



# BÁO CÁO ĐỒ ÁN TỐT NGHIỆP ĐỀ TÀI NHẬN DIỆN BIỂN BÁO GIAO THÔNG TRÊN FPGA PYNQ-Z2

GVHD: TS. Võ Quê Sơn  
SVTH: Huỳnh Thịnh Phát - 2114369

Tháng 5 2025  
VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

# Nội dung

- 1 Giới thiệu về đề tài
- 2 Giới thiệu về CNN và BNN
- 3 Tổng quan về FPGA PYNQ-Z2
- 4 Các bước thực hiện đề tài
- 5 Chạy kiểm nghiệm trên Jupyter Notebook
- 6 Kết luận và hướng phát triển

## 1 Giới thiệu về đề tài

2 Giới thiệu về CNN và BNN

3 Tổng quan về FPGA PYNQ-Z2

4 Các bước thực hiện đề tài

5 Chạy kiểm nghiệm trên Jupyter Notebook

6 Kết luận và hướng phát triển



# Giới thiệu về đề tài

Đề tài "Nhận diện biển báo giao thông trên FPGA PYNQ-Z2" nghiên cứu và phát triển hệ thống nhận diện biển báo giao thông sử dụng công nghệ FPGA kết hợp với các phương pháp học sâu.

Hệ thống áp dụng mạng Nơ-ron tích chập (CNN) để phân loại chính xác các loại biển báo giao thông, ứng dụng model BNN để triển khai trên nền tảng FPGA giúp giảm độ trễ và tiết kiệm năng lượng.

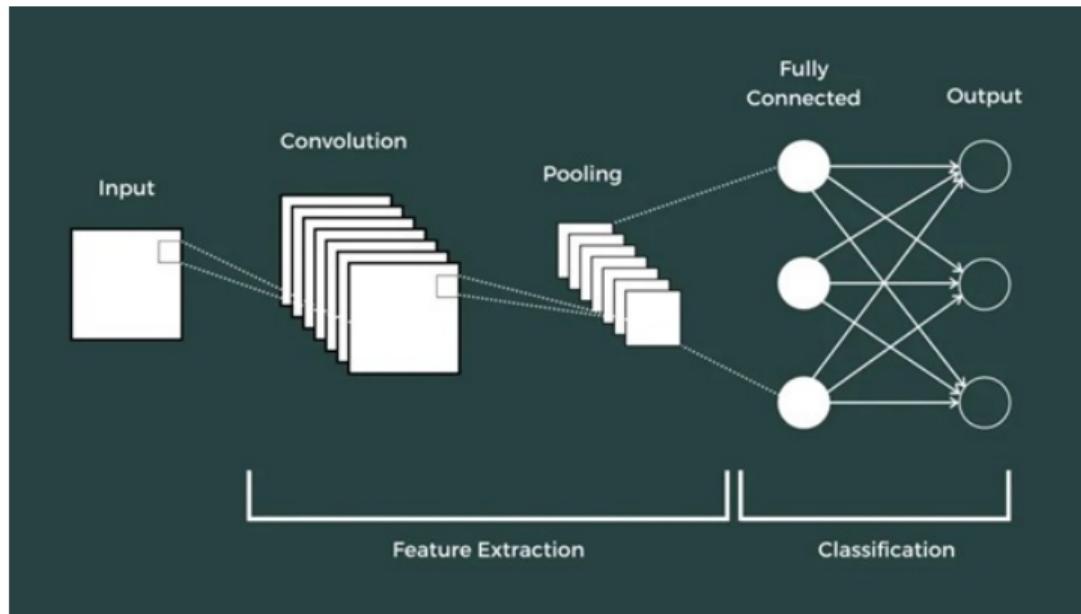
# Nội dung

- 1 Giới thiệu về đề tài
- 2 **Giới thiệu về CNN và BNN**
- 3 Tổng quan về FPGA PYNQ-Z2
- 4 Các bước thực hiện đề tài
- 5 Chạy kiểm nghiệm trên Jupyter Notebook
- 6 Kết luận và hướng phát triển



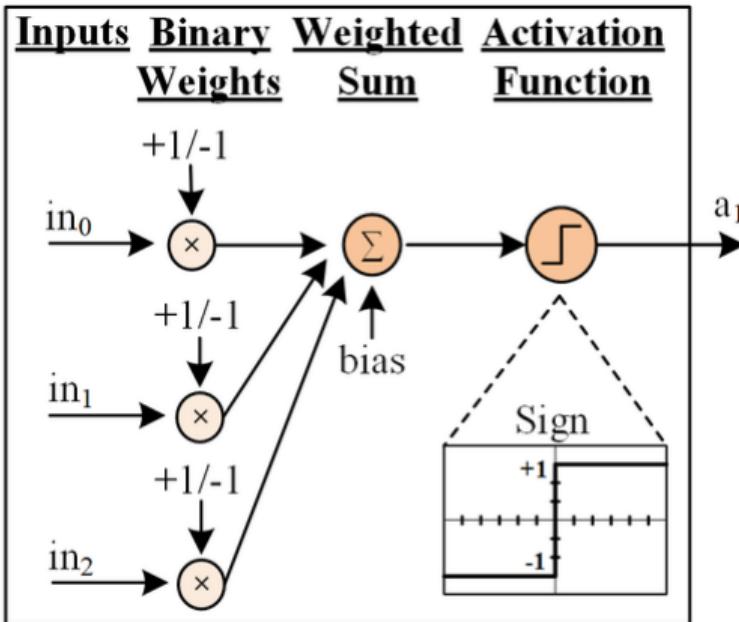
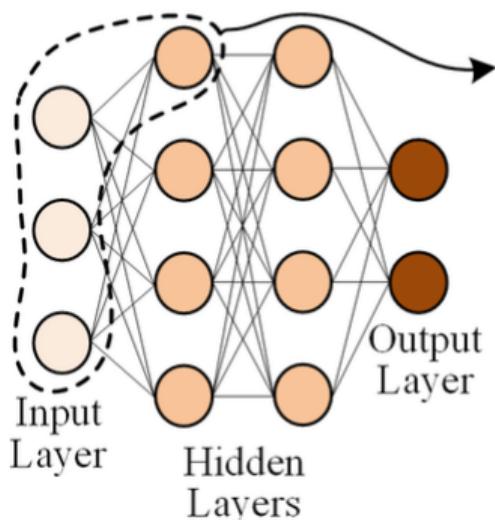
# Mạng Nơ-ron Tích chập (CNN)

