Rochester Institute of Technology

RIT Digital Institutional Repository

Theses

5-2024

# Evolutionary Neural Network for Optimized Clock Tree Synthesis

Patrick Jeffery

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation

Jeffery, Patrick, "Evolutionary Neural Network for Optimized Clock Tree Synthesis" (2024). Thesis. Rochester Institute of Technology. Accessed from

This Master's Project is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

# EVOLUTIONARY NEURAL NETWORK FOR OPTIMIZED CLOCK TREE SYNTHESIS

by

PATRICK JEFFERY

GRADUATE PAPER

Submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE
in Electrical Engineering

Approved by:

__________________________________________

Mr. Mark A. Indovina, Lecturer
*Graduate Research Advisor, Department of Electrical and Microelectronic Engineering*

__________________________________________

Dr. Ferat Sahin, Professor
*Department Head, Department of Electrical and Microelectronic Engineering*

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

MAY, 2024

# Dedication

To my parents, David, and Tricia Jeffery; my brothers, Christopher and Mathew Jeffery; and my partner, Lily Kimpel. I could have never done it without you. Love you all.

Patrick Jeffery

# Declaration

I hereby declare that except where specific reference is made to the work of others, that all content of this Graduate Paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This Graduate Project is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Patrick Jeffery

May, 2024

# Acknowledgements

I would like to first thank my graduate advisory and mentor, Mark A. Indovina, for his continuous support and direction, this project, and many of my academic and professional achievements, would have not been possible without his support. I would like to thank my academic advisers, Stephanie Krebbeks and Sarah Dresnack-Radtke, for helping me navigate my college career from start to finish. Finally, I'd like to thank the RIT Department of Electrical and Microelectrical Engineering, for allowing me the opportunity to grow my professional and personal life in such an incredible way.

Patrick Jeffery

# Abstract

Clock Tree Synthesis (CTS) is a complex and in depth process that, in modern designs, would take an individual months if not years to get a working design. Tools such as Cadence Innovus and Synopsys ICC provide excellent support for CTS and can be used to create well optimized trees; allowing the user to edit the tree's generation down to a single buffer. However, even these tools can fall short of a perfectly optimized route and oftentimes need a capable user to direct them in the right direction just to get a functioning clock tree. The aim of this work is to provide an additional tool to further aid users in pursuit of fully functional and highly optimized clock trees. The developed network is an evolutionary neural network, built on NEAT-Python, trained on ten variations of a results character conversion block designed for a dual-tone multi-frequency receiver. The network is meant to provide users with clock tree generation parameters such that the routed tree will be optimal. The network's success is evaluated based on its growth throughout training and its ability to suggest optimal parameters for one hundred variations of the same design. It is found that the network grows steadily, suggesting that given enough time it could master CTS. That being said, the final test reveals that the network is only slightly superior to default CTS parameters and is far too inconsistent to be considered successful. These results are carefully evaluated to suggest improvements for future attempts to develop a similar neural network.

# Contents

**Contents** v

**List of Figures** viii

**List of Tables** ix

**1 Introduction** 1

1.1 Research Goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

**2 Bibliographic Research** 5

2.1 Clock Tree Synthesis . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2 ML and ANNs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.3 ML and ANNs for CTS . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.4 Supporting Documentation . . . . . . . . . . . . . . . . . . . . . . . 8

**3 Clock Tree Synthesis** 10

3.1 Clock Tree Implementations . . . . . . . . . . . . . . . . . . . . . . 11

3.1.1 Conventional . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.1.2 Multi-Source . . . . . . . . . . . . . . . . . . . . . . . . . 12

3.1.3 Mesh . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.2 Clock Tree Structure . . . . . . . . . . . . . . . . . . . . . . . . 13
3.3 Multi-bit Versus Single-bit DFFs . . . . . . . . . . . . . . . . . . . 15

**4 NeuroEvolution of Augmenting Topologies (NEAT) Algorithm 17**

4.1 Brief NEAT Overview . . . . . . . . . . . . . . . . . . . . . . . . 17
4.2 Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.2.1 NEAT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.2.2 Stagnation . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.2.3 Reproduction and Species Set . . . . . . . . . . . . . . . . . . 20
4.2.4 Default Genomes . . . . . . . . . . . . . . . . . . . . . . . . 21
4.2.4.1 Node Activation . . . . . . . . . . . . . . . . . . . . 21
4.2.4.2 Node Aggregation . . . . . . . . . . . . . . . . . . . 22
4.2.4.3 Node Bias . . . . . . . . . . . . . . . . . . . . . . . 23
4.2.4.4 Genome Compatibility . . . . . . . . . . . . . . . . . 24
4.2.4.5 Connection and Mutation . . . . . . . . . . . . . . . . 25
4.2.4.6 Feed Forward . . . . . . . . . . . . . . . . . . . . . 26
4.2.4.7 Network Parameters . . . . . . . . . . . . . . . . . . 27
4.2.4.8 Node Response . . . . . . . . . . . . . . . . . . . . 28
4.2.4.9 Connection Weight . . . . . . . . . . . . . . . . . . . 29

**5 Training Environment 31**

5.1 The Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.2 Trainer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.2.1 Network Inputs . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.2.2 Network Outputs . . . . . . . . . . . . . . . . . . . . . . . . 37

5.2.3 Network Fitness Function . . . . . . . . . . . . . . . . . . . . . . . 40
5.2.4 Randomizing Floor Plans . . . . . . . . . . . . . . . . . . . . . . . . 41
5.2.5 Data Reporting and Collection . . . . . . . . . . . . . . . . . . . . . 43

**6 Results 44**
6.1 Final Neural Network . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
6.2 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
6.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

**7 Conclusion 54**
7.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

**References 56**

**I Source Code I-1**
I.1 Training Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . I-1
I.2 Automatic File Management (FileMan) . . . . . . . . . . . . . . . . . . I-17
I.3 Testing Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . I-26

**II Supporting Code II-36**
II.1 Synopsys tcl Run File . . . . . . . . . . . . . . . . . . . . . . . . . . II-36

**III ICC Shell III-54**
III.1 Clock Tree Options . . . . . . . . . . . . . . . . . . . . . . . . . . . III-54
III.2 Floor Plan Options . . . . . . . . . . . . . . . . . . . . . . . . . . . III-56

# List of Figures


3.1 Clock Tree Implementations. Left to Right: Conventional, Multi-Source, and Mesh Clock Tree Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
3.2 Clock Tree Structures. A) Y-Tree; B) H-Tree; C) X-Tree; D) Serial Tree . . . 14
3.3 Replacing Single-Bit with Multi-Bit Flip Flops . . . . . . . . . . . . . . . . . 16

5.1 Results Character Conversion Block with Default Settings (Clock Tree Highlighted in Yellow) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
5.2 'L' Shaped RCC Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
5.3 'X' Shaped RCC Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
5.4 Floor Plan Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42

6.1 Average Fitness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
6.2 Best Fitness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
6.3 Standard Deviation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
6.4 Best Genome (top), Best Genome Without Newly Generated Genomes (bottom) 48
6.5 Generation Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
6.6 Final Genome Layout . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
6.7 Starting Genome Layout . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
6.8 Final Results Histogram. Average = +6.4720 . . . . . . . . . . . . . . . . . 51

# List of Tables

4.1 NEAT Configure Data . . . 19
4.2 Stagnation Configure Data . . . 20
4.3 Reproduction and Species Set Configure Data . . . 21
4.4 Node Activation Configure Data . . . 22
4.5 Node Aggregation Configure Data . . . 23
4.6 Node Bias Configure Data . . . 24
4.7 Genome Compatibility Configure Data . . . 25
4.8 Connection and Mutation Configure Data . . . 26
4.9 Feed Forward Configure Data . . . 27
4.10 Network Parameters Configure Data . . . 28
4.11 Node Response Configure Data . . . 29
4.12 Connection Weight Configure Data . . . 30

5.1 Design Characteristics used for Network Inputs . . . 38
5.2 Non-Boolean CTS Parameters . . . 39

# Chapter 1

# Introduction

All digital designs require a clock to keep time and drive digital components, the clock is somewhat comparable to a "heart" of a design. A well optimized clock can help a design avoid critical issues such as excess power consumption, resource usage, area overhead and more. Unfortunately, the clock is also one of the most resource and power hungry components in modern circuitry, sometimes consuming as much as 30-50% of the total power [1] and occupying a quarter of the available metal layers. Moreover, optimizing a clock tree is becoming increasingly challenging as the size and complexity of modern circuitry continues to grow. Along with this growth, electronic design automation (EDA) tools have also grown in complexity, demanding developers to be more well versed with settings and parameters that are often easily overlooked.

Further increasing the difficulty of optimized Clock Tree Synthesis (CTS), dozens of developments have been made in the attempt to increase the size of a developers tool box. Mesh and Multi-source CTS have become popular alternatives to conventional CTS, although each has their own use case and conventional is still, arguably, the most popular [2]. Beyond the layout of the tree, designers have to worry about the shape; H-Tree or otherwise, single-bit versus multi-bit DFFs, differently sized inverters in place of buffers, and more [3].

To aid with the daunting task of optimizing a clock tree, multiple machine learning (ML) models have been developed, often to great success [4, 5]. In specific scenarios, these models can take the place of an experienced developer and tune the parameters for clock tree generation to provide a well optimized tree for a range of designs. Specifically, [5] proposes a versatile and effective generative adaptive neural-net algorithm trained on thousands of datum to provide specific clock tree settings for the Cadence Innovus tool, and has become a major motivator for the work presented here.

## 1.1 Research Goals

The objective of this work is to first, provide an overview of CTS and its variations, and second, propose an effective Evolutionary Neural Network as a tool for further CTS optimization. The tool will, ideally, take the place of an experienced designer and will suggest optimal CTS parameters based on a given design. The neural net will be given a series of characteristics of the design based on its default clock tree and will provide the optimal parameters to re-generate the clock tree to have reduced maximum skew, power consumption, and resource consumption. The primary goals of this research is summarized below:

- To research and understand clock tree synthesis, including its variations and limitations.
- To develop an effective evolutionary neural network which will aid designers by suggesting effective CTS parameters based off a designs default routing.

## 1.2 Organization

The structure of this graduate paper is as follows:

- Chapter 2: This chapter gives an in depth analysis of each of the sources used for this work. Including all CTS, machine learning, and neural network research.
- Chapter 3: This chapter provides an overview of clock tree synthesis. Including conventional, multi-source, and mesh clock tree architectures, as well as a discussion of H-tree versus X-tree structures and multi-bit versus single-bit DFFs.
- Chapter 4: This chapter discusses the evolutionary neural network and its python implementation, NEAT-Python. Specifically, this chapter discusses the configuration of the neural network.
- Chapter 5: This chapter provides detail as to how the neural network was trained; including the digital design used, how the design is randomized to provide variable inputs to the network, and how the network interfaces with Synopsys IC and the digital design itself.
- Chapter 6: This chapter discusses how the final, most fit, neural network is tested and its performance on these tests.
- Chapter 7: This chapter gives an overview of the work done and provides suggestions for how future attempts to train an evolutionary neural network for CTS might be improved.
- Appendix I: This section gives the entirety of the python training environment and testing environment used for this work.
- Appendix II: This section gives the tcl file used for integrating the python environments with Synopsys IC and for generating the post-route reports.
- Appendix III: This section gives complete definitions, as described by [6], of some important terms; specifically, the leveraged CTS and floor plan parameters supported by

Synopsys.

# Chapter 2

# Bibliographic Research

## 2.1 Clock Tree Synthesis

The evolutionary neural network designed for this work is indented specifically for optimizing clock tree synthesis through parameter suggestions. As such, much of the research conducted was for CTS. To begin, both [2] and [7] compare the most popular CTS types: conventional, multi-source, and mesh; with the main comparison being between conventional and multi-source. It is found that multi-source takes advantage of the reduced latency and skew offered by mesh, while maintaining a lower power and resource consumption than a pure mesh tree. Multi-source seems to be the best of both worlds, though still limited by power consumption and, of course, timing. Further, [2] discusses the superiority of multi-bit flip-flops as using these buffers can significantly reduce the number of transistors required in VLSI, while [7] suggests some heuristic approaches to improving multi-source specifically for skew optimization. Supporting this research, [8] and [9] offer web article alternatives comparing the three types of clock trees in a much easier to digest manner. Focusing more specifically on multi-source clock tree's, [10] gives an excellent example of how to apply a multi-source symmetric h-tree clock tree

for 7nm technology. Also backs the success of the tree with dozens of experiments performed using Cadence Innovus. Similarly, [1] provides another good example of how to implement a multi-source clock tree, though the "hybrid multi-source clock tree" they propose attempts to a avoid any clock delay calculation accuracy issues by pre-calculating the clock tree driver size needed based on the driver's range. Going back to the multi-bit flip flops (MBFF), [11] compares MBFF to SBFF, specifically for power optimization while taking into account the clock tree to fully optimize power and time saves. Moreover, [12] suggests a method of clock gating where the clock tree is constructed simultaneously with the insertion of clock gates. Alternative methods insert the clock gates after the clock tree insertion, causing a large change to the clock slew, however, building them together allows the slew to be kept in check throughout insertion. Finally, [13] discusses the use of serial clock trees; that is, clock trees that connect from one sink to another in a straight line, and how an averaging technique with a flexible and re-configurable serial clock tree can provide good results with minimal wire usage. Further, provides an excellent discussion of the difference of H-tree, X-tree, Y-tree, and serial clock trees. Similarly, [14] is a web article which compares the popular clock tree shapes using many straightforward and informative diagrams. Many of the sources gathered for this research were found through the help of the index built by [3] which lists other publications discussing different CTS techniques and gives brief overview of the paper's results, making it easy to find relevant sources.

## 2.2 ML and ANNs

A significant portion of the work presented here relies on machine learning (ML), specifically artificial neural networks (ANNs). Because of this, much of the preliminary research conducted focused on general purpose machine learning models and their applications. First, [15] gives an

excellent overview of how and when to use supervised and unsupervised learning. Specifically, the different available approaches to both types of learning, including neural networks, and some of their strengths and weaknesses. While discussed, neural networks are not the main focus but rather the application of machine learning as a whole, hence why this paper is listed here instead of next chapter. Similarly, [16] serves as a modern, real example of how to apply machine learning to advanced driving assistance systems (ADAS). Likewise, [17] discusses a new evolutionary system: evolutionary programming (EPNet) that focuses on developing an artificial neural network's behavior by using mutations such as partial training and node splitting to maintain the behavioral links between generations. EPNet proves to be effective for outcome prediction, though it takes more generations to highly root itself into any one solution. [18] propose a recurrent neural network for any non-smooth convex optimization problem, serving as a good example of the best practices for developing and applying neural networks. More directly related to the work presented here, [19] delves deeper into NEAT, recall NEAT is the evolutionary neural network model used for this work, and provides a comprehensive comparison to the temporal difference method: Sarsa. They're experiments find that NEAT can be more accurate than Sarsa, though it takes more generations to do so. Further, they find that NEAT learns deterministic environments best, making it preferable for more complex problems. Finally, both [20] and [21] discuss in great detail the NeuroEvolution of Augmenting Topologies (NEAT) used extensively throughout this work. [21] is the original and more commonly referenced of the two, though both papers are used to gain the best understanding of NEAT.

## 2.3 ML and ANNs for CTS

Many researchers have attempted to develop a machine learning model specifically for CTS, as this work also does, many of whom succeeded. [5] is the primary motivator for the work presented here. They discuss a generative adversarial network (GAN) capable of, first, predicting the outcomes of certain CTS parameters on a given design and, second, optimizing the clock tree by suggesting CTS parameters; this is almost identical to what is attempted in this grad paper. However, GAN is a highly complex network requiring experienced developers to oversee its development, while NEAT is more fluid and, ideally, independent during training. [4] also uses machine learning, specifically TUNA, to fine tune CTS parameters for optimized timing and power consumption. Although, they're paper is very brief, while it doesn't delve deep into how the model is trained, it does give a good, easy to read overview of the possible effectiveness of using machine learning for optimized CTS. Lastly, [22] builds a large database of 1300 samples from 65 designs of C-to-FPGA results to train a machine learning model to accurately predict the post-implementation metrics of the entire design, not specifically CTS, given a large set of features pertaining to the design. While the work they present is similar to the work of this grad paper, the scope of their work is much larger and relies on a massive database that's infeasible for this graduate paper.

## 2.4 Supporting Documentation

Some manuals and open-source code is referenced throughout this work, the most relevant of which are discussed here. First, [23] is used to gain the best understanding of the place and route work flow. This manual is specific to Cadence Innovus, though its suggested flow and explanations are universally applicable for EDAs. Likewise, the information provided in [6] is invaluable for this work as it heavily relies on Synopsys ICC for placing clock trees and

analyzing them.

On the other hand, [24] provides the open-source NEAT-Python implementation used for all neural network training in this work, and has become a cornerstone of the research conducted here. Similarly, [25], written by the developers of [24], is the excellent documentation for NEAT-Python. The documentation is referenced constantly throughout this work, specifically for the discussion of the configuration file used to structure the neural network.

# Chapter 3

# Clock Tree Synthesis

Clock tree synthesis is the step in the digital design process in which the clock is added to a design. According to the design flow outlined by [23], the CTS step should come just after placing and optimizing all standard cells and tie cells, and before adding filler cells and routing the design. The mere fact that CTS comes before standard routing is proof of the step's importance. Even still, it may be surprising to learn that the clock can often consume a significant amount of the total power budget; around 30-50% [1], and will usually have at least two metal layers dedicated solely to the clock.

