TÉCNICO LISBOA

Instituto Superior Técnico

Digital Systems Design

1º Semester 2021/2022

3º Report
k-Nearest Neighbors classification algorithm

Group 7

Nome	Número
André Pereira	90016
Tiago Gonçalves	90195

Day of the week:Friday, from 10:00 AM to 1:00 PM.

Professor: Horácio Neto

1 Problem definition

The problem tackled in this lab was the k-Nearest Neighbors classification algorithm, an algorithm used for classification and regression. The circuit needs to be able to classify the species of a iris flower from four given measurements. In order to classify an input, it is chosen the κ nearest elements in a training data set that have the best evaluation (in our case, we only need to be able to compute for κ = 1 or 3). In the given problem, the best evaluation comes from having the closest morphology, meaning, having the closest measurements from all the four measured attributes of the flower.

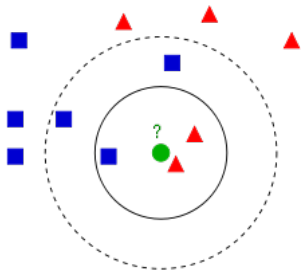

Figure 1: Representation of the points chosen by k-Nearest Neighbors algorithm. In this case, when κ = 3, the green dot (input) would be classified as a triangle, since it has the greatest representation on the group of the 3 closest elements to the input.

It is provided to the circuit a memory with a training data set of 108 measurements of flowers and their respective species (which can be *setosa*, *versicolor* or *virginica*), a single input that represents the 4 measurements of the flower targeted for evaluation and a ”flag” that indicates if κ is either 1 or 3.

- If it is 1, we only need to find the flower in the data set with the best evaluation and the species of the target flower will be the one from the best evaluated flower on the data set.
- If it is 3, we need to find the three flowers in the data set with the best evaluation and the species of the target flower will be either the one that has the greatest representation in the group, or if they are all represented equally, it will be the best evaluated flower on the data set.

2 Moore Machine

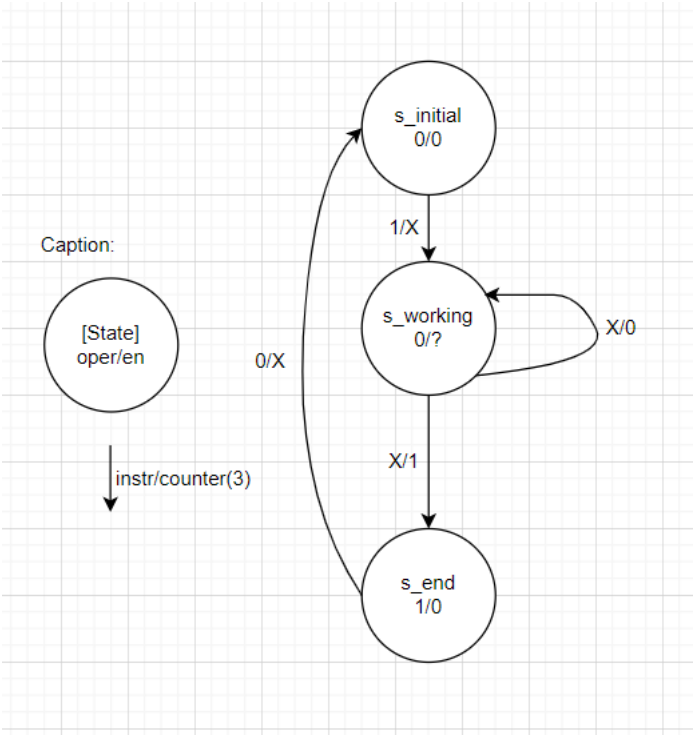

Figure 2: Moore state machine. Each circle represents the state, while showing the std_logic_vector ”oper” on the left and ”en” on the right. Each arrow represents the next state decision making, made possible by the std_logic ”instr” and ”counter(3)”.

The Moore machine is a little different from the usual ones. This time the control unit will have in mind the pipeline of the circuit. It will also need to loop through all the addresses of the memory, notice that this variable is not in the Moore machine because will have multiple values in the same state. Also the address variable (raddr) is a registor, that means that will only change on the rising edge of the clock. Let's see how this works.

First the initial state. This state is the usual initial state, all variables set to 0, also the raddr will be set to all zeros too. While the signal instr is '0' the state doesn't change. However, when instr is one, the next state switches to s_working.