Mạng Nơ-ron Tích chập (CNN) là một mô hình học sâu rất hiệu quả trong nhận diện hình ảnh và video. CNN sử dụng các lớp tích chập để tự động trích xuất các đặc trưng từ hình ảnh mà không cần sự can thiệp của con người. Cấu trúc chính của CNN bao gồm ba loại lớp:



# Mạng Nơ-ron Nhị phân (BNN)

Mạng Nơ-ron Nhị phân (BNN) là một phương pháp tối ưu hóa các mô hình học sâu bằng cách nhị phân hóa các trọng số và kích hoạt trong mạng nơ-ron, giúp giảm yêu cầu bộ nhớ và tính toán.

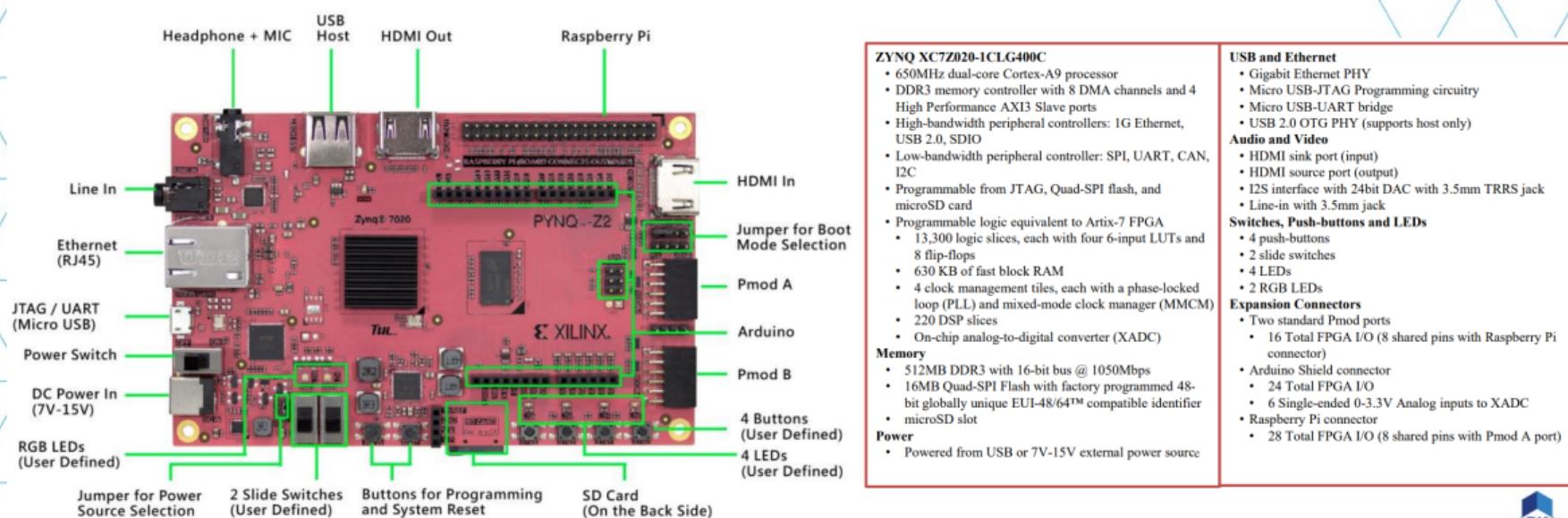


# Nội dung

- 1 Giới thiệu về đề tài
- 2 Giới thiệu về CNN và BNN
- 3 **Tổng quan về FPGA PYNQ-Z2**
- 4 Các bước thực hiện đề tài
- 5 Chạy kiểm nghiệm trên Jupyter Notebook
- 6 Kết luận và hướng phát triển

