Behaviour Analysis in Binary SoC Data

By

DAVID M^{C}EWAN

Department of Electrical and Electronic Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

JAN 2022

Thirty five thousand words

Abstract

Modern System-on-Chips (SoCs) are highly complex systems specified chiefly using digital clocked logic and implemented in silicon circuitry. The applications served by SoC-based products vary from tiny embedded devices to massive supercomputers. A common factor in systems across the scale is that designers require a thorough understanding of component interactions lest they sacrifice unnecessary energy, performance, development time and other important resources.

The approach taken in this thesis is via behaviour analysis on key internal signals carrying binary data. Systems with appropriate instrumentation can provide engineers with signals on the workings and interactions of components in state format (e.g. `transactionIsInflight`) or event format (e.g. `transactionInitiated` and `transactionCompleted`). Gaining an understanding of a system's runtime behaviour entails uncovering knowledge about key statistics of these signals and, with equal importance, knowledge about the underlying relationships connecting them.

Statistical experiments are devised to find which interpretations of the term "correlation" are most useful, and whether individual performance counters can be used effectively for correlation analysis over a single time window. Metrics based on covariance and independence are found to be more useful than those based on sets or geometrical concepts. Individual performance counters are shown to be inadequate for discerning useful correlation information within a single time window; however, the inclusion of at least one pairwise counter enables useful correlation metrics to be calculated. Building upon these results, a hardware design is developed to discover what is required to implement a device for passive on-chip correlation monitoring and data collection. A novel sampling method is developed out of necessity to enable correlation to be detected between signals with a slight but unknown delay, and a novel window function which enables low-cost windowing counters. The correlator device is demonstrated via integration with UltraSoC's commercial toolkit for embedded analytics in a highly complex multicore system. It is observed that behaviour analysis involves a large amount of data that is difficult to consume using existing tools and techniques (e.g. time-series plots and tables). This is addressed by tackling the problem of how to effectively present large amounts of correlation data in a way which allows a system designer to comprehend the required statistics without being overwhelmed.

This work prepares SoC designers for understanding their system's behaviour at a theoretical level via a thorough evaluation of correlation metrics, and at a practical level via an exploration of both implementation in digital logic hardware and novel visualisation techniques to present behaviour.

Dedication and Acknowledgements

I could not be more grateful to my supervisors Jose Nunez-Yanez, Kerstin Eder, and Gajinder Panesar, who've given me so much guidance and support over the course of this project. Also, many thanks to the tireless efforts of the other university staff, particularly Mark Beech and Suzanne Binding. Sponsorship from UltraSoC and the industrial expertise provided by Marcin Hlond, Rupert Baines, Matt Lokes, Iain Robertson, and Hanan Moller has proven invaluable in realising sound and practical hardware. I'm especially grateful to Χρυσανθη, Lolly, Lockhart, Robb, and Paul for their help with writing and great patience. And finally, my friends and colleagues in the Communications CDT Adam, Roger, Justin, Michael, and Mark who have helped spawn interesting ideas on beermats and napkins.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

TABLE OF CONTENTS

Page

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**
- 1.1 General Problem Description . . . 1
- 1.2 Objectives . . . 3
- 1.3 Thesis Outline . . . 3
- 1.4 Notation Summary . . . 5

2 Background and Literature Review **7**
- 2.1 Abstract Hypothesis . . . 7
  - 2.1.1 Knowledge Discovery and Interestingness . . . 7
  - 2.1.2 Behaviour and Function . . . 9
  - 2.1.3 Correlation . . . 11
- 2.2 Data Collection . . . 11
  - 2.2.1 Pre-existing Datasets . . . 12
  - 2.2.2 Collecting SoC Data . . . 13
- 2.3 Visualisations for Behaviour . . . 15
  - 2.3.1 Heuristic Evaluation . . . 16
  - 2.3.2 Existing Tools and Techniques . . . 19
- 2.4 Summary and Direction . . . 25

3 Correlation Metrics in Binary SoC Data **27**
- 3.1 Aim . . . 27
  - 3.1.1 Problem Description . . . 27
  - 3.1.2 Objective . . . 29
  - 3.1.3 Approach . . . 29
- 3.2 Definition of Metrics . . . 30
  - 3.2.1 Binary Sets . . . 32

3.2.2 Geometric Vectors . . . 32
3.2.3 Probabilistic Metrics . . . 33
3.3 Experimental Procedure . . . 34
3.3.1 Probabilistic Assumptions . . . 34
3.3.2 Further Description of System Construction . . . 35
3.3.3 Methods of Scoring . . . 36
3.4 Metric Scoring Results . . . 39
3.5 Learning New Metrics . . . 42
3.5.1 Calculating Metrics with Counters . . . 42
3.5.2 Neural Network Parameters and Setup . . . 43
3.5.3 Metric Learning Results . . . 47
3.6 Conclusion . . . 51

4 Hardware for SoC Correlation Analysis 53
4.1 Aim . . . 53
4.1.1 Problem Description . . . 53
4.1.2 Objective . . . 55
4.1.3 Approach . . . 55
4.2 Background . . . 56
4.2.1 Sampling . . . 56
4.2.2 Window Functions . . . 58
4.2.3 Synthesis, Layout, and Characterisation . . . 60
4.3 Correlator Device . . . 62
4.3.1 Sampling Mechanism . . . 64
4.3.1.1 Introducing Variance to the Sample Period . . . 65
4.3.1.2 Jittery Sample Strobe Mechanism . . . 66
4.3.2 Windowing Counters . . . 69
4.3.2.1 Rectangular . . . 69
4.3.2.2 Logdrop . . . 71
4.3.3 Consuming Correlation Results . . . 76
4.3.3.1 Real-time Results via $\Delta\Sigma$ and Low-Pass Filter . . . 77
4.3.3.2 Calculating Correlation Using Counters . . . 77
4.3.4 Synthesis Characterisation on FPGAs . . . 79
4.4 Integration Case Studies . . . 84
4.4.1 Case Study 1: Slow Datalink . . . 85
4.4.2 Case Study 2: Fast Datalink . . . 91
4.5 Conclusion . . . 94

5 Visualizing Pairwise Correlations **97**
5.1 Aim . . . 97
5.1.1 Problem Description . . . 97
5.1.2 Objective . . . 99
5.1.3 Approach . . . 99
5.2 Visualisations for Binary SoC Data . . . 100
5.2.1 Statistics on Binary Signals . . . 101
5.2.2 Netgraph View . . . 103
5.2.2.1 Statistics on Individual Signals . . . 103
5.2.2.2 Identicons as Aids for Orientation . . . 105
5.2.2.3 Circular Graphical Layout . . . 106
5.2.2.4 Interactive Features . . . 108
5.2.3 Tabdelta View . . . 108
5.3 Gestalt Evaluation Against Heuristics . . . 112
5.3.1 Nielsen's Usability Heuristics . . . 112
5.3.2 Shneiderman's Visual-Information-Seeking Tasks . . . 115
5.3.3 Gerhardt-Powals' Cognitive Engineering Principles . . . 116
5.3.4 Forsell and Johansson's Visualisation Heuristics . . . 118
5.3.5 Direct Comparison with State of the Art . . . 120
5.4 Demonstration Case Studies . . . 122
5.4.1 Case Study 1: Static Behaviours . . . 122
5.4.1.1 Description of praxi System . . . 122
5.4.1.2 Behaviour Overview in Netgraph View . . . 124
5.4.1.3 Behaviour Overview in Tabdelta View . . . 125
5.4.2 Case Study 2: Changing Behaviours . . . 127
5.4.2.1 Description of tinn System . . . 127
5.4.2.2 Behaviour Changes in Netgraph View . . . 127
5.4.2.3 Behaviour Changes in Tabdelta View . . . 131
5.5 Conclusion . . . 132

6 Conclusions **133**
6.1 Summary of Thesis . . . 133
6.2 Achieved Objectives and Contributions . . . 135
6.3 Future Work . . . 136

Appendices **138**

A List of Publications **139**

B Parameter Sets **141**

C Results From Section 3.4 By System Type **143**

D Model Summaries for FFNN-based Metrics **149**

D.1 Counter Inputs Combination `perfCntrs` 149

D.2 Counter Inputs Combination `withAssist` 152

D.3 Counter Inputs Combination `fullAssist` 155

D.4 Counter Inputs Combination `withIsect` 158

D.5 Counter Inputs Combination `withSymdiff` 161

D.6 Counter Inputs Combination `withIsectSymdiff` 164

E Quantified Confidence in $f_{\mathbf{max}}$ Robustness **167**

E.1 Confidence In A Single Design 168

E.2 Comparison of Competing Designs 170

E.3 Comparing Similar Configurations 172

F Modelling the Jittery Strobe Counter Circuit **175**

G BytePipe Protocol **179**

H USB-FS Electrical Interface Over Five Pins **185**

I Colourspace for Bounded 2D Data **189**

Bibliography **195**

Glossary **210**

List of Tables

Table **Page**

3.1 FFNN architectures in metric-learning experiments. 45
3.2 Hyper-parameters for FFNN experiments. 46
3.3 Input sets for FFNN experiments. 46

5.1 Direct comparison of usability features. 121

E.1 Comparison of PRNG implementations on Lattice iCE40LP. 173

List of Figures

Figure **Page**

2.1 Example data on a waveform viewer. 20
2.2 Spiral and Helical Visualisations of Periodic Time-Series. 21
2.3 Example kernel data on Trace Compass. 22
2.4 Example corrgram. 23
2.5 Example of Solar Correlation Map. 24

3.1 Example system with src and dst nodes connected via binary operations. 35
3.2 Probabilistic model for Binary SoC Data. 37
3.3 KDE plots of metric score PDFs. 40
3.4 Illustrations FFNN structure and input combinations. 44
3.5 Epoch loss in metric learning (Part 1 of 2). 49
3.6 Epoch loss in metric learning (Part 2 of 2). 50

4.1 Usage of correlator device. 62
4.2 Microarchitecture of a correlator engine. 63
4.3 Correlator data capture mechanism. 64
4.4 Event/counter anti-phase sampling timeline. 65
4.5 Logical design of sampling strobe generator. 66
4.6 PDF of sampling strobe period. 68
4.7 Logical design of scalable counter with rectangular windowing. 71
4.8 Logdrop windowing function with small N. 73
4.9 Logdrop windowing function with large N. 74
4.10 Logical implementation of Logdrop. 75
4.11 Logical design of scalable counter with Logdrop windowing. 76
4.12 Logical design of correlator metric calculators. 78
4.13 Multi-PnR results correlator on Lattice iCE40LP8K. 81
4.14 Correlator floorplans in Lattice iCE40LP. 82
4.15 Correlator floorplan in Xilinx 7-Series FPGAs. 83
4.16 OpenPiton Memory Hierarchy Datapath. 85

4.17 Logical connections of case study 1. 86
4.18 Experimental setup of case study 1. 87
4.19 Oscilloscope plots showing correlator results. 88
4.20 Floorplans of OpenPiton+Ariane on VC707. 90
4.21 UltraSoC Status Monitor augmented with correlator engines. 92
4.22 Plots of recorded data and correlations from correlator. 93

5.1 Visualisation of individual signal's statistics. 104
5.2 Identicon derived from signal name. 105
5.3 Netgraph visualisation with edges in 2D colourspace. 107
5.4 Mouse interaction with netgraph visualisation. 109
5.5 Features of tabdelta visualisation. 110
5.6 Tabdelta view from praxi read sub-system over time. 111
5.7 Demonstration system "praxi". 123
5.8 System Verilog specifying actual behaviour of `slv.idle`. 124
5.9 Identifiable behaviour sub-systems in praxi. 125
5.10 Specific behaviour of praxi `idle` signal visible in tablar view. 126
5.11 Demonstration system "tinn". 128
5.12 Changing behaviour patterns visible in netgraph. 129
5.13 Tabdelta view of tinn over time. 131

B.1 Linear diagram of SoC parameter sets. 142

C.1 KDE plots of metric score PDFs on AND-only logical relationships. . . 144
C.2 KDE plots of metric score PDFs on OR-only logical relationships. . . . 145
C.3 KDE plots of metric score PDFs on XOR-only logical relationships. . . 146
C.4 KDE plots of metric score PDFs on mix of logical relationships. 147
C.5 KDE plots of metric score PDFs on LHA logical relationships. 148

E.1 Multi-PnR results of prototype BytePipe gadget. 169
E.2 Multi-PnR comparison of functionally equivalent design choices. 171
E.3 Multi-PnR results of Xoroshiro gadgets. 174

F.1 State machine of strobe generation down-counter c. 176
F.2 PMF of number of cycles in a period with $s = 100$. 177

G.1 BytePipe single read transaction. 181
G.2 BytePipe single write transaction. 181
G.3 BytePipe burst read transaction. 182
G.4 BytePipe burst write transaction. 183

H.1 Schematic for external USB Full Speed electrical interface. 186
H.2 PCB rendering of external USB-FS electrical interface. 187

I.1 Variants of bounded 2D colourspace. 191
I.2 Worked example of colour calculation. 193
I.3 Example SoC analysis assisted by colourspace. 194

CHAPTER 1

INTRODUCTION

1.1 General Problem Description

Developing SoCs which are tuned for the desired balance of compute performance, power consumption, cost, development time, and other product constraints is a difficult task. Detailed knowledge of many layers of technology is required, from the intricacies of physical silicon manufacture to high levels of abstraction in software. The logical layer is concerned with producing a logical construction of primitive components which can perform the required data transformations. For example, a manufacturing process might provide AND, OR, and NOT primitives which can be combined to implement a Central Processing Unit (CPU). This research is principally concerned with analysis on the logical layer which can have immediate and far-reaching consequences on the physical implementation as well as the performance of higher level system components such as software.

The complexity of SoCs has continued to grow at an astonishing pace, particularly those which are aimed towards high-performance applications. For example, Graphcore's most recent chip, the GC200 [1], can process 8332 concurrent threads over 1472 independent CPUs. At the extreme end of the scale, wafer-scale (as opposed to conventional chip-scale), the Cerebras CS-1 [2] contains a massive 400 k CPUs consuming an enormous area of approximately 46 225 mm^2. Managing and scheduling tasks between so many workers is extremely complex and often requires specialised software tools (e.g. TensorFlow [3]) to analyse the structure of programs and utilise the hardware efficiently. At the other end of the scale, embedded devices often use abstract concepts to enable system designers to comprehend

their systems' behaviour such as hardware events [4], peripheral tasks [5], or, at a fundamental level, communicating sequential processes [6]. As such, system designers apportion their creations into manageable blocks of functionality. For example, a complex piece of software is composed of connected functions, procedures, and data objects. Similarly, the logical circuit of a complex SoC is composed of connected modules which must co-operate in order to perform the desired function. Proper co-operation between these sub-systems is essential, otherwise the system may fail to meet expectations on aspects such as performance, power consumption, scalability, price, and time to market.

Naturally, systems do not always operate in the correct or desired manner, so engineers require methods of monitoring the activity and behaviour of sub-system blocks. For example, in a many-CPU system running several distinct software programs, the designer might have a firm idea that particular groups of CPUs will operate independently, but co-operate within each group. Using a dedicated side-channel for passive monitoring, the designer can observe internal signals to confirm or deny that the CPUs are co-operating or are independent as desired [7].

The majority of a modern SoC is typically digital sequential (clocked) logic, at least in terms of logical complexity. Sequential elements, e.g. flip-flops or latches, take binary logical values on their inputs and produce binary logical values on their outputs. Frequencies of the clocks which drive sequential elements are relatively fast in comparison to those in other fields of science with rates of 2 GHz and higher being common in consumer products. Due to the physical requirements of recording high-bandwidth data, a system designer must be selective about which internal signals they record. These bases of construction mean that data collected from SoCs have several recognisable characteristics: (1) time is discretised by sampling on clock edges; (2) data is sampled in parallel using a common clock; (3) values are binary, either states or events; (4) time-series contain many samples; and (5) the number of sampled signals is relatively small. These informally described characteristics are addressed in detail in the Chapter 2 and Chapter 3.

Investigating the activity and behaviour of signals in SoC data can be a difficult task, because picking through long binary time-series in order to understand their connections is laborious and time-consuming without the appropriate tools. Despite being difficult, properly understanding inter-component interactions is essential for the development of competitive SoCs because the consequences of unintended behaviours can be detrimental to system operation, and thus the product's success.

1.2 Objectives

This thesis is concerned with improving the process of investigation and understanding of complex SoCs in order to aid their development. Chapter 2 discusses the importance of behaviour analysis in complex systems, which entails gaining knowledge and understanding of both the context surrounding key signals, and the relationships between them. Using the outcomes of this work, a SoC designer should be better equipped to understand their system's behaviour by learning the inter-signal correlations which comprise its behavioural structure. Also discussed in Chapter 2, with a detailed expansion in Chapter 3, is the gap in the current literature with regards to how correlations should be usefully quantified. Two further knowledge gaps are identified as a direct consequence: (1) the requirements and practicalities of low-cost hardware are currently unknown for on-chip assistance with real-time correlation analysis; (2) without the knowledge of how correlations should be measured and quantified, effective techniques for their presentation are also uncertain. Thus, three objectives are addressed in this thesis:

1. Methods of measuring and quantifying correlation in SoCs are to be characterised. This requires a model of the nature of SoC signal data.

2. Requirements and practicalities surrounding low-cost on-chip support for correlation analysis are to be examined. Issues around the data collection, system integration, and real-time monitoring are of particular interest.

3. Effective and ergonomic methods of presenting correlation are to be examined with a view to improving behaviour analysis in SoCs.

These three objectives correspond to the contents of Chapter 3, Chapter 4 and Chapter 5 respectively.

1.3 Thesis Outline

Core concepts and key philosophies behind the use of correlation analysis in the development of complex systems are explored in Chapter 2. The fundamental nature of knowledge discovery is described in conjunction with a discussion around the concepts of interestingness and understanding. These concepts provide an abstract basis for an analysis strategy focused on the behaviour and function of SoC signals. The importance of understanding behaviour by the context and details around correlations is explained to provide a background to the reasoning behind this work. Additionally, inconsistencies around the term "correlation"

among different fields of science are described which highlight the need for precise definitions in this work. A discussion on SoC datasets sets the scene for Chapter 3 by explaining why this field of study is different from other popular research topics such as visual feature extraction, speech recognition, and textual sentiment analysis. Practical methods of collecting data from running systems are then described to clarify where Chapter 4 fits in. Lastly, a background on the use of heuristic evaluation is provided to support the approach of Chapter 5 by comparing the qualities of a careful selection of visualisation heuristics.

Chapter 3, based principally on [8], defines the concept of "Binary SoC Data" as, briefly, a relatively small number of very long parallel discrete-time-series binary vectors, which are thought to be somehow related via unknown logical functions. The usefulness of several adapted definitions of correlation are then explored using a Monte-Carlo experiment based on a probabilistic model of SoC structure and operation. Further, the usefulness of counters for compressing event and binary-state data is explored. Pairwise counters which are based on the intersection or symmetric difference are shown to provide significantly more useful information for correlation analysis than counters of single signals.

Results from Chapter 3 are used in Chapter 4 as the basis of a novel hardware device for real-time correlation analysis on pairs of binary signals. A proof-of-concept design is systematically devised with the goal of low-cost high-performance correlation analysis which is applicable to a wide range of scenarios. The method of operation is designed to approximate and mimic the process of an experienced SoC engineer trying to understand how waveform signals are related to each other. The correlator device is first developed as a stand-alone device, then incorporated as part of a larger project to form two case studies. Usefulness of the techniques is demonstrated via examples of cache behaviour in the "OpenPiton+Ariane" project, i.e. a modern large multi-core system running Linux.

Chapter 5 explores visualisation techniques for presenting correlation results with the aim of improving behaviour analysis. Building upon the exploration of existing methods given in Chapter 2, a novel set of techniques is developed based on a circular graphical layout and a particular table arrangement. These techniques are first critically evaluated and described using appropriate heuristics from Nielsen, Shneiderman, Gerhart-Powals, and Forsell and Johansson. In line with the conclusions of Hvannberg et al. [9], the novel techniques are necessarily subjected to gestalt evaluation before using two case studies to demonstrate their utility.

In the final chapter, the thesis is summarised, achievement of objectives is discussed, and recommendations are made for possible directions of future work.

1.4 Notation Summary

Common mathematical notations, such as $\Pr(X|Y)$ meaning "posterior probability of event X given Y", are used where possible. For brevity and precision some additional notation is introduced and used frequently throughout this work. This notation is fully explained at appropriate points in the following chapters and listed here simply as a convenient reference.

f_x	Sequence of samples in the unit interval $[0,1]^n \subset \mathbb{R}^n$, i.e normalised. Assumed to be further restricted to the boolean domain $\{0,1\}^n = \mathbb{B}^n$ unless otherwise stated, i.e. binarised.
$f_x \odot f_y$	Element-wise multiplication of vectors f_x and f_y. Equivalent to element-wise logical-and ($\wedge$) for binary vectors.
$\lvert f_x - f_y \rvert$	Element-wise absolute difference. Equivalent to element-wise exclusive-or ($\oplus$) for binary vectors.
$\mathbb{E}[f_x]$	Expected value (expectation) of vector f_x treated as a random variable. Equivalent to weighted arithmetic mean.
$\mathbb{E}[f_x \lvert f_y]$	Conditional expectation of f_x given f_y.
$\dot{\mathrm{H}}\mathrm{am}(f_x, f_y)$	Statistical metric of correlation based on Hamming distance (symmetric difference).
$\dot{\mathrm{T}}\mathrm{mt}(f_x, f_y)$	Statistical metric of correlation based on Tanimoto coefficient (ratio of union to intersection).
$\dot{\mathrm{C}}\mathrm{ls}(f_x, f_y)$	Statistical metric of correlation based on Euclidean distance (geometric closeness).
$\dot{\mathrm{C}}\mathrm{os}(f_x, f_y)$	Statistical metric of correlation based on cosine similarity.
$\dot{\mathrm{C}}\mathrm{ov}(f_x, f_y)$	Statistical metric of correlation based on covariance and Pearson correlation coefficient.
$\dot{\mathrm{D}}\mathrm{ep}(f_x, f_y)$	Statistical metric of correlation based on the theory of independence.

Additionally, these symbols are used throughout Chapters 3, 4, and 5 to denote various quantities.

m	Number of signal/vectors in a given dataset.
t	Discrete time index.
N	Number of samples in a time window.
n	Window index $\in [0, N-1)$.
u	First time index in a time window, i.e. $n = 0$.
w	Signal width measured in bits. Disparate from $w[t]$ which denotes a window function.
δ	Time offset measured in number of clock cycles. $\delta \in (-\Delta, 0]; \quad f_{x_{\langle\delta\rangle}}[t] := f_x[t+\delta]$
Σ_w	Sum of a window function's coefficients.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Abstract Hypothesis

The fundamental question this research aims to address is how SoC designers can be better supported in the process of building up knowledge and gaining an understanding of a complex system. With the availability of improved learning processes, more resources are available for the development of higher quality products. The interpretation of "quality" is, naturally, dependent on the SoC's intended application, but would typically include a balance of physical size, power consumption, computational latency, and computational bandwidth.

2.1.1 Knowledge Discovery and Interestingness

Fayyad et al. [10, 11] discuss the process of knowledge discovery at an abstract level, with a focus on automated methods instead of manual inspection. Inspecting datasets manually is a comparatively unattractive option because it is slower, more expensive, and more prone to human bias than automated methods. This is particularly true in very large datasets where the sheer volume of data means that a manual comprehensive survey is infeasible. Knowledge Discovery in Databases (KDD) is defined as the process of identifying patterns in a dataset which are valid, non-trivial, potentially useful, novel, and understandable. Requiring a discovered pattern to be valid means that the pattern must be both relevant and statistically significant, as opposed to a simple coincidence. Human bias in manual analysis makes us prone to assigning too much importance to patterns whose occurrence

is merely a coincidence [12] (apophenia and pareidolia), e.g. perceiving faces in inanimate objects. Automated analyses which are immune to pareidolia are therefore desirable in the search for knowledge about qualitative and quantitative features in SoC data. Patterns contributing to knowledge must be derived through non-trivial inference or searching, as opposed to simple or obvious quantities. Fayyad's example is "the average value of a set of numbers"; however, in binary data viewed on a waveform analyser, the average value may not be a simple or obvious quantity. Due to the difference in scale of axes (many values horizontally, only two values vertically) comprehending the average requires an involved search-and-count process. Thus, the interpretation of the "non-trivial" requirement is dependent on the nature of the data.

The properties of usefulness and understandability are noted as subjective qualities which depend on both the context and the viewer. The concept of interestingness is closely related to these properties, and is used to identify and extract pieces of knowledge which the viewer is likely to deem most useful. Hilderman and Hamilton [13] survey 17 metrics of interestingness, classifying them by their foundations and formulations. Each metric is a mathematical interpretation of the term "interesting"; i.e. that a particular data feature should be highlighted for further investigation. A common theme among all formulations is that they are based on the statistical relationship between two or more items of a dataset, expressed in terms of estimated prior and posterior probabilities ($\mathbb{E}[X]$ and $\mathbb{E}[X|Y]$).

Roddick and Rice [14] identify interestingness metrics using the concrete example of the reporting of events in sports, e.g. cricket. Reports of events are classified as either static facts which have already occurred, or anticipated events which are estimated to occur in the close future. Three simple interestingness metrics are given which link an occurrence of X to a potential cause occurring Y in a dataset of size N: (1) confidence = $\mathbb{E}[X|Y]$, (2) completeness = $\mathbb{E}[Y|X]$, and (3) support = $\mathbb{E}[X \wedge Y]$. For example, let X be "Alice scores a point", let Y be "Bob is on the field", and say the confidence $\mathbb{E}[X|Y]$ is high (because Alice has been observed to score more often when Bob is also on the field). A reporter should mention this fact when Bob enters the field because viewers will pay particular attention to Alice in order to confirm or deny their expectation that these two events are related. Sports reporting, like many forms of temporal analysis, bears some resemblance to the investigation of SoC data – A reporter's purpose is to view the game and present a filtered summary of the important events and their context to listeners. The first step in investigating SoC data (or other temporal data) is to identify the most important events based on the context of which events have previously occurred, then condense this knowledge into some sort of summary. By

mentally sorting pieces of knowledge by their interestingness we can prioritise investigation effort. Roddick and Rice also note the importance of non-linear functions, in particular threshold functions, for determining interestingness by selectively discarding uninteresting data rather than simply investigating with lower priority. The use of threshold functions for removing extraneous results is also useful in visual presentations in order to avoid overwhelming a viewer. This is discussed further in Chapter 5.

A recurring theme in the literature on KDD (also known as "data mining") and metrics of interestingness, is that the most valuable information is in the identification of relationships between items.

2.1.2 Behaviour and Function

SoCs comprising modern silicon products are often built on a foundation of lifetimes of work by hundreds of engineers implementing systems of extremely high complexity [15]. The sheer scale of the complexity makes it all but impossible for one systems architect to have a complete understanding of every part of a design. Instead, individual engineers with deep and detailed knowledge of specific components rely on well-defined abstractions and interfaces in order to have confidence that the SoC products using these components will function correctly [16, 17].

The concepts of behaviour and function can be distinguished by a familiar analogy: the function of a waitress, moving food between a kitchen and tables, is different from her behaviour, which may range from fast to slow and from silent to chatty. A restaurant employing a waitress, as a component of the business, is able to easily and explicitly specify the exact function of a waitress, i.e. which plates to transfer to which tables. Behaviour, on the other hand, is a set of qualities difficult to pin down but essential to the success of the business and specific to the restaurant's target market. The scale of complexity in modern SoCs means that, while at first glance a system may appear to be functioning, there may be unintended interactions between system components leading to undesirable consequences such as reduced performance, increased energy usage, information leakage, or unexpected susceptibility to faults.