The goal of CTS is to ensure the clock is uniformly distributed to all sequential elements of a design, allowing blocks to reliably communicate with each other. Further, CTS is often a lengthy and complicated process as it aims to optimize the maximum skew, power consumption, wire length, and other characteristics of a design. Oftentimes the optimization is necessary to allow a design to work at all, for example, the first pass of CTS done by commercially available tools may result in timing violations, meaning if the design were printed to actual silicon and tested, it almost certainly would not work. As such, most commercial tools take multiple passes and many attempts to get a well designed tree, even if the parameters are ideally set. If the tools

parameters are poorly set, tools as powerful as Innovus or Synopsys IC may take many minutes to build, test, and redesign a tree and, even still, it may never build a functioning one.

The parameters and their impact on CTS will be discussed in great detail later on, for now it may be more valuable to discuss some of the variations and difficulties presented by CTS. Beyond the tools specific parameters, designers will also have to worry about the clock tree's implementation. There are three common clock tree implementations: Conventional, Multi-source, and Mesh [2], all of which will be described below. Further, designers are presented with more choices such as the structure of the tree: H-tree, X-Tree, etc. [14], the buffers and registers used [11], and more [3].

## 3.1 Clock Tree Implementations

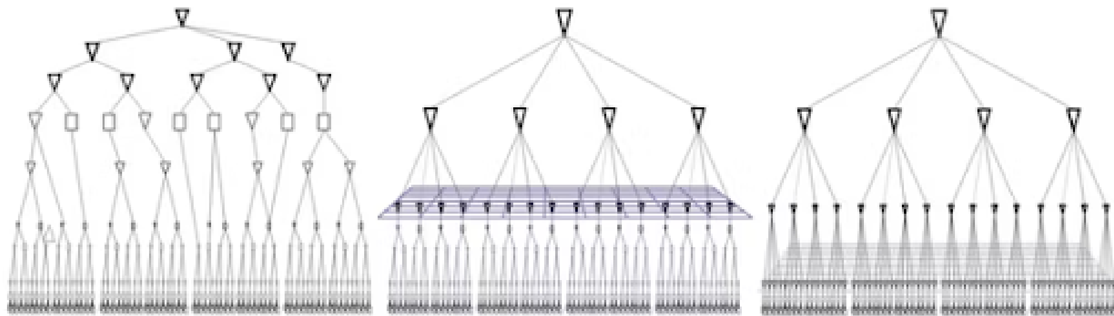

Figure 3.1: Clock Tree Implementations. Left to Right: Conventional, Multi-Source, and Mesh Clock Tree Structure

### 3.1.1 Conventional

Conventional clock tree's are the most popular for smaller designs as it is significantly more simple than its alternatives while still providing reliable results. A Conventional tree will

have one source node, the system clock, which will branch out to buffers, inverters, and any sequential logic blocks that need a clock, see Figure 3.1[8]. It is characterized by its similarity to an actual tree's roots, hence the name "clock tree." The trade off for the simplicity of the conventional tree structure is the relatively poorly optimized results; these trees will often have greater skew and max latency than either mesh or multi-source, as the system will have much greater fan-out and wire length between the root node and the furthest end node [8]. Conventional trees also suffer more from on chip variations (increasing skew and latency) as designs grow larger and more complex. Further, the placement of the source node can have dramatic effects on the final clock tree's effectiveness, more on this later.

### 3.1.2 Multi-Source

Multi-source clock tree's are similar in structure to conventional trees with the exception of a large mesh like grid of metal connecting the clock nets a few layers into the tree, see Figure 3.1. The goal of the mesh is to provide, effectively, a reset for the clock signal as once it reaches this mesh from any of the branches of the root node, the signal can be regenerated and synchronized with neighboring nodes thus reducing on chip variation [2]. This way, the tree can rely on a single source node but still benefit from the decreased skew and latency of having many sources scattered throughout the design. In other words, the single source node allows for decreased power consumption and more reliability in the main shared signal vein, while still allowing the lower level blocks to benefit from well synchronized and minimal fan-out clock nets. Multi-source clock trees are becoming increasingly popular for high performance and high complexity designs such as fast ALUs or GPUs [8].

### 3.1.3 Mesh

Lastly, Mesh clock trees use a similar grid of metal as multi-source, but the grid is placed much lower in the tree's branches, see Figure 3.1, and is an order of magnitude or two more dense than the multi source grid. This, as with multi-source, allows the clock signal to be somewhat regenerated by neighboring nodes just before it is passed to terminal nodes such as sequential logic blocks or registers. The purpose of placing the grid low is that the shared branches above the mesh can be more easily prioritized and made faster, while lower level branches can be very quickly and cheaply passed to terminal nodes. Nevertheless, the shared branches still maintain the majority of the tree's insertion delay and the dense mesh consumes significantly more routing resources than the multi-source mesh but amplifies the synchronization and further decreases skew [8].

## 3.2 Clock Tree Structure

Additionally, designers are faced with multiple clock tree structures, specifically Y-Tree, H-Tree, X-Tree, or Serial, see Figure 3.2[13]. Y-Trees, or wishbones, are very simple and direct methods to get a clock signal to terminal nodes and are very useful for small designs but can have very high skew due to a design's topology. H-Tree structures are the widely agreed upon best solution, as they can minimize skew across all the terminal nodes while using relatively few buffers, but can require more routing resources and are more difficult to implement than other structures. Similarly, X-Trees function on the same principles as H-trees but are more direct and can shorten wire lengths; however, they are only applicable to designs that are non-rectangular which is fairly uncommon [14]. Lastly, serial tree structures, while simple, are very impractical and more or less a proof of concept; however, they are useful in certain situations such as pipeline or very basic designs.

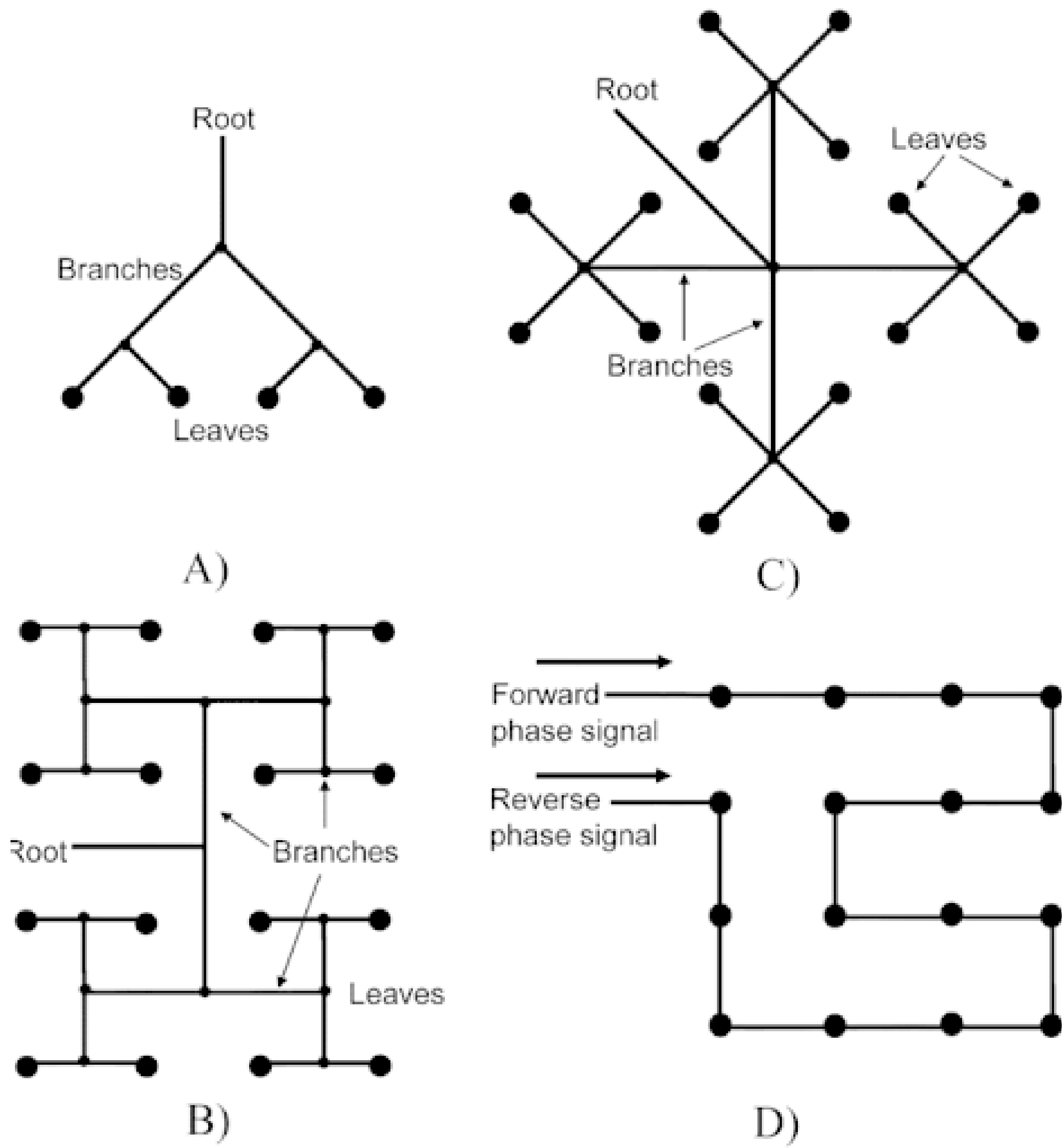


Figure 3.2: Clock Tree Structures. A) Y-Tree; B) H-Tree; C) X-Tree; D) Serial Tree

## 3.3 Multi-bit Versus Single-bit DFFs

Finally, another choice faced by designers is the use of Multi-bit flip flops (MBFFs) versus Single-bit flip flops (SBFFs). Flip flops are used for CTS as they can, effectively, separate the tree into smaller sections, reducing the head node's required drive strength and, therefore, reducing the overall complexity and power consumption of the tree. Multi-bit flip flops are made up of one or more single-bit flops, using the same clock signal to drive drive multiple outputs; Figure 3.3 exemplifies how to replace a series of single-bit flops with multi-bit ones. While this can further reduce a designs complexity and power consumption [11], it also requires a design to be structured such that single-bit flops are close enough to be replaced by a larger flop. This consideration can greatly affect the final structure of a clock tree, adding more strain to a designers work load.

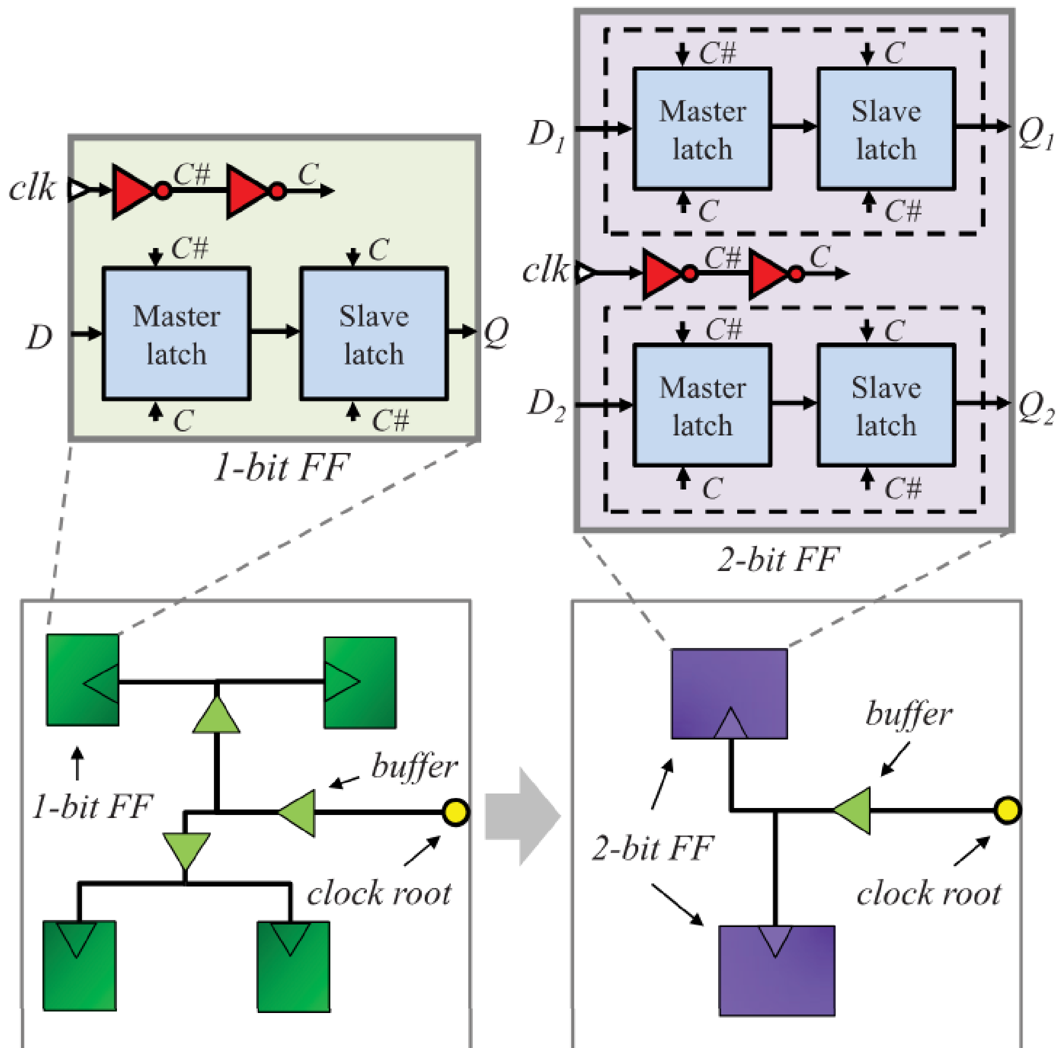


Figure 3.3: Replacing Single-Bit with Multi-Bit Flip Flops

# Chapter 4

# NeuroEvolution of Augmenting Topologies (NEAT) Algorithm

NeuroEvolution of Augmenting Topologies (NEAT) is an evolutionary neural network algorithm to create artificial networks. NEAT is first proposed in [20] which suggests NEAT's success is due to its crossover of different topologies, specification, and incremental growth, more on these points later. NEAT was selected due to its adaptability, incredible versatility, and speed for solving deterministic environments [19]. Further, [24]offers an excellent open-source python module for neat as well as a website that compiles and discusses the majority of the module's functionality. This module and the training environment developed for this paper are the cornerstones of the work presented here.

## 4.1 Brief NEAT Overview

Traditional neural network topologies rely on a developer to set the size and structure of the network while the training is solely responsible for determining the weights of the node's

connections. This methodology can be incredibility effective for well known problem spaces, however, evolutionary neural networks offer much greater versatility and often converge on optimal solutions faster than fixed networks [17]. As mentioned above, NEAT specializes in adaptation and incremental growth. That is to say, this method proposes the best way to develop an evolving artificial neural network with genetic algorithms is to start small and allow the network to grow and learn during its training. This idea is emphasized and made readily available by the python module developed by [24].

## 4.2 Configuration

A configuration file is used for every new run of NEAT, it contains all the required information for the source code to generate and train an Artificial Neural Network (ANN). Most notably, the configure file houses details for the structure of the network and its evolution; mutation, stagnation, reproduction, etc. Below, each section of the configure file is briefly explained and each design choice is discussed. Please refer to [25] for the complete overview of all configure file data and a user friendly overview of [24] as a whole. Note that all Parameter Overviews are paraphrased or direct quotes from [25].

### 4.2.1 NEAT

There are a few mandatory parameters for every network, including the population size, fitness threshold, fitness function, and whether or not to reset on extinction. Table 4.1 shows the parameters set for this section. The most important of which are the fitness_threshold and the pop_size. The threshold was set unachievable high, as the goal of this work is to produce the best ANN possible with no upper limit. The pop_size was set such that each generation would have many attempts to generate the best genome, while still completing simulations in a

reasonable amount of time; at a population size of 50 each generation takes approximately one hour to train.

Table 4.1: NEAT Configure Data

| Parameter | Data | Overview |
|---|---|---|
| fitness_criterion | max | How a species' success will be judged |
| fitness_threshold | 1000.0 | In this case, the minimum required fitness for a genome to pass |
| pop_size | 50 | The number of individuals per generation |
| reset_on_extinction | False | Whether or not the network will reset with a full set of new individuals in the case where the population completely extincts |

### 4.2.2 Stagnation

The stagnation parameters, shown in Table 4.2, determine when, if ever, a genome will be declared stagnant and removed from the gene pool. Due to the relative complexity of CTS and time constraints, the network was set to be fairly strict with species to encourage more development of those showing improvements, even if improvements are only slight. As such, the max_stagnation was set to only five generations to quickly remove poor species. To balance this, species_elitism is set to two to guarantee that at least the best two species will persist, avoiding total extinction.

Table 4.2: Stagnation Configure Data

| Parameter | Data | Overview |
|---|---|---|
| Species_fitness_func | max | How a species success will be judged |
| max_stagnation | 5 | The maximum number of generations any species can show no improvement before being declared stagnant and removed |
| species_elitism | 2 | The minimum number of species that will be protected from stagnation |

### 4.2.3 Reproduction and Species Set

Reproduction and Species Set are separate sections of the configuration file, but have been grouped together to since both are quite small. Both sections are listed in Table 4.3. Reproduction determines when and how genomes will combine to generate new genomes for the next generation. Elitism is set to five, ten-percent of the population, to encourage most species to evolve, while still maintaining a significant portion of the best individuals as-is. Survival_threshold is, similarly, set low to allow only the most successful individuals to reproduce.

