ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO ĐỒ ÁN 2

HIỆN THỰC SoC CAMERA

GIẢNG VIÊN HƯỚNG DẪN:	**TRƯƠNG VĂN CƯƠNG**
SINH VIÊN THỰC HIỆN:	**NGUYỄN TRẦN TRÍ THỨC – 19521007**

Ho Chi Minh City, 01/2023

MỤC LỤC

DANH MỤC HÌNH ẢNH

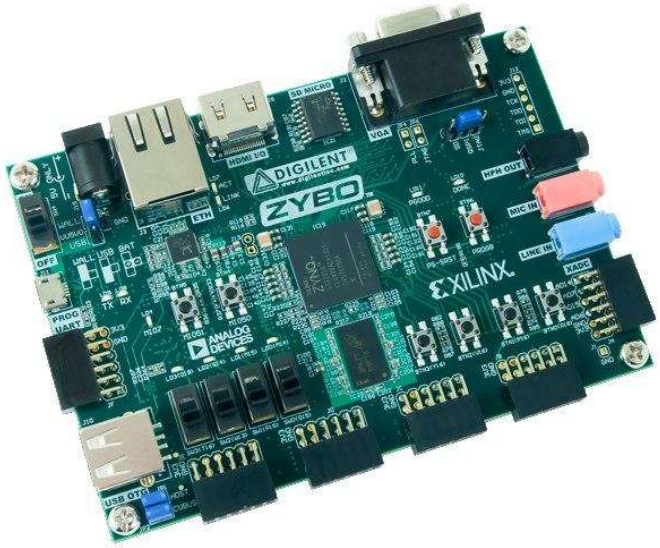
DANH MỤC BẢNG

Chapter 1. GIỚI THIỆU TỔNG QUAN

Với sự bùng nổ của cuộc Cách mạng Công nghiệp 4.0, sự ra đời của các thiết bị thông minh ngày càng giúp cho đời sống của con người trở nên tiện lợi và dễ dàng hơn. Trong đó có camera, một thiết bị không thể thiếu trong việc phục vụ vấn đề an ninh an toàn. Trên thị trường hiện nay không khó bắt gặp nhiều loại camera với nhiều mẫu mã và chức năng khác nhau, kèm theo đó là giá thành cũng đi kèm với chất lượng sản phẩm.

Với mong muốn sử dụng các thiết bị thông minh - trong đó có camera, có chức năng ổn định, giá cả hợp lý. Nhóm đề xuất ý tưởng thiết kế một IP Camera có chức năng phát luồng video trực tiếp qua mạng với giao thức RTSP.

Trong giới hạn đề tài, nhóm ưu tiên sử dụng các board tích hợp Xilinx Chip để nghiên cứu và hiện thực đề tài. Đơn cử như dòng board Zypo Z7 (Zynq-7000 ARM/FPGA SoC Development Board)**.**



Hình 1.1 Board Zybo Z7 (Zynq 7000)

Chapter 2. TÌM HIỂU LÝ THUYẾT

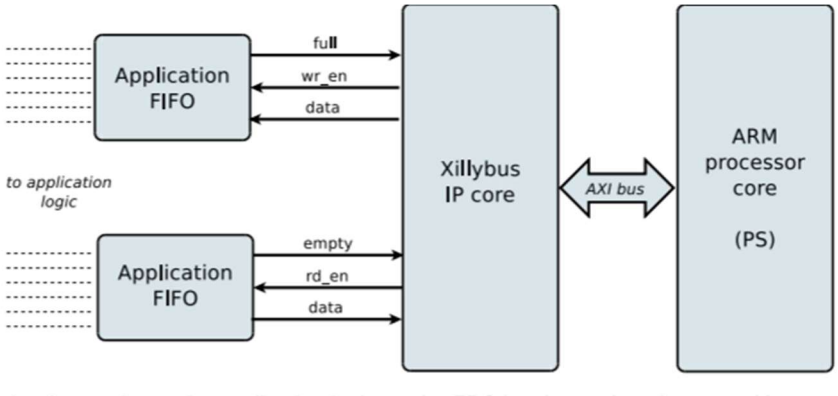
2.1. Board Zybo-Z7

2.1.1. Xillybus là gì

Là một giải pháp chìa khóa trao tay từ đầu đến cuối đơn giản, trực quan, hiệu quả dựa trên (DMA) – hay Truy cập dữ liệu trực tiếp, để truyền dữ liệu giữa FPGA và một máy chủ chạy Linux hoặc Windows. Trong đó, DMA là cơ chế của hệ thống máy tính cho phép một thành phần phần cứng truy cập đến bộ nhớ dữ liệu chính (như RAM) một cách độc lập với CPU[1].