Cao's exposition on behaviour understanding [18] explores the concept of behaviour thoroughly. The field of behaviour informatics is concerned with the modelling, construction, simulation, presentation, and usage of data about the action or reaction of an entity, human or otherwise, to stimuli in its environment. While Cao's two case studies and most of the accompanying examples refer to analysis of human choices, it is emphasised that the conclusions apply to complex

systems in general. A more abstract interpretation is that the concept of behaviour is about identifiable patterns of relationships between observable events. Further, behavioural data is often recorded only implicitly and therefore requires significant processing to extract features. The example given by Cao is of economic data where only the time, price, and volume of stock transactions is recorded, but patterns indicating human social behaviours (like confidence in a commodity) are implicitly contained.

Wang et al. [19] describe an ontology-based system of analysis and identify that behaviour analysis is fundamentally based upon the identification of relationships and the subsequent uncovering of contentions and co-operation between entities. Two fundamental limitations are identified by Cao: (1) analysis of the exact details of a relationship are not fully supported; (2) analysis of intention is not supported either. Behaviour analysis is shown to be immensely important in the understanding and development of refined business processes, but it does not provide immediately usable results without context. Instead, behaviour analysis is a useful complement to traditional pattern-analysis techniques to "join the dots" between observed trends and the issues driving those trends. In Cao's examples of investigating customer behaviour in relation to business processes, an understanding of the reasons behind favourable (or unfavourable) customer actions enables the business to adapt its methodology in a holistic and considered manner. The first limitation, that the extraction of precise relationship details is not supported, means that behaviour analysis can be used to uncover the presence of a relationship. By highlighting that a relationship exists, the involved entities are marked as more interesting and worthy of higher-priority further analysis. The second limitation, that behaviour analysis cannot uncover intention, means that knowledge can only be extracted if it exists in a dataset (implicitly or explicitly), i.e. intention exists only in the mind of a system's designer.

In the context of SoC development, a simulation might record the times of different events, e.g. Advanced/ARM eXtensible Interface (AXI) transactions on interconnect nodes, to a database for later behavioural analysis. Interactions and relationships between interconnect nodes are implicitly contained in this record, and the ability to extract this behavioural data allows a system designer to gain a deeper understanding. Ultimately, more efficient methods of gaining deep understanding about a SoC allow it to be developed more efficiently, thereby reducing the time (and cost) of development or accelerating the production of higher-quality products.

2.1.3 Correlation

Correlation as a concept, if not the term, has been around as long as humanity itself as it is the core concept behind all learning, and therefore essential to understanding our environment. The term "correlation" has different several meanings, depending on context. For example, in the field of signal processing, the term usually refers to the cross-correlation operation [20, 21] on a pair of ordered sample sets:

$$(f \star g)[n] = \sum_{m} f[m]g[m+n] \tag{2.1}$$

By contrast, the Pearson Correlation Coefficient [22, 23] is the most common interpretation in the field of statistics, which presents the ratio of covariance to the product of standard deviations:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{2.2}$$

Even in the field of machine learning which is fundamentally based on statistics, "correlation" can have a different meaning, i.e. learned posterior probability [24]. These are vastly different interpretations of the term with no bijective mappings between interpretations. For example, while both Equation (2.1) and Equation (2.2) may be calculated for a pair of number sequences, only Equation (2.2) may be calculated for a pair of unordered sample sets; i.e. the index term n of Equation (2.1) requires that the data has a natural ordering but Equation (2.2) does not have such a requirement.

Thirteen ways of interpreting the term are given by Rodgers and Nicewater [25] including several algebraic, geometric, and trigonometric interpretations. Each of these shares the common notion that two things are related but without the requirement that one thing leads to another. In this work, the term "correlation" is used in the broadest and most literal sense to mean a reciprocal relationship between two objects. Different mathematical interpretations which somehow quantify how the extent of a pairwise relationship are referred to as "correlation metrics". Chapter 3 examines six correlation metrics to evaluate their usefulness for the purpose of identifying relationships in SoC data.

2.2 Data Collection

This work is about the analysis of datasets containing information from SoCs, which naturally requires a source of data to evaluate. There are several options for acquiring SoC data:

- Open/publicly available dataset.
- Private/proprietary dataset.
- Collect data from a collection of SoC models.
- Collect data from a collection of real SoCs in operation.
- Don't use a collection of data. Instead, use an abstract model of data.

2.2.1 Pre-existing Datasets

Other fields of research rely on publicly available datasets, e.g. computer vision research often uses the CIFAR dataset [26]. The free availability of this dataset ensures that valid comparisons may be drawn between competing researchers. Additionally, allowing the data to be openly scrutinised ensures that any problems, e.g. biased samples, can be openly discussed and addressed. This option provides the easiest path to high-quality research by allowing researchers to focus solely on the target problem without much up-front effort required to collect data.

At the present time, there are no well-established publicly available datasets suitable for the evaluation of SoC behaviour analysis techniques. This situation is perhaps unsurprising, as silicon design companies are traditionally secretive about the precise construction of their designs, given the high level investment required to create them. In recent years, open-source hardware has become more prevalent, chiefly driven by development [27, 28] around the RISC-V Instruction Set Architecture (ISA) specification [29] and Free Open Source Software (FOSS) tools for Field Programmable Gate Array (FPGA) development [30]. However, while designs for open-source hardware are available, simulation data, e.g. Value Change Dump (VCD) recordings, of these designs running under realistic scenarios are not widely available. Compiling such a dataset is certainly possible, but would require significant collaborative effort to compile a large and varied set of designs with corresponding scenarios, gather data from simulations or instrumented systems, and organise the results.

For sensitive, private, or internal research, a company may choose to compile its own dataset. Care must be taken to acknowledge and address biases which will inevitably appear in data collected from a single perspective without the benefit of open scrutiny. Torralba and Efros [31] describe a simple experiment where a Support Vector Machine (SVM) classifier is trained to estimate from which one of 12 datasets an image originates. The high rate of success for most of their candidate datasets shows that despite the best efforts of the dataset compilers, an

identifiable “signature” indicates fundamental bias in the collection process. For example, the Caltech101 dataset [32] contains 101 categories of image including “cars”; however, a bias in the collection process has meant that most images of cars are from a side-on perspective. This means that models trained on this dataset are likely to struggle with alternative or unusual angles of perspective.

The phenomena of bias in data collection has connotations for privately collected and proprietary SoC datasets. For example, a collection of real-world test cases for a particular company’s own SoCs may have diversity in terms of many different systems; however; if their systems depend upon a common architectural framework, e.g AXI networks [33], then it is difficult to say with confidence that any results apply to systems using different architectural frameworks, e.g. Open Core Protocol (OCP) networks [34].

2.2.2 Collecting SoC Data

SoCs can be simulated at different levels of abstraction including analogue, gate, logical, and functional levels. At the lowest level of abstraction, analogue models of each transistor are executed to simulate specific features of hardware [35]. Digital logic is typically designed and manufactured using a standard library of pre-characterised logic cells [36]. Each cell is designed to implement a specific logical function (e.g. 2-input AND, D-Type Flip-Flop (DFF), etc.) which enables use of a simplified logical model to enhance simulation throughput versus analogue models. A typical modern digital logic system is composed of an arrangement of these cells which are selected and connected in the process of logic synthesis. Systems specified principally in Register Transfer Language (RTL) (e.g. SystemVerilog [37]) permit further performance optimisations for simulation such as modelling an entire 32 b adder circuit with an addition instead of as 32 separate logic functions. At even greater levels of abstraction, functional models may be used for simulation of high-level interfaces such as an instruction set simulator (e.g. Spike [38] for RISC-V a processor).

This work is most closely aligned with data collected at RTL-level simulations. Higher-level functional simulations are not intended to be accurate on a cycle-by-cycle basis and therefore do not contain the necessary timing information for investigation of temporal logic issues. Lower-level analogue or gate simulations certainly contain all of the information necessary for investigating SoC behaviour, but require orders of magnitude more power, memory, and time to execute. RTL simulations contain timing information on the values and transitions of every logical wire, commonly stored in the VCD format [39], and are thus suitable for

behaviour analysis. However, simulations in general are not without their issues. Logical simulations operate inherently more slowly than physical devices, which poses difficulty in working with real-time data.

Lagraa's thesis [40] provides a review of currently available methods of extracting data from real-time SoCs including (1) software instrumentation, (2) dedicated trace interfaces, (3) hardware performance counters, and (4) hardware instrumentation.

Software instrumentation is the insertion of additional instruction sequences to record information about the state of execution as the program progresses. One approach to software instrumentation is to use send data to a dedicated support peripheral [41], and another approach is to store instrumentation data in a memory for later retrieval. In Chapter 5, the second case study uses the first approach to monitor the execution state of a multi-processor system. As Lagraa writes, a notable downside to software instrumentation is that the sequence of executed instructions is necessarily altered, which may affect the validity of conclusions.

Systems which are centred around CPUs may also feature a dedicated interface for tracing the flow of execution [42, 43]. This approach is desirable when visibility of full CPU state is required in real-time operation. While execution trace is useful for understanding every detail of a system's software, other SoC components with more specialised functions are unable to make use of these interfaces.

Performance counters are dedicated hardware components which passively count occurrences of logical events during system operation. This has the advantage of not requiring alterations to software, thus validity of captured data is unquestionable on that basis. For example, Zellweger et al. [44] examine the use of event-based performance counters in multi-core x86 SoCs. Their system is trained to learn which events are most related to specific performance characteristics via analysis of counters – effectively a low-cost high-loss compression of event data. Amortizing over many time windows to view changes in behaviour is identified as a shortcoming in existing approaches where only a coarse-grained view of inter-component relationships is available. In contrast to using many time windows, the approach of this work is to identify correlations, and thus system behaviour, within a single time window.

Performance counters are a specific type, but the hardware instrumentation typically refers to hardware support which enables the extraction of information about particular wires in a SoC. For example, in the development of a FPGA-based system, a small set of wires may be routed to pins which can then be monitored with an oscilloscope. This is particularly useful for observing the precise relative timing of events known to be related, e.g. packet arrived and packet processed.

General Purpose Input Output (GPIO) pins are a limited resource on FPGAs and each pin must be connected to external test equipment, so extracting many events in this way is not generally feasible.

To assist development on FPGA platforms, some vendor-specific tools are available which can extract a small set of binary signals. Xilinx offers ChipScope [45] and Altera (now Intel) offers SignalTap [46] which both make use of otherwise unused Joint Test Action Group (JTAG) pins. While these tools are useful for extracting specific signal values under specific conditions, consideration must be given to the fact that their data is travelling over a single pin; i.e. it is not possible extract an arbitrary number of fast signals over an arbitrary period of execution. For example, a SoC operating at a modest 100 MHz with 50 interesting event signals produces uncompressed data at $5\,\mathrm{Gb\,s^{-1}}$ – an unattainable rate for a JTAG port. With some deeper knowledge of how events are specified, careful configuration of match and trigger logic may enable data from a small set of signals to be sufficiently compressed to allow continuous real-time collection. Although the tools SignalTap and ChipScope are only available for FPGA systems, similar tools exist for Application Specific Integrated Circuit (ASIC) systems, such as UltraSoC's Static Instrumentation and Bus Monitor modules [41, 47].

In the range of hardware options available for extracting SoC event data, none provides explicit support for general correlation analysis. This is identified as a gap in the current knowledge and addressed more fully in Chapter 4 with the design of a specialised device.

2.3 Visualisations for Behaviour

Formal and static analyses of RTL source code, e.g. proofs, visualisation of logical structures, and linting, are immensely useful for understanding static aspects of a system [48], but are of limited use for understanding temporal system behaviour; i.e. simply having access to system source code is not enough to fully understand system behaviour in all but the most trivial cases. Challenges noted by Chen et al. [49] include the scalability of formal and static approaches in terms of computation but also the human part of the process (specification). Engineers, as people, can only comprehend so much about a system before becoming overwhelmed, which is why dynamic analysis is the only practical methodology for understanding complex temporal behaviour [50]; i.e. observation of a system's execution.

People learn mostly by applying analogies with different rules to suit the application [51]. Gentner and Markman [52] in particular argue that the most

effective information to learning about a system is in the relationships between its components. Effective visualisation therefore relies on intuitive rules to convert back and forth between a visual attribute and its meaning. Visual analogies include statements like “Darker is more significant”, “I want to maximise the number of data points on the left”, or “A smoother line is better”. Chapter 5 approaches behaviour analysis through the exploration of visualisation techniques to present correlations in binary SoC data. In order to critically evaluate new techniques, heuristic evaluation is applied and comparisons are made to existing tools and techniques.

2.3.1 Heuristic Evaluation

Heuristic evaluation, as a method of evaluating the usability of visual presentation techniques, has its roots in the field of Human-Computer Interaction (HCI) – pioneered most famously by Nielsen [53, 54] primarily to improve the usability of early websites. Nielsen’s goal in developing a set of heuristics was to enable development of robust and effective user interfaces in a cost-constrained research environment.

N1 Visibility of system status.

N2 Match between system and the real world.

N3 User control and freedom.

N4 Consistency and standards.

N5 Error prevention.

N6 Recognition rather than recall.

N7 Flexibility and efficiency of use.

N8 Aesthetic and minimalist design.

N9 Help users recognise, diagnose, and recover from errors.

N10 Help and documentation.

This set of heuristics has remained popular with HCI practitioners because they are effective for identifying problems in existing interfaces. Where a website has been implemented and must be fixed to improve usability, Nielsen’s heuristics provide a reliable framework for identifying issues. Although Nielsen’s heuristics provide insight into potentially negative aspects, they offer less insight into the especially positive aspects of an interface.

Shneiderman [55] offers a smaller set of heuristics aimed more towards interactive exploratory analysis, based around his mantra: “Overview first, zoom and filter, then details-on-demand.” Seven abstract tasks are given which a user attempts when visually seeking information. Naturally, each of Shneiderman’s tasks should be straightforward, intuitive, and low-effort for the user:

S1 Overview: Gain an overview of the entire collection.

S2 Zoom: Zoom in on items of interest.

S3 Filter: Filter out uninteresting items.

S4 Details-on-demand: Select an item or group and get details when needed.

S5 Relate: View relationships among items.

S6 History: Keep a history of actions to support undo, replay, and progressive refinement.

S7 Extract: Allow extraction of sub-collections and of the query parameters.

Gerhardt-Powals [56] compiled a different set of heuristics (referred to as "cognitive engineering principles") aimed toward producing cognitively friendly interfaces rather than finding problems in existing ones:

GP1 Automate unwanted workload.

GP2 Reduce uncertainty.

GP3 Fuse data.

GP4 Present new information with meaningful aids to interpretation.

GP5 Use names that are conceptually related to function.

GP6 Group data in consistently, meaningful ways.

GP7 Limit data driven tasks.

GP8 Include in the displays only that information needed by the operator at a given time.

GP9 Provide multiple coding of data.

GP10 Practise judicious redundancy.

Her study compared the effectiveness of three interfaces, each using a different design strategy. The candidate interface designed using Gerhardt-Powals' heuristics, i.e. the cognitively engineered interface, was found to enable faster and more accurate interactions. Additionally, the cognitively engineered interface was found to be less tiring to use and generally preferred by most test subjects.

Armed with Nielsen's usability heuristics, Shneiderman's visual-information-seeking tasks, and Gerhardt-Powals' cognitive engineering principles, we have the necessary frameworks to find problems in an interface, predict the user's intended tasks, and construct effective user interfaces. In the closely related field of information visualisation, similar frameworks are used to construct and evaluate novel methods. Amar and Stasko [57] take an abstract philosophical approach to

heuristic evaluation of both user interfaces and information visualisation. Their main conclusions are that careful consideration is required to close the rationale gap and the world-view gap; i.e. users must be able to rationalise about every aspect of a presentation, and that the information shown contains everything they need to make further decisions. These gaps can only be closed by considering representational primacy, whereby the representation must be faithful to how we think about it, and ensure that the construction of a diagram does not contain black-box functions. The world-view gap and the relationship with analytical primacy are not considered applicable to this work because these deal with knowledge at a higher level than a single system. Zuk et al. [58] note in their meta-analysis of heuristic sets, Amar and Stasko's approach depends on subject-matter experts with intimate knowledge of a large corpus similar data. However, in this work it is not assumed that decisions made about one system will necessarily apply to others.

Zuk et al's meta-analysis [58] is used and expanded upon by Tarrell et al. [59] in their search for an optimal set of visualisation-specific heuristics. Four main categories of visualisation heuristics are elucidated (perception, cognition, usability, and interaction) in their search for an effective set of information visualisation heuristics. Forsell and Johansson [60] further refine visualisation heuristics into seven categories and compile heuristics from several sources in a thorough exposition of differences between sets. A total of 63 heuristics are extracted from previous works, including those mentioned above, to perform a meta-analysis aimed at finding the 10 most effective heuristics for information visualisation.

FJ1 Information coding.

FJ2 Minimal actions.

FJ3 Flexibility

FJ4 Orientation and help.

FJ5 Spatial organisation.

FJ6 Consistency.

FJ7 Recognition rather than recall.

FJ8 Prompting.

FJ9 Remove the extraneous.

FJ10 Dataset reduction.

Information visualisation is only one form of user interface, so some heuristics from Nielsen's set are not be generally applicable. The set proposed by Forsell and Johansson is specialised towards information visualisation.

In Chapter 5, visualisations are first evaluated against the less-specific sets from Nielsen, Shneiderman, and Gerhardt-Powals, then against Forsell and Johansson's set. Hvannberg et al's comparison [9] of Nielsen vs Gerhardt-Powals'

heuristics notes that effectively using a set requires considering the set as a whole instead of combining similar-sounding heuristics from different sets. In combining heuristics from different sets based only on their title, meaning and intention are lost and evaluation based on the combination may be flawed. For this reason, the techniques in Chapter 5 are evaluated holistically against different sets instead of combining similar sounding heuristics from different sets.

2.3.2 Existing Tools and Techniques

A digital SoC engineer's most familiar visualisations include logical schematics, floorplans, waveform diagrams, and waveforms analysers. Logical schematics and waveform diagrams are used in specification of low-level logic, and floorplans represent how physical components are used to implement a logical specification. While specification diagrams are useful for understanding low-level details and predicting how a design will respond to specific inputs, they are static figures which provide little help in understanding high-level dynamic behaviour.

Waveform analysers are specialised multi-plot tools, often bundled with logic simulators, for browsing binary signal values over time. The primary data format for storing data from logical simulations is VCD which is defined as part of the first Verilog standard [39]. Additional formats are used by different tools to offer faster data retrieval and smaller file sizes, but the VCD format highlights a key attribute of binary SoC data: Only changes in value are stored because each signal is considered to be a 1 b state machine; i.e. signals are expected to have the same value over many consecutive times, so storage space is reduced by only recording changes. In a simulation running for many millions of cycles, explicitly recording the value of every signal (i.e. in a tabular format) would produce an extremely large file. Consequently, tools working with SoC simulation data effectively require native support for a waveform-specific storage format (e.g. VCD) instead of pre-processing via a supported format (e.g. Comma Separated Values (CSV)). The unwieldy process of creating an extremely large intermediate file thereby inhibits the usability of generic visualisation software for exploratory analysis of binary SoC data.

Waveform analysers often feature several types of analysis on individual signals which are intuitive to digital designers. For example, GTKWave [61] allows plotting a multi-bit signal in binary, hexadecimal, or transforming via Gray code, and VCS [62] allows plotting of simple user-defined expressions. However, visualisation of pairwise information like correlations is not well-supported. Expressions implementing correlation metrics may be written to produce multi-plots, but multi-

plots do not fare well against heuristic evaluation. To demonstrate this point, GTKWave is evaluated using the heuristic sets from Nielsen and Shneiderman for the purpose of correlation analysis rather than the intended use of browsing signal values over time.

Three from Nielsen's set (N1 - Visibility of system status; N8 - Aesthetic and minimalist design; and N10 - Help and documentation) are discounted in this case because they are too subjective. N7 - Flexibility and efficiency of use - is partially supported because the user can plot whatever they desire, but writing many expressions may not be considered efficient. For the same reason N3 - User control and freedom - is fully adhered to. As the user has full flexibility and strange-but-intended expressions cannot be distinguished from mistakes, three heuristics are violated (N4 - Consistency and standards; N5 - Error prevention; and N9 - Help users recognise, diagnose, and recover from errors). In exploratory analysis, N6 - Recognition rather than recall - does not make sense to apply when N4, N5, and N9 are not adhered to. The most significant heuristic, according to Amar and Stasko [57], is about representational primacy; N2 - Match between system and the real world - is interpreted to mean that there should be visual indicators of pairwise relationships. In a conventional waveform analyser there are no visual indications of how the relationship between pairs of plot-lines changes over time.

All seven of Shneiderman's visual-information-seeking tasks require writing cumbersome expressions for correlation analysis, even though all seven heuristics are well-supported for the use case of browsing signal values over time.

In this succinct evaluation of waveform analysers against two sets of usability heuristics, the time-value analysis tool most familiar to digital engineers is shown to be inadequate for correlation analysis. Observing signals in a waveform viewer in order to learn something about system behaviour is therefore difficult and error-prone due to the presentation and format of information.

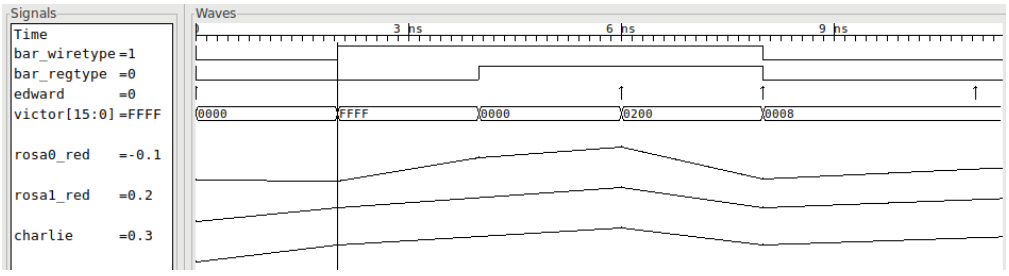
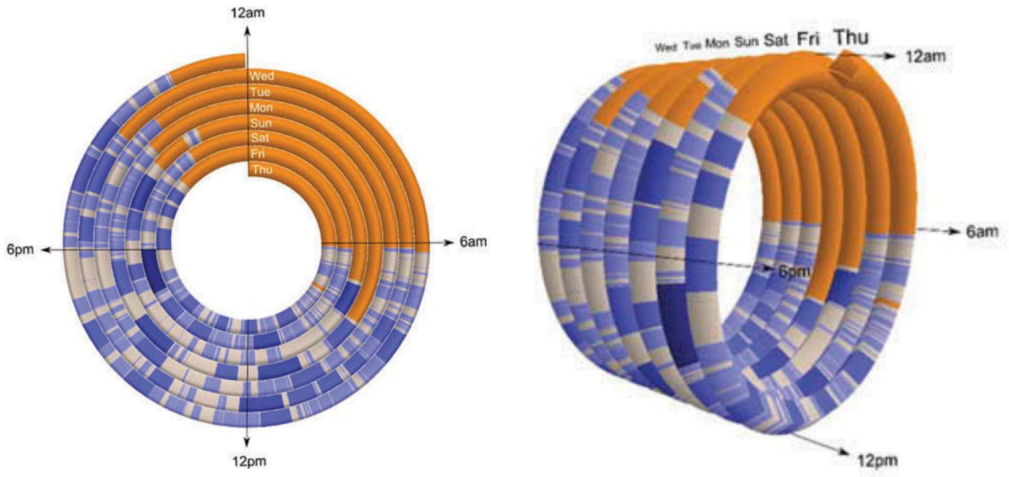


Figure 2.1: Example data of 2-state, event, bit-vector, and quantised-real types displayed using the GTKWave waveform analyser.



(a) (b)

Figure 2.2: Spiral and Helical Visualisations of Periodic Time-Series. Reproduced from Loudon and Granat [64].

In order to squeeze a larger range of time into a single view, Gautier et al [63] exploit the presence of cyclical components by presenting time-series data as a virtual helix or spiral. By contrast, common multi-plots such as waveforms present larger ranges of time by changing the horizontal scale such that nearby value transitions are merged indistinguishably. The example reproduced in Figure 2.2 highlights that when rendering data as a 3D shape, some results are necessarily hidden due to the eventual projection onto a 2D space; i.e. paper or a computer screen. Cyclical compression techniques such as helical diagrams which associate the cyclical and linear attributes of time-series data are best suited to data with strong periodic components as exemplified by Loudon and Granat [64] to display health monitoring data. Healthcare data is, by its very nature, highly periodic as we all require some level of daily routine for sleep and basic interaction with wider society. Use of a helix with an adjustable radius, or the corresponding period, vividly exhibits the daily routine of patients, allowing anomalies to be flagged to the appropriate physician. Health scientists are interested in finding correlations between long time-series of events in order to better understand the behaviour of patients so that they can provide the most appropriate medical interventions. However, helical and spiral timeline compression techniques are limited in their support for presenting more than handful of signals at once because the helix grows too “thick” as signals are added. This inherent limitation in the number of parallel signals means that the spiral and helical techniques are inferior to conventional multi-plot techniques, such as waveforms, for SoC analysis.

Low-level software, including the Linux kernel, often uses the Common Trace Format (CTF) for recording the precise timing of software events [65]. The most common CTF plotting tool Trace Compass [66] is similar in appearance and usability to waveform analysers, as the example shows in Figure 2.3. Kernel trace, and low-level software development in general, is tightly coupled to the field of frontend logic design, therefore it is unsurprising that the tools share commonalities. In the use case of correlation analysis, these commonalities mean that Trace Compass also fails to support the same usability heuristics.

The most relevant type of visualisation common to other fields is the corrgram [67] which maps each element of a correlation matrix to a colour. Figure 2.4 shows an example of a corrgram which highlights two main usability issues in relation to binary SoC data. Each row and column in a corrgram relates to a single variable, and each cell is coloured according to the covariance between its row and column variables.

The first issue is that variables should be ordered such that "similar" variables are adjacent. However, the definition of "similar" is not always readily apparent. For example, the cache interface signals probed in Section 4.4 could be grouped according to either their Network-on-Chip (NoC), tile, or direction, with no grouping having a clear superiority. Friendly and Kwan [68] suggest a pre-processing stage to find an optimal ordering based on Singular Value Decomposition (SVD) of the correlation matrix. This type of pre-processing is not suitable for temporal data because the ordering is likely to change as behaviour changes over time, thereby violating heuristics relating to consistency and visual recognition. As shown on the left of Figure 2.4, using an arbitrary (but consistent) ordering is also an unsatisfactory choice because, as Friendly and Kwan note, patterns of relations among variables are not easily discerned. A further complication with the lack of natural ordering between SoC signals is that a lower resolution

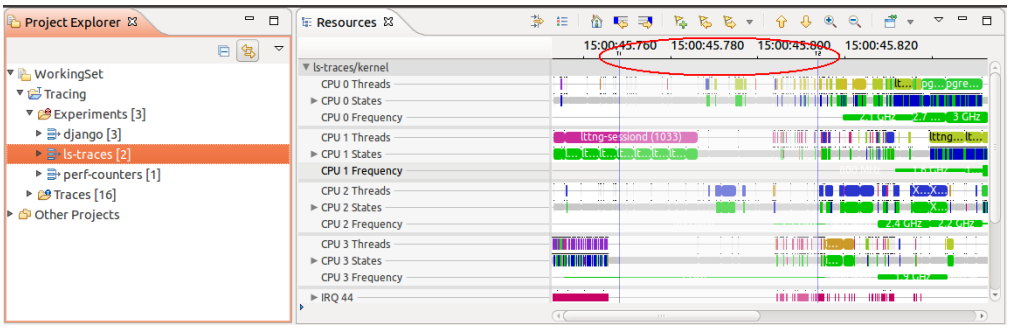


Figure 2.3: Example kernel event data displayed using Trace Compass. Image from `https://www.eclipse.org`.

or zoomed-out view does not convey an overview of the system's correlations. Shneiderman's mantra specifically begins with "Overview first" because having a clear understanding of context is pertinent to understanding the details. Thus, corrgrams violate heuristics related to gaining an overview, maintaining context, and spatial organisation.

