IMPLEMENTATION OF RANDOM FOREST ON FPGA

Presented by:

SIVARAM JALLU - 621252

Faculty Guide:

Mr.M.Srinivas

INTRODUCTION

- This project implements the Random Forest algorithm on FPGA hardware to overcome computational limitations and achieve faster, more efficient predictions.
- Leveraging FPGA’s parallel processing capabilities, the system aims to deliver high-speed, low-latency, and energy-efficient machine learning solutions.
- The implementation is designed to benefit real-world scenarios like IoT and edge computing, enabling scalable and responsive predictions for time-critical tasks.

PROBLEM STATEMENT

Develop a high-performance Random Forest algorithm on FPGA hardware. The goal is to achieve fast, low-latency, and energy-efficient predictions by leveraging FPGA's parallel processing capabilities. This system aims to enhance the performance of real-time applications, such as classification and regression tasks, by optimizing the execution of Random Forest models on hardware.

MOTIVATION

1. Real-Time Predictions:

FPGAs provide low-latency processing, enabling faster real-time predictions than traditional systems, crucial for applications requiring quick decision-making.

2. Scalability:

With FPGA’s parallel processing, large datasets and complex models can be efficiently handled, ensuring scalability as data volumes and complexities grow.

3. Energy Efficiency:

FPGAs are more power-efficient than CPUs and GPUs, making them ideal for deploying machine learning models in resource-constrained environments while maintaining performance.

MOTIVATION

4. Cost Reduction:

Implementing Random Forest on FPGA minimizes the need for expensive CPU/GPU clusters, reducing operational costs and making large-scale model deployment more affordable.

5. Customization and Flexibility:

FPGAs allow for hardware-level customization, enabling the optimization of the Random Forest algorithm for specific applications, datasets, and performance needs.

6. Improved User Experience:

FPGA-based execution accelerates both training and inference times, providing faster predictions, enhancing system responsiveness, and enabling real-time decision-making.

LITERATURE SURVEY

TITLE AND JOURNAL	ALGORITHM USED	PROS	CONS
FPGA Architectural Research(S. Brown)	FPGA DESIGN	• High-speed calculations with parallel processing on FPGA . • Low power consumption compared to CPU/GPU.	• Complex to design FPGA-based solutions. • FPGAs have limited logic blocks, memory, and DSP units
Decision Trees(Lior Rokach, Oded Maimon)	ID3(Iterative Dichotomies 3),CART(classification and regression trees),CHAID(Chi squared automatic detection)	• Decision trees are easy to understand and interpret, even for non-experts. • Capable of dealing with datasets that include missing values.	• Tend to create overly complex trees that fit the training data but generalize poorly. . • Requires pruning to mitigate this issue
Implementation of decision tree algorithm on FPGA Devices (Kritika Malhotra, Amit Prakash Singh)	Decision Tree Classification on FPGA DTC Algorithms: Single Module per Level (SMPL), Universal Node (UN)	• Generates easily interpretable models for classification tasks . • Works well with both categorical and continuous variables	• Decision trees are computationally expensive, especially for large datasets with numerous class labels.

CHALLENGES

1.Efficient Use of FPGA Resources:

- FPGAs have limited resources like logic elements, memory blocks (BRAM), and DSP units.
- **Challenge**: Efficiently mapping the Random Forest model onto the FPGA's resources without compromising performance or exceeding resource limitations.

2.Handling Large Data in Real-Time:

- Random Forest models require processing and storing large datasets for training and inference.
- **Challenge**: Ensuring that the FPGA can handle large volumes of data in real-time while maintaining smooth data flow and minimizing delays during prediction.

CHALLENGES

3.Complexity of Decision Tree Models:

- As the number of trees and their depth increases, the Random Forest model becomes more complex and resource-intensive.
- **Challenge:** Optimizing the model to fit within the FPGA's constraints while maintaining accuracy and performance across multiple decision trees.

4.Toolchain and Implementation Complexity:

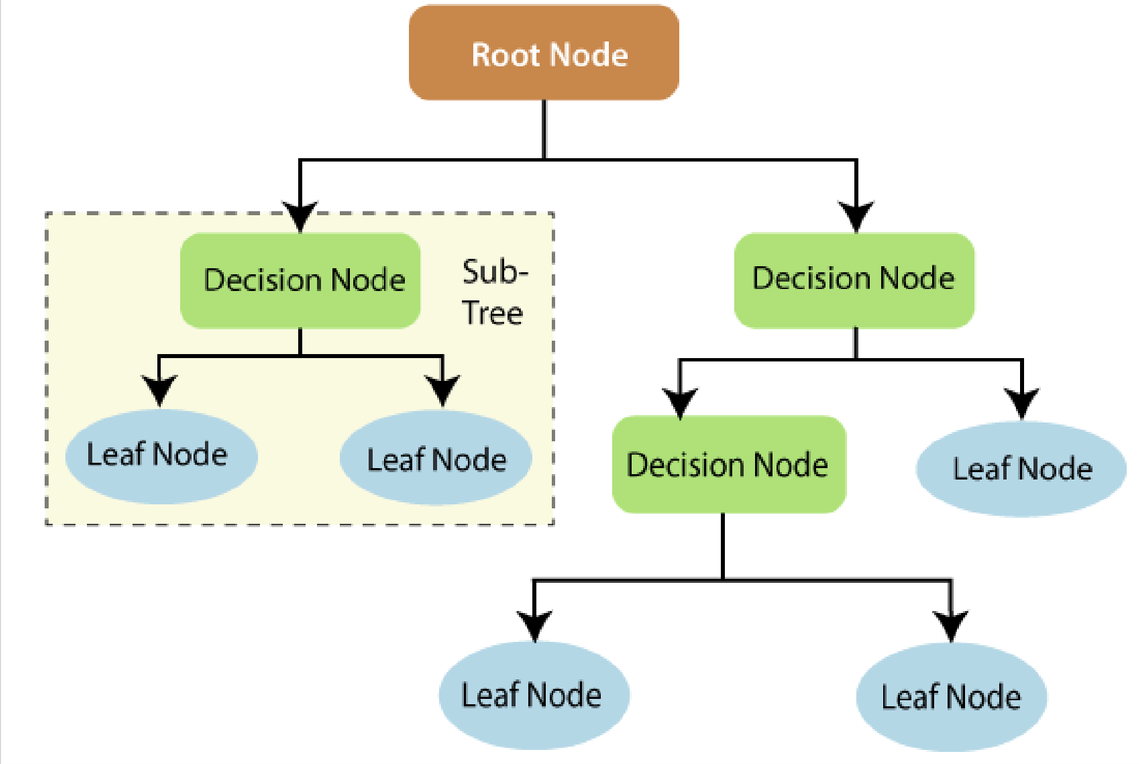
- Translating the Random Forest model into FPGA-compatible code (e.g., Verilog/VHDL) is complex.
- **Challenge**: Using FPGA design tools like Xilinx Vivado to accurately implement and optimize the Random Forest algorithm on hardware, ensuring proper synchronization and resource utilization.

DECISION TREES

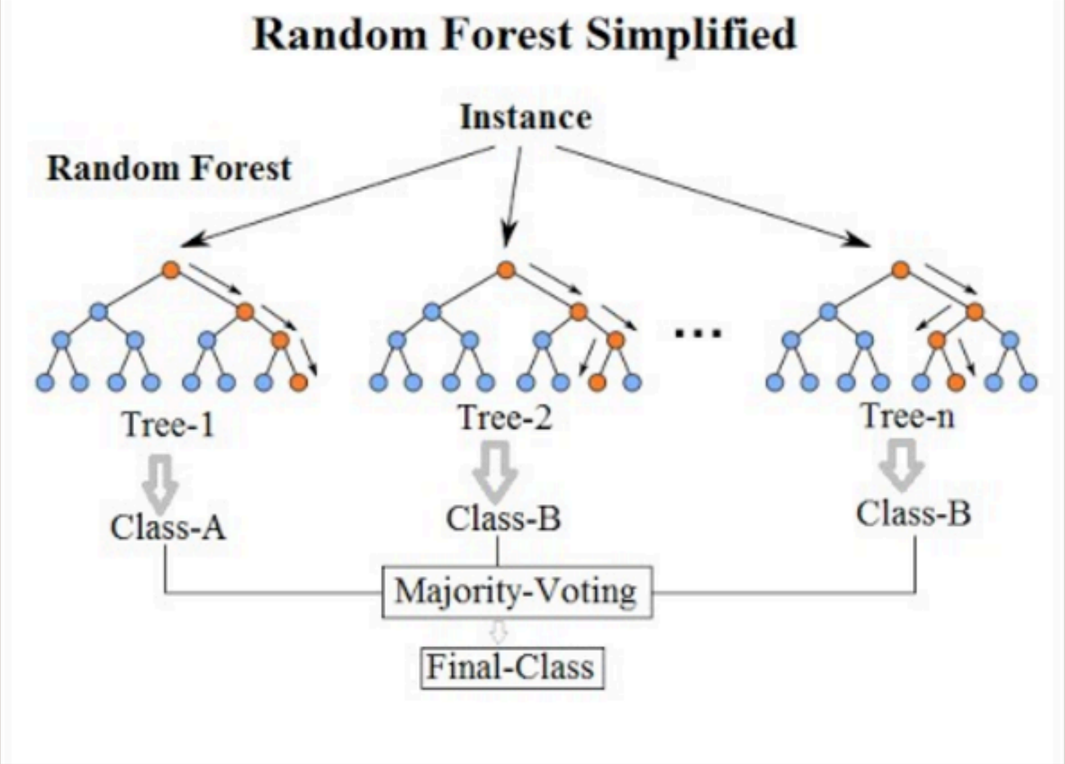
- A decision tree is a popular machine learning algorithm used for classification and regression tasks. It works by recursively splitting the data into subsets based on feature values, with the goal of making predictions as simple as possible.

- The structure of a decision tree resembles a flowchart, where each internal node represents a decision based on the value of a feature, each branch represents the outcome of that decision, and each leaf node represents the final output (a class label or continuous value).

DECISION TREE



RANDOM FOREST



METHODOLOGY

Design of Decision Tree Algorithm:

The core of the project involves designing the Decision Tree algorithm that will be implemented on the FPGA. The algorithm is adapted to work efficiently with FPGA hardware, ensuring it can handle the decision-making process of splitting nodes and creating branches based on feature values.

Verilog Code Implementation:

The Decision Tree logic is translated into Verilog code, which is a hardware description language (HDL) used to implement the algorithm on FPGA. This step involves developing the hardware architecture to simulate the decision tree's logic, including how nodes are split and how branches are traversed based on the input data.

