|  |
| --- |
| **205 Girl Hackathon Ideathon Round: Solution Submission** |
| Project Name: Predictive timing analysis for RTL design. |
| Participant Name: UMA G |
| Participant Email ID: umag49219@gmail.com |
| Participant GOC ID: 105657784419 |
| ReadMe File Links (Eg:Github) |
| **Brief Summary**  Please summarize your problem statement and solution in a short paragraph.  Timing analysis in complex IP/SoC design is a time-consuming process, as it requires complete synthesis to identify timing violations. This project aims to develop an AI-driven algorithm to predict the combinational logic depth of critical signals in an RTL module without full synthesis. By leveraging machine learning, the solution will analyze key features such as fan-in, fan-out, and gate count to estimate logic depth quickly, helping identify potential timing violations early in the design phase. |
| **Problem Statement**  What are you doing, why, and for whom?  I developed an AI-based algorithm to predict the combinational logic depth of critical signals in an RTL module without requiring full synthesis. Traditional timing analysis relies on synthesis reports, which are time-consuming and can delay project timelines. By using machine learning, our solution aims to quickly estimate logic depth based on key RTL features such as fan-in, fan-out, and gate count. This will help hardware designers and verification engineers identify potential timing violations early in the design cycle, reducing the need for repeated synthesis runs and optimizing the overall development process. |
| **The approach used to generate the algorithm.**   **Dataset Processing**   * The dataset consists of RTL signals with attributes such as fan-in, fan-out, gate count, path delay, and combinational depth. * The dataset is loaded and analyzed to check for missing values and overall data structure.    **Feature Engineering**   * Key parameters influencing combinational depth are extracted, such as fan-in, fan-out, and gate count. * These features are used to train the machine learning model.    **Machine Learning Model Selection**   * The dataset is split into training and testing sets. * A machine learning model is applied to predict combinational depth based on the input features. * Different models may be tested to determine the best-performing one.    **Training and Evaluation**   * The model is trained on the dataset using supervised learning techniques. * Performance metrics such as accuracy or mean absolute error (MAE) are used to evaluate the model. * The predicted logic depth is compared with actual values from the dataset.    **Predictions and Validation**   * The trained model is used to predict combinational depth for new RTL signals. * The results are validated against known values to ensure accuracy. |

|  |
| --- |
| **Proof of Correctness**   1. **Ground Truth Comparison**    * The predicted combinational depth from the machine learning model is compared against the actual synthesis report values obtained from EDA tools.    * A low mean absolute error (MAE) or root mean square error (RMSE) indicates a high degree of accuracy. 2. **Cross-Validation**    * The dataset is split into training and testing sets to ensure the model generalizes well.    * K-fold cross-validation is used to prevent over fitting and assess model reliability. 3. **Feature Importance Analysis**    * The influence of fan-in, fan-out, and gate count on prediction accuracy is analyzed to ensure the model is learning meaningful relationships.    * Feature selection techniques such as SHAP values or correlation matrices confirm that relevant parameters drive the predictions. 4. **Performance Metrics**    * The model is evaluated using standard metrics like R² score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).    * Higher R² values and lower MAE/RMSE indicate stronger predictive capability. 5. **Real-World Testing**    * The model is tested on unseen RTL designs and compared with actual synthesis reports to validate correctness.    * If discrepancies arise, additional training data or hyper parameter tuning is performed. |
| **Complexity Analysis**  In my project, I extract features from the RTL design by processing each signal to collect parameters such as fan-in, fan-out, gate count, and path delay. Since each signal is processed once, this feature extraction step has a time complexity of O(n), where n is the number of signals. For the training phase, the complexity depends on the chosen machine learning model. For instance, if a decision tree or random forest is used, training typically operates at an average-case time complexity of O(n log n), with n representing the number of training samples.  Alternatively, if a neural network is employed, the training complexity can be estimated as O(e \* p \* n), where e is the number of epochs, p is the number of parameters, and n is the number of samples. Once the model is trained, each prediction (i.e., inference) runs in constant time, O(1), making it significantly faster than performing full synthesis for each signal. The space complexity is largely linear, depending on the size of the training dataset and the number of model parameters. This analysis shows that our approach provides a more efficient alternative to traditional synthesis-based timing analysis, significantly reducing the time and computational resources required to predict combinational logic depth. |
| **Alternatives Considered**  Include alternate design ideas here which you are leaning away from.  1. Static Timing Analysis (STA)  - Pros: Standard industry tool, well-validated.  - Cons: Requires full synthesis, making it slow.  - Decision: Used STA results as ground truth for AI model validation.  2. Rule-Based Estimation  - Pros: Simple to implement, no ML required.  - Cons: Does not generalize well to complex designs.  - Decision: Rejected due to poor adaptability to new RTL designs.  3. Other ML Models (Neural Networks, XGBoost)  - Pros: Can learn complex patterns.  - Cons: Higher computation cost compared to Random Forest.  - Decision: Random Forest was chosen for its balance of accuracy, interpretability, and speed. |
| **References and Appendices**  Any supporting references, mocks, diagrams or demos that help portray your solution. Any public datasets you use to predict or solve your problem.  Datasets Used  - Custom dataset of 400+ RTL designs with combinational depth labels.  - Open-source Verilog benchmarks from IWLS 2005, OpenCores, and industrial partners.  Supporting Materials  - Schematic diagrams showing different RTL structures.  - Performance comparison charts between AI-based prediction and synthesis tools.  - Training logs and hyperparameter tuning results. |