The second issue with corrgrams, also known as "correlograms", is that they are inherently 2D; i.e. corrgrams are specialised for rendering matrices, not tensors. The crux of Chapter 3 is that there are several valid interpretations of correlation and it makes sense to use multiple metrics concurrently, e.g. $\dot{\text{C}}\text{ov}$ and $\dot{\text{D}}\text{ep}$. Using n correlation metrics on m signals produces a correlation tensor of shape $m \times m \times n$ for a single time window. Additional ranks for time index and δ offsets compound the issue that correlation analysis in binary SoC data is a problem which cannot be easily visualised with a 2D matrix. Attempting to force high-dimensional analysis into 2D visualisations thus violates FJ1 - Information coding – and Amar and Stasko's notion of representational primacy.

Other novel methods of correlation visualisation such as Zapf and Kraushaar's "Solar Correlation Map" [69] (see Figure 2.5) suffer from the same issues, i.e. a lack of natural ordering or intuitive adaptations to higher-rank data. In the solar correlation map method, the main issue is that one signal must be selected as the "sun", thereby limiting the amount of viewable data and depriving the viewer of an overview.

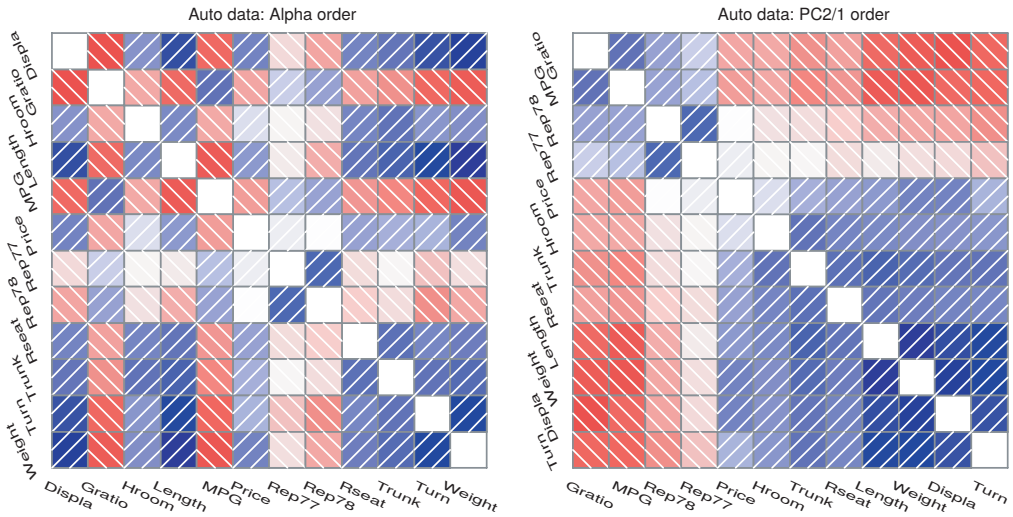


Figure 2.4: Example corrgram visualising correlations between physical measures of auto-mobile models. Correlation values are depicted by colour. Left: Alphabetically ordered variables Right: Variables ordered by angles of the first two eigenvectors. Image from [68].

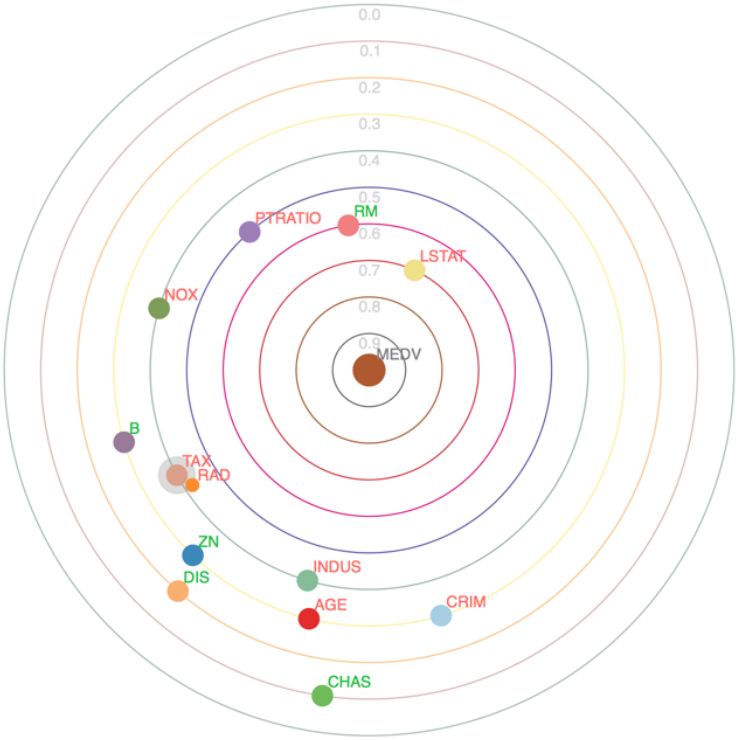


Figure 2.5: Example of the Solar Correlation Map technique from Stefan Zapf and Christopher Kraushaar showing the Pearson correlation coefficient between variables thought to affect the value of homes in Boston. The median value of a home `MEDV` is selected as the variable for inspection, and other variables are seen to "orbit" more closely when they are more closely related. Reproduced from `https://www.oreilly.com/content/a-new-visualization-to-beautifully-explore-correlations/`.

Yeh [70] gives examples of incorporating higher-rank data into a corrgram, primarily by replacing each cell with a sub-plot. Cramming too much into corrgram-based visualisations also violates heuristics related to minimalist aesthetics, error prevention, and gaining an overview. The examples are concerned with chemicals and doses in pharmaceutical products which are, again, static analyses rather than temporal. However, Yeh does hint at the use of graphical (network) structures for exploratory correlation analysis by treating a covariance matrix as an undirected adjacency matrix. It is noted that the representation of a pharmaceutical treatment provides a visual signature derived from correlations between dosage and presence of particular chemicals in subjects' bodies. A parallel can be drawn between the behaviour of a SoC and the behaviour of a subject's metabolism which is noted to be intuitive and invites interactivity.

Graphical techniques are used in other scientific fields because they are intuitive for displaying sets of items and relationships between items. Rendering graphical visualisations in an attractive and usable manner has received attention from Gansner and North[71], and later from Ellson et al. [72]. Well-defined mathematical methods are followed to produce figures which generally evaluate favourably against information visualisation heuristics. Gansner and Koren [73] explore usability issues around circular layouts including longer edges, visual regularity obscuring relevant information. Following paths on circular layouts can also be difficult. Their arguments and proposed relief measures include re-positioning nodes, routing edges outside of the circle, and combining parallel edges. These techniques and their relation to binary SoC data are tackled in Section 5.2, with the argument centred around the balance which must be struck between usability for both static and dynamic behaviour analysis.

2.4 Summary and Direction

SoC designers desire efficient methods to understand their systems, thereby enabling them to focus their development and sharpen their competitive edge. The most valuable type of information when learning about a complex system is its behaviour, which is comprised of the context surrounding key internal signals and their correlations. There is no definitive method for measuring correlation, and different fields of science take diverse approaches. Chapter 3 explores which approaches are best suited to identifying correlations in binary SoC data.

Achieving knowledge of correlations is infeasible using static analyses so observations are made from running systems and collected for further analysis. Chapter 4 explores practical aspects of data collection and calculating correlation in low-cost real-time hardware.

Simply having the calculated values of correlation, using whatever set of metrics is appropriate, is not enough for a human SoC designer. The results must be presented in a useful fashion to facilitate efficient learning. Existing tools and techniques are evaluated using well-known sets of heuristics and Chapter 5 extends this evaluation with proposed techniques for presenting correlation using graphical and table-based models.

CHAPTER 3

CORRELATION METRICS IN BINARY SOC DATA

3.1 Aim

This chapter is concerned with the statistical evaluation of correlation metrics for binary SoC data. Chapter 4 builds upon the results of this chapter to implement practical physical hardware for use in real-time monitoring of correlations between selected pairs of events. In a slightly different direction, Chapter 5 then builds upon the results of this chapter to explore effective visualisation techniques. With a statistical evaluation of correlation metrics, practical hardware to collect and process the required data, and effective methods of consuming the results, SoC designers have the tools required to discover relationships between signals, and thus perform effective behaviour analysis. Material from this chapter has previously been published in [8].

3.1.1 Problem Description

SoC designs include the processors and the associated peripheral blocks of silicon chip-based computers. They are an intrinsic piece of modern computing, owing their complex design to huge volumes of work by many hardware and software engineers. For example, the SoC in a RaspberryPi [74] includes 4 ARM processors, memory caches, a graphics processor, timers, and all the associated interconnect components. Measuring, analysing, and understanding the behaviour of these systems is important for the optimisation of cost, size, power usage, performance, and resilience to faults.

System components are often a mixture of hardware and software which should work in harmony to achieve the designer's goal, and the designer will usually have some idea of how this harmony should work. For example, a designer might like to confirm that software is using cache efficiently by analysing the interaction of events such as `cache_miss` and `execute_someProcedure`. By recording events and measuring inter-event relationships, a system designer can decide if the set of design parameters should be modified, thus aiding the SoC design-optimisation process.

"Correlation" is a vague term which has several possible interpretations [25], including treating data as high-dimensional vectors, sets, and population samples. A wide survey of binary similarity and distance measures by Choi et al. [75] lists 76 methods from various fields and classifies them as either distance, non-correlation, or correlation-based. A similarity measure is one where a higher result is produced for more similar data, whereas a distance measure will give a higher result for data which are further apart, i.e, less similar. The distinction between correlation and similarity can be shown with an example: If it is noticed over the course of many of parties that the pattern of attendance is similar for Alice and Bob, then it may be inferred that there is some kind of relationship connecting them. In this case the attendance patterns of Alice and Bob are both similar and correlated. However, if Bob is secretly also seeing Eve it might be noticed that Bob only attends parties if either Alice or Eve attend, but not both at the same time. In this case Bob's pattern of attendance may not be similar to that of either Alice's or Eve's, but will be correlated with both. It can therefore be seen that correlation is a more powerful approach for detecting relationships, although typically involving more calculation than similarity.

The problem this chapter deals with is that different metrics of correlation are suited to different types of data because there is no one-size-fits-all solution for correlation analysis. For example, in information theory and signal processing, the hamming weight (number of equivalent bits) is often used to quantify the similarity between bit-vectors. Further afield in medicinal chemistry, bit-vectors are used to represent the existence of chemical properties and thereby measure the similarity of different molecules using the Jaccard Index [76] (or the closely related Tanimoto and Sorensen-Dice coefficients). Both fields of study examine data composed of very long parallel sequences of bits which appear similar on the surface but are backed by vastly different meaning. As described in Chapter 2, in order to perform effective correlation analysis and thereby build up knowledge of system behaviour, it is necessary to know which correlation metrics are most effective for SoC data.

3.1.2 Objective

Sampling the voltage levels of many individual wires is typically infeasible due to bandwidth and storage constraints so sparser event-based measurements are often used instead; e.g. Observations like "`cache_miss` @ 123 ns". This gives rise to datasets of very long parallel sequences of bits (occurrence/non-occurrence), so an understanding of how these events are related is key to the design optimisation process. It is therefore desirable to have an effective estimate of the connectedness between bit-vectors to indicate the existence of pairwise relationships.

Given that a SoC may perform many different tasks, its inter-signal relationships may change over time, which means that a windowed or, more generally, a weighted approach is required in a real system. The objective of the experiments presented in this chapter is to characterise simple but effective metrics for estimating the presence of a logical relationship between binary SoC signals. By assuming that behaviour is independent of time, i.e. stationary, a windowed approach is not required in this chapter.

3.1.3 Approach

Relationships between bit-vectors are modelled as Boolean functions composed of negation (NOT), conjunction (AND), inclusive disjunction (OR), and exclusive disjunction (XOR) operations since these fit well with natural language, and this approach has previously been successfully applied to many different system types [77]; e.g. Relationships of a form like "`cache_flush` occurs when `cache_filled` AND `read_access` occur together".

Lo et al. [78] describe a system for describing behaviour with a series of statements using a search-space exploration process based on Boolean set theory. While this work has a similar goal of finding temporal dependencies, it is acknowledged that their mining method does not perform adequately for the very long traces often found in real-world SoC data. Modelling relationships as Boolean functions has been used for measuring complexity and pattern detection in a variety of fields including complex biological systems from the scale of proteins to groups of animals [79].

With the collection of metrics defined and their usefulness measured against a probabilistic data model in Section 3.3, a Neural Network (NN) approach provides an approximation of the upper bounds on their usefulness. A comparison of a collection of input sets and NN parameters against their achievable accuracy demonstrates the insufficiency of performance counters in binary correlation analysis over a single time window. This comparison does however provide clues to

the design of novel correlation counters which are implemented in hardware for real-time analysis in Chapter 4 and software in Chapter 5 for visual presentation.

This chapter provides the following novel contributions:

- A probabilistic model for binary SoC data which allows representative data to be generated and studied on demand.
- An empirical study on the performance of several correlation, distance, and similarity metrics in the use of relationship estimation.
- An evaluation of counter information for use with correlation metrics.

Six different correlation metrics are reviewed and formally defined along with the reasoning behind any deviation from standard formulations. Next, the rationale and methodology behind a probabilistic model of binary SoC data is described and that model is used to generate a set of SoC-like systems which can be simulated. Applying the six different metrics to samples from these generated systems allows the usefulness of these metrics to be empirically evaluated. Further, the use of counter-compatible formulations is explored using a series of NN models to discover which counters provide the most useful information.

3.2 Definition of Metrics

A measured sequence of event occurrences is written as f_i where i is an identifier for one particular event source such as `cache_miss`. Let $f_i[t] = 1$ indicate that event i occurred at time t, and $f_i[t] = 0$ indicating i did not occur at time t. A windowing or weighting function w is used to create a weighted average of each measurement to give an expectation of an event occurrence.

$$\mathbb{E}[f_i] = \frac{1}{\sum_t w[t]} \sum_t w[t] f_i[t] \quad \in [0,1] \tag{3.1}$$

Due to the probabilistic generative approach used in this chapter only the simple case of a rectangular window is required, i.e. $w[t] = 1$. Therefore, the expectation of a vector with N samples can be simplified to $\mathbb{E}[f_i] = \frac{1}{N} \sum_t f_i[t]$ for this chapter.

Defining the intersection of binary vectors as the element-wise multiplication $f_x \odot f_y$ allows Bayes' theorem to be re-arranged to find the conditional expectation.

$$\Pr(X|Y) = \frac{\Pr(Y|X)\Pr(X)}{\Pr(Y)} = \frac{\Pr(Y \cap X)}{\Pr(Y)}, \quad \text{if } \Pr(Y) \neq 0 \tag{3.2}$$

$$\mathbb{E}[f_x|f_y] := \begin{cases} \dfrac{\mathbb{E}[f_x \odot f_y]}{\mathbb{E}[f_y]} & : \mathbb{E}[f_y] \neq 0 \\ \text{NaN} & : \text{otherwise} \end{cases} \tag{3.3}$$

It is not sufficient to look only at conditional expectation to determine if X and Y are related. For example, the result $\Pr(X|Y) = 0.9$ may arise from X's relationship with Y, but may equally arise from the case $\Pr(X) = 0.9$.

A naïve approach might be to estimate how similar a pair of bit-vectors are by counting the number of matching bits. The expectation that a pair of corresponding bits are equal is the Hamming Similarity [80], as shown in Equation (3.4). Where X and Y are typical sets [24] this is equivalent to $|\mathbb{E}[X] - \mathbb{E}[Y]|$. The absolute difference $|X - Y|$ may also be computed on binary data using a bitwise exclusive-OR operation.

$$\dot{\mathrm{H}}\mathrm{am}(f_x, f_y) := 1 - \mathbb{E}\big[\big|f_x - f_y\big|\big] \tag{3.4}$$

The dot above the name in this notation is used to show that this metric is similar to, but not necessarily equivalent to, a standard definition. Modifications to standard definitions include disallowing NaN, restricting or expanding the range to exactly $[0,1]$, or reflecting the result. For example, by reflecting the result of $\mathbb{E}\big[\big|f_x - f_y\big|\big]$ in the definition of $\dot{\mathrm{H}}\mathrm{am}$, a metric is given where 0 indicates fully different and 1 indicates exactly the same. For consistency and valid comparison, all six metrics defined in this chapter are functions of the same form defined by Equation (3.5). These formulations take a pair of equal-length normalised vectors, i.e. each element is between 0 and 1, and each metric produces a scalar result between 0 and 1.

$$(\{\mathbb{R} \cap [0,1]\}^N, \{\mathbb{R} \cap [0,1]\}^N) \to (\mathbb{R} \cap [0,1]) \tag{3.5}$$

$$(\mathbb{B}^N, \mathbb{B}^N) \to (\mathbb{R} \cap [0,1]) \tag{3.6}$$

Most SoC designs are proprietary and commercially sensitive, which means that compiling a dataset of real SoC data is not a viable approach, hence the Monte-Carlo approach used in this chapter. The restriction to binarised data in this research is due to the difficulty in defining a probabilistic model for what normalised data should look like. This research explicitly covers only the restricted case of binarised vectors, defined in Equation (3.6), although metrics have been defined using the more general case to allow for future work finding relationships between binary and non-binary SoC signals. For example, finding the relationship between a binary vector `cache-miss` and a normalised vector `temperature` may be useful in energy optimisation for low-power applications.

3.2.1 Binary Sets

A similar approach to the Hamming Similarity is treating a pair of bit-vectors as a pair of binary sets, as per Equation (3.8). The Jaccard index, first described for comparing the distribution of alpine flora [81] and later refined for use in general sets, is defined as the ratio of the size of the intersection to the size of the union, shown in Equation (3.7). Tanimoto's reformulation [82] of the Jaccard index, also shown in Equation (3.7), was given for measuring the similarity of binary sets.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}, \quad |X \cup Y| \neq \varnothing \tag{3.7}$$

$$\dot{\mathrm{T}}\mathrm{mt}(f_x, f_y) := \frac{\mathbb{E}[f_x \odot f_y]}{\mathbb{E}[f_x] + \mathbb{E}[f_y] - \mathbb{E}[f_x \odot f_y]} \tag{3.8}$$

3.2.2 Geometric Vectors

Treating measured signals as points in bounded high-dimensional space, i.e. each signal comprised of N samples is a point in N-dimensions, allows the Euclidean distance between signals to be calculated. Reflecting and normalizing the Euclidean distance, as shown in Equation (3.9), provides a metric of closeness rather than distance. This approach is common for problems where the alignment of physical objects is to be determined, such as facial detection and gene sequencing [83]. Such analogies between binary vectors of SoC data and physical objects are straightforward for most people to comprehend, making this approach attractive for quickly understanding a dataset.

$$\dot{\mathrm{C}}\mathrm{ls}(f_x, f_y) := 1 - \sqrt{\mathbb{E}\left[|f_x - f_y|^2\right]} \tag{3.9}$$

This formulation is seen to be similar to the Hamming distance, albeit growing quadratically rather than linearly as the number of identical bits increases. Another geometric approach is to treat a pair of measurements as bounded high-dimensional vectors and calculate the angle between them using the cosine similarity as is often used in natural language processing [84] and data mining [85].

$$\mathrm{CosineSimilarity}_{X,Y} = \frac{X \cdot Y}{|X||Y|},\ X, Y \neq 0 \quad \in [-1, 1] \tag{3.10}$$

$$\dot{\mathrm{C}}\mathrm{os}(f_x, f_y) := \frac{\mathbb{E}[f_x \odot f_y]}{\sqrt{\mathbb{E}[f_x^2]}\sqrt{\mathbb{E}[f_y^2]}} \quad \in [0, 1] \tag{3.11}$$

The domain of the measured samples $f_x, f_y \in [0,1]$ means that $\dot{\mathrm{C}}\mathrm{os}$ always gives a non-negative result.

3.2.3 Probabilistic Metrics

The above metrics attempt to uncover relationships by finding pairs of bit-vectors which are similar to each other. These may be useful for simple relationships of forms similar to "`X` leads to `Y`" but less useful for finding relationships which incorporate multiple measurements via a function of Boolean operations such as "`A` AND `B` XOR `C` leads to `Y`". Treating measurement data as samples from a population invites the use of covariance, shown in Equation (3.12), or the Pearson correlation coefficient, shown in Equation (2.2), as a distance metric. The covariance between two bounded-value random variables is also bounded, as shown in Equation (3.13).

$$\operatorname{cov}(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{3.12}$$

$$X, Y \in [0,1] \implies \operatorname{cov}(X,Y) \in \left[\frac{-1}{4}, \frac{1}{4}\right] \tag{3.13}$$

$$\dot{\operatorname{Cov}}(f_x, f_y) := 4\left|\mathbb{E}[f_x \odot f_y] - \mathbb{E}[f_x]\mathbb{E}[f_y]\right| \quad \in [0,1] \tag{3.14}$$

For binary measurements with equal weights, i.e. $\sigma_X = \sigma_Y = \frac{1}{2}$, $\dot{\text{C}}\text{ov}$ is seen to be equivalent to the Pearson correlation coefficient. Equivalence to $\rho_{X,Y}$ exists only for the specific case where the product of standard deviations of X and Y is equal to $\frac{1}{4}$, which is quite specific and unlikely for most SoC signals. A metric exactly matching $\rho_{X,Y}$ is not used in this research because calculating standard deviation, i.e. $\sigma_X = \sqrt{\mathbb{E}[(X - \mathbb{E}[X])^2]}$, requires knowledge of the expectation $\mathbb{E}[X]$ and the signal X concurrently. Requiring both pieces of information concurrently means that, for a hardware implementation all samples from X must be stored, requiring an unviable amount of memory when dealing with long signals.

From the above definition, it can be seen that if two random variables are independent then $\dot{\operatorname{Cov}}(X,Y) = 0$. However, the reverse is not true in general as the covariance of two dependent random variables may be 0. The definition of independence in Equation (3.15) is used to define a metric of dependence.

$$X \perp\!\!\!\perp Y \iff \Pr(X) = \Pr(X|Y) \tag{3.15}$$

$$\dot{\operatorname{Dep}}(f_x, f_y) := \left|\frac{\mathbb{E}[f_x|f_y] - \mathbb{E}[f_x]}{\mathbb{E}[f_x|f_y]}\right| \tag{3.16}$$

Normalizing the difference in expectation $\mathbb{E}[f_x|f_y] - \mathbb{E}[f_x]$ to the unit interval $[0,1]$ allows this to be re-arranged showing that $\dot{\operatorname{Dep}}(X,Y)$ is an undirected similarity; i.e. the order of arguments X and Y is unimportant.

$$\dot{\operatorname{Dep}}(f_x, f_y) = \frac{\mathbb{E}[f_x|f_y] - \mathbb{E}[f_x]}{\mathbb{E}[f_x|f_y]} = 1 - \frac{\mathbb{E}[f_x]\mathbb{E}[f_y]}{\mathbb{E}[f_x \odot f_y]} = \dot{\operatorname{Dep}}(f_y, f_x) \tag{3.17}$$

The metrics defined above (Ḣam, Ṫmt, Ċls, Ċos, Ċov, and Ḋep) all share the same codomain $[0,1]$ where 1 means the strongest relationship. In order to compare these correlation metrics an experiment has been devised to quantify their effectiveness as described in the next section.

3.3 Experimental Procedure

To create the dataset 1000 systems with static behaviour were generated, with $N = 10000$ samples from each signal taken from each system. Systems are constructed as a random graph structure where each node represents a binary SoC signal which is sampled to produce a bit-vector. This procedure is repeated for each metric for each system and the Probability Density Function (PDF) of each metric's accuracy is plotted using Kernel Density Estimation (KDE) to see an overview of how well each performs over a large number of different systems. The number of systems and samples are not particularly special or chosen with great care; i.e. parameters are chosen to be large enough to produce observable trends, but not too large to require excessive computation.

3.3.1 Probabilistic Assumptions

This experiment constructs a large number of SoC-like systems according to a probabilistic structure, then records event-like data from them. Each system has a fixed topology, which means that the actual relationships between bit-vectors are fixed and known. Metrics are then applied to the recorded data and compared to the known relationships, which allows the effectiveness of each metric to be demonstrated empirically.

The maximum number of measurement nodes n_{maxm} is set to 100 to keep the size of systems within reasonable limits. Each system is composed of a number of measurement nodes $f_{i\in[1,m]}$ of either type "src" or "dst", i.e. $m = m_{\text{src}} + m_{\text{dst}}$, arranged in a bipartite graph as shown in Figure 3.1. In each system the numbers of measurement nodes are chosen at random $m_{\text{src}}, m_{\text{dst}} \sim \text{U}(1, \frac{n_{\text{maxm}}}{2})$. Src nodes are binary random variables with a fixed density $\sim \text{Arcsin}(0,1)$ where the approximately equal numbers of high and low-density bit-vectors represent equal importance in detecting relationships on reflected values. The function of each dst node is formed by combining a number of edges $\sim \text{Lognormal}(0,1)$ from src nodes.

There are five types of system which relate to the method by which src nodes are combined to produce the function of a dst node. One fifth of the systems uses only AND operations ($\wedge$) to combine connections to each dst node, another fifth

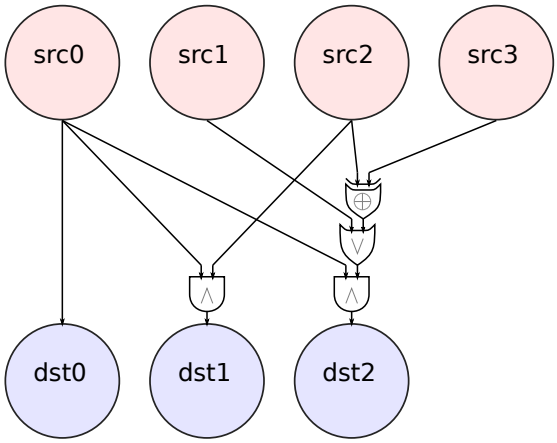


Figure 3.1: Example system with src and dst nodes connected via binary operations.

uses only OR ($\vee$), and another fifth uses only XOR ($\oplus$). The fourth type of system uniformly chooses one of the $\wedge$, $\vee$, $\oplus$ methods to give a mix of homogenous functions for each dst node. The fifth type gets the functions of each dst node by applying chains of operations $\sim \mathrm{U}(\{\wedge, \vee, \oplus\})$ combined via a Left Hand Associative (LHA) rule. By keeping different connection strategies separate, it is easier to see how the metrics compare for different types of relationships.

3.3.2 Further Description of System Construction

(a) A generated system is comprised of m signal nodes, each of which is on either the src side or the dst side. m_{src} is a (uniform) randomly-chosen integer between 1 and 50 (inclusive), and m_{dst} is chosen from the same range. Thus, m is a positive integer, triangularly distributed between 2 and 100.

(b) Nodes on the src side are binary random variables with fixed densities assigned at the time of system creation. A density of 0.1 means that 10% of measured samples have a value of 1 and the remaining samples have a value of 0. Density of a binary random variable must be in $[0, 1]$, and densities of src nodes are picked from an Arcsine distribution, meaning that some nodes will be high-density, around the same number will be low-density, and a lesser proportion will be medium-density.

(c) Connections are made from the src side to nodes on the dst side by logic trees which combine the values of one or more src nodes via a tree of logic operations. The number of connections to each dst node is chosen from a

lognormal distribution with a mean $\mu = 0$ and variance $\sigma = 1$. Therefore, dst nodes with a small number of connections are more common than highly connected ones.

(d) Five types of system are defined to ease the validation of metrics over different types of relationships. The type is chosen uniform-randomly at system construction time. For example, Ḣam is useful for finding relationships based primarily on AND operations, but less useful for relationships based on XOR operations. The "mix" type means that operations comprising a single dst node are all the same, but the operations between different dst nodes are mixed uniform-randomly. The LHA type, as per the example in Figure 3.1, uses chains of binary operations to connect src nodes to dst nodes by complex logic functions.

These four probability distributions comprising the probabilistic model of Binary SoC Data are visualised using their respective probability mass or density functions in Figure 3.2. Naturally, real systems are unlikely to exactly match any probabilistic model. However, in the absence of a large corpus of real SoC datasets this model provides a straightforward method of generating datasets which are similar enough to real SoCs.