# Giới thiệu về kit FPGA PYNQ-Z2

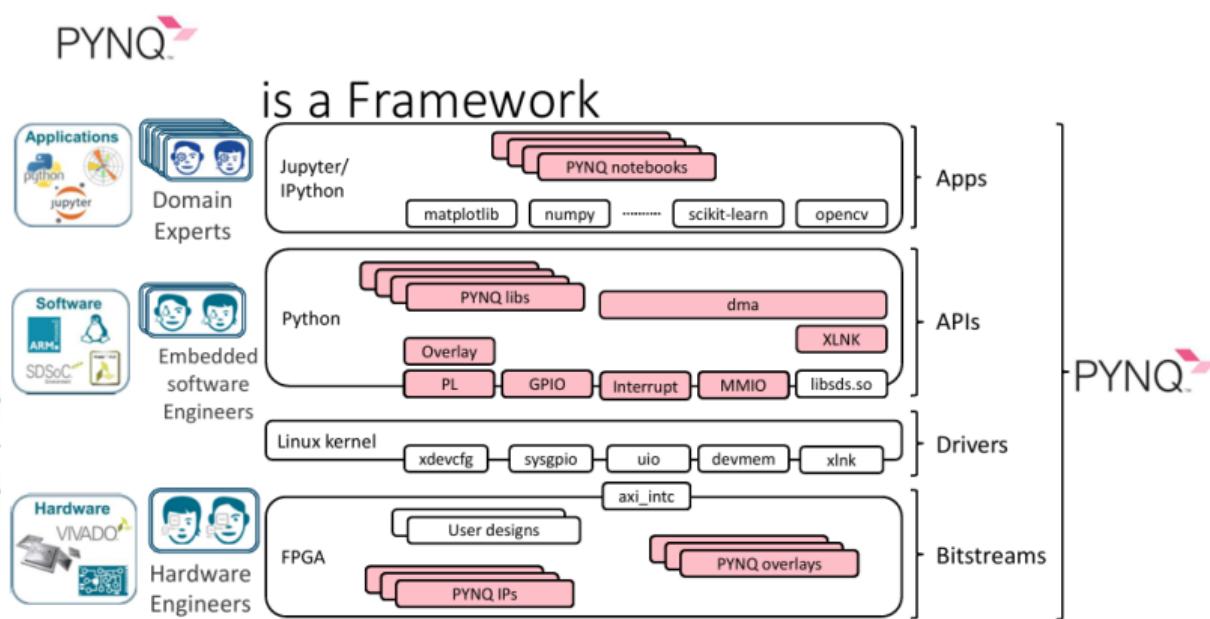
PYNQ-Z2 là một nền tảng học tập và nghiên cứu vi mạch số (FPGA) hiệu quả, kết hợp phần cứng FPGA mạnh mẽ của Xilinx với phần mềm Python linh hoạt. PYNQ-Z2 cung cấp cho người dùng môi trường lập trình trực quan, dễ sử dụng, cho phép họ thiết kế, mô phỏng và triển khai các vi mạch số một cách nhanh chóng và hiệu quả.



# Giới thiệu về PYNQ Framework

PYNQ Framework cho phép lập trình FPGA bằng Python, giúp các nhà phát triển dễ dàng sử dụng và triển khai ứng dụng mà không cần viết mã phần cứng.

Tính năng: Tích hợp dễ dàng với Jupyter Notebook, giúp các nhà phát triển kiểm tra và tối ưu hóa thiết kế FPGA nhanh chóng.

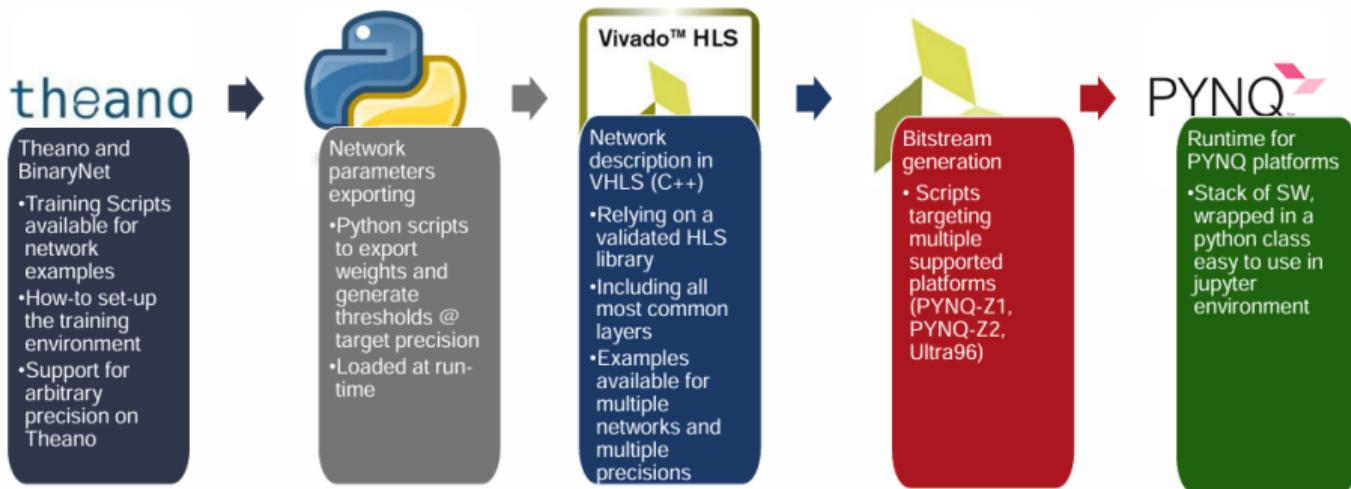


# Nội dung

- 1 Giới thiệu về đề tài
- 2 Giới thiệu về CNN và BNN
- 3 Tổng quan về FPGA PYNQ-Z2
- 4 **Các bước thực hiện đề tài**
- 5 Chạy kiểm nghiệm trên Jupyter Notebook
- 6 Kết luận và hướng phát triển