Species set is used to determine species. Individuals in a species are pitted against one another such that only the best member from each species will be carried on. The compatibility_threshold is left as default.

Table 4.3: Reproduction and Species Set Configure Data

| Parameter | Data | Overview |
|---|---|---|
| elitism | 5 | The number of individuals of a species that will be preserved from one generation to the next |
| survival_threshold | 0.1 | The fraction of each species allowed to reproduce each generation |
| compatibility_threshold | 3.0 | The minimum genetic distance between individuals in different species |

### 4.2.4 Default Genomes

Characteristics for each default genome including the number of inputs, outputs, hidden nodes, the mutation probability and more. Effectively, this section specifies how default genomes will be shaped and behave.

#### 4.2.4.1 Node Activation

The activation settings for newly created nodes, Table 4.4, specify the output type and mutation rates. For this network, all nodes are forced to use sigmoid functions, which generate outputs between (0, 1), since all valid outputs are positive numbers and so the outputs could be linearly scaled to include all valid values.

Table 4.4: Node Activation Configure Data

| Parameter | Data | Overview |
|---|---|---|
| activation_default | sigmoid | The function that, effectively, dictates the range of the outputs of any one node |
| activation_mutate_rate | 0.0 | The probability that node could mutate to use an activation function other than that supplied by activation_default |
| activation_options | sigmoid | A list of alternative activation functions a node could mutate to use |

#### 4.2.4.2 Node Aggregation

The aggregation of a neural network is the method in which many weights and inputs are combined to a single value. For this network, simple summation was selected as its the most straightforward method which incorporates all weights and inputs in the final value while still giving a high precision to the output given the wide variability, and high number, of inputs. Shown in Table 4.5.

Table 4.5: Node Aggregation Configure Data

| Parameter | Data | Overview |
|---|---|---|
| aggregation_default | sum | The method in which all the weights and inputs of a node are combined to a single value |
| aggregation_mutate_rate | 0.0 | The probability that a node could mutate to use an aggregation method other than that supplied by aggregation_default |
| aggregation_options | sum | A list of alternative aggregation functions a node could mutate to use |

#### 4.2.4.3 Node Bias

The bias term supplies a high amount of randomness to the outputs of the network, allowing for more versatility and exploration. All values for node bias a left as default to help balance the exploration versus exploitation caused by node bias as shown in Table 4.6.

Table 4.6: Node Bias Configure Data

| Parameter | Data | Overview |
| --- | --- | --- |
| bias_init_mean | 0.0 | The average value of the bias |
| bias_init_stdev | 1.0 | The standard deviation of the bias |
| bias_max_value | 30.0 | The maximum value of the bias |
| bias_min_value | -30.0 | The minimum value of the bias |
| bias_mutate_power | 0.5 | The standard deviation of the zero-centered normal distribution from which a bias value mutation is drawn |
| bias_mutate_rate | 0.7 | The probability that a node's bias could mutate by adding a random value to it |
| bias_replace_rate | 0.1 | The probability that a node's bias could be replaced with a completely new value |

#### 4.2.4.4 Genome Compatibility

The genome compatibility configure data aids with separating genomes into species; as mentioned earlier, species are used to organize competition to more quickly converge on an optimal genome. Similar to the bias terms, genome comparability parameters are kept default, Table 4.7, to avoid over complicating the networks growth.

Table 4.7: Genome Compatibility Configure Data

| Parameter | Data | Overview |
| --- | --- | --- |
| compatibility_disjoint_coefficient | 1.0 | The coefficient for the disjoint and excess gene count's contribution to the genetic distance |
| compatibility_weight_coefficient | 0.5 | The coefficient for the for the difference, between any two genomes, of each comparability term's contribution to the genetic distance |

#### 4.2.4.5 Connection and Mutation

The connections and mutation rates, Table 4.8, determine the shape of the neural network, specifically, where nodes are placed and how they're connected. All of these values are left default, similar to node bias, to achieve a balance of exploration versus exploitation.

Table 4.8: Connection and Mutation Configure Data

| Parameter | Data | Overview |
| --- | --- | --- |
| conn_add_prob | 0.5 | The probability that a mutation will connect two previously unconnected nodes |
| conn_delete_prob | 0.5 | The probability that a mutation will disconnect two previously connected nodes |
| enabled_default | True | Whether or not a node is enabled by default |
| enabled_mutate_rate | 0.01 | The probability that a node will change its enabled status |
| node_add_prob | 0.2 | The probability that a new node in place of a connection |
| node_delete_prob | 0.2 | The probability that a node, and all its connections, will be deleted |

#### 4.2.4.6 Feed Forward

The Feed Forward section of the configuration file, Table 4.9, gives critical information regarding the shape and functionality of the generated networks. NEAT, and NEAT-Python, support recurrent and non-recurrent (i.e. feed-forward) networks; for this work, all networks are forced to be non-recurrent, meaning they have no feedback connections and therefore no "memory," this was selected to help balance the complexity of the networks and decrease training time. It is likely, however, that this decision will also decrease the overall effectiveness of the network, so this parameter may need to be scrutinized more closely in future experiments.

Further, the initial_connection parameter in this section determines the default connectivity

of newly generated genomes. For this work, this is set to full_non-direct, meaning all new genomes will have a direct connection between all input and all hidden nodes, all hidden nodes are connected to all outputs, and all input nodes are never directly connected to outputs. The full connectivity between input and hidden nodes is to, effectively, cover all the bases, allowing the genome to determine for itself which connections are most valuable. Similarly, the disconnected inputs to outputs, forcing all paths through the hidden network layers, is to account for the known high complexity of CTS. Lastly, this fairly extreme connectivity status is well balanced by the high conn_add/delete_probs discussed in the previous section.

Table 4.9: Feed Forward Configure Data

| Parameter | Data | Overview |
|---|---|---|
| feed_forward | True | Whether or not generated networks are forced to be non-recurrent |
| initial_connection | full_nodirect | Specifies the initial connectivity of newly generated genomes |

#### 4.2.4.7 Network Parameters

The Network Parameters define the number of inputs, outputs, and hidden nodes of the network; Table 4.10. Because NEAT is an evolutionary network, these parameters can be set to any value. For this project, there are eleven input data and ten output parameters, discussed in chapter 5. Two hidden nodes are used to accommodate for the vast complexity of CTS and high number of inputs and outputs, while attempting to avoid over-fitting the problem. It is possible, however, that the number of hidden nodes should have been set differently; while CTS is complex and there are many inputs and outputs, its possible that the relations between the inputs and outputs are more simple than originally assumed, meaning two hidden nodes will merely

over complicate the networks and increasing training time. That being said, passing eleven inputs through a single hidden node, or none at all, may result in a sever lack of interconnections between inputs, causing networks to quickly stagnate. For these reasons, two hidden nodes were chosen as a middle-ground, though future work may benefit from more closely investigating the optimal number of hidden nodes.

Table 4.10: Network Parameters Configure Data

| **Parameter** | **Data** | **Overview** |
|---|---|---|
| num_hidden | 2 | The number of hidden nodes to add to each genome in the initial population |
| num_inputs | 11 | The number of input nodes |
| num_outputs | 10 | The number of output nodes |

#### 4.2.4.8 Node Response

The response configuration data, Table 4.11, determine the attributes of a node. The response serves as a multiplying scalar, prior to the bias addition scalar, to the aggregation of the inputs. They are left as default, off, for this work as the outputs are manually scaled in the training environment, the bias scalar is in use, and keeping the response scalars disabled slightly reduces the network's training complexity.

Table 4.11: Node Response Configure Data

| Parameter | Data | Overview |
| --- | --- | --- |
| response_init_mean | 1.0 | The mean value of the response multiplier |
| response_init_stdev | 0.0 | The standard deviation of the response multiplier |
| reponse_max_value | 30.0 | The maximum value of the response multiplier |
| response_min_value | -30.0 | The minimum value of the response multiplier |
| response_mutate_power | 0.0 | The standard deviation of the zero-centered normal distribution from which a response multiplier mutation is drawn |
| response_mutate_rate | 0.0 | The probability that a node's response multiplier could mutate by adding a random value to it |
| response_replace_rate | 0.0 | The probability that a node's response multiplier could be replaced with a completely new value |

#### 4.2.4.9 Connection Weight

The Connection Weight configure data, Table 4.12, represents the strength of the connection between any two nodes. That is, when a node, input or hidden, passes a value to another, the value is multiplied by the weight of the two node's connection. For this work, the weights are left as default to, again, achieve a fair balance of exploration versus exploitation.

Table 4.12: Connection Weight Configure Data

| **Parameter** | **Data** | **Overview** |
|---|---|---|
| weight_init_mean | 0.0 | The mean value of the weight value |
| weight_init_stdev | 1.0 | The standard deviation of the weight value |
| weight_max_value | 30 | The maximum value of the weight value |
| weight_min_value | -30 | The minimum value of the weight value |
| weight_mutate_power | 0.5 | The standard deviation of the zero-centered normal distribution from which a weight value mutation is drawn |
| weight_mutate_rate | 0.8 | The probability that a connection's weight value could mutate by adding a random value to it |
| weight_replace_rate | 0.1 | The probability that a connection's weight value could be replaced with a completely new value |

# Chapter 5

# Training Environment

To train an evolutionary network to aid with clock tree synthesis, a synthesis tool and an arbitrary, fairly complex, digital design are required. Originally, Candace Design System's Innovus was to be the tool in question, however, considering the sheer quantity of floor plans and routes that would need to occur to train the network, the Synopsys IC Compiler (ICC) is used instead due to its relatively quiet processing, quick turn around times and readily available integration with the Synopsys Custom Compiler. A tcl launch file is used to quickly start the tool and report the routers results, see appendix II for the complete tcl file. The remainder of this chapter will discuss the design used to train the neural network and the python training environment that did so.

## 5.1 The Design

The objective of the neural network is that it can be applied to relatively large designs and quickly supply optimal CTS parameters for said design, as such, a relatively large design is required to train the network on. Fortunately, Mr. Mark A. Indovina has access to his design

of the Results Character Conversion (RCC) Block created for a Dual Tone Multi-Frequency (DTFM) Receiver. Below are the relevant characteristics of this RCC when routed using default Synopsis ICC settings and a square floor plan post-route:

- Number Sinks: 157
- Max Global Skew: 0.00870 sec
- Number of Nets: 1598
- Number of Cells: 1414
- Buf/Inv area: 261.77 $\mu m^2$
- Total Cell Area: 5737.05 $\mu m^2$
- Total Dynamic Power: 21.51 $\mu W$
- Cell Leakage Power: 190.50 $\mu W$

Figure 5.1 shows the routed design with the clock tree highlighted in yellow.

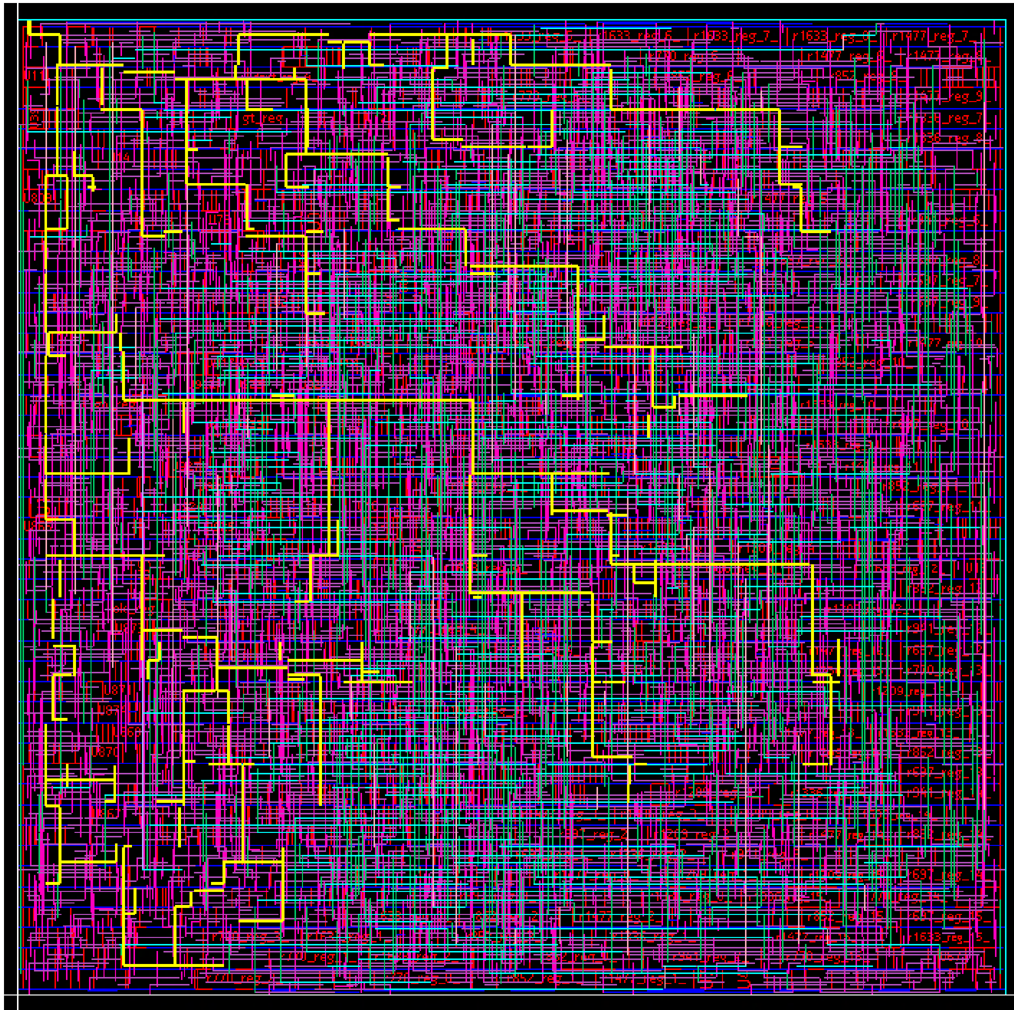

Figure 5.1: Results Character Conversion Block with Default Settings (Clock Tree Highlighted in Yellow)

Training the network on a single, rigid design would result in nearly perfect suggested parameters for CTS, but only for this one design. To make the network more versatile, and therefore usable on more than just this single design, the RCC's floor plan is randomized for each generation of the network. This way, from the network's perspective, every design it is working with is different, as changing the floor plan can randomly alter the design's Max Global Skew, Buf/Inv area, and many other characteristics that are used as inputs for the network.

Figure 5.2 shows the same RCC design using a pseudo-random 'L' shaped floor plan instead of a square one. Once again, the clock tree is highlighted in yellow. Giving the following altered characteristics:

- Number Sinks: 157
- Max Global Skew: 0.01782 sec
- Number of Nets: 1618
- Number of Cells: 1434
- Buf/Inv area: 285.91 $\mu m^2$
- Total Cell Area: 5518.74 $\mu m^2$
- Total Dynamic Power: 21.24 $\mu W$
- Cell Leakage Power: 262.91 $\mu W$

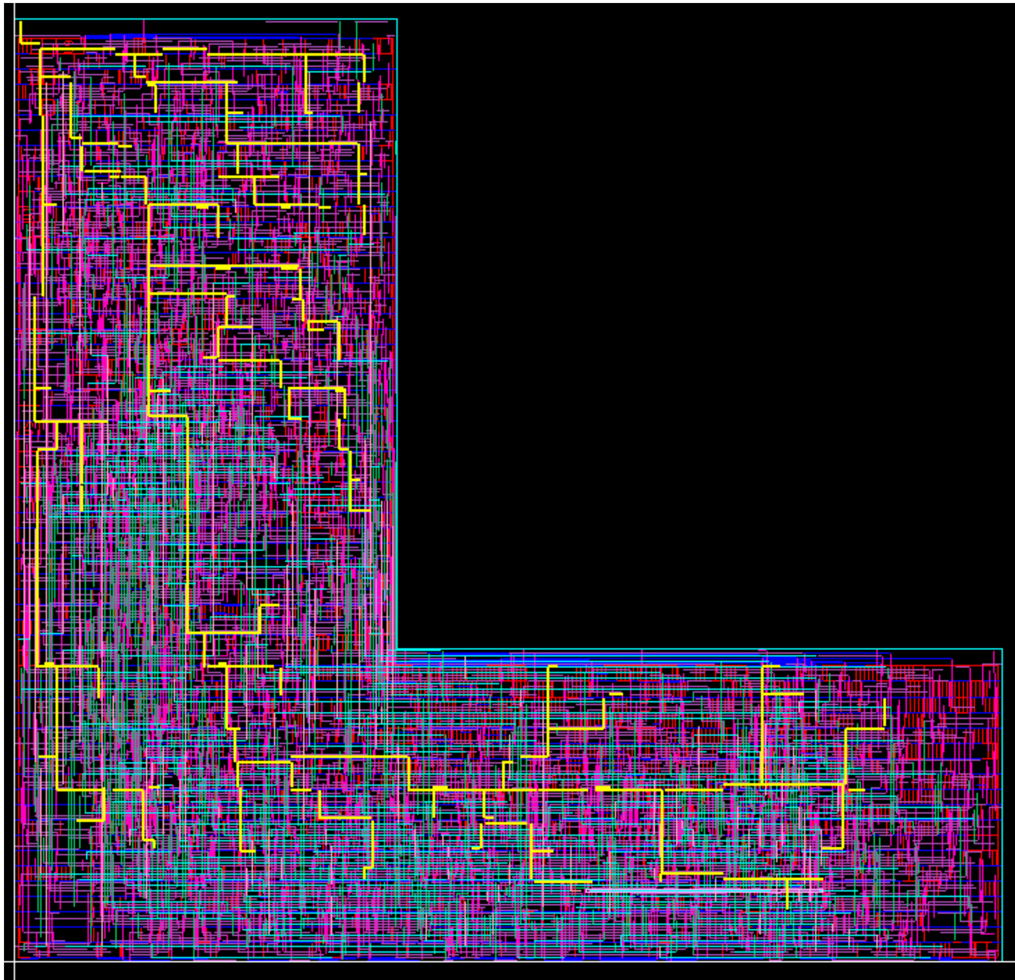

Figure 5.2: 'L' Shaped RCC Block

Similarly, figure 5.3 uses an 'X' shaped floor plan, resulting in:

- Number Sinks: 157
- Max Global Skew: 0.02678 sec
- Number of Nets: 1617
- Number of Cells: 1433
- Buf/Inv area: 317.17 $\mu m^2$

- Total Cell Area: 5561.69 $\mu m^2$
- Total Dynamic Power: 22.85 $\mu W$
- Cell Leakage Power: 431.92 $\mu W$

Figure 5.3: 'X' Shaped RCC Block

## 5.2 Trainer

As discussed in chapter 4, NEAT-python is used to create and train the neural network. However, the training environment needs to supply the network the Synopsys ICC's results in order to gauge the success of the networks proposed CTS parameters. Further, as discussed above, the environment must randomize the designs floor plan before routing it to give each genome a new challenge to learn from.