3.3.3 Methods of Scoring

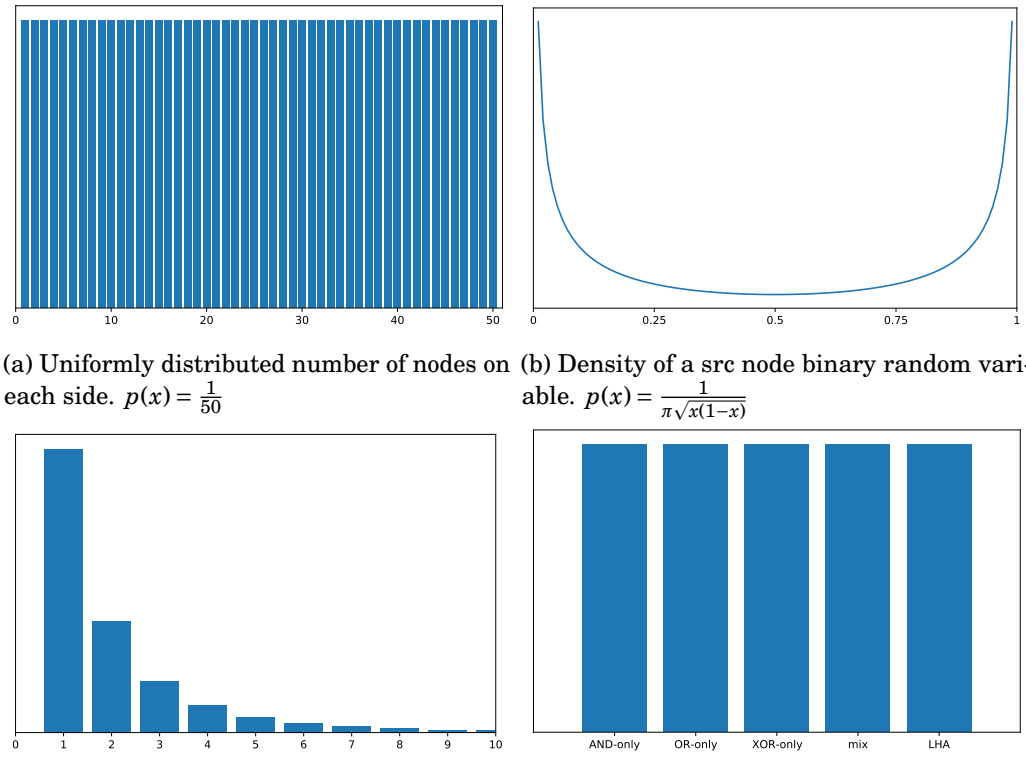
The known relationships were used to construct an adjacency matrix K, whose elements indicate whether a pair of nodes are connected or not, i.e. $K_{ij} = 1$ indicates that node i is connected to node j, and $K_{ij} = 0$ otherwise. The diagonal is not used because these tautological relationships will provide a perfect score with every metric, without providing any new information about the metric's effectiveness. Each metric is applied to every pair of nodes to construct an estimated adjacency matrix E; i.e. the six metrics discussed in Section 3.2 produce six estimated adjacency matrices: $\underset{\dot{\mathrm{H}}\mathrm{am}}{E}$, $\underset{\dot{\mathrm{T}}\mathrm{mt}}{E}$, $\underset{\dot{\mathrm{C}}\mathrm{ls}}{E}$, $\underset{\dot{\mathrm{C}}\mathrm{os}}{E}$, $\underset{\dot{\mathrm{C}}\mathrm{ov}}{E}$, $\underset{\dot{\mathrm{D}}\mathrm{ep}}{E}$. Each element E_{ij} is compared with K_{ij} to give a value of True-Positive (TP) and False-Negative (FN) where $K_{ij} = 1$, or a value of True-Negative (TN) and False-Positive (FP) where $K_{ij} = 0$.

$$\mathrm{TP} = \sum_{i,j} \min\left(K_{ij}, E_{ij}\right) \qquad (3.18)$$

$$\mathrm{FN} = \sum_{i,j} \min\left(K_{ij}, 1 - E_{ij}\right) \qquad (3.19)$$

$$\mathrm{FP} = \sum_{i,j} \min\left(1 - K_{ij}, E_{ij}\right) \qquad (3.20)$$

$$\mathrm{TN} = \sum_{i,j} \min\left(1 - K_{ij}, 1 - E_{ij}\right) \qquad (3.21)$$



(a) Uniformly distributed number of nodes on each side. $p(x) = \frac{1}{50}$

(b) Density of a src node binary random variable. $p(x) = \frac{1}{\pi\sqrt{x(1-x)}}$

(c) Lognormal distributed number of edges, showing $m_{\text{src}} < 10$, adjacent to a dst node. $p(x) = \frac{1}{x\sqrt{2\pi}} \exp\left(\frac{\ln(x)^2}{-2}\right)$

(d) Uniformly distributed choice of system type, by operators used. $p(x) = \frac{1}{5}$

Figure 3.2: Probabilistic model for Binary SoC Data depicted as Probability Mass Function (PMF) or PDF of random variables used to construct each system. Vertical axes are relative likelihood and horizontal axes are value.

For example, if a connection is known to exist ($K_{ij} = 1$) and the metric calculates a value of 0.7, then the True-Positive and False-Negative values would be 0.7 and 0.3 respectively, with both True-Negative and False-Positive equal to 0. Alternatively, if a connection is known to not exist ($K_{ij} = 0$) then True-Negative and False-Positive would be 0.3 and 0.7, with True-Positive and False-Negative equal to 0.

A confusion matrix composed of TP, TN, FP, and FN is constructed for each of the metrics to compare their function as binary classifiers using eight statistics commonly used for this purpose. True Positive Rate (Sensitivity) (TPR) measures the proportion of connections which are correctly estimated, and the True Negative Rate (Specificity) (TNR) similarly measures the proportion of non-connections correctly estimated.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{3.22}$$

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} \tag{3.23}$$

Positive Predictive Value (Precision) (PPV) and Negative Predictive Value (NPV) measures the proportion of estimates which are correctly estimated to equal the known connections and non-connections.

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{3.24}$$

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} \tag{3.25}$$

Accuracy (ACC) measures the likelihood of an estimation matching a known connection or non-connection. For imbalanced datasets ACC is not necessarily a good way of scoring the performance of these metrics as it may give an overly optimistic score. Normalizing TP and TN by the numbers of samples gives the Balanced Accuracy [86], which may provide a better score for large systems where the adjacency matrices are sparse.

$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FN}+\mathrm{TN}+\mathrm{FP}} \tag{3.26}$$

$$\mathrm{BACC} = \frac{\mathrm{TPR}+\mathrm{TNR}}{2} \tag{3.27}$$

Book-Maker's Informedness, also known as Youden's J statistic, attempts to capture the performance of a binary classifier by combining the sensitivity and specificity to give the probability of an informed decision.

$$\mathrm{BMI} = \mathrm{TPR}+\mathrm{TNR}-1 \tag{3.28}$$

Matthews Correlation Coefficient finds the covariance between the known and estimated adjacency matrices which may also be interpreted as a useful score of metric performance.

$$\mathrm{MCC} = \frac{\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}} \tag{3.29}$$

With the scoring methods defined in Equation (3.22) thru Equation (3.29) each metric is then evaluated and compared.

3.4 Metric Scoring Results

Given the large number of systems of various types, it is necessary to take an overview of the scores, rather than focusing on the detail of individual systems. PDF plots are used, and approximated using kernel density estimation to give an overview, as shown in Figure 3.3. Due to the empirical and general nature of the experiment, exact values of the results are unimportant. The most important feature is that more weight on the right-hand side towards 1 indicates a better metric for all eight scoring methods.

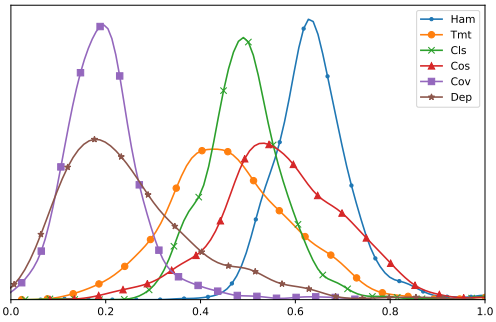
The overall results indicate that Ḣam, Ṫmt, Ċos and Ċls are close to useless for detecting connections in datasets resembling the SoC data model described above. Ċov and Ḋep, based on covariance and the theory of independence, are shown to be generally useful for the desired purpose.

Figure 3.3a shows that Ċov and Ḋep correctly identify approximately 25% of existing connections, and that other metrics identify many more connections. However, Figure 3.3b shows that Ċov and Ḋep are much more likely to correctly identify non-connections than other metrics, especially Ḣam and Ċls.

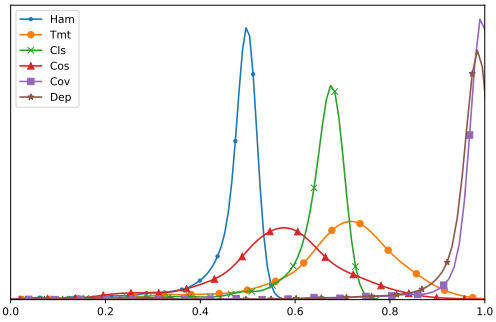
For a metric to be considered useful for detecting connections the expected value of both PPV and NPV must be greater than 0, and ACC must be greater than 0.5. It can be seen in Figure 3.3d that all metrics score highly for estimating non-connections; i.e. when a connection does not exist, they give a result close to 0. This does not carry much meaning on its own because a constant 0 will always give a correct answer. Similarly, a constant 1 will always give a correct answer for positive links. Therefore, the upper four plots must be considered together with the lower four plots to judge the usefulness of a metric.

Results from this experiment provide an imbalanced dataset, because each node has a likelihood of much less than 50% of being connected to each other node. Given that ACC is potentially misleading for imbalanced datasets such as this one it is essential to check against Balanced Accuracy (BACC). Ḣam usually has ACC of close to 0.5, which alone indicates that it is no better than a random guess for estimating the presence of a logical connection. The wider peaks of Ċos and Ṫmt in both ACC and BACC indicate that these metrics are much more variable in their performance than Ċls, Ċov, and Ḋep. In this pair of plots Ċov and Ḋep both have much more weight towards the right-hand side, which indicates that these metrics are more likely to give a good estimate of connectedness.

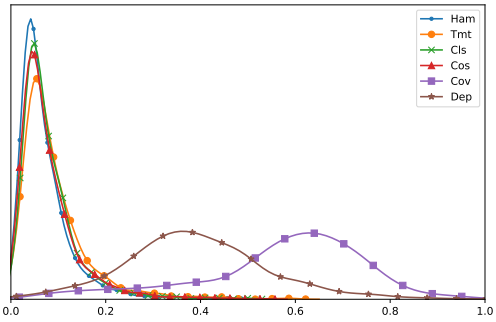
Finally, using Figure 3.3h and Figure 3.3g as checks, it can be seen that Ċov and Ḋep once again outperform the other four metrics. Matthews Correlation Coefficient (MCC) actually has an interval of $[-1, 1]$, although the negative side is



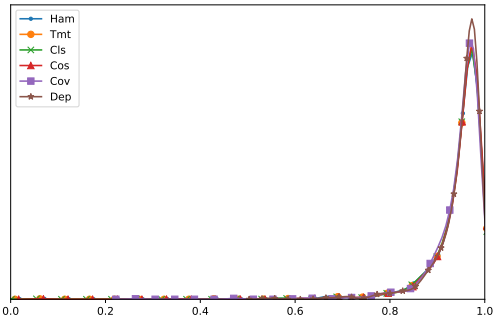
(a) True Positive Rate (Sensitivity).



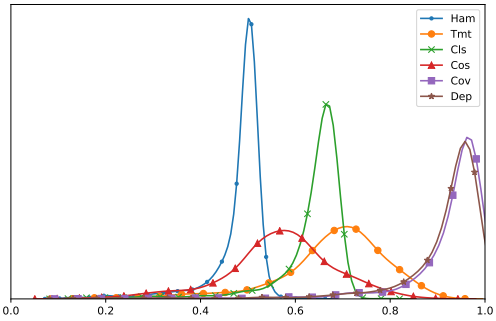
(b) True Negative Rate (Specificity).



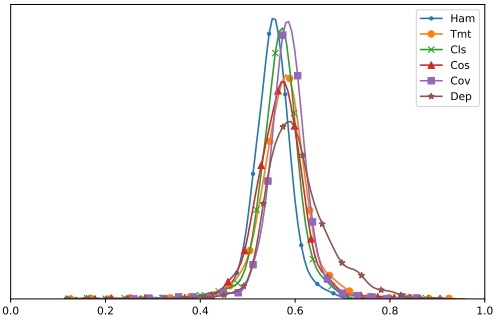
(c) Positive Predictive Value (Precision).



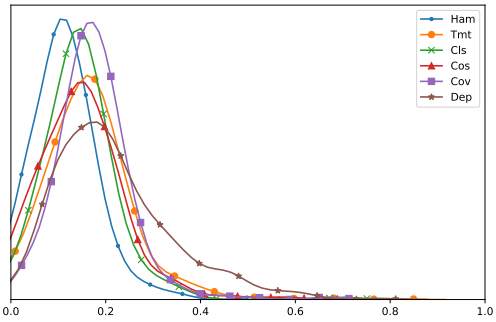
(d) Negative Predictive Value.



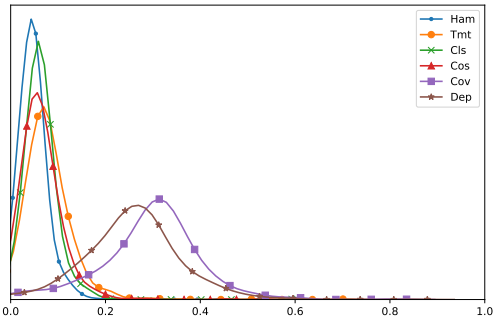
(e) Accuracy.



(f) Balanced Accuracy.



(g) Book-Maker's Informedness.



(h) Matthews Correlation Coefficient.

Figure 3.3: KDE plots of score PDFs averaged across all system types. Vertical axes are relative likelihood and horizontal axes are score result. More weight on the right-hand side is always more desirable.

not shown. All metrics have weight on the positive side, which shows that all six defined metrics contain at least some information on the connectedness.

A characteristic feature employed by both Ṫmt and Ċos is the intersection $f_x \odot f_y$, whereas Ḣam and Ċls employ the absolute difference $|f_x - f_y|$. The best performing metrics Ċov and Ḋep have consistently higher accuracy scores and employ both the intersection and the product of expectations $\mathbb{E}[f_x]\mathbb{E}[f_y]$.

The simplicity of these metrics allows clues about system behaviour to be found efficiently, albeit without further information about the formulation or complexity of the relationships. Any information which can be extracted about a system's inner workings may be used to ease the work of a SoC designer. For example, Chapter 5 describes how results may be presented in an easily comprehensible visualisation. This allows a SoC designer to make more educated choices about their design parameters in order to provide a more optimal design for their chosen market.

Similar results might be obtained algebraically by using the distributions defined in Section 3.3.1, potentially requiring less computing power than the Monte-Carlo method of this experiment. Generating a large number of systems and performing simulations does require a significant amount of computation; however, it is straightforward to alter the model by substituting any of the four distributions. Deriving results algebraically requires significant time, effort and skill, whereas the Monte-Carlo method allows for quick and low-effort alterations. Full results, broken down by system type, are included in Appendix C.

3.5 Learning New Metrics

3.5.1 Calculating Metrics with Counters

In the analysis of a real SoC, only a simulation will provide stored access to the values of every signal over time. Storing and processing the value of a signal at every timeslot is infeasible for real-time analysis because this large dataset must be transferred off-chip, requiring a prohibitively expensive high-bandwidth communication link. Information about signals and their interactions must therefore be compressed. The most common method is using w-bit "performance counters" which count the number of events in a given window of N discrete times, with the assumption that $N < 2^w$. Once sufficient time has passed and the window is finished, the counter value is transferred off-chip for storage and later analysis. The counter process is, effectively, a high-loss low-complexity data compression from N bits to w bits. While other methods of compression based on techniques such as exploiting statistical redundancy or cryptographic hashes may be possible, these are not considered in this research. Counters are the favoured method of lossy compression because of their simplicity, both in theory and implementation, compared to other methods such as Huffmann coding or run-length coding.

All six metrics described in Section 3.2 can be expressed in terms of the expected values of bit-vectors, rather than requiring the raw bit-vectors. This property is convenient because it allows the use of counters instead of requiring long bit-vectors to be stored. Using three counters to represent $\mathbb{E}[f_x]$, $\mathbb{E}[f_y]$, and $\mathbb{E}[f_x \odot f_y]$ allows calculation of $\dot{\mathrm{T}}\mathrm{mt}$, $\dot{\mathrm{C}}\mathrm{ov}$, and $\dot{\mathrm{D}}\mathrm{ep}$. To calculate $\dot{\mathrm{H}}\mathrm{am}$, an additional counter for the symmetric difference $\mathbb{E}[|f_x - f_y|]$ is required, which is equivalent to counting the set bits of $f_x \oplus f_y$. Similarly, $\dot{\mathrm{C}}\mathrm{ls}$ and $\dot{\mathrm{C}}\mathrm{os}$ require additional counters for $\mathbb{E}[f_x^2]$, $\mathbb{E}\left[f_y^2\right]$, and $\mathbb{E}\left[|f_x - f_y|^2\right]$. A full examination of using counters is given in Section 4.3.2 where definitions accommodating normalised data, i.e. in the unit interval $[0,1]$ rather than the set $\mathbb{B} = \{0,1\}$, are important for applying window functions. Briefly, each time a signal expression is observed "high", a counter c is incremented by k; Once the window has completed, the expectation can be computed directly $\mathbb{E}[f_i] = \frac{c_i}{kN}$.

3.5.2 Neural Network Parameters and Setup

Feed-Forward Neural Network (FFNN) models have been proven capable of learning arbitrary functions, given sufficient size, computational power, and training over an appropriate dataset [87]. For example, a densely connected FFNN of sufficient width, depth, and with appropriate activation functions could be trained to represent $\dot{\text{C}}\text{ov}$, or any other metric. Known as the Universal Approximation Theorem, this is used as the basis of a further experiment with the two aims of exploring counter-based correlation metrics on binary SoC data:

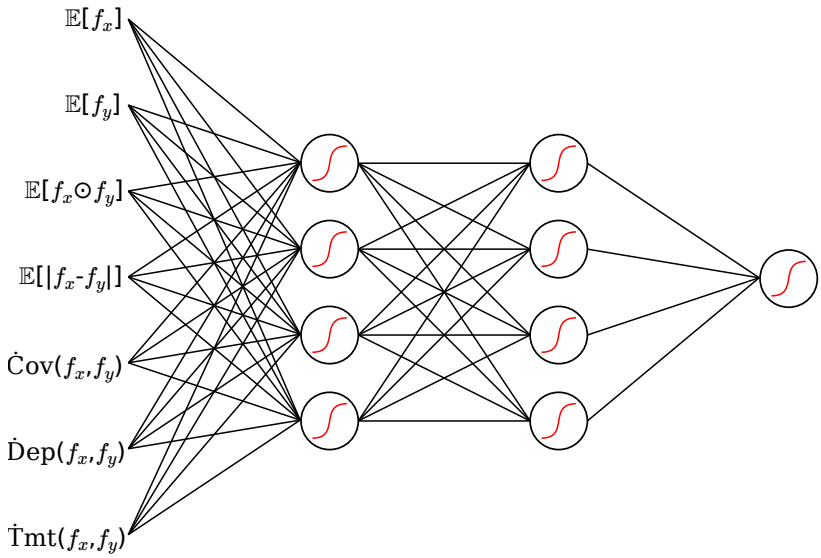
- Determine which kinds of counters provide the most useful information for off-chip analysis.
- Approximate the maximum usefulness of any metric.

A modern Python-based Machine Learning (ML) framework, TensorFlow/Keras, is used to compare the viability of metrics based on specific sets of counter inputs over a variety of NN structures. To avoid over-fitting the models, the structures are kept simple and relatively small with less than 20 neurons. Figure 3.4 depicts two of the FFNN structures used in this series of experiments.

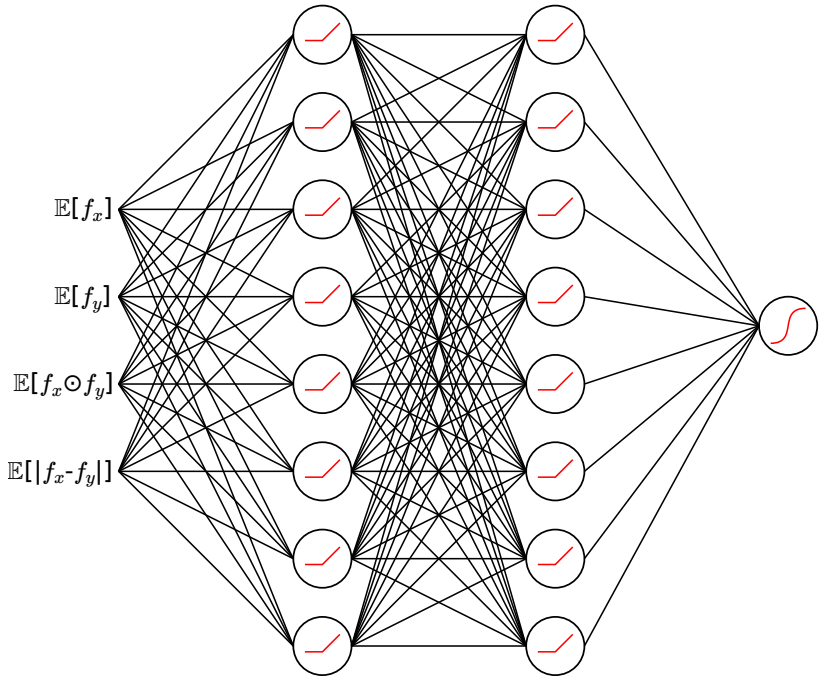
Each FFNN structure is trained, and the epoch loss is recorded over a fixed number of training steps. The small size of these NNs also makes their implementation on-chip a feasible proposition, suggested as a future direction of this research in Section 6.3. In this series of statistical experiments, datasets are generated in the same manner as Section 3.3. These datasets contain adjacency matrices of known and estimated connections, as well as expectations representing various counters.

While not an explicit goal of these experiments, the possibility of implementing FFNN-based metrics in hardware has been kept in mind. Accordingly, the networks are small, comprised of two [1] densely-connected same-size hidden layers of two, four, or eight neurons. The details of neuron structures are listed fully in Appendix D. Each structure is named using a regular scheme based on the first hidden layer, then second hidden layer, then output layer, separated by underscores. Table 3.1 lists all nine structures tested. All structures use a sigmoid activation function on the output layer in order for the output to be continuously differentiable, and hence compatible with the gradient-based Adam [88] learning strategy.

[1] A structure (`2qsig_0_sigm`) of only one layer with two neurons is also tested, demonstrating learning on a tiny configuration with an activation function which is friendly to fixed-point implementations.



(a) Structure is `4sigm_4sigm_sigm`. Combination of inputs is "withAssist".



(b) Structure is `8relu_8relu_sigm`. Combination of inputs is "withIsectSymdiff".

Figure 3.4: Illustrations of FFNN structure and input combinations. Only two of the nine tested structures, and two of the six tested input combinations are shown.

Name	1st hidden layer		2nd hidden layer	
	#Neurons	Activation	#Neurons	Activation
`2qsig_0_sigm`	2	Quantised Sigmoid	0	not applicable
`2qsig_2qsig_sigm`	2	Quantised Sigmoid	2	Quantised Sigmoid
`4qsig_4qsig_sigm`	4	Quantised Sigmoid	4	Quantised Sigmoid
`4sigm_4sigm_sigm`	4	Sigmoid	4	Sigmoid
`4tanh_4tanh_sigm`	4	Hyperbolic Tangent	4	Hyperbolic Tangent
`8qsig_8qsig_sigm`	8	Quantised Sigmoid	8	Quantised Sigmoid
`8relu_8relu_sigm`	8	ReLU	8	ReLU
`8sigm_8sigm_sigm`	8	Sigmoid	8	Sigmoid
`8tanh_8tanh_sigm`	8	Hyperbolic Tangent	8	Hyperbolic Tangent

Table 3.1: FFNN architectures used in metric-learning experiments.

Four activation functions are tested including Hyperbolic Tangent $\left[\tanh(x)\right]$, Rectified Linear Unit $\left[\max(0,x)\right]$, Sigmoid $\left[(1+e^{-x})^{-1}\right]$, and a quantised sigmoid defined in Equation (3.30). This definition of a quantised sigmoid is chosen for its potential simplicity in a hardware implementation using fixed-point arithmetic; i.e. the division by 4 may be performed using a right-shift by 2 operation.

$$\mathrm{qsig}(x) = \max\left(0,\ \min\left(1,\ \frac{x+2}{4}\right)\right) \tag{3.30}$$

The selection of activation functions and layer sizes has been chosen arbitrarily to demonstrate that the learning process on SoC counters is not restricted to a particular NN configuration. Similarly, other hyper-parameters such as the size of datasets and the number of epochs have been chosen as arbitrary round numbers to demonstrate that there is nothing inherently special about the values used. Hyper-parameters to the FFNN learning process are listed in Table 3.2. The first aim of these experiments, to determine which counters are most useful, does not depend on achieving a particular value of accuracy or loss, only that a learning trend is observable. Figure 3.3g shows a most likely Book-Maker's Informedness (BMI) for $\dot{\mathrm{C}}\mathrm{ov}$ and $\dot{\mathrm{D}}\mathrm{ep}$ of around 0.2. Therefore any FFNN which learns how to estimate correlations with a BMI loss of less than -0.2 is said to be more useful. Approximating the maximum usefulness is done by observing the loss value which the learning process trends towards over many epochs. After an initial period of fast learning, epoch loss is expected to "settle" around the value of maximum usefulness for that particular structure and set of inputs.

This FFNN-based experiment trains models to learn an "ideal" metric for estimating the presence of a logical relation between a pair of binary SoC bit-vectors. The learning process attempts to minimise the Book-Maker's Informedness

Parameter	Value
Size of training dataset	4000 systems
Size of validation dataset	1000 systems
Cost function	Book-Maker's Informedness, Equation (3.28)
Learning strategy	Adam [88] ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\hat{\epsilon} = 10^{-7}$)
Number of epochs	100
Number of steps per epoch	100

Table 3.2: Hyper-parameters common to all variations of FFNN experiments.

with loss recorded after every epoch of 100 training steps. Loss is plotted over 100 epochs in Figure 3.5 and Figure 3.6, where substantive learning is indicated by a clear downward trend from left to right, i.e. increasing BMI as training progresses. The lowest steady value of epoch loss provides empirical evidence on the limits of how good a counter-based metric may be.

Name	# Inputs	Inputs
perfCntrs	2	$\mathbb{E}[f_x]$, $\mathbb{E}[f_y]$
withAssist	7	perfCntrs + $\mathbb{E}[f_x \odot f_y]$, $\mathbb{E}[\lvert f_x - f_y \rvert]$, $\dot{\mathrm{C}}\mathrm{ov}(f_x, f_y)$, $\dot{\mathrm{D}}\mathrm{ep}(f_x, f_y)$, $\dot{\mathrm{T}}\mathrm{mt}(f_x, f_y)$
fullAssist	11	withAssist + $\mathbb{E}[f_x \lvert f_y]$, $\mathbb{E}[f_y \lvert f_x]$, $\dot{\mathrm{C}}\mathrm{os}(f_x, f_y)$, $\dot{\mathrm{C}}\mathrm{ls}(f_x, f_y)$
withIsect	3	perfCntrs + $\mathbb{E}[f_x \odot f_y]$
withSymdiff	3	perfCntrs + $\mathbb{E}[\lvert f_x - f_y \rvert]$
withIsectSymdiff	4	perfCntrs + $\mathbb{E}[f_x \odot f_y]$, $\mathbb{E}[\lvert f_x - f_y \rvert]$

Table 3.3: Input sets for FFNN experiments based on counters and pre-calculated assistance values also based on counters.

