Technion – Israel Institute of Technology

The Andrew and Erna Viterbi Faculty of Electrical & Computer Engineering

VLSI Laboratory

Semester Winter 2024-2025

KNN Classifier Accelerator

Authors:

Roy Rapaport, 209149996

Dan Katz, 314628496

Supervised by: Goel Samuel

This project was done in collaboration with Apple Israel.

Abstract

In today's world, computers and humans process vast amounts of data, much of which must be classified to predict trends and make informed decisions. Classification tasks assign data to predefined categories based on patterns and similarities, but defining decision boundaries in complex, high-dimensional spaces can be challenging. The **K-Nearest Neighbors (KNN) algorithm** addresses this by classifying new data points based on their K closest neighbors, making it effective for non-linear and multi-class problems with minimal assumptions about data distribution.

The proposed KNN classifier accelerator leverages parallel processing and a custom hardware architecture to enhance the performance of the KNN algorithm. Our work involved software implementation, architectural and logic design, the verification and simulation of the system.

Our primary goal throughout the project was performance efficiency. We aimed to design a chip that operates with minimal power consumption, area, and hardware resources, while maximizing the classification speed of the algorithm.

Table of Contents

List of Tables

List of Figures

1. Introduction

The K-Nearest Neighbors (KNN) algorithm is a non-parametric, supervised algorithm used for classification and regression tasks.
The algorithm uses proximity in feature space to perform classification or prediction about the grouping of an individual data points.
The algorithm is one of the oldest machine learning algorithms methods. It was first introduced in the 1950's and a few years later in the 1960's was formalized for pattern recognition tasks. The main reason for the algorithm popularity is its simplicity and effectiveness in real-world applications such as image-recognition, medical diagnosis, anomaly detection and such on.

1.1. Key Terminologies in Machine Learning

1. **Non-Parametric Algorithm:** An algorithm that does not assume any predefined distribution for the obtained data, allowing it to adapt to the underlying structure of the dataset.
2. **Supervised Learning:** A machine learning approach in which a model is trained using labeled data, where each training instance is associated with a known output.
3. **Labeled Data:** A dataset in which each sample consists of input features along with the corresponding correct output label, used for training supervised learning models.
4. **Classification:** A type of supervised learning task aimed at predicting discrete labels or categories based on input data.
5. **Regression:** A supervised learning task focused on predicting continuous numerical values based on input features.
6. **Test Point:** A previously unseen data sample for which the model must determine the most relevant label or prediction.
7. **Training Dataset:** A collection of labeled data samples used to train a machine learning model.
8. **Training Point:** An individual data sample from the training dataset that the algorithm utilizes for learning and making predictions.

2. KNN Functionality

2.1. Data representation

In the KNN algorithm, the way data is represented is crucial for accurate predictions. Thus, because the algorithm is based entirely on the distance between different data points to make predictions.

Each data point is represented as a vector in an n-dimensional space, where each dimension corresponds to a property.

Property Data types:

Numerical Data	Categorical Data
The data is based on Numbers.	The data is based on characteristics
Works naturally in KNN.	Needs to be converted into numerical data.

Table 1: Numerical vs. Categorical Data

Since KNN is based on distances between data points, differences in scale between features (e.g., age ranges from 0 to 120, while price has no upper limit). Can significantly damage the reliability of the results. Hence, we would like to normalize the data, common normalization techniques are:
Min-Max: rescale features to [Min value, Max value].
Z-score: convert the data so that it will be distributed in a normal distribution.

2.2. Distance Metric

Determine proximity between 2 different data point in the feature space is made by calculating the distances between those 2 points.

There are a few methods to measure the distance, with the two most common methods are:

1. **Euclidean Distance:** $d(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$
2. **Manhattan Distance**: $d(A,B) = \sum_{i=1}^{n}|A_i - B_i|$

2.3. K selection:

The parameter K specifies the number of nearest neighbors which is needed to determine the group of the given data point.

There are a few common methods for choosing the ideal K value:

1. **Rule of Thumb:**
   Choose the k by the general equation: $K = \sqrt{N}$.
   Where N is the total number of data samples.
2. **Cross-Validation:**
   Divide the train set into two different sets, training set and validation set. Then calculate the performance of the forecaster for different values of k, and by those results choose the ideal K that will give us the best validation score.

Those methods, as said, will lead to choosing the ideal K for the algorithm. There are a few implications regarding the value of the K that was chosen:

- **Small K** – choosing small K can lead to overfitting, which means the prediction model will perform well on the training data but will predict badly on new, unseen data. Small K will lead to high variance and will make the prediction model more sensitive to noise.
- **Large K** – choosing large K can lead to underfitting, which means the prediction model is too simple, and perform badly on both training and new, unseen data. Large K will lead to high bias, and we can refer to training points that do not represent the classification well.

2.4. Overview of the KNN Algorithm Process

1. **Data Preparation:** Store all the labeled training data in the memory.
   The data is represented by two axis: X, Y
2. **Compute distance:** For each new test point, calculate the distance to all training points.
3. **Neighbor selection:** Identify the K closest training points by sorting the distances, based on the chosen distance metric.
4. **Predict output:**

Classification: assign the class based on majority.
Regression: assign the average value of the K-nearest neighbors.

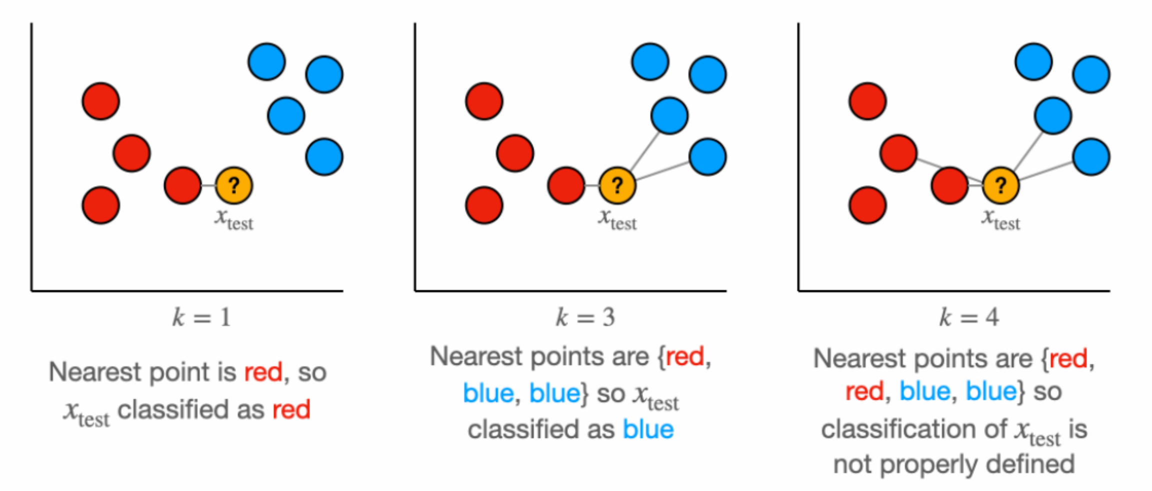
2.5. Advantages and Disadvantages of the KNN algorithm.

Advantages	Disadvantages
Simple and easy implementation	High memory requirements
Effective for small datasets	Sensitive to irrelevant or redundant features.
No assumptions about the data distribution	Performance depends on the choice of K and distance metric.
Can be used for both classification and regression	

Table 2:Advantges and Disadvantages of KNN Algorithm

2.6. Example

The figure below illustrates the classification process, where the red and blue circles represent the training points, and the yellow circle with a question mark indicates the test point. The classification process is demonstrated by the positioning of the colored circles in relation to the test point (X_test), with the chosen value of K influencing the classification based on the distances between the test point and the training points.


?
x_{test}
$k = 1$
Nearest point is red, so x_{test} classified as red
?
x_{test}
$k = 3$
Nearest points are {red, blue, blue} so x_{test} classified as blue
?
x_{test}
$k = 4$
Nearest points are {red, red, blue, blue} so classification of x_{test} is not properly defined

[1] Figure 1: Example of KNN Algorithm

3. Goals, Requirements and Specifications

The project has several requirements and specifications we defined at the beginning of the project:

- The algorithm operates with a K value of 5.
- There are two labeled groups: 0,1.
- The distance is calculated using the Manhattan distance method.
- The total number of training points is 128.
- The max number of Inputs/Outputs is 48.
- Each data point will have values on both X and Y axes, with each value ranging: [0,1023].
- All outputs of the chip must come directly from a flipflop/register.
- The accelerator will be fabricated in TSMC with a technology process of 65 nm.
- The chip uses a single clock. The target clock frequency is 100 MHz.
- The chip target area is 1.0x1.0 sq. mm.
- The Y dimension of the chip is 830 microns without bond pads.

One of the key parameters considered in chip design was performance efficiency. Our objective was to design a chip that operates with minimal power consumption and area, while maximizing speed. We aimed to achieve a reasonable trade-off between these parameters to optimize overall performance.

Frontend Design Section

4. Software Implementation

The initial stage of our project involved implementing the K-Nearest Neighbors (KNN) algorithm in Python. The primary objectives of the software implementation phase were as follows:

1. **To evaluate and determine the most suitable distance calculation method** for our application, by comparing:
   - Euclidean distance
   - Manhattan distance
2. **To establish a reliable reference** implementation that would serve as a benchmark for validating the results of the subsequent hardware implementation.

To determine the most appropriate distance metric, we conducted extensive testing using multiple datasets composed of a large number of points in the X-Y plane. We analyzed whether the choice of distance metric led to significant differences in classification results.

After selecting the method that best suited our needs, we used the chosen algorithm to classify a test set. This same test set was later applied to the hardware implementation, enabling a direct comparison between software and hardware results. This comparison ensured the correctness and accuracy of our hardware design.

5. Top Level Architecture

5.1. Top Level Design Interface

The KNN classifier accelerator receives 128 labeled training points, and one unlabeled test point as inputs. The accelerator stores the values of these points, calculates the distances between the test point and each training point, and sorts the distances to find the 5 smallest distances. From them , it will classify the group of the test point according to the most common label of those 5 smallest distances. This group will be the result of the KNN classifier accelerator and will act as output.

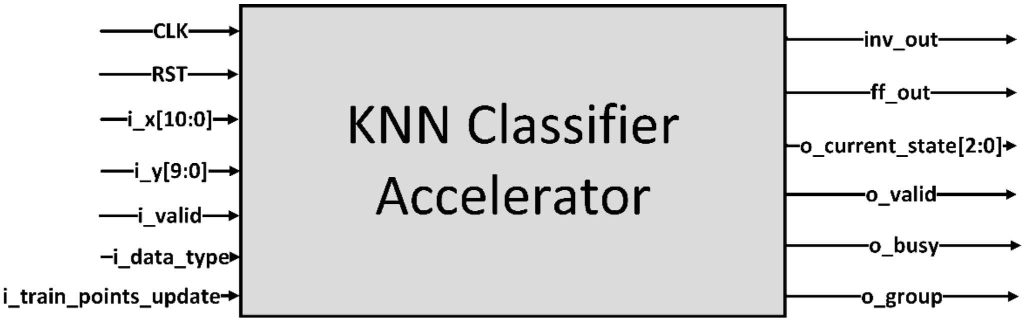


Figure 2:Top Level I/0 Diagram

Data packet-

The i_x Input contains 11 bits that are represented as follows:

1. **Group** – The most significant bit, representing the label of the current training point.
2. **Value bits**- the 10 least significant bits, represent the x axis value of the current training/test point.

Note: The i_y input contain only 10 bits.

The KNN algorithm operates by utilizing labeled training points to accurately classify a given test point. Since the purpose of the algorithm is to assign a label to the test point, having a predefined label for it is unnecessary. Consequently, the most significant bit (MSB) is relevant only for training points, whereas the 10 least significant bits (LSBs) are applicable to both training and test points.

Signal Name	Input / Output	Size (bits)	Description
Clk	Inputs	1	System clock.
Rst		1	Asynchronous reset signal, active on positive edge.
i_x		11	Msb – The label of the training point (irrelevant for the unlabeled test point). [9:0] bits – The X axis value of both training/test point.
i_y		10	The Y axis value of both training/test point.
i_valid		1	Indicate if the i_x, i_y, i_data_type inputs are valid and should be processed
i_data_type		1	Indicate if the current data is test point (1) or train point (0).
i_train_points_update		1	Indicate that the accelerator is prepared to process a new set of 128 train points.
inv_out	Outputs	1	Inverter output – sanity check
ff_out		1	Flip-flop output – sanity check
o_current_state		3	Current state of the state machine : Idle, Memory_write, transition_st, sort, knn_finished.
o_valid		1	Indicate that the group output is ready and relevant.
o_busy		1	Indicate that the accelerator will ignore any input except clk and rst.
o_group		1	The predicted label of the current test point.

Table 3: Top Level I/0 Signals

5.2. Top Level Block Diagram

The top-level architecture contains the following sub-modules:

1. **Memory** – Stores the 128 labeled training points. The Memory stores the X values and the Y values separately.
2. **Distance calculator** – responsible for calculating the Manhattan distance between the test point and the labeled training points. The top-level architecture contains 4 units of this sub-module.
3. **Bitonic sort** – responsible for collecting the distances and finds the 5 smallest distances by sorting them.
4. **Group decider** - responsible for classifying the group of the current test point.
5. **Controller** - FSM that provides control signals to all the other sub-modules.
6. **x/y test point reg** – Stores both X, Y values of the test point.
7. **FF** – Makes sure that all the outputs come directly from a flipflop/register. The top-level architecture contains 4 FF.
8. **Inverter** – Invert the i_data_type.

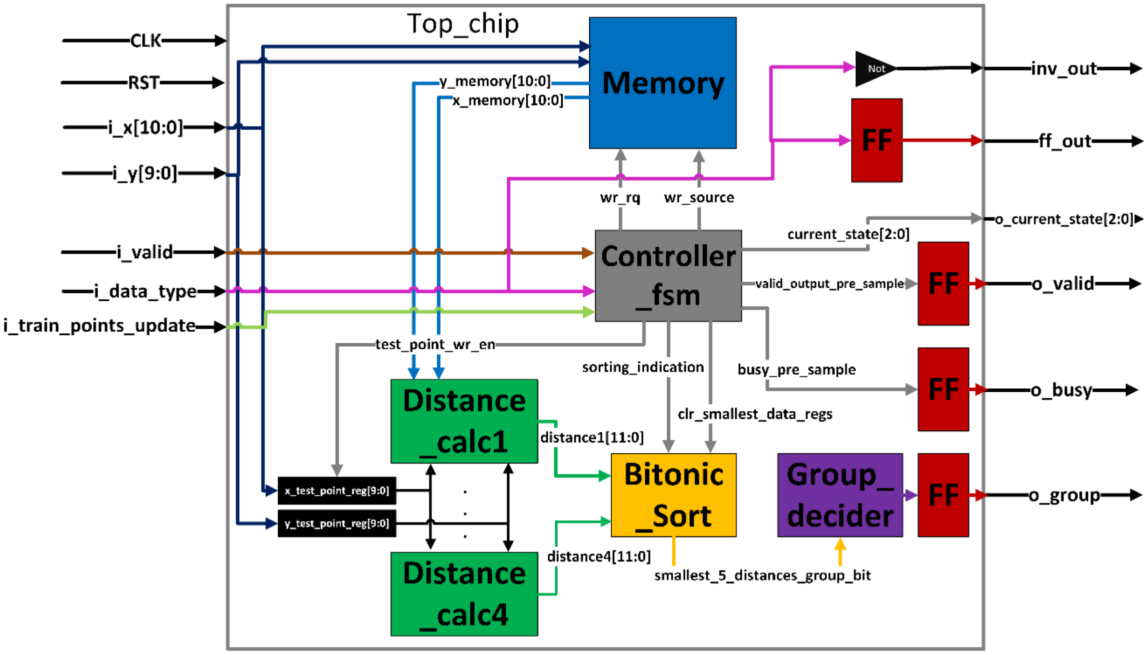


Figure 3: Top Level Architecture Scheme

6. Memory

6.1. Memory implementations

During the development of the project, we explored multiple approaches to implementing the memory module:

1. **Option 1 –** "Addresses" approach, this option allows access to each register both from the inputs and the outputs. It enables writing to and reading from each register using the "address" control signal.
2. **Option 2 –** "Shift Register" approach with a cyclic implementation, the input has access only to the first register, while the outputs can access the last four registers.

6.2. Advantages and Disadvantages

The table below summarizes the key characteristics of the two design options:

	Option 1	**Option 2**
Advantages	Consists of 8 multiplexers (MUX) with a 32-to-1 configuration, which is approximately equivalent to 248 MUX units of 2-to-1 (as each 32-to-1 MUX comprises 31 MUX units of 2-to-1).	Lower complexity in control signals management.
		The inputs are connected to a single register.
Disadvantages	High Complexity in control signals management.	Requires 256 mux units of 2-to-1.
	Each Input is connected to all registers.	

Table 4:Comparison of Implementation Options for a Memory Module

Considering the advantages and disadvantages of both options, and particularly the significant benefits of reduced complexity in control signal management, we chose the second option for implementing our memory module.

6.3. Functionality

In this section we will describe the implementation of the second option.

In the memory module we have 2 main phases of work:

1. **Write Phase –** In this phase, data is written into the registers. During each write cycle, a new labeled training point is stored in the top register, while the existing data shifts to the next register. This means that data in register N is transferred to register N+1, following a queue -like structure.
   The Y data is stored in a 10-bit register, while the X data, representing also the group of the point, is stored separately in an 11-bit register.
2. **Read Phase -** In this phase, data is read from the lowest four registers (both X and Y registers) and retrieved from memory.
   During each reading cycle, the data in the four lowest registers is updated by shifting the contents from register N to register N+4. This approach ensures that the data remains available for future use and is not lost.

