

# CS577: PROJECT REPORT

Project Name:	Digit Recognition
Group Number:	16
Name of the Top Module:	DigitRec
Github Repository Link:	<a href="https://github.com/Shivank-thapa/VLSI_Project.git">https://github.com/Shivank-thapa/VLSI_Project.git</a>

**Submitted By:**

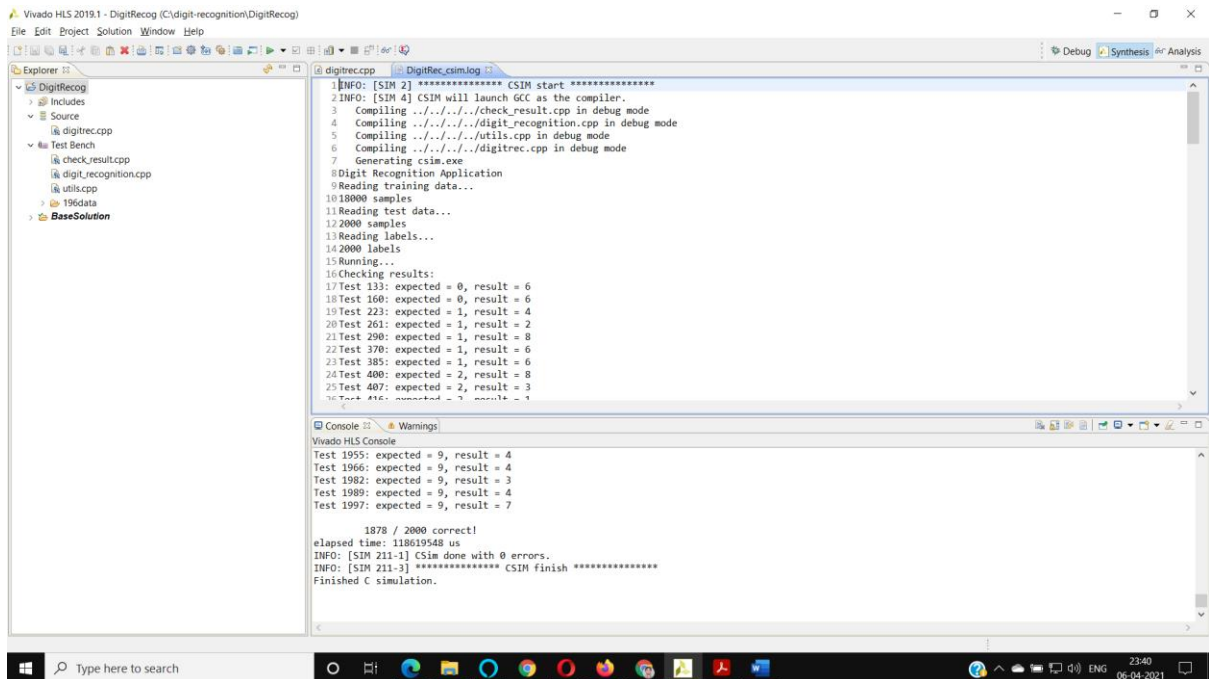
**Group Members Details:**

Group Member Name	Roll Numbers
Shivank Thapa	204101053
Hemant Regar	204101027
Gaurav Kumar	204101068
Dan Singh Pradhan	204101020
Ashish Kumar	204101015

## INTRODUCTION:

## 1. Running the algorithm

### 1.1 C Simulation Screenshot:



### 1.2 C-Synthesis Screenshot:

## Synthesis Report for 'DigitRec'

## General Information

Date: Sat Apr 10 19:39:38 2021  
Version: 2019.1 (Build 2552052 on Fri May 24 15:28:33 MDT 2019)  
Project: DigitRecognition  
Solution: solution1  
Product family: kintexuplus  
Target device: xcku5p-ffvb676-2-e