Xillybus hỗ trợ các loại FPGA của Xilinx như Virtex-5T, Spartan-6T, Virtex-6T, các dòng Series – 7, tất cả dòng Ultrascale và Ultrascale+, Zynq-7000/Ultrascale+; của Intel FPGA như Cyclone, Stratix, … Hiện nay các bản phân phối Linux đã bao gồm có Xillybus Driver, đối với Windows, Xillybus hỗ trợ các phiên bản Windows 7, 8 và 10 (32 và 64 bit).

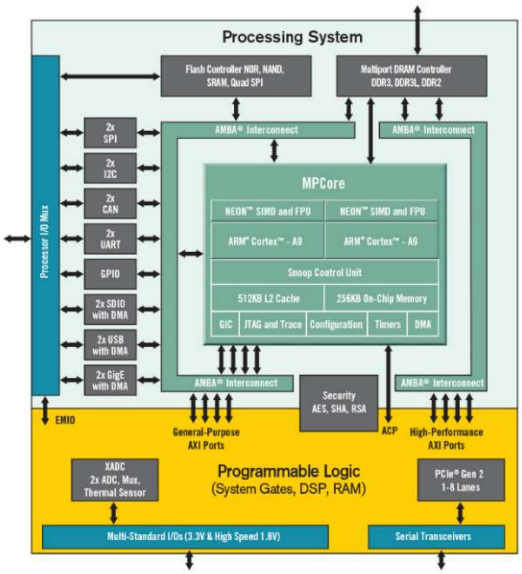
2.1.2. Luồng dữ liệu



Hình 2.1 Sơ đồ luồng dữ liệu

Dòng board Zynq-7000 có kiến trúc được chia làm hai mảng chính bao gồm PS (Processing System) và PL (Programmable Logic). Trong đó PS là hệ thống nhân xử lý đóng vai trò thực hiện các thao tác của ứng dụng phần mềm và PL là phần FPGA có vai trò thực hiện các thao tác của ứng dụng phần cứng. Hai phần này

sẽ giao tiếp thông qua FIFO với bộ Xillybus IP core làm trung gian để phân phối dữ liệu vào các vùng DMA (Direct Memory Access) (Hình 2.1). [2]



Hình 2.2 Kiến trúc của họ chip Zynq 7000

2.1.3. Hành vi của FIFO

Tùy thuộc vào hướng dữ liệu, mà các Tín hiệu "empty" hoặc "full" của FIFO được kết nối vào lõi IP Xillybus và được nhận ra để xác định sẽ xảy ra bùng nổ một loạt dữ liệu (data burst) hay không. Một khi xảy ra, các tín hiệu này được nhận ra để đảm bảo lõi IP Xillybus không có gắng đọc từ một FIFO trống hoặc ghi vào một FIFO đã đầy.

Trong một FIFO hoạt động bình thường, tín hiệu "empty" có thể lên cao chỉ trong 1 chu kì clock sau khi read enable được xác nhận. Tín hiệu "full" có thể lên cao trong 1 chu kì clock sau khi write enable được xác nhận. Hai tín hiệu có thể xuống thấp tại bất cứ lúc nào.

Xillybus IP Core dựa trên hành vi như sao. Khi FIFO báo hiệu cho lõi rằng nó đã sẵn sàng cho việc truyền dữ liệu (tùy điều kiện áp dụng mà có thể là "empty" hoặc "full"), một máy trạng thái trong lõi có thể bắt đầu một chuỗi sự kiện mà sẽ dẫn đến việc chuyển ít nhất một phần tử dữ liệu (data element), bất kể tín hiệu của FIFO tại thời điểm đó là gì. Tuy việc này về tính toàn vẹn là vô hại, nhưng có thể

dẫn đến dữ liệu không mong muốn và không thể dự đoán trước được dữ liệu trong luồng.[3]

2.1.4. Đọc dữ liệu từ FPGA đến host (PS)

Đối với luồng dữ liệu từ FPGA đến host (ở đây là phần PS – cụ thể hơn là hệ điều hành linux). Nếu đây là luồng dữ liệu bất đồng bộ thì IP cores sẽ cố gắng đẩy dữ liệu để làm đầy phần DMA bất cứ khi nào có thể. Điều này có nghĩa là bất cứ khi nào có một device được truy cập và dữ liệu có sẵn và các vùng buffer của nó trống thì việc đẩy dữ liệu sẽ diễn ra.

Trong khi đó, đối với các luồng dữ liệu đồng bộ, IP cores sẽ lấy dữ liệu từ các mạnh logic ứng dụng của người dùng (thường là FIFO) chỉ khi các ứng dụng phần mềm của hệ điều hành yêu cầu đọc dữ liệu từ các device. Nói cách khác, khi các ứng dụng phần mềm đang ở giữa chu trình gọi hàm read().