6.4. Interface and Signals

Signal Name	Input / Output	Size (bits)	Description
clk	Inputs	1	System clock.
rst		1	Asynchronous reset signal, active on positive edge.
X		11	The written training point's X-axis value. **MSB**: Label of the training point. **[9:0] bits**: X-axis coordinate.
Y		10	The Y-axis coordinate of the written training point.
Wr_source		1	Control signal indicating the current memory phase: **0**: Write phase. **1**: Read phase.
Wr_rq		1	Control signal determining whether data should be written in the current clock cycle.
X_out	Outputs	4x11	The retrieved X-axis values of the four lowest stored training points.
Y_out		4x10	The retrieved Y-axis values of the four lowest stored training points.

Table 5: Memory I/0 Signals

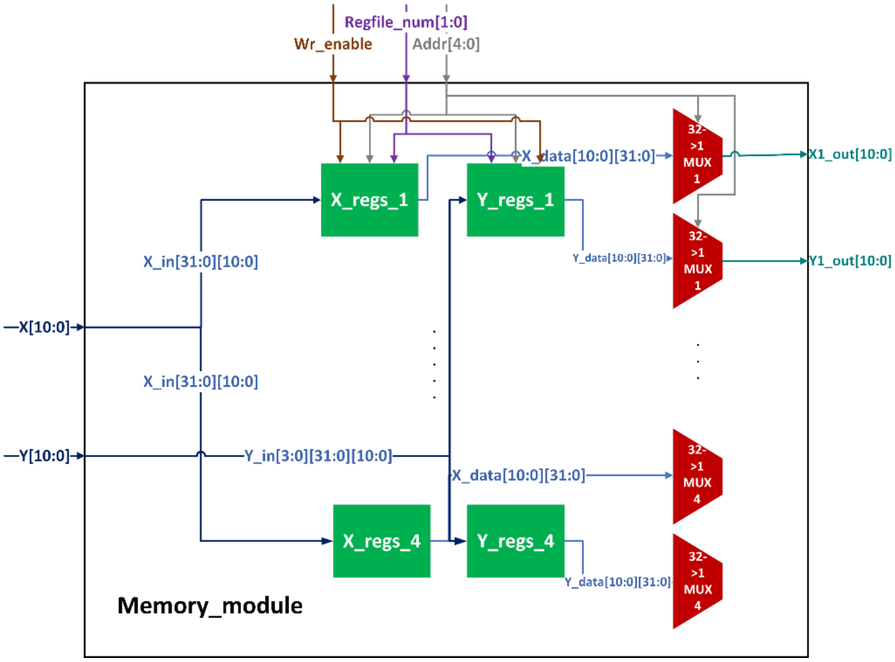


Figure 4: Memory Implementation Option 1

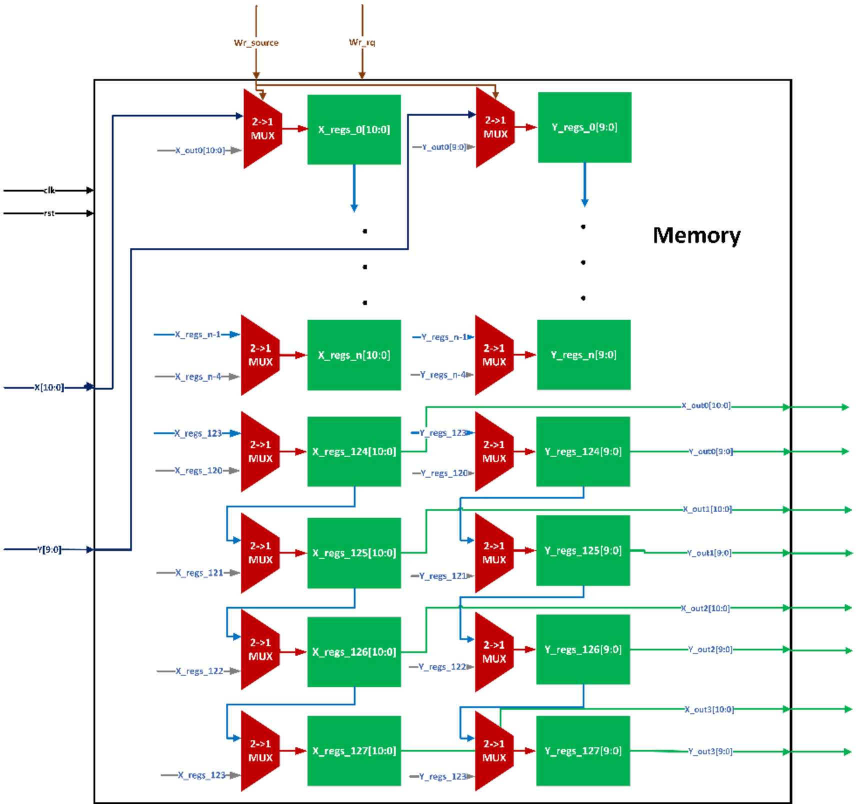


Figure 5:Memory Implementation Option 2

7. Distance calculator

This module executes the logical computation of the distance between the test point and each training point to implement the Manhattan distance calculation. To reduce the calculation latency from 128 clock cycles to 32 clock cycles, four instances of this module are implemented within the chip.

7.1. Functionality

The X and Y coordinate values of the training points are retrieved from memory, while the X and Y values of the test points are obtained from designated registers that store these values.

First, the module determines which X and Y values are greater to ensure only positive differences. It then separately subtracts the X value (excluding the most significant bit) and the Y value before summing the results.

The output of this addition represents the Manhattan distance. To this result, the most significant bit (bit 11) of the X value from memory is concatenated, serving as the labeled group identifier for the training point and acting as the MSB of the final output.

7.2. Interface and Signals

Signal Name	Input / Output	Size (bits)	Description
X_test_point_reg	Inputs	10	Test point X-axis value.
Y_test_point_reg		10	Test point Y-axis value.
X_mem		11	Training point's X-axis value. **MSB**: Label of the training point. **[9:0] bits**: X-axis coordinate.
Y_mem		10	Training point's Y-axis value.
Distance	Outputs	12	Manhattan distance. **MSB**: Label of the relevant training point. **[10:0] bits**: Manhattan distance.

Table 6:Distance Calculator I/O Signals

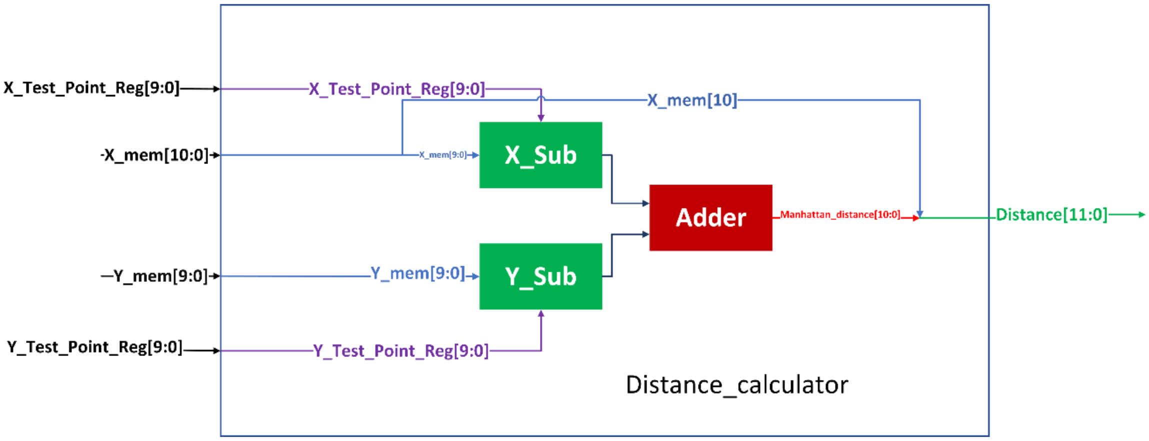


Figure 6: Distance Calculator Unit

8. Sort

8.1. Sorting implementations

During the development of the project, we explored multiple approaches to implementing the sort module:

1. **Option 1** – In the pipeline approach, the five smallest distances are stored in the upper registers. During each pipeline cycle, a new distance is inserted into the pipeline. The five distances stored in the "lower" registers are compared with the distances held in the five smallest distance registers. Each distance is compared with only one register that holds one of the five smallest distances. In this comparison approach, during each cycle, the five smallest distances are moved to the upper registers, while the remaining distances are retained in the lower registers.
2. **Option 2** – In the trivial solution, during each clock cycle, all the current 9 distances are compared with each other (the 5 smallest distances from the previous cycle and the 4 new distances). The 5 smallest distances are then stored as the new set of smallest distances.
3. **Option 3** – Bitonic sort algorithm, this approach modifies the established bitonic sort algorithm, which typically sorts 8 numbers, by adding an additional number. As a result, the sorting process handles a total of 9 distances.

8.2. Advantages and Disadvantages.

The table below summarizes the key characteristics of the three design options:

Criteria	Option 1	Option 2	Option 3
Sorting Methodology	Uses pipeline – boosts sorting throughput.	The trivial solution at the logical level.	Well known sort algorithm. Non-trivial solution.
Hardware Resource Usage	High usage of hardware: • 40 Registers • 20 comparators • 36 Multiplexers of 2-to-1.	Require the most hardware: • 9 Register • 36 comparators • 35 Multiplexers of 2-to-1 • 9 adders	Requires the least hardware: • 9 Registers • 22 Comparators • 40 Multiplexers of 2-to-1
Sorting Latency and Instantiation	57 clock cycles of sorting, with 4 Instances.	33 clock cycles, with 1 instance.	33 clock cycles, with 1 instance.
Control Logic Complexity	High complexity in control signals management.	Moderate complexity in control signals management.	Low complexity in control signals management.

Table 7:Comparison of Implementation Options for a Sort Module

Considering the advantages and disadvantages of each option, and particularly the significant benefits of reduced complexity in control signal management and low usage of hardware, we chose the third option for implementing our sort module.

8.3. Functionality

8.3.1. Bitonic sort

Bitonic Sort is a classic parallel sorting algorithm based on the idea of bitonic sequences. A bitonic sequence is a sequence of numbers that first increases and then decreases or can be made to increase and decrease with a cyclic shift. The Bitonic Sort algorithm works by recursively dividing the sequence into smaller bitonic sequences and sorting them in parallel. It is especially efficient for parallel processing environments.

In our implementation, we use a bitonic sorting network designed to sort a sequence of 8 numbers. The sorting process consists of several structured stages, as outlined below:

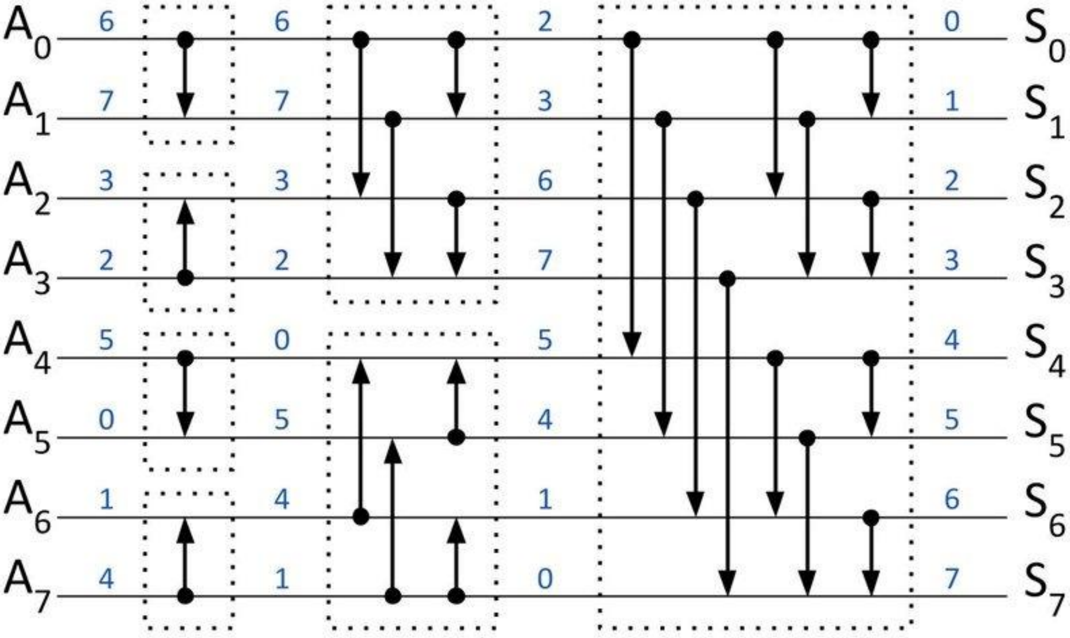
1. **Initial Sequence:**
   We begin with an unsorted sequence of 8 input elements.

2. **Step 1 – Pairwise Comparison:**
   In the first stage, the 8 elements are divided into adjacent pairs. Each pair undergoes a **compare-and-swap** operation, resulting in 4 **sorted sub-sequences** of 2 elements each. These sub-sequences are sorted in ascending or descending order, depending on their position in the network.

3. **Step 2 – 4-Element Comparison:**
   The sorted pairs are grouped into two sets of 4 elements. Each group contains two previously sorted pairs. These 4-element groups then undergo a **4-element comparison stage**, which merges the two pairs into a larger sorted sub-sequence.

   - One group is sorted in **ascending order**.
   - The other group is sorted in **descending order**.
     This results in a **bitonic sequence**, where the first half increases and the second half decreases.

4. **Step 3 – Bitonic Merge:**
   Finally, the two 4-element sequences are merged into a single group of 8 elements. This full sequence is now a **bitonic sequence**, which is sorted using a final series of comparisons that convert it into a fully **sorted sequence in ascending order**.



[2] Figure 7: Example of Bitonic Sort with 8 Numbers

8.3.2. Our implementation – 9 Numbers bitonic sort

Our main insight is that it is unnecessary to sort all 128 distances or determine their complete order. Instead, we only need to identify the five smallest distances.

In this implementation of a 9-number bitonic sort, we built upon the 8-number bitonic sort presented in Section 7.3.1, making several modifications that enabled us to extend the sorting capability to 9 numbers.

In each cycle, we receive four new distances from the distance_calculator module and merge them with the five smallest distances identified in the previous cycle. We then sort these nine values to update the set of the five smallest distances.

We can "split" the implementation into 3 phases:

The implementation can be divided into three phases:

1. **Phase 1** – In this phase, we fully implemented the first step of the 8-number bitonic sort. We began by dividing the eight distances into four adjacent pairs. Then, we applied the compare-and-swap operation, resulting in four

sorted subsequences, each containing two elements. Throughout the module, the compare-and-swap operation was implemented using a single comparator and two multiplexers for each operation. The comparator compared the two distances, and its output (the comparison result) served as the selector for the multiplexers. In one multiplexer, the raw signal was used directly, while in the other, the signal was inverted before being applied as the selector.
At this stage, these sub-sequences are sorted in ascending or descending order.

2. **Phase 2** –In this phase, we fully implemented the second step of the 8-number bitonic sort. We took the four previously sorted pairs and merged them into two new sets, each containing four elements. These sets were then sorted internally, ensuring that each was arranged in either ascending or descending order.
   To achieve this, we used the same compare-and-swap technique as in phase 1, employing a comparator and multiplexers to determine and enforce the correct order.

3. **Phase 3** – As previously mentioned, our goal is not to sort all 128 distances, nor do we need to fully sort all 9 numbers. We only need to identify the 5 smallest distances. In the standard implementation of the 8-number bitonic sort, the last four compare-and-swap operations serve different purposes:
   - Two of these operations sort the four smallest distances. However, since all four of these values are guaranteed to be among the five smallest distances we seek, we can omit these operations.
   - One operation sorts the two largest distances (the 7th and 8th largest values). Since these numbers are not relevant to our goal, we can also skip this step.
   - The final operation sorts the 5th and 6th smallest distances. This step is necessary, as we are specifically interested in determining the five smallest distances.

   To extend the 8-number bitonic sort to accommodate 9 numbers, we perform one additional comparison: we compare the 5th smallest distance found in the previous step with the 9th number. This ensures that the final set contains the five smallest distances.

Once the five smallest distances are identified, they are stored in a designated register, which will be used in the next cycle to continue the sorting process efficiently. After 32 cycles, the process will yield the absolute five smallest distances.

8.3.3. Interface and Signals

Signal Name	Input / Output	Size (bits)	Description
clk	Inputs	1	System clock.
rst		1	Asynchronous reset signal, active on positive edge.
Clr_smallest_data_regs		1	Control signal indicating a reset of the data_registers to the highest possible value (12'hfff).
Sorting_indication		1	Control signal serving as a write request to the register.
Distance_i		4x12	Represents the distances computed by each of the four distance_calculator modules
5_smallest_distances_group_bit	Outputs	5	The most significant bit (MSB) of each register within smallest_data_registers, which stores the five smallest distances. The MSB indicates the labeled group of each distance.

Table 8: Bitonic Sort I/O Signals

Sorting Indication – This signal functions as a write request to the register. In the first cycle, the intention is to write exclusively to the data_register, which stores the distances received from the distance_calculator modules. Consequently, this signal is directly connected to these registers. In the final cycle, the intention is to write solely to the smallest_data_registers, which store the smallest distances. To achieve this, the signal is delayed by one clock cycle using a flip-flop (FF) before being connected to the registers.