### 5.2.1 Network Inputs

To be trained, the Neural Network needs a reliable representation of the design its meant to optimize to serve as the networks inputs. For this work, eleven design characteristics are chosen based on their ease of access and nearly complete overview of the relevant parts of the design. They are listed in table 5.1 below along with a brief overview of their relevance.

### 5.2.2 Network Outputs

Synopsys ICC supports a wide range of CTS optimization parameters, many of which are simple Boolean inputs such as "buffer_sizing" and "advanced_drc_fixing." The nature of neural networks using a sigmoid activation function lends itself better to a range of possibilities rather than Boolean expressions, as such, it was decided that this network would be responsible for all non-Boolean parameters. This way, the network can experiment with a wide range of control over the final design, while still keeping the number of outputs to a manageable ten, see table 5.2 for all ten parameters the network can control.

The overview of each parameter is a summary of its functionality, please see appendix III for the complete description of all CTS parameters and their effect on the final design as explained by [6].

Table 5.1: Design Characteristics used for Network Inputs

| **Characteristic** | **Overview** |
| --- | --- |
| Number of Sinks | The total number of terminating nets |
| Number of Nets | The total number of nets, connections, within the design |
| Number of Cells | The total number of cells, combinational logic, filler or otherwise, used in the design |
| Combinational Area | The total area occupied by the combinational logic cells |
| Buf/Inv Area | The total area occupied by the buffers and inverters, not necessarily used for CTS |
| Total Cell Area | The total area occupied by all design cells |
| Max Global Skew | The maximum global skew of the design |
| Cell Internal Power | The total power required to switch the inputs of the design, but not the outputs |
| Net Switching Power | The total power required to switch the inputs and output of the design |
| Total Dynamic Power | The sum of the leakage and dynamic powers |
| Cell Leakage Power | The total leakage power from all cells |

Table 5.2: Non-Boolean CTS Parameters

| **Expression** | **Range** | **Overview** |
|---|---|---|
| target skew | 0-1 s | The maximum skew any specified clock tree can have throughout the design |
| target early delay | 0-1 s | The minimum insertion delay the longest path of any specified clock tree must have |
| leaf max transition | 0-1 s | The maximum allowable transition time the buffers and inverters used for leaf nets may have |
| max transition | 0-1 s | The maximum allowable transition time the buffers and inverters used, anywhere except for leaf nets, may have |
| max capacitance | 0-1200 nf | The maximum capacitance any specified clock tree may have |
| max fanout | 0-4000 | The maximum fanout any specified clock tree may have |
| layer list for sinks start | 0-8 | The lowest metal layer the CTS router can use for clock sinks |
| layer list for sinks length | 2-10 | The number of metal layers above the layer list for sinks start the CTS router can use for clock sinks |
| layer list start | 0-8 | The lowest metal layer the CTS router can use for clock signals |
| layer list length | 2-10 | The number of metal layers above the layer list for sinks start the CTS router can use for clock signals |

### 5.2.3 Network Fitness Function

To judge the success of the network, each genome's outputs are used to reroute the design using the same floor plan as from when they were gathered. This way, the design characteristics discussed in section 5.2.1 can be regathered and compared to those gathered using default CTS parameters. See below for the python code which compares the characteristics, scales them for the fitness function's bias, and sums them. Note that genome.fitness is set to '50' prior to this subtraction; the "resultsData" python dictionary holds the designs characteristics using the default CTS prameters while the "optimalResultsData" python dictionary holds the characteristics of the design using the parameters suggested by the neural network.

Recall that the intention of this network is to provide optimized CTS parameters, focusing on the design's maximum global skew, power consumption and resource usage, in that order. Therefore, the fitness function is scaled to represent these priorities.

```
1  diff = 0
2  diff += 15*(optimalResultsData["Max Global Skew"] -
       resultsData["Max Global Skew"])
3  diff += 3*(optimalResultsData["Cell Internal Power"] -
       resultsData["Cell Internal Power"])
4  diff += 3*(optimalResultsData["Net Switching Power"] -
       resultsData["Net Switching Power"])
5  diff += 3*(optimalResultsData["Total Dynamic Power"] -
       resultsData["Total Dynamic Power"])
6  diff += 3*(optimalResultsData["Cell Leakage Power"] -
       resultsData["Cell Leakage Power"])
7  diff += 1*(optimalResultsData["Combinational area"] -
```

```
    resultsData["Combinational area"])
8 diff += 1*(optimalResultsData["Buf/Inv area"] - resultsData["
    Buf/Inv area"])
9
10 genome.fitness -= diff
```

Further, to more quickly train the network to output valid values, there is a check to ensure the leaf_max_transition is equal to or less than the max_transition parameter. Similarly, to avoid erroneous and unreliable routing, note that the in table 5.2, the layer list and layer list for sinks parameters must be at least two metal layers long and must start below the eighth metal layer. If any of this conditions are not met, the genome's fitness is immediately set to zero and it does not run the router, the next genome is started immediately.

```
1 if (predictedParams["leaf_max_transition"] > predictedParams["
    max_transition"]) or
2 (predictedParams["layer_list_start"] >= 9) or
3 (predictedParams["layer_list_for_sinks_start"] >= 9) or
4 (predictedParams["layer_list_start"] < 2) or
5 (predictedParams["layer_list_for_sinks_leng"] < 2):
6   genome.fitness -= 50
7   break
```

### 5.2.4 Randomizing Floor Plans

Synopsys ICC offers a wide range of floor plan versatility, allowing users to develop their design into, almost, any shape. See appendix III for the complete list and explanations of each floor plan options used. Considering only one design is used to train the neural network, the

versatile floor plans are used to emulate multiple designs. Every ten generations of the neural network is given a new randomized floor plan, ranging from the ’L’ and ’X’ shapes listed in figures 5.2 and 5.3, to ’U’ and ’T’ shapes, all with randomized side lengths, resulting in ten total “designs,” each generation of ten populations getting a different one. See figure 5.4 for how the shapes are sized. When randomized, every side length, ’a-d’ for ’L’ and ’a-f’ otherwise, are set between 10-40 units with a core utilization of 80% and an io2core distance of 0.5 units on all sides. Using this method, every generation will receive a new design, at least from the perspective of the neural network as all the network’s inputs will be scrambled since the routed designs can be vastly different.

It was decided that every ten generations should be given a new floor plan, instead of one per generation or one per one hundred generations, because, this way, the network has ten opportunities to grow and learn from each floor plan before being given another one. However, the network will still be made versatile since it is trained on a variety of floor plans rather than just one. In other words, every member of every ten generations will receive identical inputs and each will have the opportunity to suggest optimal CTS parameters.

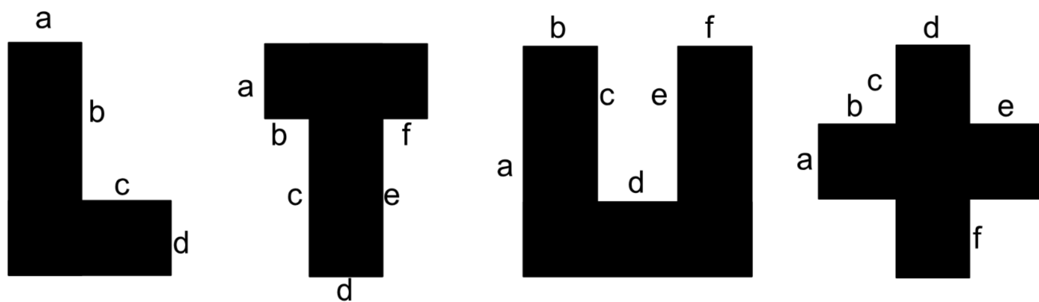


Figure 5.4: Floor Plan Structures

### 5.2.5 Data Reporting and Collection

Synopsys ICC further supplies excellent report generation options. Every genome, that proposes valid CTS parameters, is routed; from the design, the post-route clock_tree, clock_tree_options, timing, area, and power are all reported. These reports are used to gather a series of characteristics, table 5.1, used to compare the default parameters to the proposed clock tree parameter results.

# Chapter 6

# Results

Throughout the networks training processes statistics on its performance were being captured constantly. This chapter will display and discuss these statistics, including what worked well, what didn't, and suggestions for how to improve future attempts at developing an evolutionary neural network for optimized clock tree synthesis. Further, this chapter will discuss the final and best genome's ability to optimize clock trees by applying it to ten variations of the design it has been trained on.

## 6.1 Final Neural Network

Shown below are the graphs of the networks performance throughout its training, accompanied by a short discussion of the graphs.

Also, note that, though not well captured in the reports generated by the neural network training environment, it is believed that trial six resulted in the extinction of a significant portion of the population due to stagnation, leaving generation seven (populations 70-80) with a large percentage of newly generated genomes, which would explain the fairly low fitness, low

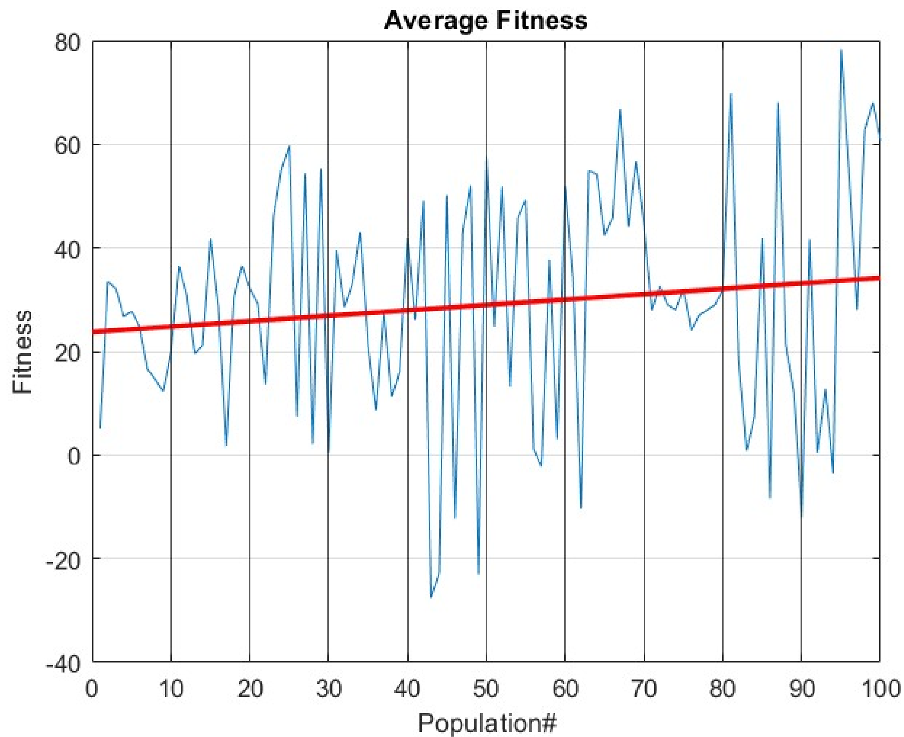


Figure 6.1: Average Fitness

standard deviation, and low generation times for this generation. Fortunately, the configuration of this network guarantees that at least the best two genomes will be protected from stagnation, see Table 4.2, meaning the final genome used for analysis has been present in all ten generations.

Figures 6.1 and 6.2 below give the average and best fitness, respectively, of each of the ten populations of each of the ten generations, one hundred members in total. These graphs give the best indication to the success of the network, specifically the line of best fit. From this line it can be seen that the average and best fitness of each population is trending upwards, though very slowly, suggesting that given enough training time, the final network will provide more and more optimal CTS parameters for a given design.

Figure 6.3 gives the standard deviation of the fitness of each of a populations members, indicating to the exploration of the network; in other words, how well is the network converging

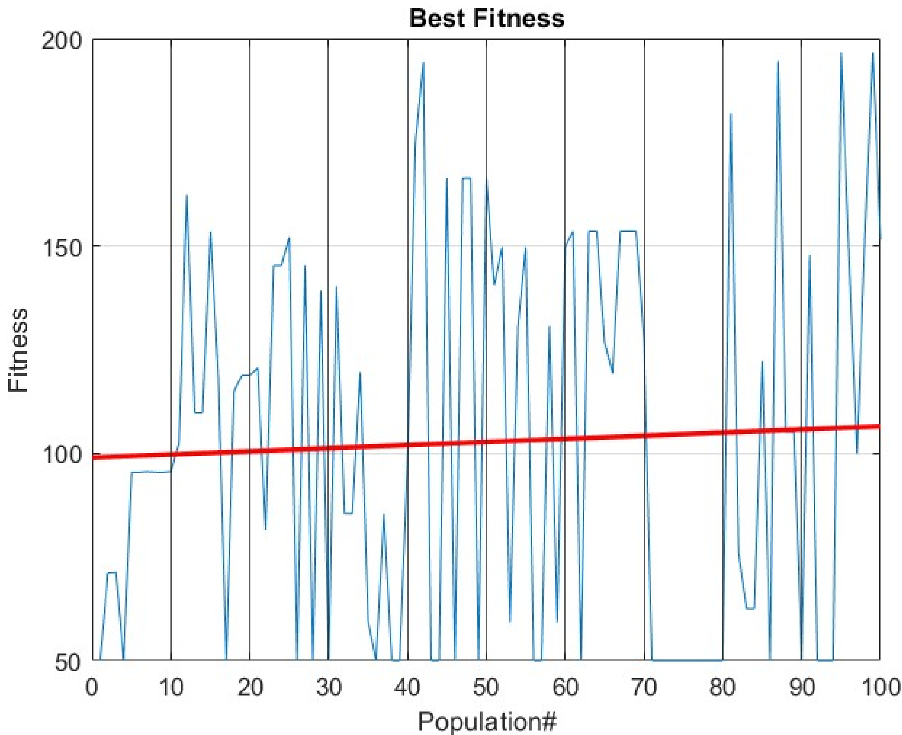


Figure 6.2: Best Fitness

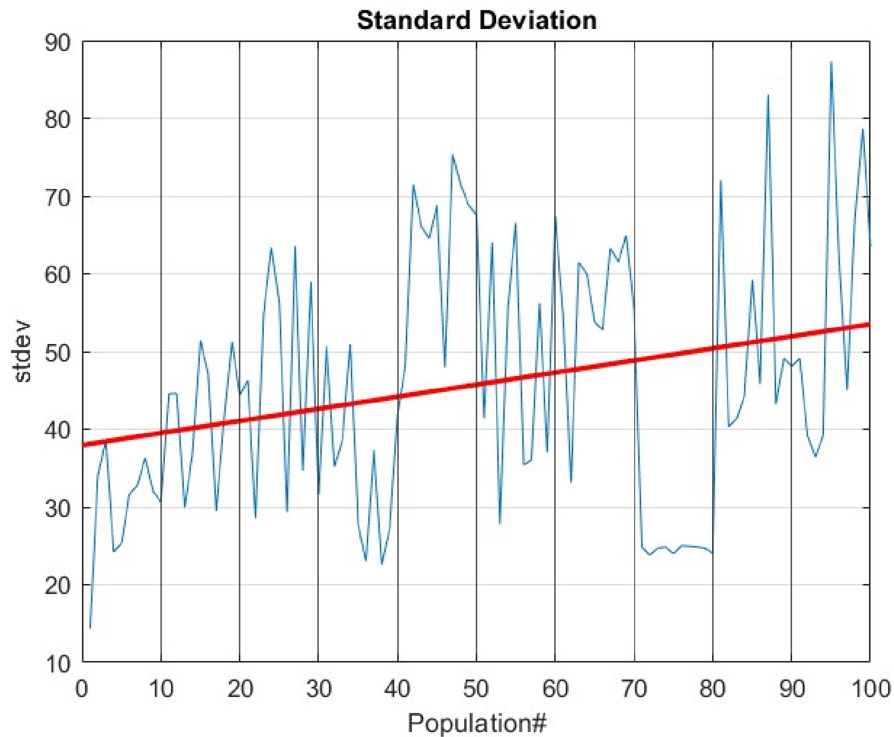


Figure 6.3: Standard Deviation

on a single solution. Generally, the standard deviation of a successful network would go down, as the network begins focusing down to one optimal solution. For this work, however, the standard deviation rises rapidly throughout training.