Luồng đồng bộ sẽ phù hợp cho các ứng dụng mà dữ liệu cần được đọc thì quan trọng. Và luồng bất đồng bộ thì lại rất phù hợp cho các ứng dụng cần băng thông dữ liệu lớn.

2.1.5. Ghi dữ liệu từ host (PS) đến FPGA

Đối với luồng dữ liệu từ host đến FPGA, các luồng bất đồng bộ sẽ hoạt động bằng cách cứ ứng dụng phần mềm khi gọi hàm write() thì nó sẽ return ngay lập tức miễn là vùng DMA có đủ chỗ cho dữ liệu truyền vào. Sau đó việc truyền dữ liệu đến FPGA với tốc độ phụ thuộc vào các tính hiệu logic của FPGA và không còn liên quan gì đến các ứng dụng phần mềm tại host.

Các IP cores được xây dựng trên chuẩn AXI thì các luồng dữ liệu sẽ được chuyển đến FPGA chỉ khi một trong các yếu tố sau xảy ra.

- Vùng DMA hiện tại đã đầy.
- Ứng dụng phần mềm trực tiếp yêu cầu đẩy dữ liệu.
- Đóng các device.
- Bộ đếm thời gian bị tràn và không có bất kỳ dữ liệu nào mới được đẩy và DMA (thông thường là 10ms).

Bên cạnh đó, các luồng đồng bộ, hàm write() sẽ không return cho đến khi dữ liệu được gửi đến các ứng dụng logic trên FPGA. Điều này giúp đảm bảo không mất mát dữ liệu nhưng lại không phù hợp với các ứng yêu cầu dữ liệu có bang thông cao.

2.2. Kiến trúc mạng tích chập (CNN)

2.2.1. Hiện thực bằng phần mềm

Hiện thực mạng CNN (convolutional neural network) đơn giản với bộ dataset MNIST và thư viện keras (Bảng 2.1) bằng python.

Layer	Output Shape	Param #
Conv2d	(None, 26, 26, 32)	320
Max_pooling2d	(None, 13, 13, 32)	0
Flatten	(None, 5408)	0
FC (activation=ReLu)	(None, 100)	1010
FC (activation=SoftMax)	(None, 10)	1010

Bảng 2.1 Kiến trúc của mạng CNN

2.2.2. Hiện thực bằng phần cứng

Một số thành phần của mạng CNN sẽ được hiện thực bằng phần cứng:

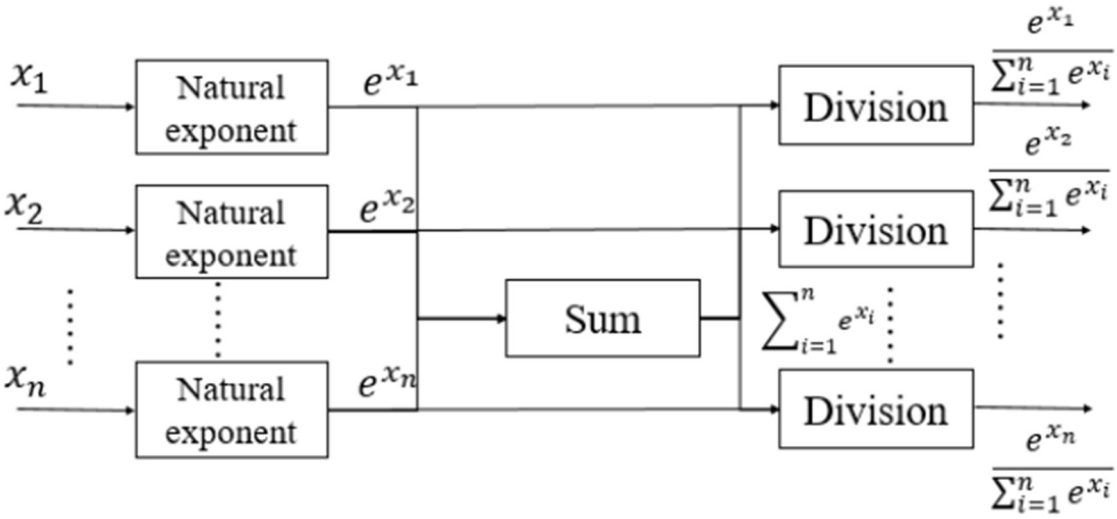
Khối convolution có khả năng hoạt động với kích thước kernel tối đa bằng 3, stride tối đa bằng 2 và có thể padding. Việc thiết kế khối convolution có nhiều hướng tiếp cận khác nhau [4]:

Nhân ma trận, với hướng tiếp cận này thì hệ thống sẽ coi khối tích chập như một mạch nhân ma trận lớp. Lúc này các giá trị đầu vào sẽ được sắp xếp vào các ma trận cột, và tất cả các ma trận này sẽ được nối lại như định dạng ma trận của ngôn ngữ C.

