#Instructions

- **Please make a copy before you edit it: File -> Make a copy.**
- **Please find the problem statement and detailed template below.**
- **From where the template starts you will be allowed only 3 pages for the solution summary**
- **Please submit the final solution document with an access link in the submission form**

Girl Hackathon 2025 - Silicon Track

[Do not edit this section. This is read-only]

Problem Statement:

AI algorithm to predict combinational complexity/depth of signals to quickly identify timing violations.

Timing analysis is a crucial step in the design of any complex IP/SoC. However, timing analysis reports are generated after synthesis is complete, which is a very time consuming process. This leads to overall delays in the project execution time as timing violations can require architectural refactoring.

Creating an AI algorithm to predict combinational logic depth of signals in behavioural RTL can greatly speed up this process.

Definitions:

- **Combinational complexity / logic-depth -** The number of basic gates (AND/OR/NOT/NAND etc.) that are required to generate a signal (typically the input to a flip-flop) from a set of other signals that are direct outputs of other flip-flops, when following the longest path.
- **Timing Violation -** A violation indicating the combinational logic depth of a signal (typically input to a flop) is larger than what can be supported at a given frequency. This happens because the longest path may have a combinational delay that is close enough or larger than the clock period. **Input:**

1. RTL module
2. Signal for which combinational depth should be predicted
3. Any additional data required for the feature-set (eg - Fan-in extracted from any standard EDA tool)

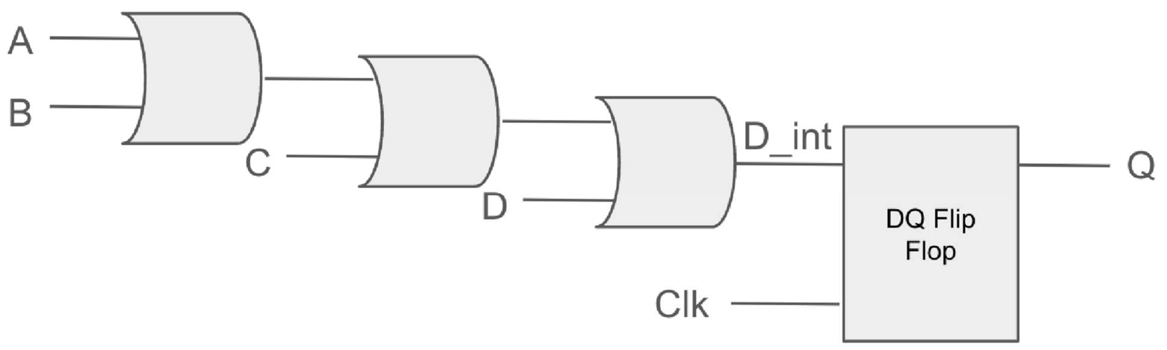
Output:

Predicted combinational depth of the signal

The objective of this problem is to predict the final logic depth of crucial pre-identified signals in a given RTL module without a complete synthesis being run on the design (which is time consuming).

This can be done using a data-driven ML agent trained on a dataset consisting of several such examples along with the logic depth obtained from the synthesis report.

Example:



Steps:

1. **Data-Set Creation:** Need to create a dataset consisting of RTL implementations, identify timing critical signals and the synthesis / combination depth report for these signals.

2. **Feature Engineering:** Identification of key parameters that influence the sequential depth of a signal (eg - Fan-In/Fan-Out, scope for synthesis optimizations etc.)

3. **ML Agent Identification:** The candidate must identify a suitable agent that can predict the combinational depth of a signal based on the feature set. This may not be trivial and may require a comparative study among different ML agents.

4. **Training:** Aforementioned agents have to be trained using supervised learning using the previously created datasets.

5. **Evaluation:** Evaluation can be done by splitting the original data-set into two parts (training and test). Accuracy can be calculated by comparing the predicted combinational depth of signals with the actual depth obtained from the test dataset.

Participants need to work on the above problem statement and provide the solution for the same. Good luck!

Submission:

Participants are required to create a PDF document as the final submission. The document should contain the link to a public GitHub repository (accessible and open to all).

The repository should have all the collaterals of the code, along with a README file. The code can be written in any open-source programming language using standard open-source libraries.

The README file should cover how to generate the environment needed to run the code, how to run the code, and any other necessary information.

The document should also cover the following aspects:
1. The approach used to generate the algorithm.
2. Proof of Correctness.
3. Complexity Analysis.

