bị đóng băng tại thời điểm thiết kế. Vì lý do này, ASIC thường chỉ được chế tạo để tăng tốc một khía cạnh nhất định của NN, chẳng hạn như tính toán một phần của phép tích chập hoặc lớp được kết nối đầy đủ, nhưng hiếm khi tính toán toàn bộ mạng lưới thần kinh. Một nổi bật ngoại lệ là các mạch tích hợp thần kinh, sử dụng các mạch điện tử tương tự để bắt chước các nơ-ron và mạng nơ-ron trên các vi mạch được thiết kế tùy chỉnh.

1.3 Field-Programmable Gate Arrays – FPGA.

Field-Programmable Gate Arrays – FPGA là một thiết bị bán dẫn có thể được lập trình để thực thi các thuật toán khác nhau sau khi chế tạo. Các thiết bị FPGA hiện đại bao gồm tới hai triệu ô logic có thể cấu hình để thực hiện một

loạt các thuật toán phần mềm. Do các khối có thể lập trình khác nhau, người dùng có thể cấu hình lại chip mà không phải thiết kế chip từ đầu.

Các thành phần cơ bản của FPGA bao gồm:

- Look – up table (LUT): Phần tử này thực hiện các phép toán logic. Trên thiết bị Xilinx FPGA. Khối tính toán phức tạp nhất có sẵn trong FPGA Xilinx là khối DSP. Khối DSP là một đơn vị logic số học (ALU) được nhúng vào kết cấu của FPGA.
- Flip – Flop (FF): Phần tử thanh ghi, dùng lưu trữ kết quả của LUT. Flip – flop là đơn vị lưu trữ cơ bản trong kết cấu FPGA. Phần tử này luôn được ghép nối với một LUT để hỗ trợ đường ống logic và lưu trữ dữ liệu.
- Dây: Dùng để kết nối các phần tử với nhau.
- Cổng vào ra: Các cổng vật lý sẵn có này nhận dữ liệu vào và ra của FPGA: như UART, PS2 Keyboard, MS2 Mouse, HDMI, VGA,..

Ngoài các thành phần cơ bản, kiến trúc FPGA hiện nay còn kết hợp với các khối bổ sung chức năng tính toán, lưu trữ. Điều này làm tăng mật độ tính toán và hiệu quả của thiết bị, bao gồm:

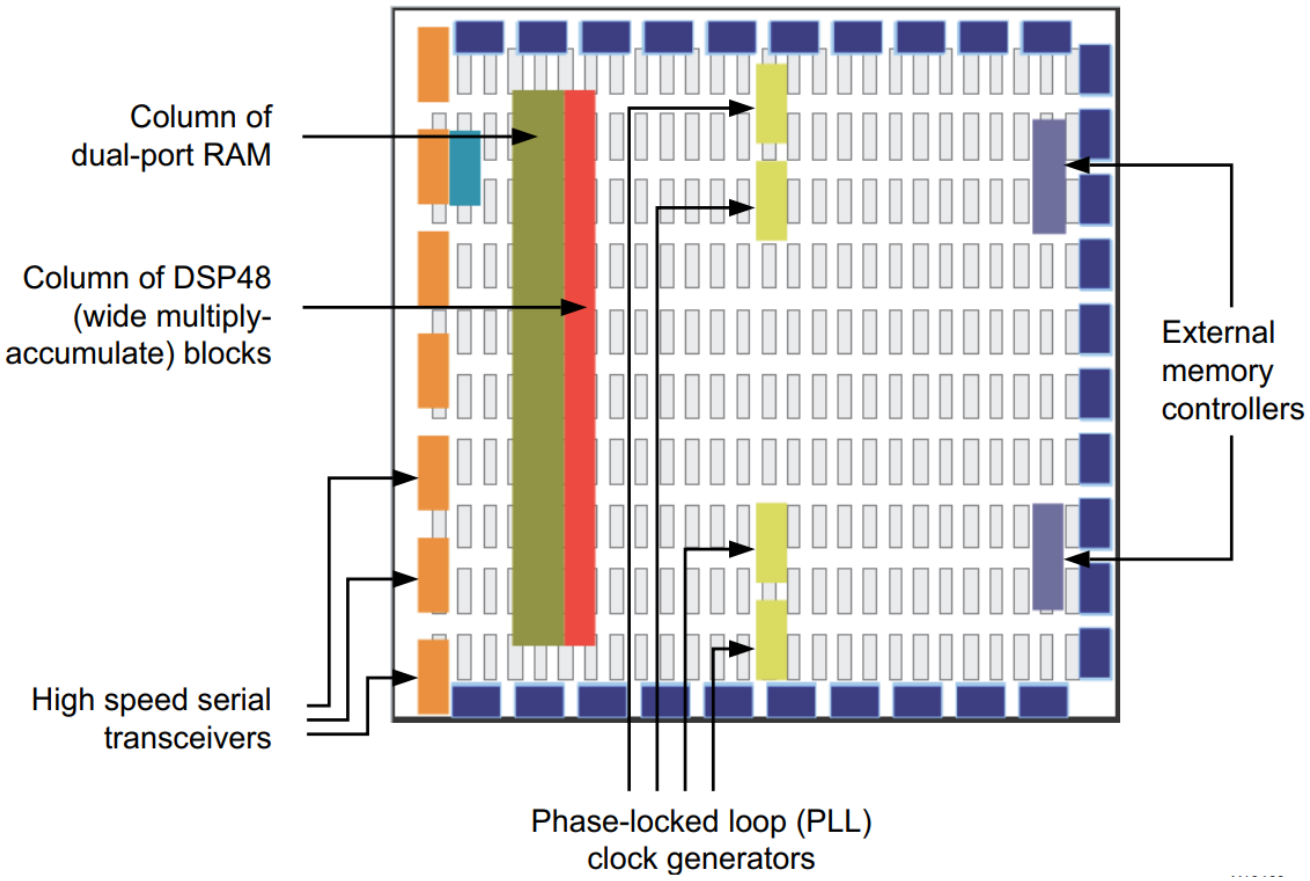
- Bộ nhớ nhúng: Lưu trữ dữ liệu phân tán có thể kể đến như RAM, ROM. BRAMs, UltraRAM,..
- PLL: Điều khiển cấu trúc FPGA ở các tốc độ xung nhịp khác nhau.
- Bộ thu phát nối tiếp tốc độ cao.
- Bộ điều khiển bộ nhớ ngoài chip
- Các khối nhân tích lũy: Multiply-accumulate blocks.

1.4 High Level Synthesis

1.4.1. Giới thiệu chung về HLS

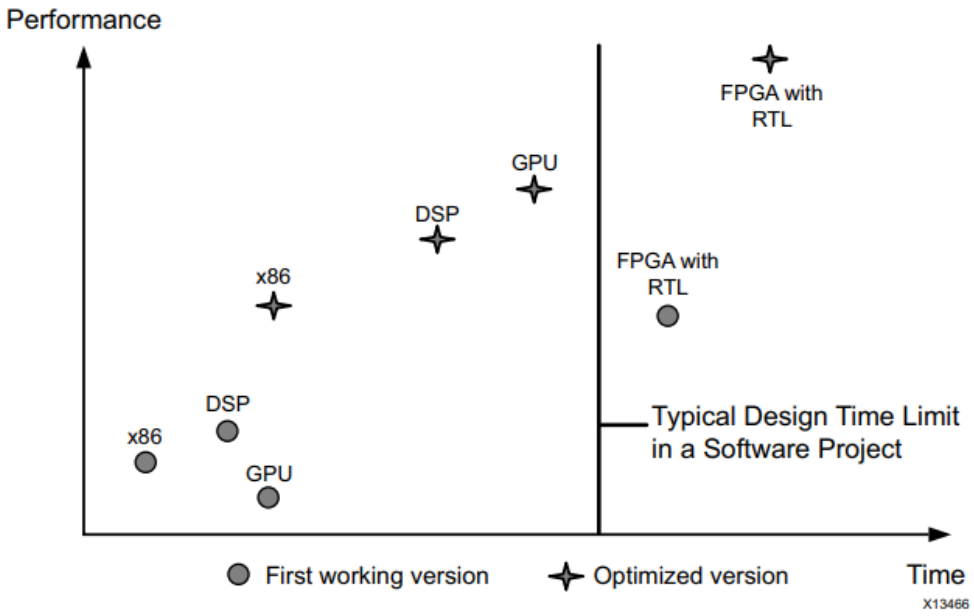
Các thuật toán triển khai trên FPGA được mô hình bằng ngôn ngữ mô tả phần cứng như Verilog HDL hay VHDL ở mức RTL. Hình 1.6 bên dưới minh họa sự khác biệt về ảnh hưởng mô hình lập trình tới thời gian thực hiện và hiệu suất đạt được cho các nền tảng tính toán khác nhau. Triển khai thiết

kế ở mức RTL và ứng dụng được tối ưu hóa cho FPGA mang lại hiệu suất cao nhất. Tuy nhiên, thời gian phát triển để đạt được hiệu quả bằng cách triển khai này vượt quá phạm vi về thời gian phát triển phần mềm điển hình. Do đó, FPGA được sử dụng theo truyền thống chỉ dành cho những



