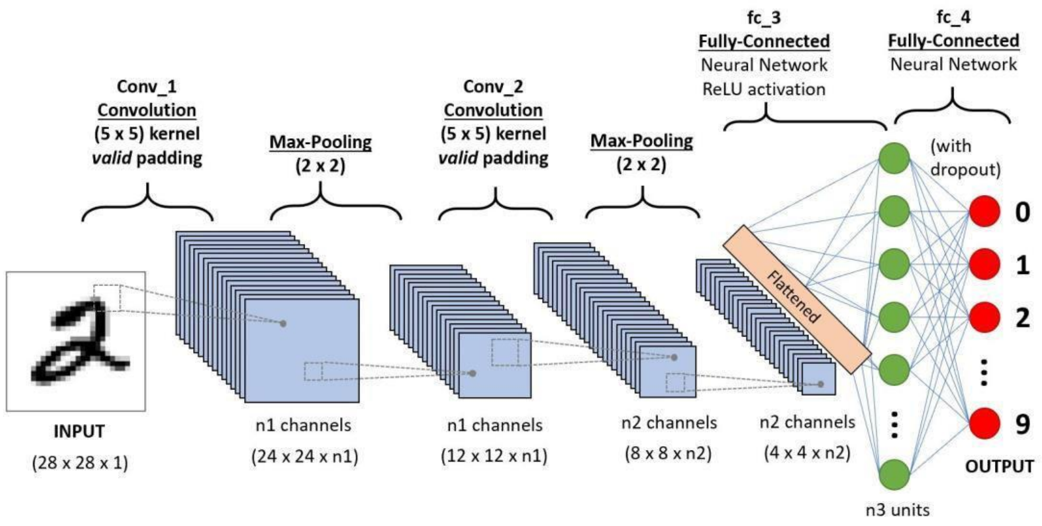
Group No.	35		
Name	Aakash Kumar	**Roll Number**	160108048
Name	Gaurav Israni	**Roll Number**	160108042
Name	Kevin Jose	**Roll Number**	160108047
Supervisor(s)	Dr. Shaik Rafi Ahamed		
Project Title	Efficient hardware implementation of Convolutional Neural Networks		

Introduction with Functional Block Diagram

Now a days, Convolutional Neural Networks(CNNs) has emerged as a reliable and powerful tool for deep learning and has various real life applications like Real Time Facial Recognition, Automatic Game playing, etc. To ensure the programmable flexibility and shorten the development period, field programmable gate array(FPGA) is appropriate to implement the CNN models.



In a CNN model, there are several different layers connected in feed-forward pattern, so as to generate a weight assigned to each possible output value, in the final layer i.e. at the output CNN model.

Different layers have different significance and play vital role in generating the overall result with maximum accuracy. Various layers in a CNN model are as follows:-

- **Convolutional Layers**
- **Pooling Layers**
- **Fully Connected Layers**

Normally, pooling layers follow convolutional layers and fully connected layers are the last few layers.

In this project, we are keeping our focus on implementing convolution layer efficiently since convolution is the most computationally demanding part and approximately 90% of the computation time is spend in convolution layer only.

The theory of parallel **fast finite impulse response algorithm** (FFA) is used to implement convolutions. Based on FFAs, the corresponding **fast convolution units** (FCU's) are developed for the computation of convolutions in the CNN models.

To process the convolutions, a **Processing Unit(PU)** is employed which is nothing but a grid of FCU's.

Since implementation of single FCU on breadboards is taking a lot of space and components and ICs, in this project, we are implementing one of the **FCU on breadboard**(for demonstration) and rest are implemented using **FPGA kit on Zedboard.**

Discussions on Design with Illustrations; Circuit Connections

In the overall CNN Model ,convolution layer dominates over all other layers and takes maximum computation time as compared to all other layers.Reducing the convolution time would always result in much more faster version of CNN. Given the previous fact, there will be a trade- off between convolution time and system complexity and size, but to align our result with lesser convolution time and system complexity, we are implementing Fast Convolution Algorithm (FCA) which implements Fast FIR Algorithm (FFA) for computing the convolutions faster.

- ***Parallel Fast FIR Algorithm***

An FIR filter with N taps can be expressed as:

$$y(n) = h(n) * x(n) = \sum_{i=0}^{N-1} h(i)x(n-i),$$
$$n = 0, 1, 2, \cdots, \infty,$$

where x(n) is an infinite input sequence and h(n) contains the coefficients of an N-length FIR filter. The convolution shown in can be expressed in z domain:

$$Y(z) = H(z)X(z) = \sum_{n=0}^{N-1} h(n)z^{-n} \sum_{n=0}^{\infty} x(n)z^{-n}$$

We can decompose the input sequence into even and odd parts and hence we have, we have $X(z) = X_0 + z^{-1}X_1$.