Thứ hai là Fast Fourier Transformation, hướng tiếp cận này trong việc ứng dụng thực tế thì một mạch tích chập trong miền không gian sẽ tương ứng với một phép nhân từng phần tử trong miền Fourier.

Cuối cùng là hướng tiếp cận bằng cách trượt các filter 2D. Đối với cả hai hướng tiếp cận trên là nhân ma trận và FFT sẽ phụ hợp với các kiến trúc chung như GPUs và chúng cũng đòi hỏi tài nguyên phần cứng lớn hơn. Do đó để phù hợp cho board Zynq-7000 và tối ưu thì nhóm đề xuất sử dụng kiến trúc tích chập trực tiếp.

Khối softmax với kiến trúc (Hình 2.3) với các khối chức năng thành phần bao gồm một khối số mũ tự nhiên, một khối tổng và khối chia (các khối hoạt động trên số single precision floating point).



Hình 2.3 Kiến trúc của một khối softmax [5]

Các khối chức năng khác như maxpooling và ReLu.

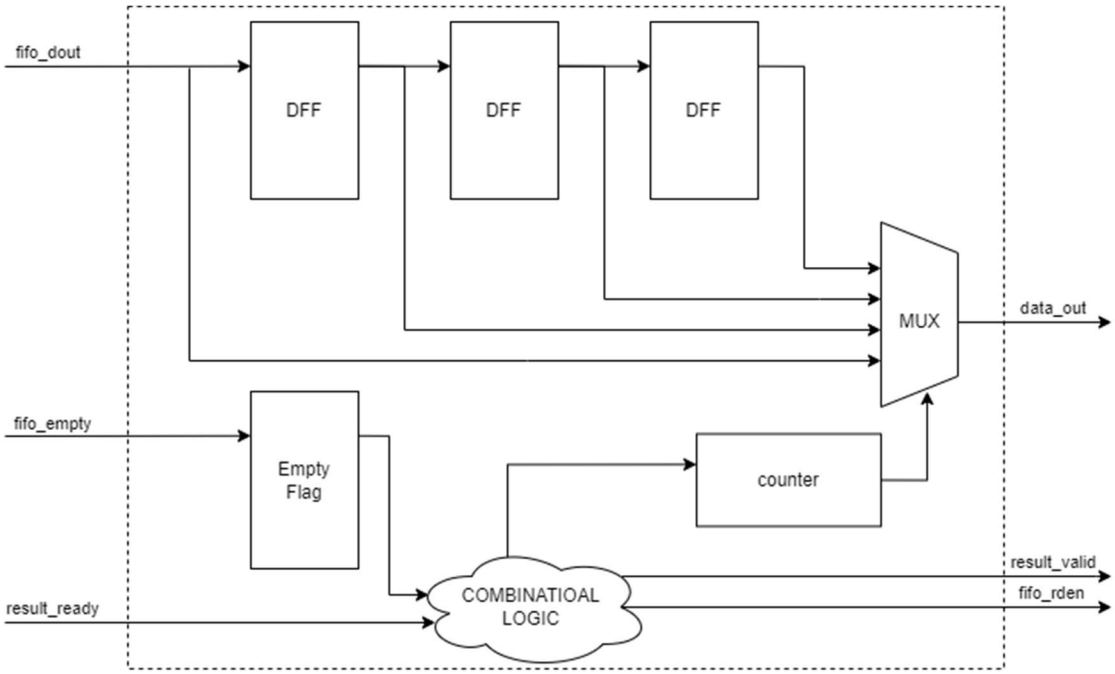
Khối flatten sẽ được thay thế bằng một FIFO bằng cách đẩy lần lượt các giác trị cần làm phẳng vào FIFO theo thứ tự.

Chapter 3. THIẾT KẾ VÀ HIỆN THỰC

3.1. Giao tiếp với dữ liệu dạng hình ảnh

3.1.1. Hiện thực khối đọc/ghi FIFO

Hình 3.1 là kiến trúc của khối đọc FIFO, với khối buffer 3 thanh ghi đảm bảo rằng sẽ luôn sẵn sàng có dữ liệu cho các khối tiếp theo. Với hành vi của FIFO, tín hiệu empty chỉ chuyển trái lên mức cao (nếu trống) khi có tín hiệu đọc gửi đến FIFO và nếu không empty thì 1 chu kỳ sau mới có dữ liệu thì kiến trúc cần một DFF để capture lại tín hiệu empty từ đó đồng bộ được việc capture data phía sau.