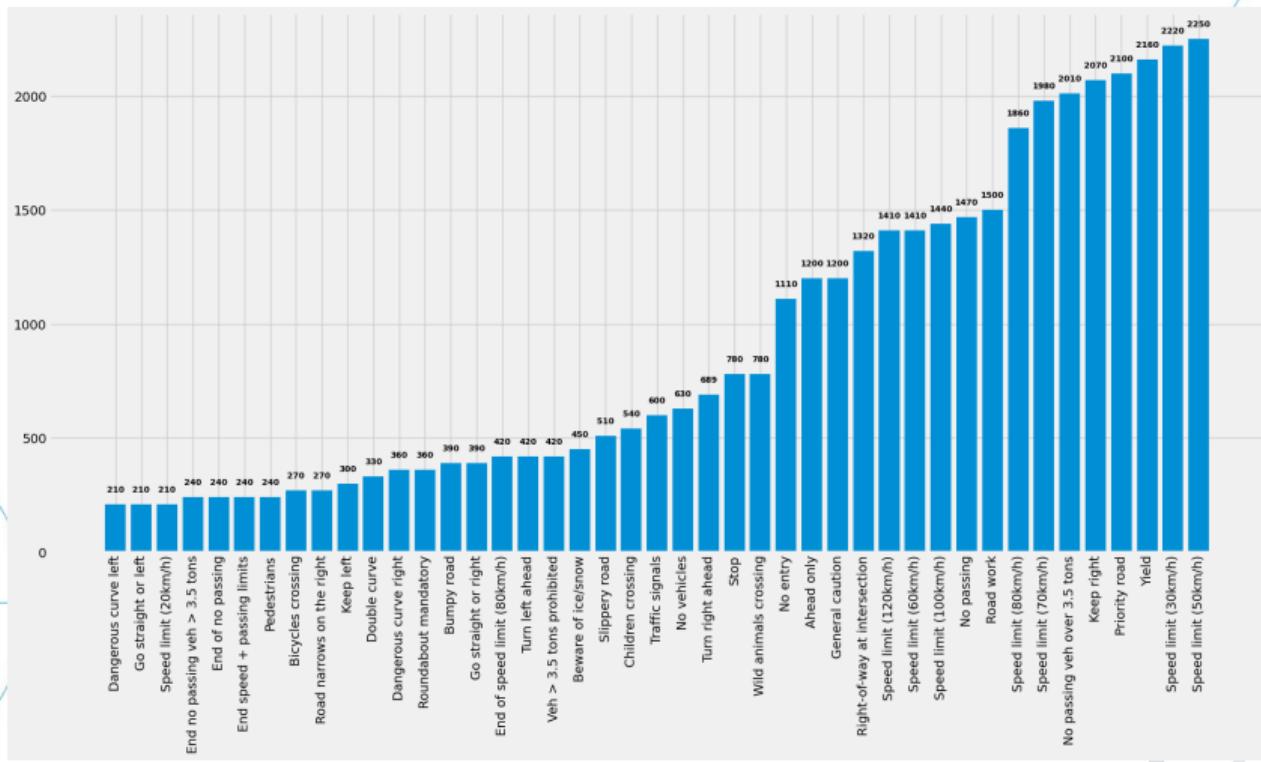
# Các bước thực hiện đề tài

- ① Huấn luyện CNN trên Python.
- ② Áp dụng BNN để tối ưu hóa mô hình.
- ③ Triển khai trên nền tảng FPGA PYNQ-Z2.



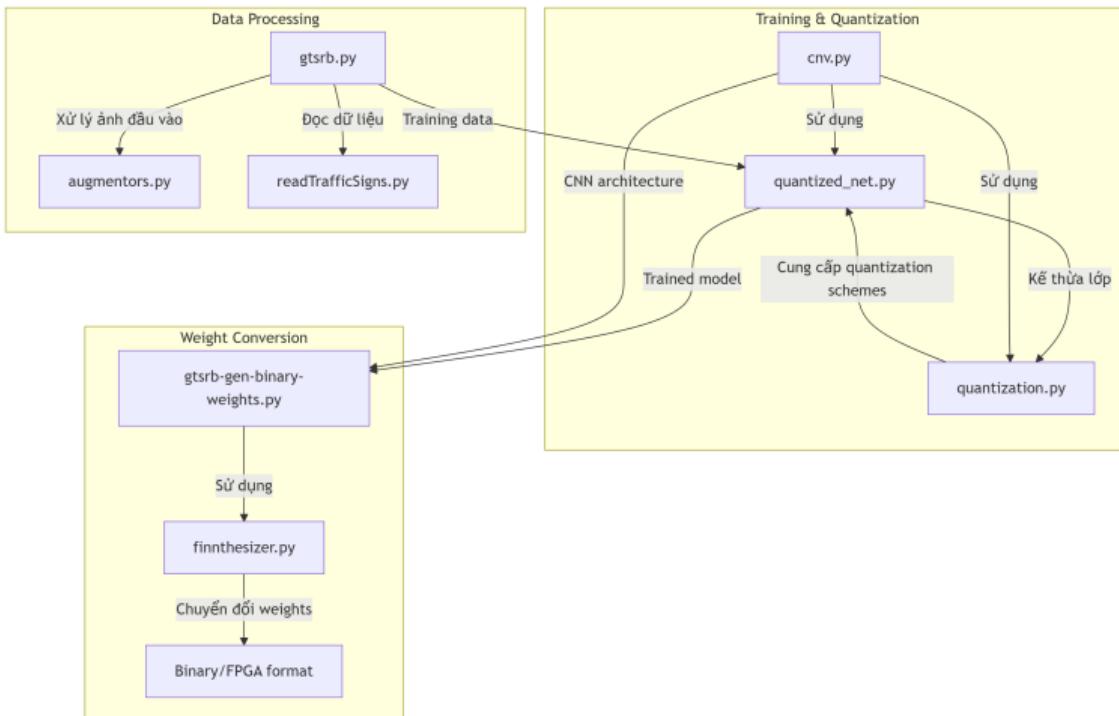
# Chuẩn bị dữ liệu đầu vào

Biểu đồ phân bố của khoảng 39210 hình ảnh được phân loại theo 43 class.

