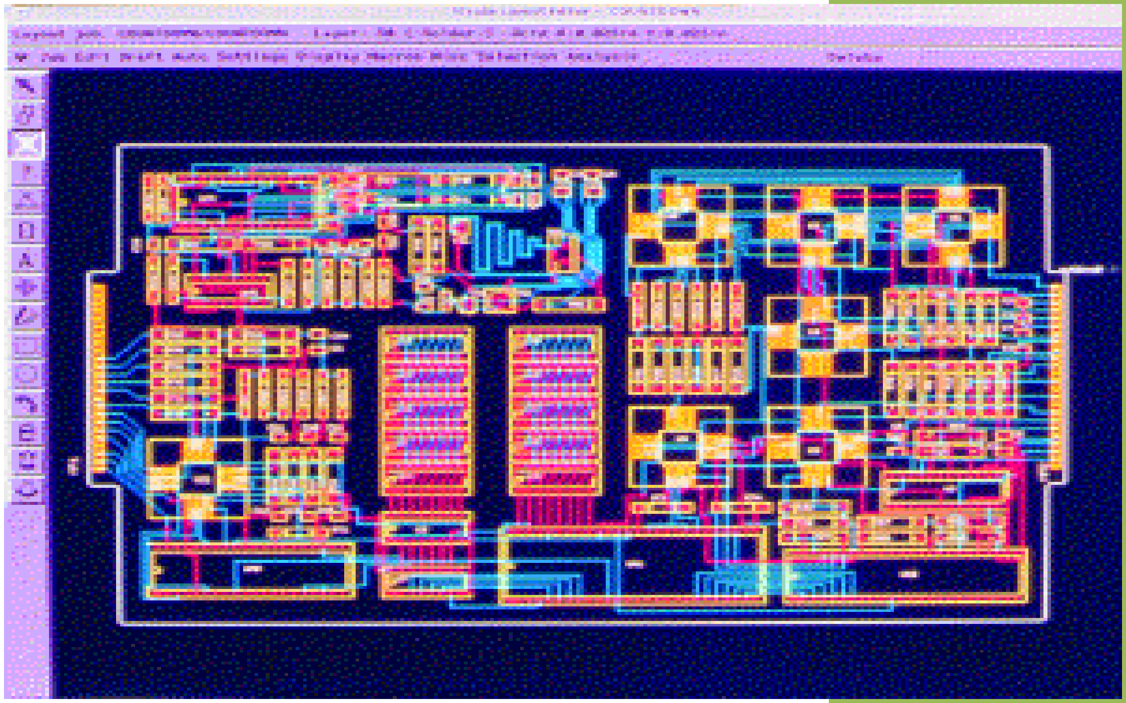
Computer Engineering Department
Faculty of Engineering
Cairo University

Spring 2021
CMP 305
VLSI

DCNN Accelerator CNN Team

2021



VLSI Semester Term Project
CNN Team

VLSI Project

Computer Engineering Department

Objectives

- To better understand Digital Design Flow
- To be proficient at VHDL
- To create real world hardware applications
- To understand the mapping between Algorithm Specifications & Hardware Implementation
- To understand Design Trade-offs
- To learn how to optimize Hardware Designs

Introduction

In real world applications, Convolution neural networks (CNN) achieved a great success in analyzing images. CNNs achieved 99.2% accuracy in classifying the human written digits. In this project, you will design a detailed low-level design of a chip that applies a CNN classifier over a grayscaled image (MNIST handwritten digits dataset). You will design & implement the Convolution/Pooling HW logic for the CNN chip. Additionally, you will provide the needed control to apply 2 sizes of windows (3x3 & 5x5 filters).
As you will see next section, convolution operation is very very costy. So it is required to design the mentioned chip to help the CPU completing the process fast. You will also experience the whole cycle of Design & Fabricating a system on chip, best described as in Figure 1.