The working stage will be the tricky part. Here raddr will need to increment until all the memory has been read. So the way this is done is by having an internal signal (flag). This flag will be responsible to increment the raddr, so when flag is equal to '1' the address will increment, if the flag is '0' then the raddr will remain the same. When the raddr reaches the end of the memory, a new counter is initialized. The counter was set to zero before in the working stage, while the raddr iterates through the memory, and this counter will now start to increment, and this is the counter that appears in the Figure 2 (this is the counter that will be waiting for the pipeline to finish). So, when the

counter reaches 4 (”100”, that means counter(3) equals to ’1’), the machine can finally go to the next and final stage. The en variable is also changing in this state, when raddr is iterating through the memory, this en is set to ’1’, otherwise is set to ’0’.

In this stage the oper variable will finally be set to ’1’, this oper will be responsible to tell the circuit that it can finally compute the final results, namely the class of the flower. Finally, when the instr signal is set to 0, the machine returns to the initial state.

3 Design

To perform the KNN algorithm, we will have to read from memory the 108 training instances, that will be used to make the classification. Meaning, we will need at least to access the memory containing the values 108 times. Besides, after accessing each set of the 4 measurements, we have to compute the distance that the target flower is from these four attributes, in order to be able to decide the species, and if κ is either 1 or 3 we need to keep in track possibly more than one entry of the memory. Therefore, it was decided that in order to optimize the number of cycles of the circuit, it should be used a pipeline structure, like so:

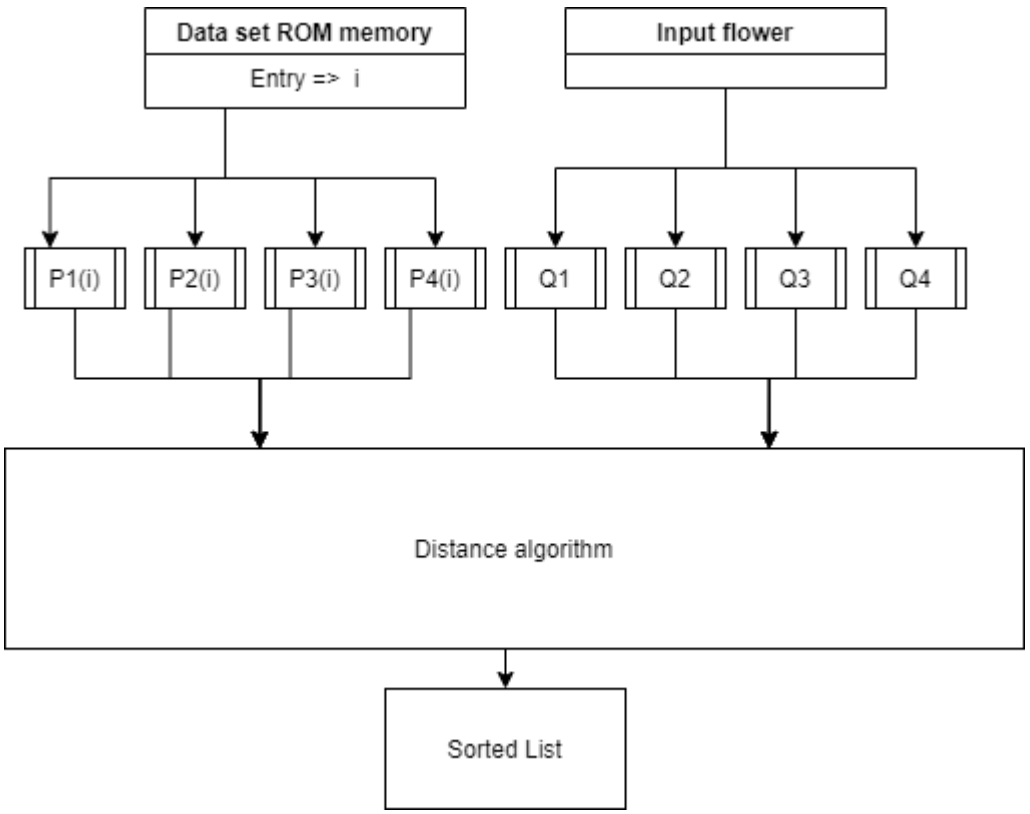

Figure 3: Circuit pipeline.