Similarly, $H(z) = H_0 + z^{-1}H_1$ and the output :

$$Y(z) = Y_0 + z^{-1}Y_1 = (X_0 + z^{-1}X_1)(H_0 + z^{-1}H_1),$$

where

$$Y_0 = H_0X_0 + z^{-2}H_1X_1,$$
$$Y_1 = H_1X_0 + H_0X_1.$$

Based on the fast FIR algorithm (FFA), Y0 and Y1 can be expressed in the form of 2-parallel FFA,

$$Y_0 = H_0X_0 + z^{-2}H_1X_1,$$
$$Y_1 = (H_0 + H_1)(X_0 + X_1) - H_0X_0 - H_1X_1.$$

We can decompose the input sequence in 3 parts also :

$$\begin{aligned} Y &= Y_0 + z^{-1}Y_1 + z^{-2}Y_2 \\ &= (X_0 + z^{-1}X_1 + z^{-2}X_2)(H_0 + z^{-1}H_1 + z^{-2}H_2) \\ &= (X_0 + z^{-1}V)(H_0 + z^{-1}W), \end{aligned}$$

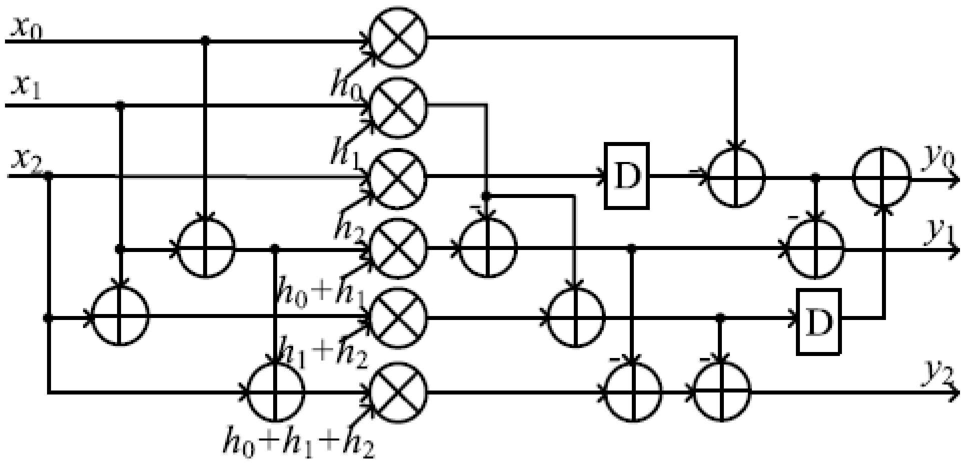
Where $V = X_1 + z^{-1}X_2$ and $W = H_1 + z^{-1}H_2$.

The last line of above equation and its intermediate term VW are both in the 2-parallel FIR filter form and can be computed by employing above equation recursively. Hence the 3 parallel FFA can be computed as follows:

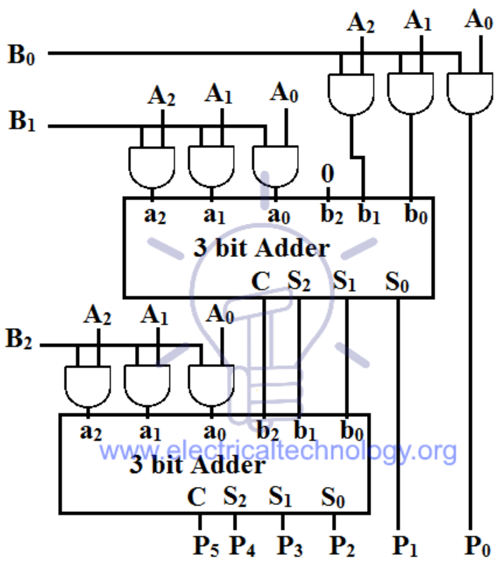
$$\begin{aligned} Y_0 &= H_0X_0 - z^{-3}H_2X_2 + z^{-3}[(H_1+H_2)(X_1+X_2) - H_1X_1 \\ Y_1 &= [(H_0v+H_1)(X_0+X_1) - H_1X_1] - [H_0X_0 - z^{-3}H_2X_2] \\ Y_2 &= [(H_0 + H_1 + H_2)(X_0 + X_1 + X_2)] \\ &\quad - [(H_0 + H_1)(X_0 + X_1) - H_1X_1] \\ &\quad - [(H_1 + H_2)(X_1 + X_2) - H_1X_1]. \end{aligned}$$

- ***Fast Convolution Unit***

Since we are convoluting a 2D(k x k) kernel with a 2D input vector, the 2D kernel will be segregated into K 1D convolutions and each input vector row will be convoluted will be convoluted with corresponding kernel row. Each 1D convolution can be implemented by a k-tap FIR filter, where the tap coefficients are the corresponding kernel weights. Based on the parallel FFA, a Fast Convolutional Unit (FCU) is proposed to perform the convolution in a CNN.