## Performance Estimates

### ▣ Timing (ns)

#### ▣ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.474	1.25

### ▣ Latency (clock cycles)

#### ▣ Summary

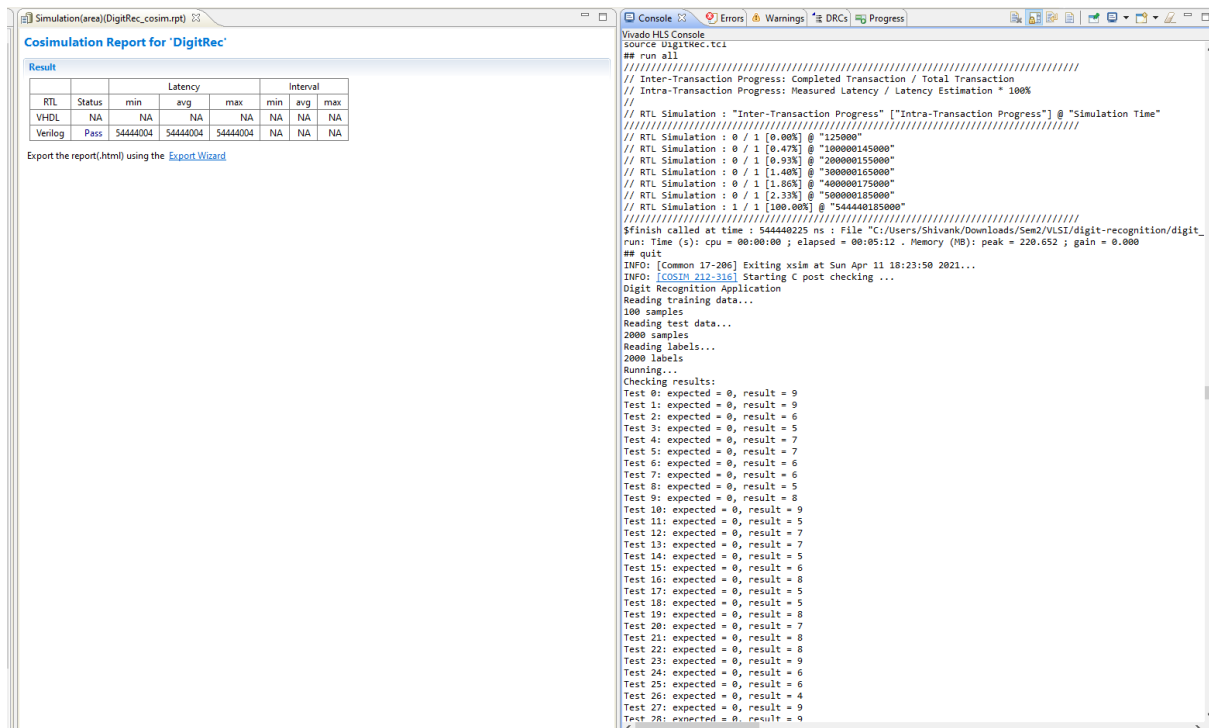
Latency		Interval		
min	max	min	max	Type
36004	9584050004	36004	9584050004	none

## Utilization Estimates

### ▣ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	4	0	1041	-
FIFO	-	-	-	-	-
Instance	0	-	399	1728	0
Memory	287	-	22	6	0
Multiplexer	-	-	-	308	-
Register	-	-	1059	-	-
Total	287	4	1480	3083	0
Available	960	1824	433920	216960	64
Utilization (%)	29	~0	~0	1	0

## 1.2 Co-Simulation Report



The screenshot displays the Vivado HLS Co-simulation Report for 'DigitRec'. The report is divided into two main sections: a 'Result' table and a 'Console' log.

**Result Table:**

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	54444004	54444004	54444004	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