METHODOLOGY

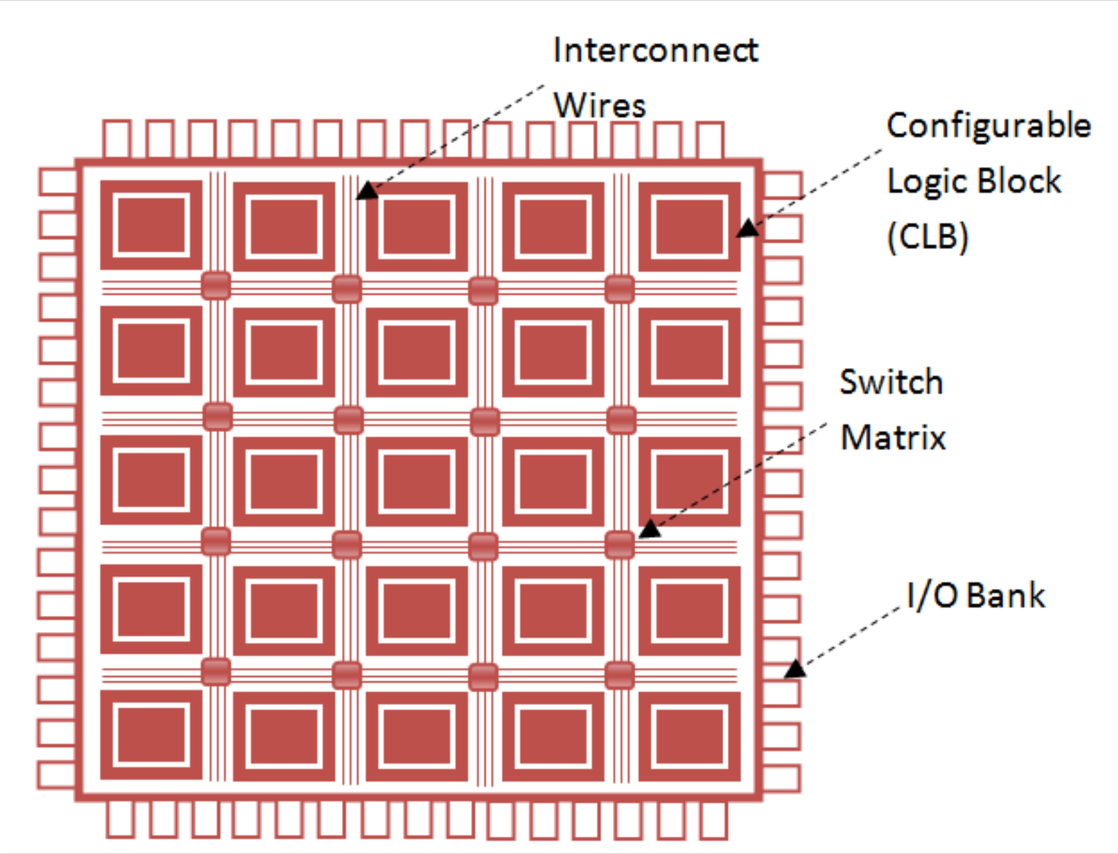
FPGA Synthesis and Optimization:

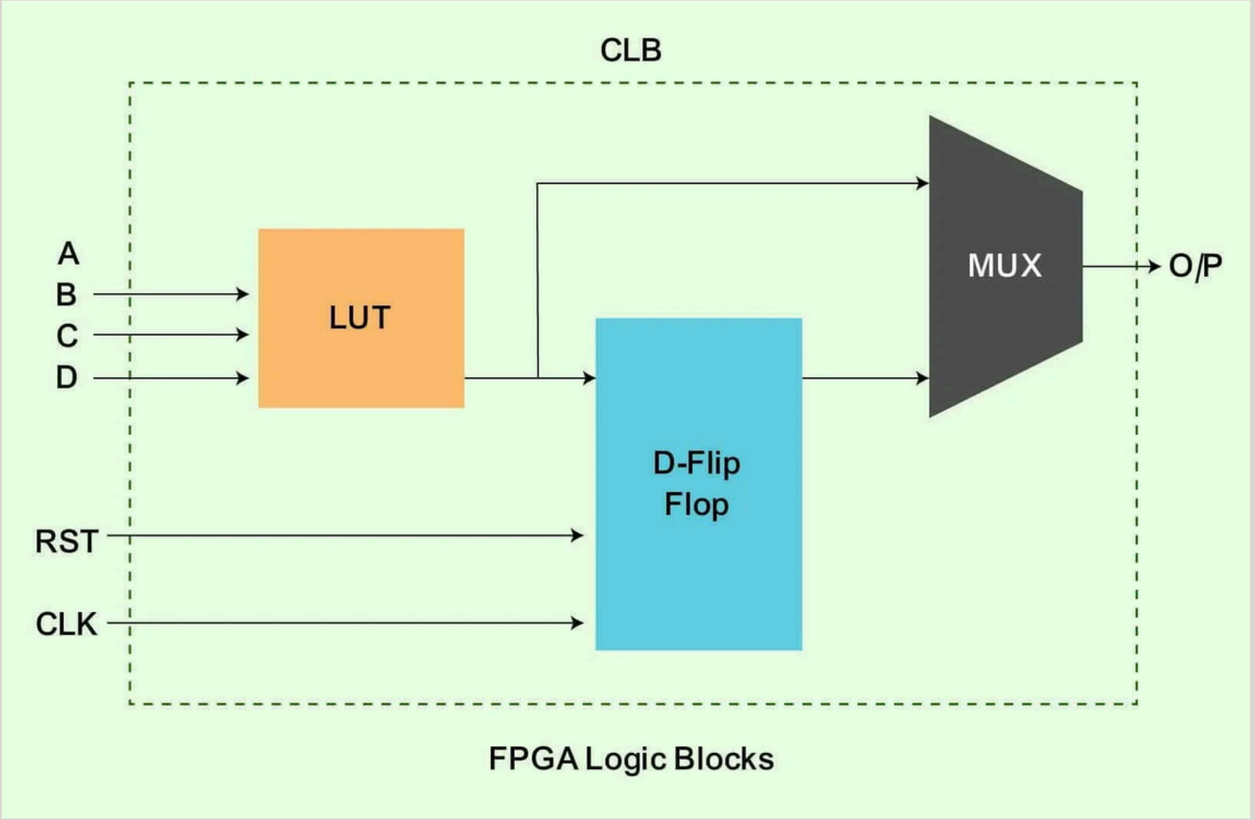
The Verilog code is synthesized using FPGA toolchains (e.g., Xilinx Vivado) to convert the high-level code into FPGA-compatible logic. This step includes optimizing the design for resource utilization (logic blocks, memory, DSP units) to ensure the model fits within the FPGA's capacity while maintaining performance.