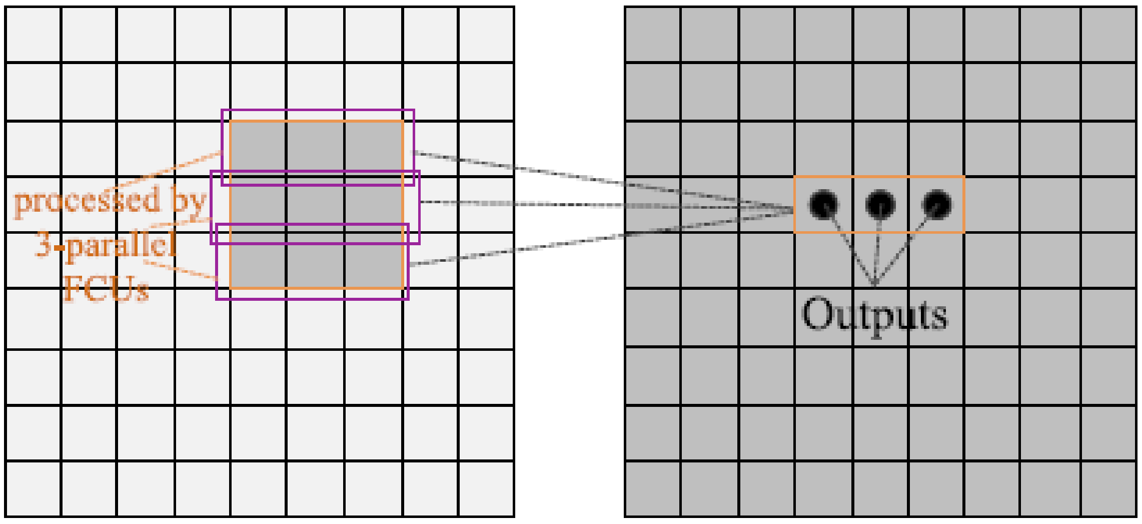
We are using 3 x 3 kernel, and a 4 x 3 input vector as the inputs to the FCU unit, the coefficients h0, h1, h2 denote the kernel weights, x0, x1, x2 and y0, y1, y2 are inputs and outputs, respectively. They all belong to one row or column in a 2D field. Note that we have reversed the kernel weights before multiplying with the corresponding row elements since a linear convolution input vector and convoluting vector grow in opposite order/directions.



Schematic of 3x3 Multiplier
Using 3-Bit Full Adder

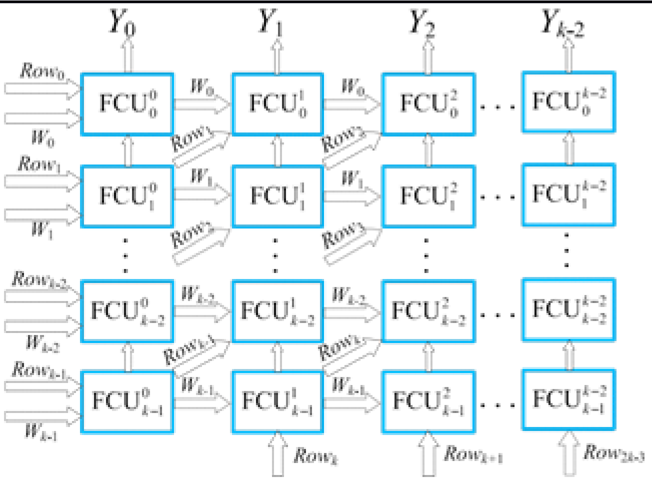
Since the 3-bit multiplier ICs(74284 and 74285) were not available, we have implemented the multiplier using 4 bit parallel full adders keeping the MSBs of every full adder inputs zero.

In order to finish a 2D convolution, k identical FCU's are employed. The outputs of all FCU's are added together.



The 2D convolution with three 3-parallel FCU's are illustrated in above figure

In the processing unit, each row is convoluted with corresponding kernel row.For each FCU unit corresponding to same column, the outputs are added and thus the output vector is generated.