The same experiment is repeated for different sets of inputs, as listed in Table 3.3. A downward trend in epoch loss for all structures implies that an input set contains the required information for effectively estimating correlation, thereby allowing substantive learning. Absence of a downward trend indicates that additional counters are required for correlation analysis. Analysis using a series of time windows is a separate problem, not covered in this research, because the pair of bit-vectors is effectively replaced by a pair of normalised real-vectors.

While the metrics defined in Section 3.2 are specified to allow use with normalised data, i.e, in the interval $[0,1]$, their effectiveness has not been investigated in this work.

To assure that a FFNN model will learn a metric at least as good as $\dot{\text{C}}\text{ov}$ or $\dot{\text{D}}\text{ep}$, some pre-calculated metrics are given as additional inputs. Summaries of each model are given in Appendix D listing the number of trainable parameters, epoch loss after 100 epochs, accuracy, and mean-squared-error.

3.5.3 Metric Learning Results

The set "perfCntrs", plotted in Figure 3.5a, is used as a baseline because this set is often available and used in real-world situations. For example, to identify whether a particular CPU is affected by memory bottlenecks one might count transactions on the interconnect, beside both the CPU and the memory controller. Counting and plotting the volume of these memory transactions allows the engineer to postulate whether the bandwidth consumed by the CPU is likely to affect other components via a bottleneck effect [44]. Where this approach tells the engineer how much bandwidth is being consumed at particular locations of the interconnect, it does not tell them directly that the CPU is affecting the behaviour of another component sharing the memory; That information is guessed and requires further effort, often by constructing new experiments, to validate the hypothesis. Using a correlation metric gives this information directly, thereby cutting down engineering time and providing value. For a set of m signals, the number of counters required to implement the perfCntrs input set is equal to m. It can be seen in Figure 3.5 that performance counters alone do not contain the requisite information to perform efficient correlation analysis over a single time window. The slightly lower epoch loss in Figure 3.5a corresponding to `8relu_8relu_sigm` does not constitute a significant trend and is attributed to the well-known dying ReLU problem [89].

The next set "withAssist", plotted in Figure 3.5b, adds counters for the intersection and symmetric difference as well as three pre-computed metrics which have simple hardware implementations. Adding more assistance counters to give the "fullAssist" input set shows similar learning rates and a similar stable level of loss. Comparing the plots for withAssist and fullAssist demonstrates that the required information is present in the withAssist set, and adding further inputs only serves to confuse the learning process, as seen by the slower learning rate in Figure 3.5c.

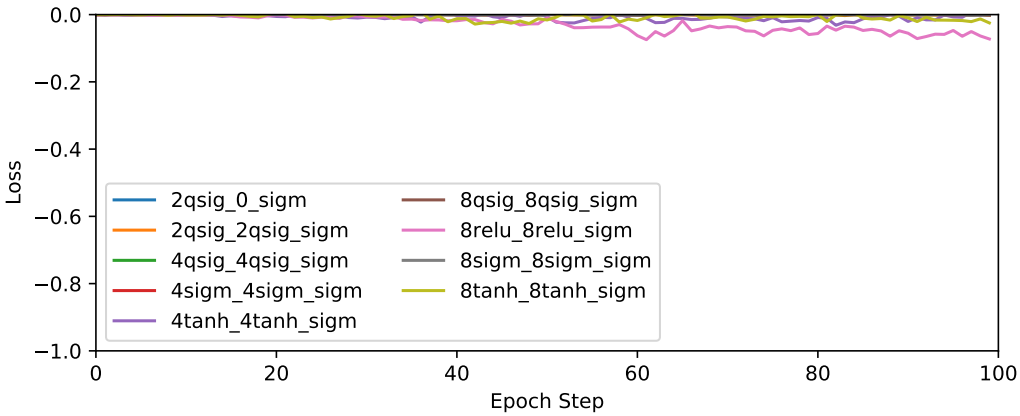
The limit on usefulness is estimated to have a Book-Maker's Informedness of around 0.4 as seen by the level at which the epoch loss levels off for FFNNs using

the withAssist and fullAssist input sets.

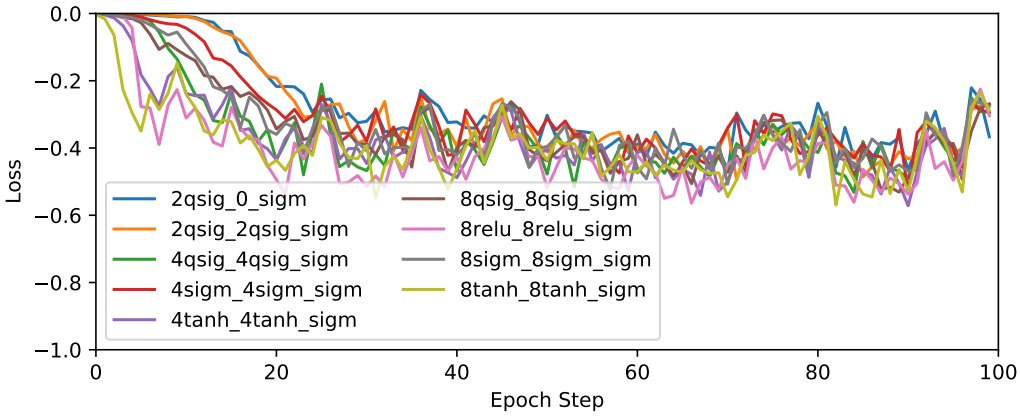
Transferring results off-chip incurs an undesirable real-world cost somewhere along the silicon production line, directly impacted by the efficiency of the transfer process. It is therefore desirable to minimise the amount of data transferred while still acquiring enough to provide compelling analysis. To this end, three input sets, plotted in Figure 3.6, are compared to visualise their contribution to the learning process. Figure 3.6a and Figure 3.6b each adds one counter per pair of signals for the intersection and symmetric difference respectively. For a set of m signals, the number of counters required to perform pairwise correlation with one counter per pair is $\frac{m^2+m}{2}$, including perfCntrs. It can be seen in Figure 3.6 that for analysis with access to either $\mathbb{E}[f_x \odot f_y]$ or $\mathbb{E}[|f_x - f_y|]$ allows learning to occur at a steady rate for most of the tested FFNN structures with the smaller structures learning at a slower pace.

Input from counters for both the intersection and symmetric difference allows all FFNN models to learn at a clear and steady rate, as plotted in Figure 3.6c. For a set of m signals, the number of counters required to perform pairwise correlation with two counters per pair is m^2, including perfCntrs.

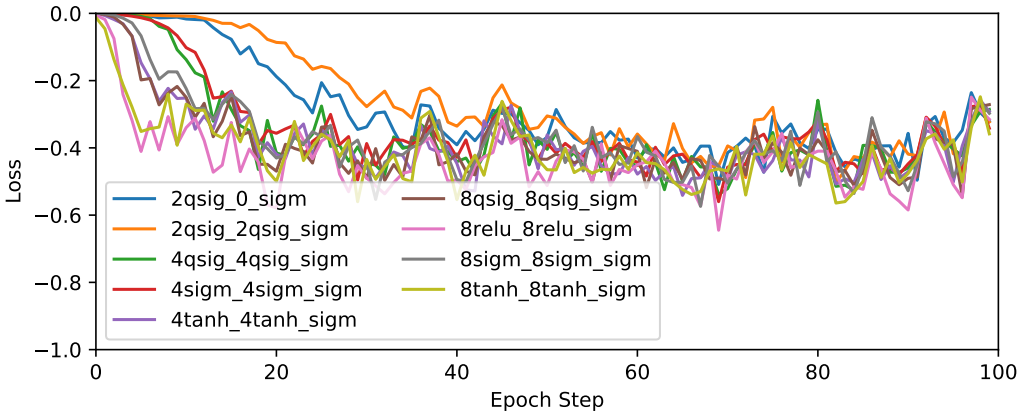
For the smallest structures, the symmetric difference appears to provide more information than the intersection, which is noteworthy because the most useful explicit metrics $\dot{\text{C}}\text{ov}$ and $\dot{\text{D}}\text{ep}$ are instead based on the intersection. The reason behind this is not fully understood, highlighting one of the disadvantages of using a trained-FFNN metric, i.e. lack of explainability. A system designer must therefore decide between a minimal addition of correlation counters $\frac{m^2+m}{2}$, or assigning more resources to implement m^2 counters. The device developed in Chapter 4 includes circuitry allowing off-chip retrieval of both the intersection and symmetric difference.



(a) Performance counters only, i.e. only $\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$.

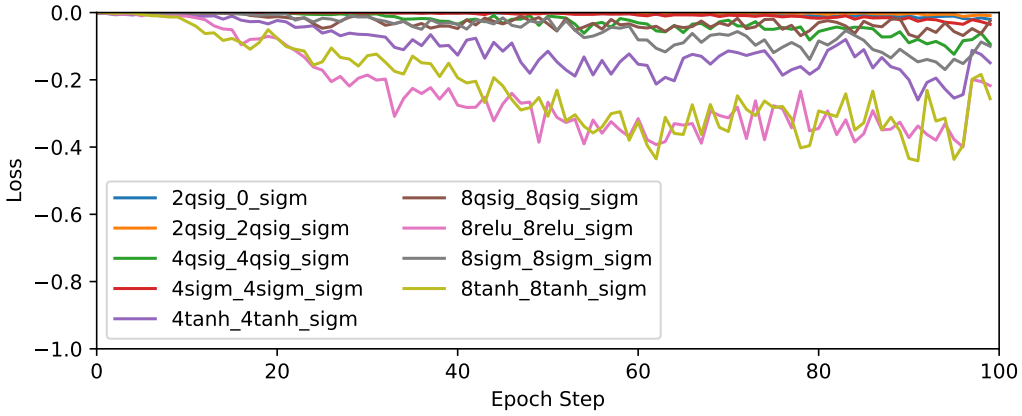


(b) Performance counters, plus some "assistance" inputs $\mathbb{E}[|f_x - f_y|]$, $\mathbb{E}[f_x \odot f_y]$, $\dot{\text{C}}\text{ov}(f_x, f_y)$, $\dot{\text{D}}\text{ep}(f_x, f_y)$, and $\dot{\text{T}}\text{mt}(f_x, f_y)$

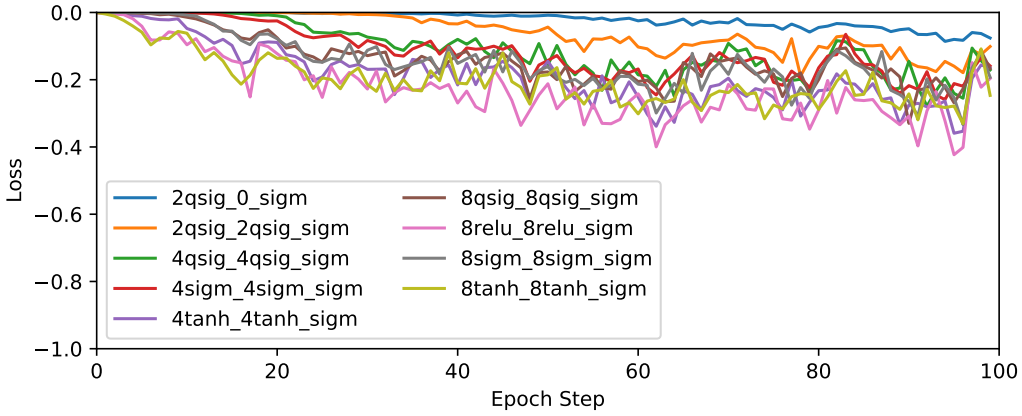


(c) Performance counters, with assistance inputs, plus further assistance inputs $\dot{\text{C}}\text{os}(f_x, f_y)$, $\dot{\text{C}}\text{ls}(f_x, f_y)$, $\mathbb{E}[f_x|f_y]$, and $\mathbb{E}[f_y|f_x]$.

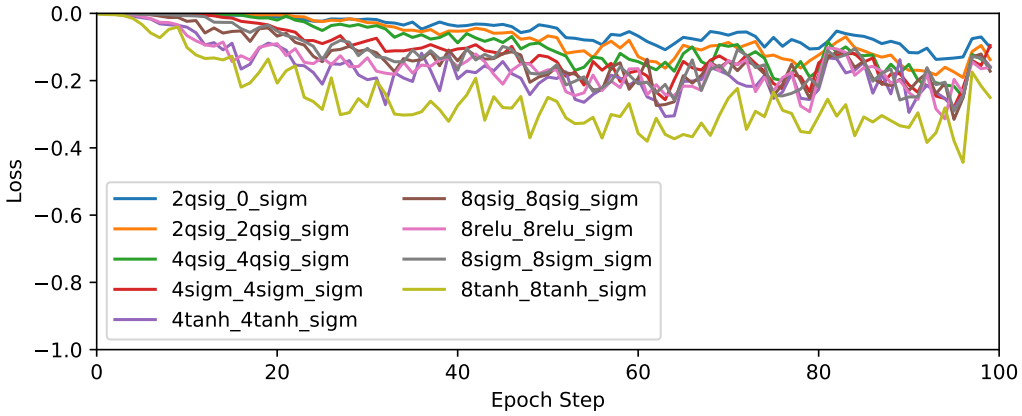
Figure 3.5: Epoch loss over 100 training steps for three different input sets (Part 1 of 2). Each subfigure compares nine FFNN architectures.



(a) Performance counters plus intersection counter $\mathbb{E}[f_x \odot f_y]$.



(b) Performance counters plus symmetric difference counter $\mathbb{E}[|f_x - f_y|]$.



(c) Performance counters plus both intersection and symmetric difference counters.

Figure 3.6: Epoch loss over 100 training steps for three different input sets (Part 2 of 2). Each subfigure compares nine FFNN architectures.

3.6 Conclusion

This chapter has explored a statistical approach to a key area of SoC behaviour analysis using a series of Monte-Carlo experiments. All of the code used to perform these experiments, written in Python, is provided online.[2] Key points and contributions of this chapter are thus:

- A probabilistic model of binary SoC data is defined in Section 3.3.1 to work around the lack of access to a large high-quality dataset.
- Ċov and Ḋep are shown to consistently estimate the existence of logical relationships in SoC-like data with more useful results than metrics used in other fields for correlation in binary data such as cosine similarity or the Tanimoto coefficient.
- Performance counters alone are shown to be insufficient for detecting pairwise logical relationships over a single time window. However, including counters for one or both of the pairwise intersection and symmetric difference is shown to be sufficient for correlation analysis.
- On datasets resembling the given probabilistic model, the maximum achievable Book-Maker's Informedness is found to be around 0.4.

The formulation and rationale behind six methods of measuring similarity or correlation has been given, to estimate relationships between bit-vectors comprising binary SoC data. These formulations may also be applied more generally to bounded data in the range $[0, 1]$, although this is not explored in this work and may be the subject of future work. It has been shown that methods which are common in other fields such as the Hamming distance, Tanimoto coefficient, Euclidean distance, or Cosine similarity are ill-suited to low-cost relationship detection when the relationships are potentially complex. This result highlights a potential pitfall for scientists working with related binary sequences: that of not considering a system's logical construction. Knowledge that the metrics based on covariance (Ċov) and independence (Ḋep) provide the most useful metrics gives confidence that detection systems may employ these approaches to make meaningful gains in the process of optimizing SoC behaviour. By using more accurate metrics unknown

[2] All pieces of code are in a single git repository available at `https://github.com/DaveMcEwan/dmppl`. Methods described in Section 3.2, Section 3.3, and Section 3.5 correspond to the files `dmppl/nd.py`, `dmppl/experiments/relest/relest.py`, and `dmppl/experiments/relest/relest_learn.py` respectively.

relationships may be uncovered, giving SoC designers the information they need to optimise their designs and sharpen their competitive edge.

Various FFNN structures have been presented, composed of different numbers and types of neurons, in order to attempt learning better quality correlation metrics, albeit without explainability. Comparing the learning process using different sets of input data suggests that hardware implementing at least one of the counters for pairwise intersection or symmetric difference can provide system analysts with the necessary information to estimate pairwise correlation. FFNNs have also been used to demonstrate the maximum usefulness of correlation metrics, specified in terms of BMI, i.e. an informedness of around 0.4 when data for $\mathbb{E}[f_x]$, $\mathbb{E}[f_y]$, $\mathbb{E}[f_x \odot f_y]$, and $\mathbb{E}[|f_x - f_y|]$ are available.

Due to the expansive nature of many-to-one logic functions it is clearly impossible to design a metric which can detect every logical connection accurately by relying on a limited number of observations. Results from this chapter have paved the way for hardware implementation of metrics which are shown to be good for the general case. By implementing the suggested counters and metrics which can be derived from their results, direct estimation of the presence of a logical connection is possible from a single window of observation. This approach allows for tracking of fast-changing SoC behaviour which contrasts against the many windows required for using performance counters alone which can only be used for correlation analysis when behaviour changes relatively slowly.

CHAPTER 4

HARDWARE FOR SOC CORRELATION ANALYSIS

4.1 Aim

This chapter is concerned with the practical RTL implementation of the correlation metrics explored in Chapter 3 for the purpose of making tools that can assist in understanding SoC behaviour. State-of-the-art techniques described in Chapter 2 and the mathematical basis for correlation metrics laid out in Chapter 3 are used to guide the development of RTL circuitry in this chapter. Where Chapter 5 deals primarily with visualisation in a simulation context, this chapter focuses on real-time low-cost data collection and analysis in the context of physical systems.

4.1.1 Problem Description

In a simulation context, all values of each bit-vector are expected to be accessible, thus providing all of the required information for software-based analysis. However, in a physical context such as a running FPGA or ASIC, gathering and storing the full data from many binary signals is infeasible due to the implicit cost of providing a capable datalink. For example, monitoring fifty binary signals clocked at a modest 200 MHz requires a datalink and storage medium capable of sustaining at least $10\,\mathrm{Gb\,s^{-1}}$. Data compression techniques may be applicable to some systems, but this is not a trivial endeavour and is likely to incur non-trivial costs. Hardware support for correlation analysis is therefore desirable in at least three scenarios.

The first scenario is to simply collect data required for later (likely software-based) analysis. While general compression of bit-vectors may not be feasible,

the metrics explored in Chapter 3 are compatible with counter-based information. Chapter 3 concludes that only using performance counters is not sufficient to perform correlation analysis; i.e. at least one of $\{\mathbb{E}[f_x \odot f_y], \mathbb{E}[|f_x - f_y|]\}$ is required in addition to $\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$. Hardware support is therefore required to calculate the intersection and/or symmetric difference of each pair of signals in a running system.

The second scenario is real-time monitoring of system behaviour. By their nature, monitors should be passive components which do not affect the operation of the system they are monitoring; otherwise they would be part of the operation. Thereby, a monitor which simply collects data must depend on the receiving equipment to accept all data without negotiation or delay. This is seen as an onerous burden where engineers might expect to use a common desktop computer as the receiving equipment. Further, the analysis latency introduced by encoding, transmitting, and decoding results over a datalink, e.g. Universal Serial Bus (USB) [90], might stretch the total processing time beyond a reasonable definition of real-time. Hardware support offers the ability to process data closer to the source, thus enabling low-latency calculation of correlation metrics.

Embedded analytics, the silicon Intellectual Property (IP) market led by UltraSoC [1] , is centred around on-chip components designed to enable non-intrusive collection and real-time analysis of SoC data [7]. On-chip monitoring solutions are desirable because they can be tailored to the specific application of the main system, allowing behaviour to be analysed and potentially improved throughout the product life-cycle. This work deals with calculating correlation over a single time window, not a sequence of windows. Where existing components feature performance counters which allow for cross-correlation [91], e.g. UltraSoC's Bus Monitor or Status Monitor modules [47, 92], there are no products on the market which currently offer immediate support for the pairwise data required for effective single-window correlation analysis. Some support for pairwise counting is provided by the "match logic" feature of UltraSoC's Status Monitor but, as discussed in subsequent sections of this chapter, further functionality is required for sampling and windowing.

Naturally, these three scenarios are not comprehensive or mutually exclusive so scenarios involving an on-chip module to perform both real-time monitoring and data collection are to be expected. This chapter aims to address these three scenarios by exploring their issues and offering solutions that can be implemented in standard RTL-based toolflows.

[1] UltraSoC are now known as the Tessent Embedded Analytics division of Siemens.

4.1.2 Objective

Real-time correlation analysis allows system engineers to better understand the behaviour of their creations and use interactive methods to experiment with design attributes. Over the course of SoC project development, this allows well-reasoned decisions to be reached more quickly than without the assistance of correlation analysis. Where a large proportion of development time is devoted to verification, validation, and performance analysis, any technique which quickens entry to market is valuable from a business perspective.

The objective of this chapter is to explore the issues surrounding hardware support for correlation analysis of binary SoC signals. Design and implementation issues affecting the scenarios of data collection, real-time monitoring, and on-chip monitoring are addressed in detail. Validation of proposed solutions is provided by two case studies which use the same parent SoC project running on an FPGA.

4.1.3 Approach

Background material, essential to frame the context of particular issues, is covered before the available options to address each issue are explored. The culmination of proposed solutions is the creation of a “correlator” device which is applicable to both data collection and real-time embedded analytics.

The correlator is first developed as a stand-alone tool, with intended usage somewhat similar to an oscilloscope, employing at least two measurement probes to perform real-time correlation analysis. Where an oscilloscope employs one probe to measure and present voltage over time, the correlator employs a pair of probes to measure and present pairwise correlation over time. Next, integration of the correlator into an existing FPGA SoC project is explored to demonstrate usage as an embedded analytics tool. Developing the correlator as a stand-alone device before any integration is a useful approach because characterisation and validation of each component can be performed in isolation. Stand-alone development also ensures that the resulting device is generally useful rather than too specific to one particular target system.

High and low bandwidth data collection scenarios are accommodated by providing the option to consume results from the analogue or digital domains. Two case studies are considered to demonstrate usage of the developed circuitry via implementation of a USB Full Speed (12Mb/s) (USB-FS) gadget, and integration with a commercial embedded analytics toolkit. Due to the high cost of ASIC manufacture, only FPGA platforms are tested, although all RTL work presented is also applicable for ASIC technologies.

4.2 Background

4.2.1 Sampling

As a passive monitoring device, the correlator has no control over when or how often any probed voltage transitions occur. In using an RTL design (digital logic) to measure correlation in binary SoC data from physical wires, the uncertainty around when voltages might transition begets three distinct issues.

The first sampling issue is that of metastability. It is unknown whether probed wires operate synchronously, mesochronously, plesiochronously, or even asynchronously [93]; i.e. each wire attached to the correlator's probes, while assumed to be carrying binary data, operates in a clock domain with unknown properties of period, phase, and jitter. Therefore, the data on each probed wire must be transferred to the correlator clock domain via a synchronisation process. In practical terms, this means that each probe must be directly connected to at least two series DFFs operating in the correlator clock domain, with no logic on the input path to either DFF [17]. With the addition of these two DFFs to each probe, the issue of metastability is satisfactorily overcome; however, a thorough analysis of synchronisation failure modes is given by Dally and Poulton [93].

The second sampling issue is addressed by the Nyquist-Shannon Sampling Theorem [94]. In order to achieve accurate correlation results, samples must be collected at or above the Nyquist rate. Fourier analysis of N samples collected below the Nyquist rate begets frequency aliasing, distorting the result of N complex values. Problems with aliasing arise from the conversion from continuous signals in the physical domain to discrete signals in the digital domain, with artefacts of the reconstruction process most visible, literally, in image processing applications [95]. Fourier analysis produces a vector of N complex elements, whereas a correlation metric produces a scalar real, therefore sampling too slowly simply reduces accuracy rather than introducing a complex response like aliasing. However, the extent of the inaccuracy introduced by sampling too slowly cannot be known by a passive device like the correlator. Henceforth, it is assumed that wires probed by the correlator carry binary voltage levels, transitioning at less than the Nyquist frequency of the correlator sampling clock.

The third, and most significant, sampling issue is due to the nature of digital logic construction. In a practical SoC, it is often desirable to know the relationships between a pair of signals with a small relative time offset; i.e. it is necessary to look slightly forward or backward in time to find a result such as "`X` is likely to occur δ cycles before `Y`". For example, in an AXI interconnect, a read response

event is expected to be highly correlated with a read request event occurring in the recent [2] past because transactions between request and response channels are strictly ordered [33]. Notation $f_{i\langle\delta\rangle}[t] := f_i[t+\delta]$ is used to represent the notion of measurement i being shifted by $\delta \in (-\Delta, 0]$ cycles.

A significant part of the issue is that when asked "Is `X` correlated with `Y`?", an engineer will naturally interpret the question as "Is `X` correlated with $\texttt{Y}_{\langle\delta\rangle}$ for any value of δ in some (implicit) range?". In a simulation context where the full history of each bit-vector is accessible, this is acceptable because the correlation can be calculated Δ times and some or all results returned to the inquirer. However, in a hardware implementation with limited resources, calculating Δ results would require Δ times more computation hardware, which may impose an unacceptable cost. Additionally, circuitry to accommodate calculations over all Δ offsets requires, at a minimum, Δ bits of state. The range of δ which would be considered reasonable is dependent on the context of the signals being probed. For example, a low-level signal expected transition at close to the Nyquist frequency might be searched for correlation with other low-level signals delayed by up to ten cycles, but a high-level signal with fewer transitions might be expected to correlate with signals delayed by up to a thousand cycles. The difference in what might be considered reasonable means that the naïve approach of computing correlation for many values of δ is not a feasible proposition for a real-time hardware device.

The approach taken for the correlator device is to purposefully introduce a configurable amount of jitter into the sampling clock in order to statistically combine multiple delayed versions of a signal $(f_{i\langle-\Delta+1\rangle}, \cdots, f_{i\langle-2\rangle}, f_{i\langle-1\rangle}, f_{i\langle0\rangle})$ into a single signal for consideration. Naturally, combining delayed versions of a signal into one introduces some inaccuracy to the calculation. Acknowledging some inaccuracy in correlation results is acceptable in the search for logical relationships, because knowing that a relationship may exist at all is more important than knowing the exact level of correlation; i.e. results from the correlator device are intended to guide an exploration of SoC behaviour rather than definitively state exact values of correlation. Details of how this jittery sampling process can be modelled and implemented are covered in Section 4.3.1.

[2] The definition of "recent" is dependent on the module, SoC, and application running on the AXI interconnect. A simple peripheral such as a timer might respond within five cycles, but a complicated peripheral such as a matrix multiplication accelerator might be reasonably expected to respond within five hundred cycles.

4.2.2 Window Functions

Window functions are used in Digital Signal Processing (DSP) applications for a variety of reasons including: (1) Reduce spectral leakage with Fourier analysis [96], (2) Convert a known probability distribution to a different distribution [97, 98], and (3) Focus on a particular region of data; i.e. perform a weighted analysis [99].

The simplest window function, rectangular, is defined in Equation (4.1) using notation consistent with modern texts on Fourier analysis [20]. N represents the length of a window in number of samples which is often a power of two. Non-negative integer t represents a discrete point in time, u is the first value of t in a particular window, and n is the index within a particular window (starting at zero). The entirety of time is discretely indexed by t, and a particular window spans from the point where $t = u$ up to, but not including, $t = u + N$. Other popular window functions include the power-of-sine family shown in Equation (4.2) which includes the rectangular ($\alpha = 0$), sine ($\alpha = 1$), raised-cosine ($\alpha = 2$), and alternative-Blackman ($\alpha = 4$).

$$\underset{\text{rect}}{w[n]} = 1 \tag{4.1}$$

$$\underset{\sin^\alpha}{w[n]} = \sin^\alpha\left(\frac{n\pi}{N-1}\right) \qquad \in \mathbb{R} \cap (0,1] \tag{4.2}$$

In Fourier analysis applications, window functions are commonly used to address spectral leakage by focusing a signal's energy into the fundamental frequency component rather than frequency-domain sidelobes [100]. Non-rectangular window functions are typically bell-shaped in the time domain, where the exact construction of the bell curve has a profound impact on results in the frequency domain. For a given sample rate, a longer window (with more samples) provides a higher-resolution view of the frequency domain. However, the energy in each frequency component must be assumed to be constant over the entire window. In order to detect frequency components which change relatively quickly, either the sampling rate must also be relatively fast, or a shorter window must be used with the corresponding loss of resolution.