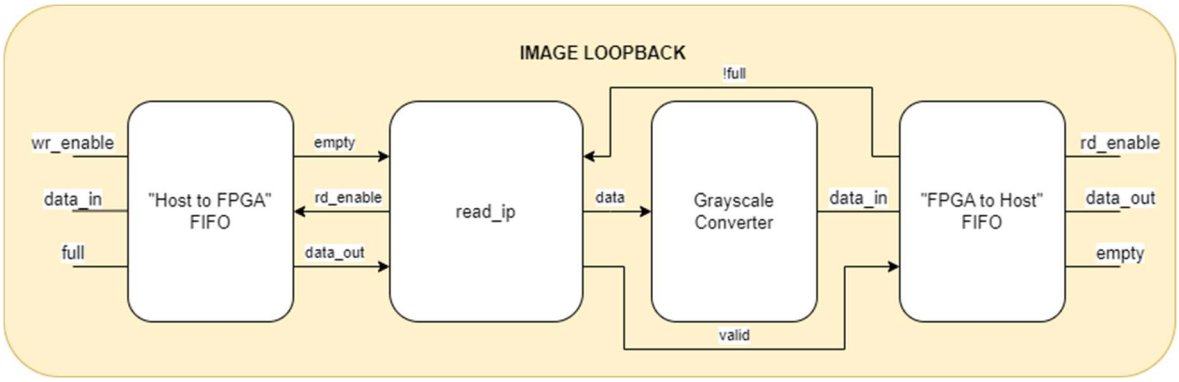
Hình 3.1 Kiến trúc khối đọc FIFO

Đối với việc ghi dữ liệu vào FIFO lại tương đối đơn giản với việc chỉ cần khi khối output cuối cùng sẵn sàng dữ liệu thì sẽ gửi tín hiệu ghi đến FIFO. Lúc này dữ liệu sẽ được capture bởi FIFO.

3.1.2. Thực nghiệm với bộ chuyển đổi ảnh xám

Để kiểm tra chức năng của bộ đọc FIFO, nhóm hiện thực một mạch chuyển ảnh xám đơn giản với việc chuyển đổi ảnh xám được hiện dựa trên công thức:

gray = (blue >> 5 + blue >> 4) + (green >> 4 + green >> 1) + (red >> 5 + red >> 2)



Hình 3.2 Kiến trúc mạch chuyển ảnh xám

Ngoài ra còn cần phải thiết kế ứng dụng phần mềm trên host để có thể nhận và gửi một ảnh có kích thước (height x width x channel).

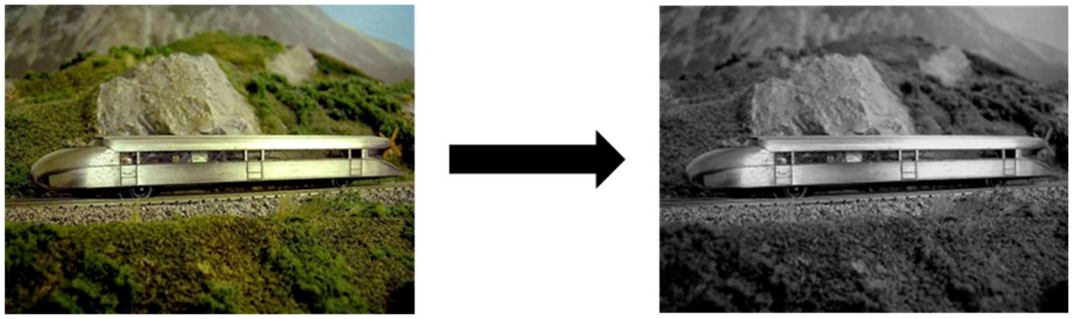
```
buff ← new(int)
buff_idx ← 0
for i ← 0 to height – 1 begin
   for j ← 0 to height – 1 begin
      for c ← 0 to channel – 1 begin
         buf[buff_idx++] ← image[i][j][c]
      buf[buff_idx++] ← ‘/0’
   end
end
while not eof(buff) begin
   write(buf, fd)
end
```

Bảng 3.1 Thuật toán đọc ảnh và ghi vào FPGA

```
buff ← new(int)
i ← 0
j ← 0
image ← initial
while i != height begin
   buff ← read(fd)
   buff_idx ← 0
   while not eof(buff) begin
      for c ← 0 to channel – 1 begin
         image[i][j][c] ← buff[buff_idx++]
      end
      j ← max(width, j++)
      if j == width begin
         j ← 0
```

```
        i++
      end
   end
end
```