For completion, Figures 6.4 and 6.5 are included to show the best fitness of each generation, and the generation time of each population, respectively. Recall that generation seven's population consisted of a large number of newly generated genomes, hence the sudden drop in fitness and generation time at genome eight and nine. Ignoring this drop, it can be seen that the best genome's fitness tends to rise, slowly, during training.

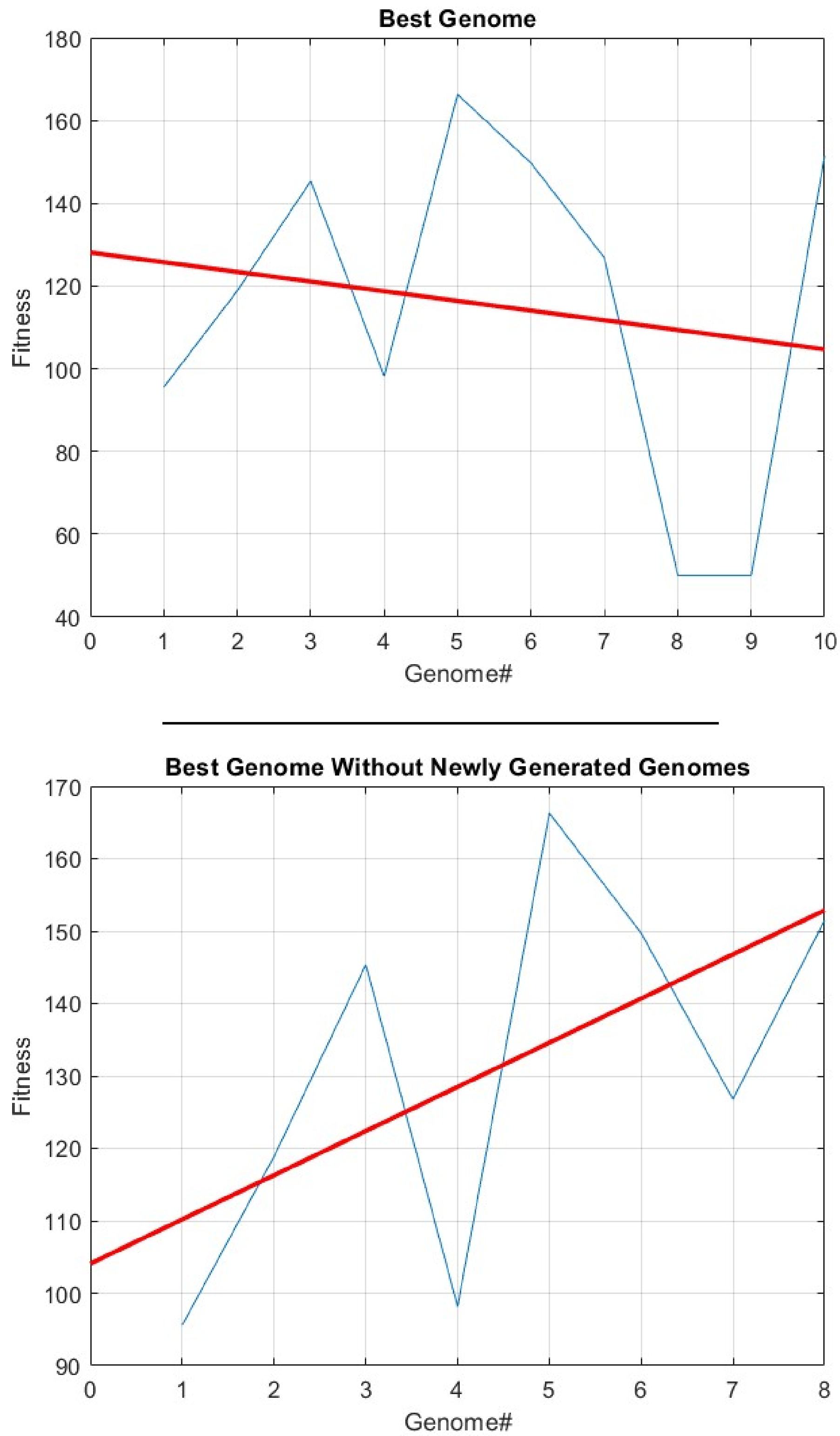


Figure 6.4: Best Genome (top), Best Genome Without Newly Generated Genomes (bottom)

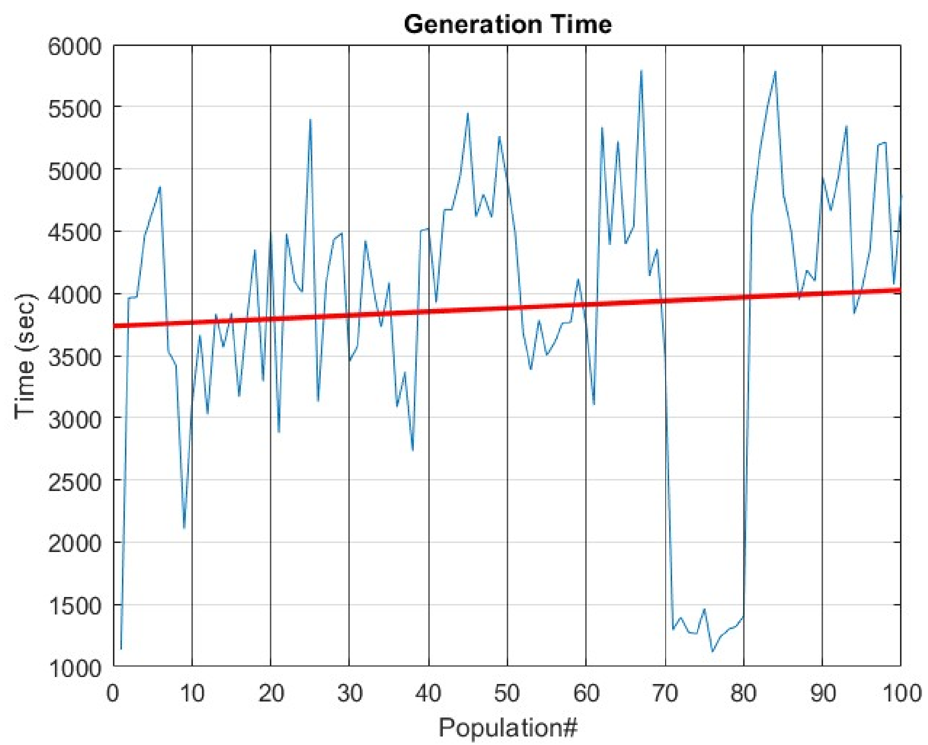


Figure 6.5: Generation Time

## 6.2 Analysis

Below, Figure 6.6, shows the layout of the final and most optimal neural network generated. For reference Figure 6.7 shows the layout of an untrained network using the same configuration: eleven inputs, ten outputs, and two hidden nodes. Obviously, the training process dramatically affected the shape and connectivity of the network; in fact, seven of the starting eleven inputs and five of the original ten outputs were disconnected completely, meaning the network, from its training, deemed these inputs and outputs irrelevant. While a good designer would highly disagree, Figures 6.1 and 6.2 suggest this method did in fact work, though only slightly.

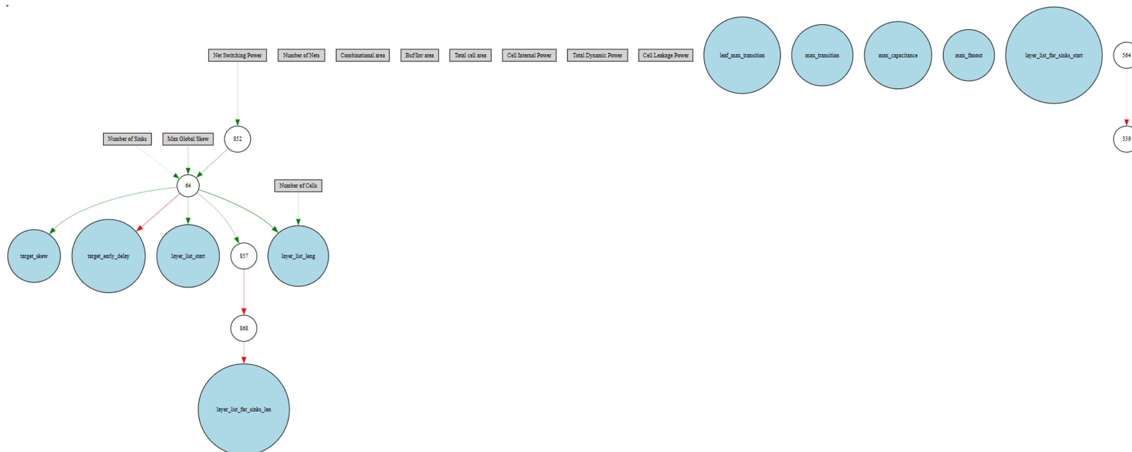


Figure 6.6: Final Genome Layout

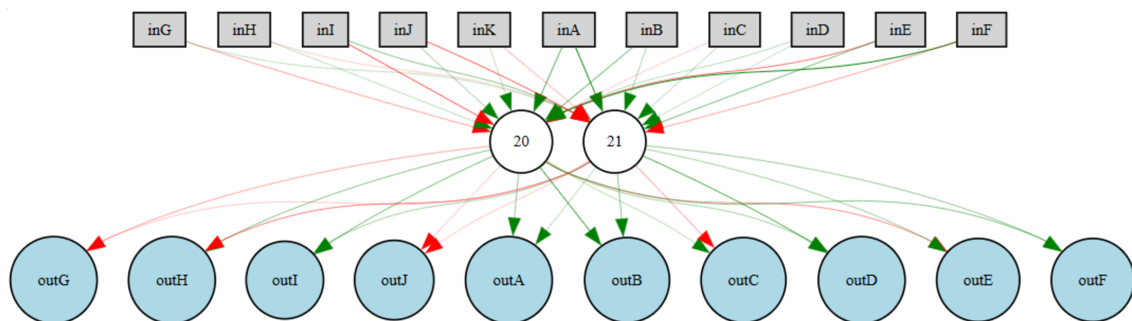


Figure 6.7: Starting Genome Layout

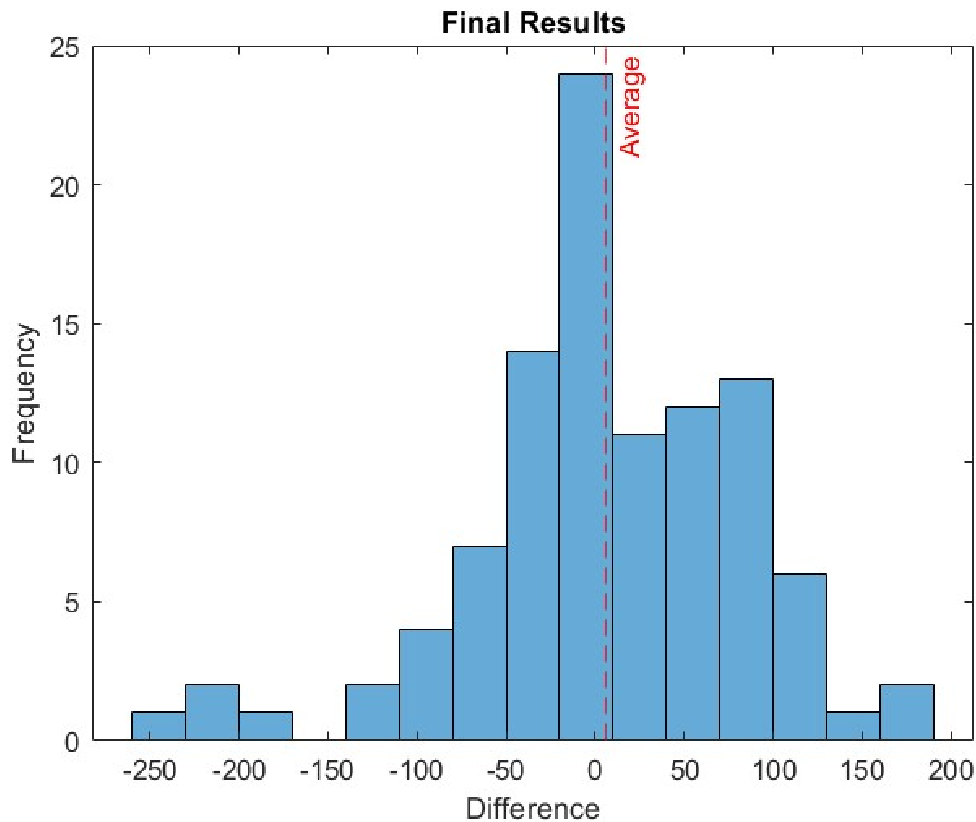


Figure 6.8: Final Results Histogram. Average = +6.4720

To evaluate the effectiveness of the network, a series of one hundred designs using randomized floor plans are fed into the final neural network. Each floor plan is first routed using Synopsys default CTS parameters, and again using the networks recommended parameters. The results are compared using the same fitness function as defined in section 5.2.3 to give each floor plans "difference" value, where a positive difference is an improvement over the default parameters, compiled and shown in Figure 6.8.

## 6.3 Discussion

From Figure 6.8, it can be seen that, unfortunately, the final network was hardly successful. An average improvement of only 6.472 is not enough to mark the network as better than the default settings. However, given the rising average fitness and best fitness discussed in section 6.1, it is likely that given enough time, the network would continue to improve.

That being said, knowing these results, there are some clear improvements that can be made. First, using a single design is most likely the greatest bottleneck to the networks success. While routing the design using randomized floor plans can emulate a variety of designs, the number of sinks will always be the same, and the other characteristics can only shift by so much. The network would likely benefit heavily from being trained using multiple, large designs.

Further, considering the networks immediate but slow growth and dramatic topology change, the configuration file is likely too strict. It seems that the network was too quickly shoe horned into a single, sub-optimal solution rather than exploring the complexities of CTS to find better, more complex solutions.

Lastly, the fitness function developed and discussed in Section 5.2.3 likely weights the skew too lightly. As shown, only the max global skew characteristic is used to represent the skew, while there are six characteristics used to represent the size of the design, and four used for the power consumption. While the fitness function would seem to weight the max global skew by the most, as its scalar is the largest, one serious flaw with the function is that the data is never normalized. That is, the difference in the max global skew can be in the hundredths of a decimal, while the difference in area can be in the tens, making the scalar value almost useless. This issue could be easily avoided by first dividing both the default and "optimized" characteristics by the default, normalizing both before comparison. However, since the final results shown in Figure 6.8 are gathered using the same fitness function, the fact that the characteristics are not

normalized does not invalidate this data; normalized or not, the network still gives minimal improvements to CTS parameters.

# Chapter 7

# Conclusion

The evolutionary neural network developed for this work was intended to aid designers by suggesting optimal CTS parameters based on a set of characteristics of any given design. The primary motivation for this work was to gain a better understanding of CTS and to reduce the development time for complex digital designs. Research was conducted to form a solid understanding of CTS as well as neural networks, specifically the NEAT methodology for developing ANNs. The evolutionary neural network trained showed promising results based on its average fitness's growth throughout training. Unfortunately, testing the network via a series of randomized designs proved its minimal ability to suggest optimal CTS parameters, being only slightly more successful than the default CTS parameters posed by the Synopsys IC Compiler. More work will need to be conducted to uncover the exact source of the network's shortcomings to ensure another attempt will be more successful.

## 7.1 Future Work

The framework built here can be easily expanded upon for further research to develop a more capable neural network. As mentioned in section 6.3, the primary issues with the network is most likely the lack of variability in the deign used to train it; the randomized floor plan method is powerful for emulating multiple large designs, though it falls short of the variety of completely different designs. Similarly, the networks configuration is possible too strict, forcing the network into a sub-optimal solution too early in its training, hence the immediate but very slow improvements. Lastly, the fitness function used to judge the success of the network needs to be normalized so the network can properly prioritize skew, power consumption, and resource consumption.

# References


[1] A. B. Chong, "Hybrid multisource clock tree synthesis," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–6.

[2] P. V. Vishnu, A. R. Priyarenjini, and N. Kotha, "Clock tree synthesis techniques for optimal power and timing convergence in soc partitions," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 2019, pp. 276–280.

[3] G. M. Madhuri, J. Selvakumar, and K. S. Krishna, "Performance analysis on skew optimized clock tree synthesis," in *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, 2022, pp. 01–06.

[4] P. Ray, V. S. Prashant, and B. P. Rao, "Machine learning based parameter tuning for performance and power optimization of multisource clock tree synthesis," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022, pp. 1–2.

[5] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, "Gan-cts: A generative adversarial framework for clock tree prediction and optimization," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[6] Synopsys, *Synopsys IC Compiler Manual*, Synopsys, Mar. 2024. [Online].

Available: https://www.synopsys.com/support/licensing-installation-computeplatforms/synopsys-documentation.html

[7] W.-H. Chen, C.-K. Wang, H.-M. Chen, Y.-C. Chou, and C.-H. Tsai, “A comparative study on multisource clock network synthesis,” in *Proc. SASIMI*, 2016, pp. 1–5. [Online]. Available: https://scholar.google.com/scholar?q=A+Comparative+Study+on+Multisource+Clock+Network+Synthesis&hl=en&as_sdt=0&as_vis=1&oi=scholart

[8] H. Toyama, “What’s the difference between cts, multisource cts, and clock mesh?” Electronic Design, Mar. 2012. [Online]. Available: https://www.electronicdesign.com/news/products/article/21765665/whats-the-difference-between-cts-multisource-cts-and-clock-mesh

[9] AnySilicon, “Ultimate guide: Clock tree synthesis,” AnySilicon, Sep. 2022. [Online]. Available: https://anysilicon.com/clock-tree-synthesis/

[10] V. Srivatsa, A. P. Chavan, and D. Mourya, “Design of low power and high performance multi source h-tree clock distribution network,” in *2020 IEEE VLSI DEVICE CIRCUIT AND SYSTEM (VLSI DCS)*, 2020, pp. 468–473.

