Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis

Jieru Zhao[1], Tingyuan Liang[1], Sharad Sinha[2], Wei Zhang[1]

[1]Hong Kong University of Science and Technology (HKUST)

[2]Indian Institute of Technology Goa (IIT Goa)

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

IIT GOA
INDIAN INSTITUTE OF TECHNOLOGY GOA

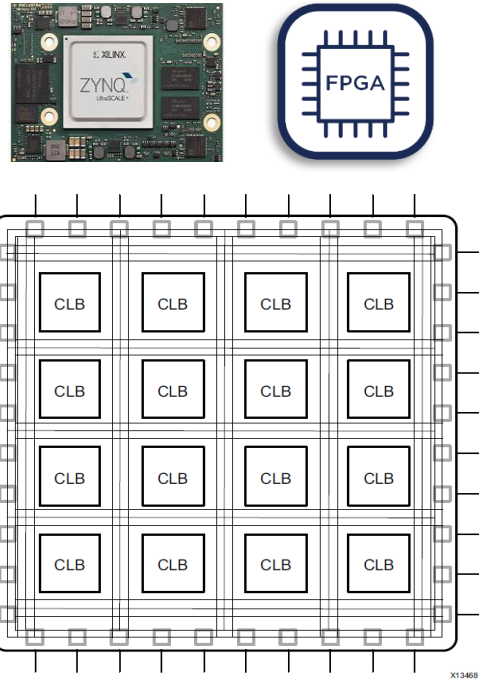
Outline

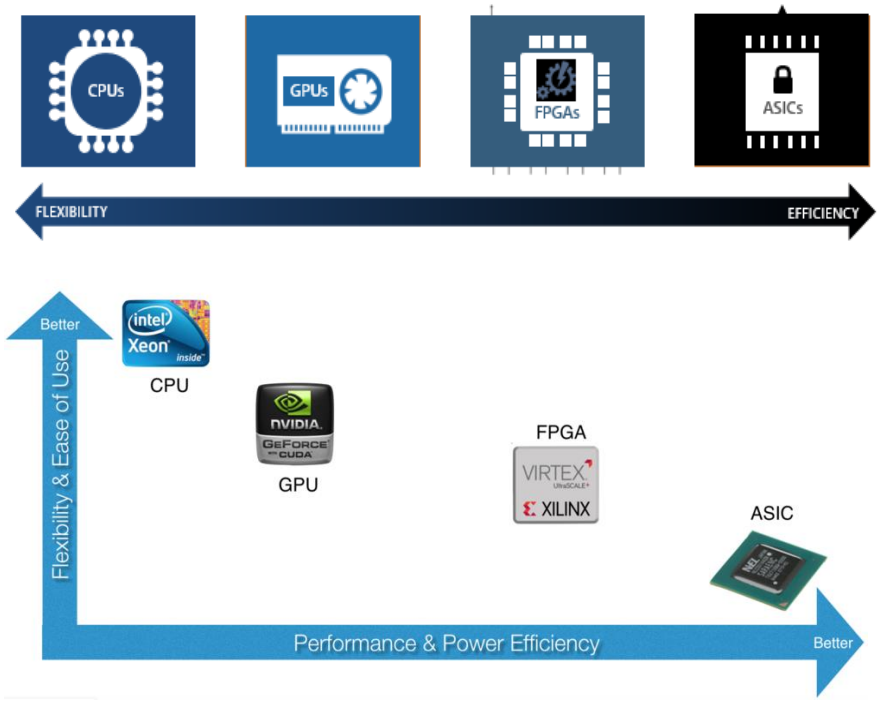
- Background
- Machine-Learning Based Methodology
- Experimental Results
- Conclusion

Outline

- Background
- Machine-Learning Based Methodology
- Experimental Results
- Conclusion

Background: FPGA





Basic FPGA Architecture:

- LUT
- Flip-flop
- DSP Block
- Storage elements: BRAM

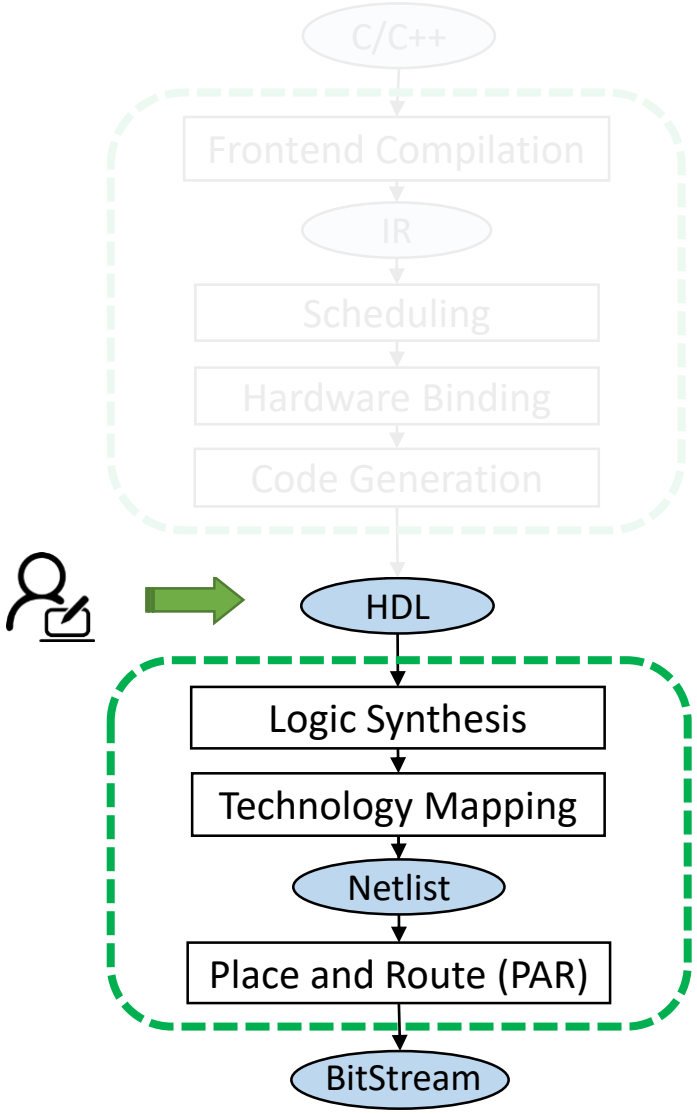
As an accelerator, FPGA has:

- High performance
- Low power
- Programmability & Flexibility
- Short time-to-market

A wide range of applications:

- Data center
- Image processing
- Communications
- …

FPGA CAD design flow

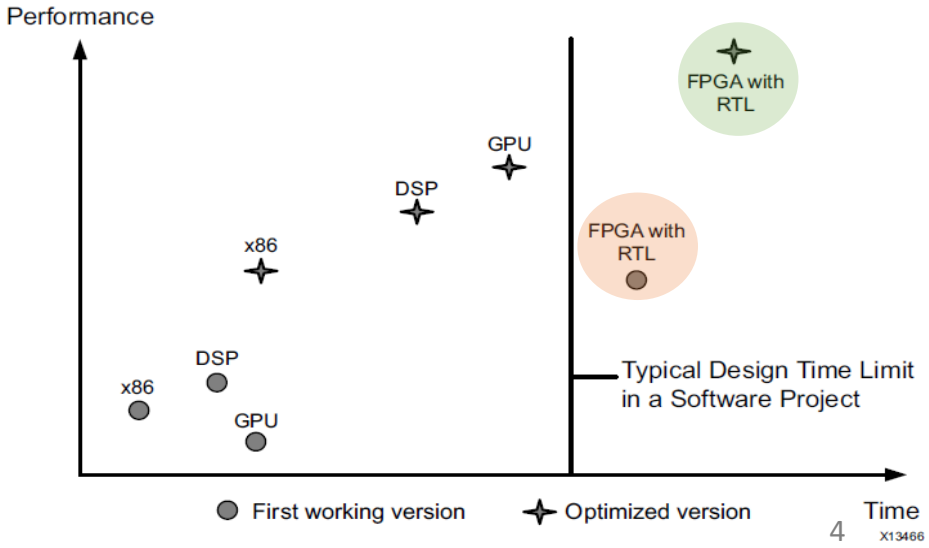


RTL Implementation Flow

- ☺ High performance.
- ☹ High development cost.

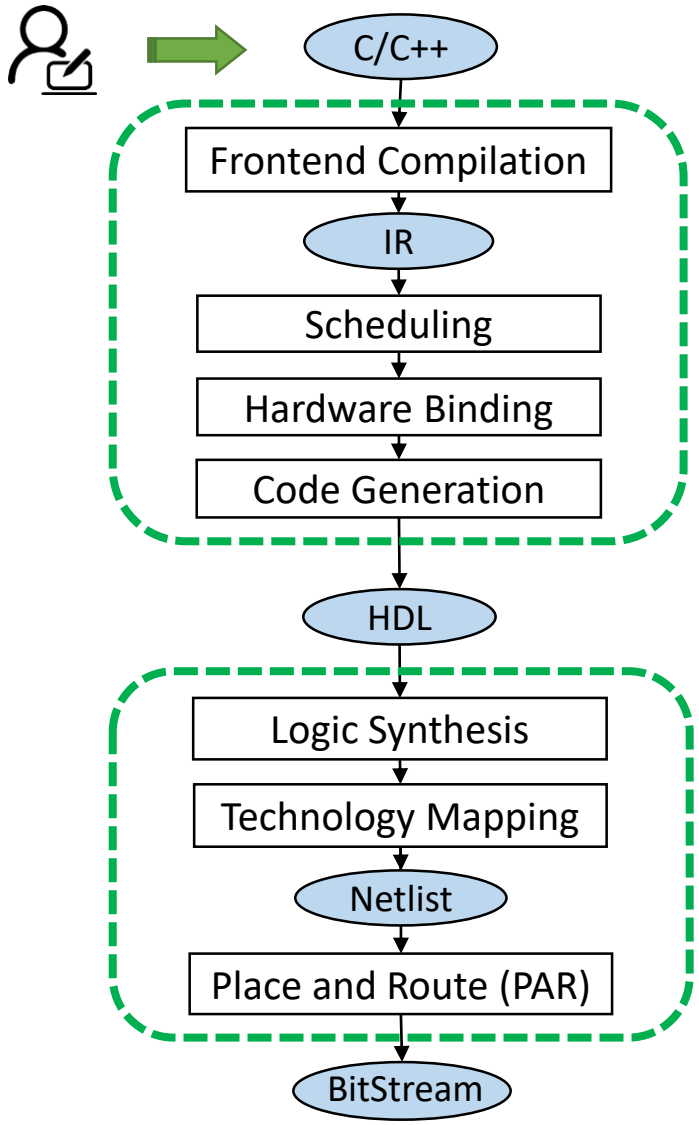
  Error-prone and time-consuming.

  Hard to debug.



[Source from Xilinx]

FPGA CAD design flow

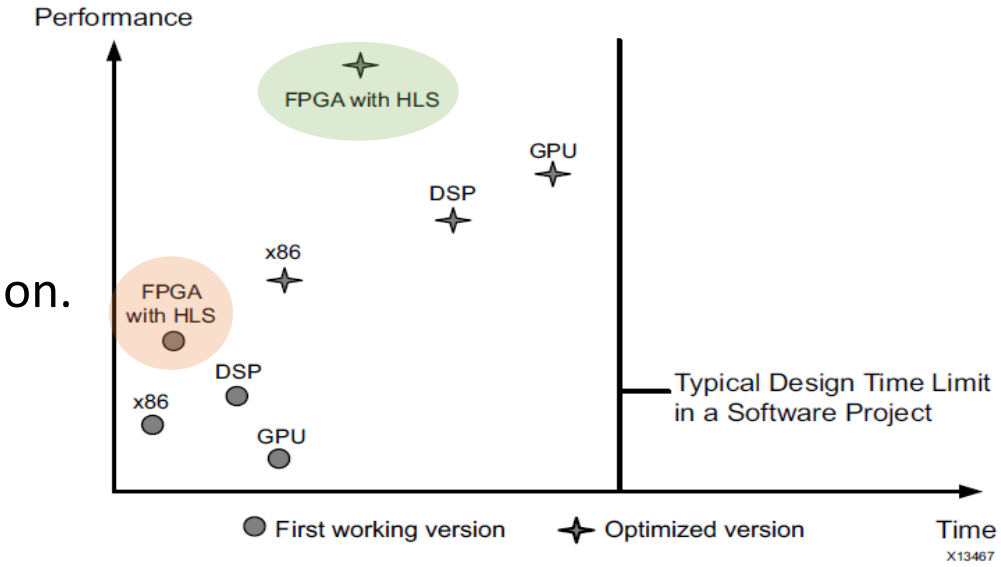


High-Level Synthesis Flow

☺ Easier to implement.

Fast design space exploration.

☹ Hard to grasp hardware implementation details.