Hình 1.4 Kiến trúc FPGA

ứng dụng yêu cầu cấu hình hiệu suất không thể đạt được bằng bất kỳ phương tiện nào khác, chẳng hạn như thiết kế có nhiều bộ xử lý. Hầu hết các thiết kế được triển khai ở RTL, các thuật toán được triển khai bằng vô số các quy trình được hoạt động song song trên trường dữ liệu nhị phân. Các tín hiệu thay đổi dựa trên sự thay đổi của xung đồng hồ. Cách mô tả RTL rất gần với việc mô tả các cổng logic và dây dẫn. Khi ấy việc tổng hợp thành phần cứng được kiểm soát chặt chẽ.



Hình 1.5. Ảnh hưởng mô hình lập trình

Tuy nhiên với những ứng dụng có thuật toán phức tạp hơn, cần phải chia nhỏ thuật toán để triển khai thành các khối logic, các máy trạng thái hữu hạn – FSM,.. Khi ấy việc thiết kế và triển khai lập trình và nâng cấp thiết kế gặp nhiều khó khăn và tốn kém. Việc triển khai như vậy đòi hỏi người lập trình phải có kiến thức chuyên môn tốt.

Để tăng mức độ trừ tượng của thiết kế, các công cụ tổng hợp mức cao – HLS cho phép triển khai thuật toán bằng ngôn ngữ lập trình bậc cao như C/C++ hay SystemC. Trình biên dịch HLS có thể xử lý, chuyển đổi các mô tả phần mềm tuần tự thành mô tả phần cứng được thực thi song song, thường ở mức RTL. Trình biên dịch HLS phổ biến nhất là Vivado High-Level Synthesis Vivado High Level Synthesis (VHLS) của Xilinx Inc. Với VHLS, có thể sử dụng vòng lặp, mảng, cấu trúc, float, hầu hết các phép tính số học, lệnh gọi hàm và thậm chí cả các lớp hướng đối tượng. Các lệnh này được tự động chuyển đổi thành bộ đếm, bộ nhớ, lõi tính toán và các giao thức cũng như các máy trạng thái và lập lịch đi kèm.

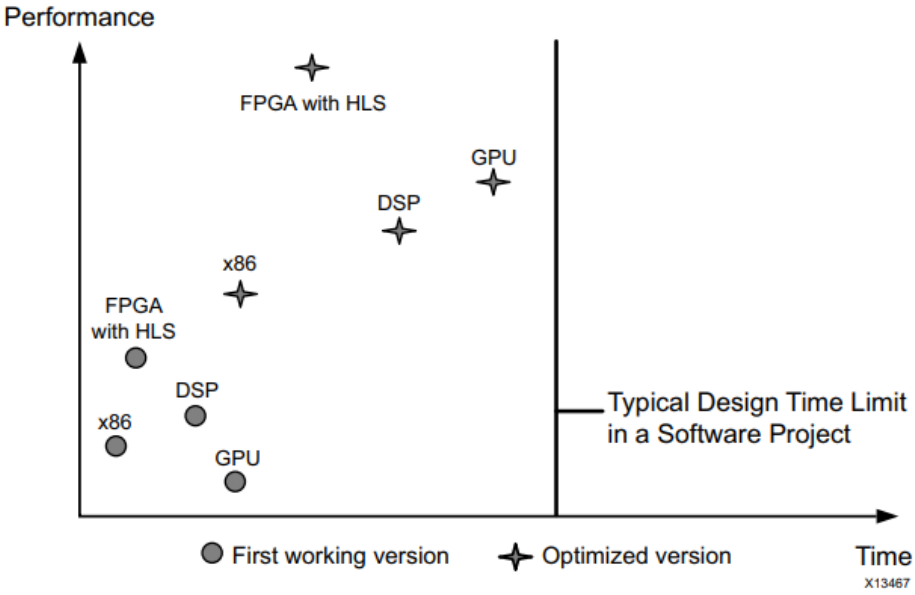
Hình 1.7 bên dưới so sánh kết quả của trình biên dịch Vivado HLS với các bộ xử lý khác nhau cho cùng một giải pháp phần mềm có sẵn.

1.4.2. Trình biên dịch của HLS.

Công cụ Vivado HLS giúp chuyển thiết kế từ ngôn ngữ lập trình C/C++ thành mã RTL. Mỗi liên hệ được minh hoạt thông qua bảng 1.1 bên dưới, thông qua chỉ chị #pragma được chèn vào đoạn mã C/C++.

Bảng 1. Triển khai từ HLS sang phần cứng

Nội dung	C/C++	RTL code
Interfaces	C/C++ function	Các function/ module.
	Đối số của hàm	input/output của module/function.
Luồng dữ liệu	Các lệnh/dữ liệu được thực thi tuần tự #pragma HLS DATAFLOW	Vivado HLS phân tích phần tử dữ liệu nào được tạo ra, sử dụng trong các process riêng biệt. ⇨ Thêm vào các kênh (double-buffer/pingpong RAMs or FIFOs) giữa các hàm, consumer loops, producer ⇨ Dữ liệu được sử dụng ngay khi chúng sẵn sàng. ⇨ Tận dụng khả năng song song hóa. VD: Load các Image Pixel Buffer để thực hiện phép tích chập
Array	Array (chỉ mảng tĩnh, kích thước cố định) #pragma HLS RESOURCE	Loại bộ nhớ (RAM, ROM, FIFO), kiểu triển khai(Block RAM, Thanh ghi dịch)



Hình 1.6. Hiệu quả của trình biên dịch Vivado HLS.

CHƯƠNG 2. TRIỂN KHAI THIẾT KẾ

2.1. Luồng thiết kế

Luồng thiết kế ở mức HLS cho NN được minh họa như hình bên dưới

Bước 1: Chọn kiến trúc NN:

Kiến trúc NN có thể được lựa chọn từ các tài nguyên có sẵn như TensorFlow, Caffe, Keras.

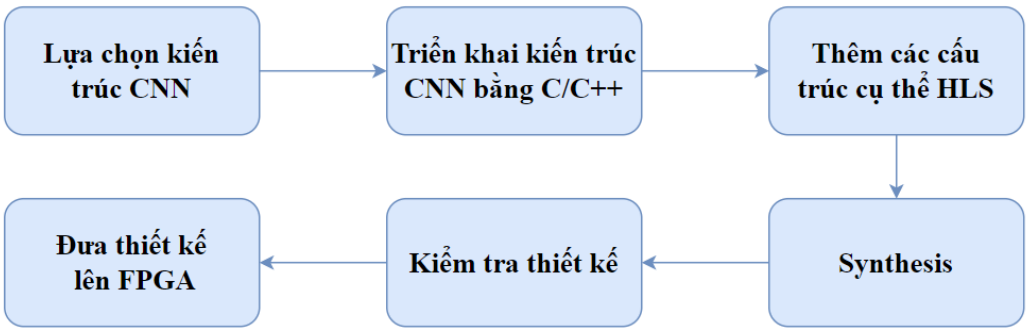
Các ngôn ngữ sử dụng có tính trừ tượng cao như Matlab, Python.

Bước 2: Triển khai kiến trúc NN bằng C/C++

Khi lựa chọn được kiến trúc mạng phù hợp với yêu cầu thiết kế phần cứng, ta triển khai thuật toán bằng ngôn ngữ C/++.