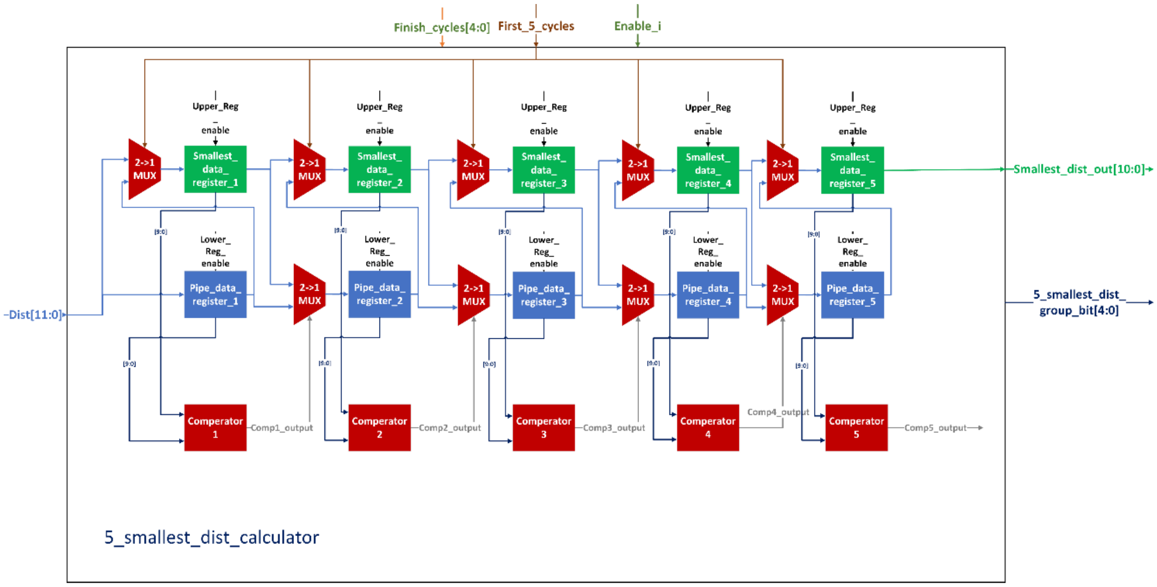


Figure 8:Sort Module Implementation Option 1

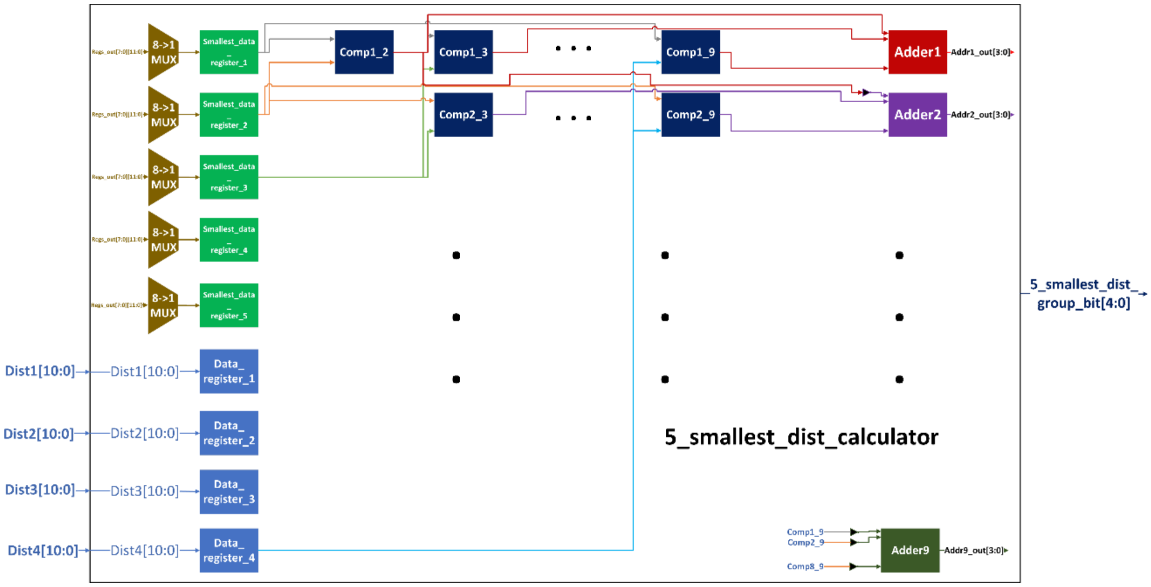


Figure 9:Sort Module Implementation Option 2

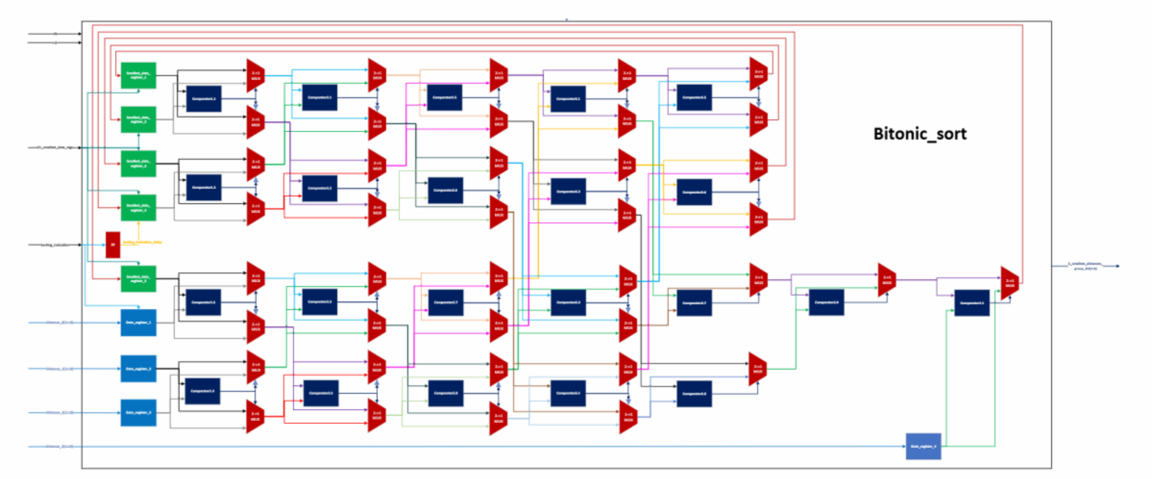


Figure 10:Bitonic Sort Unit (Option 3)

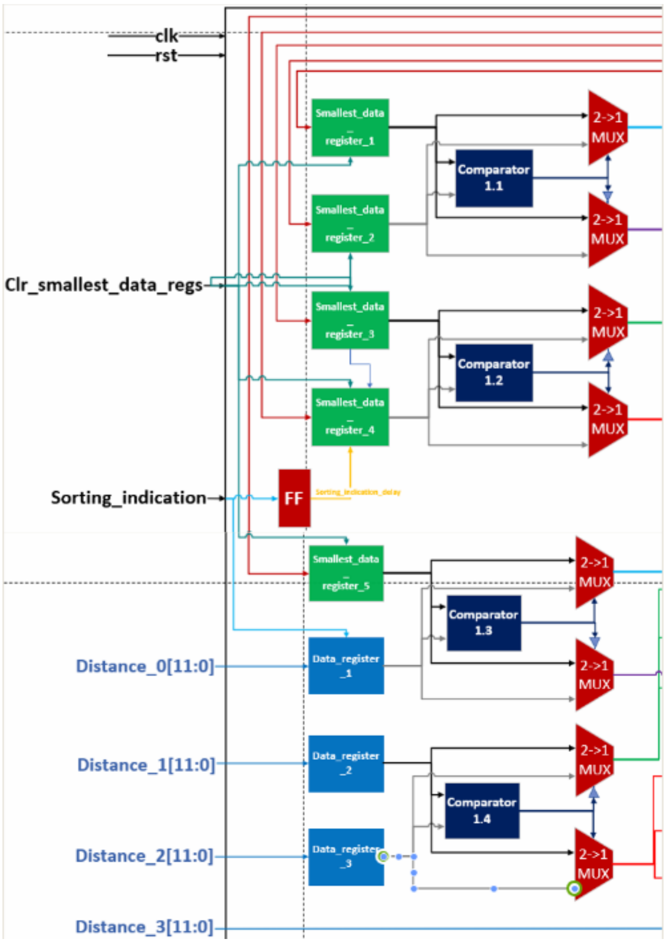


Figure 11: Bitonic Sort Unit - Phase 1

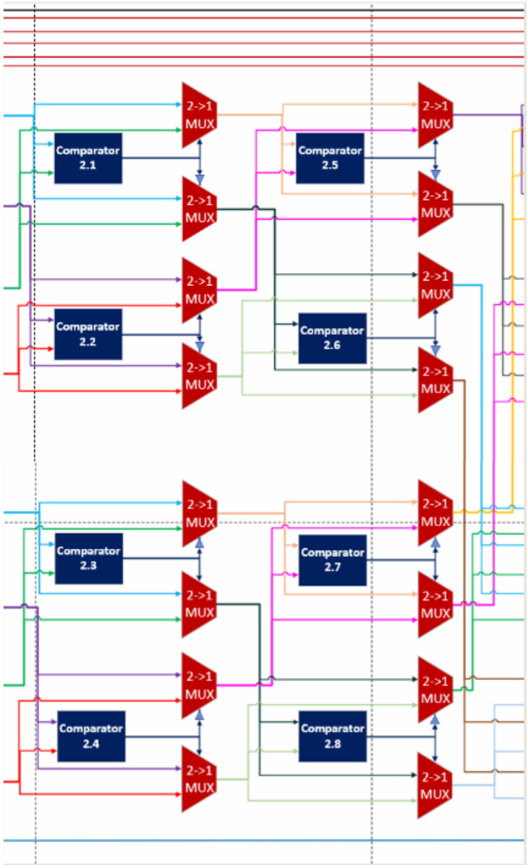


Figure 12: Bitonic Sort Unit - Phase 2

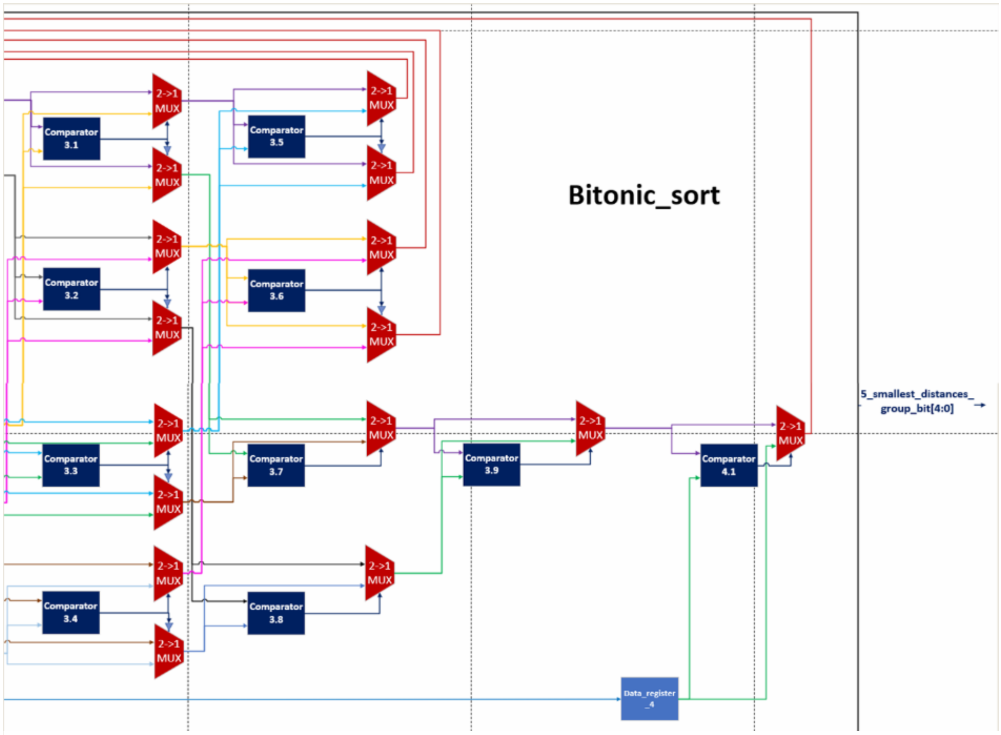


Figure 13: Bitonic Sort Unit - Phase 3

9. Group_decider

This module computes the label that classifies the test point based on the K-Nearest Neighbors (KNN) algorithm, using the labeled group corresponding to the five closest training points.

9.1. Functionality

This module receives, as a vector, the labeled group corresponding to each of the closest training points. It then sums these five bits to obtain a total count and uses a comparator to determine whether the sum exceeds 2.

- If the sum is greater than 2, the most common label among the five closest training points is 1, indicating that the test point should be classified as 1.
- If the sum is less than or equal to 2, the most common label is 0, meaning the test point should be classified as 0.

9.2. Interface and Signals

Signal Name	Input / Output	Size (bits)	Description
5_smallest_dist_group_bit	Input	5	A vector representing the labeled group of each of the five closest training points.
Group	Output	1	The classified group of the test point based on the K-Nearest Neighbors (KNN) algorithm. For K = 5.

Table 9: Group Decider I/O Signals

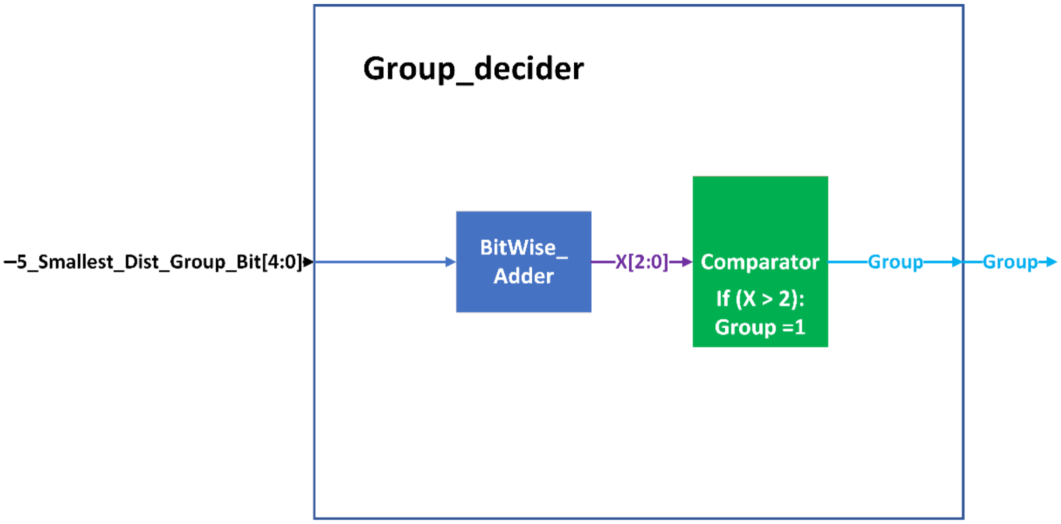


Figure 14: Group Decider Unit

10. Controller

The controller functions as a finite state machine and interfaces with all system units except the Distance_calculator modules. It interacts with these units through command signals, which are transmitted from the controller to the respective modules, and status signals, which originate from external sources beyond the chip. The primary role of the controller is to coordinate communication between the modules and oversee data flow across the entire system.

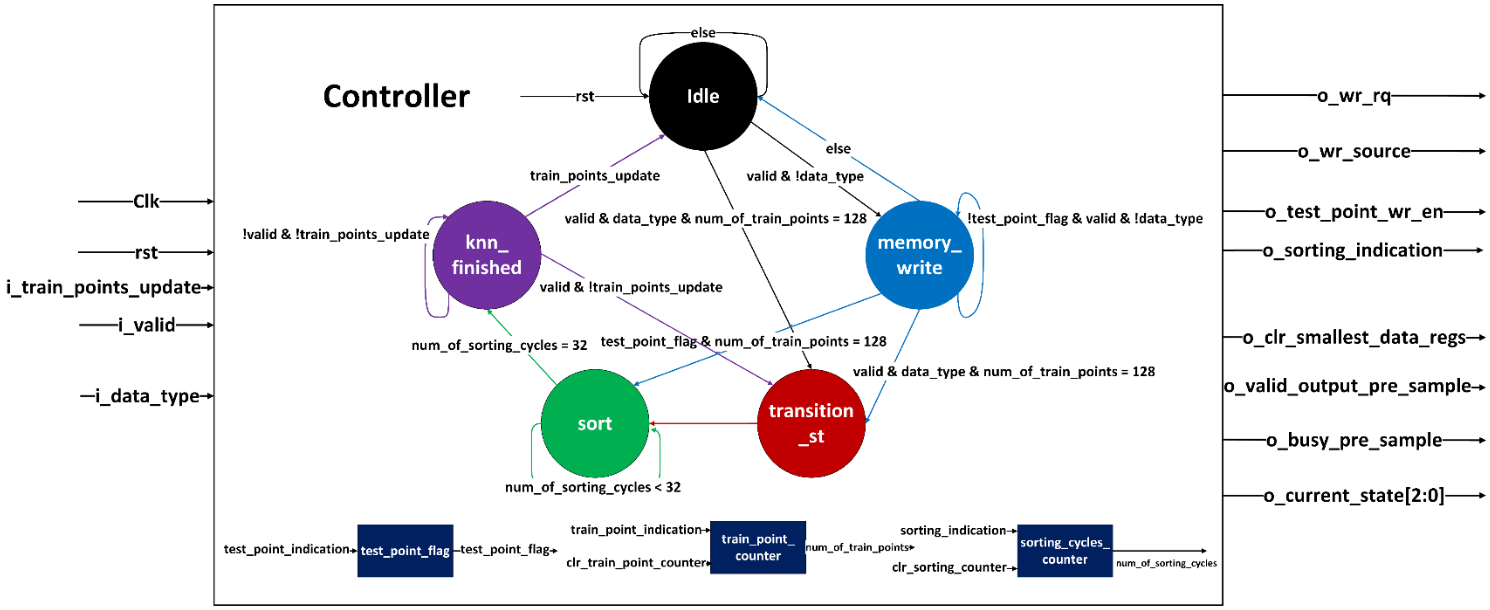


Figure 15: Controller Unit

10.1. Interface and Signals

Signal Name	Input / Output	Size (bits)	Description
clk	Inputs	1	System clock.
rst		1	Asynchronous reset signal, active on positive edge.
i_valid		1	Indicates if the i_x, i_y, i_data_type inputs are valid and should be processed
i_data_type		1	Indicates if the current data is test point (1) or train point (0).

i_train_points_update		1	Indicates that the accelerator is ready to process a new set of 128 train points.
o_wr_source	Outputs	1	Control signal indicating the current memory phase: **0**: Write phase. **1**: Read phase.
o_wr_rq		1	Control signal determining whether data should be written to the memory in the current clock cycle.
o_test_point_wr_en		1	Control signal determining whether data should be written to the test point register.
o_sorting_indication		1	Control signal serving as a write request to the registers inside the Bitonic_sort module.
o_clr_smallest_data_regs		1	Control signal for the data_register in the Bitonic_sort module to its maximum possible value (12’hfff).
o_valid_output_pre_sample		1	Indicates that the group output is ready and relevant. Pre-sample stage.
o_busy_pre_sample		1	Indicates that the accelerator will ignore any input except clk and rst. Pre-sample stage.
o_current_state		3	Current state of the FSM.