**Console Log:**

```
Vivado HLS Console
source digitrec.tcl
## run all
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
//
// RTL Simulation : 0 / 1 [0.00%] @ "125000"
// RTL Simulation : 0 / 1 [0.47%] @ "100000145000"
// RTL Simulation : 0 / 1 [0.93%] @ "200000155000"
// RTL Simulation : 0 / 1 [1.40%] @ "300000165000"
// RTL Simulation : 0 / 1 [1.86%] @ "400000175000"
// RTL Simulation : 0 / 1 [2.33%] @ "500000185000"
// RTL Simulation : 1 / 1 [100.00%] @ "544440185000"
//
// Finish called at time : 544440225 ns : File "C:/Users/Shivank/Downloads/sem2/VLSI/digit-recognition/digit-
run: Time (s): cpu = 00:00:00 ; elapsed = 00:05:12 . Memory (MB): peak = 228.652 ; gain = 0.000
## quit
INFO: [Common 17-206] Exiting xsim at Sun Apr 11 10:23:50 2021...
INFO: [COSIM 212-316] Starting C post checking ...
Digit Recognition Application
Reading training data...
100 samples
Reading test data...
2000 samples
Reading labels...
2000 labels
Running...
Checking results:
Test 0: expected = 0, result = 9
Test 1: expected = 0, result = 9
Test 2: expected = 0, result = 6
Test 3: expected = 0, result = 5
Test 4: expected = 0, result = 7
Test 5: expected = 0, result = 7
Test 6: expected = 0, result = 6
Test 7: expected = 0, result = 6
Test 8: expected = 0, result = 5
Test 9: expected = 0, result = 8
Test 10: expected = 0, result = 9
Test 11: expected = 0, result = 5
Test 12: expected = 0, result = 7
Test 13: expected = 0, result = 7
Test 14: expected = 0, result = 5
Test 15: expected = 0, result = 6
Test 16: expected = 0, result = 8
Test 17: expected = 0, result = 5
Test 18: expected = 0, result = 5
Test 19: expected = 0, result = 8
Test 20: expected = 0, result = 7
Test 21: expected = 0, result = 8
Test 22: expected = 0, result = 8
Test 23: expected = 0, result = 9
Test 24: expected = 0, result = 6
Test 25: expected = 0, result = 6
Test 26: expected = 0, result = 4
Test 27: expected = 0, result = 9
Test 28: expected = 0, result = 9
```

## 2. Algorithm

**Digit Recognition** uses K-nearest-neighbour algorithm to classify hand-written digits. The algorithm uses Hamming distance in addition to KNN voting to determine a digit in a given image.

The algorithm begins by computing the Hamming distance between the test input and each training sample. The labels of K training samples with shortest distances are stored and the voting algorithm then chooses among the K-labels to decide the label of the test sample.

**Hamming distance** computation uses Manhattan distance to find the similarity/distance between two samples. Each sample is an array of pixel values from the image; therefore, the Manhattan

distance is computed as bitwise XOR of the two samples and counting the number of set bits in the result.

**Voting algorithm** compares all the Hamming distances to find the K-nearest training samples and gives as a result the most frequently occurring label amongst them. The main computation inside Voting algorithm is sorting the distances and comparing them to find most frequent label.

There are two tuneable parameters included in the design:

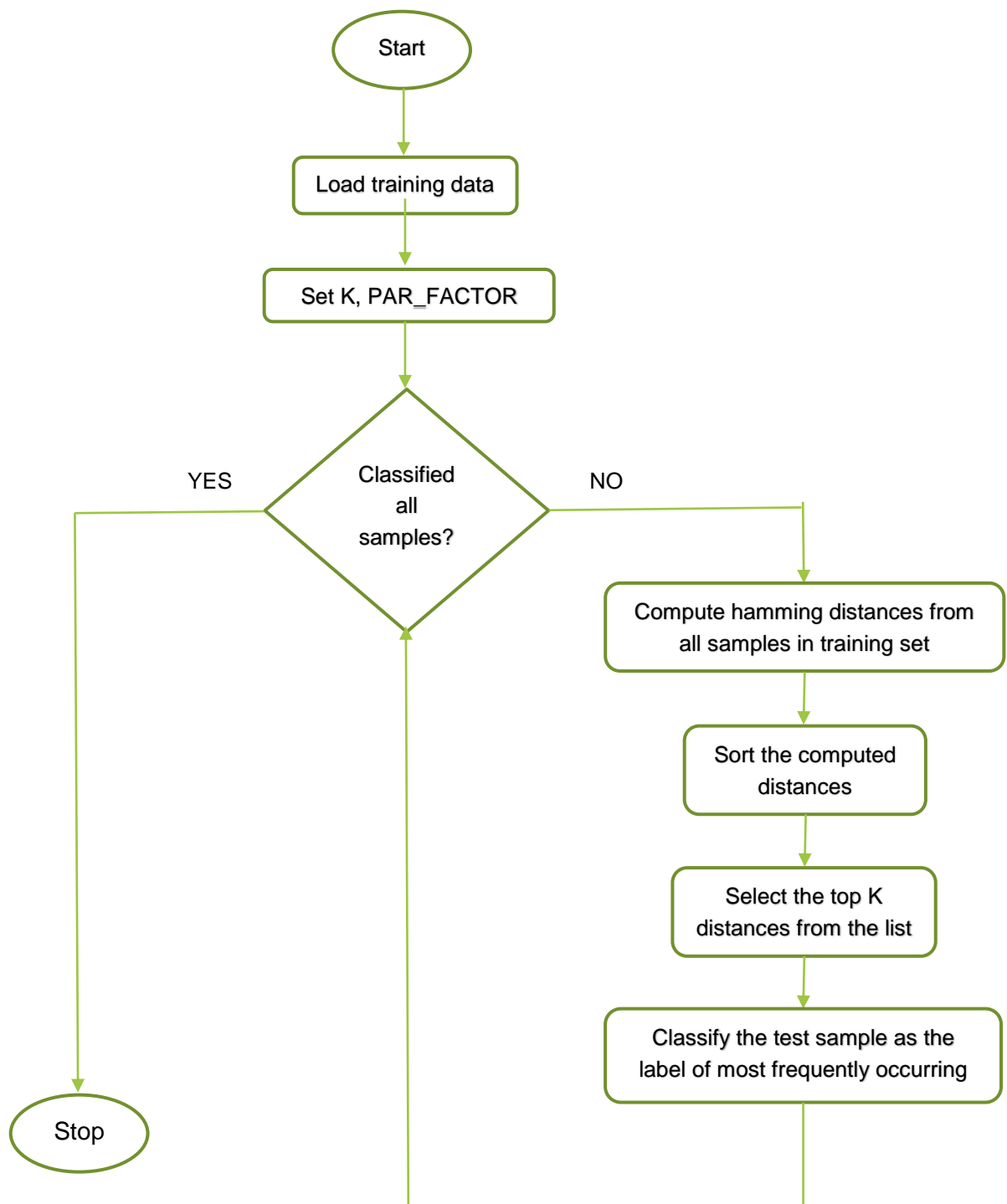
**K:** denoting the number of nearest neighbours.

**PAR\_FACTOR:** denoting the number of parallel Hamming distance units.

### 3. Data

The experiment uses a down-sampled subset of MNIST handwritten digits database. Every image is down-sampled to a 14X14 pixel image and stored in a 196-bit unsigned number. There are 18000 training samples and 2000 testing samples used which are evenly split amongst 10 classes for digits.

## 4. Flowchart



## 5. Result

FPGA Part	Name of the Top module	LUT	FF	BRAM	DSP	Latency	Interval
kintexuplus	DigitRec	3083	1480	287	4	9584046004	9584046004

### Result Explanation:

- Our top function is DigitRec().
- We have used kintexuplus target device for the project simulation.
- In top function total 1480 flip-flops are used.
- 3083 Lookup table is used in which 308 is used for multiplexer and 1041 is used for Expressions.
- 287 BRAM is used as memory to store data during execution. 257 units of BRAM are used to store training data and 29 units of BRAM used for testing data. And 1 BRAM used for storing final result.
- 4 DSP(data signal processing) is used in this project.
- Latency of the top function is 9584046004 cycles.
- Initialization interval is also same as latency.

### Problems faced and Solution:

- **Unknown value of Latency and Initiation interval:** In case the bound on the number of time loop will execute if it is unknown or dynamic then in such cases C- Synthesis result may give unknown value of latency and initiation intervals.  
**Solution:** By the use Loop TRIPCOUNT Pragma, we can specify that the Minimum and Maximum number of iterations the Loop will execute. And this resolved our problem of unknown latency and interval.