Bước 3. Thêm các cấu trúc cụ thể cho HLS

Như đã phân tích ở trên, HLS đi kèm với một số chỉ thị #param cho phép người dùng hướng dẫn trình biên dịch tạo ra các thiết kế mong muốn



Hình 2.1 Luồng thiết kế

Bước 4: Tổng hợp.

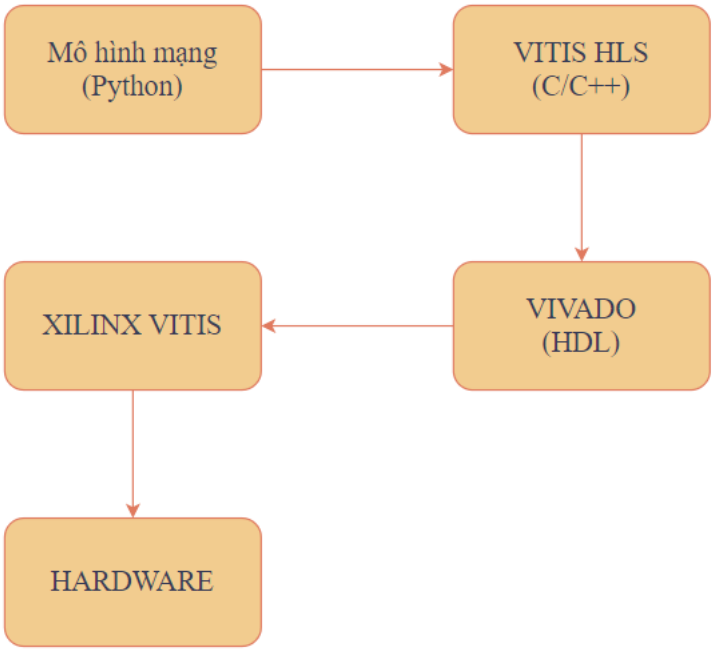
Mã nguồn ở bước 3 được biên dịch chéo từ ngôn ngữ C/C++ thành ngôn ngữ mô tả phần cứng Verilog/ VHDL.

Bước 5. Xác minh thiết kế.

Xây dựng các testbench để kiểm tra các dữ liệu đầu vào

Bước 6. Đưa thiết kế lên kit FPGA.

Triển khai thiết kế thông quan nền tảng Xilinx Vitis



Hình 2.2. Luồng dữ liệu

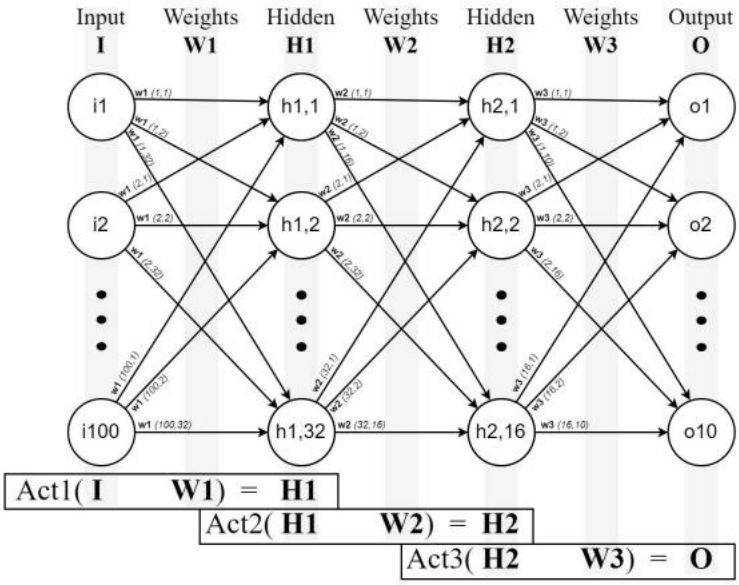
Mô hình mạng, thông qua các công cụ đào tạo, weight được sử dụng cho VITIS HLS, ở đây mô hình mạng được tổng hợp thành các Vivado IP core

Vivado IP được dùng ở công cụ VIVADO để tổng hợp sử dụng các nền tảng phần cứng thông qua XILINX VITIS để có thể thực thi thiết kế trên nền tảng FPGA.

2.2. Mô hình Neural Network.

Mô hình mạng được triển khai là một mô hình đa lớp. Mạng có 100 nodes ở input layers, 32 neurons trong hidden layer đầu tiền, 16 neurons ở hidden layer thứ 2, và 10 đầu ra ở output layers. Cụ thể mô hình được minh họa ở hình 2.3 bên dưới.

Các phép toán để thực hiện trong mô hình mạng là các phép nhân ma trận và các kích hoạt phi tuyến. Hai hàm kích hoạt đầu tiên đều là ReLU. Tổng số weights được đào tạo là $100 . 32 + 32 . 16 + 16 . 10 = 3872$ giá trị.



Hình 2.3 Mô hình Neural Network

CHƯƠNG 3. TRIỂN KHAI NEURAL NETWORK TRÊN VITIS HLS

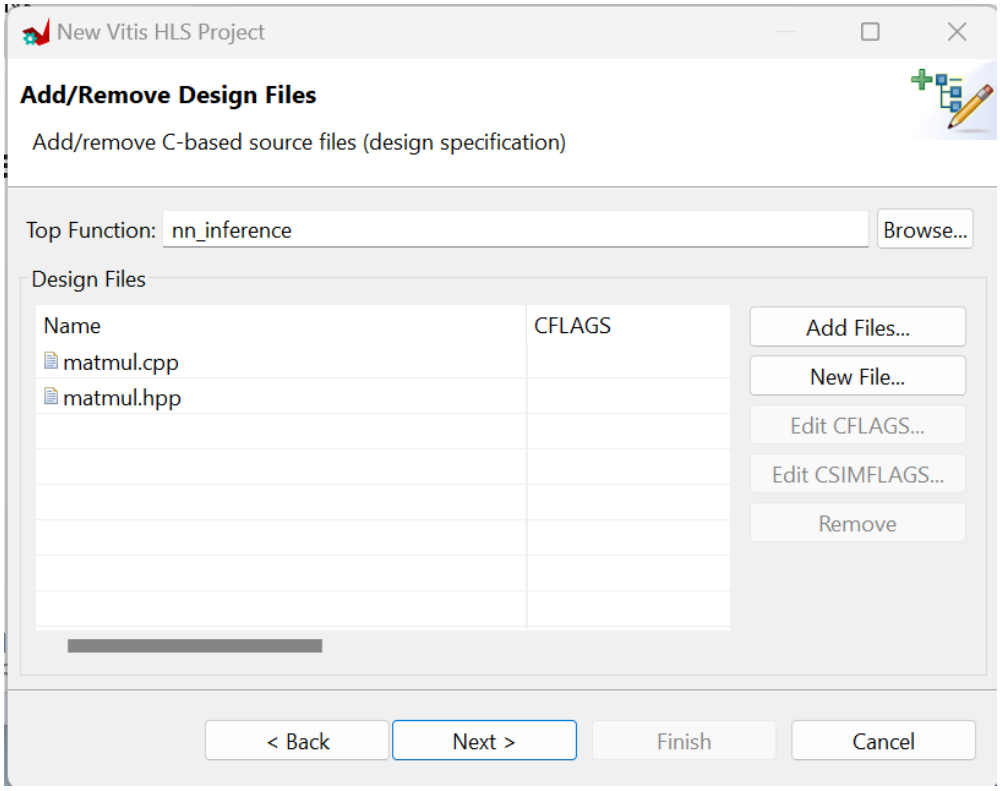
Từ thiết kế của hệ thống, chúng ta cần triển khai các module con. Cùng với đó là xác minh thiết kế thông qua testbench. Cụ thể công cụ được sử dụng ở đây là VITIS HLS của hãng Xilinx

3. 1. Giới thiệu chung.

Xilinx cung cấp cho người dùng phần mềm VITIS HLS để thực hiện việc tổng hợp thiết kế, biên dịch chéo từ ngôn ngữ C/C++ thành ngôn ngữ mô tả phần cứng HDL như Verilog, VHDL hoặc SystemC.