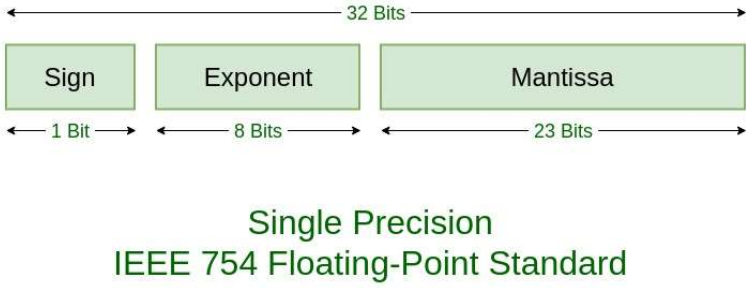
Bảng 3.2 Thuật toán đọc dữ liệu từ FPGA và chuyển thành ảnh



Hình 3.3 Kết quả trong thực nghiệm với ảnh có kích thước 640x480

3.2. Xử lý dữ liệu dạng số thực (single precision floating point)

3.2.1. Hiện thực ứng dụng đọc/ghi số floating point bằng ngôn ngữ lập trình C

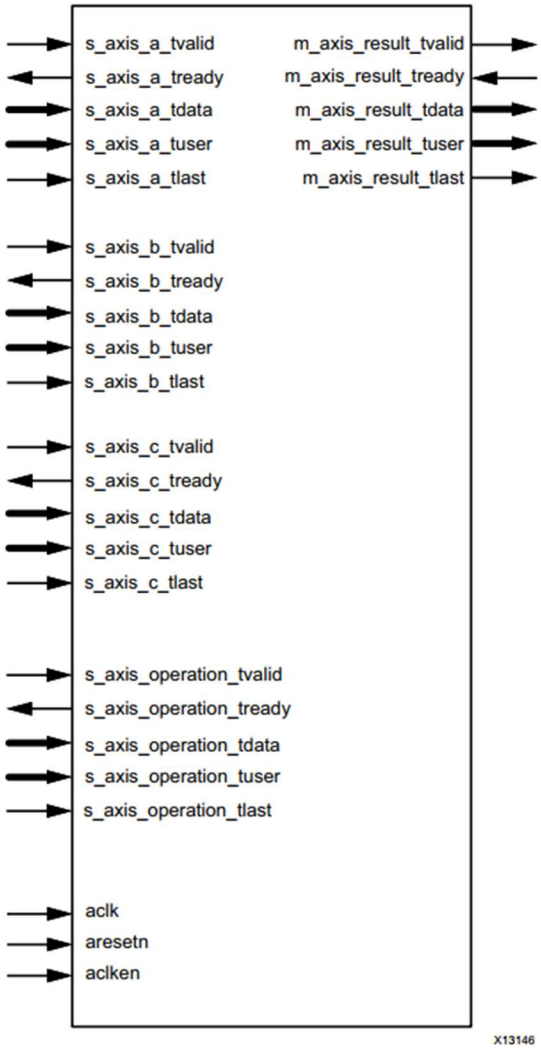


Hình 3.4 Cấu trúc của một số floating point 32-bit

Do mỗi số signle precsion floating có định dạng 32-bit nên chỉ cần thay đổi kiểu dữ liệu của buff (Bảng 3.1 và Bảng 3.2) và lấy dữ liệu trực tiếp cần ghi/đọc vào trong buff.

3.2.2. Thực nghiệm với bộ tách biên số floating point

Đối với các phép tính trên số floating point nhóm sử dụng các IP có sẵn của vivado với chuẩn giao tiếp bắt tay blocking (dữ liệu sẽ được capture hoặc get khi và chỉ khi cả hai tín hiệu valid và ready được kích hoạt) (Hình 3.6).



Hình 3.5 Lõi IP sơ đồ khối

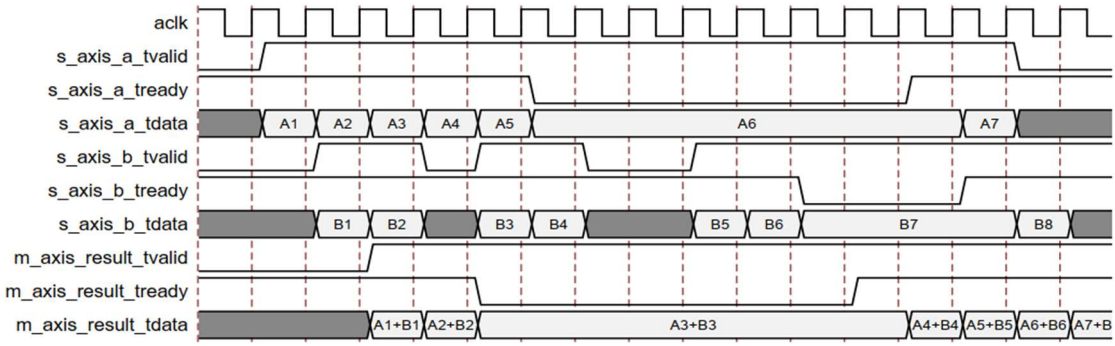
Mỗi Floating Point IP có thể có 3 channel input và 1 channel output (Hình 3.5)