FPGA Optimization and Performance Tuning:

The FPGA design is optimized further to minimize power consumption, maximize processing speed, and reduce latency. This involves fine-tuning the Verilog code and FPGA configurations to ensure real-time performance and efficient hardware utilization.

FPGA ARCHITECTURE





FPGA ARCHITECTURE

1.Configurable Logic Blocks (CLBs)

- These are the primary building blocks of an FPGA. Each CLB contains a combination of Lookup Tables (LUTs), Flip-Flops (FFs), and Multiplexers (MUXes) to perform both combinational and sequential logic operations.

Components of a CLB:

Lookup Tables (LUTs):

- The LUT is a small memory element that stores the truth table for a specific logic function. It can implement any boolean function.
- Common sizes for LUTs are 4-input or 6-input, meaning they can perform logic functions on up to 4 or 6 input signals.

Flip-Flops:

- Flip-flops (FFs) store the output of the logic function implemented by the LUT.
- They are used for sequential logic, holding state information between clock cycles

FPGA ARCHITECTURE

Carry Chains:

- These are used to implement arithmetic operations like addition and subtraction efficiently by carrying bits from one CLB to the next.

Multiplexers:

- Multiplexers are used within the CLB to select between various LUT outputs or other internal signals.

2. Input/Output Blocks (I/O)

- I/O blocks manage communication between the FPGA and external devices (e.g., sensors, memory, processors, displays). supporting different voltage standards and bidirectional signaling.

3. Block RAM (BRAM)

- FPGAs include dedicated memory blocks called Block RAM (BRAM), which provide on-chip memory storage for data and instructions.

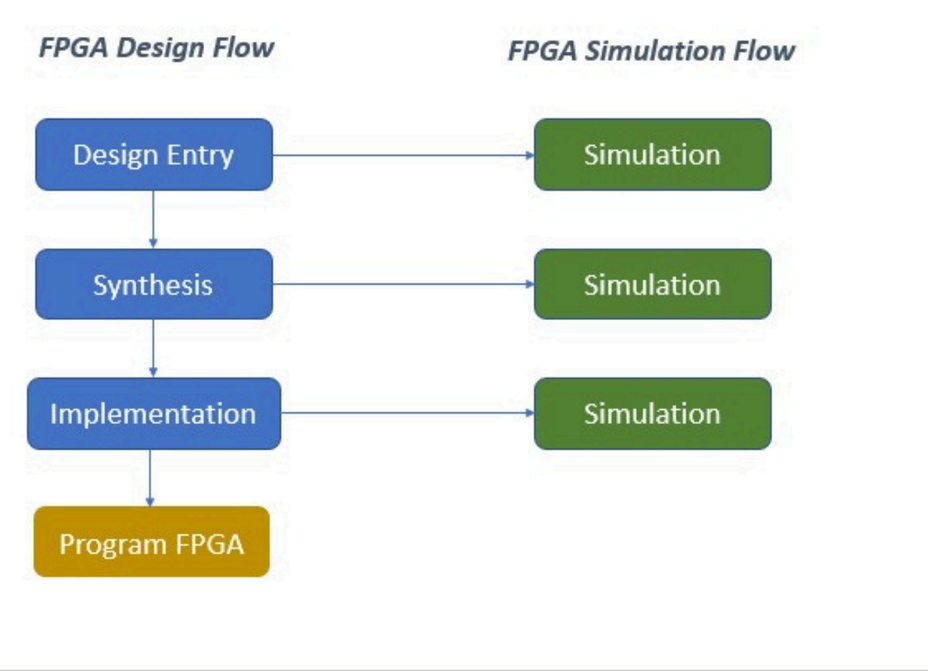
4.DSP Slice:

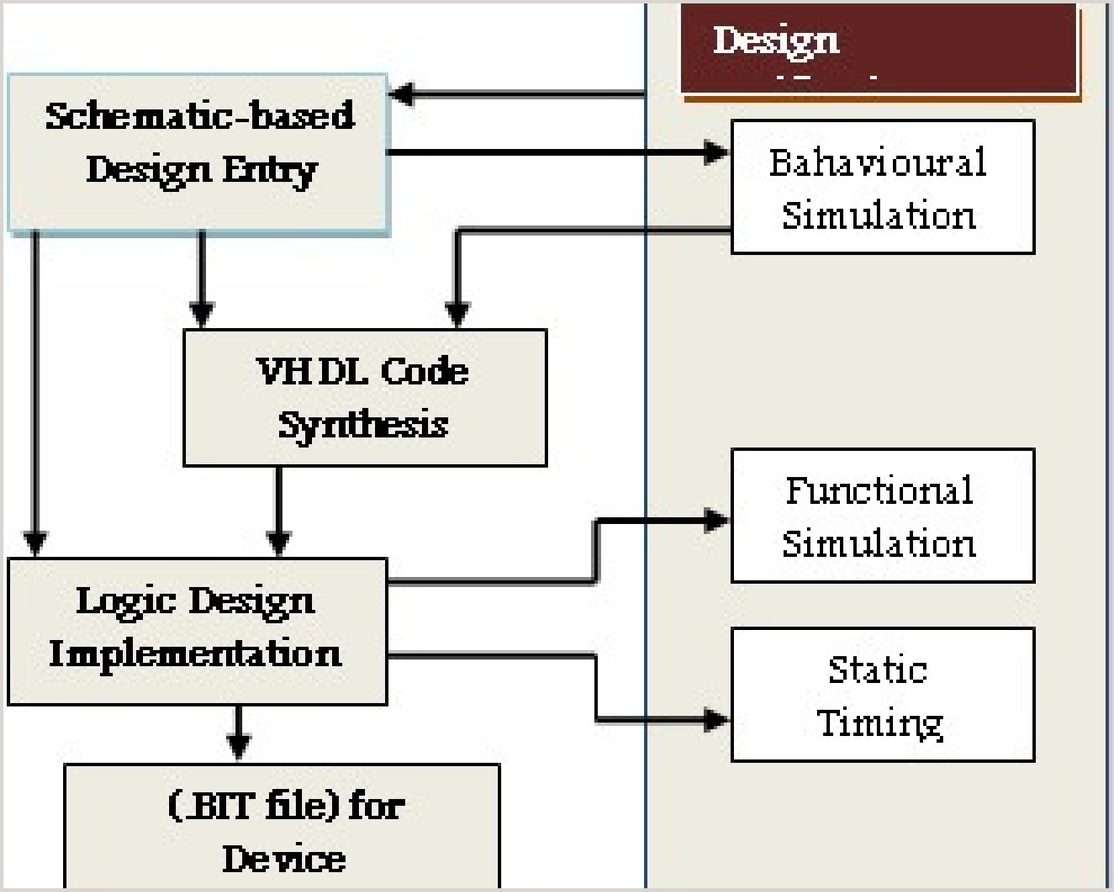
- DSP slices in an FPGA are specialized hardware blocks designed to perform high-speed arithmetic operations, particularly multiplication, addition, and accumulation.

FPGA ARCHITECTURE

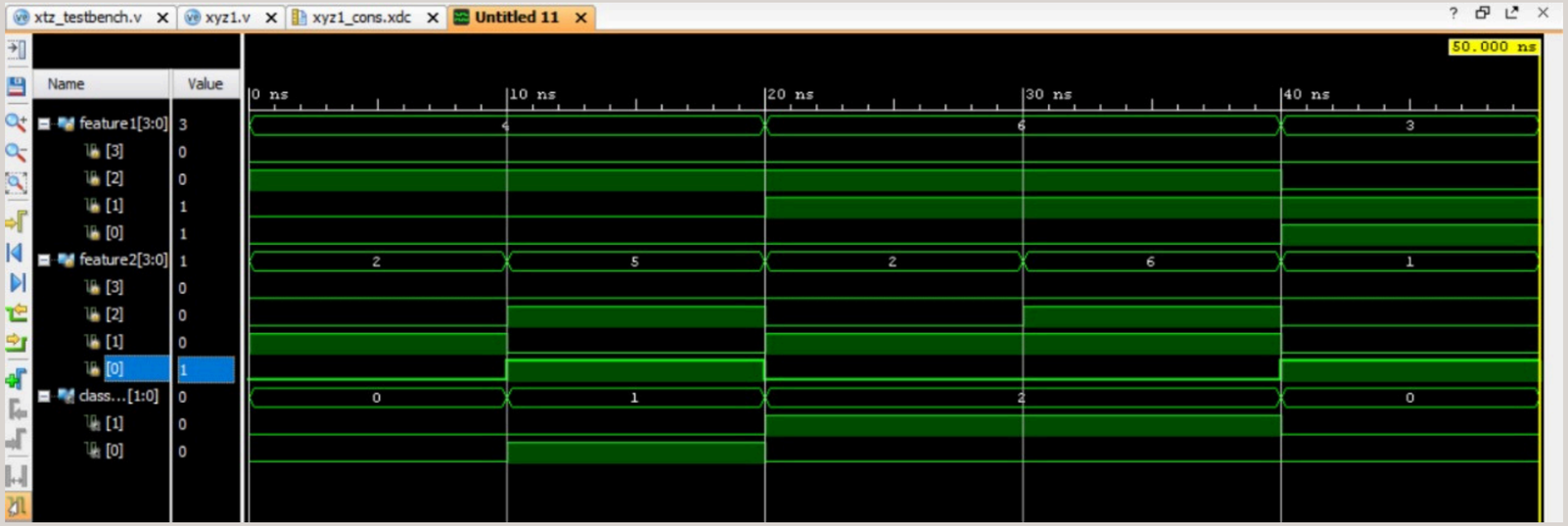
- **The Interconnect/Routing Matrix** in an FPGA is a crucial component that facilitates communication between various elements of the FPGA, such as Configurable Logic Blocks (CLBs), Block RAM (BRAM), DSP slices, and I/O blocks
- Board that we used is ARTIX 7 XC7A35TFTG256