We had to import from the previous lab the distance circuit, and adapted these two:

- The distance algorithm performed in the previous lab, such that it now works in a pipeline fashion and works on real numbers represented in unsigned fixed-point format Q3.9 .
- The memory entries from the training set are read has a single input stream of 48 bits, instead of 4 entries of 12 bits each. This means that we can read all the flower measurements from the training set in a single clock cycle, instead of four.

The input of the circuit has 48 bits. This is a problem because the FPGA only has 16 switches, so the way our circuit stores the input is by storing each attribute individually. The first 12 switches (0 to 11) are the values of the attributes, and we use the buttons in order to store the desired attribute. The way to see each attribute in the LCD is with switches 13 and 12. If they are ”00”, the first attribute is shown, if it is ”01” then the second attribute is shown, ”10” is for the third attribute, and finally ”11” is for the fourth and last attribute. The input will now be the concatenation of all these four attributes (first attribute is for the less significant bits, and so on). Finally the LED's will give us the output. LED 14 is when our instr button is on and the LED's 0 to 2 are responsible for representing the result: ”001” for 01 class; ”011” for 10 class; ”111” for 11 class.

Table 1: FPGA IO

I/O	Function
BTNR	Stores the 12 lest significant bits of the input flower
BTND	Stores the bits [23:12] of the input flower
BTNC	Starts the KNN algorithm (sets oper flag to 1)
BTNU	Stores the bits [35:24] of the input flower
BTNL	Stores the 12 most significant bits of the input flower
SW15	Global reset for the circuit
SW14	Sets the κ of the algorithm to be either 3 (when the switch is active) or 1 (when otherwise)

3.1 Distance circuit: Pipeline

$$D(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2 + (p_4 - q_4)^2$$

Figure 4: Target equation to be implemented in the circuit, where p_n portrays the measurements from the data set flowers and q_n portrays the measurements from the input flower.

In order to perform said equation, we need to do the operations step-by-step:

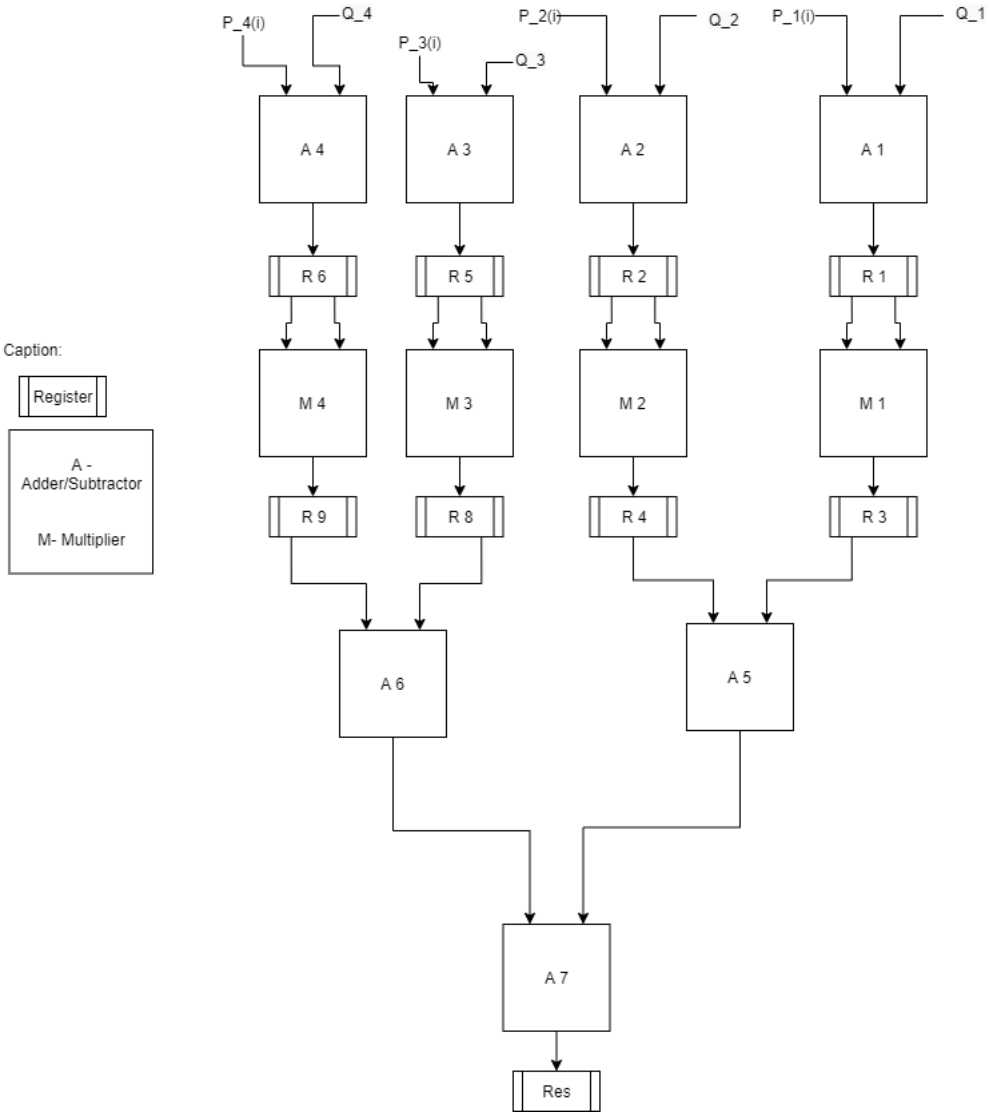

Figure 5: Distance circuit pipeline.

By reshaping the original circuit, it is possible to improve the circuit with pipelining. By drawing it like Figure 12, we can have ”circuit floors” that greatly speedup the iterations of the algorithm. Even though each iteration still takes the 3 cycles to complete, now each iteration can run concurrently.

- In the first clock cycle, we only use floor one and calculate the terms corresponding to (p_1 - q_1),(p_2 - q_2),(p_3 - q_3) and (p_4 - q_4) of the first memory entry.
- In the second clock cycle, we use the first floor to calculate the terms corresponding to (p_n - q_n) of the next memory entry, but also use the second floor to compute $(p_n - q_n)^2$.

- Finally, the final floor calculates the result from the first memory entry (unsigned fixed-point format Q8.18) and the rest of the floor continue to perform the pattern explained previously.

Thus, each memory entry does take 3 cycles to calculate the algorithm result, but now the total amount of cycles needed to calculate all 108 memory positions would be 111 cycles instead of the previously implementation without pipeline, that would take 324 cycles.

3.2 Insert sort unit

To implement the species decision from the KNN algorithm, it was used a insert sort unit, that stores the three flowers from the data set with the best evaluation, whether κ is equal to 1 or 3. The insert sort unit is implemented as a cascade of register-based cells. Each cell stores the distance (in a Q8.18 format) and species (that takes up 2 bits) of an ordered element. The distance and species of the 108 flowers from the data set are fed as input to the list, and will be sorted inside the list based on the distance, where each cell can either:

- Insert directly a new element and shift the old to the following-cell.
- Receive a shifted-element from the previous cell and shift the old to the following-cell.
- Simply not store the value since it has a distance value higher than the one currently stored.

Besides, the species are stored such that:

- class=1 $\longrightarrow$ setosa.
- class=2 $\longrightarrow$ versicolor.
- class=3 $\longrightarrow$ virginica.

In this problem, the list will be sorted in a descending order of the distances, meaning, the flower with the best evaluation (lowest distance computed from the input flower) will be stored in the first cell.

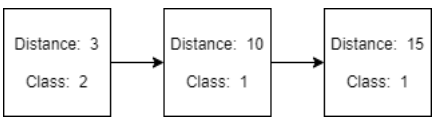

Figure 6: Example of a sorted list of flowers.

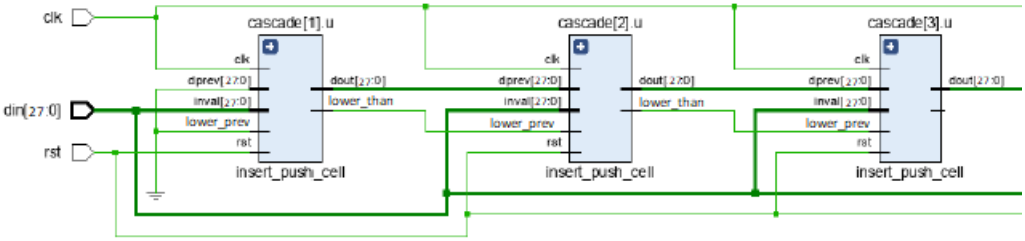

Figure 7: Organization of the cells inside the sorted list circuit, where din, dprev and inval contain both the distance and the species of the flower from the data set (26 bits for distance and 2 bits for the class).