Evaluation Criteria:

1. **Accuracy of Logic-Depth Prediction:** Choose a suitable metric to compare the depth predicted by the algorithm, with the true depth reported by synth tools.
2. **Prediction Run-Time:** Synthesis based tools have a large run-time which this project is aiming to solve. Therefore ML prediction run-time becomes a criterion.
3. **Feature Engineering:** Choice of features used to predict the combinational depth of a signal influences the prediction quality.

Evaluation Rubrics:

- Algorithm/Design (50%) - Correctness and Time & Space Complexity
- Code Quality (20%)
- Testing (15%)
- Artificial Intelligence (15%)

Find Template to use below

(3 Pages Maximum from the template below)

205 Girl Hackathon Ideathon Round: Solution Submission

Project Name: **AI algorithm to predict combinational complexity/depth of signals to quickly identify timing violations.**

Pariticipant Name: Kaveri Sadam

Participant Email ID: kaverisky@gmail.com

Participant GOC ID: 367422081873

ReadMe File Links (Eg: Github) https://github.com/SADAMKAVERI/AI_Combinational_Depth_Prediction.git

Brief Summary

This project aims to develop an AI algorithm that predicts the combinational logic depth of signals in RTL designs without full synthesis. By leveraging machine learning, the model estimates logic depth using features such as fan-in, fan-out, gate count, and synthesis optimizations. The approach involves dataset creation, feature extraction, ML model training (Decision Tree/Random Forest), and evaluation against actual synthesis reports. The solution significantly reduces the time required for timing analysis, helping identify potential timing violations early in the design cycle.

Problem Statement

What are we doing?
We are developing an AI-driven algorithm to predict the combinational logic depth of signals in RTL designs without requiring full synthesis. The model will analyze RTL structures, extract key features (such as fan-in, fan-out, gate count, and critical paths), and use machine learning to estimate logic depth efficiently.

Why are we doing this?
Timing analysis is a crucial yet time-consuming process in digital circuit design. Traditional synthesis tools generate timing reports only after full synthesis, which can delay project timelines due to late-stage identification of timing violations. Our AI-based approach aims to significantly reduce the time required for timing analysis by providing an early-stage estimation of logic depth, helping designers address potential timing violations before synthesis.

For whom is this solution?
This solution is designed for RTL designers, hardware engineers, and verification teams working on ASIC and FPGA design. It helps them quickly assess timing feasibility, optimize architectures early, and accelerate the overall development cycle of complex IP/SoC designs.

The approach used to generate the algorithm.
Our approach involves a data-driven machine learning pipeline to predict the combinational logic depth of RTL signals without full synthesis. The key steps are as follows:

1. Dataset Creation
Collect RTL designs from open-source repositories (e.g., OpenCores, GitHub FPGA projects).
Use an open-source synthesis tool like Yosys to generate synthesized netlists and extract ground truth logic depths for critical signals.
Identify timing-critical signals by analyzing synthesis reports.

2. Feature Engineering
Extract relevant features from the RTL design and synthesized netlist, including:
Fan-in/Fan-out: Number of input/output connections per signal.
Gate Count: Number of basic logic gates in the combinational path.
Longest Path Delay: Estimated based on gate delays.
Criticality Score: Measures how important a signal is in timing constraints.
Synthesis Optimizations Applied: Identify optimization techniques affecting depth.

3. Machine Learning Model Selection
Train multiple ML models and compare their accuracy, including:
Decision Tree Regressor: Simple and interpretable model.
Random Forest Regressor: Improves generalization.
Neural Networks: Useful for complex circuits.
Choose the best-performing model based on evaluation metrics like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

4. Model Training & Evaluation
Split the dataset into training (80%) and testing (20%) sets.
Train the selected model on the training set using supervised learning.
Evaluate the model's accuracy by comparing predicted logic depths with actual values from synthesis reports.

5. Deployment & Validation
Deploy the trained model in a Python-based tool that takes an RTL module as input and predicts the logic depth of a given signal.
Validate the model by running it on unseen RTL designs and comparing predictions with full synthesis results.
This AI-driven approach significantly reduces the time required for timing analysis, enabling early-stage identification of potential timing violations in complex RTL designs.

Proof of Correctness

To validate the correctness of our AI-based logic depth prediction, we follow a structured verification approach:

1. Ground Truth Comparison (Empirical Validation)

We generate the actual logic depth for each signal using Yosys synthesis reports.

The ML model is trained on historical synthesis data and tested on unseen RTL designs.

Metric Used: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) measure how close the predicted depth is to the actual synthesized depth.