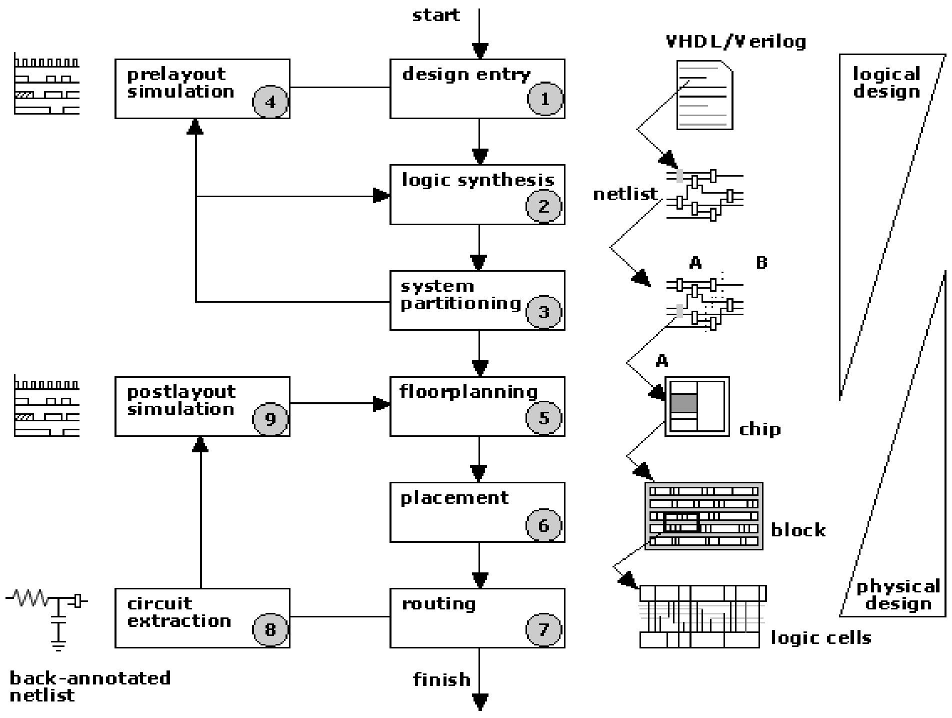


Figure 1: ASIC Chip Design Flow

CNN Background

Discrete convolution for images basically takes two inputs, a matrix, and a filter where the size of the filter is less than the size of the input matrix. I.e. image (matrix) may be 256x256 and the filter may be 3x3.

Put the center of the filter on the first pixel of the image. Multiply filter values by the covered area of the matrix element wise. Sum over the result to get 1 pixel value in the output image. The 1 pixel output is then passed over a ReLu activation function (out=x if x>0 else out=0). Repeatedly, move the filter and make the same steps over and over until you file the output matrix.

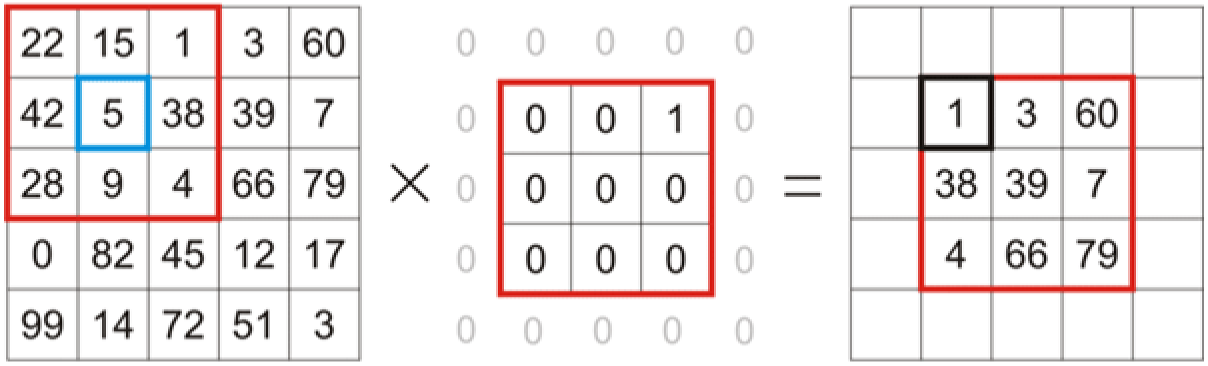


Figure 2-1: one Convolution step

CNNs contains multiple convolution layers, and each layer applies a specific filter. And instead of human crafted filters like Sobell filter, and Canny. Neural network tries mathematically to optimize the filter weights.