In the fields of image processing and photography, using bell-shaped window functions is referred to as foveation and may be used as a data compression technique or to encourage the viewer to look at a particular area [101, 102]. In correlation analysis a window function can be applied to focus attention onto a particular region of time. The corresponding interpretation of bell-shaped window functions is that the further away from the centre a sample is taken, the less information it contributes to the analysis. Intuitively, applying a window function

to time-series data is equivalent to the human process of paying more attention to samples gathered around one point in time.

Chapter 3 explores the use of several correlation metrics, all of which are functions of expected values, not the bit-vectors directly. Using metric constructions which only use expected values, not bit-vectors directly, allows these metrics to be calculated with counting techniques; i.e. for each time an event occurs, an accumulator is incremented by an amount dependent on the time of occurrence. This is important because storing a bit-vector as a weighted sum is effectively a lossy data compression, thereby allowing computations to scale in a practicable manner with N. As such, counters are used widely in SoC analysis which are often referred to as "performance counters" [47, 29, 103]. However, experiments in Chapter 3 showed that in order to effectively measure correlation over a single time window, it is insufficient to use only a counter for each f_i; At least one pairwise counter such as $\mathbb{E}[f_x \odot f_y]$ or $\mathbb{E}[|f_x - f_y|]$ is required for pairwise correlation analysis.

Two further definitions are useful to describe the use of window functions precisely and concisely.

$$\Sigma_w := \sum_{n=0}^{N-1} w[n] \qquad \in \mathbb{R} \tag{4.3}$$

$$\mathbb{E}[f_i]_{|_{t\in[u,u+N)}} := \frac{1}{\Sigma_w} \sum_{t=u}^{u+N-1} w[t-u]\, f_i[t] \qquad \in \mathbb{R} \cap [0,1] \tag{4.4}$$

Equation (4.3) defines a convenient notation Σ_w to represent the sum of all coefficients in a window function. For example, for a rectangular window where all N coefficients are equal to 1, the sum of all coefficients $\Sigma_w = N$. The (windowed) expectation $\mathbb{E}[f_i]$ is calculated by summing coefficients where elements of the bit-vector f_i are asserted, shown in Equation (4.4). The first term in Equation (4.4) ($\frac{1}{\Sigma_w}$) is constant for a given window function and serves to normalise the result $0 \leqslant \mathbb{E}[f_i] \leqslant 1$.

Section 4.3.2.1 describes a hardware implementation technique which enables subsequent processing of the counter value to proceed without needing to be scaled according to N. This attribute is important because it enables the correlator device to adjust the transmission rate of results by only sending the most significant bits; i.e. the need to send the value of N is avoided. Section 4.3.2.2 builds on the same principle and describes how choosing a coefficients such that Σ_w is a power of two enables efficient hardware implementation of a non-rectangular window function.

4.2.3 Synthesis, Layout, and Characterisation

A digital RTL module intended for use as an on-chip monitoring component for a wide variety of parent systems must be able to be implemented on a wide variety of FPGA technologies. The process which takes a set of RTL design files and produces an FPGA configuration is comprised of two tightly coupled steps: synthesis, then Place and Route (PnR). The function of the synthesis step, taking RTL code and technology library as inputs, is to create a netlist which is composed exclusively of cells from the technology library connected such that the logic in the RTL code is implemented. The PnR step, also called layout, takes the netlist, a set of timing constraints, and a model of the target FPGA's structure as input. An FPGA configuration, also called a bitstream, is created via the PnR tool finding a way of assigning cells in the netlist to physical cells of an FPGA. Different FPGA technologies are therefore supported by construction of an appropriate technology library and structural model. Converting an RTL design to a functional FPGA configuration is a complex process, highly dependent on the RTL being suitably constructed for the target FPGA. To enable implementation on multiple FPGA technology families, RTL designs must be carefully written to avoid being limited by the constraints inherent in each technology. There are three inter-related types of constraints which should be avoided when creating a technology-portable child design that integrates easily with a parent system.

(1) Technology constraints. If a design requires resources specific to one particular technology, e.g. a specific memory macro, then that design cannot be implemented on other technologies until further redesign work is undertaken. For example, an RTL design initially created on a Lattice iCE40 FPGA might wish to make use of the "SB_GB_IO" primitive for its pad-adjacent logic and DFFs; however, implementing the same design on a Xilinx FPGA is not possible unless an equivalent component is created.

(2) Resource constraints. If a design requires too much of resource such as logic cells (often synonymous with Look Up Tables (LUTs)) then they cannot be used by the parent system. For example, the largest member of the Lattice iCE40LP family [104] of FPGAs has 7680 logic cells (each comprised of a 4-LUT + 1 DFF), whereas members of the Xilinx Virtex-7 family [105] have up to 1955 k logic cells (each comprised of a 6-LUT + 2 DFFs). If an FPGA is not able to provide enough resources to logically implement both the parent and child designs, then the integration is not possible.

(3) Layout constraints. In order to operate digital clocked logic in the usual manner, there are timing constraints which govern how physically far apart logi-

cally adjacent cells may be placed. Where logic is clocked at a high rate, timing constraints must be tightened to check that voltage from the output of one DFF can propagate through appropriate combinatorial logic cells to the input of another DFF within a specific time-frame. Tight timing constraints therefore require that some cells are placed within a certain distance of each other; i.e. there are constraints on the layout of cells. Layout constraints are, naturally, more difficult to satisfy when a design requires a higher proportion of available resources. If a combined parent and child design is constrained such that no layout can be found to meet all timing constraints, then the system cannot function at its intended clock frequency.

Therefore, to avoid synthesis issues in integrating a child component with a parent system an ideal child design should use only generic logic constructs, require few logic resources, and have loose timing constraints. By aiming toward these design goals, a SoC engineer can expect the integration to pass through an FPGA toolchain and produce a successful configuration.

FPGAs and ASICs can be manufactured using a variety of different process technologies; Some allow operation at higher clock frequencies, some offer lower power characteristics, and others can be manufactured cheaply. For example, Lattice manufacture their own chips using a 40 nm process for their iCE40LP family with focus on low-power [104], whereas Xilinx has their 7-Series family manufactured using TSMC's 28 nm for their focus on high-performance [105]. It can be reasoned that if a design can be successfully implemented on an FPGA technology which is relatively restricted in terms of logic resources and clock frequency, then implementation on a more advanced FPGA technology is also likely to be successful. Intel have famously used similar reasoning in their "tick-tock" development model for their CPUs [106] where each "tick" took advantage of a process shift to implement a die shrink, giving power and performance benefits with change to the logic design.

PnR is often a non-deterministic process, instead based on randomised algorithms such as simulated annealing [107, 108], which means that multiple solutions are possible. Appendix E fully describes the method used to characterise an RTL design in relation to the synthesis and layout process for the low-performance Lattice iCE40, based on comparison of different PnR solutions. Based on the hypothesis that a process shift to a higher performance FPGA technology can be expected to have looser timing and layout constraints, the correlator device has been initially developed and characterised on the low-performance Lattice iCE40LP FPGA technology.

4.3 Correlator Device

As explained in Section 4.1.3, the issues surrounding hardware support for correlation analysis of binary SoC signals are explored via the design of a passive monitoring tool, the correlator.

The purpose of this device is to enable correlation analysis of binary signals in running systems using the metrics explored in Chapter 3. Basic setup of the correlator is intended to be similar to that of an oscilloscope or logic analyser. Instead of one probe for measuring voltage over time, each correlator "engine" uses two probes for measuring pairwise correlation data over time. In Figure 4.1, a correlator with a single engine is shown, controlled by a host computer over USB, and the correlation between the probed signals is presented in the brightness of an LED.

Where Figure 4.1 depicts a stand-alone correlator device, the alternative intended usage scenario is an integrated on-chip monitor suitable for embedded analytics. In an integrated scenario the probes are connected directly to potentially interesting signals within the RTL code. To enable flexibility in choosing which pair of RTL signals each correlator engine can monitor, the set of probes are routed

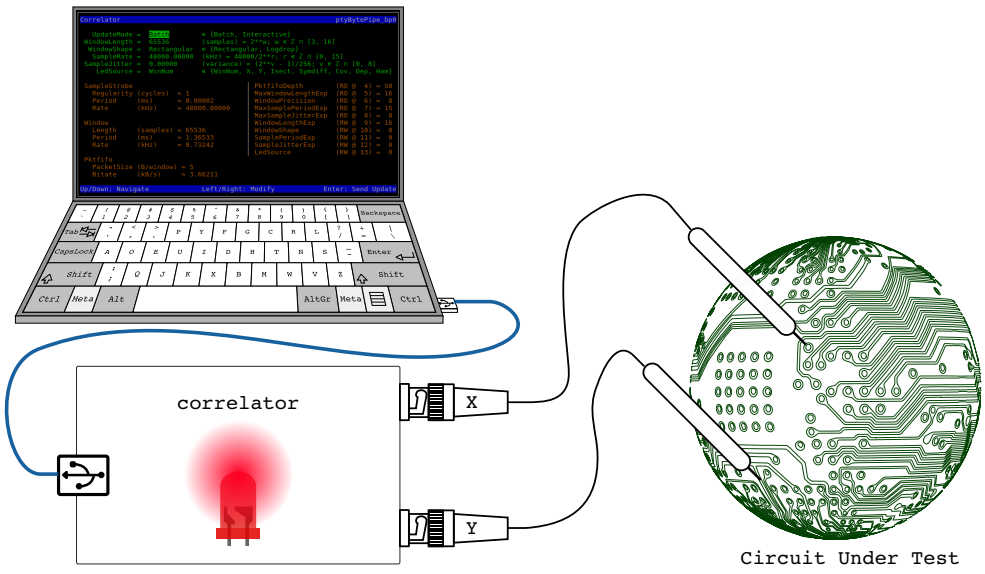


Figure 4.1: Usage of stand-alone correlator device. A host computer is used to configure the device and retrieve data over a USB-FS link. A pair of probes are placed on signals in the circuit under test. Results may be obtained using the Light Emitting Diode (LED), an oscilloscope (not shown), or retrieved for offline analysis.

through a set of multiplexers, a crossbar. This enables each engine to dynamically select which pair of probed signals to monitor. For example, an implementation built with 3 probes (x, y, z) and 2 engines could be configured at runtime to calculate $\dot{\mathrm{C}}\mathrm{ov}(f_x, f_y)$ and $\dot{\mathrm{D}}\mathrm{ep}(f_x, f_z)$. Each engine contains independent circuitry to facilitate: (1) jittery sampling, (2) counting binary assertions with a pre-applied window function, (3) low-latency calculation of $\dot{\mathrm{C}}\mathrm{ov}$, $\dot{\mathrm{D}}\mathrm{ep}$, and $\dot{\mathrm{H}}\mathrm{am}$ metrics, and (4) recording necessary data for offline analysis.

Figure 4.2 shows an overview of a correlator engine's microarchitecture to set the scene for further sections which detail the reasoning and function of the main sub-systems. Section 4.3.1 covers how signal data enters the monitor via a jittery sampling mechanism which works in conjunction with the windowing counters covered in Section 4.3.2. The need for a specialised construction of counters is rationalised by the requirement of flexibility in a tool used in a variety of systems. Two solutions for obtaining the counter data required for correlation analysis are described in Section 4.3.3; One for transferring data to the host computer for external analysis, and another for low-cost real-time correlation monitoring through the use of a $\Delta\Sigma$ modulator. In line with the reasons set out in Section 4.2.3, the RTL design is characterised for implementation on the Lattice iCE40LP then the Xilinx 7-Series FPGA technologies.

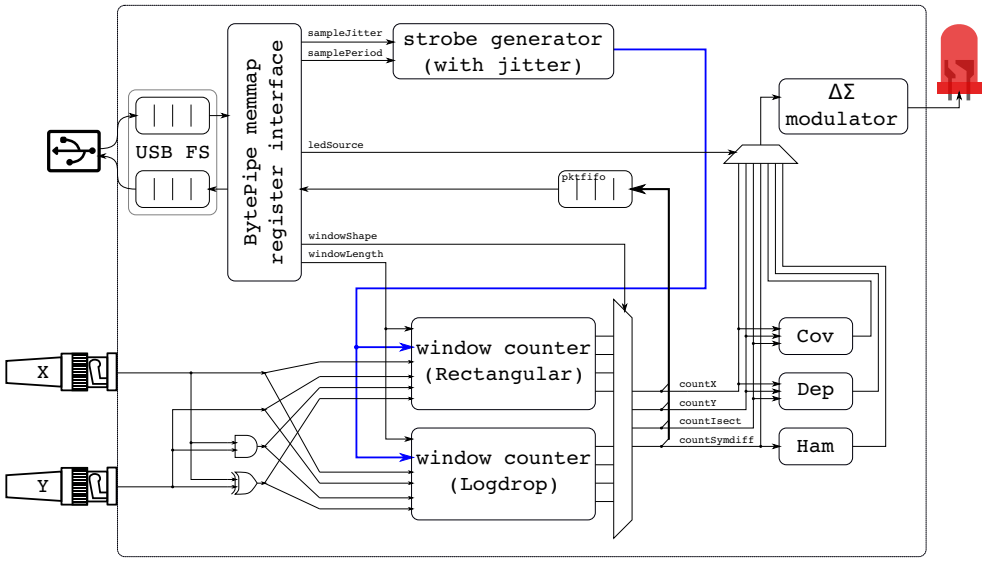


Figure 4.2: Microarchitecture of a standalone correlator with a single engine. Strobes if the anti-phase sampling mechanism are shown as a single blue line. Output of the $\Delta\Sigma$ modulator is depicted regulating the brightness of an LED, but could also be measured using an oscilloscope.

4.3.1 Sampling Mechanism

In order to acquire data from the circuit under test, a sampling mechanism is required to interpret voltage levels on wires operating in unknown clock domains. As explained in Section 4.2.1, the assumptions must be made that probed signals carry binary data and transition at less than the Nyquist frequency. Figure 4.3 depicts the main structures in the data capture mechanism. A pair of signals are selected from a set of probes after passing through a standard meta-stability mitigation (2 series DFFs). Four signals representing f_x, f_y, $(f_x \odot f_y)$, and $|f_x - f_y|$ carry the binary value present when `sampleStrobeX` and/or `sampleStrobeY` was last asserted. For all DFFs to operate in a single clock domain, thus straightforward FPGA implementation [109], the `sampleStrobe*` signals control re-circulating multiplexers which control data-flow into the correlator.

In an on-chip integration scenario where hardware is designed to monitor signals of a parent system, the set of monitorable signals is fixed in RTL code and cannot be changed at runtime. Using a crossbar (set of multiplexers) to flexibly route signals to a limited set of correlator engine probes is useful for system designers when it is unclear what the most useful signals will be in a running application. This data capture scheme mostly comprises of techniques common to other embedded analytics modules [92]; however, the novelty of this work lies in the logic behind the `sampleStrobe*` signals.

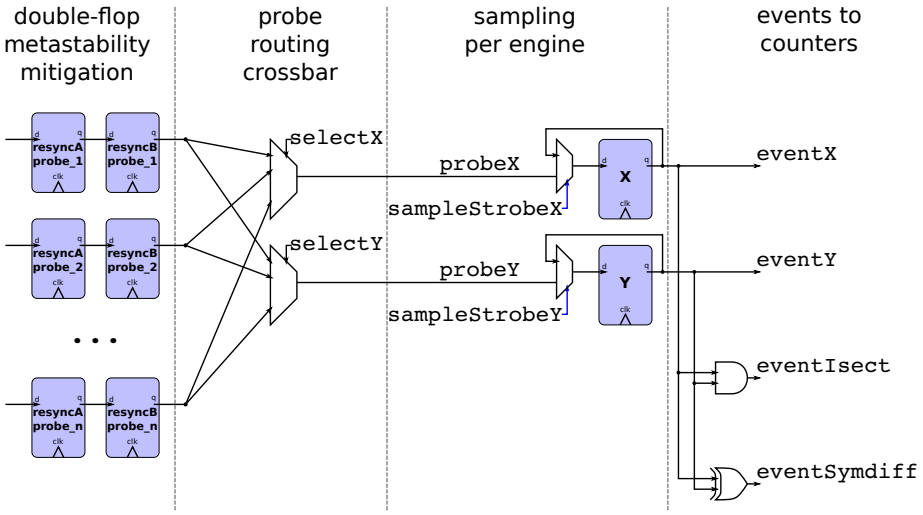


Figure 4.3: Correlator data capture mechanism. Metastability issues are mitigated by the double-flop technique [17], shown on the left. Flexible source selection, essential for embedded analytics scenarios, is enabled via the probe routing crossbar. Events, sampled from probes on assertions of `sampleStrobe*`, are delivered to the windowing counters as `event*`. All DFFs operate in the same clock domain.

4.3.1.1 Introducing Variance to the Sample Period

The nature of digital logic design means that it is intuitive to find correlations between events occurring close to each other, rather than just concurrently. For example, a pair of status signals `busy` and `idle` on the same module can be reasonably expected to be highly correlated. This is still a reasonable expectation even when the relationship is not a simple inversion, i.e. $\neg$`busy` $\neq$ `idle`. Instead, one might expect that this pair of signals will transition close to each other, e.g. `busy` get de-asserted at the end of a calculation, then some wind-up routine is performed, then `idle` is asserted. As described in Section 4.2.1, the notation $f_{i_{\langle -\delta \rangle}}$ is defined to represent f_i delayed by δ cycles of the correlator clock. Hardware to deterministically support searching for correlations over many values of δ presents a significant challenge to practical implementation. Real-time correlation monitoring across Δ versions of each signal would naturally require Δ additional bits of state, as well as Δ calculations running in parallel for each pair signal pair. However, accepting some error in the correlation analysis offers an avenue for exploitation.

Statistically combining a range of δ-shifted versions of a signal allows the sampling mechanism to capture some of the potential occurrences into one counter. Knowing that a signal pair is correlated at all is a more salient feature of behaviour than knowing the exact result of a correlation metric. Seeing a small but significant result when the sampling period has a large variance may motivate an engineer to investigate further by experimenting with the period variance and specific values of δ to find the most correlated one. Searching over δ in this way therefore requires only a single calculation, but experimenting with Δ different values of δ still requires the Δ bits of state.

Introducing introducing jitter to the sampling process delivers the timeline shown in Figure 4.4. Samples of `probeX` and `probeY` are most like to be taken around the same time, rather than at precisely regular intervals.

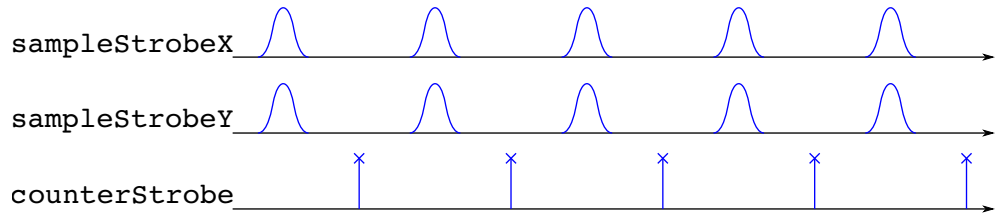


Figure 4.4: PDF of event/counter anti-phase sampling timeline. Counters increment at regular intervals spaced as far as probabilistically possible from when events are sampled to avoid missed/repeated samples; i.e. the sampling processes operate in anti-phase to the counter processes.

4.3.1.2 Jittery Sample Strobe Mechanism

The most useful number of samples in a window and the level of jitter in the sampling process is dependent on the system being analysed. Therefore, circuitry driving the `sampleStrobe*` signals in Figure 4.3 must provide a method of controlling both the period mean length, and the period variance. Analogue and closed-loop solutions for driving the sample strobes might make use of a Phase-Locked Loop (PLL) or Delay-Locked Loop (DLL) [110, 111], which invites comparison with a FPGA solution using a digital equivalent [112]. In contrast to this application of aiding correlation analysis, typical DLL circuits aim to reduce jitter rather than introduce a specific variance to the period. Open-loop control allows for implementations requiring fewer logic cells than closed-loop PLL or DLL constructions, particularly when the desired frequency range (sample rate in the correlator) is very large [113].

In digital design, test-benches often take an open-loop approach to generating clocks in simulation, e.g. `always #(PERIOD/2) clk = !clk;`. A synthesisable version of this approach is to employ a higher-frequency root clock and a counter to implement the delay between toggles, e.g. `always_ff @(posedge rootClk) if (count == PERIOD/2) clk <= !clk;`. Counters are well-understood constructions offering

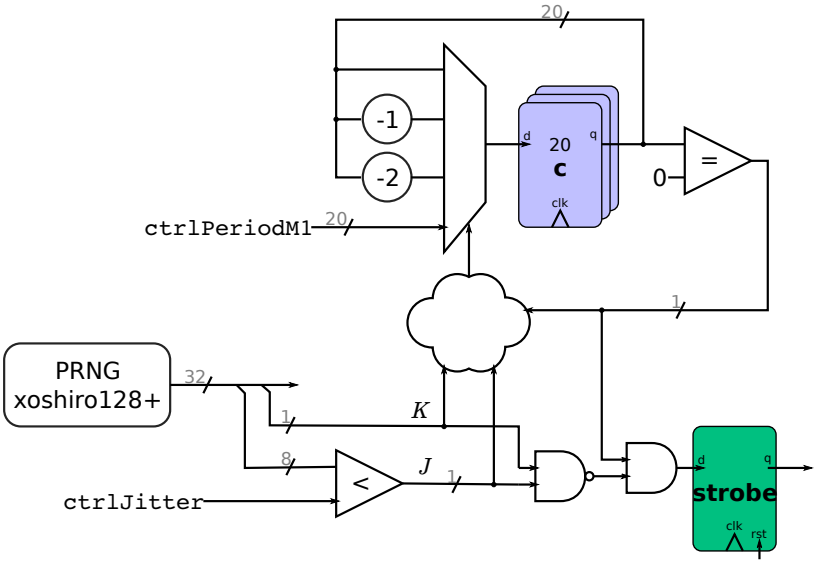


Figure 4.5: Logical design of sampling strobe generator with both period mean and variance controllable. Mean number of correlator clack cycles between assertions of the strobe μ, is programmable in the range $\mu \in [2, 2^{20}]$. Jitter, in the form of variance is programmable in the range $\sigma \in [0, \frac{2^8-1}{2^8}]$. Values 20 and 8 are those used in the case studies of Section 4.4, but are not special so may be parameterised. Arithmetic operations are unsigned (subtract, and compare-less-than).

efficient implementation on modern FPGAs by taking advantage of specialised carry paths.

In the absence of suitable designs in literature, a method of generating sampling strobes with the required control over the period is presented. Figure 4.5 shows the logical topology of the most significant components of the sampling strobe generator implemented in the correlator. To configure the mean interval between strobe pulses to a value μ, counter c is then initialised by holding `ctrlPeriodM1` at the unsigned value $\mu - 1$, then counts down towards 0. Each time c reaches 0, the strobe output is asserted for a single-cycle and the counter is re-initialised to the value of `ctrlPeriodM1`. The counting process is affected by the output of the Pseudo-Random Number Generator (PRNG) such that on each correlator clock cycle the decrement is chosen to be 0, -1, or -2.

True random number generators are not appropriate for a digital design which must be simulated with reproducible results, but a PRNG is an appropriate approach for introducing randomness into the sample period. In this application, cryptographic-strength is not a factor in choosing a suitable PRNG algorithm. Instead, a suitable algorithm is chosen based on the ability to produce a multi-bit value on each clock cycle, while using a minimum of resources, and exhibiting good statistical randomness. Although Blackman and Vigna's xo(ro)shiro [114] family of PRNGs were originally designed and optimised for software implementations, some members of the family allow for efficient implementation in FPGA hardware. From the 32 b generators in the xo(ro)shiro family [3] , "xoshiro128+" has been chosen for the relatively low requirements of single 32 b adder and 128 b state.

Multiple bits are required on each cycle to provide two Bernoulli random variables, marked $J \sim \mathrm{B}(j)$ and $K \sim \mathrm{B}\left(\frac{1}{2}\right)$. On each cycle, J is tested to decide whether the counter should decrement by one or something else (zero or two). Consistently decrementing by one results in the counter reaching zero in exactly μ cycles. If J determines that the decrement should not be one, K is tested to decide whether the counter should decrement by zero or two. When the counter is initialised to s, the probability that $c = 0$ after n cycles can be modelled [4] as the simple random walk on $\mathbb{Z}$ but shifted in the positive direction by s. The simple random walk on $\mathbb{Z}$ is a well-known problem [115], and the PDF of the number of correlator clock cycles between strobe assertions is approximated by the Gaussian function in Equation (4.5). Figure 4.6 visualises the expected number of correlator clock cycles between each sample with a selection of variances.

[3] Appendix E provides characterisation of a selection of this family's PRNGs.

[4] Further analysis, showing how closely the Gaussian PDF approximates the PMF of the counter process, is given in Appendix F.

$$\Pr(c = 0 \text{ in exactly } n \text{ cycles}) \approx \frac{1}{\sqrt{sj}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{n-s}{\sqrt{sj}}\right)^2\right) \tag{4.5}$$

The plot in Figure 4.6 visually highlights how control signals of the mechanism in Figure 4.3 relate to the statistical properties of the sampling process described by Equation (4.5).

Two instances of this mechanism are required to drive the two signals `sampleStrobeX` and `sampleStrobeY` in Figure 4.3 in order for these strobes to operate independently. A further instance with the variance control `ctrlJitter` fixed to zero is used to drive the `counterStrobe` signal of Figure 4.4, whose purpose is described in Section 4.3.2.

Purposefully introducing variance to the interval between samples, i.e. using a jittery sampling process, enables samples to be taken from signals which may be slightly offset in time. The mechanism described in this section enables jittery sampling in a runtime-configurable manner which is straightforward to implement on an FPGA.

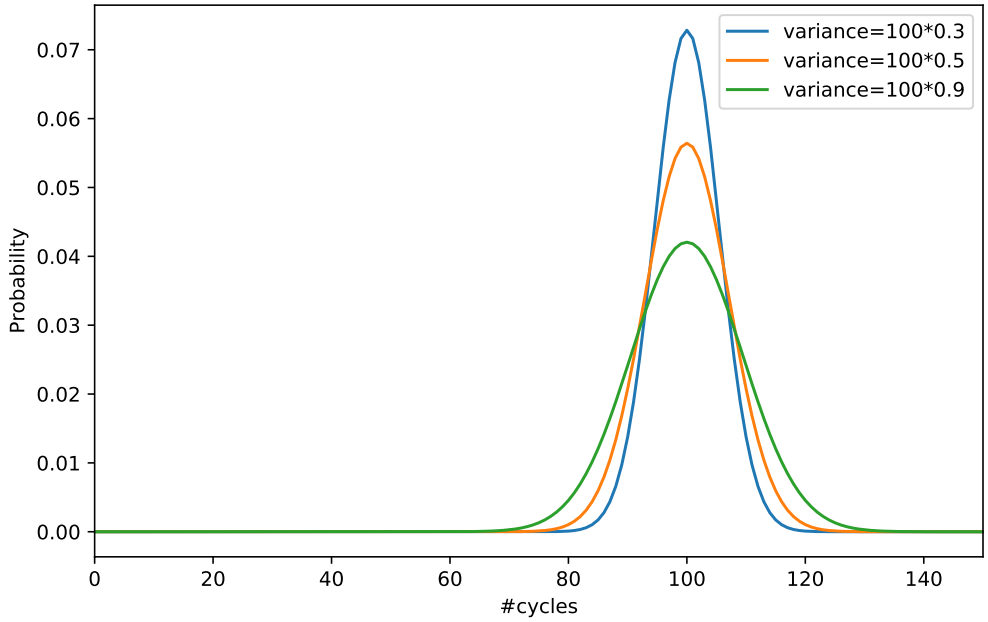


Figure 4.6: PDF of sampling strobe period, counted in number of clock cycles. Three values of variance are shown, corresponding to setting `ctrlJitter` in Figure 4.5 to $0.3 \approx \frac{77}{2^8}$, $0.5 \approx \frac{128}{2^8}$, and $0.9 \approx \frac{230}{2^8}$. Period is centred around a mean of 100 cycles, corresponding to holding `ctrlPeriodM1` at a value of 99, i.e. $s = 100$.