After all 108 entries from the data set are analyzed, the circuit will decide the species of the input flower based on the sorted list and the κ:

- If κ is equal to 1, the circuit purely takes a look at the firs cell and decides that the species of the input flower is equal to the stored class in that cell.
- If κ is equal to 3, the circuit takes a look at the most common class from all the cells and decides that the species of the input flower is equal to the most represented class. If all the classes are represented equally, the chosen species will be the one with the closest distance to the input flower, meaning, the one in the first cell.

4 Benchmark and results obtained

Now we simulate our circuit and the results are shown in Figure **??** and Figure **??**. It was tested the flower with the given measurements below (in bits):

- Sepal length $\longrightarrow$ 110000000000 $\longrightarrow$ 6.0 cm
- Sepal width $\longrightarrow$ 011000000000 $\longrightarrow$ 3.0 cm
- Petal length $\longrightarrow$ 100110011001 $\longrightarrow$ 4.8 cm
- Petal width $\longrightarrow$ 001110011001 $\longrightarrow$ 1.8 cm
- The class result is stored as Iris-virginica.

In this test we ended up with a sorted list (after analyzing the 108 entries from the data set) with the classes 3,2 and 2 (the list is stored in the variable wires_classes and sorted by best evaluation), which translates to the species virginica, versicolor and versicolor, in this order.

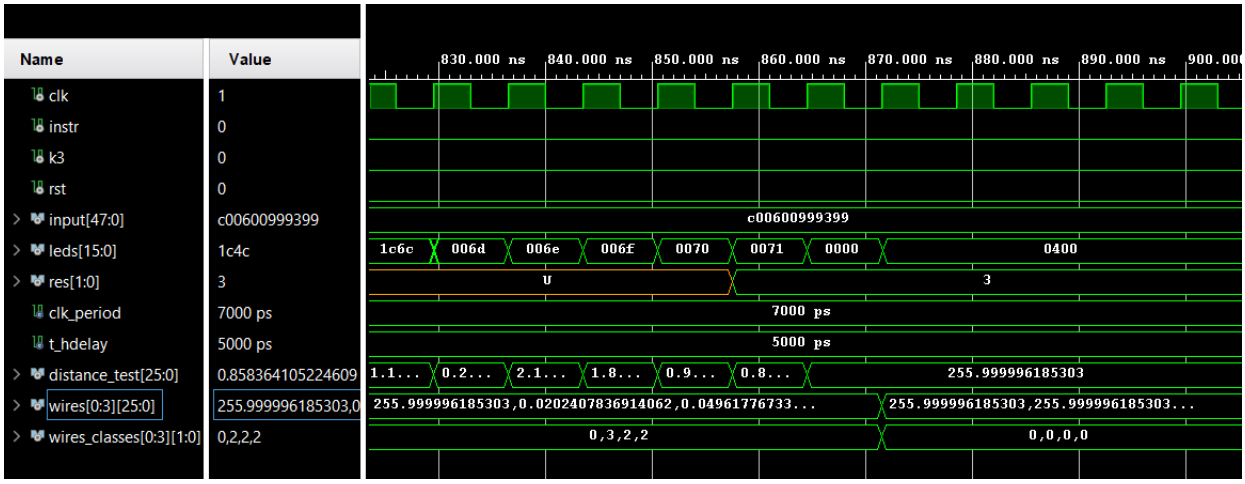

Figure 8: Testbench case 1
Species classification when κ = 1.

When κ = 1, the circuit classifies the input flower as virginica (the chosen class is given by the res signal).