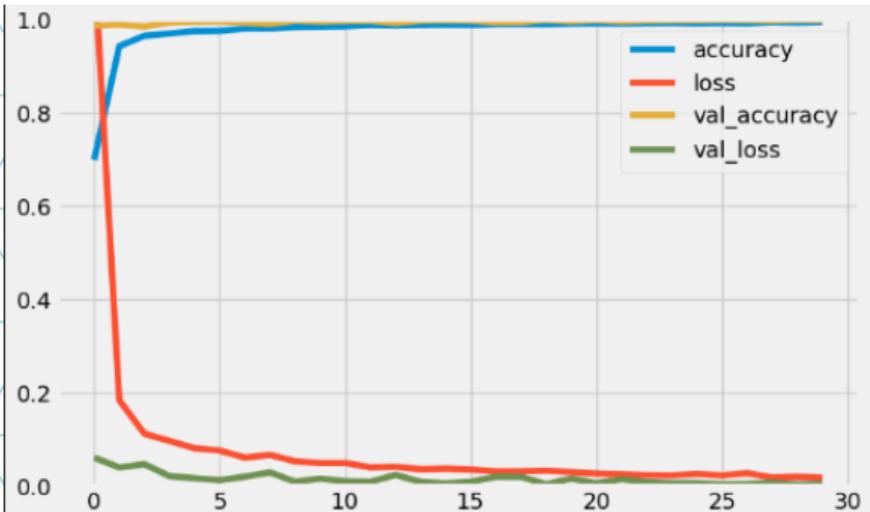


# Huấn luyện CNN trên Python

Sơ đồ giải thuật tổng quát, file training được chia ra 0.7 cho train và 0.3 cho validation.



# Kết quả huấn luyện CNN



## Các kỹ thuật áp dụng:

- **Augmentation dữ liệu:**

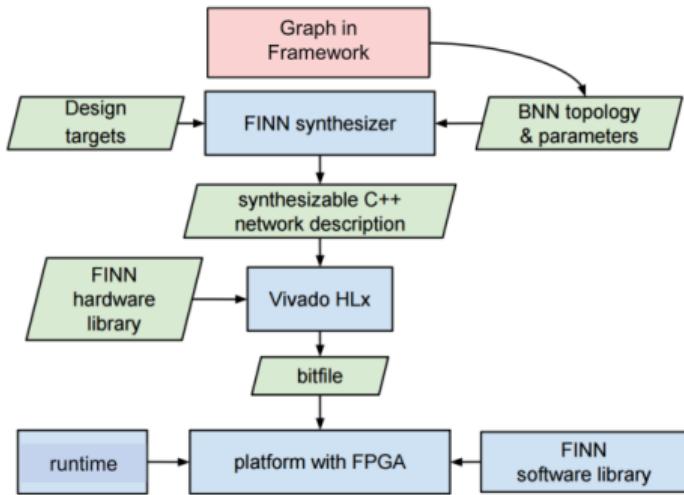
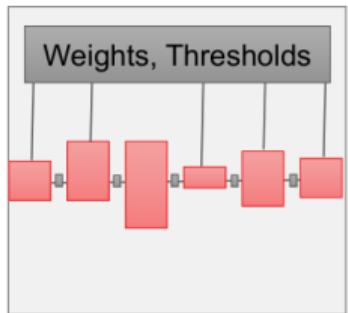
- Xoay ảnh: 15°
- Dịch chuyển ngang/dọc: 10%
- Zoom: 15%

- **Regularization:**

- Dropout (tỉ lệ 0.5)

- Batch normalization

# Chuyển đổi tham số bằng FINN



- Each layer custom parameters

- Number of Bits
- Scaling factor
- Folding factor

# Xuất ra file dùng cho chạy nhị phân

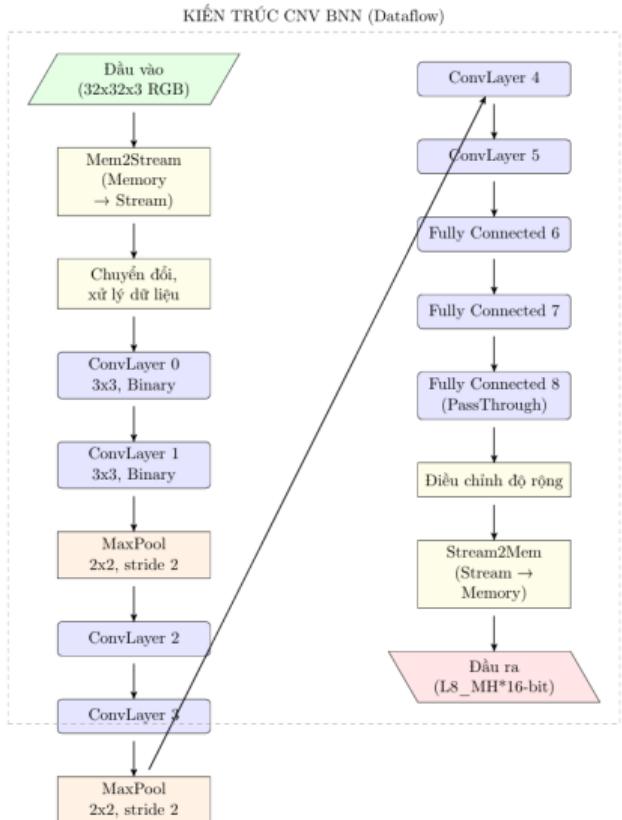
Khi huấn luyện hoàn thành, tệp NPZ chứa trọng số sẽ được chuyển đổi thành các tệp nhị phân thông qua gtsrb-gen-binary-weights.py và finnthesizer.py. Các tệp này sẽ được sử dụng để tải vào bộ nhớ của FPGA, nơi mô hình sẽ được triển khai và thực hiện nhận diện biển báo giao thông trong thời gian thực.