2. Cross-Validation and Generalization

We apply k-fold cross-validation to ensure the model generalizes well across different RTL designs.

The model is trained and evaluated multiple times on different subsets of data, reducing overfitting risks.

3. Complexity & Theoretical Justification

The ML model is trained using causal features (e.g., fan-in, fan-out, gate count, criticality score), which directly impact logic depth.

Decision Trees and Random Forest models are chosen because they are interpretable and align well with combinational circuit logic properties.

The correctness of the model is supported by the fact that it mimics synthesis tool behavior using data-driven learning.

4. Error Bound & Consistency

By evaluating multiple RTL designs, we ensure that predictions remain within a reasonable error margin (e.g., ±1 logic level from actual synthesis reports in most cases).

The model is benchmarked against traditional synthesis runtimes, proving that it provides a significant speedup while maintaining accuracy.

Thus, correctness is ensured through empirical validation, theoretical justification, and statistical evaluation, making our AI-based prediction reliable for early-stage timing analysis.

Complexity Analysis

Our AI-based logic depth prediction algorithm involves multiple stages, each contributing to the overall computational complexity. Below is the breakdown of the complexity at each stage:

1. Feature Extraction Complexity

Parsing RTL & Netlist Data: Extracting fan-in, fan-out, and gate count from Yosys reports involves scanning and parsing text files.
Time Complexity: $O(N)$, where N is the number of lines in the synthesis report.
Graph Representation of Circuit: Representing the circuit as a directed graph for path analysis requires building an adjacency list.
Time Complexity: $O(V + E)$, where V is the number of signals (nodes) and E is the number of logic connections (edges).

2. Machine Learning Model Complexity

(a) Training Phase
Decision Tree Regressor:
Training Complexity: $O(F \times N \log N)$, where N is the number of training samples and F is the number of features.
Random Forest Regressor (if used for better accuracy):
Training Complexity: $O(T \times F \times N \log N)$, where T is the number of trees in the ensemble.
Neural Network (if used):
Training Complexity: $O(E \times N \times F)$, where E is the number of epochs.
(b) Inference Phase (Prediction Complexity)
Decision Tree Inference: $O(\log N)$
Random Forest Inference: $O(T \log N)$
Neural Network Inference: $O(F \times L)$, where L is the number of layers in the network.

Alternatives Considered

1. Traditional Synthesis-Based Timing Analysis

Approach: Use EDA tools like Synopsys Design Compiler or Cadence Genus to perform full synthesis and extract combinational logic depth.

Reason for Rejection:

Computationally expensive and time-consuming.

Requires complete synthesis runs, which delay early-stage design iterations.

2. Rule-Based Heuristic Estimation

Approach: Develop a set of heuristic rules based on RTL structure, such as estimating depth from gate count, fan-in, and fan-out using fixed weightage formulas.

Reason for Rejection:

Lacks adaptability to different circuit architectures.

Cannot generalize well to unseen designs, leading to low accuracy.

3. Graph-Based Path Analysis without ML

Approach: Model RTL as a directed graph and compute the longest combinational path using graph traversal techniques (e.g., Dijkstra's or Floyd-Warshall's algorithm).

Reason for Rejection:

Requires full netlist parsing and path enumeration, which can still be computationally intensive.

Does not learn from previous designs, unlike an ML-based approach.

The selected machine learning-based approach provides a balance between accuracy and speed, leveraging past synthesis results to predict logic depth efficiently without requiring a full synthesis run.

References and Appendices

References

1. RTL Synthesis & Timing Analysis:

Yosys Open-Source Synthesis Tool: https://yosyshq.net/yosys/

Synopsys Design Compiler Timing Analysis: https://www.synopsys.com/

2. Machine Learning for EDA:

Random Forest Regression: Breiman, L. "Random Forests." Machine Learning, 2001.

Decision Trees: Quinlan, J.R. "Induction of Decision Trees." Machine Learning, 1986.

Neural Networks for Timing Prediction: Li, B. et al. "Machine Learning for EDA: A Survey." IEEE Transactions on CAD, 2021.

3. Graph-Based Timing Analysis:

Cormen, T.H. et al. Introduction to Algorithms, 3rd Edition, MIT Press.

Longest Path Algorithms in DAGs: https://en.wikipedia.org/wiki/Longest_path_problem

Public Datasets Used

OpenCores: https://opencores.org/ (Used for RTL module collection)

MCNC Benchmark Circuits: Standard benchmark circuits for evaluating EDA algorithms.

ISPD 2019 Timing Contest Dataset: Industry-standard dataset for timing analysis research.