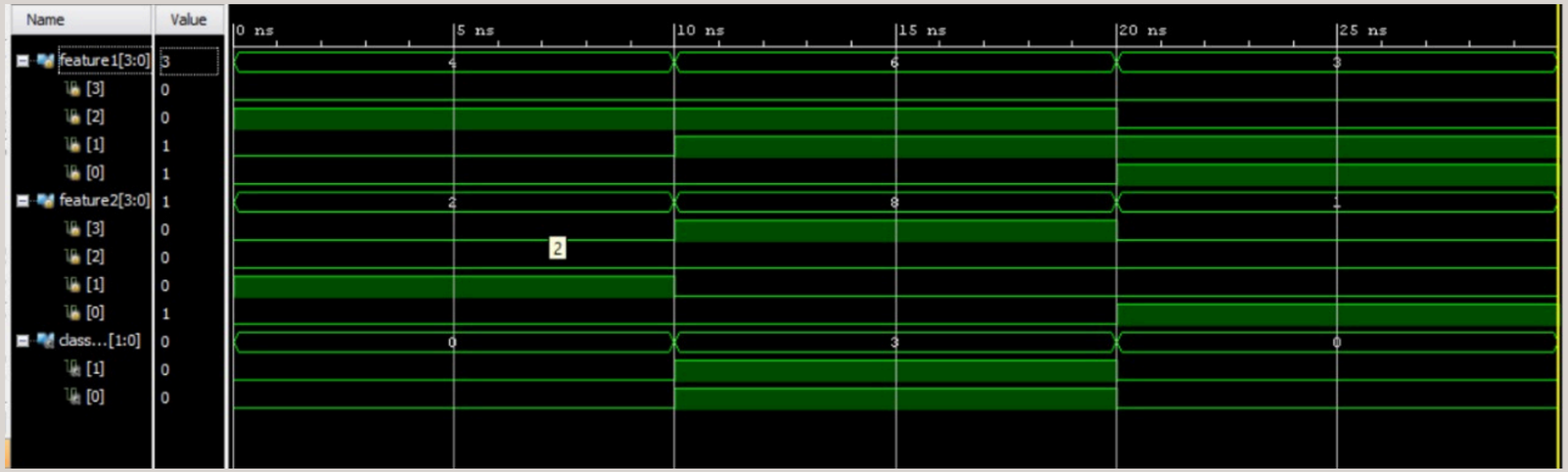
FPGA Design Flow

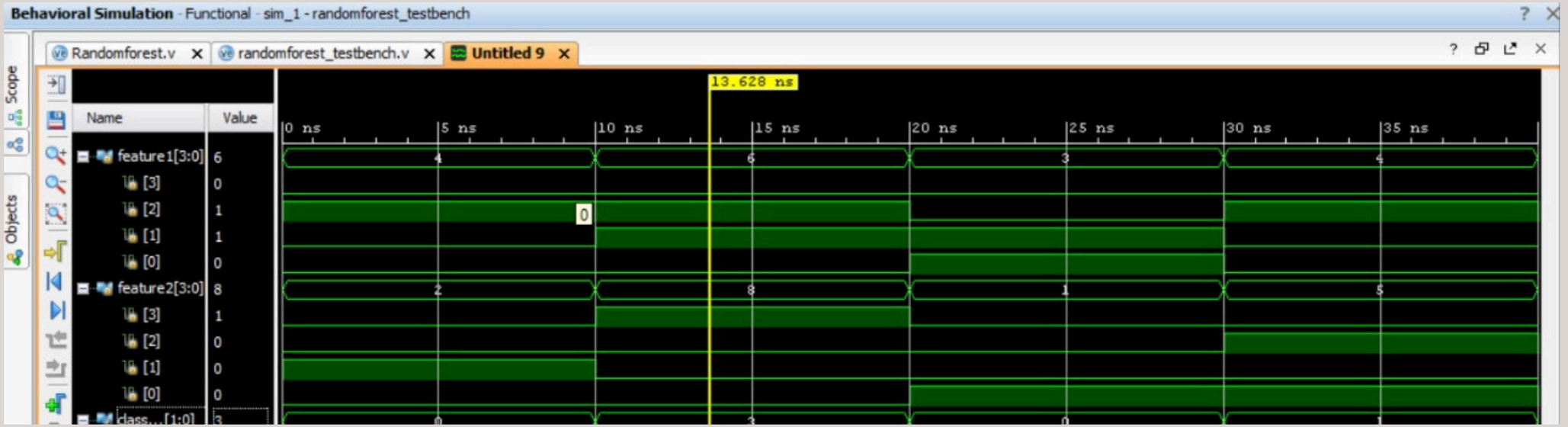


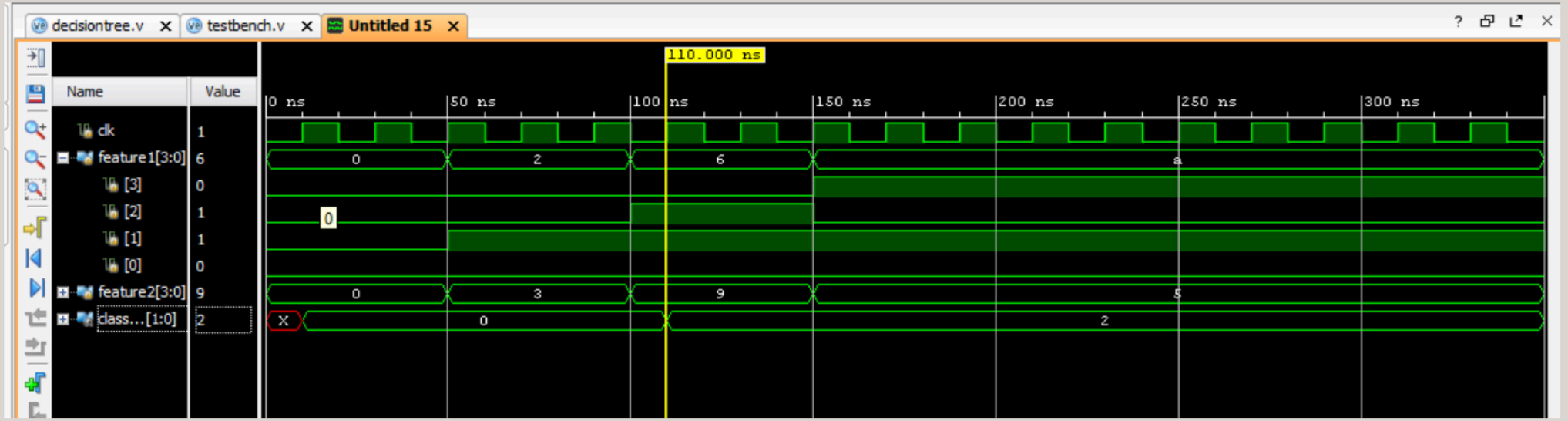


Simulation Results






Behavioral Simulation - Functional - sim_1 - randomforest_testbench
Randomforest.v
randomforest_testbench.v
Untitled 9
Scope
Objects
Name
Value
feature1[3:0]
[3]
[2]
[1]
[0]
feature2[3:0]
[3]
[2]
[1]
[0]
class...[1:0]
13.628 ns
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
35 ns


decisiontree.v
testbench.v
Untitled 15
Name
Value
clk
feature1[3:0]
[3]
[2]
[1]
[0]
feature2[3:0]
class...[1:0]
110.000 ns
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

COMPARISIONS

METRIC	SOFTWARE(PYTHON)	FPGA(ARTIX7)
TIME TAKEN TO PROCESS 10,000 SAMPLES	0.001728	0.0006
LATENCY	10	3
SPEEDUP	1	2.9(TIMES FASTER)

Sample Codes