Kết quả sau khi test với tệp Test khoảng 12631 dữ liệu hình ảnh.

**395/395** ————— **2s 5ms/step**

**Test Data accuracy: 98.40855106888361**

# Thiết kế kiến trúc HLS cho CNN

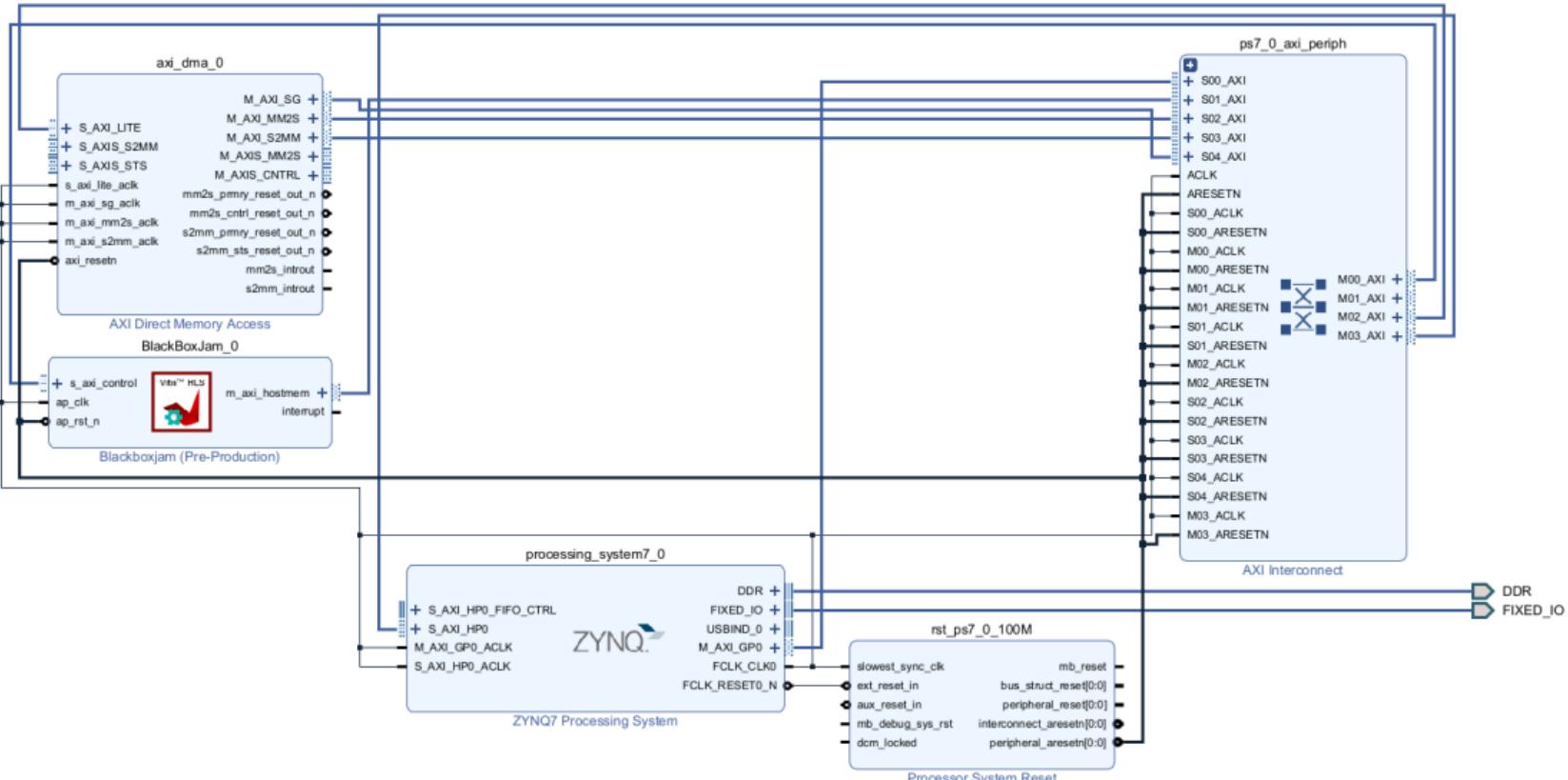


## Đóng gói thiết kế thành IP

The screenshot shows the Vivado IDE interface with several windows open:

- Explorer**: Shows the project structure with files like config.h, bnn-library.h, weights.hpp, activations.hpp, interpret.hpp, and mvau.hpp.
- Text Editor**: Displays the C++ code for `top.cpp`, which includes headers for config, bnn-library, weights, activations, interpret, and mvau. The code defines various static variables for `BinaryWeights` and `ThresholdedActivation` functions, along with a `paddedsize16W` function.
- Outline**: Shows the `Synthesis Summary` for the solution, listing various components and their properties.
- Flow Navigator**: Provides navigation for simulation, synthesis, and reports.
- Vitis HLS Console**: Displays the terminal output of the HLS build process, including compilation logs and memory usage statistics.