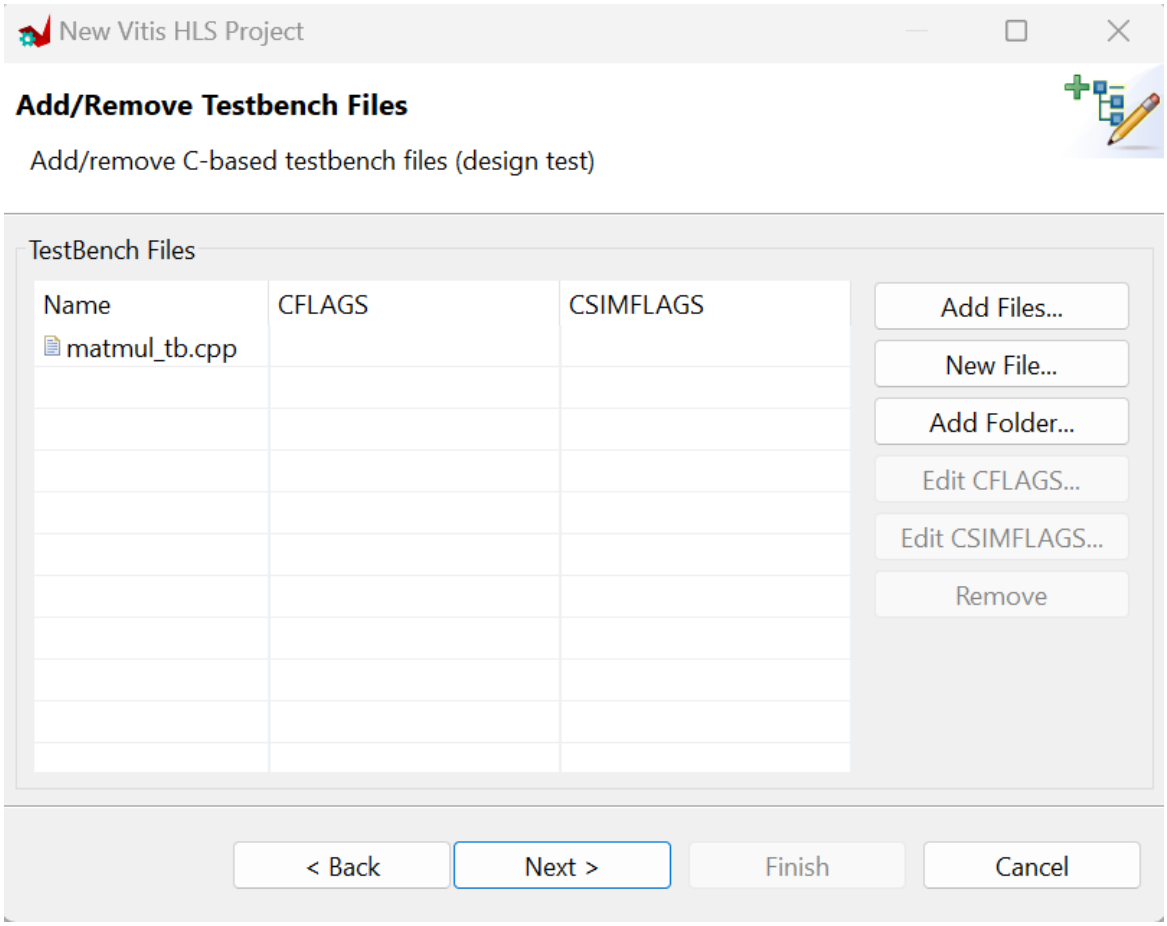
Trước hết chúng ta cần đưa thiết kế lên phần mềm. Cần thêm các design file, testbench file, và lựa chọn nền tảng phần cứng để triển khai. Ở đây, nền tảng phần cứng được sử dụng là *ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1)*

Hình 3.1 bên dưới minh họa việc đưa các file thiết kế lên phần mềm.



Hình 3.1. Thêm các design file

Tiếp đến là thêm cái file testbench. Hình 3.2 bên dưới minh họa việc đưa file testbench lên phần mềm để xác minh thiết kế.

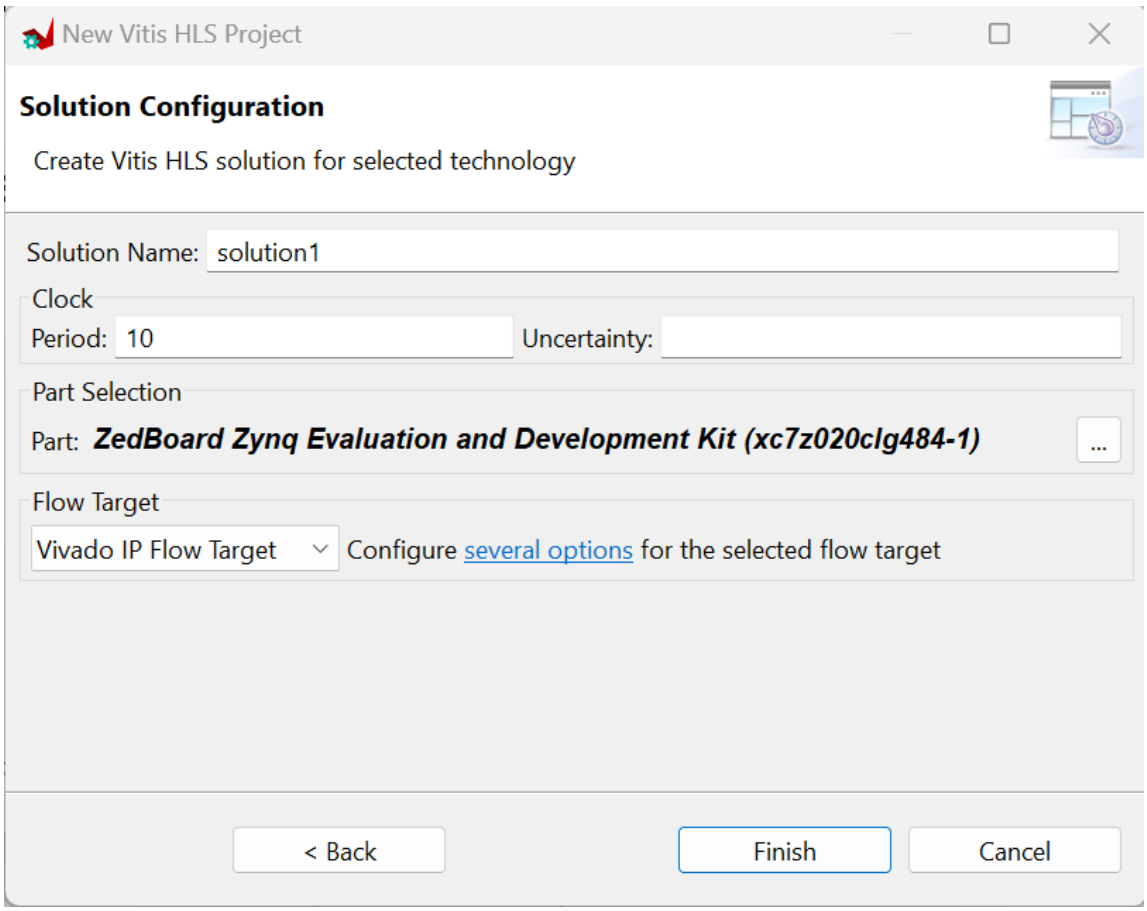


Hình 3.2. Thêm file testbench

Hình 3.3 bên dưới minh họa việc lựa chọn nền tảng phần cứng
Đảm bảm tính chính xác của thiết kế trước khi synthesis, ta cần phải viết chạy file testbench để mô phỏng thiết kế.

Sau khi thiết kế được bảm bảo chính xác, bước kế tiếp theo là synthesis thiết kế. Kết quả là file HDL: Verilog và VHDL.

Xuất các mã RTL trên thành các IP core được sử dụng cho VIVADO ở chương tiếp theo.



Hình 3.3. Lựa chọn nền tảng phần cứng.