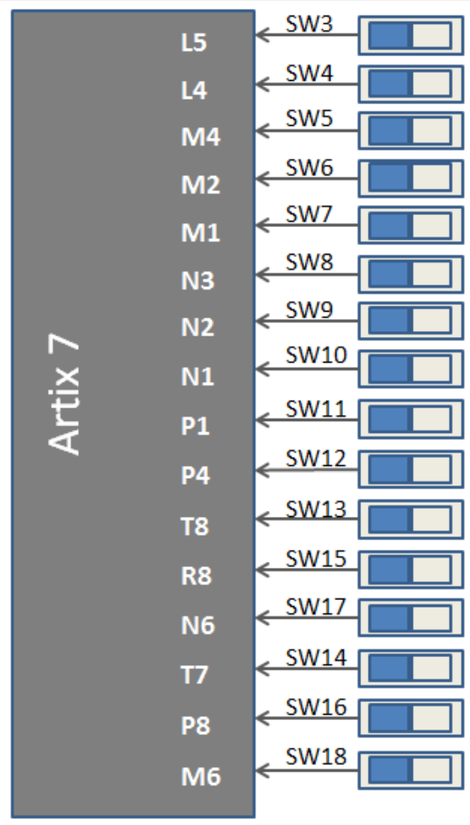
```python
import numpy as np
from sklearn.tree import DecisionTreeClassifier
import time

# Generate synthetic dataset
np.random.seed(42)  # For reproducibility
X_train = np.random.rand(1000, 3)  # 1000 samples, 3 features
y_train = np.random.randint(0, 4, size=1000)  # 4 classes (0, 1, 2, 3)

# Train a Decision Tree
clf = DecisionTreeClassifier(max_depth=5, random_state=42)
clf.fit(X_train, y_train)

# Generate test data
X_test = np.random.rand(10000, 3)  # 10,000 test samples

# Measure inference time
start_time = time.time()
predictions = clf.predict(X_test)
end_time = time.time()

# Print the results
print(f"Time taken to predict 10,000 samples: {end_time - start_time:.6f} seconds")
```

```
Time taken to predict 10,000 samples: 0.001728 seconds
```