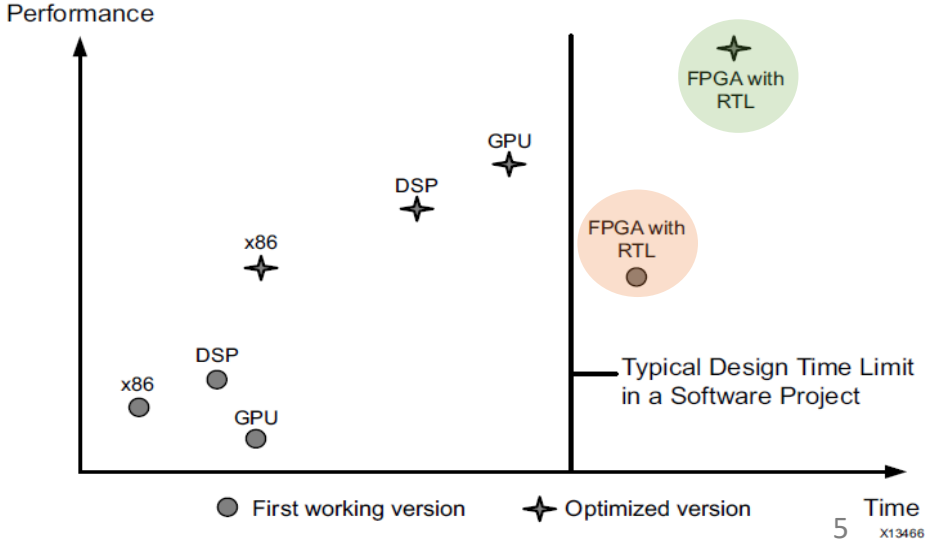
RTL Implementation Flow

☺ High performance.

☹ High development cost.

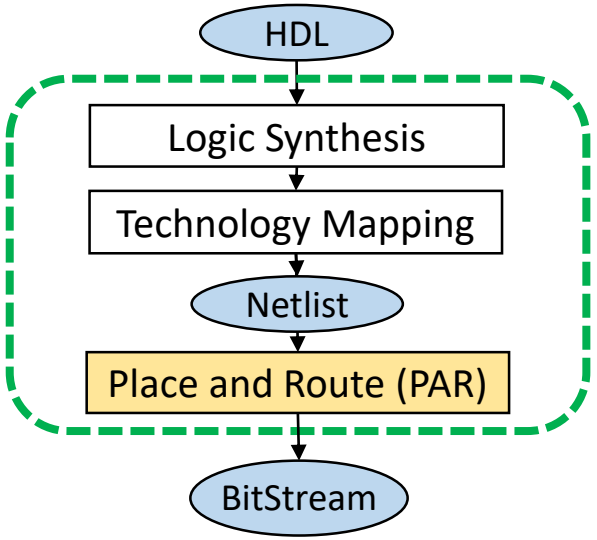
Error-prone and time-consuming.

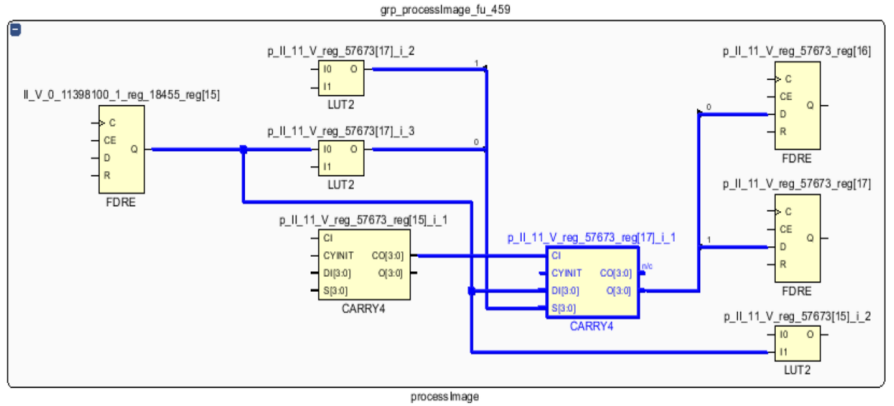
Hard to debug.



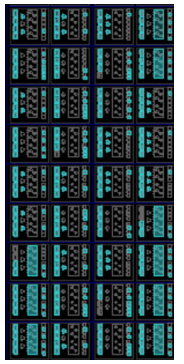
[Source from Xilinx]

Issue: Routing Congestion

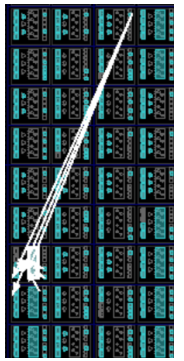




Netlist



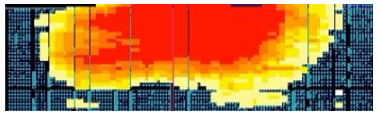
Placement



Routing

In FPGA design, routing contributes a lot to delay and resource utilization.

Congested region



➡ Wires are detoured ➡ Longer delays

More routing resources ➡ Degrade performance

Implementation failures

How to solve this issue?

- Physical design: Routability driven placement.
  - Time-consuming: invoke the router repeatedly.
  - Congestion prediction model guides the placement.
  - Machine learning techniques.

    Features: *#bounding boxes, half-perimeter wirelength (HPWL), #pins …*

- When the abstraction level increases…
  - Beneficial: easier to resolve congestion at C source-code level.
  - Time-consuming: run the C-to-bitstream flow.
  - Congestion prediction is required.

Question: How to predict routing congestion in HLS?

Challenges

- GUI-based “back to verilog” for each CLB after PAR.
  - Not applicable to build a dataset containing a large number of samples.
  - We need “back to C”.

➡ Automate the process: congestion metrics per CLB -> operations in C code.

- Lack of physical metrics or features at a higher level of abstraction.
  - Physical features obtained during placement: *#bounding boxes, HPWL, #pins …*

➡ Extract new informative features in HLS.

- Choice of machine learning models.
  - Linear, ANN, Decision tree…

➡ Explore different kinds of machine learning models.

Motivational Example

```
2897  #pragma HLS inline off
2898  MyPoint p;
2899  int result;
2900  int step;
2901
2902  int u,v;
2903  int x,y,i,j,k;
2904
2905  /** Image Line buffer ( 24 BRAMs ) */
2906  unsigned char L[WINDOW_SIZE-1][IMAGE_WIDTH];
2907  #pragma HLS array_partition variable=L complete dim=1
2908
2909  /** Image Window buffer ( 1250 registers )*/
2910  static int_I I[WINDOW_SIZE][2*WINDOW_SIZE];
2911  #pragma HLS array_partition variable=I complete dim=0
2912
2913  /** Integral Image Window buffer ( 625 registers )*/
2914  static int_II II[WINDOW_SIZE][WINDOW_SIZE];
2915  #pragma HLS array_partition variable=II complete dim=0
2916
2917  /** Square Image Window buffer ( 1250 registers ) **/
2918  static int_SI SI[WINDOW_SIZE][2*WINDOW_SIZE];
2919  #pragma HLS array_partition variable=SI complete dim=0
2920
2921  /** Square Integral Image Window buffer ( 625 registers )*/
2922  static int_SII SII[SQ_SIZE][SQ_SIZE];
2923  #pragma HLS array_partition variable=SII complete dim=0
2924
2925
2926  Initialize0u :
2927  for ( u = 0; u < WINDOW_SIZE; u++ ){
2928  #pragma HLS unroll
2929    Initailize0v:
2930    for ( v = 0; v < WINDOW_SIZE; v++ ){
2931    #pragma HLS unroll
2932      II[u][v] = 0;
2933    }
2934  }
```

For HLS-based designs,

- Applications are implemented with C/C++.