3.2. Triển khai các layer

Hàm nhân ma trận lớp thứ nhất được mô tả ở bảng 2

Bảng 2. Mô tả tham số hàm *hwmm_layer1*

Đối số	Mô tả	Kích thước
Input	Đầu vào hàm *hwmm_layer1*	[100]
Weight	Trọng số cho input layer và hidden layer1	[100][32]
Output	Đầu ra hàm *hwmm_layer1*	[1][32]

Hàm nhân ma trận lớp thứ hai được mô tả ở bảng 3

Hàm nhân ma trận lớp thứ ba được mô tả ở bảng 4

Hàm nhân ma trận lớp thứ nhất được mô tả ở bảng 5

Bảng 3. Mô tả tham số hàm *hwmm_layer2*

Đối số	Mô tả	Kích thước
Input	Đầu vào hàm *hwmm_layer2*	[1][32]
Weight	Trọng số cho hidden layer 1 và hidden layer 2	[32][16]
Output	Đầu ra hàm *hwmm_layer2*	[1][16]

Hàm nhân ma trận lớp ba

Bảng 4. Mô tả tham số hàm *hwmm_layer3*

Đối số	Mô tả	Kích thước
Input	Đầu vào hàm *hwmm_layer3*	[1][16]
Weight	Trọng số cho hidden layer 2 và output layer	[16][10]
Output	Đầu ra hàm *hwmm_layer3*	[1][10]

Hàm kích hoạt ReLU lớp thứ nhất

Bảng 5. Mô tả tham số hàm *hw_act_layer1*

Đối số	Vai trò	Kích thước
Input	Đầu vào hàm *hw_act_layer1*	[1][100]
Output	Đầu ra hàm *hw_act_layer1*	[1][100]

Hàm kích hoạt ReLU lớp thứ hai.

Bảng 6. Mô tả tham số hàm hw_act_layer2

Tham số	Vai trò	Kích thước
Input	Đầu vào hàm *hw_act_layer2*	[1][16]
Output	Đầu ra hàm *hw_act_layer2*	[1][16]

Hàm kích hoạt Softmax lớp thứ ba.

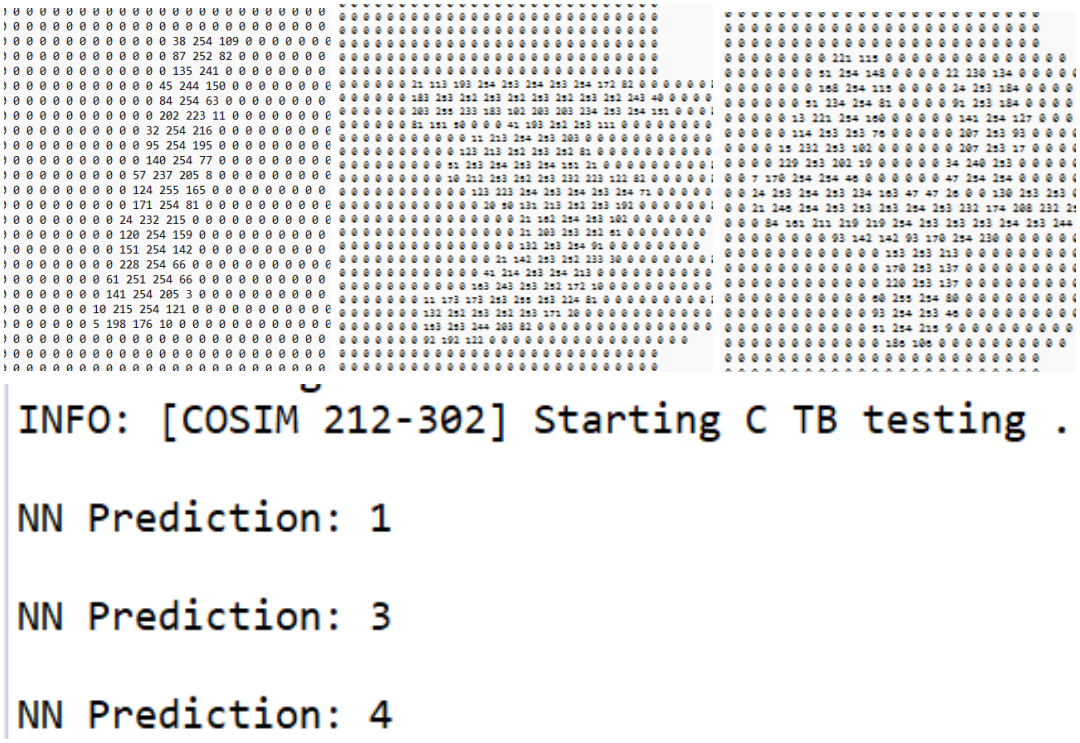
Bảng 7. Mô tả tham số hàm hw_act_layer3

Tham số	Vai trò	Kích thước
Input	Đầu vào hàm *hw_act_layer3*	[1][10]
Pred	Đầu ra hàm *hw_act_layer3*	Int

3.3. Kiểm tra thiết kế

C/RTL CoSimulation

Xây dựng testbench kiểm tra thiết kế. Đầu vào lớp đầu tiên là mảng 1 chiều mang dữ liệu của ảnh được. Kết quả được minh họa như hình 3.5 bên dưới.



Hình 3.5. Dữ liệu đầu vào ra kết quả đầu ra.

3.4. Đánh giá, tối ưu thiết kế

Các tham số đánh giá thiết kế:

- Tài nguyên phần cứng chiếm dụng
- Latency của mạch. Latency là trễ lan truyền dữ liệu trong mạch, được tính từ thời điểm bắt đầu của dữ liệu đầu vào tới khi có dữ liệu đầu ra (theo số cycle hoặc theo đơn vị thời gian chuẩn).

- Xem xét các vi phạm xảy ra: Slack time,..

3.4.1. Đánh giá thiết kế khi chưa tối ưu.

Giữ nguyên thiết kế khi triển khai bằng C/C++, chưa bổ sung các chỉ thị #pragma

Kết quả như hình 3.6 bên dưới:

Latency của mạch là 19658 cycles, với tần số 10ns hay 100MHz thì latency tuyệt đối là 0,197ms

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
19658	19658	0.197 ms	0.197 ms	19659	19659	none

Detail

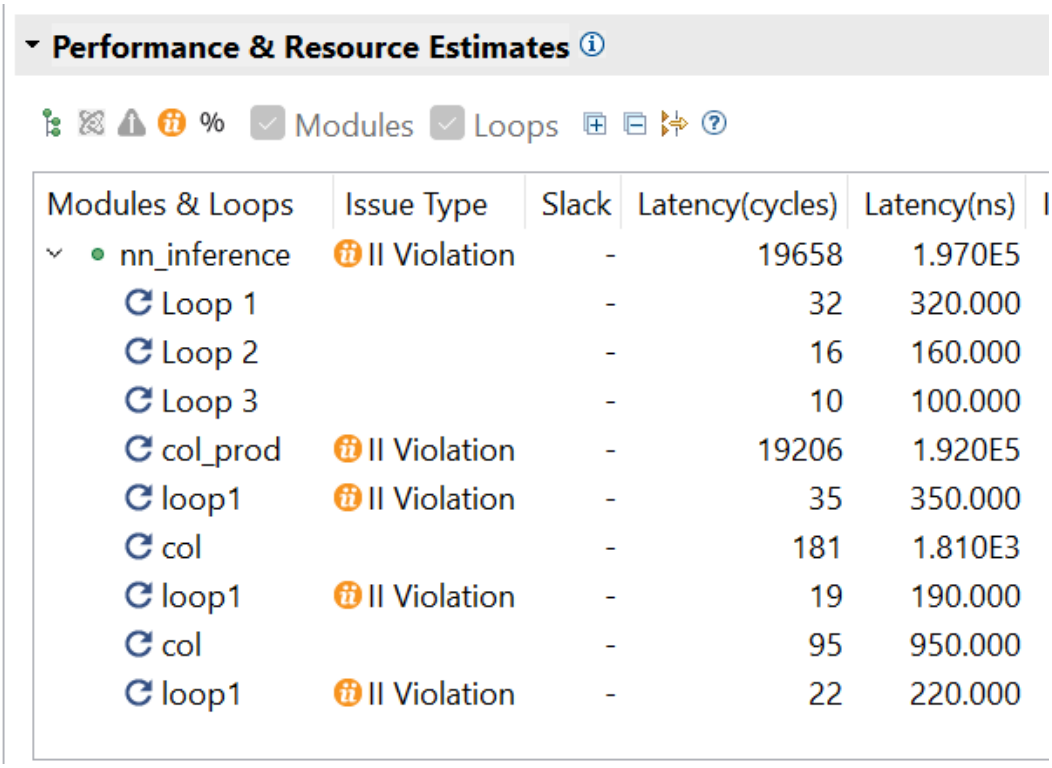
Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	516	-
FIFO	-	-	-	-	-
Instance	-	160	11136	22752	-
Memory	10	-	1664	349	-
Multiplexer	-	-	-	1428	-
Register	-	-	6197	192	-
Total	10	160	18997	25237	0
Available	280	220	106400	53200	0
Utilization (%)	3	72	17	47	0

Hình 3.6 Tài nguyên mà Neural Network chiếm dụng khi chưa tối ưu.

