Index

1. Problem Statement

With supported FPGA/ASIC synthesis, simulation results, compare 3-4 distance computation methods in k-nearest neighbor algorithm (kNN) inference hardware design

2. Approach to the problem

1. Understand KNN Algorithm
2. Take a dataset and create a KNN model using existing python libraries
3. Quantize the database to int8 and check the accuracy of the quantized dataset
4. Design and develop KNN algorithm in python without using libraries and check the accuracy.
5. Design the architecture of the KNN in RTL
6. Test RTL using simulation using the same database for python
7. Compare the results

3. KNN Algorithm

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification but can also be used for regression tasks. It works by finding the "k" closest data points (neighbors) to a given input and makes a prediction based on the majority class (for classification) or the average value (for regression). Since KNN makes no assumptions about the underlying data distribution it makes it a non-parametric and instance-based learning method.

3. 1 Inferencing KNN Algorithm

For inferencing the algorithm,

- the dataset used is diabetics database
- K will be fixed as 5

4. KNN Algorithm in Python Using Python Existing Libraries

The KNN algorithm was inferred in python using the existing ML libraries. The code of the same is available in the zip file attached with the document
*Location: **/training/knn_model.py***

The data was analysed properly, the log for the data analysis can be found in
/training/log/data_analysis.log

```
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

By using this database, and using **Manhattan distance about 70.8% accuracy was obtained**.

```
===== DATA SPLIT =====
Training samples : 614
Testing samples  : 154

===== MODEL CONFIGURATION =====
Algorithm         : k-NN
k (neighbors)     : 5
Distance metric   : Manhatten

===== TEST RESULTS =====
Accuracy : 70.13 %
```

4. 1 Quantizing to INT8

All data in the database was quantized to 0 to 255 using python (INT8) and accuracy test was performed. **After quantising, the accuracy was tested using the testing_samples and the accuracy was around 70.8%. (log can be found in /training/log/quantized_log.log)**

```
===== FLOAT MODEL RESULT =====
Accuracy : 70.13 %

===== DATA TYPES AFTER FULL INT8 QUANTIZATION =====
Pregnancies                 int8
Glucose                     int8
BloodPressure               int8
SkinThickness               int8
Insulin                     int8
BMI                         int8
DiabetesPedigreeFunction    int8
Age                         int8
Outcome                     int8
dtype: object

===== INT8 MODEL RESULT =====
Accuracy : 70.78 %
```

5. KNN Algorithm in Python Using Python (Only Using Equations)

KNN Algorithm was modified in python using simple equations rather than using ML libraries to transfer the algorithm to FPGA (RTL). As per the python model, following was the accuracy obtained using different distance methods

```
Evaluating 154 samples with k=5...

Metric          | Accuracy
------------------------------
Manhattan       | 68.83%
Euclidean       | 68.83%
Chebyshev       | 68.83%
```

Average accuracy Obtained: 68.83%

5. 1 Extraction Of Data to RTL

The train data and test data was separated from the original database in 80:20 ratio. So, 614 data will be used for training and 154 samples for testing. These data were converted to hex values in 8 bits for feature and 1 bit for label.

Sl No	Data	Field	Bit Width	No of Samples	Total
1	Train Data	9 8 for features 1 for label	8	614	44,208 bits 5526 bytes **5.5KB**
2	Test Data	9 8 for features 1 for label	8	154	11,088 bits 1386 bytes **1.3KB**

```
Assignment-1 > Training > train_data.r
1   1E6B88481595760D00
2   1E696D392472FF1100
3   4B7A9F46008F408401
4   1E61863E12763E3300
5   00F0AB243681810401
6   5A70A700005C002F00
```