  ➡ Hardware-friendly C code.

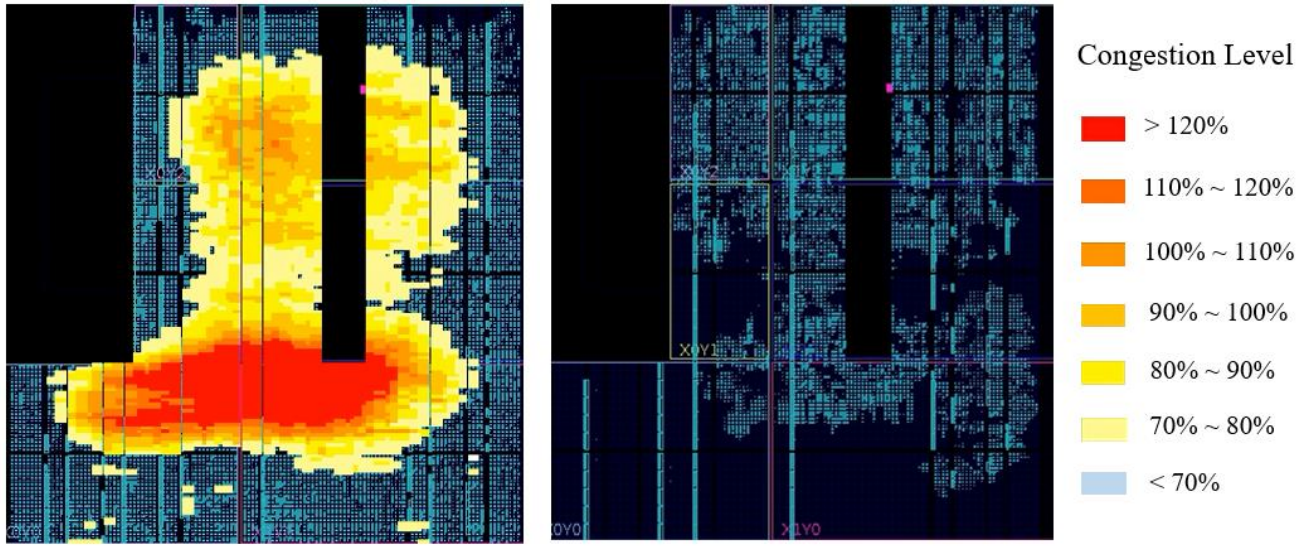
- Synthesis directives are applied to configure the design.
  - *Loop unrolling*
  - *Loop pipelining*
  - *Array partitioning*
  - *Function inlining*
  - …

  ➡ Careful selection of directives.

Motivational Example

Implementation	WNS (ns)	Max Freq. (MHz)	Latency (cycles)	Max Congestion (%)
With directives	-13.643	42.3	1.08×10^6	178.96
Without directives	-0.066	99.3	1.73×10^7	58.51

Note: WNS denotes the worst negative slack.



(a) With directives (b) Without directives

Routing Congestion Maps.

Application: *Face Detection.*

By applying several directives as shown in the application code:

- Latency (cycles) ↓ ☺
- Congestion ↑ ☹
- Max frequency ↓ ☹

Time cost:

- Logic synthesis and PAR: nearly 7 hours.
- HLS: several minutes.

By early detection, we can avoid non-friendly coding style and improper directives!