Table 10: Controller Module I/O Signals

10.2. Finite State Machine

The core component within the controller is a finite state machine (FSM), implemented as a Mealy machine.

The Default values of the FSM are:

Signal	Default value
Next_state	Current_state
O_wr_rq	1’b0
O_wr_source	1’b0
O_valid	1’b0
O_wr_test_point_en	1’b0
O_busy	1’b0
O_clr_smallest_data_regs	1’b0
Test_point_indication	1’b0
Train_point_indication	1’b0
Sorting_indication	1’b0
Clr_sorting_counter	1’b0

Table 11: Controller FSM Default Values

The controller is assisted by two counters and a 1-bit register to manage state transitions:

1. The first counter, train_point_counter, tracks the number of training points received from the user. It is an 8-bit counter that increments from 0 to 128.
   The counter has two input signals:
   - **train_point_indication:** This signal is asserted high during each clock cycle in which a training point is received. When high, the counter increments by 1.
   - **clr_train_point_counter:** This signal is asserted high when a new set of 128 training points is to be received. When high, the counter resets to its initial value of 0.

   This counter enables the system to receive training points and test points in any possible order.
2. The second counter, sorting_cycle_counter, tracks the number of clock cycles spent in the *Sort* state. It is a 7-bit counter that increases from 0 to 32.
   The counter has two input signals:
   - **sorting_indication**: This signal is asserted high during each clock cycle while the system remains in the *Sort* state. When high, the counter increments by 1.

- **clr_sorting_counter**: This signal is asserted high upon completion of the *Sort* state. When high, the counter resets to its initial value of 0.

  This counter determines when to transition from the Sort state to the knn_finished state.

3. The 1-bit register, test_point_flag, is used to indicate whether the test point has already been received. This register serves as a status signal that assists the controller in determining the appropriate state transition.

The controller state machine has five states:

- **Idle –** This is the initial state of the chip, meaning that when the reset signal is asserted, the chip transitions to the Idle state. This state indicates that the chip is not actively processing any tasks. However, if a test point is received before all 128 training points have been stored in the memory, the chip will remain in the Idle state.

Next State	Inputs & Internal Signals	Outputs & Internal Signals	Description

Memory_write	i_valid = 1’b1
	i_data_type = 1’b0
	num_of_train_points = 128
		o_wr_rq = 1’b1	A training point has been received. Although 128 training points have already been stored, the test point has not yet been received.

Memory_write	i_valid = 1’b1
	i_data_type = 1’b0
	num_of_train_points < 128
		o_wr_rq = 1’b1
		train_point_indication = 1'b1
		test_point_indication = 1'b1
			A training point has been received, but fewer than 128 training points have been stored. The system continues storing training points while ignoring whether a test point has been received.

Transition_st	i_valid = 1’b1
	i_data_type = 1’b1
	num_of_train_points = 128
		o_busy = 1’b1
		o_wr_test_point_en = 1'b1
			The chip has received all 128 training points, and a test point is currently being received.

Idle	i_valid = 1’b1
	i_data_type = 1’b1
	num_of_train_points < 128
		o_wr_test_point_en = 1’b1
		test_point_indication = 1’b1
			A test point has been received before all 128 training points have been stored in memory.

Table 12: Possible State Transitions from Idle State

- **Memory write –** This state indicates that we are currently writing a training point data into the Memory module.

Next State	Inputs & Internal Signals	Outputs & Internal Signals	Description

Memory_write	i_valid = 1’b1
	i_data_type = 1’b0
	test_point_flag = 1’b0
	num_of_train_points < 128
		o_wr_rq = 1'b1
		train_point_indication = 1'b1
			A training point has been received. The test point has not yet been received, and fewer than 128 training points have been stored.

Memory_write	i_valid = 1’b1
	i_data_type = 1’b0
	test_point_flag = 1’b0
	num_of_train_points = 128
		o_wr_rq = 1'b1	A training point has been received. While the test point has not yet been received, all 128 training points have already been stored. The newly received training point replaces the oldest stored training point in memory.

Sort	test_point_flag = 1’b1
	num_of_train_points = 8'd128
		o_wr_source = 1'b1
		sorting_indication = 1'b1
		o_wr_rq = 1'b1
		o_busy = 1'b1
			The last training point (training point number 128) has been received, and the test point was already received earlier.

Transition_st	i_valid = 1’b1
	i_data_type = 1’b1
	num_of_train_points = 8'd128
		o_wr_test_point_en = 1'b1
		test_point_indication = 1'b1
		o_busy = 1'b1;
			The test point is received in the clock cycle immediately following the reception of the last training point (i.e.,

			training point number 128).

Idle	i_valid = 1’b1
	i_data_type = 1’b1
	num_of_train_points < 8'd128
		o_wr_test_point_en = 1'b1
		test_point_indication = 1'b1
			The test point has been received, but not all 128 training points have been collected.

Idle	i_valid = 1’b0	-	No new data has been received.

Table 13:Possible State Transitions from Memory Write State

- **Transition_st –** The purpose of this state is to introduce a one-clock-cycle delay when the last received data from the user is the test point.
  As part of the chip's operation, the system calculates the distances between the stored training points and the test point, with the resulting values stored in dedicated registers allocated for distance storage, as described in Section 4.2 (Top-Level Block Diagram). When the last received data is the test point, it must first be safely stored. Only in the following clock cycle can distance calculations be performed correctly. Without this delay, the first four distance calculations (managed by the four distance_calculator modules) would be based on incorrect data stored in the test point registers, leading to erroneous distance computations.

The clock wave diagram in figure 13 illustrates the necessity of the Transition state. If this state were removed, distance calculations would be performed using incorrect data stored in the x_test_point_reg and y_test_point_reg registers. Specifically, instead of using the correct value (200), the registers would still contain 0, leading to incorrect distance calculations.

Distance Calculation Formula:
Distance=| x_test_point_reg – x_reg_i |+| y_test_point_reg – y_reg_i |
Correct Distance Calculation:
Distance=∣200−50∣+∣200−50∣=300
Incorrect Distance Calculation (if Transition_st is removed):
Distance=∣0−50∣+∣0−50∣=100

As shown in the waveform, during the Memory state, the x_test_point_reg and y_test_point_reg registers still hold a value of 0. The Transition state ensures that the correct test point values (200) are stored before distance calculations occur in the Sort state. Without this delay, the first few calculations would be based on invalid test point data, leading to inaccurate results.

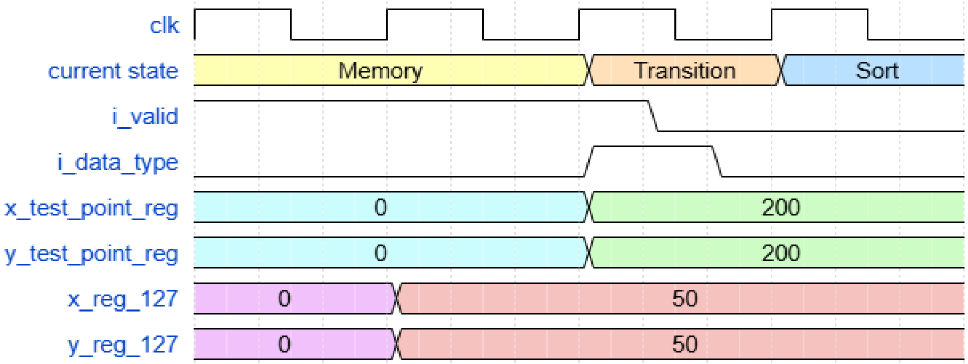


Figure 16:Illustration of the Necessity of the Transition State.

Next State	Inputs & Internal Signals	Outputs & Internal Signals	Description

Sort	-	o_busy=1'b1
		o_wr_source = 1'b1
		o_wr_rq = 1'b1
		sorting_indication = 1'b1
			-

Table 14:Possible State Transitions from Transition State

- **Sort –** This state represents the execution of the 32 clock cycles required for the sorting operation as part of the chip's functionality.
  During this state, the system also performs the distance calculations between the test point and the stored training points. These computations occur in parallel with the sorting process.
  The sorting process is managed by the Bitonic_sort module. During this state, all external inputs to the chip are ignored, ensuring that the sorting operation proceeds without interference.

Next State	Inputs & Internal Signals	Outputs & Internal Signals	Description

Sort	num_of_sorting_cycles < 32	o_wr_rq=1'b1
		o_wr_source = 1'b1
		sorting_indication = 1'b1
		o_busy = 1'b1
			The chip executes the sorting operation.

Knn_finished	num_of_sorting_cycles = 32	clr_sorting_counter = 1'd1
		o_busy = 1'b1
			The sorting operation has been successfully completed.

Table 15:Possible State Transitions from Sort State

- **Knn_finished –** This state indicates that the chip has completed the KNN algorithm's classification process. At this stage, the classified group of the test point will be available at the chip's output.

Next State	Inputs & Internal Signals	Outputs & Internal Signals	Description

Knn_finished	i_valid = 1’b0
	train_points_update = 1’b0
		o_valid = 1’b1	The chip has successfully completed the classification process, and no new data has been received.

transition_st	i_valid = 1’b1
	i_data_type = 1’b0
	train_point_update = 1’b0
		o_valid = 1’b1
		o_wr_rq = 1’b1
		o_wr_source = 1’b1
		o_clr_smallest_data_regs =1’b1
			The chip has successfully completed the classification process, and a new training point has been received.

transition_st	i_valid = 1’b1
	i_data_type = 1’b1
	train_point_update = 1’b0
		o_valid = 1’b1
		o_wr_test_point_en = 1’b1
		test_point_indication = 1’b1
		o_clr_smallest_data_regs = 1’b1
			The chip has successfully completed the classification process, and a new test point has been received.

Idle	train_point_update = 1’b1	o_valid = 1’b1
		o_clr_smallest_data_regs=1’b1
		clr_train_point_counter = 1’b1
			The chip has successfully completed the classification process. In this state, the memory can be

			initialized, and 128 new training points are received. Test point has not been received in the current cycle.

Idle	train_point_update = 1’b1
	i_valid = 1’b1
	i_data_type = 1’b1
		o_valid = 1’b1
		o_clr_smallest_data_regs=1’b1
		clr_train_point_counter = 1’b1
		o_wr_test_point_en = 1’b1
		test_point_indication = 1’b1
			The chip has successfully completed the classification process. In this state, the memory can be initialized, and 128 new training points are received.
			Test point has been received in the current cycle.
			Therefore, we will write this new test point into the designated register.

Table 16:Possible State Transitions from Knn_Finished State

10.3. Possible State Transitions in the Finite State Machine (FSM)

The following figure illustrates two possible state transitions within the Finite State Machine (FSM) among many potential transitions.

The figure presents two possible state transition scenarios within the FSM:

1. **First Scenario:** The first received data is a test point. This is followed by 128 training points, processed sequentially over 128 clock cycles. Afterward, the system transitions to the Sort state, which takes 32 clock cycles, before finally reaching the knn_ finished state.

2. **Second Scenario:** The first received data consists of 128 training points, processed over 128 clock cycles. Subsequently, a test point arrives, causing the state machine to enter the transition_st state, which requires one clock cycle for processing. Similar to the first scenario, the system then transitions to the Sort state (taking 32 clock cycles) before reaching the knn_finished state.

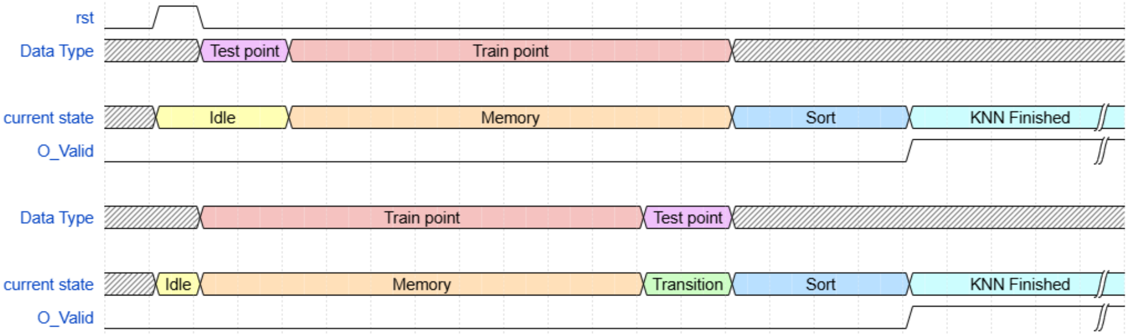


Figure 17: Example of Two Distinct Flows within the State Machine.

11. Timing Analysis

One of the project requirements was that the chip operate with a single clock at a target frequency of up to 100 MHz. This requirement necessitated ensuring that the longest delay path within our architecture does not exceed 10 ns.

To determine the longest path in the chip, we identified that it occurs within the Bitonic_sort module. This critical path passes through seven comparators and seven multiplexers. We evaluated the delay of this path and ensured that it remains within the 10 ns constraint.

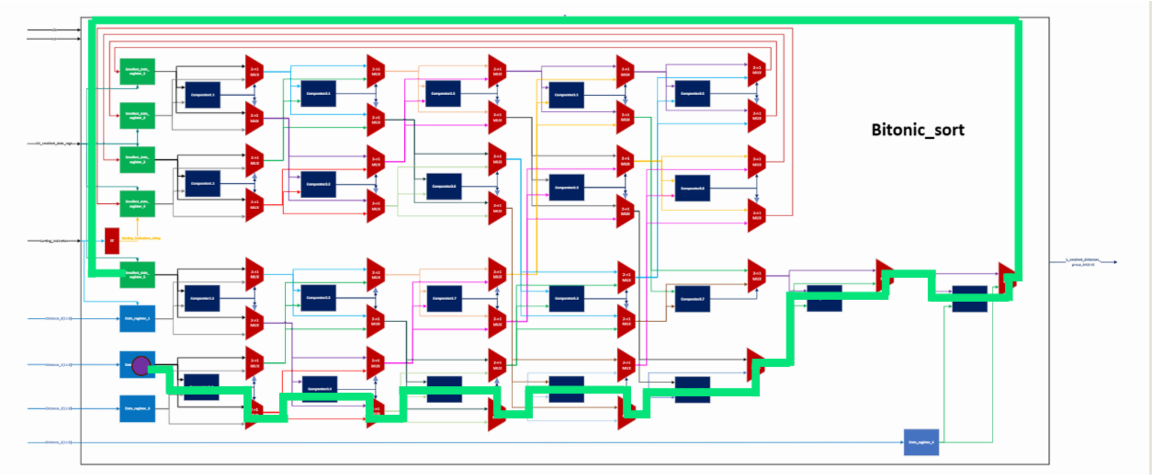


Figure 18: Illustration of the Critical Path

12. Simulations and Verification

Verification is a crucial stage in the front-end design process, ensuring that the chip functions as intended and identifying potential defects in the design or implementation.

After implementing each sub-module of the chip, we verified its logical behavior through logic simulation, ensuring that the actual behavior aligned with the expected functionality.

This section presents a subset of the simulations performed. The simulations were conducted using the VSC debug and verification management platform from Synopsys.

12.1. Memory

In the following simulation, we examined the write phase of the memory to ensure that 128 training points could be written sequentially into the memory over 128 clock cycles.