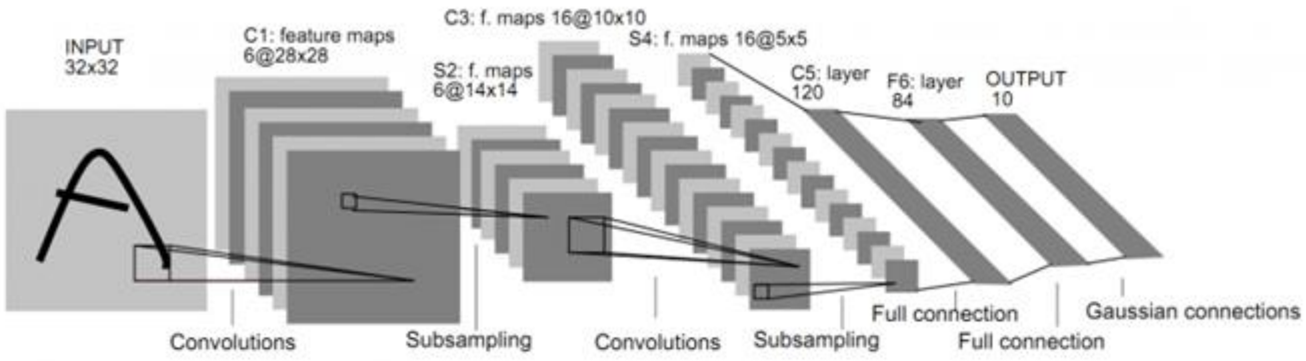


Figure 2-2: Full Convolution Neural Network

Design Requirements

You are required to build a detailed hardware design for a Convolution/Pooling (CNN) module. The system is built for applying **fixed-point** operations (multiplication/addition/shifting) in order to apply a CNN convolution/pooling layers. The CNN module (green box) overview is shown in figure 3. The module is designed to compute a single filter step at a time (ex: convolute a 5x5 filter over a 5x5 window in parallel).