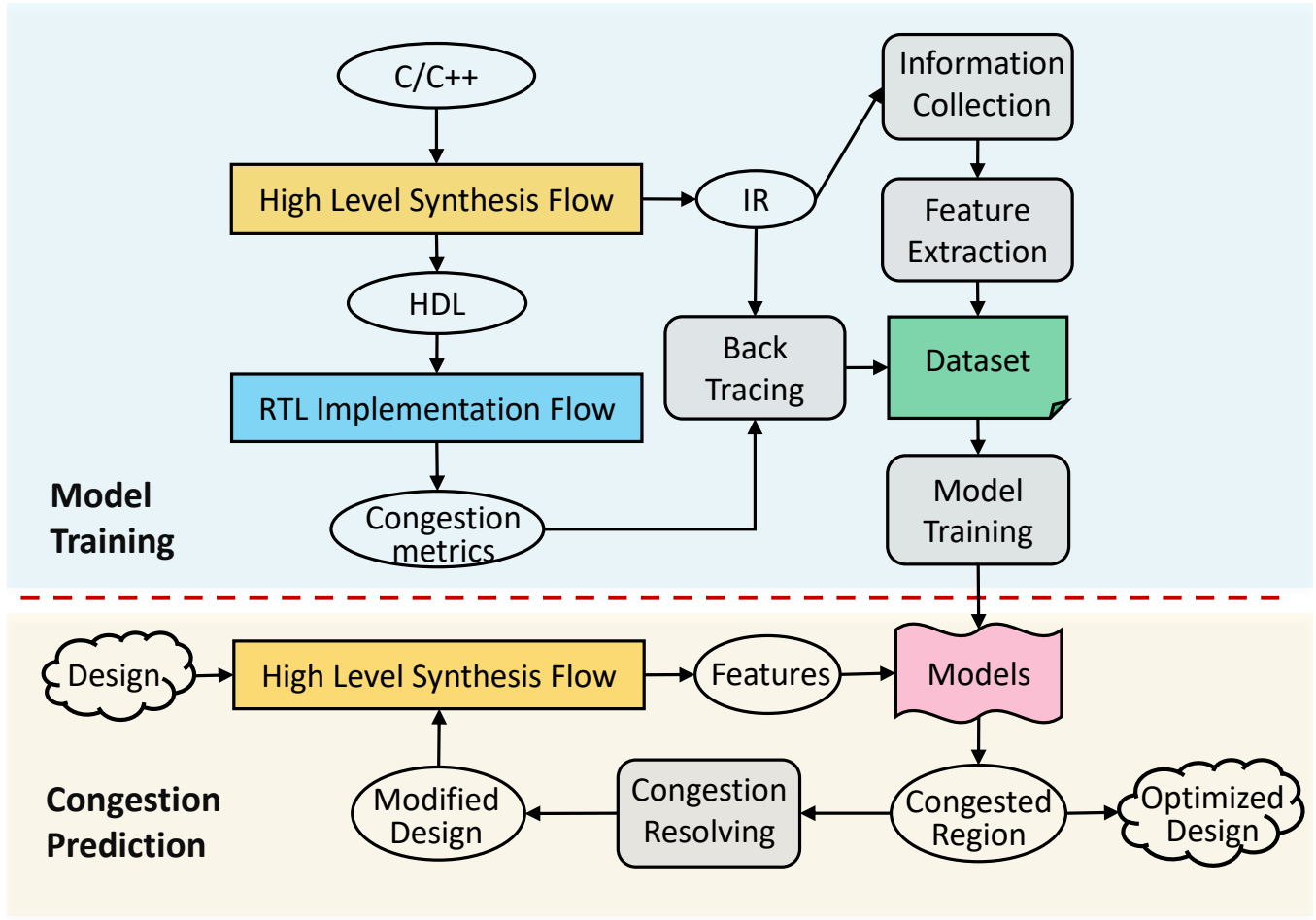
Our Contributions

- To the best of our knowledge, we are the first to build a routing congestion prediction model in FPGA HLS.
- We develop an automatic tool to back trace the congestion metrics of CLBs and link with the HLS IR.
- We propose seven informative categories of features and compare three machine learning models.
- We propose effective solutions to resolve routing congestion.

Outline

- Background
- Machine-Learning Based Methodology
- Experimental Results
- Conclusion

Overview of our approach



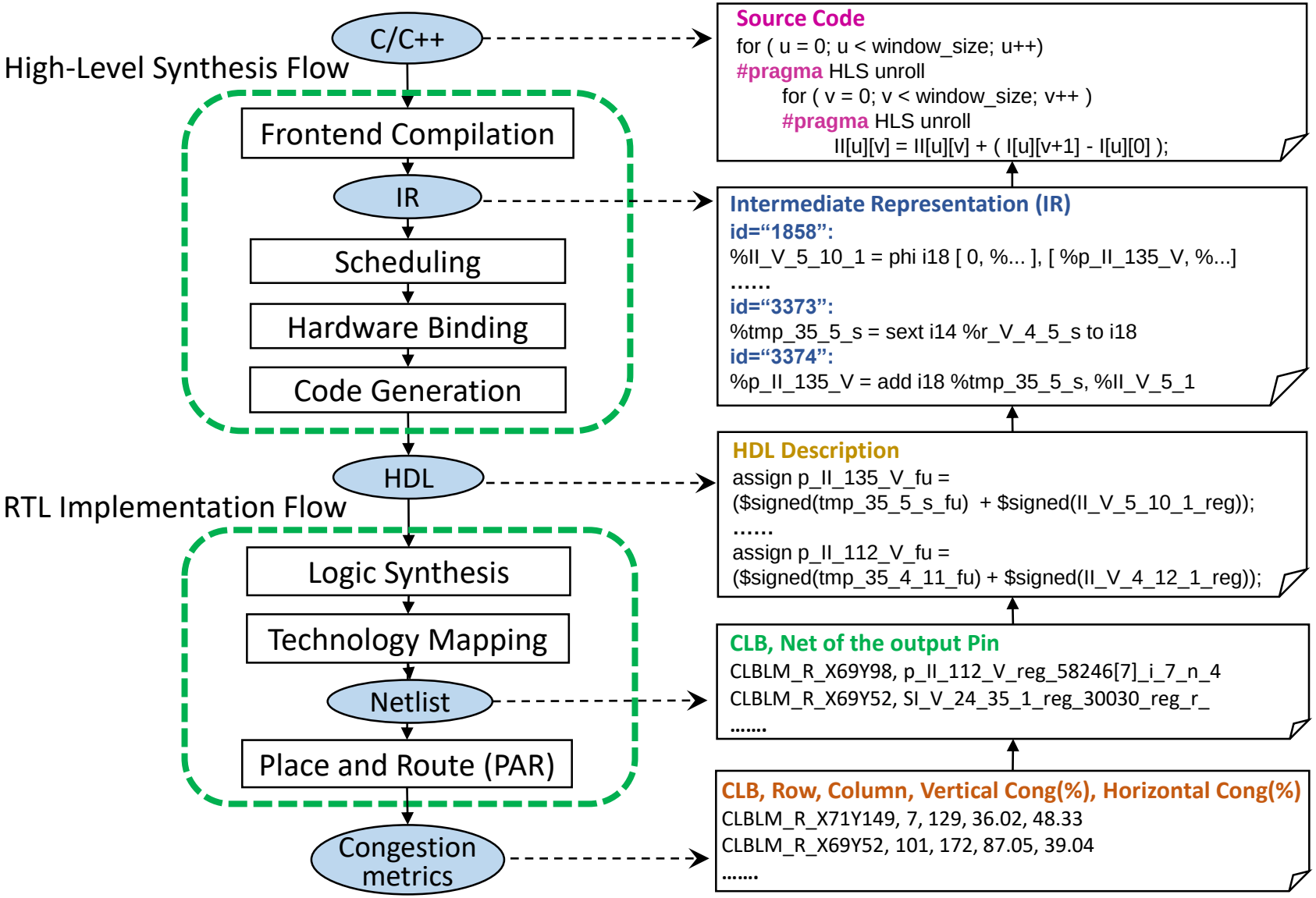
Training Phase

- Construct the dataset for training.
  - Back tracing
  - Information collection
  - Feature extraction
- Model training

Prediction Phase

- Congestion prediction
- Congestion resolving

1. Automatic Back Tracing



C statements in the source code.

IR operations in LLVM IR.

RTL operations in HDL description.

Net of the output pin for each CLB.

Congestion metrics and corresponding CLB coordinates.

2. Information Collection

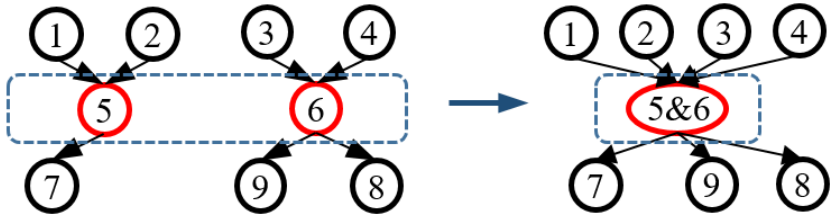
Each sample in the dataset contains:

- Features of each operation.
- Labels: corresponding congestion metrics.