```
Assignment-1 > Training > test_data.
1   0F708A3B1B70130000
2   69A38E7E249B4D5E01
3   69829B671E951A6600
4   00E6A3A203EFFF1101
5   1E75865230997E0801
6   0FA17D0000783A6E01
```

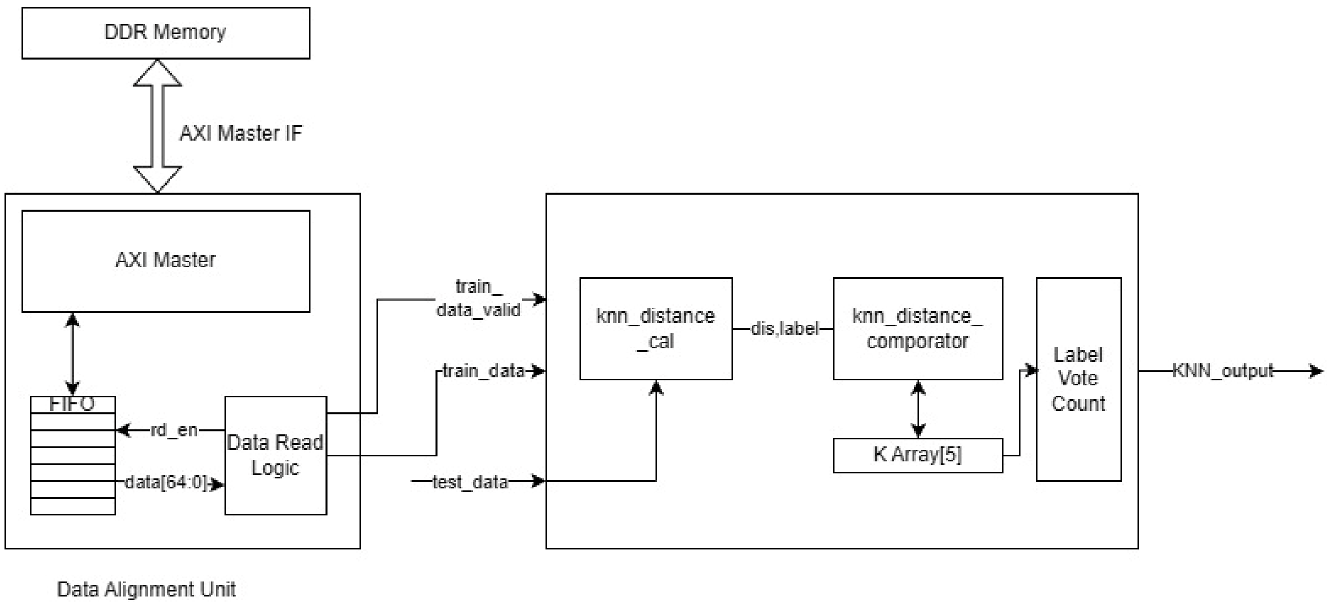
sample data converted to hex

Note: These mem files can be found in /training/data/mem/

6. KNN Accelerator Module Design

The following block diagram explains the module architecture for KNN accelerator

6. 1 Module Architecture



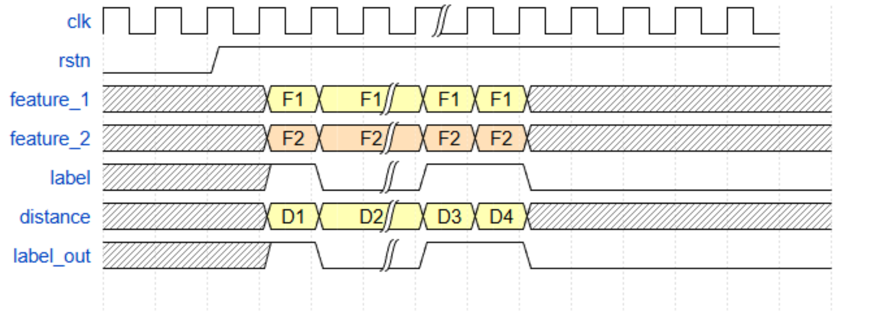
6.1.1 Data-Alignment Unit

This module is responsible for taking data from DDR to the accelerator implemented in FPGA. Currently this is not implemented, instead the training data and test data will be fed into the accelerator module using simulation test bench.

Note: For real implementation and actual implementation of the module, data reception from DDR must be implemented.

6.1.2 KNN Distance Calculator

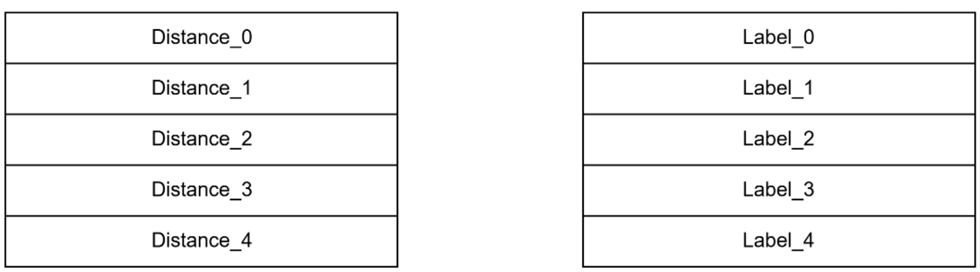