Đối với các channel input sẽ có các chân cơ bản sau:

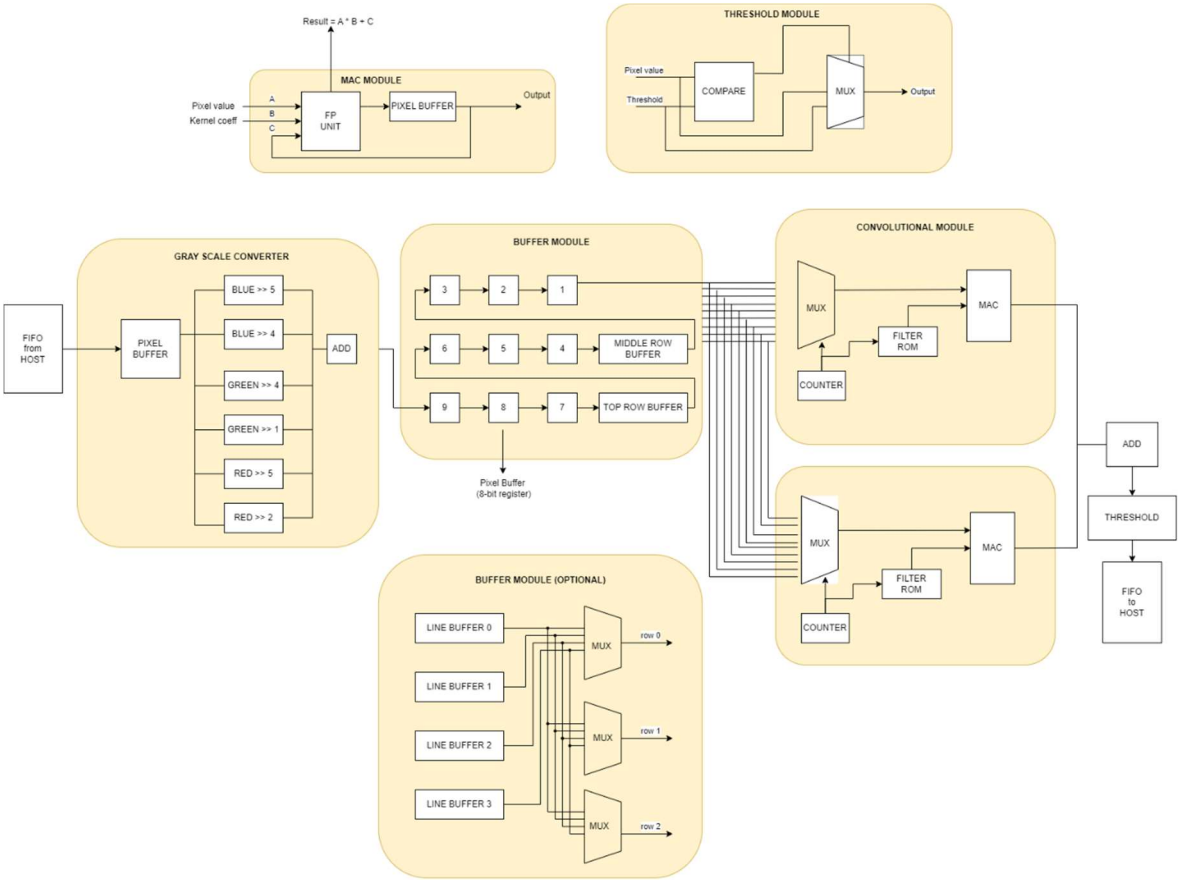
- tvalid: tín hiệu đầu vào cho biết dữ liệu đầu vào đã sẵn sàng.
- tready: tín hiệu đầu ra cho biết lõi IP sẵn sàng cho dữ liệu mới.
- tdata: dữ liệu đầu vào.
- tuser: tín hiệu đầu vào (tùy mục đích sử dụng của người dùng, tín hiệu này sẽ được shift theo dữ liệu đầu vào).
- tlast: tín hiệu thông báo đây là dữ liệu đầu vào cuối cùng (dùng để reset đối với mạch cộng dồn).