Hình 3.7 bên dưới cho thấy thiết kế đang bị vi phạm.



Hình 3.7 II Violation của thiết kế chưa tối ưu

3.4.2. Tối ưu thiết kế.

#pragma HLS UNROLL

Unroll loops để tạo nhiều toán tử/hoạt động độc lập thay vì một bộ/khối toán tử/hoạt động. pragma biến đổi các vòng lặp bằng cách tạo ra nhiều bản sau của loop body trong thiết kế RTL. Cho phép một hoặc tất cả các lần lặp vòng lặp này được thực thi song song. Khác thác tối đa tính song song của thiết kế khi triển khai trên kit FPGA.

Từ lỗi II Violation, ta bổ sung thêm các #pragma HLS UNROLL vào sau vòng lặp của nơi xảy ra lỗi. Kết quả như hình 3.8 và 3.9 bên dưới, II Violation đã được khắc phục. Latency của thiết kế giảm gần 5 lần (từ 19658 cycles xuống 4016 cycles). Tuy nhiên, tài nguyên phần cứng mà thiết kế chiếm dụng vượt quá tài nguyên của kit FPGA đã chọn. Cụ thể thiết kế cần 58499 khối LUT vượt quá 109% so với tài nguyên của phần cứng.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
4016	4016	40.160 us	40.160 us	4017	4017	none

Detail

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2012	-
FIFO	-	-	-	-	-
Instance	32	160	30241	54269	-
Memory	2	-	1664	349	-
Multiplexer	-	-	-	1741	-
Register	-	-	6368	128	-
Total	34	160	38273	58499	0
Available	280	220	106400	53200	0
Utilization (%)	12	72	35	109	0

Hình 3.8a Kết quả của thiết kế sau khi Unroll một số vòng lặp nhất định.

Performance & Resource Estimates

% Modules Loops

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)
nn_inference		-	4016	4.016E4
hwmm_layer1		-	3584	3.584E4
hw_act_layer1		-	31	310.000
Loop 1		-	32	320.000
Loop 2		-	16	160.000
Loop 3		-	10	100.000
col		-	181	1.810E3
col		-	95	950.000

Hình 3.8b Kết quả của thiết kế sau khi Unroll một số vòng lặp nhất định.

Unroll loops các vòng lặp trong thiết kế.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
906	906	9.309 us	9.309 us	907	907	none

Detail

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1937	-
FIFO	-	-	-	-	-
Instance	-	80	15598	59073	-
Memory	2	-	64	8	-
Multiplexer	-	-	-	2049	-
Register	-	-	1593	-	-
Total	2	80	17255	63067	0
Available	280	220	106400	53200	0
Utilization (%)	~0	36	16	118	0

Hình 3.9a Kết quả của thiết kế sau khi Unroll loops các vòng lặp

Performance & Resource Estimates

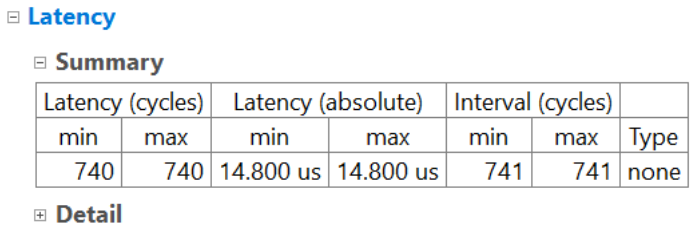
% Modules Loops

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)
nn_inference		-2.97	906	9.309E3
hwmm_layer1		-2.97	521	5.353E3
hwmm_layer2		-1.05	173	1.730E3
hwmm_layer3		-0.91	85	850.000
hw_act_layer1		-	31	310.000
Loop 1		-	32	329.000
Loop 2		-	16	164.000

Hình 3.9b Kết quả của thiết kế sau khi Unroll loops các vòng lặp

Sau khi Unroll tất cả các vòng lặp trong thiết kế, số phần tử logic tiêu thụ tăng lên. Tuy nhiên, Latency của mạch tiếp tục giảm đi khoảng 4.4 lần

Thiết kế bị vi phạm về Slack. Để giải quyết vấn đề này, ta giảm tần số hoạt động đi 2 lần, chu kì là 20ns thay vì 10ns như thiết kế trước. Và vấn đề đã được giải quyết. Kết quả như hình 3.10 bên dưới


Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
740	740	14.800 us	14.800 us	741	741	none

Detail

Hình 3.10 Latency của thiết kế hoạt động ở 50MHz

Tóm lại ta có bảng đánh giá sau

Bảng 8. So sánh các thiết kế

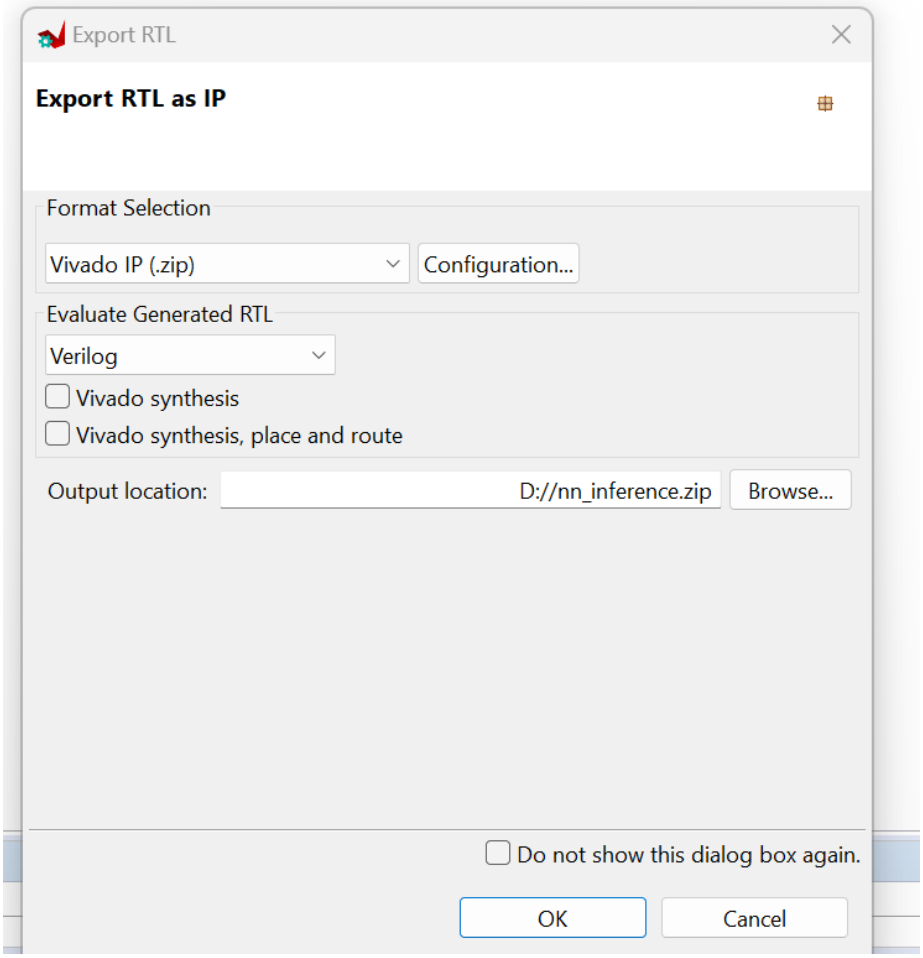
Thiết kế	Mô tả	Latency	Khác
1	Không Unroll loop	0.197ms (100MHz)	Không vi phạm tài nguyên Vi phạm timing
2	Unroll 1 phần	40.16us (100MHz)	Vi phạm tài nguyên Không vi phạm timing
3	Unroll toàn bộ	9.309us (100MHz)	Vi phạm timing Vi phạm tài nguyên
4	Unroll loop toàn bộ	14.8us (50MHz)	Vi phạm tài nguyên Không vi phạm timing

3.5. Đóng gói thiết kế

Trong thiết kế IC, FPGA hiện đại, ngày càng có nhiều chức năng hệ thống được tích hợp và các chip đơn, kiểu thiết kế này được gọi là thiết kế hệ thống trên chip – SoC. Hầu hết các chip SoC kết hợp với bộ xử lý tiêu chuẩn và các chức năng được tiêu chuẩn hóa, có khả năng tái sử dụng thiết kế trên nhiều IC bởi nhiều nhà cung cấp. Trong các hệ thống này, lõi IP được thiết kế và đóng gói sẵn đóng một vai trò quan trọng. IP core là một khối chức

năng logic hoặc dữ liệu được sử dụng trên FPGA, là một module độc lập, có thể là một phần của thiết bị hoặc hệ thống lớn hơn. IP core là một đơn vị có thể tái sử dụng.