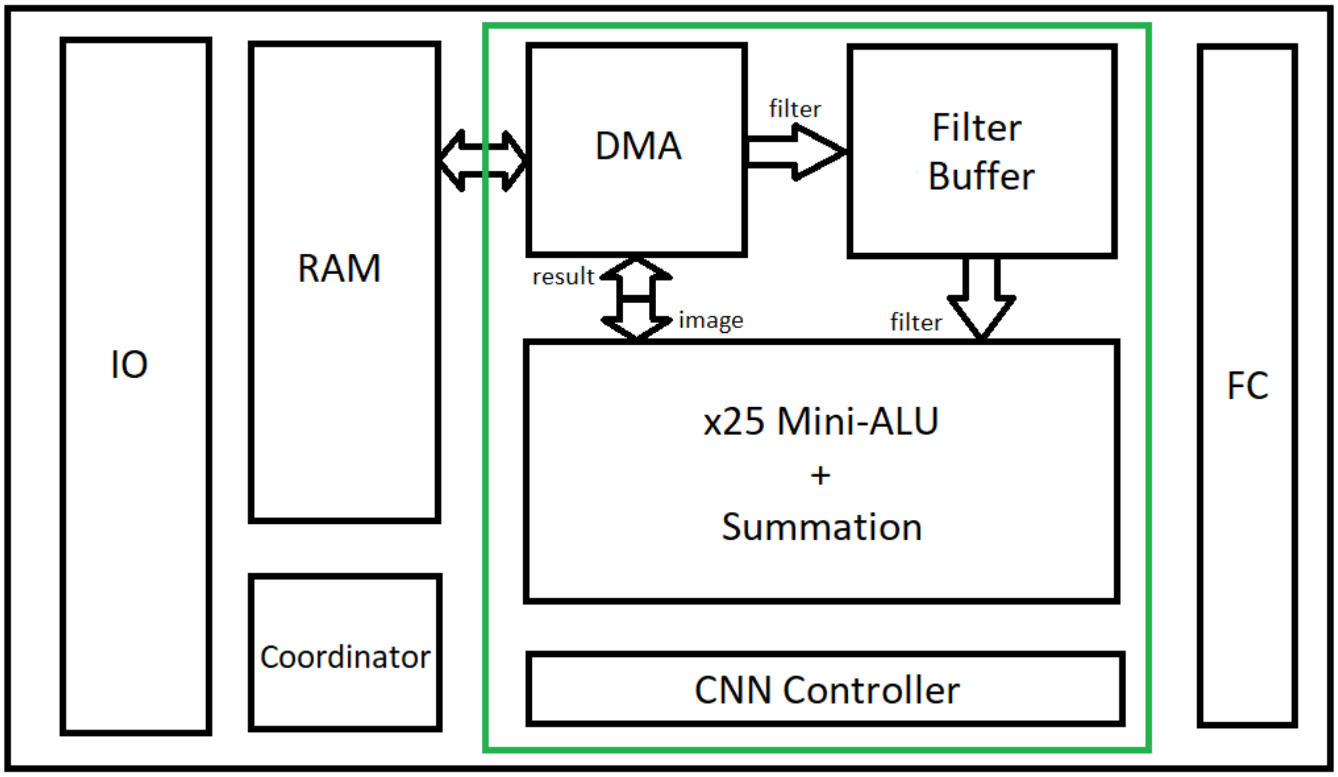


Figure 3: A simplified system design

Note that the RAM/Coordinator are common modules that is used-by/control the 3 main modules in the chip, so you need to coordinate with the rest of the team.
Take in consideration that your design will be implemented in VHDL and synthesized in the next phases. So try to make the design as clear as possible to easily implement it. Prior optimization is a plus.

CNN Scenario

Apple company has assigned your team for designing & implementing DCNN accelerator to be added into IPhoneX12. The accelerator operates on grayscale images of size 28x28 pixels (MNIST dataset images). The IO module loads the required image into a specific part in the RAM.

After the image & CNN filters are loaded in the RAM (by IO module), the coordinator will send a start signal to the CNN module along the number of layers & their sequence. Your design will process each layer as follows:

1. Loop on the number of Layers
    a. Check if the layer is Convolution or Pooling
    b. If the layer is convolution
        i. Load the filter weights
    c. Else do nothing
    d. Load the layer input from the RAM (Image or intermediate result)
    e. For each window in the image
        i. If the layer is convolution apply convolution algorithm
        ii. Else apply pooling algorithm
        iii. Save the result in the RAM (intermediate result)
2. After finishing all layers send done signal to the coordinator

Note that the multiplication & summation will need a predefined data/result size. (ex: 3-bit * 3-bit will generate a 6-bit output). So you need to use a fixed size precision for your fixed-point operations (ex: the input is 32-bit & the output is also 32-bit, and larger data will be trimmed).

In the following section, you will find the definition and details for the operations applied (convolution & pooling), and the HW logic behind multiplication & fixed-point operations.

Convolution Algorithm

Convolution operation in DCNN is essentially a dot product between 2 matrices Filter & Window.

```
1) Sum = 0
2) For each row i
    a) For each col j
        i) Sum = Sum + (Filter[i,j] * Window[i,j])
3) Result = Max(Sum, 0)
```

Pooling Algorithm

```
1) Sum = 0
2) For each row i
    a) For each col j
        i) Sum = Sum + Window[i,j]
3) If Filter_Size == 3x3
    a) Result = Sum >> 3            (Where ">>" is Arithmetic Shift Right)
4) Else If Filter_Size == 5x5
    a) Result = Sum >> 5
```