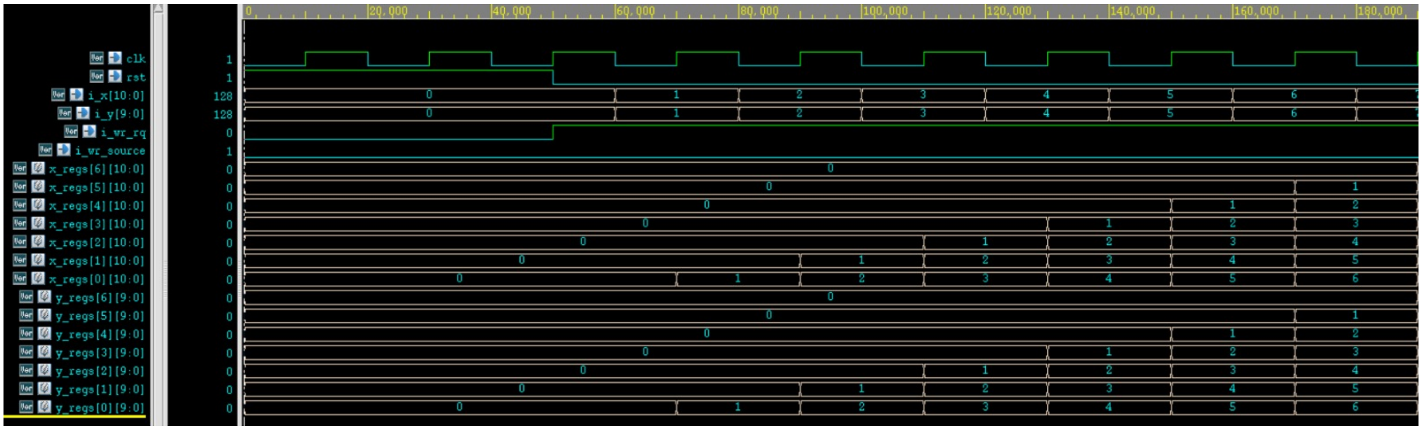
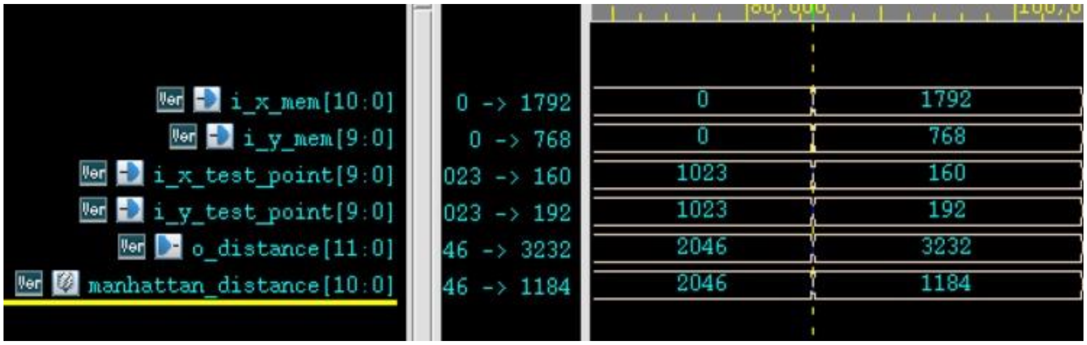


Figure 19:Memory- Write Phase Simulation

In the provided simulation, it is evident that during each clock cycle, the values of i_x and i_y increment by 1 (ranging from 0 to 6, with the remaining 122 values progressing further in the simulation). The memory is in the write phase, as indicated by i_wr_source == 1’b0. Additionally, in each clock cycle, the input data is written into the registers due to i_wr_rq == 1’b1, and the data shifts by one register per cycle, as discussed in Section 5.3 and can be seen in X_regs, Y_regs.

12.2. Distance Calculator

In the following simulation, we verified that the module correctly performs addition and subtraction for both positive and negative numbers.



```
        i_x_mem[10:0]    0 -> 1792     0       1792
         i_y_mem[9:0]    0 -> 768      0       768
  i_x_test_point[9:0]  023 -> 160     1023     160
  i_y_test_point[9:0]  023 -> 192     1023     192
     o_distance[11:0]   46 -> 3232    2046     3232
manhattan_distance[10:0]  46 -> 1184  2046     1184
```

Figure 20: Distance Calculator Simulation

In the first clock cycle, the values i_x_mem and i_y_mem are both set to 0. We then subtract i_x_test_point = 1023 and i_y_test_point = 1023 from these values, resulting in negative numbers. The absolute values of these differences are summed up to compute the Manhattan distance, which in this case is 2046.

Since the labeled group in this case is 0, the Manhattan distance is equal to o_distance, where o_distance represents the Manhattan distance concatenated with the most significant bit (MSB) of the i_x_mem indicating the labeled group.

In the second clock cycle, i_x_mem is initially 1792. However, since the MSB represents the labeled group of the training point (which is 1), it is removed from the distance calculation, resulting in i_x_mem = 768 and i_y_mem = 768. Next, we subtract i_x_test_point = 160 and i_y_test_point = 192 from these values, obtaining positive differences. The sum of these differences yields a Manhattan distance of 1184.

In this case, the labeled group is 1, meaning the Manhattan distance is not equal to o_distance. Instead, o_distance is expected to be 3232.

12.3. Bitonic Sort

In the following simulation, we initialized the smallest data registers to their maximum possible values. Then, over three cycles, we wrote four random numbers to the data registers per cycle.
At the end of the process, we verified that the five smallest numbers were correctly stored in the smallest _data_registers.

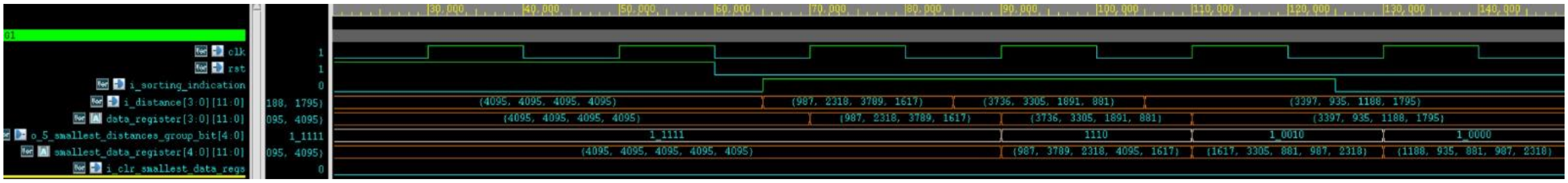


Figure 21: Bitonic Sort Simulation

Initially, the smallest data registers are set to their highest possible value, 4095.

- In the first clock cycle, the values 987, 2318, 3789, 1617 are written. When excluding the most significant bit (MSB), which represents the labeled group, these values correspond to 987, 270, 1741, 1617.
- In the second clock cycle, the values 3736, 3305, 1891, 881 are written. Without the MSB, these values correspond to 1688, 1257, 1891, 881.
- In the third clock cycle, the values 3397, 935, 1188, 1795 are written. When the MSB is removed, the resulting values are 1349, 935, 1188, 1795.

After three cycles, the smallest data registers contain the values 1188, 935, 881, 987, 2318. Without the MSB, these values are 1188, 935, 881, 987, 270.
The value of o_5_smallest_distances_group_bit is 10000.

12.4. Group Decider

In the following simulation, we verified that for each possible 5-bit input value (ranging from 0 to 31), the module correctly determines the group classification.

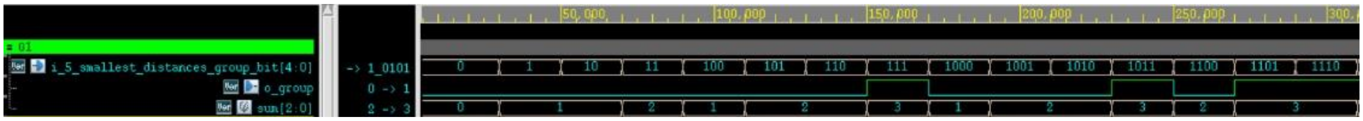


Figure 22: Group Decider Simulation

In the provided simulation, we observe that i_5_smallest_distances_group_bit transitions from 0 to 14, with the remaining 17 values continuing to progress throughout the simulation.

The output O_group is determined based on the sum of the input bits:

- If the sum of the bits is less than or equal to 2, O_group is set to 0.
- If the sum of the bits is greater than 2, O_group is set to 1.

12.5. Top Chip

In the following simulation, we verified that all sub-modules functioned correctly both individually and as part of the complete system.

As part of the top-chip simulation, we also tested the Controller sub-module, ensuring that the finite state machine (FSM) operated as expected. Additionally, we confirmed that each state correctly managed the corresponding sub-modules intended to function in that specific state.

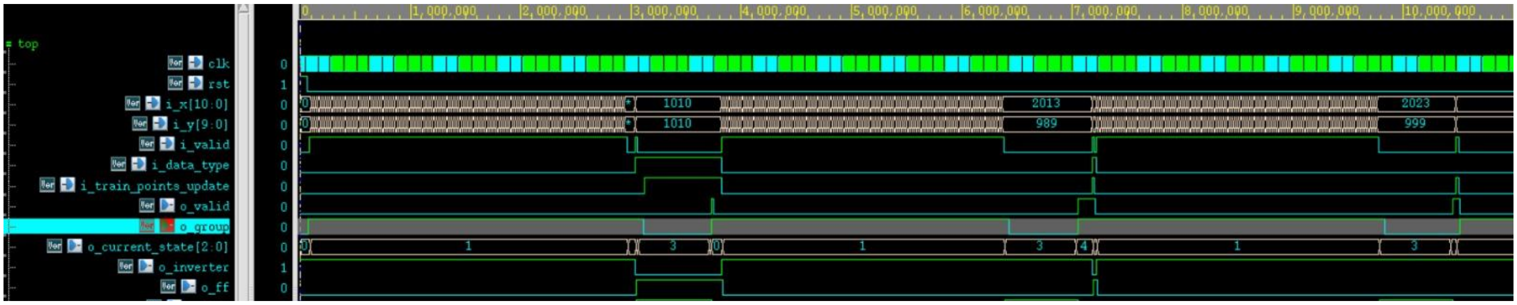


Figure 23: Top Chip Simulation

The simulation demonstrates three full cycle operations of the KNN Classifier Accelerator. In each cycle, new values for the test point and training points are provided. At the end of each operation, the test point is classified based on the labeled training points.

- In the first cycle, the test point values are (0,0), and it is classified as 1 (o_valid = 1, o_group = 1).
- In the second cycle, the test point values are (1010,1010), and it is classified as 1 (o_valid = 1, o_group = 1).
- In the third cycle, the test point values are (800,800), and it is classified as 0 (o_valid = 1, o_group = 0).

Additionally, the simulation shows that o_inverter is the inverted value of i_data_type, while o_ff represents a one-cycle delayed version of the same signal.

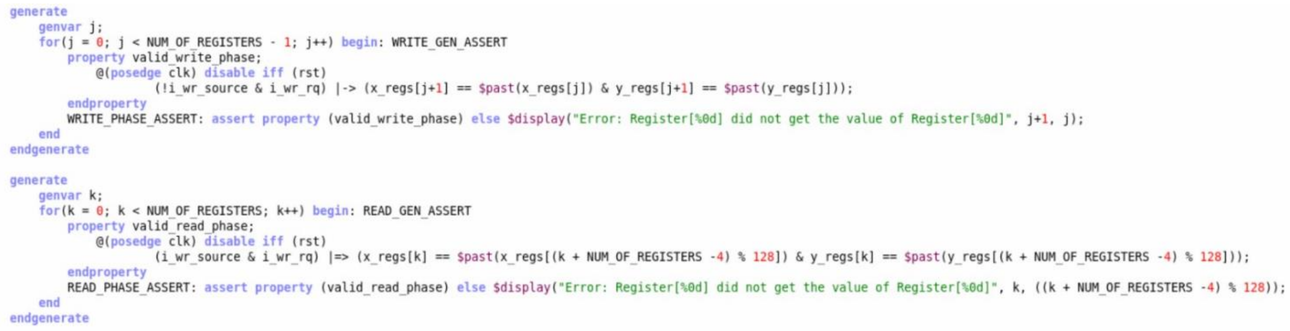
Furthermore, as long as o_current_state is in the transition or sort states (which have values 2 and 3, respectively), o_busy remains 1.

12.6. Assertions

Assertions are a crucial part of the logic design process, enabling verification of code correctness by automatically detecting violations of expected behavior during simulations. They help identify bugs early, enhance debugging efficiency, and ensure design reliability. For these reasons, we incorporated assertions into our design process.

The first module for which we implemented assertions is the Memory module. In this module, we introduced two assertions to verify design correctness:

1. The first assertion ensures that the writing phase operates as expected, confirming that during each write operation, the data shifts by one register.
2. The second assertion ensures that the reading phase functions correctly, verifying that during each reading operation, the data shifts by four registers.

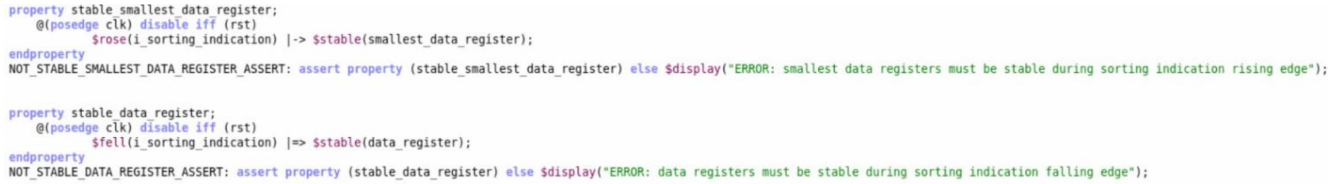


```
generate
    genvar j;
    for(j = 0; j < NUM_OF_REGISTERS - 1; j++) begin: WRITE_GEN_ASSERT
        property valid_write_phase;
            @(posedge clk) disable iff (rst)
                    (!i_wr_source & i_wr_rq) |-> (x_regs[j+1] == $past(x_regs[j]) & y_regs[j+1] == $past(y_regs[j]));
        endproperty
        WRITE_PHASE_ASSERT: assert property (valid_write_phase) else $display("Error: Register[%0d] did not get the value of Register[%0d]", j+1, j);
    end
endgenerate

generate
    genvar k;
    for(k = 0; k < NUM_OF_REGISTERS; k++) begin: READ_GEN_ASSERT
        property valid_read_phase;
            @(posedge clk) disable iff (rst)
                    (i_wr_source & i_wr_rq) |=> (x_regs[k] == $past(x_regs[(k + NUM_OF_REGISTERS -4) % 128]) & y_regs[k] == $past(y_regs[(k + NUM_OF_REGISTERS -4) % 128]));
        endproperty
        READ_PHASE_ASSERT: assert property (valid_read_phase) else $display("Error: Register[%0d] did not get the value of Register[%0d]", k, ((k + NUM_OF_REGISTERS -4) % 128));
    end
endgenerate
```

Figure 24: Memory Module Assertions

The second module for which we implemented assertions is the Bitonic Sort module. In this case, the assertions were written to verify that the sorting indication mechanism operates as expected.

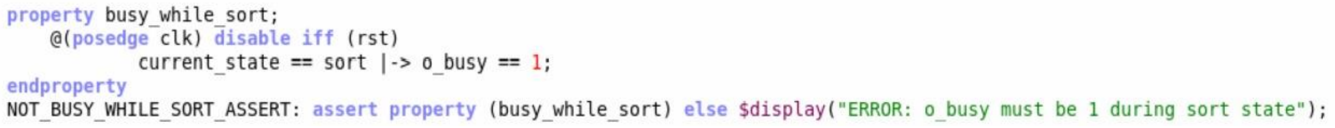
1. The first assertion ensures that during the first clock cycle after i_sorting_indication is triggered, data is written only to the data registers, while the values in smallest_data_registers remain stable.
2. The second assertion verifies that when i_sorting_indication transitions from 1 to 0, data is written only to the smallest_data_registers, while the values in data_registers remain stable.



```
property stable_smallest_data_register;
    @(posedge clk) disable iff (rst)
            $rose(i_sorting_indication) |-> $stable(smallest_data_register);
endproperty
NOT_STABLE_SMALLEST_DATA_REGISTER_ASSERT: assert property (stable_smallest_data_register) else $display("ERROR: smallest data registers must be stable during sorting indication rising edge");

property stable_data_register;
    @(posedge clk) disable iff (rst)
            $fell(i_sorting_indication) |=> $stable(data_register);
endproperty
NOT_STABLE_DATA_REGISTER_ASSERT: assert property (stable_data_register) else $display("ERROR: data registers must be stable during sorting indication falling edge");
```

Figure 25: Bitonic Sort Module Assertions

The final assertion is implemented in the top_chip module. It ensures that when the chip is in the Sort state, the o_busy output remains high.



```
property busy_while_sort;
    @(posedge clk) disable iff (rst)
            current_state == sort |-> o_busy == 1;
endproperty
NOT_BUSY_WHILE_SORT_ASSERT: assert property (busy_while_sort) else $display("ERROR: o_busy must be 1 during sort state");
```

Figure 26: Top Chip Assertion

Backend Design Section

13. Main stages

The following figure illustrates the main stages of the backend design, which will be described in detail in the subsequent sections.