Đối với channel output sẽ có các chân cơ bản sau:

- tvalid: tín hiệu đầu vào cho biết lõi IP đã tính toán xong và dữ liệu đầu ra đã sẵn sàng.
- tready: tín hiệu đầu vào cho biết khối tiếp theo yêu cầu dữ liệu mới.
- tdata: dữ liệu đầu ra.
- tuser: tín hiệu đầu vào (tùy mục đích sử dụng của người dùng, tín hiệu này sẽ được ra cùng dữ liệu đầu vào cùng lúc).
- tlast: tín hiệu thông báo đây là dữ liệu đầu ra cuối cùng.



Hình 3.6 Blocking mode



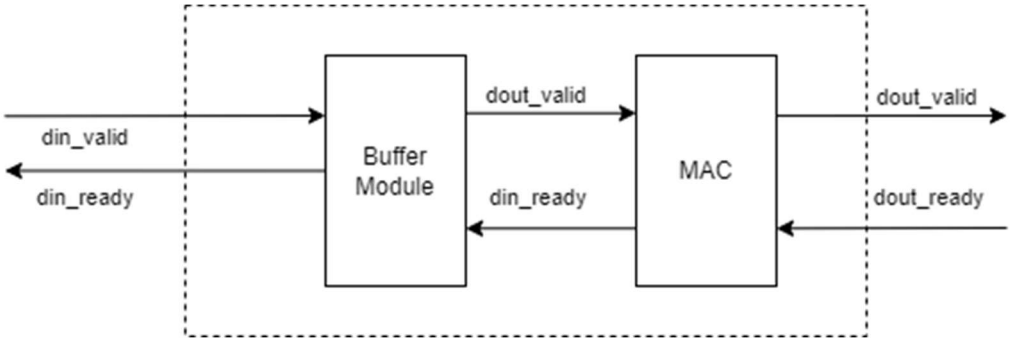
Hình 3.7 Kiến trúc của mạch tách biên cơ bản

Dữ liệu sẽ được nhận từ các DMA đến FIFO từ đó đẩy sang khối gray-scale và được chuyển thành dạng floating-point. Từng pixel lúc này sẽ được lưu trữ trong các line buffer và nhiệm vụ của khối buffer lúc này là lấy các điểm dữ liệu nằm trong cửa sổ trượt từ đó gửi dữ liệu sang khối tích chập để trích xuất đặc trưng và gửi lại FIFO để đây vào DMA cho host (Hình 3.7).

Hình 3.8 Kết quả của mạch tách biên khi chạy thực nghiệm

3.3. Mạng tích chập

Dựa trên thiết kế của line buffer và buffer module từ mạch tách biên. Nhóm phát triển thành các khối hoạt động độc lập (không cần controller bên ngoài) và thêm vào các parameter để kiểm soát kích thước đầu vào, padding, stride và kernel size. Sau đó sẽ kết nối bộ line buffer với bộ MAC (Multiple-Accumulator) để tạo thành một khối tích chập (Hình 3.9)



Hình 3.9 Khối convolution

Chapter 4. ĐÁNH GIÁ KẾT QUẢ

Nhóm đã thiết kế hoàn thiện hầu hết các khối chức năng thành phần của một mạng CNN đơn giản cũng như các ứng dụng phần mềm liên quan phục vụ cho việc giao tiếp giữa cả hai phần PS và PL. Các module hoạt động độc lập và có khả năng pipeline.

Chapter 5. HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN

5.1. Hạn chế

- Các module vẫn còn ở mức đơn giản và chưa được tối ưu về mặt tài nguyên phần cứng.
- Vẫn chưa tổ chức hoàn thiện được 1 ứng dụng phần mềm lớn để quản lý toàn bộ hệ thống.
- Vẫn chưa xử lý được trên định dạng video (ngoài trừ việc loopback dữ liệu).
- RTSP-simple-server vẫn chưa gọi được trong các ứng dụng C.

5.2. Hướng phát triển

- Hoàn thiện và tối ưu phần cứng của cả hệ thống CNN.
- Cấu trúc lại các ứng dụng phần mềm thành một phần mềm lớn và xây dựng một website điều khiển online (streaming video thông qua RTSP được gọi từ trong ứng dụng C).
- Khắc phục tình trạng không xử lý được trên video.

TÀI LIỆU THAM KHẢO

[1] Xillybus Product Brief

[2] Xillybus Getting Started Zynq

[3] Xillybus FPGA API

[4] David Gschwend, ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network, arXiv:2005.06892, 2005

[5] Quan Zhang, Jian Cao, Shiguang Zhang, Qi Zhang, Ying Zhang, Yuan wang, Efficient FPGA Implementation of softmax Layer in Deep Neural Network, 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), 2020

[6] Floating-point operator v7.1 LogiCORE IP Product Guide