# Tạo và xuất Block Design



# Tài nguyên đã sử dụng sau khi Implementation

Design Timing Summary																																																												
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	MHS(ns)	THS(ns)																																																							
0.285	0.000	0	108705	0.020	0.000																																																							
All user specified timing constraints are met.																																																												
<table border="1"><thead><tr><th>Site Type</th><th>Used</th><th>Fixed</th><th>Available</th><th>Util%</th></tr></thead><tbody><tr><td>Slice LUTs</td><td>26072</td><td>0</td><td>53200</td><td>49.01</td></tr><tr><td>LUT as Logic</td><td>24047</td><td>0</td><td>53200</td><td>45.20</td></tr><tr><td>LUT as Memory</td><td>2025</td><td>0</td><td>17400</td><td>11.64</td></tr><tr><td>LUT as Distributed RAM</td><td>1578</td><td>0</td><td></td><td></td></tr><tr><td>LUT as Shift Register</td><td>447</td><td>0</td><td></td><td></td></tr><tr><td>Slice Registers</td><td>41312</td><td>0</td><td>106400</td><td>38.83</td></tr><tr><td>Register as Flip Flop</td><td>41312</td><td>0</td><td>106400</td><td>38.83</td></tr><tr><td>Register as Latch</td><td>0</td><td>0</td><td>106400</td><td>0.00</td></tr><tr><td>F7 Muxes</td><td>892</td><td>0</td><td>26600</td><td>3.35</td></tr><tr><td>F8 Muxes</td><td>241</td><td>0</td><td>13300</td><td>1.81</td></tr></tbody></table>						Site Type	Used	Fixed	Available	Util%	Slice LUTs	26072	0	53200	49.01	LUT as Logic	24047	0	53200	45.20	LUT as Memory	2025	0	17400	11.64	LUT as Distributed RAM	1578	0			LUT as Shift Register	447	0			Slice Registers	41312	0	106400	38.83	Register as Flip Flop	41312	0	106400	38.83	Register as Latch	0	0	106400	0.00	F7 Muxes	892	0	26600	3.35	F8 Muxes	241	0	13300	1.81
Site Type	Used	Fixed	Available	Util%																																																								
Slice LUTs	26072	0	53200	49.01																																																								
LUT as Logic	24047	0	53200	45.20																																																								
LUT as Memory	2025	0	17400	11.64																																																								
LUT as Distributed RAM	1578	0																																																										
LUT as Shift Register	447	0																																																										
Slice Registers	41312	0	106400	38.83																																																								
Register as Flip Flop	41312	0	106400	38.83																																																								
Register as Latch	0	0	106400	0.00																																																								
F7 Muxes	892	0	26600	3.35																																																								
F8 Muxes	241	0	13300	1.81																																																								
<table border="1"><thead><tr><th>Site Type</th><th>Used</th><th>Fixed</th><th>Available</th><th>Util%</th></tr></thead><tbody><tr><td>Block RAM Tile</td><td>124</td><td>0</td><td>140</td><td>88.57</td></tr><tr><td>RAMB36/FIFO*</td><td>76</td><td>0</td><td>140</td><td>54.29</td></tr><tr><td>RAMB36E1 only</td><td>76</td><td></td><td></td><td></td></tr><tr><td>RAMB18</td><td>96</td><td>0</td><td>280</td><td>34.29</td></tr><tr><td>RAMB18E1 only</td><td>96</td><td></td><td></td><td></td></tr></tbody></table>						Site Type	Used	Fixed	Available	Util%	Block RAM Tile	124	0	140	88.57	RAMB36/FIFO*	76	0	140	54.29	RAMB36E1 only	76				RAMB18	96	0	280	34.29	RAMB18E1 only	96																												
Site Type	Used	Fixed	Available	Util%																																																								
Block RAM Tile	124	0	140	88.57																																																								
RAMB36/FIFO*	76	0	140	54.29																																																								
RAMB36E1 only	76																																																											
RAMB18	96	0	280	34.29																																																								
RAMB18E1 only	96																																																											
<table border="1"><thead><tr><th>Site Type</th><th>Used</th><th>Fixed</th><th>Available</th><th>Util%</th></tr></thead><tbody><tr><td>DSPs</td><td>24</td><td>0</td><td>220</td><td>10.91</td></tr><tr><td>DSP48E1 only</td><td>24</td><td></td><td></td><td></td></tr></tbody></table>						Site Type	Used	Fixed	Available	Util%	DSPs	24	0	220	10.91	DSP48E1 only	24																																											
Site Type	Used	Fixed	Available	Util%																																																								
DSPs	24	0	220	10.91																																																								
DSP48E1 only	24																																																											
...																																																												

# Nội dung

- 1 Giới thiệu về đề tài
- 2 Giới thiệu về CNN và BNN
- 3 Tổng quan về FPGA PYNQ-Z2
- 4 Các bước thực hiện đề tài
- 5 Chạy kiểm nghiệm trên Jupyter Notebook**
- 6 Kết luận và hướng phát triển

# Khởi chạy BNN trên phần cứng và phần mềm

```
In [3]: from PIL import Image
import numpy as np
from os import listdir
from os.path import isfile, join
from IPython.display import display

imgList = [f for f in listdir("/home/xilinx/jupyter_notebooks/bnn/pictures/road_signs/") if isfile(join("/home/xilinx/jupyter_no
images = []

for imgFile in imgList:
    img = Image.open("/home/xilinx/jupyter_notebooks/bnn/pictures/road_signs/" + imgFile)
    images.append(img)
    img.thumbnail((64, 64), Image.ANTIALIAS)
    display(img)
```



# Khởi chạy BNN trên phần cứng và phần mềm

```
In [4]: results = classifier.classify_images(images)
print("Identified classes: {0}".format(results))
for index in results:
    print("Identified class name: {0}".format((classifier.class_name(index))))
```