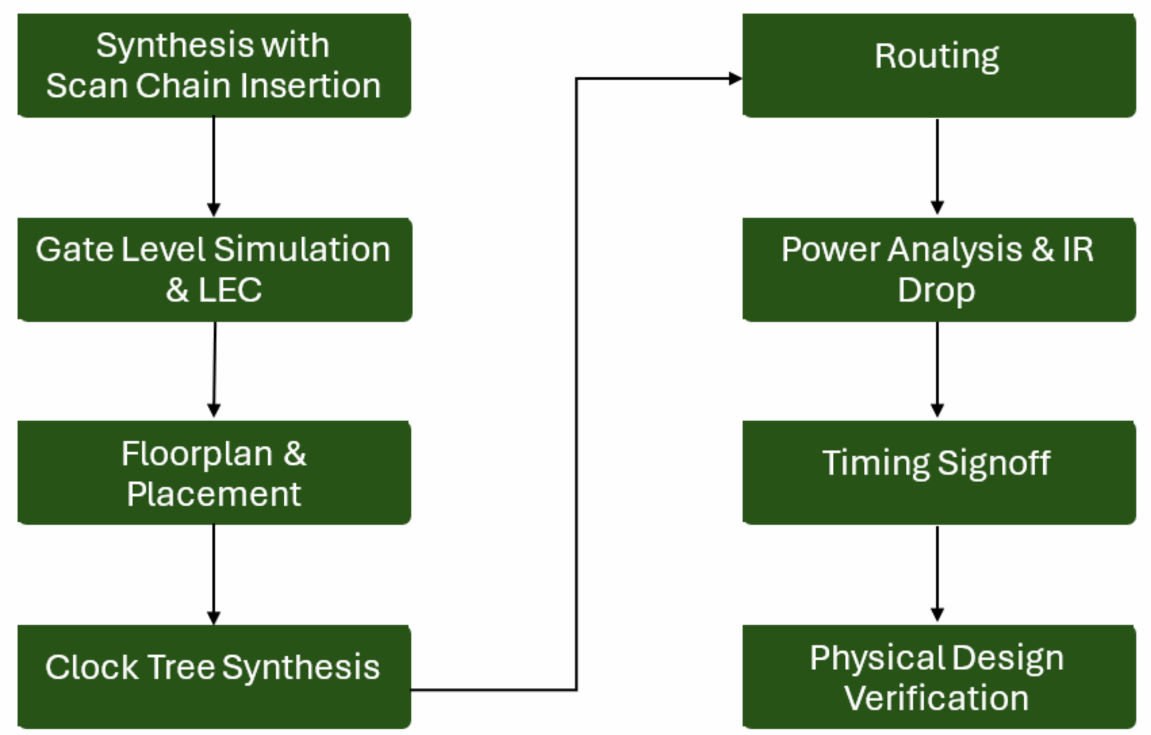

Synthesis with Scan Chain Insertion
Gate Level Simulation & LEC
Floorplan & Placement
Clock Tree Synthesis
Routing
Power Analysis & IR Drop
Timing Signoff
Physical Design Verification

Figure 27: Backend stages

14. Synthesis & Scan Chain Insertion

This stage involves translating the RTL code into a gate-level netlist using standard cell libraries. During this process, optimization algorithms are applied to minimize both area and power consumption while preserving the intended functionality of the design.
Additionally, scan chains are incorporated to enhance testability, and clock gating techniques are employed to further reduce power usage.

Synthesis was carried out using the Synopsys Design Compiler, targeting the TSMC 65 nm technology library (*tcbn65lplvttc1d0.db*).

For the synthesis, the following constraints were applied:

- **Clock frequency:** The design is required to operate at 100 MHz.
- **Scan chains:** Two scan chains were implemented, each with one data input and one data output, both controlled by an enable signal.
- **Clock gating:** No clock gating was applied in the design, the rationale for this decision is discussed in the LEC section.

15. LEC – Logical Equivalence Test

Logical Equivalence Checking (LEC) is a formal verification process widely employed in VLSI backend design to verify that the logical behavior of the synthesized netlist remains functionally identical to that of the original RTL description. This verification step is essential to ensure that any transformations and optimizations performed during synthesis—such as area, power, and timing optimizations—have not introduced unintended functional changes. By rigorously comparing the netlist against the RTL, LEC provides a high level of confidence in the correctness of the design before proceeding to subsequent backend stages.

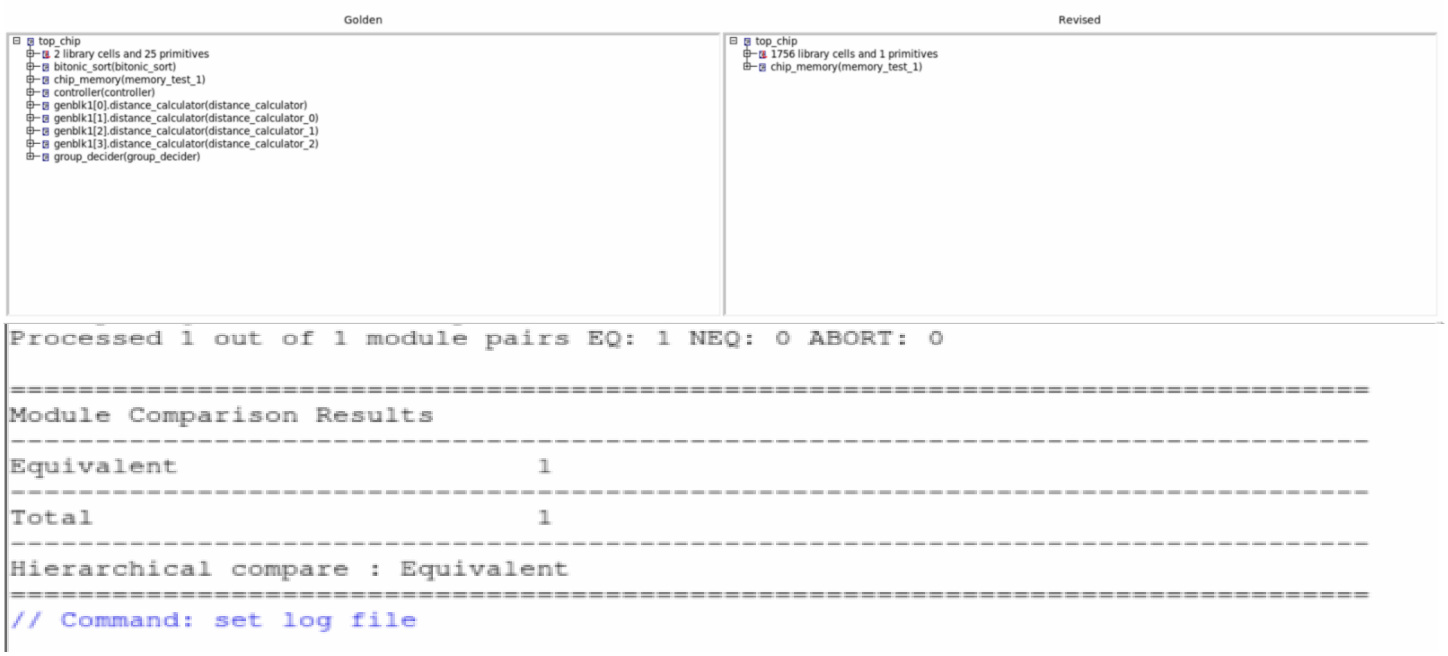
We used the Cadence Conformal LEC tool to check four cases of our design:

1. **Compile with no scan chains and no clock gating -** Initially, the bitonic sort unit failed the LEC check. To address this, we divided the unit into two sub-units (one sequential and one combinational), which together replicate the exact functionality of the original design. Following this modification, the LEC check passed successfully.
2. **Ultra compile with no scan chains and no clock gating:** The LEC check passed successfully without requiring further modifications.
3. **Ultra compile with two scan chains and no clock gating:** The LEC check passed successfully without requiring further modifications.
4. **Compile with two scan chains and clock gating:** The LEC check failed ,consequently, we decided to forgo clock gating insertion in order to proceed with the remaining stages of the design flow.

Ultimately, the selected configuration was Ultra compile with two scan chains and no clock gating.

During the backend process , we used the LEC tool in 2 stages:

1. RTL vs. post-synthesis gate level netlist (rvg).
2. Pre-layout gate-level netlist vs. post-layout gate-level netlist (gvg).



```
Golden
⊟ top_chip
  ⊞─ 2 library cells and 25 primitives
  ⊞─ bitonic_sort(bitonic_sort)
  ⊞─ chip_memory(memory_test_1)
  ⊞─ controller(controller)
  ⊞─ genblk1[0].distance_calculator(distance_calculator)
  ⊞─ genblk1[1].distance_calculator(distance_calculator_0)
  ⊞─ genblk1[2].distance_calculator(distance_calculator_1)
  ⊞─ genblk1[3].distance_calculator(distance_calculator_2)
  ⊞─ group_decider(group_decider)
```

```
Revised
⊟ top_chip
  ⊞─ 1756 library cells and 1 primitives
  ⊞─ chip_memory(memory_test_1)
```

```
Processed 1 out of 1 module pairs EQ: 1 NEQ: 0 ABORT: 0

=================================================================================
Module Comparison Results
---------------------------------------------------------------------------------
Equivalent                      1
---------------------------------------------------------------------------------
Total                           1
---------------------------------------------------------------------------------
Hierarchical compare : Equivalent
=================================================================================
// Command: set log file
```

Figure 28:LEC Output Results

16. Gate-Level Simulations

Gate-level simulations (GLS) are used to validate the functionality and timing of the design at the gate level after synthesis.
Unlike the RTL simulation performed during the frontend stage, which is based on an abstract representation of the logic, this verification uses an actual netlist containing real gates from the previously specified library and is targeted for the TSMC 65 nm technology.
This type of simulation ensures that the synthesized netlist satisfies all timing, power, and functional requirements under a variety of operating conditions.
We used the VCS tool by Synopsys to perform 2 GLS during the backend process:

1. Synthesized gate-level netlist – the output netlist of the synthesis stage, excluding pads.
2. Synthesized gate-level netlist – including pads.

In this project, gate-level simulations were performed primarily because the pads were manually inserted into the netlist. Consequently, it was critical to validate that all pads were correctly connected and that their functionality conformed to the intended design specifications.

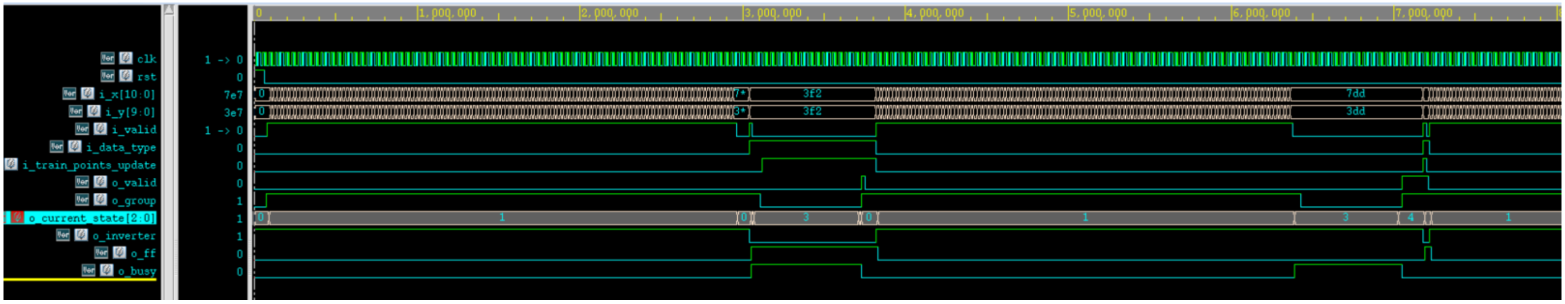


Figure 29: RTL Top Chip Simulation

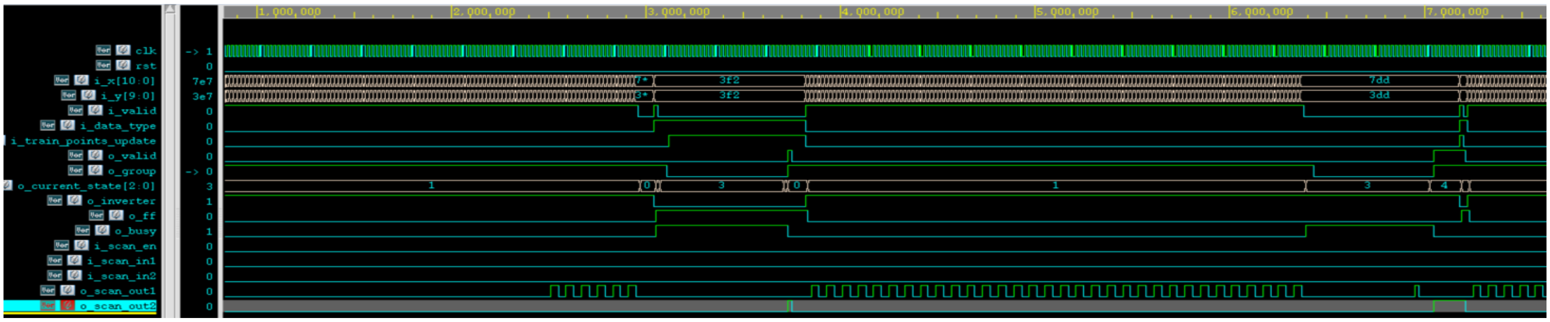


Figure 30: Synthesized Gate-Level Simulation

The above figures illustrate the waveforms obtained from both the RTL simulation (Figure 29) and the gate-level simulation (Figure 30). As expected, the results from both simulations are identical, confirming the functional equivalence of the netlist and the original RTL design.

17. Floorplan

Floorplanning represents the first stage of the layout flow. This process starts with defining the die, where the chip's physical dimensions are established based on area estimates derived from the gate-level netlist. Following the die definition, the placement of power and I/O pads is carried out.

Our design is a pad limited design, which means that the size of the core is determined by the number of pads.

Our die is with the following parameters:

- **I/O Pins –** Our die includes 48 I/Os pins, 26 data input pins, 3 scan chain input pins, 8 data output pins , 2 scan chain output pins and 9 power pins. During the die creation, we deliberately positioned the clock input pin as far as possible from the power supply pins to minimize interference and ensure signal integrity.
- **Die Size –** 1000x1060 [μm^2] includes the bond pads (that were added at a later stage of the backend). Without the bond pads the die size is 830x890 [μm^2]
  Note: from fabrication constraints the height of the chip must be exactly 1000 µm.
- Core to I/O boundary spacing – 52 µm.

The floorplan stage includes the power grid and power ring creation. Surrounding the pads, we constructed the Power Ring, which acts as the main route for distributing power around the chip. From the Power Ring, the Power Grid was routed to deliver power to each internal block.

The following figure illustrates the layout, highlighting the pads as well as the VDD, VSS, and clock signals, with the power supply located near the center and the clock routed farther away.

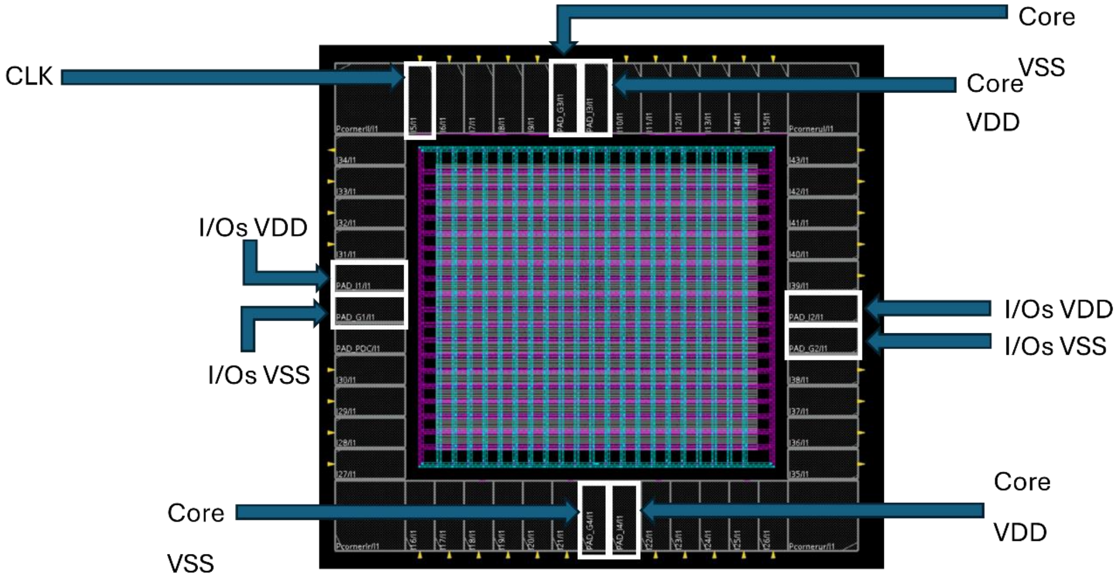


Figure 31: KNN Classifier Floorplan

18. Placement

In the placement stage, standard cells are methodically positioned on the chip to achieve an optimal layout that meets critical design objectives, including timing performance, wire length minimization, area efficiency, and power consumption.

The maximum cell density used in the design was 0.35, reflecting a careful balance between efficient area utilization and the avoidance of excessive congestion that could lead to increased IR drop.
The maximum cell density is used to minimize IR drop, the cells were intentionally spaced to ensure a balanced power distribution across the entire core area. However, lower placement density can affect timing, as the greater distances between cells result in longer interconnects, which in turn increase signal propagation delays.

The resulting placement, illustrated in the accompanying figure, shows a compact arrangement of cells concentrated around the core, with careful attention to maintaining order and balance across different regions.