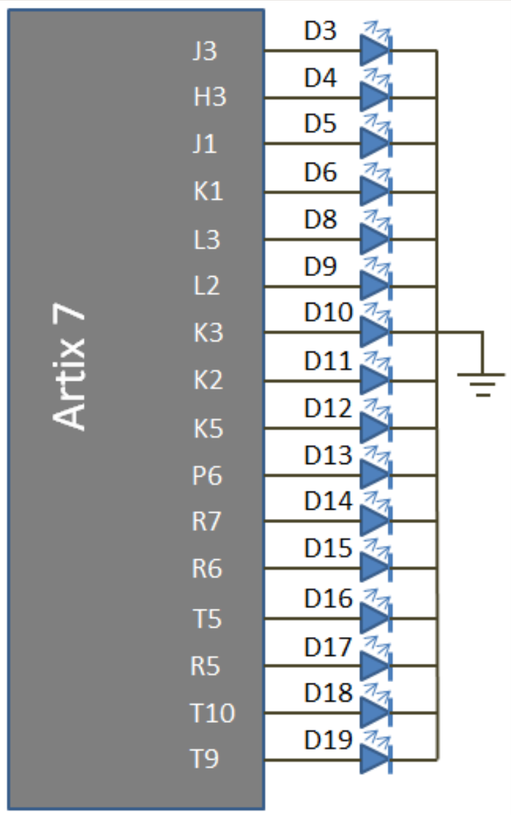
```verilog
module decisiontree(
    input clk,                  // Clock signal
    input [3:0] feature1,       // Feature 1: 4-bit input (0-15)
    input [3:0] feature2,       // Feature 2: 4-bit input (0-15)
    output reg [1:0] class_out // Predicted Class: 2-bit output (0-3)
);

always @(posedge clk) begin
    // Simple decision tree logic based on feature1 and feature2 values
    if (feature1 <= 5) begin
        if (feature2 <= 3) begin
            class_out <= 2'b00; // Class 0
        end else begin
            class_out <= 2'b01; // Class 1
        end
    end else begin
        class_out <= 2'b10; // Class 2
    end
end

endmodule
```

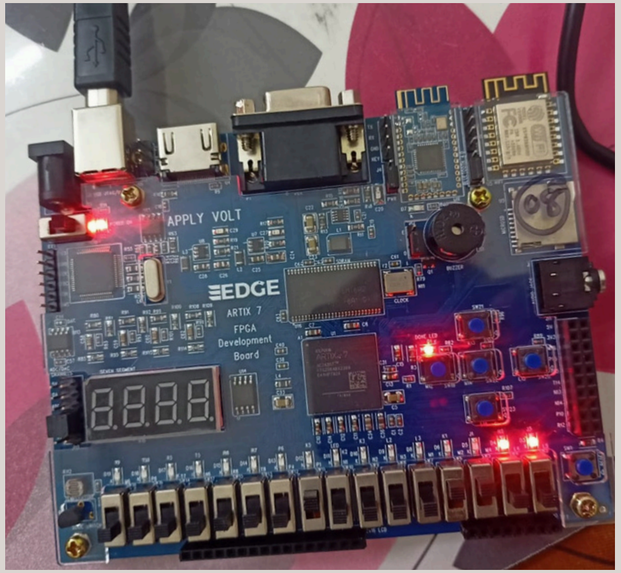
Results

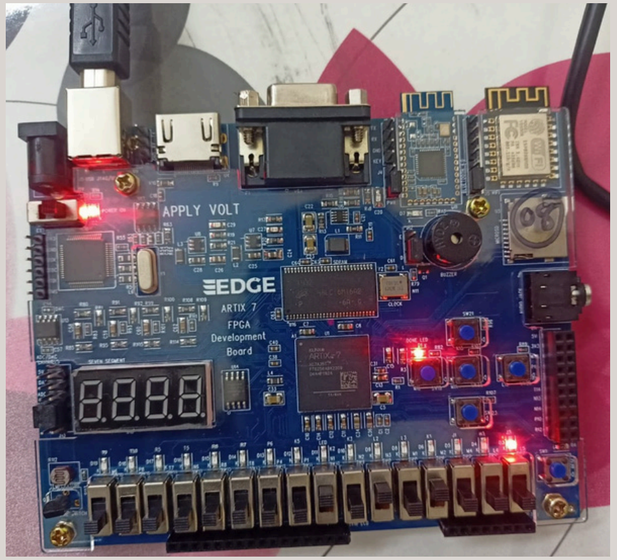
Slide Switches



LEDs







REFERENCES

1. STEPHEN BROWN, “FPGA Architectural Research: A Survey,” Proc. IEEE Int’l Conf. Design, Automation and Test in Europe, Apr. 2017, pp. 189-194.

2. Kritika Malhotra, Amit Prakash Singh, “Implementation of Decision Tree Algorithms on FPGA Devices, " IEEE Design & Test of Computers, Vol. 10, No. 1, March 2021, pp. 131 - 138

3. Lior Rokach and Oded Maimon, “Decision Trees”, Research Gate.net, Vol. 12, January 2005, pp. 166 - 189.

4. S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tuto- rial," IEEE Design & Test of Computers, Vol. 13, September 2012, pp. 42-57.