This module will calculate the distance L using the input feature data. After calculating the distance, the calculated distance and label will be fed into the comparator unit. As the calculation is done in combinational logic, the calculated distance will be computed in that clock itself.



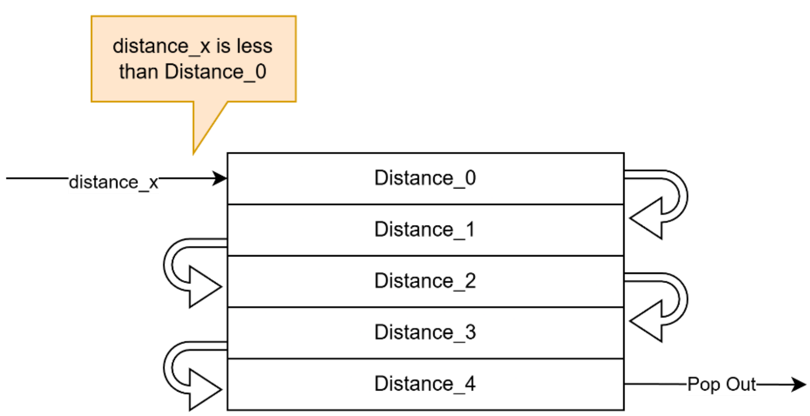
This module can be configured for different distance methods.

6.1.3 KNN Distance Comparator

This module will compare the incoming distance with the current list of distances. As the K=5, we need to get the lowest 5 distances and its label. For that a shift-down register array scheme is implemented.

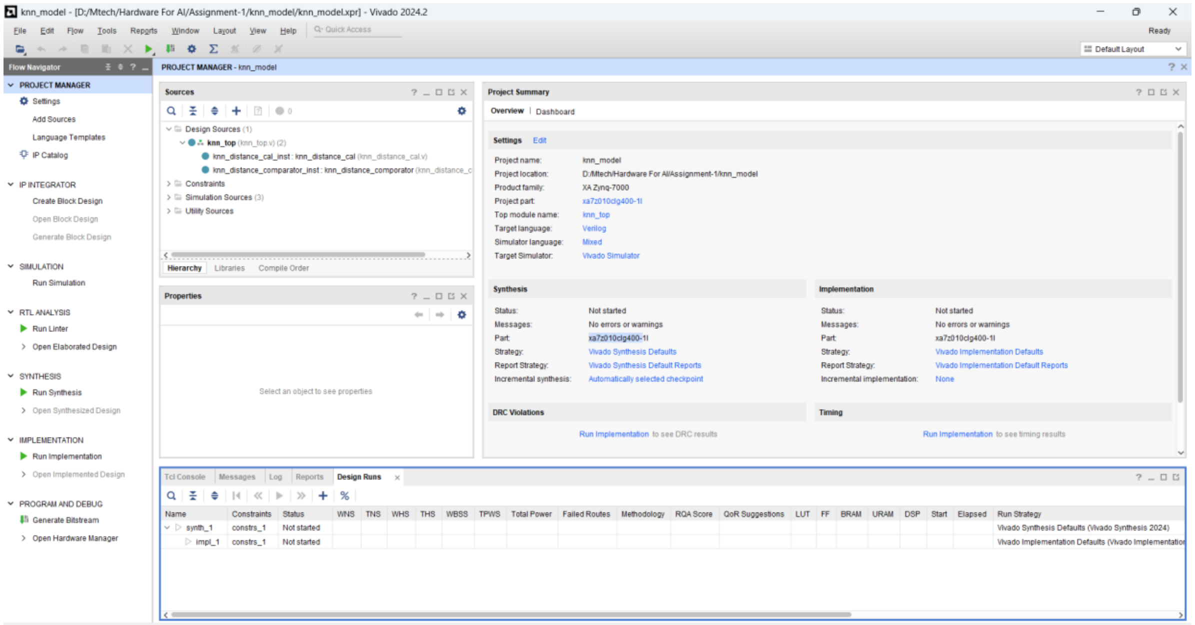


During reset, this will be filled with maximum value (FF). When new distance comes, it will check If its less than the distance_0, if its less than distance_0 then distance_0 will be shifted downwards and new member will be inserted at index 0.



6. 2 KNN FPGA Implementation

The above KNN module was realized in RTL for **FPGA device xa7z010clg400-1l**



The project can be found in /knn_model/knn_model.xpr *(Vivado 2024.2)*

6.2.1 Latency of Accelerator

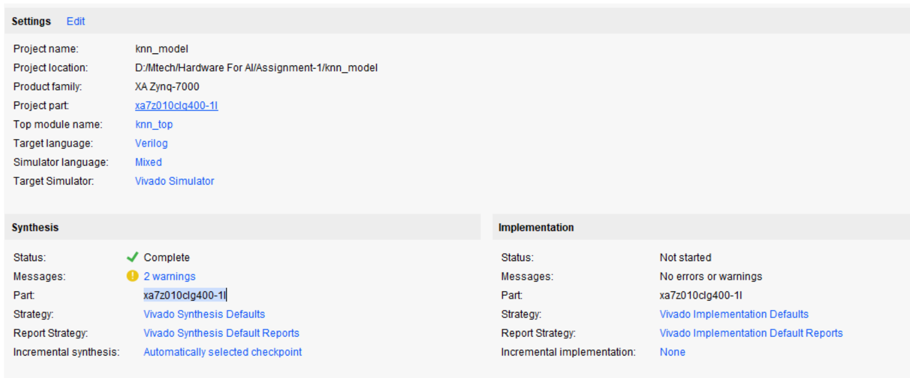