Để linh hoạt việc sử dụng các thiết kế đã triển khai, ta tiến hành đóng gói các mã thiết kế thành các IP core. Minh họa như hình 3.6. Ngôn ngữ sử dụng là Verilog.



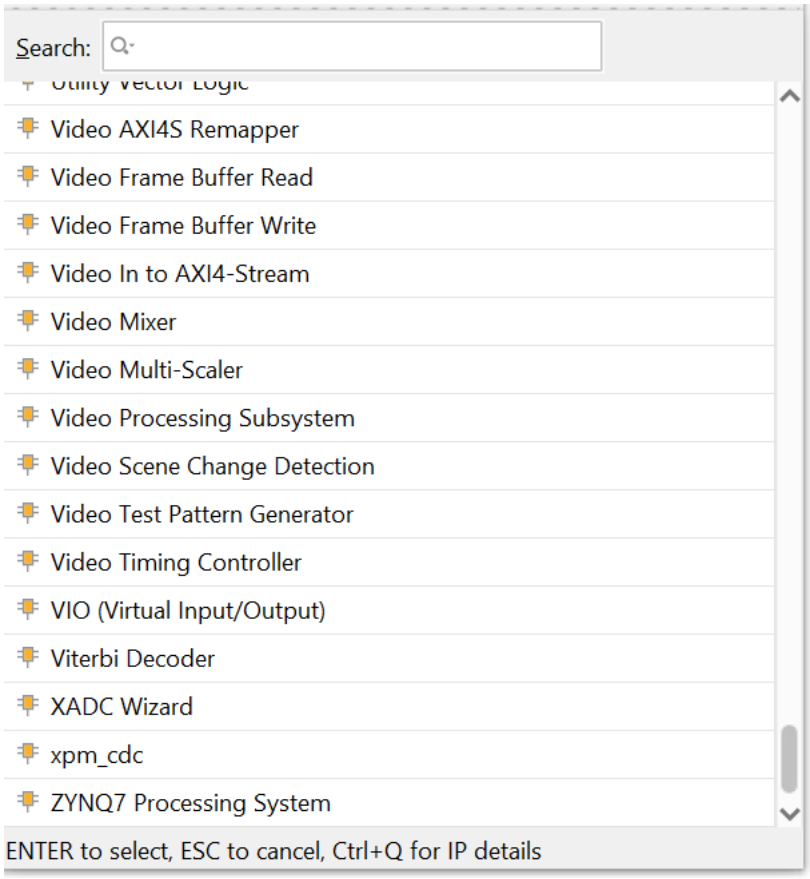
Hình 3.11 Đóng gói thiết kế thành IP core

CHƯƠNG 4. TRIỂN KHAI IP CORE TRÊN VIVADO

Chương 3 đã đề cập với việc tạo ra các IP core thông qua công cụ VITIS HLS. Chương 4 này sẽ đề cập với việc đưa các IP core này để triển khai trên nền tảng phần cứng

4.1. Triển khai IP trên phần mềm VIVADO

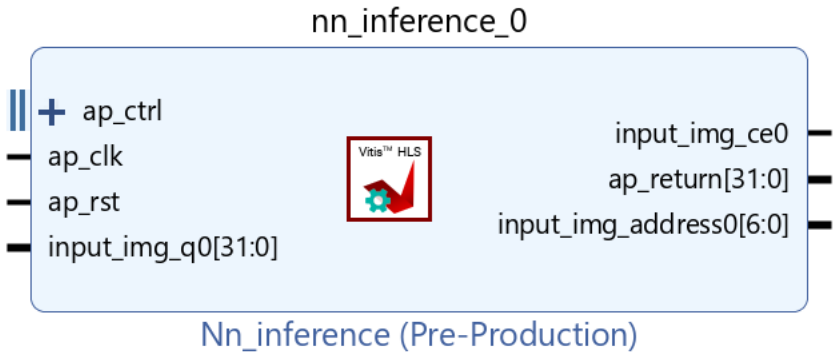
Công cụ VIVADO cung cấp các IP có sẵn để tăng tốc độ phát triển thiết kế, các IP core này có thể cấu hình linh hoạt theo yêu cầu để phù hợp với thiết kế. Hình 4.1 bên dưới là một số IP core được hỗ trợ



Hình 4.1. IP core trên VIVADO

Những IP core này được sử dụng để triển khai thiết kế trên nền tảng phần cứng bằng công vụ VIVADO được sử dụng để triển thiết kế trên nền tảng phần cứng từ những module IP được tạo ra,Mặc định HLS sẽ tạo ra một cổng ap_ctrl_hls khi tổng hợp một số module và các phần cứng xung quang sẽ

tương thích với giao thức này. Hình 4.2 bên dưới là IP được tổng hợp từ chương 3.



Hình 4.2. Nn_inference

ZYNQ7 Processing System IP ở hình 4.2. đại điện cho thành phần không phải của kit FPGA của chip Zynq. Nó phải được sử dụng trong một block design để kết nối với bất kì thứ gì với bộ xử lý để cấu hình cho các thiết bị ngoại vi phía Processing System, clk,..

AXI BRAM Controller là IP core để tích hợp với các thiết bị kết nối và hệ thống chính của hệ thống AXI để giao tiếp với local block RAM. Interface của khối như hình 4.3 bên dưới.

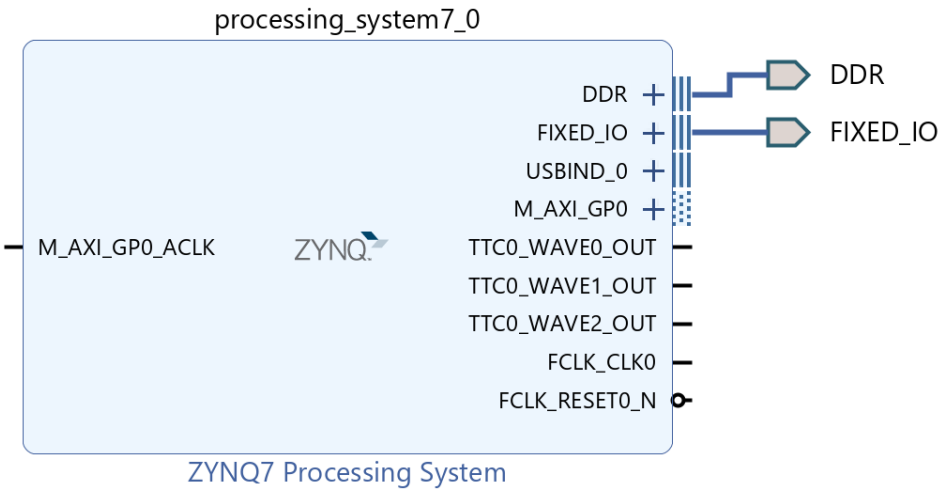
Một vài tính năng của AXI BRAM Controller IP:

- AXI4 slave interface.
- Có interface đọc và ghi riêng biệt để sử dụng cho FPGA BRAM cổng kép.
- Tương thích với Xilinx AXI Interconect.

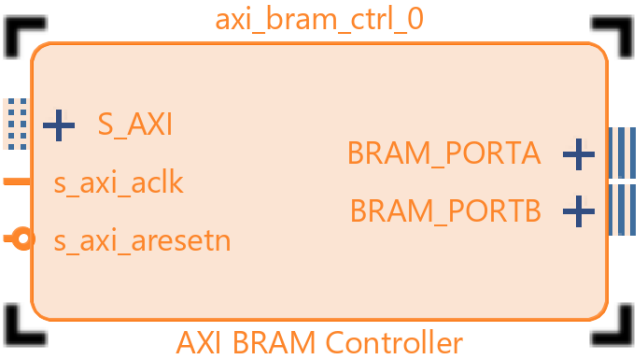
Bổ sung các module để kiểm soát mô-đun phần cứng suy luận mạng nơ-ron được tạo ra.