- **C/RTL Co-Simulation taking Long Time to Complete:**  
C/RTL Co-Simulation step is taking 7-8 hours to generate Results, so we have to wait a lot to check the status of Verilog.

## 6. Optimization:

Benchmark	Optimization Type (Latency/Area)	Resource Utilizations				Latency		Major Optimizations
		BRAM _18K	DS P	FF	LUT	Cycle time (in ns)	Total clock cycle	
Baseline solution	Base Solution	287	4	1480	3083	6.47	9584046004	Baseline, loop_tripcount
Optimization Type 1	Latency	320	4	9412	12776	8.66	72128002	Number of clock cycle
Optimization Type 2	Area	289	4	1198	2164	6.46	9584046004	LUT, FF

### Explanation for the optimization 1(Latency):

- In this optimization, our aim is to improve the latency by applying some pragma.
- We applied the ARRAY\_PARTITION pragma in the array to which results in RTL with multiple small memories or multiple registers instead of one large memory. This pragma effectively improves the throughput of the design.
- We are applying ARRAY\_PARTITION completely to partition the various array into the different memory blocks.
- This pragma requires more memory instance and registers which comparatively reduces the latency.



- We are using UNROLL pragma to unroll the loop content completely which reduces the latency on cost of the increasing hardware resources.
- We are using PIPELINE pragma also to pipeline the loop instruction which results in reducing the number of clock cycle requirement.

## Explanation for the optimization 2(Area):

- In this optimization our aim is to optimize the resources. In one of the function knn\_vote() resources used were mainly in the form of many small size array which were using the full one resource each time of the program execution. There are 3 basic arrays named as min\_distance\_list[K\_CONST], label\_list[K\_CONST], vote\_list[10]. So, we will use ARRAY\_MAP pragma to store these smaller arrays into a single array named as array3
- We were using array map horizontally to store the array into single array.
- We are using INLINE pragma to put the subprogram in the same program unit which results into improvement of resources usage.

## Pragma used in various optimization:

### 1. To remove ? in latency values:

```
#pragma HLS loop_tripcount min<value> max<value>
```

### 2. For Latency Optimization:

```
#pragma HLS unroll
```

```
#pragma HLS pipeline
```

```
#pragma HLS array_partition variable=<array_name>  
complete dim=0
```

### 3. For Area Optimization:

```
#pragma HLS inline
```

```
#pragma HLS ARRAY_MAP variable <array_name> instance  
<new_array> <mode(horizontal/vertical)>
```

## Comparision Report:

digitrec.cpp | Synthesis(latency)(DigitRec\_csynth.rpt) | Synthesis(area)(DigitRec\_csynth.rpt) | compare reports | Synthesis(base)(DigitRec\_csynth.rpt)

Vivado HLS Report Comparison

All Compared Solutions

area: xcku5p-ffvb676-2-e

base: xcku5p-ffvb676-2-e

latency: xcku5p-ffvb676-2-e

Performance Estimates

Timing (ns)

Clock		area	base	latency
ap_clk	Target	10.00	10.00	10.00
	Estimated	6.474	6.474	8.663

Latency (clock cycles)

		area	base	latency
Latency	min	36004	36004	18007
	max	9584410004	9584046004	72148007
Interval	min	36004	36004	18003
	max	9584410004	9584046004	72128002

Utilization Estimates

	area	base	latency
BRAM_18K	289	287	320
DSP48E	4	4	4
FF	1198	1480	9412
LUT	2164	3083	12776
URAM	0	0	0

Resource Usage Implementation

	area	base	latency
RTL	verilog	verilog	verilog
SLICE	-	-	-
LUT	-	-	-
FF	-	-	-
DSP	-	-	-
SRL	-	-	-
BRAM	-	-	-

Need to run vivado synthesis/implementation to populate the real data for "-"

Final Timing Implementation

## 7. Conclusion:

After applying various combination of latency and area optimization pragmas, we were able to find the significantly changes in area and latency optimization.

We also observed the trade-off between latency and area and concluded that both cannot be achieved at the same time.

To find the area optimized solution, which uses less numbers of resources, program was taking long time and to find the latency optimized solution, program was using resources heavily.