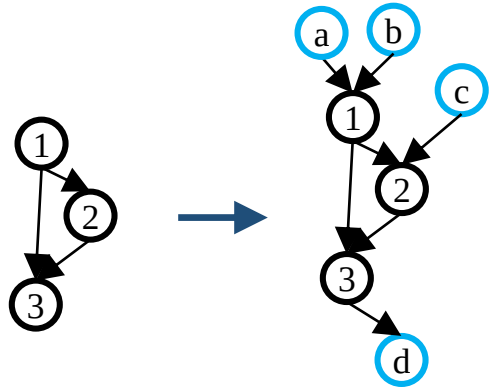
To extract features, a graph is constructed to store the HLS-based information.

- Node: HLS IR operations.
- Node attributes: resource usage, bitwidth, delay…
- Edge: dependency among operations.
- Edge weight: the number of wires for the corresponding connection.

When constructing the graph, we also consider…



Resource sharing: merge the nodes that share the RTL module.



Function Interface: add I/O port in the graph.

3. Features

Category	Feature Descriptions
Bitwidth	Bitwidth of each operation.
Inter-connection	Fan-in and fan-out of each operator and their summation; #predecessors, #successors and the summation; The max. number of wires among all the connections to one-hop neighbors and its percentage of the total fan-in and fan-out. Corresponding features after including two-hop neighbors.
Resource (for each type)	Resource usage and utilization ratios of each operation; The total resource usage and utilization ratios of all the predecessors, successors and their summation; The max. resource usage and corresponding percentage among all the one-hop neighbors; Corresponding features after include two-hop neighbors.
Timing	Delay(ns) and latency(clock cycles) of each operation.
$\frac{\#Resource}{\Delta T_{cs}}$	Resource usage and utilization ratios of predecessors/successors, divided by the subtraction of control states ΔT_{cs}; Corresponding features for two-hop neighbors.
Operator Type	The operation type of each operator; The number of each kind of operations among one-hop neighbors.
Global Information	Resource usage of the top-level function (F_{top}), the function in which the operation is located (F_{op}) and the corresponding percentage of the resources of F_{top}; Target/estimated clock period and clock uncertainty of F_{top} and F_{op}; Memories: #words, #banks, #bits and #primitives(words*bits*banks); Multiplexers: number, resource usage, input size and bitwidth.

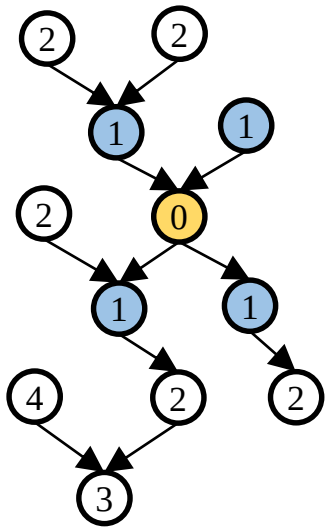
3. Features

In summary, for each operator, we consider:

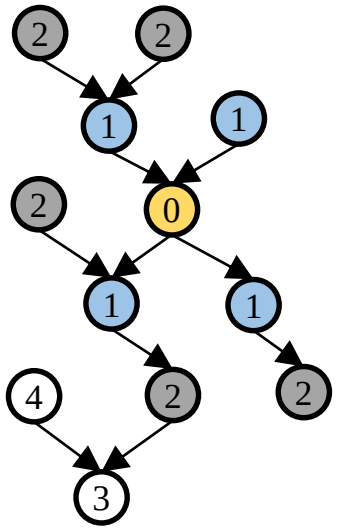
- Features of the operator itself.
  - *Bitwidth, fan-in, fan-out, timing-related features, resource usage…*
- Features that reflect the global information.
  - *Total resource usage of the functions, target/estimated clock period, #mux…*
- Features that reflect the impact of neighboring operators.
  - One-hop neighbors.
  - Two-hop neighbors.

3. Features

Neighboring operators:



One-hop neighbors

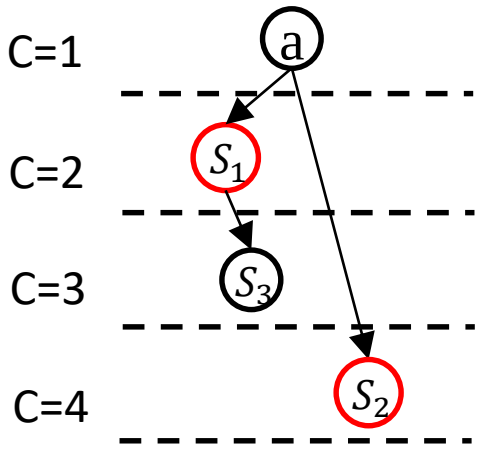


Two-hop neighbors

Features that reflect the impact of neighbors:

- ✓ Inter-connections
- ✓ Resource usage
- ✓ Spatial distance

$$\frac{\#Resource}{\Delta T_{cs}}$$



S_1 and S_2 are two successors of operator a.

- S_1: execute immediately after a;
- S_2: execute several cycles later.

➡ The distance constraints between a and S_1, S_2 are different.

For S_1, ΔT_{cs} = 1.
For S_2, ΔT_{cs} = 3.

4. Model Training

Sample filtering: filter outliers and improve the quality of the dataset.

Unroll a loop with a **large** unrolling factor.

⬇

Multiple **copies** of the same operation.

⬇

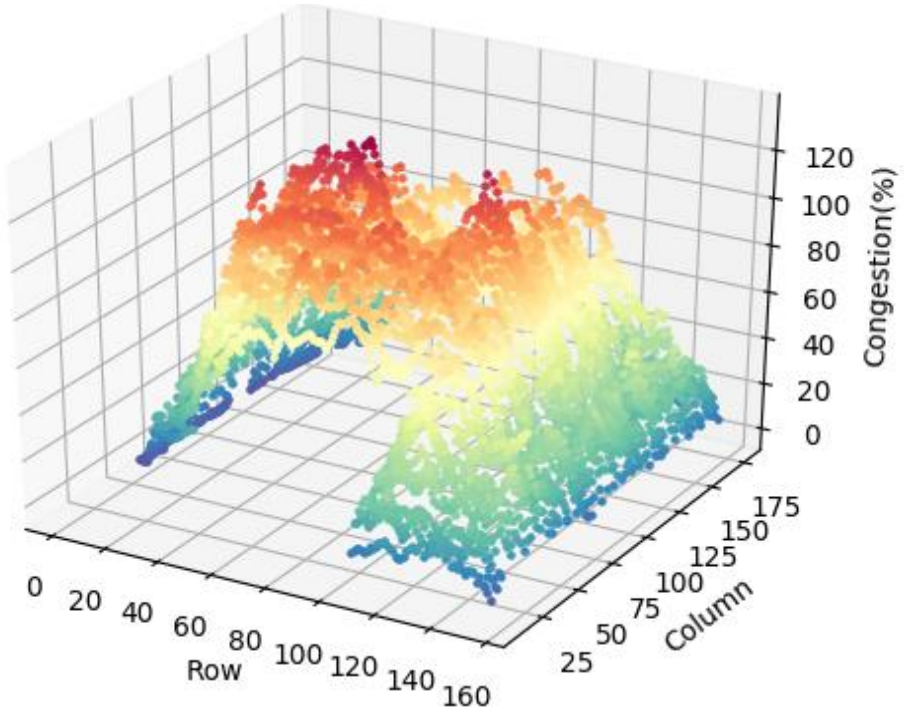
These copies can be placed to **distant** locations.