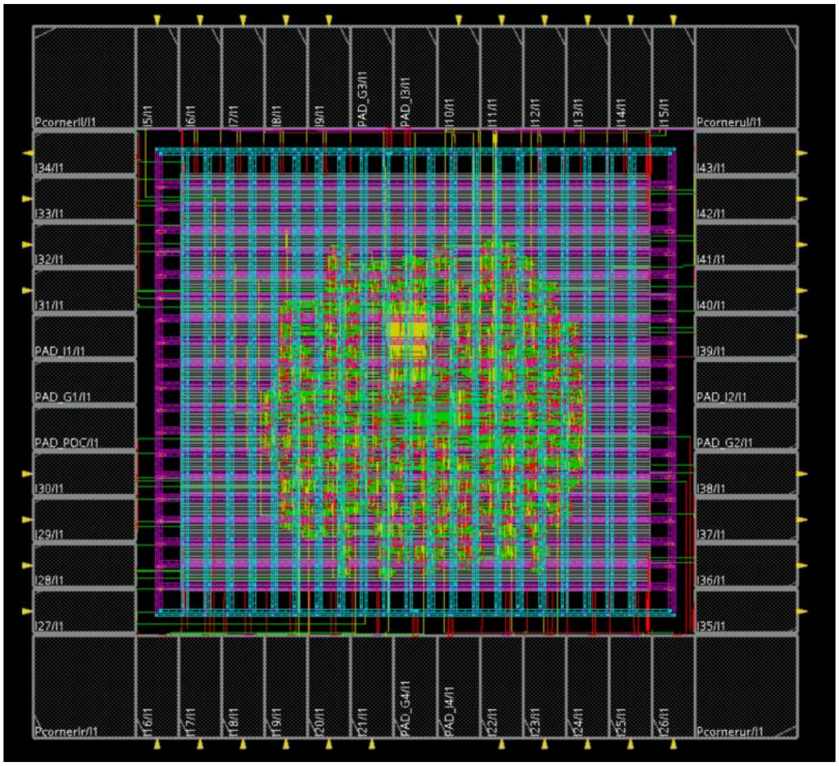


Figure 32: KNN Classifier Layout Post-Placement

19. Clock Tree Synthesis (CTS)

This critical process is designed to minimize clock skew, defined as the difference in arrival times of the clock signal at two registers, by strategically inserting buffers along the clock propagation paths. Additionally, it seeks to reduce insertion delay, which represents the time required for the clock signal to travel from the clock pad to the registers, thereby ensuring accurate and reliable timing performance across the design.

This process also ensures that the clock signal is delivered to all flip-flops in the design with minimal timing variation, thereby promoting synchronous operation and reliable circuit performance.

In our design, the maximum clock skew was measured at 60 picoseconds, and the maximum insertion delay was 280 picoseconds. These results indicate that the clock tree is relatively well-balanced.

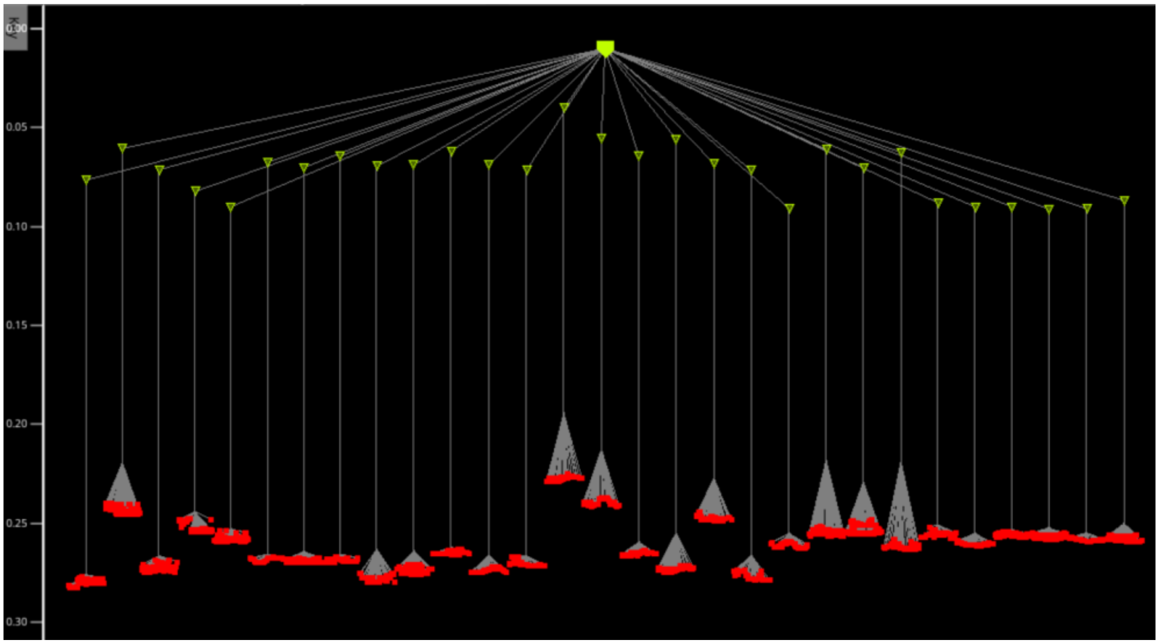


Figure 33: Clock Tree Synthesis

The following figure illustrates the distribution of the clock signal across the chip, highlighting the routing.

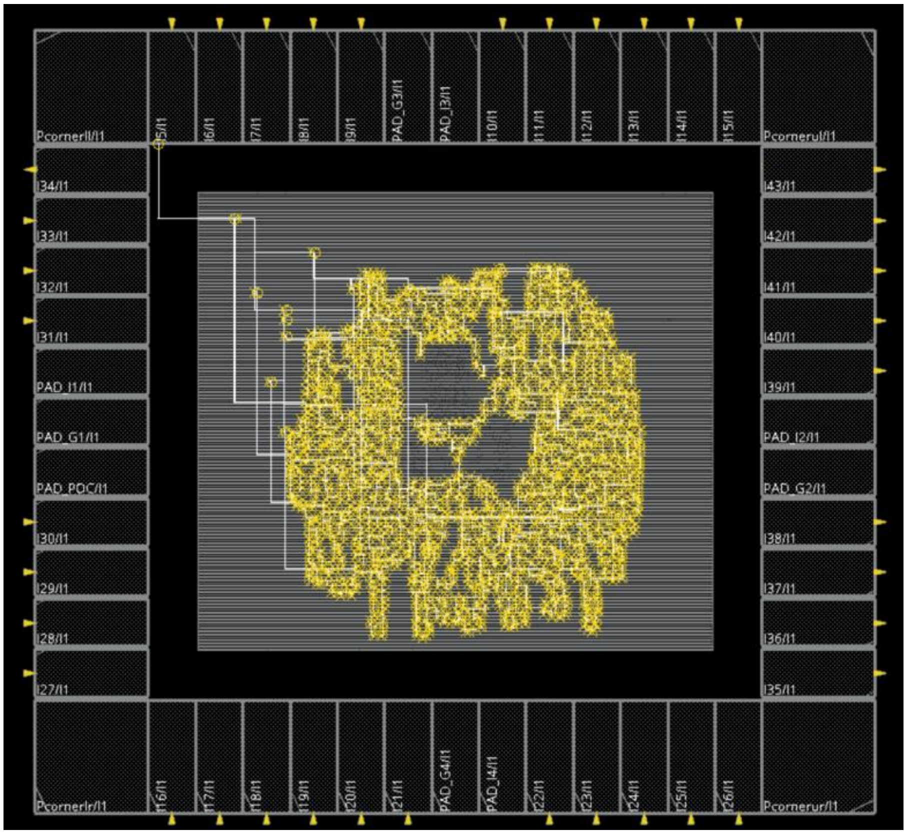


Figure 34: Routing of the Clock Signal

20. Routing

The routing process involves several key steps to establish proper electrical connections across the chip. First, metal layers are used to physically connect the signal pins. Next, the routing is optimized to efficiently link standard cells, macros, and I/O pins. Finally, electrical pathways are created using metals and vias according to the logical connections defined in the netlist. This ensures correct functionality, signal integrity, and reliable operation of the design

During the routing process, we employed two different routing commands:

1. **Sroute –** was used to connect all cells to the power supply rails, VDD and VSS. This involves establishing connections from the power grid through the metal layers to the standard cells, ensuring proper power distribution across the design.

2. **Nano route –** was used to interconnect the standard cells with each other, creating the necessary signal paths to implement the logical connections defined in the netlist.

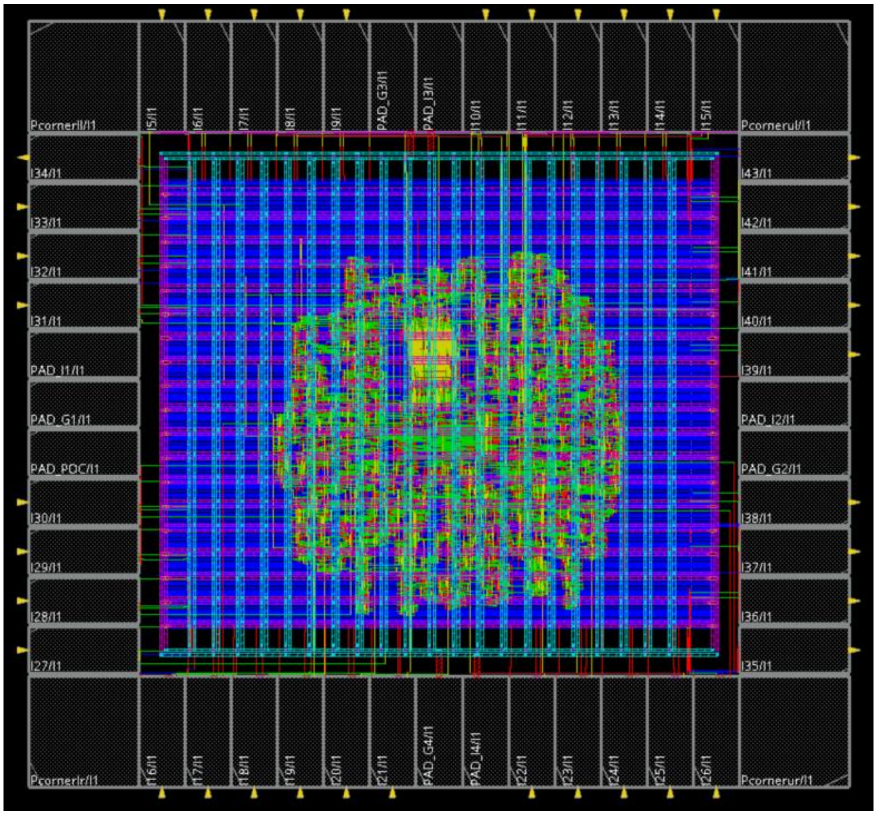


Figure 35: KNN Clasifier Post-Routing

21. Power Analysis and IR Drop

The primary objective of power analysis is to ensure that the design meets the required power constraints and does not exceed allowable consumption under various operating conditions. Accurate power estimation is critical for achieving performance targets while maintaining thermal and reliability limits. In this project, we performed both static and dynamic power analysis using the **Cadence Innovus** implementation tool. The analysis was carried out across multiple process-voltage-temperature corners, including fast and slow conditions, to evaluate worst-case scenarios.

To model realistic switching activity, we defined the following parameters for the simulations:

- Input activity: 0.2
- Flip-Flop activity: 0.4
- Clock Gate activity: 0.1 (configured but not utilized since clock gating was not implemented in the design).
- Frequency – 100 [MHz]

21.1 Static Power Analysis

Static power analysis evaluates the power consumed by the chip when operating under a constant current condition.
The table below presents the results of the static power analysis for our chip:

	Slow RC	**Fast RC**
Internal Power	15.3 [mW] (51.27%)	23.85 [mW] (55.53%)
Switching Power	14.48 [mW] (48.52%)	18.58 [mW] (43.26%)
Leakage Power	0.05 [mW] (0.21%)	0.5176 [mW] (1.21%)
Total power	**29.84 [mW]**	**42.95 [mW]**

Table 17: Static Power Analysis Results

21.2 Dynamic Power Analysis

Static power analysis evaluates the power consumed by the chip when it operates under varying current conditions.

In our chip the dynamic peak power consumption was 0.1396 [W] for both fast RC and slow RC.

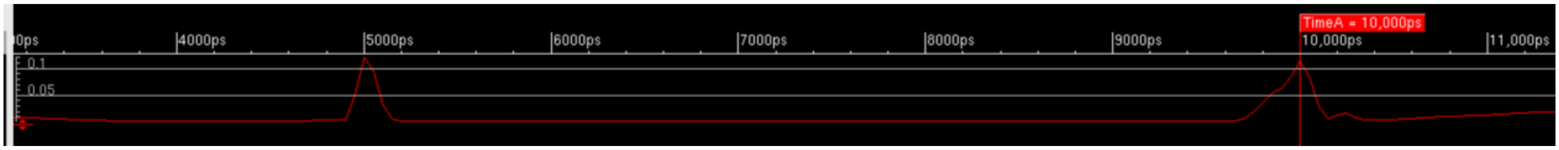


Figure 36: Dynamic Power Analysis Results

In the figure above, we can observe that the dynamic power peaks occur every 5 ns, corresponding to each clock toggle. The peak current consumption was approximately 0.122 A, and with a chip voltage of 1.2 V, this results in the peak power shown.

21.3 IR Drop

IR drop refers to the voltage drop that occurs along a conductor due to its inherent resistance when current flows through it. This effect becomes critical in integrated circuits because a significant voltage drop on power distribution networks (PDN) can degrade circuit performance, reduce noise margins, and, in severe cases, cause functional failures. After completing the power analysis, we performed an IR drop check on both VDD and VSS nets to ensure stable power delivery across the chip. The analysis identifies regions with the highest voltage drop, which typically occur in areas with dense logic or high current demand.

The IR drop limit at the project was set to below 7% of 1.2 [V], i.e. , should not exceed 84 [mV].

- VDD IR Drop: 13.28 [mV]
- VSS IR Drop: 12.87 [mV]

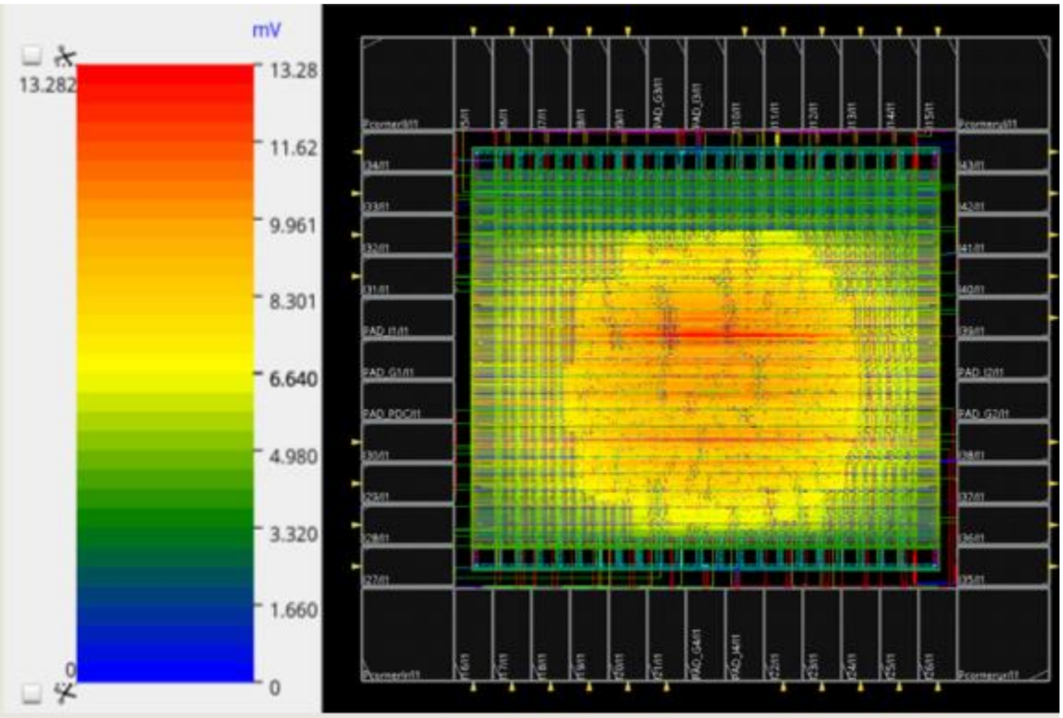


Figure 37: VDD IR Drop

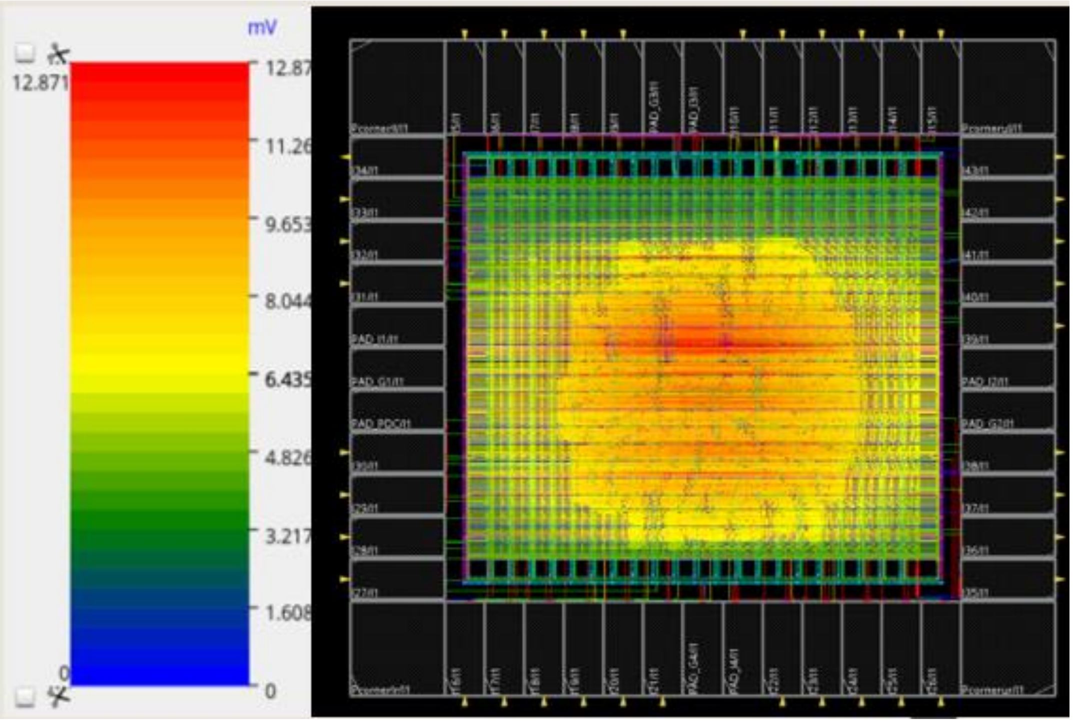


Figure 38: VSS IR Drop

22. Timing Signoff

Timming signoff perform STA (static timing analysis) before the fabrication process, it is essential to perform STA as a key verification step.
STA ensures that all logical paths in the design can function correctly at the target clock frequency, accounting for variations within the chip. For signoff-level verification, we used Synopsys PrimeTime, which allows us to identify and fix any timing violations in critical paths.
Timing analysis also incorporates On-Chip Variation (OCV), since identical components on the same chip may behave differently depending on their location. To achieve comprehensive coverage, both setup and hold constraints were checked across two process-temperature corners (-40°C and 125°C) and using three types of standard cells with different speed characteristics.