Depiction of Processing Unit(PU) using a grid of FCU

Experiments and Results; Performance Analysis

The observation results for 1 FCU unit as observed on the circuit implemented on breadboard are as follows:

First Observation:

x[n] -> [0 0 7] =>

$x[0] = (0)_{10} = (000)_2$;
$x[1] = (0)_{10} = (000)_2$;
$x[2] = (7)_{10} = (111)_2$

h[n] -> [1 2 3] =>

$h[0] = (1)_{10} = (001)_2$;
$h[1] = (2)_{10} = (010)_2$;
$h[2] = (3)_{10} = (011)_2$

y[n] -> [14 21 7

] =>

$y[0] = (14)_{10} = (001110)_2$;
$y[1] = (21)_{10} = (010101)_2$;
$y[2] = (7)_{10} = (000111)_2$;

Second Observation:

x[n] -> [1 3 3] =>

$x[0] = (1)_{10} = (001)_2$;
$x[1] = (3)_{10} = (011)_2$;
$x[2] = (3)_{10} = (011)_2$

h[n] -> [3 2 2] =>

$h[0] = (3)_{10} = (011)_2$;
$h[1] = (2)_{10} = (010)_2$;
$h[2] = (2)_{10} = (010)_2$

y[n] -> [15 12 17] =>

$y[0] = (15)_{10} = (001111)_2$;
$y[1] = (12)_{10} = (001100)_2$;

$y[2] = (17)_{10} = (010001)_2$;

The observation results for PU unit as observed on the circuit implemented on breadboard and FPGA are as follows:

X[n] = [1 2 3 ; 4 5 6 ; 7 1 2 ; 1 2 1] (This ia a (3 * 4) 2D input vector) =>

$x[0][0] = (1)_{10} = (001)_2$;
$x[0][1] = (2)_{10} = (010)_2$;
$x[0][2] = (3)_{10} = (011)_2$
$x[1][0] = (4)_{10} = (010)_2$;
$x[1][1] = (5)_{10} = (101)_2$;
$x[1][2] = (6)_{10} = (110)_2$
$x[2][0] = (7)_{10} = (111)_2$;
$x[2][1] = (1)_{10} = (001)_2$;
$x[2][2] = (2)_{10} = (010)_2$
$x[3][0] = (1)_{10} = (001)_2$;
$x[3][1] = (2)_{10} = (010)_2$;
$x[3][2] = (1)_{10} = (001)_2$

W[n] = [1 0 2 ; 4 1 1 ; 3 1 2] (This is a (3 * 3) 2D kernel) =>

$w[0][0] = (1)_{10} = (001)_2$;
$w[0][1] = (0)_{10} = (010)_2$;
$w[0][2] = (2)_{10} = (010)_2$
$w[1][0] = (4)_{10} = (100)_2$;
$w[1][1] = (1)_{10} = (001)_2$;
$w[1][2] = (1)_{10} = (001)_2$
$w[2][0] = (3)_{10} = (011)_2$;
$w[2][1] = (1)_{10} = (001)_2$;
$w[2][2] = (2)_{10} = (010)_2$

Y[n] = [39 35 47 ; 57 31 53] (This is the desired output of PU) =>

$y[0][0] = (39)_{10} = (100111)_2$;
$y[0][1] = (35)_{10} = (100011)_2$;
$y[0][2] = (47)_{10} = (101111)_2$
$y[1][0] = (57)_{10} = (111001)_2$;
$y[1][1] = (31)_{10} = (011111)_2$;
$y[1][2] = (53)_{10} = (110101)_2$

Implementation Results

We went ahead and implemented this with our current results.
The build took around one week of high precision calculations and a lot of breadboards and wires.
And finally we were able to implement 3 nodes.
We analyzed the performance and was able to achieve around 40% faster calculation for a single PU.

Summary and Future Work

In this project, we focus on the efficient hardware designs for convolution layer of CNN model. Convolutions dominate the computation complexity. The fast FIR algorithm (FFA) is introduced and Fast Convolution Units (FCU's) are developed based on FFAs. By employing the parallel FCU's in the convolutions of CNN models, the computation complexity and energy are cut down significantly.

Having implemented the convolution layer using minimum possible number of ICs, our future target is to implement pooling layers and fully connected layers efficiently.
ALso we will be heading forward to reduce down the Latency Rate in information transfer be it interlayer information transfer or intralayer information transfer.
Also data transfer rate from external memory or source to input layer and from output layer to sink, need to be improved to make this product of industrial use.