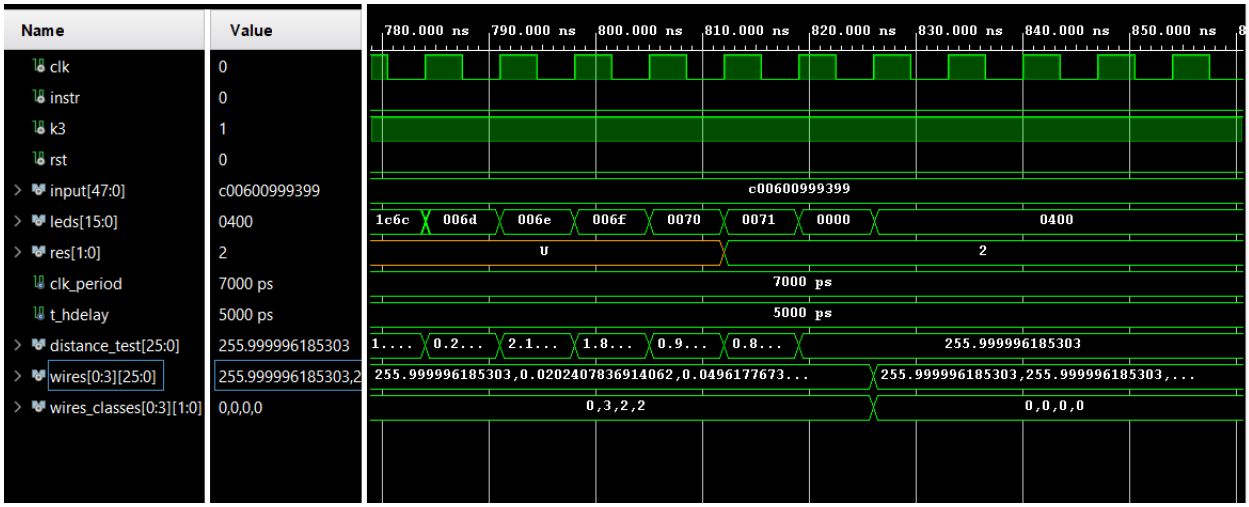

Figure 9: Testbench case 2
Species classification when κ = 3.

When κ = 3, the circuit classifies the input flower as versicolor (the chosen class is given by the res signal).

5 Timing analysis

By looking at the timing analysis with a 10 ns clock, we can conclude that we can use a faster clock (supposedly we have a 4,786 ns speedup margin). So we tried to use a 5,214 ns clock instead of the 10 ns clock that we had originally, and the circuit still works as intended.

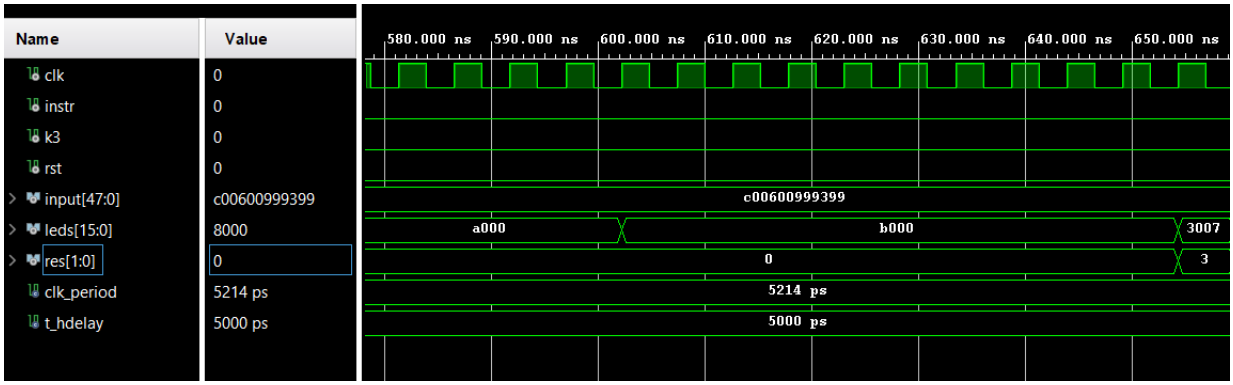

Figure 10: Testbench case 3
Species classification when $\kappa = 1$ with the new clock.

Since the circuit only starts working at the 60 ns mark and get the result at 655 ns (approximately), which means it only needs 595 ns to classify an input flower.

6 Resource Consumption

Table 2: Resource Consumption

Resource	Utilization	Available	Utilization %
LUT	187	20800	0.9
LUTRAM	4	9600	0.04
FF	352	41600	0.85
BRAM	1	50	2
DSP	4	90	4.44
IO	70	106	66.04

We were able to implement and utilize the k-Nearest Neighbors classification algorithm with a low utilization of the LUT's from the FPGA circuit board.