[11] M. P.-H. Lin, C.-C. Hsu, and Y.-C. Chen, “Clock-tree aware multibit flip-flop generation during placement for power optimization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 280–292, 2015.

[12] J. Lu, W.-K. Chow, and C.-W. Sham, “Fast power- and slew-aware gated clock tree synthesis,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 2094–2103, 2012.

[13] A. Chattopadhyay and Z. Zilic, “Flexible and reconfigurable mismatch-tolerant serial

clock distribution networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 3, pp. 523–536, 2012.

[14] K. Sharma, "Clock tree routing algorithms," VLSI- PHYSICAL DESIGN FOR FRESHERS, Apr. 2020. [Online]. Available: https://www.physicaldesign4u.com/2020/03/clock-tree-routing-algorithms.html

[15] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.

[16] A. Moujahid, M. ElAraki Tantaoui, M. D. Hina, A. Soukane, A. Ortalda, A. ElKhadimi, and A. Ramdane-Cherif, "Machine learning techniques in adas: A review," in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 2018, pp. 235–242.

[17] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/572107

[18] L. Cheng, Z.-G. Hou, Y. Lin, M. Tan, W. C. Zhang, and F.-X. Wu, "Recurrent neural network for non-smooth convex optimization problems with application to the identification of genetic regulatory networks," *IEEE Transactions on Neural Networks*, vol. 22, no. 5, pp. 714–726, 2011.

[19] M. E. Taylor, S. Whiteson, and P. Stone, "Comparing evolutionary and temporal difference methods in a reinforcement learning domain," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1321–1328. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1143997.1144202

[20] K. Stanley and R. Miikkulainen, “Efficient evolution of neural network topologies,” in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*, vol. 2, 2002, pp. 1757–1762 vol.2.

[21] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: https://ieeexplore.ieee.org/document/6790655

[22] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, “Fast and accurate estimation of quality of results in high-level synthesis with machine learning,” in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 129–132.

[23] I. Cadence Design Systems, *Virtuoso Digital Implementation (VDI)Discovery KitRapid Adoption Kit (RAK)*, Cadence Design Systems, Inc., Dec. 2023.

[24] A. McIntyre, M. Kallada, C. G. Miguel, C. Feher de Silva, and M. L. Netto, “neat-python.” [Online]. Available: https://neat-python.readthedocs.io/en/latest/index.html

[25] CodeReclaimers, *NEAT-Python Overview*, llc revision 63f4cf81 ed., readthedocs, 2019. [Online]. Available: https://neat-python.readthedocs.io/en/latest/neat_overview.html

# Appendix I

# Source Code

## I.1 Training Environment

```
1  from fileMan import *
2  import subprocess
3  import random
4  import time
5  from datetime import datetime
6
7  import neat
8  import pickle
9
10 #dummy()
11 ###    control variables    ###
12 configPath = "neat_config_strict"
13 enableRouting = True
```

```
14 newFloorplanPerGeneration = False
15 generateFloorplanForGeneration = True
16 detailedReporting = False
17 attemptsPerFloorplan = 1
18 minFloorplanWallSize = 10
19 maxFloorplanWallSize = 40
20 generationLimit = 10
21 checkpointsWhen = 2
22 reportFile = "neat_report_run4_9.txt"
23
24 startFromCheckpoint = True
25 checkpointPath = "run4_8_neat-checkpoint-81"
26
27 winnerPKL = "run4_9_best_genome.pkl"
28
29 ###    routing params        ###
30 defaultParams = {"target_skew" : 0.0, "target_early_delay" :
     0.0, "leaf_max_transition" : 0.5, "max_transition" : 0.5, "
     max_capacitance" : 600, "max_fanout" : 2000, "
     layer_list_for_sinks_start" : 0, "layer_list_for_sinks_leng
     " : 10, "layer_list_start" : 0, "layer_list_leng" : 10}
31 floorplanShapes = ["L", "T", "U", "X"]
32
33 ###    data retreavil params    ###
34 dataTag = ""
```

```
35 paramsTag = ""
36 data_required = ["Number of Sinks", "Max Global Skew", "Number
      of nets", "Number of cells", "Combinational area", "Buf/
     Inv area", "Total cell area", "Cell Internal Power", "Net
     Switching Power", "Total Dynamic Power", "Cell Leakage
     Power"]
37 params_required = ["target_skew", "target_early_delay", "
     leaf_max_transition", "max_transition", "max_capacitance",
     "max_fanout", "layer_list_for_sinks", "layer_list"]
38 predicted_params_required = ["target_skew", "
     target_early_delay", "leaf_max_transition", "max_transition
     ", "max_capacitance", "max_fanout", "
     layer_list_for_sinks_start", "layer_list_for_sinks_leng", "
     layer_list_start", "layer_list_leng"]
39
40 resultsData = {}
41 resultsParams = {}
42 resultsDataList = []
43 optimalResultsData = {}
44 optimalResultsParams = {}
45
46 ###   get source file to read and eventually overwrite    ###
47 file_src = "icc/cmp_icc_src.tcl"
48 file_dest = "icc/cmp_icc.tcl"
49 with open(file_src, 'r') as fh:
```

```
50      filedata = fh.read()
51    fileLines = filedata.split("\n")
52
53    def eval_genomes(genomes, config):
54      global generateFloorplanForGeneration, newFileDefault,
            newFileDefaultLines
55      if (generateFloorplanForGeneration):
56        generateFloorplanForGeneration = newFloorplanPerGeneration
57        ###   setup random floorplan, default CTS params    ###
58        ##  insert default cts params
59        newFileDefaultParamsData = update_params(fileLines,
              filedata, defaultParams)
60        newFileDefaultParamsDataLines = newFileDefaultParamsData.
              split("\n")
61        ##  insert psuedo random floorplan
62        random.seed(time.time())
63        data = {}
64        data["shape"] = random.choice(floorplanShapes)
65        a = str(random.randint(minFloorplanWallSize,
              maxFloorplanWallSize))
66        b = str(random.randint(minFloorplanWallSize,
              maxFloorplanWallSize))
67        c = str(random.randint(minFloorplanWallSize,
              maxFloorplanWallSize))
```

```
68        d = str(random.randint(minFloorplanWallSize,
             maxFloorplanWallSize))
69        e = str(random.randint(minFloorplanWallSize,
             maxFloorplanWallSize))
70        f = str(random.randint(minFloorplanWallSize,
             maxFloorplanWallSize))
71        if data["shape"] == "L":
72          data["dims"] = "{" + a + " " + b + " " + c + " " + d + "
               }"
73        else:
74          data["dims"] = "{" + a + " " + b + " " + c + " " + d + "
                " + e + " " + f + "}"
75
76        report_fh.write("\nCreating floorplan: " + data["shape"] +
              " " + data["dims"] + "\n")
77        newFileDefault = update_floorplan(
             newFileDefaultParamsDataLines, newFileDefaultParamsData
             , data)
78        newFileDefaultLines = newFileDefault.split("\n")
79        with open(file_dest, 'w') as fh:
80          fh.write(newFileDefault)
81
82        ###  route on random floorplan with default CTS params
             ###
83        if (enableRouting):
```

```
84        p = subprocess.Popen(['./icc.csh', '-n'])
85        p.wait()
86
87    ###   get default report data   ###
88    with open("report/icc/results_conv_saed32nm_icc_postRoute.
          rpt", 'r') as post_route_report_fh:
89      post_route_report = post_route_report_fh.read().split("\n"
          )
90
91    global resultsData
92    get_data(resultsData, data_required, post_route_report,
          dataTag)
93    get_data(resultsParams, params_required, post_route_report,
          paramsTag)
94    #print(resultsData)
95    #print("")
96    #print(resultsParams)
97    for x in resultsData:
98      resultsData[x] = float(resultsData[x])
99
100   ### format default report data for neural net   ###
101   global resultsDataList
102   resultsDataList = []
103   for x in data_required:
104     resultsDataList.append(resultsData[x])
```

```
105     #print(resultsDataList)
106
107     for genome_id, genome in genomes:
108       #report_fh.write("\nRunning genome_id " + str(genome_id) +
              " at " + datetime.now().strftime("%H:%M:%S"))
109       genome.fitness = 50.0
110       net = neat.nn.FeedForwardNetwork.create(genome, config)
111       for x in range(attemptsPerFloorplan):
112         ### give inputs to neural net, get predicted optimized
              CTS params  ###
113         output = net.activate(resultsDataList)
114         predictedParams = {}
115         for header, param in zip(predicted_params_required,
              output):
116           predictedParams[header] = round(param, 4)
117
118         ##["target_skew", "target_early_delay", "
              leaf_max_transition", "max_transition", "
              max_capacitance", "max_fanout", "
              layer_list_for_sinks_start",
119         ##"layer_list_for_sinks_leng", "layer_list_start", "
              layer_list_leng"]
120         ###   scale parameters for faster training time   ###
121         predictedParams["target_skew"] = predictedParams["
              target_skew"] * 1
```

```
122         predictedParams["target_early_delay"] = predictedParams[
                "target_early_delay"] * 1
123         predictedParams["leaf_max_transition"] = predictedParams
                ["leaf_max_transition"] * 1
124         predictedParams["max_transition"] = predictedParams["
                max_transition"] * 1
125         predictedParams["max_capacitance"] = int(predictedParams
                ["max_capacitance"] * 1200)
126         predictedParams["max_fanout"] = int(predictedParams["
                max_fanout"] * 4000)
127         predictedParams["layer_list_for_sinks_start"] = int(
                predictedParams["layer_list_for_sinks_start"] * 9)
128         predictedParams["layer_list_for_sinks_leng"] = int(
                predictedParams["layer_list_for_sinks_leng"] * 10)
129         predictedParams["layer_list_start"] = int(
                predictedParams["layer_list_start"] * 9)
130         predictedParams["layer_list_leng"] = int(predictedParams
                ["layer_list_leng"] * 10)
131         #print(predictedParams)
132
133         ###  check predicted values for validity  ###
134         if (predictedParams["leaf_max_transition"] >
                predictedParams["max_transition"]) or (
                predictedParams["layer_list_start"] >= 9) or (
                predictedParams["layer_list_for_sinks_start"] >= 9)
```

```
            or (predictedParams["layer_list_start"] < 2) or (
            predictedParams["layer_list_for_sinks_leng"] < 2):
135           genome.fitness -= 50
136           break
137
138         #report_fh.write("genome " + str(genome_id) + " passed
            checks at " + datetime.now().strftime("%H:%M:%S") ".
            Params:\n\t" + str(predictedParams) + "\n")
139         ###   if valid, setup predicted optimal CTS params
            ###
140         newFilePredictedParams = update_params(
            newFileDefaultLines, newFileDefault, predictedParams)
141         with open(file_dest, 'w') as fh:
142           fh.write(newFilePredictedParams)
143
144         ###   route predicted optimal clock   ###
145         if (enableRouting):
146           p = subprocess.Popen(['./icc.csh', '-n'])
147           p.wait()
148
149         ###   get predicted optimal report data
150         with open("report/icc/
            results_conv_saed32nm_icc_postRoute.rpt", 'r') as
            post_route_report_fh:
```

```
151             post_route_report = post_route_report_fh.read().split(
                    "\n")
152
153         global optimalResultsData
154         get_data(optimalResultsData, data_required,
                post_route_report, dataTag)
155         get_data(optimalResultsParams, params_required,
                post_route_report, paramsTag)
156         if (detailedReporting):
157             report_fh.write(datetime.now().strftime("%H:%M:%S") +
                    ": recieved report for genome #" + str(genome_id) +
                     ".\nDefaultData:\n\t" + str(resultsData) + "\
                    nParams:\n\t" + str(optimalResultsParams) + ".\
                    nNewData:\n\t" + str(optimalResultsData) + "\n\n")
158         else:
159             report_fh.write(datetime.now().strftime("%H:%M:%S") +
                    ": recieved report for genome #" + str(genome_id) +
                     ".\n")
160
161         ###   compare optimal to default report data
162         ##["Number of Sinks", "Max Global Skew", "Number of nets
                ", "Number of cells", "Combinational area", "Buf/Inv
                area", "Total cell area",
163         ##"Cell Internal Power", "Net Switching Power", "Total
                Dynamic Power", "Cell Leakage Power"]
```

```
164         for x in optimalResultsData:
165           optimalResultsData[x] = float(optimalResultsData[x])
166
167         diff = 0
168         diff += 15*(optimalResultsData["Max Global Skew"] -
               resultsData["Max Global Skew"])
169         diff += 3*(optimalResultsData["Cell Internal Power"] -
               resultsData["Cell Internal Power"])
170         diff += 3*(optimalResultsData["Net Switching Power"] -
               resultsData["Net Switching Power"])
171         diff += 3*(optimalResultsData["Total Dynamic Power"] -
               resultsData["Total Dynamic Power"])
172         diff += 3*(optimalResultsData["Cell Leakage Power"] -
               resultsData["Cell Leakage Power"])
173         diff += 1*(optimalResultsData["Combinational area"] -
               resultsData["Combinational area"])
174         diff += 1*(optimalResultsData["Buf/Inv area"] -
               resultsData["Buf/Inv area"])
175
176         genome.fitness -= diff
177
178
179 def run(config_file):
180   config = neat.Config(
181     neat.DefaultGenome,
```

```
182         neat.DefaultReproduction,
183         neat.DefaultSpeciesSet,
184         neat.DefaultStagnation,
185         config_file,
186         )
187
188     if not (startFromCheckpoint):
189         report_fh.write("running new population at " + datetime.
                now().strftime("%H:%M:%S"))
190         pop = neat.Population(config)
191     else:
192         report_fh.write("running from checkpoint " +
                checkpointPath + " at " + datetime.now().strftime("%H:%
                M:%S"))
193         pop = neat.Checkpointer.restore_checkpoint(checkpointPath)
194
195     ### added custom reporting to '/neat/nn/reporting.py -
            StdOutReporter' to report data to file instead of
            terminal
196     pop.add_reporter(neat.StdOutReporter(True, report_fh))
197     stats = neat.StatisticsReporter()
198     pop.add_reporter(stats)
199     pop.add_reporter(neat.Checkpointer(checkpointsWhen))
200
201     #neat.Checkpointer.restore_checkpoint
```

```
202     winner = pop.run(eval_genomes, generationLimit)
203
204     with open(winnerPKL, 'wb') as genome_fh:
205       pickle.dump(winner, genome_fh)
206
207     report_fh.write("\nBest Genome:\n{!s}".format(winner))
208
209     """
210     report_fh.write("\nOutput:\n")
211     winner_net = neat.nn.FeedForwardNetwork.create(winner,
          config)
212     global resultsDataList, optimalResultsData, resultsData
213     output = winner_net.activate(resultsDataList)
214
215     predictedParams = {}
216     for header, param in zip(predicted_params_required, output):
217       predictedParams[header] = round(param, 4)
218
219     ##["target_skew", "target_early_delay", "leaf_max_transition
          ", "max_transition", "max_capacitance", "max_fanout", "
          layer_list_for_sinks_start",
220     ##"layer_list_for_sinks_leng", "layer_list_start", "
          layer_list_leng"]
221     predictedParams["target_skew"] = predictedParams["
          target_skew"] * 1
```

```
222    predictedParams["target_early_delay"] = predictedParams["
           target_early_delay"] * 1
223    predictedParams["leaf_max_transition"] = predictedParams["
           leaf_max_transition"] * 1
224    predictedParams["max_transition"] = predictedParams["
           max_transition"] * 1
225    predictedParams["max_capacitance"] = int(predictedParams["
           max_capacitance"] * 1200)
226    predictedParams["max_fanout"] = int(predictedParams["
           max_fanout"] * 4000)
227    predictedParams["layer_list_for_sinks_start"] = int(
           predictedParams["layer_list_for_sinks_start"] * 9)
228    predictedParams["layer_list_for_sinks_leng"] = int(
           predictedParams["layer_list_for_sinks_leng"] * 10)
229    predictedParams["layer_list_start"] = int(predictedParams["
           layer_list_start"] * 9)
230    predictedParams["layer_list_leng"] = int(predictedParams["
           layer_list_leng"] * 10)
231
232    #report_fh.write("Input:\n\t" + str(resultsData) + "\nOutput
           Params:\n\t" + str(predictedParams) + "\nOutput Results
           :\n\t" + str(optimalResultsData) + "\n")
233    """
234
235 report_fh = open(reportFile, 'w')
```

```
236 #with open(reportFile , 'w') as report_fh:
237 run(configPath)
238 report_fh.write("\nDone: " + datetime.now().strftime("%H:%M:%S
        "))
239
240 report_fh.close()
241 print("DONE!")
242
243 """
244 psuedo:
245    Default Route:
246 route semi-random floorplan with deault clock tree settings
247 get default report data
248    Neural Net:
249 feed default report data to NEAT net
250 get predicted optimal cts input params
251    Optimal Route:
252 route new clock tree on same floorplan
253 get optimal report data
254    Neural Net
255 compare optimal to default report data
256 update Neural net
257
258 repeat until good enough?
259 """
```