4.3.2 Windowing Counters

Chapter 3 considers discrete-time binary SoC signals which assert according to a behaviour defined by a probabilistic model. However, in real systems behaviour is non-stationary and is expected to change as the system performs different tasks or reacts to different inputs. The non-stationarity of correlations which transition as a system works on different data or applications therefore requires a correlation analysis to refer to a particular time interval. Repeating correlation analyses for different intervals thereby allows a SoC engineer to see how component interactions change over time.

Chapter 3 shows that it is possible to calculate useful metrics of pairwise correlation using the expected values of both bit-vectors (f_x and f_y) and at least one of $\left(f_x \odot f_y\right)$ or $\left|f_x - f_y\right|$. It is also noted that an expected value can be computed using a counter, thus foregoing the need to store a potentially large bit-vector; Each time a signal expression is observed "high", a counter c is incremented by constant k; Once the window has completed the expectation can be computed directly $\mathbb{E}[f_i] = \frac{c_i}{kN}$. In effect, this method applies a rectangular window function; i.e. $k = \frac{1}{\Sigma_w}$ from Equation (4.3). The correlator device expands on this method to provide mechanisms supporting window functions of different lengths (different number of samples, N), and a non-rectangular window function. The same mechanisms are applied for counters used to calculate $\mathbb{E}[f_x]$, $\mathbb{E}\left[f_y\right]$, $\mathbb{E}\left[f_x \odot f_y\right]$ and $\mathbb{E}\left[\left|f_x - f_y\right|\right]$. This work considers correlation analysis over a single time window rather than the separate problem of cross-correlation of counter values over a sequence of time windows.

The simplest window function, rectangular, does not provide an accurate model of how engineers (people) visually search for patterns of interest [116]. To support a better model of foveal and peripheral vision, a bell-shaped window function is more appropriate. A novel approach, specific to digital logic implementation, is presented to implement a new window function "Logdrop" which is suitable for use with a window counter. The mechanisms behind the rectangular window counter are first explained in Section 4.3.2.1, then the Logdrop window function and counter mechanism are presented in Section 4.3.2.2.

4.3.2.1 Rectangular

A sampling strobe asserts for a single correlator clock cycle at specified intervals which signals a sampling circuit, as in Figure 4.3, to measure and store one sample from a probe. The `counterStrobe` signal in Figure 4.4 is asserted with the same mean interval between sample strobes, but in anti-phase. At the beginning of

each time window, all four rectangular-window counters are initialised to zero. On each assertion of `counterStrobe` within a time window, a sample pair from f_x and f_y are examined, then the four counters increment if, and only if, their respective conditions are met (f_x, f_y, $f_x \wedge f_y$, $f_x \oplus f_y$).

For a rectangular window counter, the scaling factor $\frac{1}{\sum_w}$ may be applied either at the point of each increment, or at the point that the result is retrieved. Scaling at the point of retrieval has the advantage of using a standard counter using an increment value with only the lowest bits set, which allows for the simplest circuitry. However, in order to interpret a fixed-width result, it must be accompanied, somehow, by the number of samples in that window N. Scaling at the point of increment, while necessitating more logic in the counter circuitry, provides two advantages: (1) The value of each result bit is fixed for all values of N; i.e. the highest bit index of the result is always the most significant bit, and the lowest bit index is always the least significant. (2) Non-rectangular window functions, which do not generally have the option of scaling at the point of retrieval (because $\forall n \in [0,N) : w[n] \leqslant 1$), can use the same mechanism.

There are multiple widely used options available for the binary format of counters and correlation results. IEEE floats [117], while supported in software compilers, require more complex logic than posits [118, 119, 120], which in turn require more complex logic than fixed-point formats [121]. Expected values lie on the fixed interval $[0,1]$, and therefore there is no need for the much larger range afforded by floating point formats. The correlator device, aiming for low-cost and small implementation uses the most common fixed-point representation, Q-format [122]. As the counter value represents $\mathbb{E}[f_i] \in [0,1]$ the position of the implicit fixed point is constant for any value of N. For example, a 32 b counter represents an expected value in unsigned Q1.31 format, where bit 30 always represents $\frac{1}{2}$, bit 29 always represents $\frac{1}{4}$, etc.

Figure 4.7 shows the mechanism used in the correlator device. The beginning of a new window ($t = u,\ n = 0$) is signalled by asserting `tWrap` to zero the counter. Restricting the range of N to powers of two means that the corresponding increment value, selected using `windowLengthExp`, is also restricted to powers of two; i.e. exactly one bit set. This restriction is convenient for RTL implementation because inconsistencies with rounding errors can be completely avoided. If arbitrary values of N are allowed then rounding errors are necessarily introduced; For example, $N = 10$ implies an increment of $\frac{1}{10}$ which lies between $0.09375 = 0.00011_2$ and $0.125 = 0.00100_2$, which cannot be represented without error in Q1.5 format (or any Qm.n format) because 0.1 (decimal) is a recurring bicimal.

The function of the rectangular window counter mechanism is to facilitate

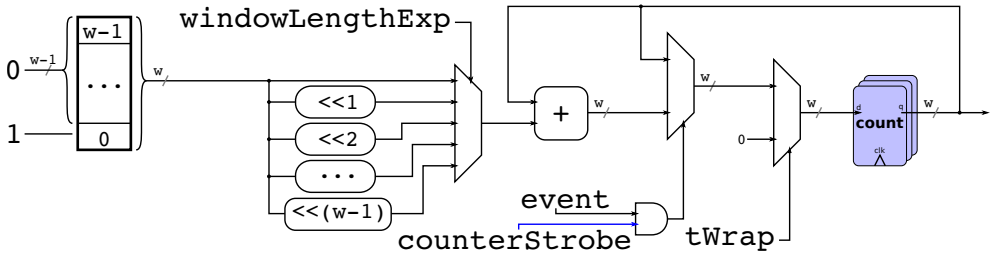


Figure 4.7: Logical design of scalable counter with rectangular windowing. Counter of w bits is cleared by `tWrap` at the beginning of each time window. For each sample the counter is incremented if `event` is asserted. Increment is a onehot value selected by `windowLengthExp`.

scaling at the point of increment by directly using the definition of N; i.e. $N = 2^{\texttt{windowLengthExp}}$. By restricting N to integer powers of two, thereby restricting increments to fractional powers of two, a simple multiplexer-based structure is permitted, thus providing a foundation for the Logdrop window counter in Section 4.3.2.2.

4.3.2.2 Logdrop

In Fourier analysis, a non-rectangular window function is often employed to focus sample energy into the fundamental frequency component. The most straightforward approach in software is to store a table of pre-computed coefficients, then at runtime simply lookup the coefficient indexed by n then multiply with the corresponding sample. The lookup-table method is attractive because it allows arbitrary window functions to be specified; however, the drawback in a resource-limited hardware implementation is that a memory is required to store the table of coefficients. For example, a window with $N = 2^{16}$ with a coefficient resolution of 16 b requires a memory of 1 Mib. This is too much to fit on a small FPGA such as the Lattice iCE40LP8K used which has only 128 Kib of available flash memory. An alternative approach may be to use a CORDIC-based [123] circuit to calculate successive coefficients. CORDIC implementations also require storage for parameters and a significant amount of logic to control their iterative process, making this approach less appropriate for resource-constrained FPGAs. For many signal processing applications, the benefits of using a window function far outweigh the cost of implementation, particularly in software-based systems where memory is relatively cheap.

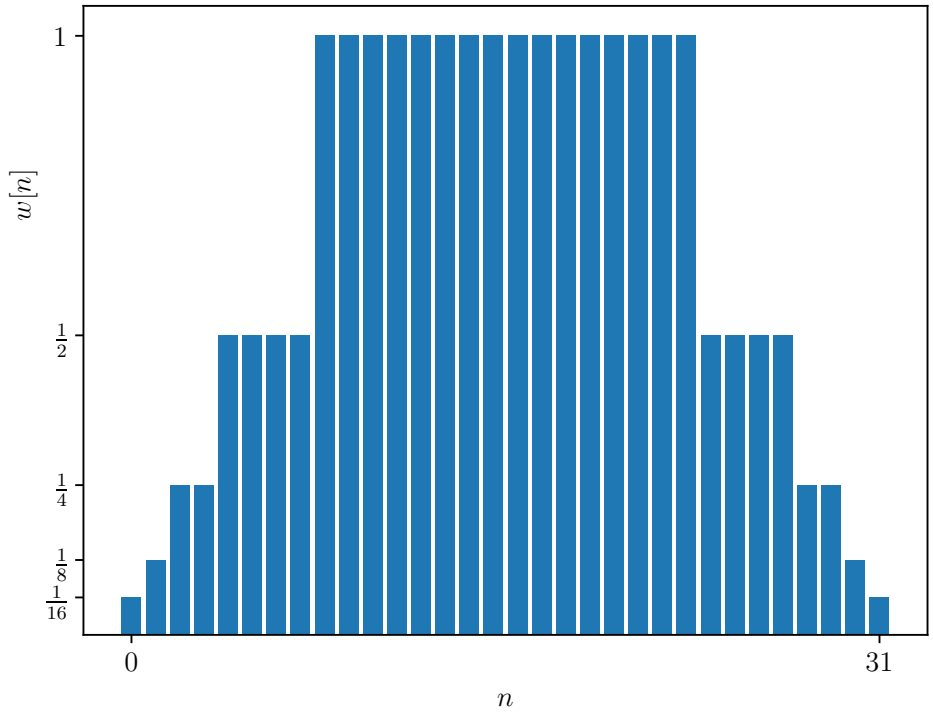
In correlation analysis, the purpose of a window function is to focus analysis on a particular region of time; i.e. to assign more weight to samples close to the centre

of a time region of interest. Real-time on-chip correlation analysis, implemented in digital logic, should aim toward smallest possible hardware requirements, as explained in Section 4.2.3. Differences in purpose and type of implementation alter the cost/benefit calculation, creating the desire for a window function that can be implemented in digital logic with no memories and few logic resources. To this end, a novel window function, "Logdrop", is presented in Equation (4.6).

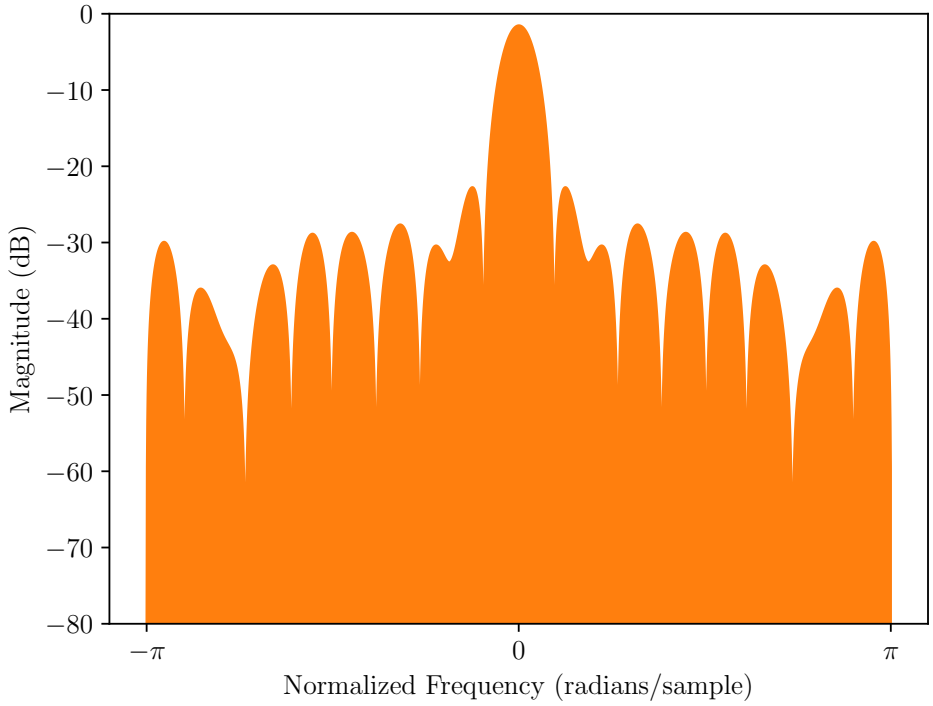
$$\underset{\text{logdrop}}{w[n]} := 2^{\lceil \log_2 \min(n+1, N-n) \rceil - \log_2 \frac{N}{2}} \qquad \in \mathbb{R} \cap (0, 1] \tag{4.6}$$

The name Logdrop comes from the action of dropping a number of bits based on $\log_2 n$. Each Logdrop coefficient is a fractional power of two, allowing multiplication to be performed using a bit-shift operation. Calculating each coefficient from the window index n means that no memory is required to store coefficients. Further, as all multiplications are possible via bit-shifting, there is no need for a multiplier unit, notorious for their logic requirements [124, 125]. As is standard for presenting window functions, plots showing coefficient values and the normalised frequency response are shown in Figure 4.8. As the coefficients towards the edges of the Logdrop window depend on N, a second plot is given in Figure 4.9 which highlights the difference in operation as N increases.

Figure 4.8 plots the coefficient values over a window with $N = 32$ using zero-indexing as is most convenient for RTL implementation. As N is increased, the edges of the Logdrop window function become smoother with an additional "step" introduced for each increase in N. Spectral performance of Logdrop with $N = 32$ appears poor for Fourier analysis with only around -30 dB of sidelobe suppression; however, this is due to the low value of N. Increasing N to a larger value of, e.g. $N = 2^{16}$ shown in Figure 4.9, smooths out the frequency response and suppresses the magnitude of sidelobes.

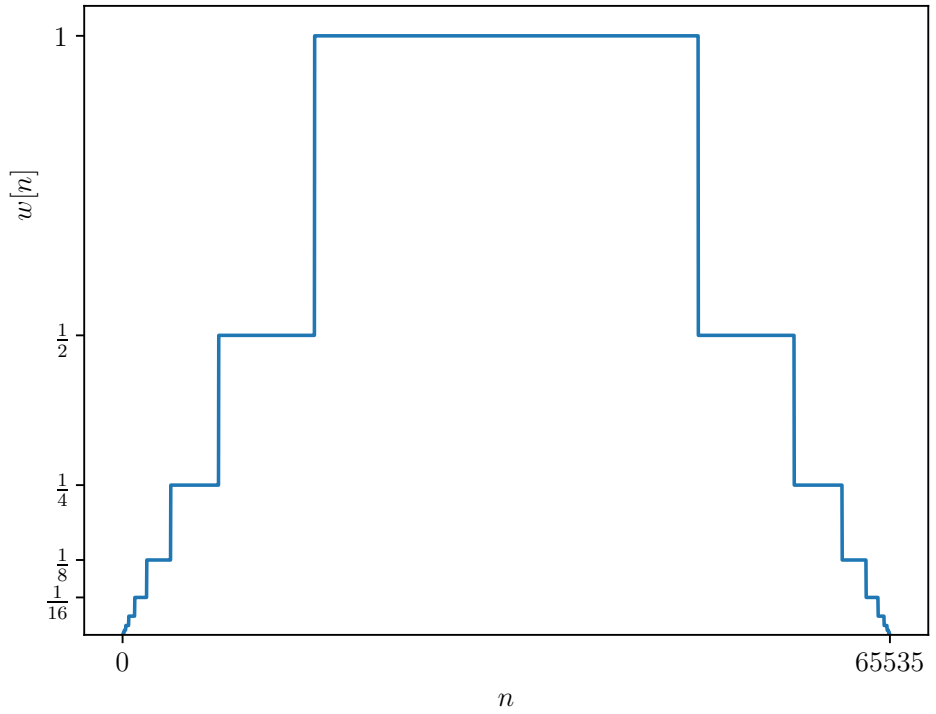


(a) Coefficients of Logdrop windowing function.

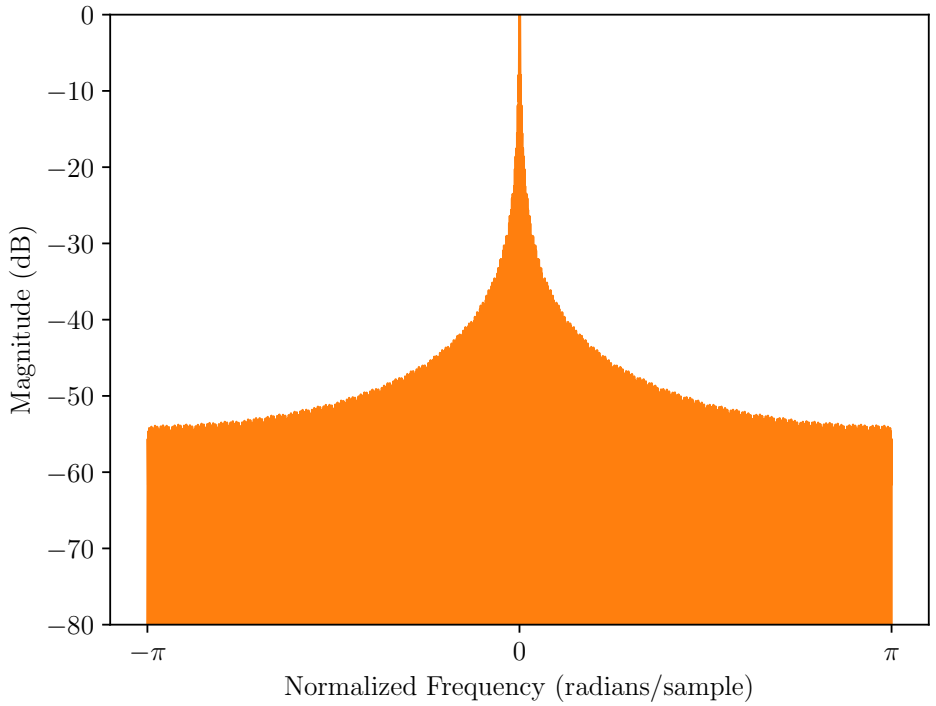


(b) Frequency response of Logdrop windowing function. Spectral sidelobes are suppressed by approximately −30 dB.

Figure 4.8: Logdrop windowing function with $N = 2^5$.



(a) Logdrop coefficients plotted as a line rather than histogram because bars would be indistinguishable. Smooth-edge effect can be seen on the smaller values.



(b) Frequency response of Logdrop large N.

Figure 4.9: Logdrop windowing function with $N = 2^{16}$.

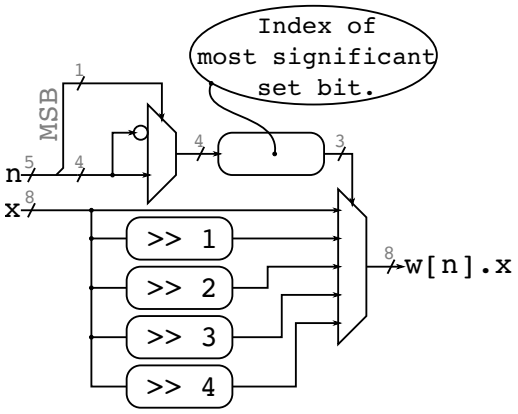


Figure 4.10: Logical implementation of Logdrop on 8 b input where $N = 32$; i.e. window index is 5 b wide. The Most Significant Bit (MSB) of n selects whether to invert the lower bits of n, the choice of which produces a triangular counter (counting up $0 \rightarrow (\frac{N}{2} - 1)$, then down $(\frac{N}{2} - 1) \rightarrow 0$) as shown on the top-left. The most-significant-set-bit operation is defined as $\text{mssb}(a) := \max(y \mid 2^y \leqslant a)$, and is used to select the number of bits to right-shift the input.

The mechanism depicted in Figure 4.10 shows that no state is required to implement Logdrop because all coefficients are straightforward to calculate from the window index n in combinatorial logic. Implementation of Logdrop is simply the matter of determining the amount by which to the input $x[t]$. Direct calculation of coefficients from the window index with fixed logic is a less flexible approach than fetching coefficients from a memory which would allow for arbitrary window functions. However, in the application to apply a statistical focus around a central point in time, the cost/benefit calculation strongly favours the low resource requirements afforded by Logdrop.

Figure 4.11 shows how the Logdrop function is used to build a windowing counter. Referring back to Figure 4.7 for comparison, it can be seen that the increment value (lower-left) is backed by the Logdrop logic which changes the coefficient as the window index progresses.

Using the scaling increment mechanism, the Logdrop window function is implemented in the correlator. This mechanism enables implementation in digital hardware of a model of how people focus their attention; i.e. more weight applied to the centre of an interesting region than the edges.

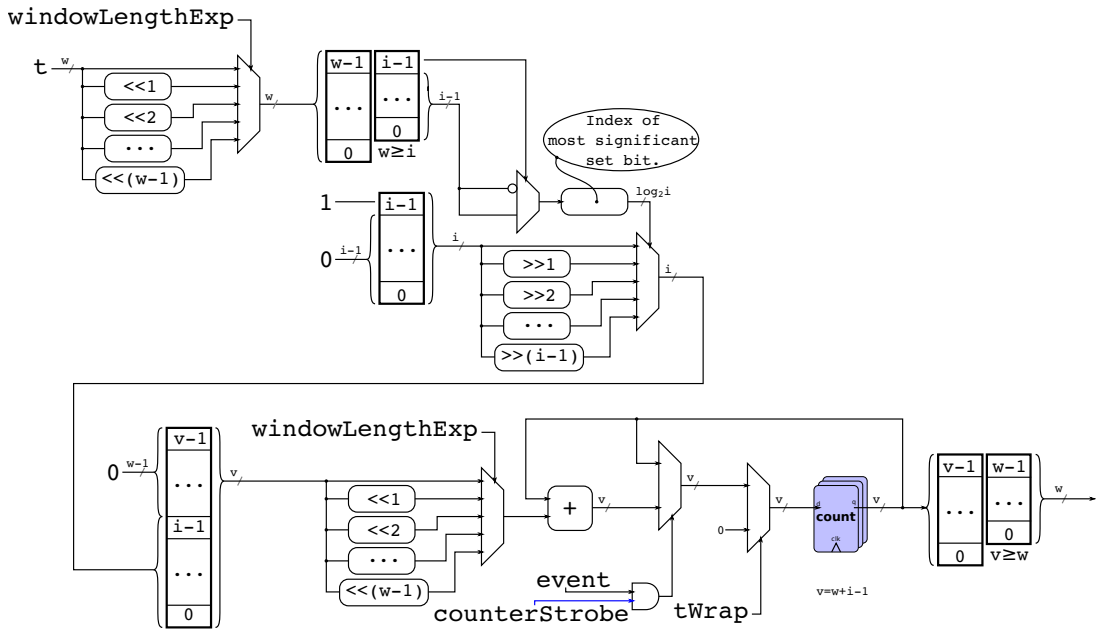


Figure 4.11: Logical design of scalable counter with Logdrop windowing. Similar to the rectangular scalable counter, except the onehot increment amount is further determined by the window index t and the counter is wider. The resulting wb value is obtained by reading the w most significant bits from the counter.

4.3.3 Consuming Correlation Results

In the correlator device, when a window completes it is immediately followed by another, which means that the counters produce results at a steady fixed rate. At the end of each window each correlator engine produces four fixed-precision results corresponding to $\mathbb{E}[f_x]$, $\mathbb{E}[f_y]$, $\mathbb{E}[f_x \odot f_y]$, and $\mathbb{E}[|f_x - f_y|]$ at a rate determined by N and the sample rate. Off-chip options for correlation analysis, perhaps using the machine-learning techniques discussed in Chapter 3, are enabled by the `pktfifo` component shown in Figure 4.2. However, very high result data rates can be specified with a high sample rate and a short window length.

While it is often desirable to transfer correlator results off-chip for later analysis, specifying a window length and sample period which produces data at a rate higher than the datalink can sustain, is a valid case for analysis. Additionally, for datalinks with unknown or uncontrollable bandwidth resources [5], such as USB [90], it is not practical to change analysis parameters to suit the datalink.

[5] USB datalinks are controlled by the host, not the device, which allows the Operating System (OS) to disconnect, throttle, or otherwise restrict devices at any time without notice. Additionally, the tree topology of USB means that bandwidth and latency properties depend on what other devices are connected.

Accommodating high data rate analysis therefore requires an alternate method of presenting results which is not dependent on the main datalink.

4.3.3.1 Real-time Results via $\Delta\Sigma$ and Low-Pass Filter

The correlator micro-architecture, depicted in Figure 4.2, features an LED driven by a $\Delta\Sigma$ modulator, which is the implemented solution for presenting high data rate results. A digital solution to moving data off-chip, i.e. using GPIO pins, allows the design to be implemented across different FPGA technology families. Human vision is too slow to distinguish pulses produced at frequencies in the MHz range. Pulse modulation therefore appears as a dimming effect via the persistence-of-vision visual phenomenon. Additionally, LEDs are a convenient low-cost method of introducing a physical threshold function to filter out low-valued results. The dimming transformation is non-linear due to both the gamma correction effect of human vision, explained in Appendix I, and the LED's IV curve. The forward region of IV curves of different LED chemistries are similar to Rectified Linear Unit (ReLU) functions with different offsets and gradients [89]. This has the effect of keeping the LED in the dark state, effectively suppressing low-value results.

To view and accurately record fast-changing behaviour, an off-the-shelf oscilloscope may be used to measure voltage on $\Delta\Sigma$ output pins. Passing the high-frequency $\Delta\Sigma$ pulses through a Low-Pass Filter (LPF) transforms the result data $[0,1] \rightarrow [\texttt{GND},\texttt{VCC}]$, allowing the LPF parameters to be adjusted appropriately for the expected rate of change in behaviour. Transmitting correlator results using a $\Delta\Sigma$ modulator is a convenient low-cost approach where each correlator engine requires only a single GPIO pin, and functions as a side-channel separate from operation of the parent system.

4.3.3.2 Calculating Correlation Using Counters

Chapter 3 defines six metrics with simple formulations (Ḣam, Ṫmt, Ċls, Ċos, Ċov, and Ḋep), investigating their effectiveness for finding pairwise logical relationships between SoC signals. As discussed in Chapter 3 and Section 4.3.2, these metrics are amenable to calculation using counters. It is found in Chapter 3 that Ċov and Ḋep are the most effective at uncovering logical relationships between SoC signals, therefore the correlator device implements these two metrics. It is also noted that for binary signals Ḣam is extremely simple to implement, requiring only a bitwise inversion of the `countSymdiff` counter result. Real-time calculation of Ḣam, Ċov, and Ḋep is performed by the mechanism shown in Figure 4.12. The $\Delta\Sigma$ modulator can then select one of these results to control the frequency of output pulses.

The other simple metrics (Ṫmt, Ċls, Ċos) are not implemented due to the significant complexity involved in their operation and their relatively poor performance in Chapter 3. An implementation of the Jaccard index (Tanimoto's reformulation Ṫmt) would require an additional divider circuit. Geometric closeness Ċls and the cosine similarity Ċos require circuits for calculating square-roots, as well as additional counters for $\mathbb{E}[f_x^2]$, $\mathbb{E}\left[f_y^2\right]$, and $\mathbb{E}\left[\left|f_x - f_y\right|^2\right]$. While both division and square-root operations are straightforward using a Finite State Machine (FSM) implementing variants of the CORDIC algorithm, the logic requirements are deemed too significant for inclusion in the correlator device.