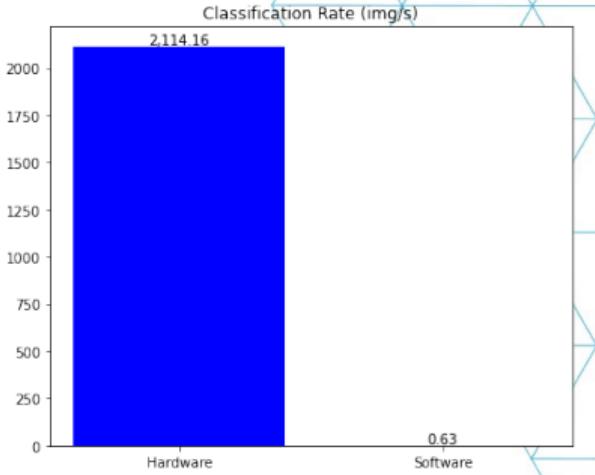
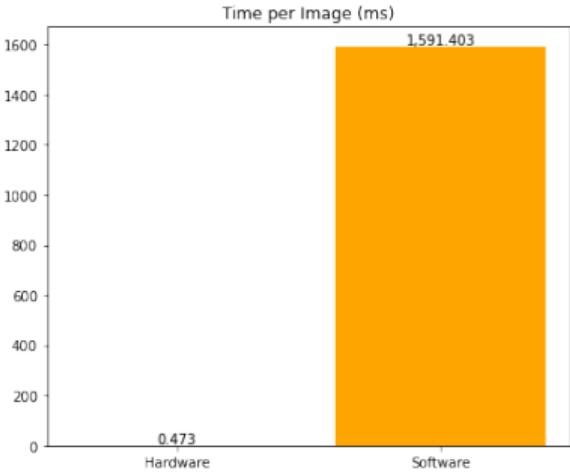
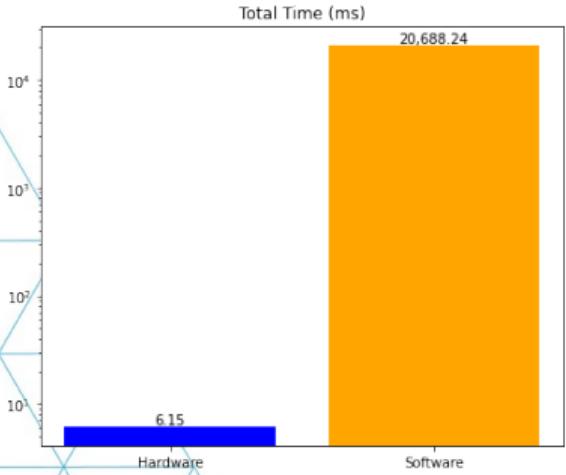
```
Inference took 915.00 microseconds, 305.00 usec per image
Classification rate: 3278.69 images per second
Identified classes: [41 27 14]
Identified class name: End of no-overtaking zone
Identified class name: Pedestrians in road ahead
Identified class name: Stop
```

```
In [5]: sw_class = bnn.CnvClassifier(bnn.NETWORK_CNW1A1, "road-signs", bnn.RUNTIME_SW)

results = sw_class.classify_images(images)
print("Identified classes: {0}".format(results))
for index in results:
    print("Identified class name: {0}".format((classifier.class_name(index))))
```

```
Inference took 1252522.97 microseconds, 417507.66 usec per image
Classification rate: 2.40 images per second
Identified classes: [41 27 14]
Identified class name: End of no-overtaking zone
Identified class name: Pedestrians in road ahead
Identified class name: Stop
```

# Khởi chạy BNN trên phần cứng và phần mềm



```
In [9]: speedup_total = sw_time_total / hw_time_total  
speedup_per_image = sw_time_per_image / hw_time_per_image  
speedup_rate = hw_rate / sw_rate  
  
print("Speedup Factors:")  
print(f"- Total time: {speedup_total:.1f}x faster in hardware")  
print(f"- Time per image: {speedup_per_image:.1f}x faster in hardware")  
print(f"- Classification rate: {speedup_rate:.1f}x higher in hardware")
```

Speedup Factors:  
- Total time: 3364.5x faster in hardware  
- Time per image: 3364.5x faster in hardware  
- Classification rate: 3355.8x higher in hardware

## Phát hiện đối tượng trong cảnh

Inference took 145902.00 microseconds, 109.70 usec per image  
Classification rate: 9115.71 images per second

Out[8]



# Nội dung

- 1 Giới thiệu về đề tài
- 2 Giới thiệu về CNN và BNN
- 3 Tổng quan về FPGA PYNQ-Z2
- 4 Các bước thực hiện đề tài
- 5 Chạy kiểm nghiệm trên Jupyter Notebook
- 6 Kết luận và hướng phát triển



# Kết luận

- Đề tài "Nhận diện biển báo giao thông trên FPGA PYNQ-Z2" đã nghiên cứu và triển khai thành công một hệ thống nhận diện biển báo giao thông sử dụng công nghệ FPGA kết hợp với các phương pháp học sâu, đặc biệt là Mạng Nơ-ron Tích chập (CNN) và Mạng Nơ-ron Nhị phân (BNN).
- Việc triển khai trên FPGA giúp giảm độ trễ, tiết kiệm năng lượng và tối ưu hóa hơn so với triển khai trên phần mềm.
- Hệ thống đã chứng minh được khả năng nhận diện chính xác nhiều loại biển báo giao thông từ bộ dữ liệu GTSRB.

# Hướng phát triển

- Tối ưu hóa mô hình học sâu, đặc biệt là các thuật toán CNN và BNN để đạt độ chính xác cao hơn khi triển khai trên FPGA.
- Mở rộng ứng dụng của hệ thống vào các nền tảng giao thông tự động hóa, kết hợp với các cảm biến và camera giám sát để nâng cao tính hiệu quả và an toàn cho giao thông thông minh.

**XIN CẢM ƠN QUÝ THẦY/CÔ ĐÃ  
CHÚ Ý LẮNG NGHE**