## I.2 Automatic File Management (FileMan)

```
1 ###lines = source file parsed line by line (can be edited)
2 ###Store_to_file = complete file which will have lines
    replaced (cannot be edited, opened as 'r')
3 ###params = dictionary of the params to be fed into the new
    file
4 ###returns: full file data, to be used to overwrite
    destination file
5 def update_params(lines, Store_to_file, params):
6   metalLayers = ["M1", "M2", "M3", "M4", "M5", "M6", "M7", "M8
      ", "M9", "MRDL"]
7   startFill = False
8   #print("recieved params")
9   #print(params)
10  for ln in lines:
11    #print("parsing line " + ln)
12    if "set_clock_tree_options \\" in ln:
13      startFill = True
14      #print("found params line")
15    elif startFill:
16      param, sep, val = ln.partition(" ")
17      if "\"" not in val:
18        val = val.replace(" ", "")
19        val = val.replace("\\", "")
```

```
20        else:
21          val = val.split('"')
22          val = val[1]
23
24        if "-target_skew" in param:          ### default: 0s
25          #print("target_skew = " + val)
26          target_skew = params["target_skew"]
27          #target_skew = 0.0
28          ln_new = ln.replace(val, str(target_skew))
29          Store_to_file = Store_to_file.replace(ln, ln_new)
30
31        elif "-target_early_delay" in param:     ### default: 0
            s
32          #print("target_early_delay = " + val)
33          target_early_delay = params["target_early_delay"]
34          #target_early_delay = 0.15
35          ln_new = ln.replace(val, str(target_early_delay))
36          Store_to_file = Store_to_file.replace(ln, ln_new)
37
38        elif "-max_transition" in param:        ### default: 0.5
            ns
39          #print("max_transition = " + val)
40          max_transition = params["max_transition"]
41          #max_transition = 0.2
42          ln_new = ln.replace(val, str(max_transition))
```

```
43            Store_to_file = Store_to_file.replace(ln, ln_new)
44
45        elif "-leaf_max_transition" in param:     ### default:
            0.5ns or max_transition
46            #print("leaf_max_transition = " + val)
47            ###leaf_max_transition must be less than or equal to
                max_transition
48            leaf_max_transition = params["leaf_max_transition"]
49            #leaf_max_transition = 0.1
50            ln_new = ln.replace(val, str(leaf_max_transition))
51            Store_to_file = Store_to_file.replace(ln, ln_new)
52
53        elif "-max_capacitance" in param:       ### default: 0.6
            pf
54            #print("max_capacitance = " + val)
55            max_capacitance = params["max_capacitance"]
56            #max_capacitance = 300
57            ln_new = ln.replace(val, str(max_capacitance))
58            Store_to_file = Store_to_file.replace(ln, ln_new)
59
60        elif "-max_fanout" in param:           ### default: 2000
61            #print("max_fanout = " + val)
62            max_fanout = params["max_fanout"]
63            #max_fanout = 100
64            ln_new = ln.replace(val, str(max_fanout))
```

```
65            Store_to_file = Store_to_file.replace(ln, ln_new)
66
67        elif "-layer_list" in param:          ### default: all
            metal layers
68          #print("layer_list = " + val)
69          metalsNew = ""
70          first = params["layer_list_start"]
71          leng = params["layer_list_leng"]
72          for x in range(first, first + leng):
73            try:
74              metalsNew += metalLayers[x] + " "
75            except IndexError:
76              break
77          ln_new = ln.replace(val, metalsNew)
78          #ln_new = ln.replace(val, "M2 M3 M4 M5 M6")
79          Store_to_file = Store_to_file.replace(ln, ln_new)
80
81        elif "-layer_list_for_sinks" in param:      ### default:
             all metal layers
82          #print("layer_list_for_sinks = " + val)
83          metalsNew = ""
84          ###ranged to omit MRDL cause idk what it means
85          first = params["layer_list_for_sinks_start"]
86          leng = params["layer_list_for_sinks_leng"]
87          for x in range(first, first + leng):
```

```
88          try:
89            metalsNew += metalLayers[x] + " "
90          except IndexError:
91            break
92        ln_new = ln.replace(val, metalsNew)
93        #ln_new = ln.replace(val, "M1 M2 M3 M4 M5")
94        Store_to_file = Store_to_file.replace(ln, ln_new)
95        startFill = False
96        break
97    return Store_to_file
98
99  def update_floorplan(lines, Store_to_file, data):
100   startFill = False
101   for ln in lines:
102     #print("parsing line " + ln)
103     if (("# Create floor plan" in ln) and not(startFill)):
104       startFill = True
105     elif startFill:
106       param, sep, val = ln.partition(" ")
107       if "{" not in val:
108         val = val.replace(" ", "")
109         val = val.replace("\\", "")
110       else:
111         val = val.replace("\\", "")
112       if "shape" in param:
```

```
113             ln_new = ln.replace(val, data["shape"])
114             Store_to_file = Store_to_file.replace(ln, ln_new)
115         elif "core_side_dim" in param:
116             ln_new = ln.replace(val, data["dims"])
117             Store_to_file = Store_to_file.replace(ln, ln_new)
118             startFill = False
119             break
120     return Store_to_file
121
122 ###results = dictionary to hold output data
123 ###args = list of required data's names
124 ###file = file to search
125 ###tag = what to append to args names in the dictionary (ex: "
        _pre", "_post", "")
126 ###returns: null
127 def get_data(results, args, file, tag):
128     for ln in file:
129         for arg in args:
130             if arg in ln:
131                 if "layer_list" in arg:
132                     #print("found layer list: " + ln)
133                     startStore = False
134                     layers = ""
135                     for x in ln:
136                         if x == "M":
```

```
137                 startStore = True
138               if startStore:
139                 layers += x
140             #results[arg.replace(" ", "_") + tag] = layers
141             results[arg + tag] = layers
142         else:
143           for token in ln.split():
144             try:
145               if (int(token)):
146                 #results[arg.replace(" ", "_") + tag] = token
147                 results[arg + tag] = token
148             except:
149               try:
150                 if (float(token)) or (not float(token)):
151                   #results[arg.replace(" ", "_") + tag] =
                      token
152                   results[arg + tag] = token
153               except:
154                 pass
155         break
156
157 def make_results_file(header_data, bufferSize):
158   header = "run # "
159   for x in header_data:
160     buff = ""
```

```
161       for y in range(bufferSize - len(x)):
162         buff += " "
163       header += x + buff + "|\t"
164
165     ###create results file
166     with open("run_results.txt", 'w') as r_fh:
167       r_fh.write(header)
168
169 def record_results(header_data, results, runNum, file,
        bufferSize):
170     results_str = "\n" + str(runNum) + "\t#\t"
171     for x in all_data:
172       buff = ""
173       try:
174         for y in range(org_buffer - len(str(results[x]))):
175           buff += " "
176         results_str += str(results[x]) + buff + "|\t"
177       except:
178         results_str += "ERROR                    |\t"
179
180     with open("run_results.txt", 'a') as r_fh:
181       r_fh.write(results_str)
182
183 def dummy():
184     print("this doesnt do anything")
```

## I.3 Testing Environment

```
1  from fileMan import *
2  import subprocess
3  import random
4  import time
5  from datetime import datetime
6
7  import neat
8  import pickle
9
10 enableRouting = True
11 minFloorplanWallSize = 10
12 maxFloorplanWallSize = 40
13
14 defaultParams = {"target_skew" : 0.0, "target_early_delay" :
       0.0, "leaf_max_transition" : 0.5, "max_transition" : 0.5, "
       max_capacitance" : 600, "max_fanout" : 2000, "
       layer_list_for_sinks_start" : 0, "layer_list_for_sinks_leng
       " : 10, "layer_list_start" : 0, "layer_list_leng" : 10}
15 floorplanShapes = ["L", "T", "U", "X"]
16
17 dataTag = ""
18 paramsTag = ""
```

```
19 data_required = ["Number of Sinks", "Max Global Skew", "Number
      of nets", "Number of cells", "Combinational area", "Buf/
     Inv area", "Total cell area", "Cell Internal Power", "Net
     Switching Power", "Total Dynamic Power", "Cell Leakage
     Power"]
20 params_required = ["target_skew", "target_early_delay", "
     leaf_max_transition", "max_transition", "max_capacitance",
     "max_fanout", "layer_list_for_sinks", "layer_list"]
21 predicted_params_required = ["target_skew", "
     target_early_delay", "leaf_max_transition", "max_transition
     ", "max_capacitance", "max_fanout", "
     layer_list_for_sinks_start", "layer_list_for_sinks_leng", "
     layer_list_start", "layer_list_leng"]
22
23 resultsData = {}
24 resultsParams = {}
25 resultsDataList = []
26 optimalResultsData = {}
27 optimalResultsParams = {}
28
29 ###   get source file to read and eventually overwrite    ###
30 file_src = "icc/cmp_icc_src.tcl"
31 file_dest = "icc/cmp_icc.tcl"
32 with open(file_src, 'r') as fh:
33   filedata = fh.read()
```

```
34 fileLines = filedata.split("\n")
35
36 def test_genome(config_file):
37   config = neat.Config(
38     neat.DefaultGenome,
39     neat.DefaultReproduction,
40     neat.DefaultSpeciesSet,
41     neat.DefaultStagnation,
42     config_file,
43     )
44   with open("run4_9_best_genome.pkl", 'rb') as genome_fh:
45     best_genome = pickle.load(genome_fh)
46   winner_net = neat.nn.FeedForwardNetwork.create(best_genome,
      config)
47
48   for x in range(30):
49     ###   setup random floorplan, default CTS params    ###
50     ##  insert default cts params
51     newFileDefaultParamsData = update_params(fileLines,
        filedata, defaultParams)
52     newFileDefaultParamsDataLines = newFileDefaultParamsData.
        split("\n")
53     ##  insert psuedo random floorplan
54     random.seed(time.time())
55     data = {}
```

```
56        data["shape"] = random.choice(floorplanShapes)
57        a = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
58        b = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
59        c = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
60        d = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
61        e = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
62        f = str(random.randint(minFloorplanWallSize,
            maxFloorplanWallSize))
63        if data["shape"] == "L":
64          data["dims"] = "{" + a + " " + b + " " + c + " " + d + "
              }"
65        else:
66          data["dims"] = "{" + a + " " + b + " " + c + " " + d + "
               " + e + " " + f + "}"
67
68        report_fh.write("\nCreating floorplan: " + data["shape"] +
              " " + data["dims"] + "\n")
69        newFileDefault = update_floorplan(
            newFileDefaultParamsDataLines, newFileDefaultParamsData
            , data)
```

```
70      newFileDefaultLines = newFileDefault.split("\n")
71      with open(file_dest, 'w') as fh:
72        fh.write(newFileDefault)
73
74      ###   route on random floorplan with default CTS params
           ###
75      if (enableRouting):
76        p = subprocess.Popen(['./icc.csh', '-n'])
77        p.wait()
78
79      ###   get default report data   ###
80      with open("report/icc/results_conv_saed32nm_icc_postRoute.
           rpt", 'r') as post_route_report_fh:
81        post_route_report = post_route_report_fh.read().split("\
             n")
82
83      global resultsData
84      get_data(resultsData, data_required, post_route_report,
           dataTag)
85      get_data(resultsParams, params_required, post_route_report
           , paramsTag)
86
87      for x in resultsData:
88        resultsData[x] = float(resultsData[x])
89
```

```
90      ### format default report data for neural net   ###
91      global resultsDataList
92      resultsDataList = []
93      for x in data_required:
94        resultsDataList.append(resultsData[x])
95
96      output = winner_net.activate(resultsDataList)
97
98      predictedParams = {}
99      for header, param in zip(predicted_params_required, output
           ):
100       predictedParams[header] = round(param, 4)
101
102     ###   scale parameters for faster training time   ###
103     predictedParams["target_skew"] = predictedParams["
           target_skew"] * 1
104     predictedParams["target_early_delay"] = predictedParams["
           target_early_delay"] * 1
105     predictedParams["leaf_max_transition"] = predictedParams["
           leaf_max_transition"] * 1
106     predictedParams["max_transition"] = predictedParams["
           max_transition"] * 1
107     predictedParams["max_capacitance"] = int(predictedParams["
           max_capacitance"] * 1200)
```

```
108        predictedParams["max_fanout"] = int(predictedParams["
              max_fanout"] * 4000)
109        predictedParams["layer_list_for_sinks_start"] = int(
              predictedParams["layer_list_for_sinks_start"] * 9)
110        predictedParams["layer_list_for_sinks_leng"] = int(
              predictedParams["layer_list_for_sinks_leng"] * 10)
111        predictedParams["layer_list_start"] = int(predictedParams[
              "layer_list_start"] * 9)
112        predictedParams["layer_list_leng"] = int(predictedParams["
              layer_list_leng"] * 10)
113
114        ###   check predicted values for validity   ###
115        if  (predictedParams["leaf_max_transition"] >
              predictedParams["max_transition"]) or (predictedParams[
              "layer_list_start"] >= 9) or (predictedParams["
              layer_list_for_sinks_start"] >= 9) or (predictedParams[
              "layer_list_start"] < 2) or (predictedParams["
              layer_list_for_sinks_leng"] < 2):
116          report_fh.write("invalid output: " + str(predictedParams
                ))
117          diff = "INVALID"
118        else:
119          ###   if valid, setup predicted optimal CTS params
                ###
```

```
120         newFilePredictedParams = update_params(
               newFileDefaultLines, newFileDefault, predictedParams)
121         with open(file_dest, 'w') as fh:
122           fh.write(newFilePredictedParams)
123
124         ###   route predicted optimal clock   ###
125         if (enableRouting):
126           p = subprocess.Popen(['./icc.csh', '-n'])
127           p.wait()
128
129         ###   get predicted optimal report data
130         with open("report/icc/
               results_conv_saed32nm_icc_postRoute.rpt", 'r') as
               post_route_report_fh:
131           post_route_report = post_route_report_fh.read().split(
                 "\n")
132
133         global optimalResultsData
134         get_data(optimalResultsData, data_required,
               post_route_report, dataTag)
135         get_data(optimalResultsParams, params_required,
               post_route_report, paramsTag)
136
137         for x in optimalResultsData:
138           optimalResultsData[x] = float(optimalResultsData[x])
```

```
139
140         diff = 0
141         diff += 15*(optimalResultsData["Max Global Skew"] -
              resultsData["Max Global Skew"])
142         diff += 3*(optimalResultsData["Cell Internal Power"] -
              resultsData["Cell Internal Power"])
143         diff += 3*(optimalResultsData["Net Switching Power"] -
              resultsData["Net Switching Power"])
144         diff += 3*(optimalResultsData["Total Dynamic Power"] -
              resultsData["Total Dynamic Power"])
145         diff += 3*(optimalResultsData["Cell Leakage Power"] -
              resultsData["Cell Leakage Power"])
146         diff += 1*(optimalResultsData["Combinational area"] -
              resultsData["Combinational area"])
147         diff += 1*(optimalResultsData["Buf/Inv area"] -
              resultsData["Buf/Inv area"])
148
149         ###   account for genome.fitness -= diff (i.e. the lower
                diff (including negatives) the better)
150         diff = -diff
151
152     report_fh.write("Given: " + str(resultsData) + "\n")
153     report_fh.write("Gave: " + str(predictedParams) + "\n")
154     report_fh.write("For: " + str(optimalResultsData) + "\n")
155     report_fh.write("diff = " + str(diff) + "\n\n")
```

```
156
157
158 report_fh = open("neat_report_zFinal.txt", 'a')
159 configPath = "neat_config_strict"
160 test_genome(configPath)
161
162 report_fh.close()
163 print("DONE!")
```