The accelerator module will take N+1 clock cycle to predict the output label where N is the number of train data sample.

6.2.2 Selecting Multiple Distance (L1,L2,L3)

The distance calculation method can be selected using the define (build option) in the module knn_distance_cal.v

7. Synthesis Results

The design was successfully synthesised. Please find the following resource usage for different distances



7. 1 Euclidean Distance

Resource	Utilization	Available	Utilization %
LUT	254	17600	1.44
FF	52	35200	0.15
IO	135	100	135.00

7. 2 Manhattan Distance

Resource	Utilization	Available	Utilization %
LUT	420	17600	2.39
FF	52	35200	0.15
IO	135	100	135.00

7. 3 Chebyshev Distance

Resource	Utilization	Available	Utilization %
LUT	270	17600	1.53
FF	52	35200	0.15
IO	135	100	135.00

8. Simulating KNN Accelerator

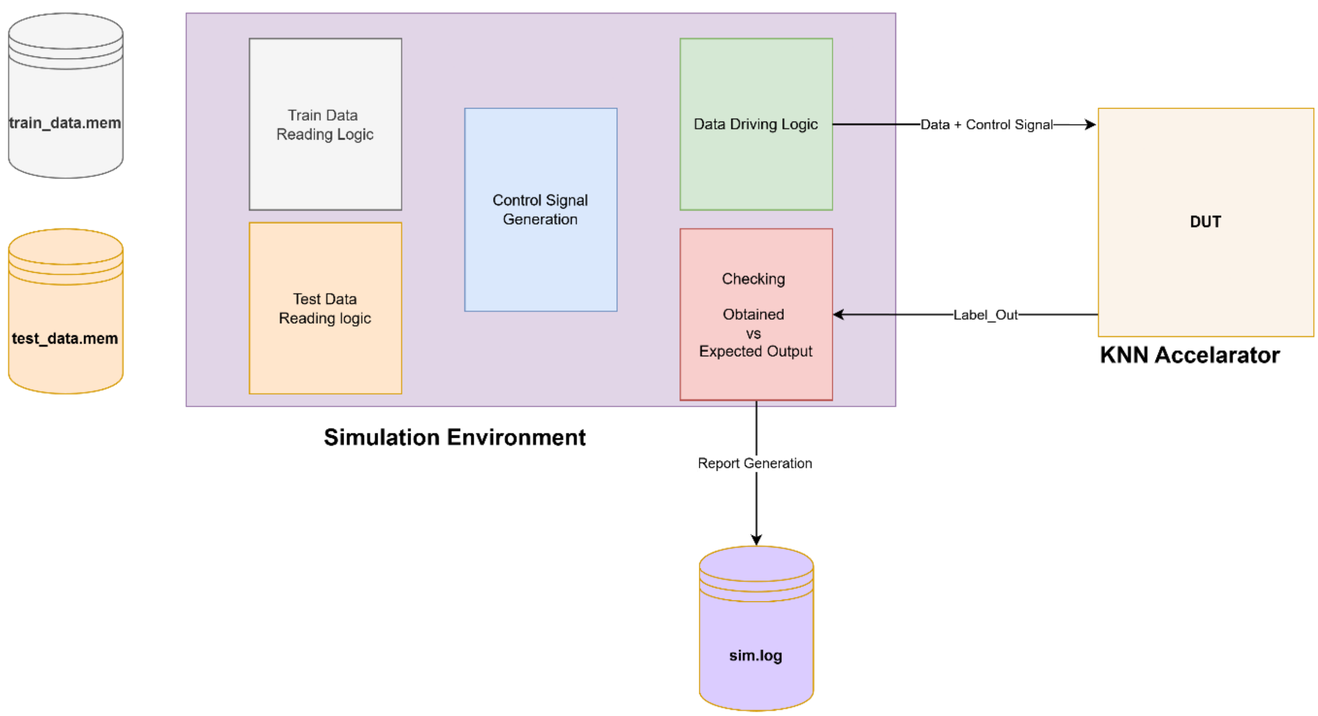
As the data transfer from AXI is not implemented, the simulation will be done by feeding train and test data to the dut from testbench. The simulation will check the final outcome of the RTL and compare it will the label of the test data to check the accuracy.