Module *nn_ctrl* : Khởi động khối nn_inference IP và hiển thị các giá trị của đầu ra. Tùy vào thiết kế, có thể hiển thị trên monitor, LCD, hoặc LED.

Module fix_address: Căn chỉnh địa chỉ đầu ra module IP nn_inference với địa chỉ đầu vào khối BRAM.



Hình 4.3. ZYNQ7 Processing System IP

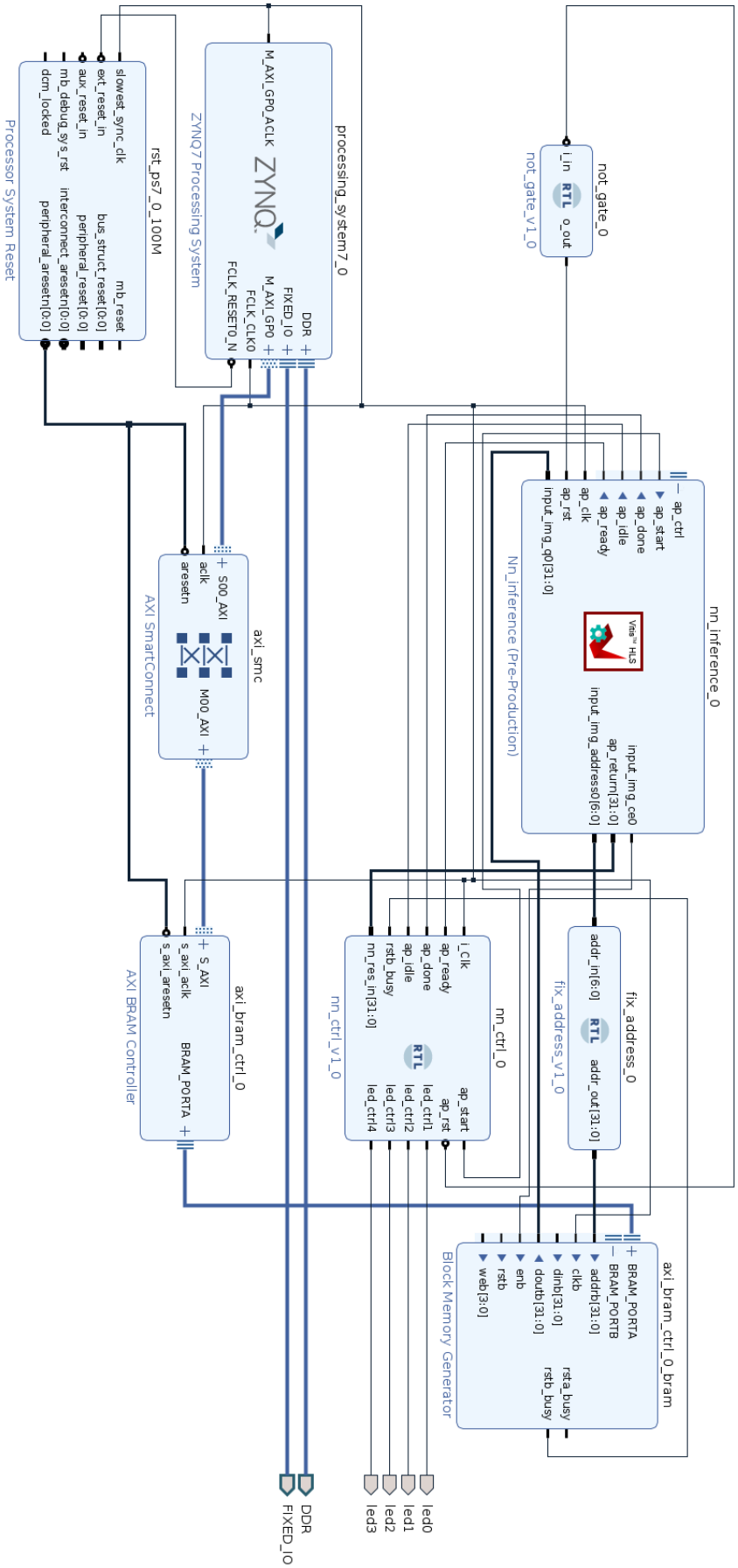


Hình 4.4. AXI BRAM Controller

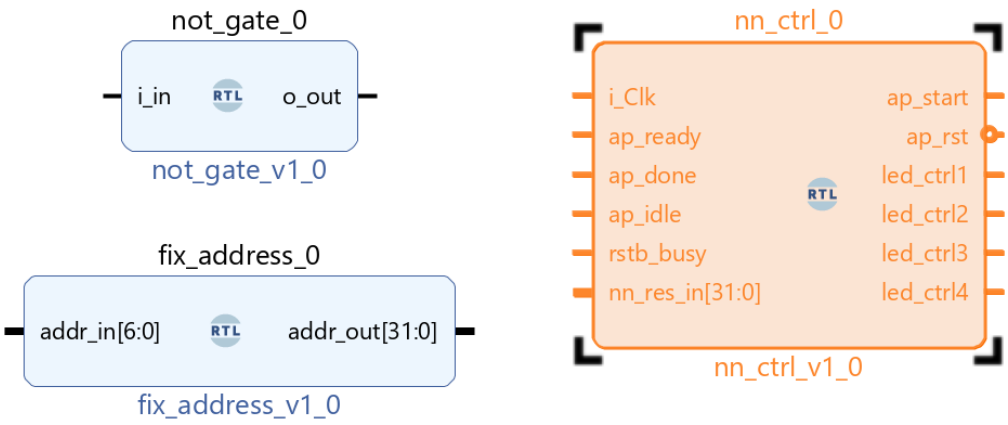
4.2. Kết nối các module.

Thông qua chức năng của các module, các module được kết nối như hình 4.5 bên dưới.

Khi đã thực hiện xong việc thiết lập kết nối giữa các module, ta tiến hành generate bitstream. Bitstream này có thể được sử dụng để nạp lên kip FPGA đã chọn.



Hình 4.5. Thiết kế toàn hệ thống



Hình 4.4. Các module điều khiển các IP và phần cứng

CHƯƠNG 5. KẾT LUẬN

Việc đưa mạng Neural Network xuống phần cứng giúp tăng tốc độ xử lý đồng thời giúp giảm thiểu việc truyền và nhận dữ liệu từ thiết bị tại biên tới máy chủ, qua đó góp phần xử lý được nhanh chóng hơn. Tuy nhiên tài nguyên phần cứng cần có thêm thời gian để tối ưu hơn nữa để giảm thiểu chi phí triển khai. Với những mạng neurons có kiến trúc phức tạp thì việc triển khai lên nền tảng FPGA gặp khó khăn về tài nguyên hạn chế như số cổng logic hay tốc độ xung đồng hồ chưa cao.

Đồ án 3 này giới thiệu cách tiếp cập trong việc triển khai AI trên thiết bị tại biên, cụ thể là FPGA. Cách tiếp cập được trình bày theo từng bước dựa trên nền tảng VITIS của XILINX.

Do kiến thức còn nhiều hạn chế, trong khoảng thời gian ngắn nên đề tài chưa hoàn chỉnh, có nhiều thiếu sót. Em rất mong nhận được sự góp ý từ thầy.

Em xin chân thành cảm ơn.

TÀI LIỆU THAM KHẢO

[1]. https://github.com/nhma20/FPGA_AI, truy cập lần cuối này 07/03/2023.

[2]. N. Malle and E. Ebeid, "Open-Source Educational Platform for FPGA Accelerated AI in Robotics," 2022 8th International Conference on Mechatronics and Robotics Engineering (ICMRE), 2022, pp. 112-115, doi: 10.1109/ICMRE54455.2022.9734102.

[3]. https://www.amiq.com/consulting/2018/12/14/how-to-implement-a-convolutional-neural-network-using-high-level-synthesis/, truy cập lần cuối ngày 07/03/2023

[4]. David Gschwend, “ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network”

[5]. https://www.xilin.com/, truy cập lần cuối ngày 07/03/2023.