# Appendix II

# Supporting Code

## II.1 Synopsys tcl Run File

```
1
2  #
3  # set design name
4  #
5  set design_name results_conv
6
7  set my_type "_icc"
8  #set my_report_pre "_icc_preRoute"
9  set my_report_post "_icc_postRoute"
10 set my_input_type "_scan"
11
12 set my_max_area 1200
13 #
```

```
14 # compile effort can be: low, medium, high
15 #
16 set my_compile_effort "high"
17
18 set hdlin_enable_presto false
19 set hdlin_keep_signal_name all
20
21 set bus_naming_style {%s[%d]}
22 set bus_inference_style {%s[%d]}
23
24 #/* connect to all ports in the design, even if driven by the
       same net */
25 #/* compile_fix_multiple_port_nets = true */
26 set_fix_multiple_port_nets -all -buffer_constants
27
28 #/* do not allow wire type tri in the netlist */
29 set verilogout_no_tri true
30
31 set verilogout_equation false
32
33 #/* to fix those pesky escaped names */
34 #/* to be used with 'change_names -hierarchy' */
35 #/* after a compile - should only be needed in */
36 #/* extreme cases when 'bus_naming_style' isn't fully working
       */
```

```
37 define_name_rules Verilog -allowed {a-z A-Z 0-9 _}
       -first_restricted {0-9 _} -replacement_char "__" -type cell
38 define_name_rules Verilog -allowed {a-z A-Z 0-9 _ []}
       -first_restricted {0-9 _} -replacement_char "__" -type port
39 define_name_rules Verilog -allowed {a-z A-Z 0-9 _}
       -first_restricted {0-9 _} -replacement_char "__" -type net
40 set default_name_rules Verilog
41
42 #
43 # for SAIF file generation
44 #
45 set power_preserve_rtl_hier_names true
46
47 remove_design -all
48 file delete -force "./gds/" "./lib"
49 file delete "./PIM_Cluster_port_map.*"
50 file delete -force "./report/icc/"
51 file mkdir "./gds/" "./lib" "./netlist" "./sdf" "./spf" \
52   "./report" "./report/dc" "./report/pt" "./report/pr" "./saif
       " \
53   "./report/icc/"
54
55 set report_dir "./report/icc/"
56 set saif_dir "./saif/"
57
```

```
58 #set my_report_dir "./report/my_icc/"
59
60 set hdlin_use_cin true
61 set synlib_model_map_effort "high"
62 set hdlout_uses_internal_busses true
63 # Turn on auto wire load selection
64 # (library must support this feature)
65 set auto_wire_load_selection true
66
67 set synlib_wait_for_design_license "DesignWare"
68
69 #/* set technology library */
70 source "dc/tech_config.tcl"
71
72 set link_library [concat  $link_library $synthetic_library]
73
74 #
75 #/*****************************************************/
76 #/*                                                   */
77 #/* Set up environment for ic compiler                */
78 #/*                                                   */
79 #/*****************************************************/
80 #
81 set SAED_EDK32nm_ROOT "/class/ee620/maieee/lib/synopsys/
     SAED_EDK32-28nm/SAED32_EDK"
```

```
82 set PR_MW_LIB [format "%s%s" [format "%s%s" "./lib/"
     $design_name] $my_type]
83 set SAED_EDK32nm_MW_TF "saed32nm_1p9m_mw.tf"
84 set SAED_EDK32nm_OA_TF "saed32nm_1p9m_oa.tf"
85 set SAED_EDK32nm_REF_LIB [list \
86    "${SAED_EDK32nm_ROOT}/lib/stdcell_rvt/milkyway/
        saed32nm_rvt_1p9m" \
87    "${SAED_EDK32nm_ROOT}/lib/stdcell_lvt/milkyway/
        saed32nm_lvt_1p9m" \
88    "${SAED_EDK32nm_ROOT}/lib/stdcell_hvt/milkyway/
        saed32nm_hvt_1p9m" \
89 ]
90
91 # Create Milkyway library
92 create_mw_lib \
93    -technology ${SAED_EDK32nm_ROOT}/tech/milkyway/${
        SAED_EDK32nm_MW_TF} \
94    -mw_reference_library ${SAED_EDK32nm_REF_LIB} \
95    -bus_naming_style {[%d]} \
96    ${PR_MW_LIB}
97
98 # Add timing and cap libraries
99 set_tlu_plus_files \
100   -max_tluplus ${SAED_EDK32nm_ROOT}/tech/star_rcxt/
        saed32nm_1p9m_Cmax.tluplus \
```

```
101    -min_tluplus ${SAED_EDK32nm_ROOT}/tech/star_rcxt/
         saed32nm_1p9m_Cmin.tluplus \
102    -tech2itf_map ${SAED_EDK32nm_ROOT}/tech/milkyway/
         saed32nm_tf_itf_tluplus.map
103
104 open_mw_lib ${PR_MW_LIB}
105
106 file copy .oalib ${PR_MW_LIB}
107
108 import_design [list [format "%s%s%s%s" "netlist/" $design_name
        $tech_lib "${my_input_type}.vs"]] \
109    -format verilog \
110    -top $design_name \
111    -cel $design_name
112
113 current_design $design_name
114 current_mw_cel [get_mw_cel $design_name]
115
116 read_sdf [list [format "%s%s%s%s" "sdf/" $design_name
       $tech_lib "${my_input_type}.sdf"]]
117
118 #
119 # apply constraints
120 #
121 source "cons/${design_name}_cons_defaults_icc.tcl"
```

```
122 source "cons/${design_name}_clocks_cons.tcl"
123 source "cons/${design_name}_cons.tcl"
124
125 set_clock_uncertainty 0 clk
126
127 current_design $design_name
128 current_mw_cel [get_mw_cel $design_name]
129
130 # Add power rails
131 set power "VDD"
132 set ground "VSS"
133 set powerPort "VDD"
134 set groundPort "VSS"
135 foreach net {VDD} {
136   derive_pg_connection −power_net $net −power_pin $net
        −create_ports top
137 }
138 foreach net {VSS} {
139   derive_pg_connection −ground_net $net −ground_pin $net
        −create_ports top
140 }
141 derive_pg_connection −tie
142
143 current_design $design_name
144 current_mw_cel [get_mw_cel $design_name]
```

```
145
146 echo [concat {++++ Floorplan Design}]
147 # Create floor plan
148 #\
149 create_floorplan \
150   -control_type aspect_ratio \
151   -core_utilization 0.80 \
152   -core_aspect_ratio 1.0 \
153   -left_io2core 0.50 \
154   -bottom_io2core 0.50 \
155   -right_io2core 0.50 \
156   -top_io2core 0.50
157 #save_mw_cel
158
159 initialize_rectilinear_block \
160   -shape L \
161   -control_type ratio \
162   -core_side_dim {16 27 26 13}\
163   -core_utilization 0.8 \
164   -left_io2core 0.50 \
165   -bottom_io2core 0.50 \
166   -right_io2core 0.50 \
167   -top_io2core 0.50
168 save_mw_cel
169
```

```
170 current_design $design_name
171 current_mw_cel [get_mw_cel $design_name]
172
173 echo [concat {++++ Create Power Rings}]
174 # Create power rings
175 create_rectangular_rings \
176   -nets {VDD VSS} \
177   -left_offset 0.2 \
178   -left_segment_width 1 \
179   -right_offset 0.2 \
180   -right_segment_width 1 \
181   -bottom_offset 0.2 \
182   -bottom_segment_width 1 \
183   -top_offset 0.2 \
184   -top_segment_width 1
185 save_mw_cel
186
187 current_design $design_name
188 current_mw_cel [get_mw_cel $design_name]
189
190 echo [concat {++++ Place Design}]
191 get_scan_chains
192 # Run placer
193 place_opt \
194   -area_recovery \
```

```
195     -congestion \
196     -power \
197     -continue_on_missing_scandef \
198     -cts
199 save_mw_cel
200
201 current_design $design_name
202 current_mw_cel [get_mw_cel $design_name]
203
204 echo [concat {++++ Route Power Rails}]
205 # Route power rails
206 preroute_standard_cells -nets {VDD VSS} \
207     -connect horizontal \
208     -extend_to_boundaries_and_generate_pins
209 save_mw_cel
210
211 current_design $design_name
212 current_mw_cel [get_mw_cel $design_name]
213
214 echo [concat {++++ Insert Clock Tree Random Settings}]
215 # Insert clock tree
216
217 set_clock_tree_options \
218     -target_skew 0.0 \
219     -target_early_delay 0.0 \
```

```
220     -max_transition 0.5 \
221     -leaf_max_transition 0.5 \
222     -max_capacitance 600 \
223     -max_fanout 2000 \
224     -layer_list "M1 M2 M3 M4 M5 M6 M7 M8 M9 MRDL" \
225     -layer_list_for_sinks "M1 M2 M3 M4 M5 M6 M7 M8 M9 MRDL"
226
227 clock_opt \
228     -fix_hold_all_clocks \
229     -area_recovery \
230     -congestion \
231     -continue_on_missing_scandef \
232     -power
233 save_mw_cel
234
235 #echo [concat {++++ Pre-Route Timing Analysis Default Settings
        }]
236 # Check timing default
237 #current_design $design_name
238 #current_mw_cel [get_mw_cel $design_name]
239 #redirect [format "%s%s" [format "%s%s" [format "%s%s"
        $report_dir $design_name] $tech_lib] "${my_report_pre}.rpt
        "] { echo [concat {Pre-Route Timing Analysis Random
        Settings}] }
```

```
240 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_clock_tree }
241 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_timing -sign 4 -max_paths 10 }
242 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_area }
243 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_area -hierarchy }
244
245 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_clock_tree_options }
246 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { echo [concat {Pre-Route Power Analysis Random
      Settings}] }
247 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
      .rpt"] { report_power -analysis_effort medium -verbose }
248 #redirect -append [format "%s%s" [format "%s%s" [format "%s%
      s" $report_dir $design_name] $tech_lib] "${my_report_pre}
```

```
    .rpt"] { report_clock_tree_power }
249
250 current_design $design_name
251 current_mw_cel [get_mw_cel $design_name]
252
253 echo [concat {++++ Route Design}]
254 # route design
255 route_opt -effort high
256 save_mw_cel
257
258 echo [concat {++++ Check, Fix DRC Errors}]
259 # Check & Fix DRC Errors
260 route_search_repair \
261   -rerun_drc \
262   -loop "4" \
263   -num_cpus "4" \
264   -run_time_limit "10"
265 save_mw_cel
266
267 echo [concat {++++ Check, Fix LVS Errors}]
268 # Check & Fix LVS Errors
269 route_zrt_eco -max_detail_route_iterations 5
270 verify_lvs -check_open_locator -check_short_locator
271 save_mw_cel
272
```

```
273 # Add filler cells
274 insert_stdcell_filler \
275   -cell_without_metal "SHFILL128_RVT SHFILL64_RVT SHFILL3_RVT
        SHFILL2_RVT SHFILL1_RVT" \
276   -connect_to_power {VDD} \
277   -connect_to_ground {VSS}
278 save_mw_cel
279
280 # Fix power nets
281 foreach net {VDD} {
282   derive_pg_connection -power_net $net -power_pin $net
        -create_ports top
283 }
284 foreach net {VSS} {
285   derive_pg_connection -ground_net $net -ground_pin $net
        -create_ports top
286 }
287 save_mw_cel
288
289 echo [concat {++++ Check DRC}]
290 # Run DRC & LVS
291 verify_route \
292   -num_cpu "4"
293
294 echo [concat {++++ Check LVS}]
```

```
295 verify_lvs
296
297 echo [concat {++++ Post-Route Timing Analysis}]
298 # Check timing
299 current_design $design_name
300 current_mw_cel [get_mw_cel $design_name]
301 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
      " $report_dir $design_name] $tech_lib] "${my_report_post}
      .rpt"] { echo [concat {Post-Route Timing Analysis}] }
302 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
      " $report_dir $design_name] $tech_lib] "${my_report_post}
      .rpt"] { report_clock_tree }
303 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
      " $report_dir $design_name] $tech_lib] "${my_report_post}
      .rpt"] { report_timing -sign 4 -max_paths 10 }
304 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
      " $report_dir $design_name] $tech_lib] "${my_report_post}
      .rpt"] { report_area }
305 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
      " $report_dir $design_name] $tech_lib] "${my_report_post}
      .rpt"] { report_area -hierarchy }
306
307 echo [concat {++++ Post-Route Power Analysis}]
308 current_design $design_name
309 current_mw_cel [get_mw_cel $design_name]
```

```
310
311 if {[file exists [format "%s%s" $saif_dir "${design_name}
        _bw.saif"]]} {
312   reset_switching_activity
313   echo [concat {Reading Backwards SAIF File}]
314   read_saif -input [format "%s%s" $saif_dir "${design_name}
          _bw.saif"] -instance_name test/top
315 }
316
317 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
        " $report_dir $design_name] $tech_lib] "${my_report_post}
        .rpt"] { report_clock_tree_options }
318 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
        " $report_dir $design_name] $tech_lib] "${my_report_post}
        .rpt"] { echo [concat {Post-Route Power Analysis}] }
319 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
        " $report_dir $design_name] $tech_lib] "${my_report_post}
        .rpt"] { report_power -analysis_effort medium -verbose }
320 redirect -append [format "%s%s" [format "%s%s" [format "%s%s
        " $report_dir $design_name] $tech_lib] "${my_report_post}
        .rpt"] { report_power -analysis_effort medium -verbose
        -hier }
321
322 echo [concat {++++ Stream out GDSII, Netlist and SDF}]
323 current_design $design_name
```

```
324 current_mw_cel [get_mw_cel $design_name]
325 # Stream out GDSII
326 set_write_stream_options \
327   -output_pin {text geometry} \
328   -keep_data_type
329
330 write_stream \
331   -format gds \
332   [format "%s%s" [format "%s%s" "./gds/" $design_name] ".gds"]
333
334 #
335 # Write Netlist and SDF
336 #
337 current_design $design_name
338 current_mw_cel [get_mw_cel $design_name]
339 change_names -rules Verilog -hierarchy
340 write_verilog [format "%s%s" [format "%s%s" [format "%s%s"
      "./netlist/" $design_name] $tech_lib] "${my_type}.vs"]
341 write_sdf -context verilog [format "%s%s" [format "%s%s" [
      format "%s%s" "./sdf/" $design_name] $tech_lib] "${my_type
      }.sdf"]
342 save_mw_cel
343 close_mw_cel
344
345 echo [concat {++++ Finished...}]
```

# Appendix III

# ICC Shell

ICC_Shell is the users manual for the Synopsys IC Compiler [6], it provides descriptions, examples, and more, for all available shell functions. Below are the ones most relevant to this work.

## III.1 Clock Tree Options

1. set_clock_tree_options: Specifies clock tree synthesis constraints and options for clocks in the design.

   (a) layer_list "layer_names": Specifies the layers that can be used for routing the clock nets in the specified clock trees. If the list has more than two elements, the lower layer should appear before the upper layer. By default, all routing layers can be used for clock nets.

   (b) layer_list_for_sinks "layer_names": Specifies the layers that can be used for routing the clock leaf nets in the specified clock trees. If the list has more than two elements, the lower layer should appear before the upper layer. This option overrides the

-layer_list option when routing leaf-level clock nets, if both options are specified.

(c) target_early_delay "insertion_delay": Specifies the minimum insertion delay constraint in design unit for the specified clock trees. When you specify this option, the clock tree synthesis engine builds an initial optimized clock tree. If the insertion delay of the longest path of this clock tree is smaller than the specified value, the clock tree synthesis engine adds a chain of cells from the reference list as needed to meet this delay. By default, the target early delay is 0.

(d) target_skew "skew": Specifies the required value for maximum skew in design unit for the specified clock trees. After the tool meets this skew target, the optimization concentrates more on other QoR goals, such as insertion delay and area. By default, the target skew is 0.

(e) max_capacitance "capacitance": Specifies the maximum capacitance design rule constraint in main library units for the specified clock trees. This value takes precedence over the maximum capacitance constraint set on the design, as well as over the one coming from the library. This constraint is used when reporting maximum capacitance violations during clock tree synthesis and is also used to control maximum capacitance DRC fixing beyond exceptions. By default, the maximum capacitance is 0.6 pf.

(f) max_transition "transition_time": Specifies the maximum transition time design rule constraint in main library unit for the buffers and inverters used while compiling the specified clock trees. This value takes precedence over the maximum transition time constraint set on the design, as well as over the one coming from the library, but can be overridden on all instances of specific types of buffers and inverters in the clock tree by specifying a different value for this constraint on the

set_clock_tree_references command line. The clock tree root cell and clock-gating cells present on the clock network are not affected by the set_clock_tree_references design rule constraints unless they are instances of buffers or inverters with overridden design rule constraints. By default, the maximum transition time is 0.5 ns.

(g) max_fanout "fanout": Specifies the maximum fanout design rule constraint for the cells in the specified clock trees. By default, the maximum fanout constraint set on the design by using the set_max_fanout command, as well as the one coming from the library are ignored during clock tree synthesis. By default, the maximum fanout is 2000.

(h) leaf_max_transition "transition_time": Specifies the maximum transition time design rule constraint in main library unit for the buffers and inverters used while synthesizing the leaf nets during the compilation of the specified clock trees. A leaf net is defined as a net which drives the clock pin of at least one register, latch, or multibit register. This value must be tighter than the value specified with the -max_transition option of the set_clock_tree_options command or the set_max_transition command. By default, the maximum transition constraint for the leaf nets is equal to the maximum transition constraint of the rest of the clock tree.

## III.2 Floor Plan Options

2. initialize_rectilinear_block: Creates L-, T-, U-, and cross-shaped floorplans for rectilinear blocks.

   (a) bottom_io2core "distance": Specifies the shortest distance between the I/O pin and

the bot- tom side of the core boundary.

(b) control_type “ratio | length”: Specifies how to interpret the list of dimensions provided for the -core_side_dim option. When the control type is ratio, each dimension in the list rep- resents the relative proportion of the dimension of the edge to the sum of all the dimensions listed. For example, if the list of dimensions of an L-shaped block is {1 2 1 1}, the tool inter- prets the dimension of side a, c, or d is 20% of the sum of the dimensions listed, and the dimension of side b is 40% of the summation. When the control type is length, the dimensions in the list represent the actual physical dimensions for each edge of the polygon.

(c) core_side_dim “{side_a side_b side_c side_d [side_e side_f]}”: Specifies the dimensions of the edges of the rectilinear block. This list contains a maximum of 6 values, depending on the shape specified by the -shape option. The semantics of the values depends on the -control_type option specified. If you provide more values than needed to describe the specified shape, the extra values are ignored. If you do not provide all the values needed to describe the specified shape, the tool issues an error.

(d) core_utilization ratio_val: Specifies a utilization number between 0 and 1.0. This number indicates the amount of the core area used for cell placement. This number is calculated as a ratio of the total cell area to the core area. The cell area is the total area of all standard and macro cells. For example, a core utilization of 0.8 means that 80 percent of the core area is used for cell placement and 20 percent is available for routing.

(e) left_io2core “distance”: Specifies the shortest distance between the I/O pin and the left side of the core boundary.

(f) right_io2core “distance”: Specifies the shortest distance between the I/O pin and right side of the core boundary.

(g) shape “L | T | U | X”: Specifies a template shape used to determine the cell boundary and core shape of the rectilinear block. The following diagram shows the definition of the edges and the orientation of the L-, T-, U-, and X-shaped rectilinear blocks.

(h) top_io2core “distance”: Specifies the shortest distance between the I/O pin and the top side of the core boundary.