Radix-2 Booth's Multiplication Algorithm

Binary multiplication uses shifting operations to compute the multiplication of 2 n-bit **signed** numbers. It could be implemented in a single-cycle (Shift-Add Approach) or multi-cycle (Booth's algorithm) approaches. **You will use Radix-2 Booth Algorithm in your mini ALUs**.

```
1) Assuming that the 2 operands are called M and R
2) Set values for A, P, S
   Where A is a (2n+1) bits & the highest n bits are set to be M (ex: n=4 A="MMMM00000")
   And S is a (2n+1) bits & the highest n bits are set to be -M (ex: n=4 S="(-MMMM)00000")
   And P is a (2n+1) bits & the bits from (1 to n) are set to be R (ex: n=4 P="0000RRRR0")
3) Loop for N iterations:
    a) Check the Least Significant (Rightmost) 2 bits in P
        i) If LSB == "00" or "11"
            (1) Do Nothing
        ii) If LSB == "01"
            (1) P = P + A
        iii) If LSB == "10"
            (1) P = P + S
    b) Arithmetic Shift P 1 bit to the right
4) Result = P[1 to 2n]      (Ignore the least significant bit)
```

Fixed Point Arithmetic

Fixed points is a binary representation for fraction numbers. The integer part is normal binary $\{2^4 2^3 2^2 2^1 2^0\}$ (increasing powers of 2). The fraction part is a decreasing powers of 2 $\{2^{-1} 2^{-2} 2^{-3} 2^{-4}\}$.

i.e: 1011(-2) ⇒ the number in the register is 1011 & the fraction is the least 2 bits

10.11

$= 1*2^1 + 0*2^0 + 1*2^{-1} + 1*2^{-2}$

$= 2 + 0 + ½ + ¼ = 2.75$

Normal binary arithmetics (as addition) are applied on fixed point number of same precision (number of bits in the fraction part), however; shifting is needed for different precision numbers.

i.e: 1.125 + 3.5 = 4.625
== 1001(-3) + 111(-1)
== 1001(-3) + 11100(-3)
== 01.001 + 11.100 = 100.101

Additionally, The multiplication and shift operations can be used similarly to the binary operations by adjusting the point position.

i.e: 1.125 * 3.5 = 3.9375
== 001.001 * 011.100 = 000011.111100
== 1001(-3) * 11100(-3) = 111111(-6)
3-bits for integer + 3-bits for fraction ⇒ 6-bits for integer + 6-bits for fraction

References for more Informations

1. https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm
2. https://en.wikipedia.org/wiki/Binary_multiplier
3. http://www.geoffknagge.com/fyp/booth.shtml
4. http://www.ecs.umass.edu/ece/koren/arith/simulator/Booth/
5. https://en.wikipedia.org/wiki/Convolutional_neural_network
6. http://cs231n.github.io/convolutional-networks/
7. http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html
8. http://aishack.in/tutorials/image-convolution-examples/
9. https://en.wikipedia.org/wiki/Fixed-point_arithmetic
10. https://spin.atomicobject.com/2012/03/15/simple-fixed-point-math/
11. LeNet-5 Python Implementation https://github.com/sujaybabruwad/LeNet-in-Tensorflow
12. TensorFlow CNN Tutorial https://tf-lenet.readthedocs.io/en/latest/tutorial/

Rules & Regulations

- CNN subteam is 3-4 members.
- You are free to design your system as you want as long as you perform the required functionality.
- You could add any additional I/O ports and modules according to your need, but you have to justify your design choices.
- Your design should be modifiable to meet the design constraints in the next phases.
- Your design should be integratable with the rest of the bigger team.
- Take care that your design is logically mappable to hardware or you will have to repeat it all again.
- Open your mind and don't limit yourself .
- You are not allowed to copy from any external resources in your implementation .
- You are allowed to consult external resources for Design but Do your OWN & you have to fully understand it.
- **Grades are based Mainly on Individual work + your team work . if you didn't work and the project was complete you will still get a zero grade. And we really mean it.**
- The document is variable to change with a previous notification.

Deliverables

- Hardware design documents:
  - Your names
  - Detailed units architecture with connections.
  - Detailed subunits design in the well-known logic gates level (counters / adders / ...).
  - Design assumptions and limitation
- SW script that simulates the CNN result