We applied a derate factor of 15% for both upper and lower limits and set the clock frequency to 100 MHz. Under these conditions, the analysis revealed a critical path delay of 9.885 ns, confirming that the design meets timing requirements.

During this stage, we analyzed the origin of the critical path delay and confirmed that it occurs within the Bitonic Sort module, as anticipated during the architecture and RTL implementation phases. Specifically, the critical path extends from the input data registers at the beginning of the module to the smallest data registers at the module's output.

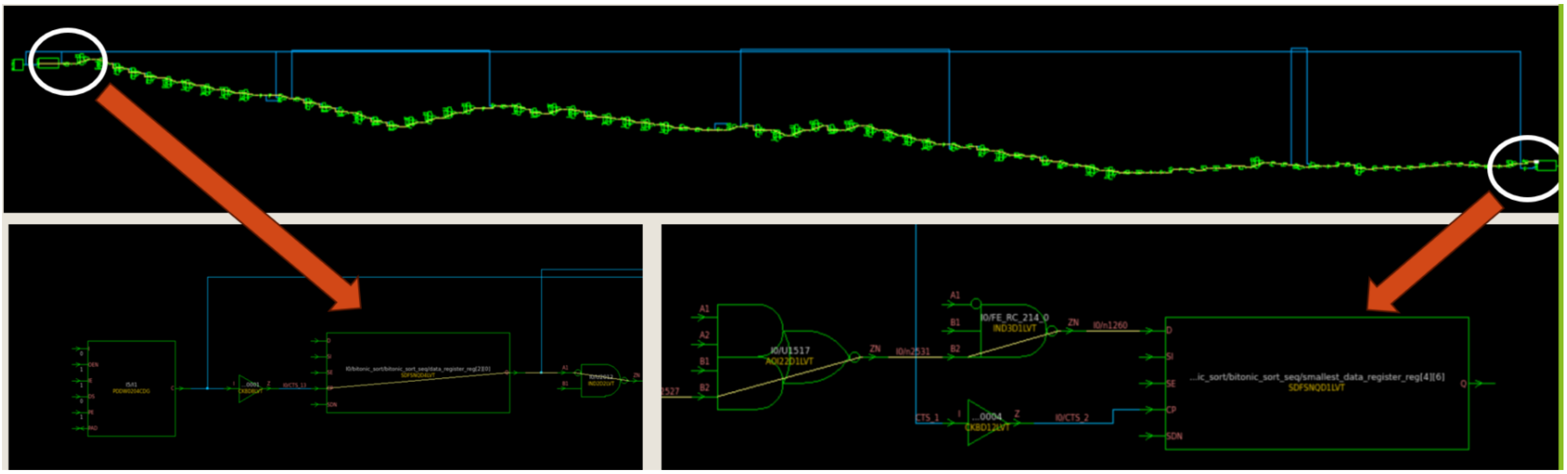
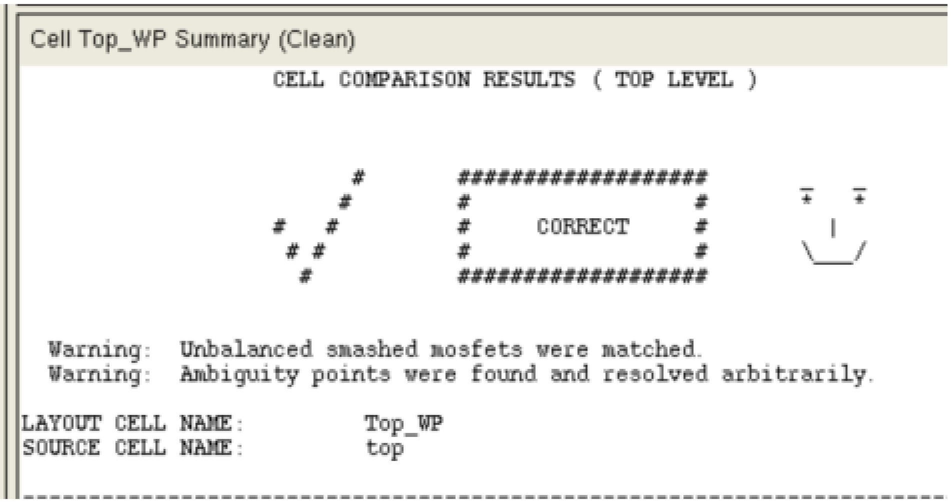


Figure 39: Critical Path

23. Physical Design Verification

The physical design verification contains 2 different processes:

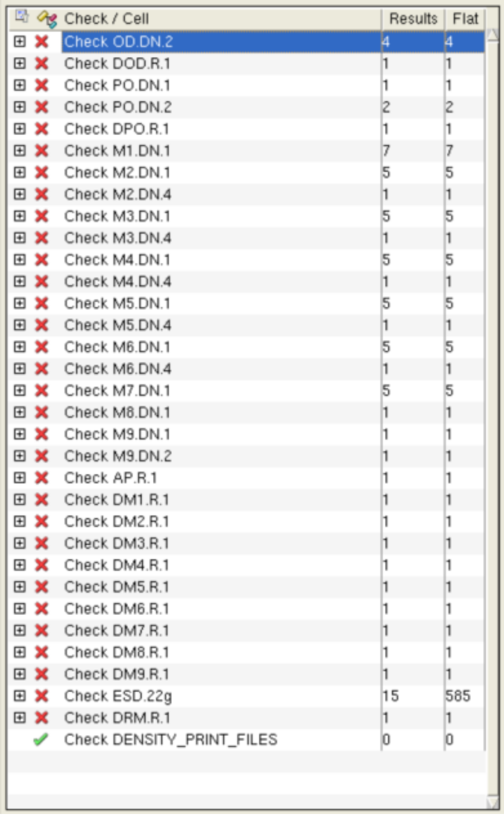
1. Layout vs. Schematics (LVS) - verification process in VLSI design that checks whether the physical layout of a circuit matches its logical schematic. It compares the extracted netlist from the layout with the original schematic netlist to ensure correct connectivity and device matching. The main goal is to detect errors like missing connections, extra components, or incorrect device parameters before tape-out, ensuring the design is functionally correct and manufacturable.



```
Cell Top_WP Summary (Clean)
                    CELL COMPARISON RESULTS ( TOP LEVEL )



                           #         ####################       _   _
                         #           #                  #       *   *
                    #    #           #     CORRECT      #         |
                     # #             #                  #       \___/
                      #              ####################


  Warning:  Unbalanced smashed mosfets were matched.
  Warning:  Ambiguity points were found and resolved arbitrarily.

LAYOUT CELL NAME:         Top_WP
SOURCE CELL NAME:         top

--------------------------------------------------------------------------
```

Figure 40: Successful LVS for the KNN Classifier

2. Design Rule Check (DRC) - verifies that the IC layout adheres to the foundry's manufacturing rules, such as minimum width, spacing, and overlap of layers. These checks ensure the layout can be fabricated without defects or yield issues. DRC is a critical step before tape-out to guarantee manufacturability and reliability of the chip.



	Check / Cell	Results	Flat
⊞ ✗	Check OD.DN.2	4	4
⊞ ✗	Check DOD.R.1	1	1
⊞ ✗	Check PO.DN.1	1	1
⊞ ✗	Check PO.DN.2	2	2
⊞ ✗	Check DPO.R.1	1	1
⊞ ✗	Check M1.DN.1	7	7
⊞ ✗	Check M2.DN.1	5	5
⊞ ✗	Check M2.DN.4	1	1
⊞ ✗	Check M3.DN.1	5	5
⊞ ✗	Check M3.DN.4	1	1
⊞ ✗	Check M4.DN.1	5	5
⊞ ✗	Check M4.DN.4	1	1
⊞ ✗	Check M5.DN.1	5	5
⊞ ✗	Check M5.DN.4	1	1
⊞ ✗	Check M6.DN.1	5	5
⊞ ✗	Check M6.DN.4	1	1
⊞ ✗	Check M7.DN.1	5	5
⊞ ✗	Check M8.DN.1	1	1
⊞ ✗	Check M9.DN.1	1	1
⊞ ✗	Check M9.DN.2	1	1
⊞ ✗	Check AP.R.1	1	1
⊞ ✗	Check DM1.R.1	1	1
⊞ ✗	Check DM2.R.1	1	1
⊞ ✗	Check DM3.R.1	1	1
⊞ ✗	Check DM4.R.1	1	1
⊞ ✗	Check DM5.R.1	1	1
⊞ ✗	Check DM6.R.1	1	1
⊞ ✗	Check DM7.R.1	1	1
⊞ ✗	Check DM8.R.1	1	1
⊞ ✗	Check DM9.R.1	1	1
⊞ ✗	Check ESD.22g	15	585
⊞ ✗	Check DRM.R.1	1	1
✓	Check DENSITY_PRINT_FILES	0	0

Figure 41: DRC Results for the KNN Classifier

Figure 41 shows that our initial DRC check reported several density-related errors. These issues were resolved by inserting dummy layers, including metals, polysilicon, and other necessary layers.

24. Final Layout

The final layout includes bond pads, which are I/O cells placed along the chip's perimeter to enable connections with external devices. The final chip dimensions are 1000 µm × 1060 µm.

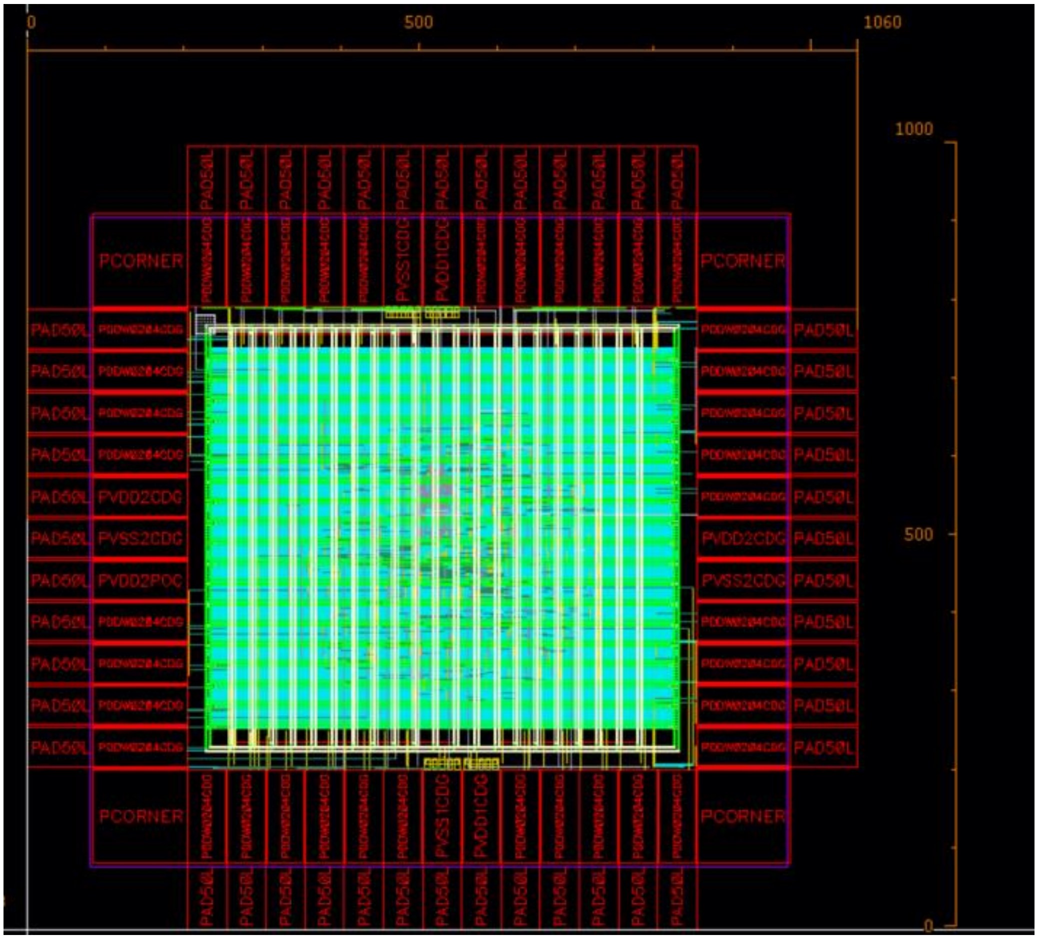


Figure 42: Final Layout - KNN Classifier

Possible Improvements

To meet the project requirements, we simplified our KNN classifier in terms of execution time, number of I/Os, chip size, and other constraints. The following chapter will discuss potential improvements for the KNN classifier.

1. **Different values of K –** In our implementation, we selected a constant value of K=5 for simplicity. However, the choice of K, which defines the number of nearest neighbors considered in prediction, can significantly affect performance. Typically, the optimal value of K is determined using a validation set, where additional data is allocated to find the best parameter. Choosing a very small K can make the model sensitive to noise, while a very large K may include points that do not accurately represent the correct classification. Therefore, using an adaptive K value could improve the performance and accuracy of our classifier.

2. **Using Euclidean Distance-** In our chip, we implemented the Manhattan distance for simplicity and ease of hardware implementation. However, using the Euclidean distance could potentially improve classification accuracy, as it takes into account the straight-line distance between points in all dimensions, providing a more precise measure of similarity between data points. This can be especially beneficial in datasets where the relationship between features is not aligned with the axes.

3. **Insertion of clock gating -** We explored the insertion of clock gating in our design as a means to reduce dynamic power consumption by disabling the clock in inactive regions of the circuit. However, as discussed in Section 15 (LEC), this modification led to LEC failures, indicating a mismatch between the gated-clock layout and the original schematic. Therefore, while clock gating could offer power savings, its implementation in our current design was not feasible without violating logical equivalence.

Summary and Conclusions

This project demonstrated the implementation of a KNN algorithm using a hardware accelerator, marking a significant step forward in our development as electrical engineers.

The front-end part of this project focused on designing and implementing a hardware-accelerated KNN algorithm. We explored the four key stages of front-end development: software implementation, chip architecture, RTL design, and verification. Throughout our studies at the Technion, we encountered each of these steps individually, but this project provided an opportunity to integrate them into a complete development process.

This experience reinforced the importance of maintaining high-quality standards at each stage, as a well-executed step simplifies and enhances the next. Additionally, we gained valuable insight into the iterative nature of problem-solving—recognizing that an initial, straightforward ("naive") solution can serve as a foundation for further optimization. This approach was particularly evident in our work on the Sort and Memory modules, where incremental improvements led to more efficient designs.

The back-end part of this project focused on the physical implementation stages, as discussed in the relevant sections. We worked with various software tools and scripts from different vendors to accomplish this stage, gaining practical experience with the detailed steps required in back-end design. Specifically, we explored synthesis, floorplanning, clock tree synthesis (CTS), placement, routing, and timing closure. Unlike the front-end stages, we had limited exposure to back-end design during our coursework at the Technion, so this project provided a valuable and comprehensive introduction to the full back-end flow, highlighting the careful attention to detail needed to produce a functional and manufacturable chip.

This project gave us valuable hands-on experience in the practical aspects of chip design, requiring careful consideration of trade-offs and constraints. It also enhanced our proficiency with industry tools and strengthened our ability to tackle challenges independently, providing a comprehensive exposure to back-end design processes.

We are deeply grateful to the VLSI lab staff for their continuous support throughout this project, and especially to our instructor, Goel Samuel, whose guidance was essential in bringing our work to completion. We also appreciate the collaboration and generous support from Apple Israel, with special thanks to Misbah Ramadan for

leading this effort. Our sincere thanks go to everyone who contributed to the success of this project.

References

[1] Y. Nami, Why Does Increasing k Decrease Variance in KNN? https://towardsdatascience.com/why-does-increasing-k-decrease-variance-in-knn-9ed6de2f5061/

[2] M. Hassan Najafi .et al, “Low-Cost Sorting Network Circuits using Unary Processing”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, pp. 1471-1480, Aug. 2018.