⬇

Some are placed around the margin of FPGA.

⬇

Their congestion metrics deviate from most of the replicas (~3% outliers).



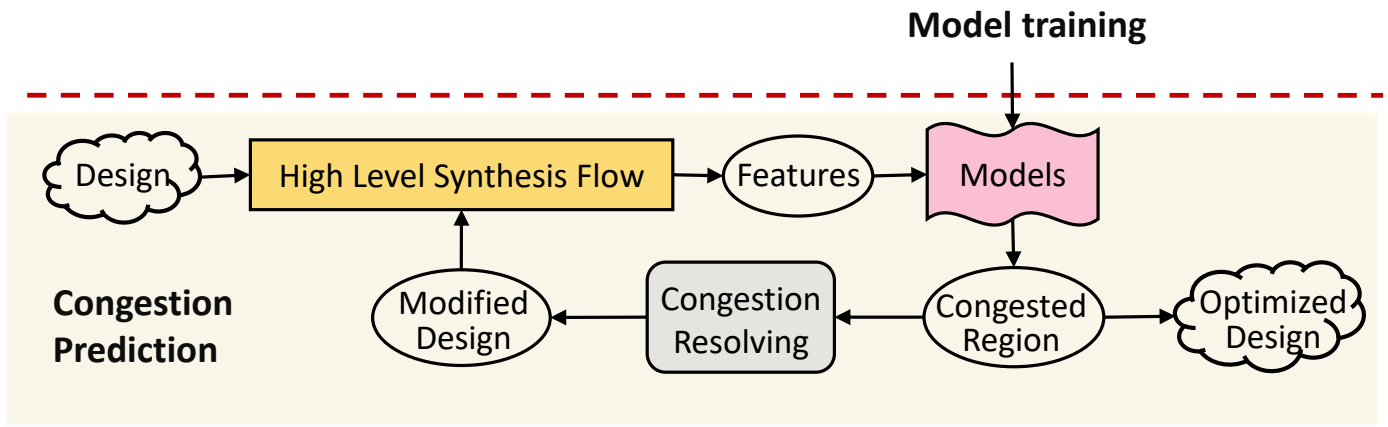
Distribution of the vertical routing congestion metrics for *Face Detection* on FPGA.

4. Model Training

Machine learning models: compare and select the best model for our problem.

- Lasso linear model.
  - Linear relationships.
  - Tuning parameter: the constant α that multiplies the L1-norm.
- Artificial neural network (ANN).
  - Several hidden layers between the input and output layers.
  - Tuning parameter: a number of hyperparameters.
- Gradient boosted regression trees (GBRT)
  - Multiple weak prediction models are combined to form a powerful regression ensemble.
  - Tuning parameter: the number of estimators, the learning rate…

5. Congestion Prediction



- Based on the trained model, the congested regions can be detected in the C/C++ source code.
- Analyze the reasons of the routing congestion and find solutions.
  - Modify source code.
  - Change synthesis directives.

➡ ***Case Study***

- Recursively mitigate congestion and optimize the target application.

Outline

- Background
- Machine-Learning Based Methodology
- Experimental Results
- Conclusion

Experiments

Benchmark suite: Rosetta.

Device: Xilinx FPGA xc7z020clg484.

Our dataset contains 8111 samples.

- 302 features in seven categories.
- 3 labels: vertical congestion metric, horizontal congestion metric and their average.

Tools: Vivado design suite 2018.1, Scikit-learn machine learning library.

The property of the benchmarks are shown in:

Congestion Metrics	WNS (ns)	Frequency (MHz)	Vertical Congestion (%)	Horizontal Congestion (%)	Avg. (V, H) Congestion (%)
Max	-3.253	75.5	133.33	178.96	144.87
Min	-13.643	42.3	5.06	8.90	6.73
Avg.	-8.386	54.4	60.58	72.47	64.89

Estimation Accuracy

Regression Models		Vertical Congestion (%)		Horizontal Congestion (%)		Avg. (V, H) Congestion (%)	
		MAE	MedAE	MAE	MedAE	MAE	MedAE
Not Filtering	Linear	13.90	10.88	18.02	12.62	13.73	9.94
	ANN	12.19	7.91	17.68	12.62	12.27	8.17
	GBRT	10.55	7.37	15.71	10.89	10.57	6.78
Filtering	Linear	12.41	9.20	17.48	12.16	12.76	9.50
	ANN	10.23	7.43	16.61	11.78	11.67	7.83
	GBRT	9.59	6.71	14.54	10.05	9.70	6.81

- 10-fold cross validation and grid search.
- Mean absolute error (MAE) measures the average value of the absolute relative errors.
- Median absolute error (MedAE) reflects the distribution of the absolute relative errors.
- Best performance: the **GBRT** model.

Important Features

Congestion Metrics	Vertical Congestion	Horizontal Congestion	Avg. (V, H) Congestion
Important Feature Categories	$\frac{\#Resource}{\Delta T_{cs}}$	$\frac{\#Resource}{\Delta T_{cs}}$	Resource
	Resource	Resource	$\frac{\#Resource}{\Delta T_{cs}}$
	Interconnection	Interconnection	Interconnection
	Global (Mux)	Global (Memory)	Global (Mux)

Importance rank

- Importance of different categories of features is assessed through the GBRT model.
- $\frac{\#Resource}{\Delta T_{cs}}$ has the greatest impact on both vertical and horizontal congestion metrics.
- The related information of multiplexers and memories has a greater effect than other global features.

Case Study

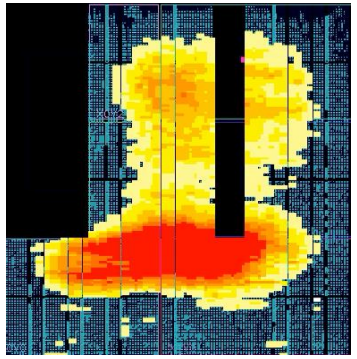
Application: *Face Detection*

Implementation	WNS (ns)	Max Freq. (MHz)	Δ*Latency* (cycles)	Max Congestion Vert, Hori (%)	#Congested CLBs (>= 100%)
Baseline	-13.643	42.3	1.08×10^6	133.33, 178.96	1272

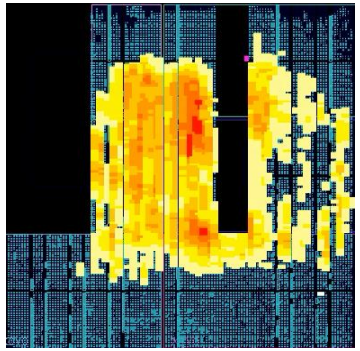
Baseline: Original design.