```

//--------------------+-------------------------------------------------------
// Filename           | knn_distance_cal.sv
// File created on    | 05 Feb 2026
// Created by         | Divine A Mathew
//                    |
//                    |
//--------------------+-------------------------------------------------------
//
//----------------------------------------------------------------------------
// KNN Distance Calculator Module
//----------------------------------------------------------------------------


`define MANHATTAN
// `define EUCLIDEAN
//  `define CHEBYSHEV
```

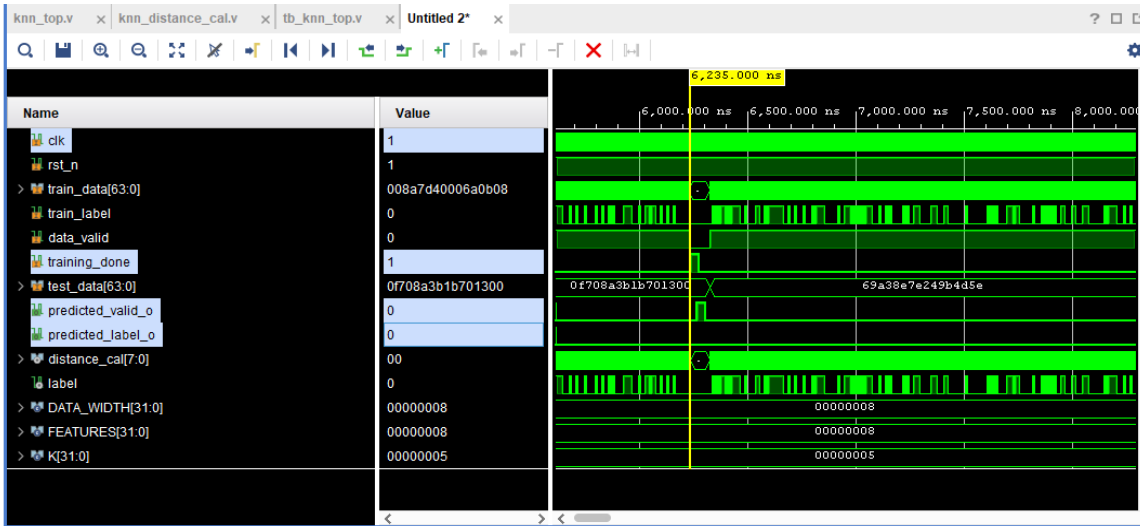
8.1 Simulation Setup



The simulation environment will take **all the train data for each test data**.

9. Simulation Results

The design was subject to simulation using linear testbench and following results was obtained



9. 1 Accuracy

Sl No	Distance Method	Software Accuracy	RTL Accuracy	Simulation log
1	Euclidean	68.3%	72.0%	\knn_model\src\tb\logs\euclidean_sim.log
2	Manhattan	68.3%	72.0%	\knn_model\src\tb\logs\manhattan _sim.log
3	Chebyshev	68.3%	74.0%	\knn_model\src\tb\logs\chebyshev_sim.log

9. 2 Latency

The time required by the **processor** to determine the **output is 0.125s**. This is time taken by a 12th Gen Intel(R) **Core(TM) i5-12450H**, 8 Core(s), 12 Logical Processor clocked at **2.0GHz**

In case of **FPGA design**, the total time taken from first input to final output data is **N+2 clock** cycle where **N is the no. of training sample**.

Here N = 614, Clock Frequency = 200Mhz
Total clock cycle = N+2 = 614 + 2 = 616
Time = clock period * number of cycles = 5ns * 616 = **3080 ns or 3.08us**

Design Inferred	Latency
Software	0.125s
Hardware	3.08us