- Severe routing congestion degrades the maximum frequency significantly.
- Performance trade-off ☹

Baseline(H)



Baseline(V)



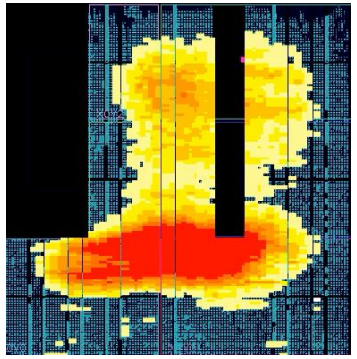
Case Study

Implementation	WNS (ns)	Max Freq. (MHz)	$\Delta Latency$ (cycles)	Max Congestion Vert, Hori (%)	#Congested CLBs (>= 100%)
Baseline	-13.643	42.3	1.08×10^6	133.33, 178.96	1272
Not Inline	-3.504	74.1	+23	129.85, 97.60	193

Step 1: Not Inline

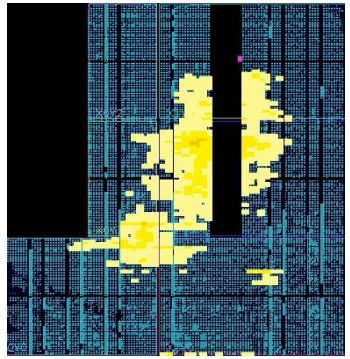
- The classification function contains 52 classifiers.
- *Function inlining* increases the complexity in C synthesis and generates a larger design.
- Not inline the classifiers.

Baseline(H)

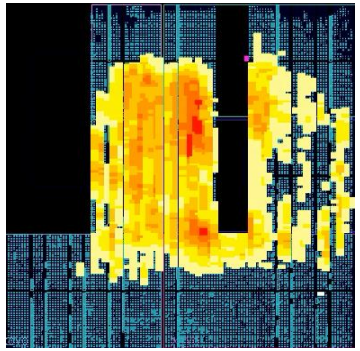


Step 1

Not Inline(H)

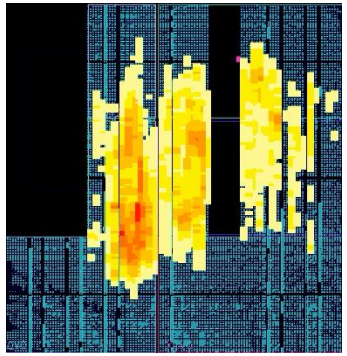


Baseline(V)



Step 1

Not Inline(V)



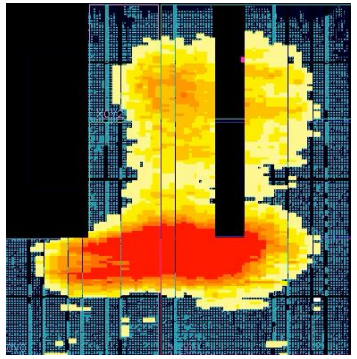
Case Study

Implementation	WNS (ns)	Max Freq. (MHz)	$\Delta Latency$ (cycles)	Max Congestion Vert, Hori (%)	#Congested CLBs (>= 100%)
Baseline	-13.643	42.3	1.08×10^6	133.33, 178.96	1272
Not Inline	-3.504	74.1	+23	129.85, 97.60	193
Replication	-0.767	92.9	+0	106.15, 104.73	17

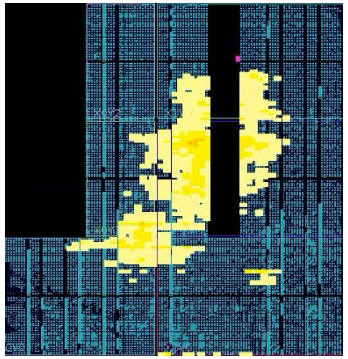
Step 2: Replication

- All the classifiers access the same completely partitioned array and multiple classifiers share the same inputs.
- A large number of interconnections.
- Modify the source code by replicating the values of the input data and sending copies to different classifiers.

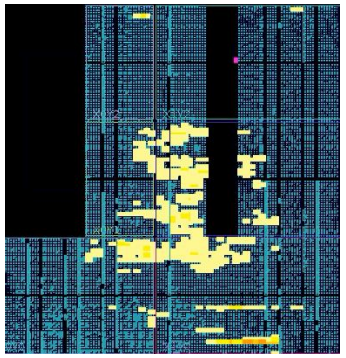
Baseline(H)



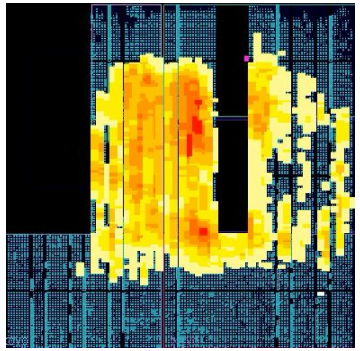
Step 1
Not Inline(H)



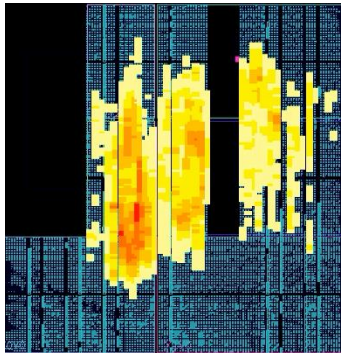
Step 2
Replication(H)



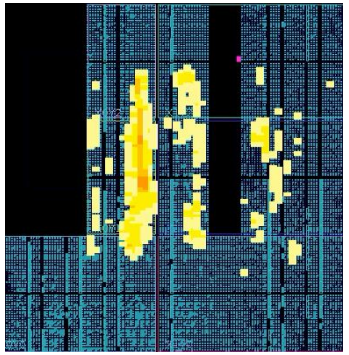
Baseline(V)



Step 1
Not Inline(V)



Step 2
Replication(V)



Outline

- Background
- Machine-Learning Based Methodology
- Experimental Results
- Conclusion

Conclusion

- We propose a novel machine-learning based methodology to predict routing congestion in FPGA HLS.
- Experiments show that the GBRT model achieves the highest prediction accuracy.
- Based on our model, routing congestion can be mitigated step by step and the performance of *Face Detection* can be improved significantly.

Thank you for listening!

Q & A