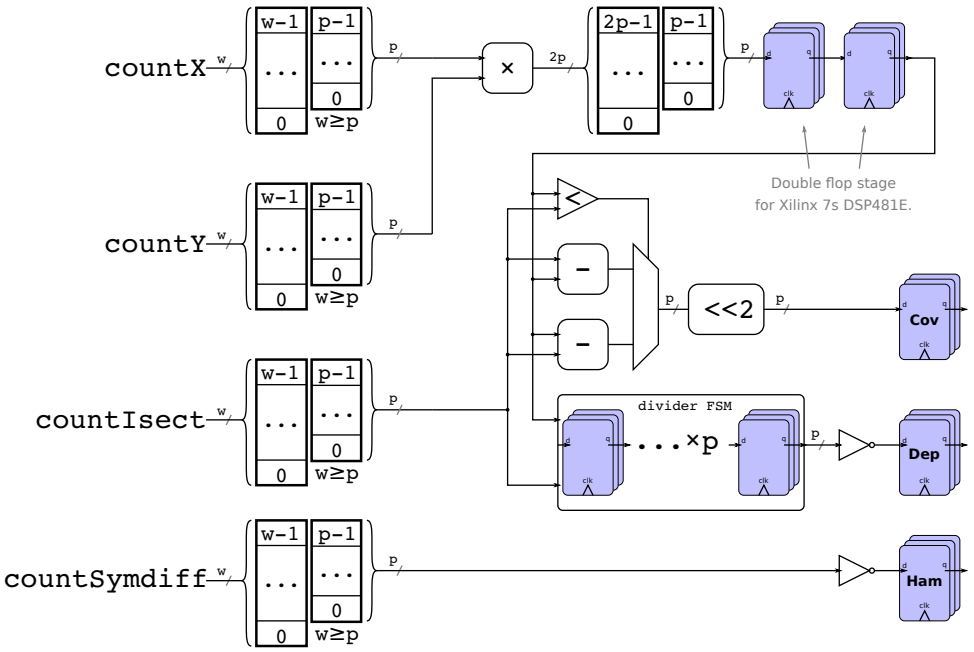


Figure 4.12: Logical design of correlator metric calculators. Ḣam is the simple reflection of $\mathbb{E}[X \oplus Y]$. Ċov involves a multiplier followed by an adder then a multiplexer. DFFs are inserted to allow FPGA synthesis tools to use DSP cells which significantly reduces timing pressure. On the Xilinx 7-Series technology two flop stages are required to infer a DSP481E cell. The Lattice iCE40LP technology does not include any multiplier primitives so the two flop stages simply relax timing constraints on a multiplier assembled with SB_LUT4 and SB_CARRY cells. Ḋep requires a divide operation which is implemented with a multi-cycle FSM because calculation latency is not critical.

4.3.4 Synthesis Characterisation on FPGAs

The correlator device is intended to be integrated into parent SoCs, and must facilitate implementation across a variety of FPGA technology families. The correlator can also be implemented as a stand-alone device, similar in concept to an oscilloscope, which allows the design to be developed and characterised without unintended effects from a parent system. As explained in Section 4.2.3, characterisation is essential to ensure that the RTL design is readily usable across different FPGA technologies, thus facilitating integration with different parent systems.

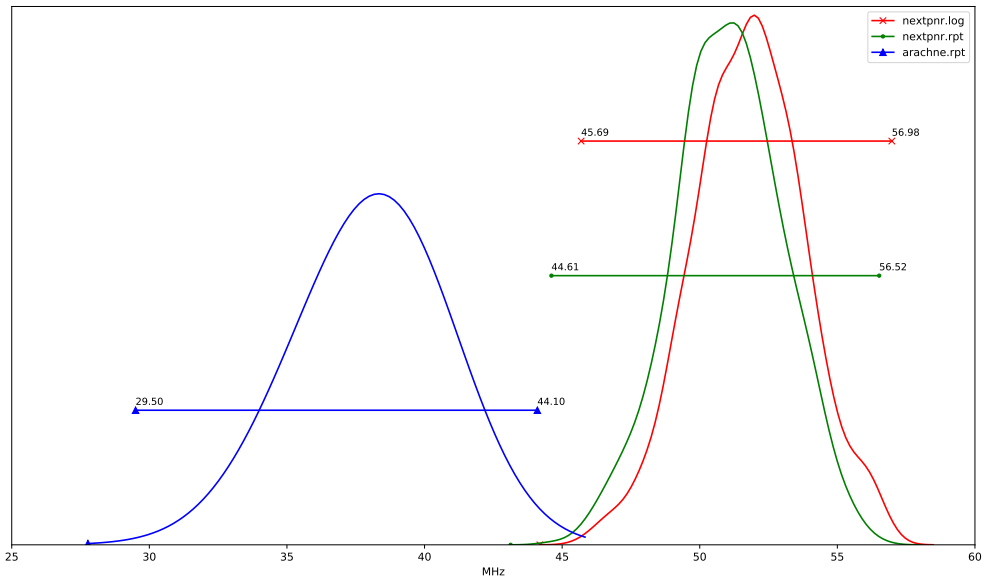
To this end, the stand-alone correlator is first characterised using the methodology described in Appendix E. Briefly, a minimal example system is synthesised to a netlist and the maximum operating clock frequency ($f_{\max}$) of many PnR solutions are compared. Viewing a PMF of the achievable $f_{\max}$ then allows the robustness of a design to be visually quantified. This multi-PnR method is applied to the correlator device on the Lattice iCE40LP process which is marketed for its low power consumption. Low power consumption comes at the cost of low achievable $f_{\max}$ relative to other FPGA technologies. The benefit of targeting a known-slow technology is that further implementations on faster technologies are likely to be straightforward. In parent systems that already consume a large percentage of their FPGA's resources, congestion is a particular concern because the number of PnR solutions are restricted [126, 127]. Floorplans of various correlator implementations are shown to visually demonstrate the flexibility available to the PnR process.

The only component of the stand-alone correlator device with a fixed clock frequency requirement is the USB-FS interface, which must run at 48 MHz to provide the required baud-rate of $12\,\mathrm{Mb\,s^{-1}}$. Hence, an $f_{\max}$ of 48 MHz is set as the target for the correlator device; i.e. the minimum $f_{\max}$ sufficient for correct operation. However, a higher value is desirable to indicate that the design has enough slack to accommodate layout and timing and layout constraints imposed by a parent system.

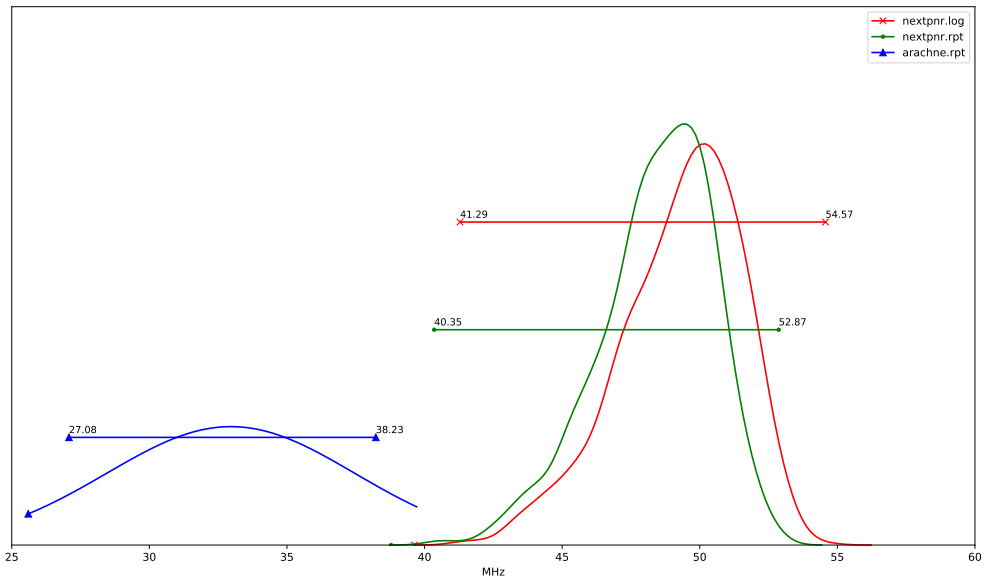
Figure 4.13 shows the estimated PMF of $f_{\max}$ from different combinations of PnR and timing report tools. It can be seen that a satisfactory $f_{\max}$ is achieved using the tool "nextpnr" because the red and green plots have most of their weight rightwards of 48 MHz. For the same two variants Figure 4.14 shows the placed and routed floorplan for the stand-alone correlator device, implemented on a Lattice iCE40LP8K. It can be seen that in the 2-engine variant, around one third of logic resources are unused, with the 3-engine variant approaching the resource limits

for this part. On the Lattice iCE40LP8K, each engine requires around 1800 LUTs with the 2-engine design in Figure 4.14a using 4435 available LUTs (57%), and a 3-engine design in Figure 4.14b requiring 6261 LUTs (81%). Despite approaching the practical limits in terms of logic resources, the 3-engine variant does not suffer too greatly in terms of achievable $f_{\max}$. The PMF seen in Figure 4.13 of nextpnr solutions (red and green) is forced downwards by around 4% ($\approx 2\,\text{MHz}$), despite moving from 57% to 81% LUT. This suggests that the additional logic of the third engine is not required to be inserted into critical regions of other two engines, allowing some flexibility in placement; i.e. layout and timing constraints are slack, as required to permit integration with parent systems.

Aside from minor changes at the electrical interface, described in Appendix H, and technology-specific changes such as pin selection and clock configuration, the same design is implemented on FPGAs from the Xilinx 7-Series family. Floorplans for the correlator device implementations on Kintex-7 and Virtex-7 (Zedboard and VC707 development boards) are shown in Figure 4.15. Due to the black-box nature of the Vivado tool currently required for developing designs on Xilinx FPGAs, it is not possible to perform a multi-PnR analysis. However, the reported $f_{\max}$ of around 250 MHz is several times greater than the target of 48 MHz, which indicates integration should be possible with fast-running parent systems. Noting the high achieved $f_{\max}$ and the low requirements on logic resource, it is predicted that the correlator RTL design is suitable for integration to a larger design on a Xilinx 7-Series FPGA. This is confirmed via successful integrations with a parent system in Section 4.4.

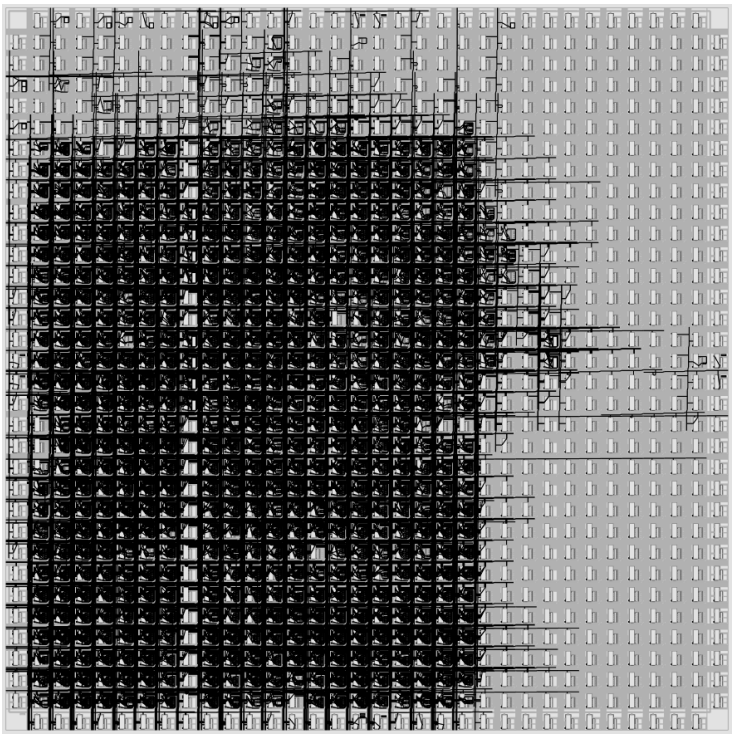


(a) 2-engine variant

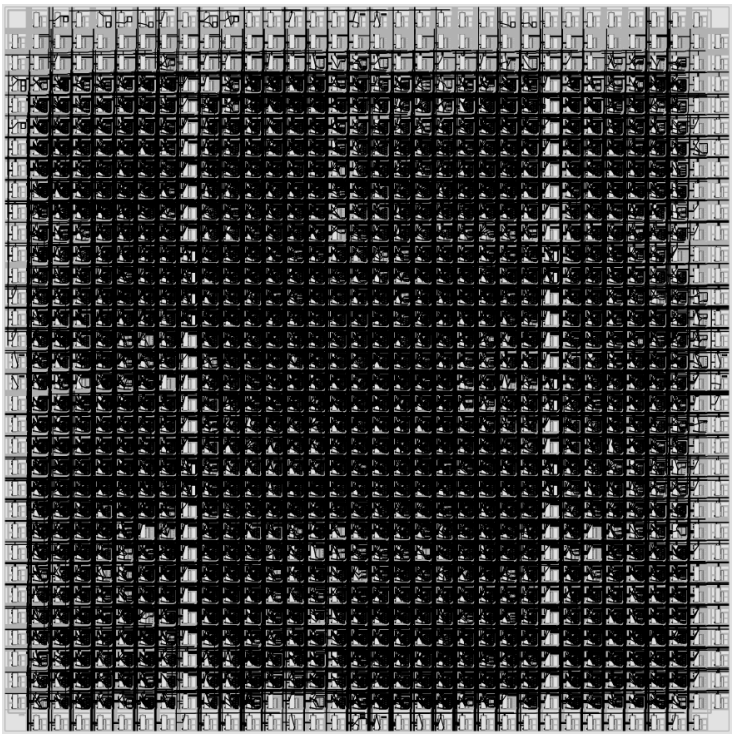


(b) 3-engine variant

Figure 4.13: Multi-PnR results of correlator device implemented on Lattice iCE40LP8K with 2-engine and 3-engine design variants. Legacy PnR tool *arachne-pnr* [128] (blue) achieves the slowest solutions, as reported by analysis tool *icetime*. Another tool *nextpnr* produces faster solutions, also analysed by icetime (green). Results reported directly by nextpnr (red) are slightly more optimistic, centred around an achievable $f_{\max}$ in excess of 50 MHz.

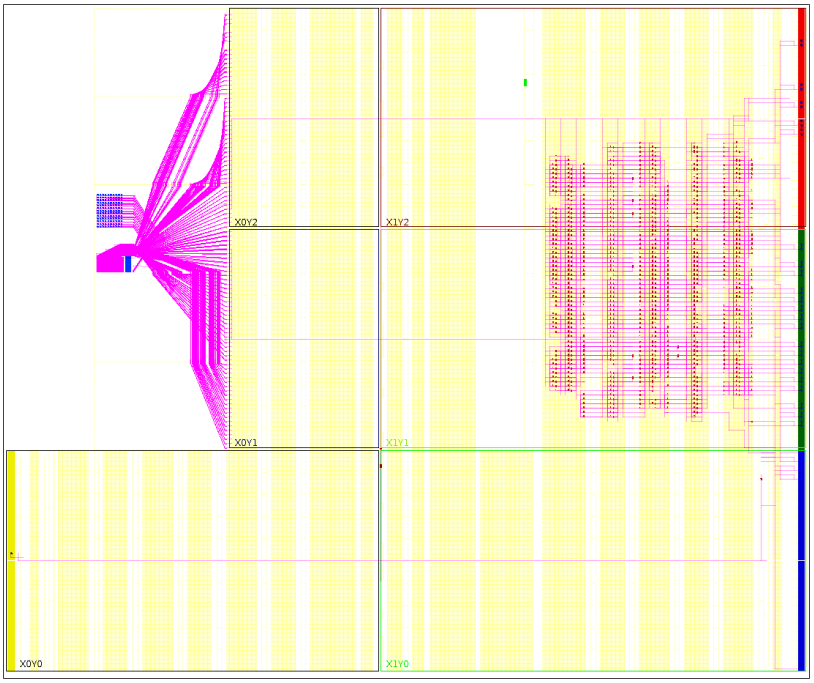


(a) 4 probes, 2 engines

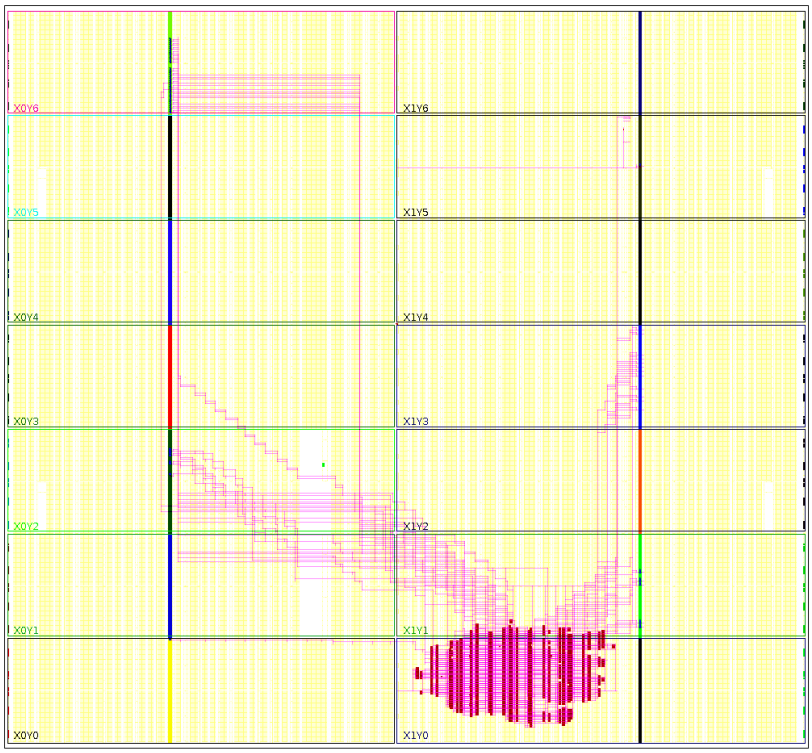


(b) 4 probes, 3 engines

Figure 4.14: Correlator floorplans in a Lattice iCE40LP part (iCE40LP8K on a TinyFPGA-BX development board).



(a) 8-engine correlator floorplan in a Xilinx Kintex-7 part (xc7z020clg484 on a Diligent Zedboard). Upper-left corner is the hard ARM core, and correlator logic is on the right.



(b) 8-engine correlator floorplan in a Xilinx Virtex-7 part (xc7vx485tffg1761 on a VC707 development board). Utilised logic on the bottom-right.

Figure 4.15: Correlator floorplan Xilinx 7-Series FPGAs.

4.4 Integration Case Studies

Demonstrating integration of the correlator device into a larger project requires an interesting pre-existing system which permits modification at the RTL level. OpenPiton+Ariane is a complex multi-faceted open-source SoC project capable of running a Linux-based OS. This system combines the work of many independently-developed open-source projects, each of which have been developed through decades of collaboration amongst the best minds in their fields. Ariane [28] is an application class RISC-V CPU (specifically RV64GC), originally developed as part of the PULP project. OpenPiton [129, 130] is a tiled cache-coherent many-core architecture for creating SoCs with many homogeneous CPU cores. Each of these sub-projects is so complex that while their source code and specifications are open, their interaction details are sufficiently complex that very few people, if any, can claim to have a full understanding of the system. Therefore, the complexity of this system is so high that it effectively constitutes black-box functionality despite source code being available for all constituent components.

Two case studies, based on the same parent system, are given to validate that the correlator design can: (1) implement correlation analysis in RTL as an on-chip module, (2) effectively support lossy output where high result rates are specified, (3) effectively support lossless output where a datalink can support the result data rate, and (4) integrate with existing embedded analytics tooling. The first case study presents an integrated correlator with a USB-FS datalink and uses an oscilloscope to record $\Delta\Sigma$ modulated data (via a LPF). This validates that correlation analysis can be successfully implemented as a passive on-chip correlation monitoring solution and that high results rates can be effectively presented by use of a simple LPF. The second case study features a correlator engine implemented in a modified version of UltraSoC's Status Monitor [92] and retrieves lossless data through an UltraSoC sub-system. This validates that the correlator design is suitable for reporting data, suitable for correlation analysis, to a host system for further processing via an existing commercial embedded analytics solution.

Correlation analysis, which quantifies how signals *do* behave, rather than *should* behave, turns the process of understanding around to a top-down approach, freeing engineers to better focus their attention towards other problems. By selecting only top-level signals for monitoring, the time-consuming and difficult task of finding the intended behaviour of each signal is necessarily avoided. The case studies are based upon the parent system, OpenPiton+Ariane, running a synthetic workload on a Linux OS. OpenPiton specifies a cache-coherent tiled

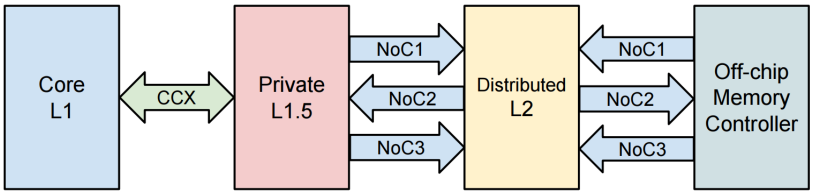


Figure 4.16: OpenPiton Memory Hierarchy Datapath [131]. Three physical NoCs, guaranteed to be deadlock-free by using a priority system, provides cache coherence across manycore architecture.

topology of processors. Three physical NoCs, shown in Figure 4.16, are defined to co-operate via a simple priority system to implement the cache-coherent protocol. Each link in each NoC is composed of a 64 b bus `data` and two binary control signals, `yummy` and `valid` (semantically equivalent to `*READY` and `*VALID` in an AXI channel [33, 131]). Integration of a correlator device is intended to be a simple process requiring little platform-specific knowledge, therefore no logic is added to the monitoring probes; i.e. no attempt is made to decode the cache-coherence or flow-control protocols. Instead, a single top-level RTL file is modified to programmatically attach a probe to each of the `valid` signals. In contrast to the stand-alone correlator characterised in Section 4.3.4, these case studies connect probes to logical RTL signals rather than external wires. To minimise the effect on synthesis and PnR decisions made by Vivado when building OpenPiton+Ariane, a delay-line of five DFFs is placed in front of each probe input. These DFFs ensure that timing constraints from monitored signals are relaxed, thus mitigating issues where addition of the correlator design might impact the achievable $f_{\max}$ relative to the parent system.

4.4.1 Case Study 1: Slow Datalink

The purpose of the "slow datalink" case study is to demonstrate that the correlator device can monitor, analyse, and present information about the behaviour of binary SoC signals, even when behaviour changes so quickly that the datalink cannot support the result rate. The relatively slow datalink (USB-FS) means that retrieving counter data is limited to cases with low sampling rates and long windows. Correlation results from each engine are instead consumed either by viewing the brightness of an LED, or by using the oscilloscope to record the voltage potential on a header pin.

The main components and their connections in the experimental setup are shown in Figure 4.18. The OpenPiton+Ariane system requires a software image, in this case a Linux OS, to be present on the SD card in the VC707 development board.

When a bitstream is written via the "USB/JTAG" interface, the OpenPiton+Ariane system is started, a Linux OS it booted, and a serial terminal is provided over the "USB/UART" interface. A synthetic workload [6] designed to stimulate interaction between cores is started, control registers in the correlator are configured, then measurements are recorded using the oscilloscope.

Control registers are used to select which counter result or metric is presented by the $\Delta\Sigma$ modulator, as well as other configuration parameters such as sample rate and window length. In use as an interactive tool, printed plots and text cannot convey every aspect of the user experience. Using key presses, the control registers are easily changed, thus enabling interactive exploration of the design similar to how other lab tools are used with knobs and buttons. For example, by selecting the output as $\mathbb{E}[f_x]$ then sweeping over all values of `selectX`, it can be seen by the voltage level reported on the oscilloscope which signals are most active.

Figure 4.19 contains voltage plots recorded with the oscilloscope shown in Figure 4.18. The observed correlations are, naturally, dependent on the type of work the system is performing. Changing the workload, by running different

[6] The synthetic workload is a group of shell scripts which forces processes to transfer data between themselves via Linux FIFO files. Source code is available online at: `https://github.com/DaveMcEwan/dmpvl/tree/master/prj/corrOPA/dummyload`

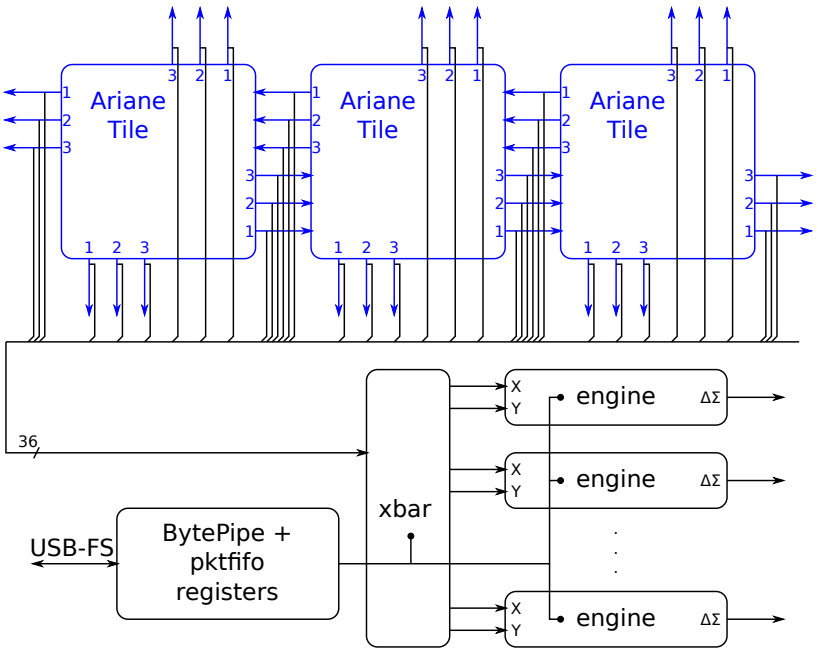


Figure 4.17: Logical connections of case study 1. Correlator and its probes (black) passively monitor the OpenPiton+Ariane parent system (blue) by connecting to the `valid` signals of inter-tile cache interfaces.

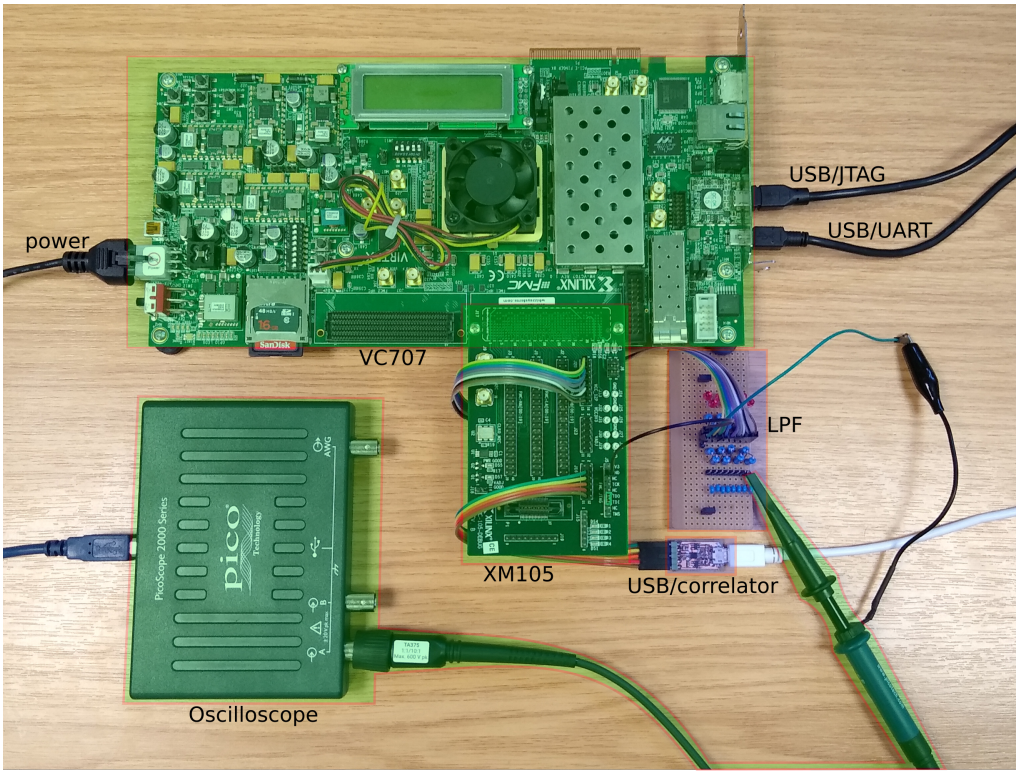
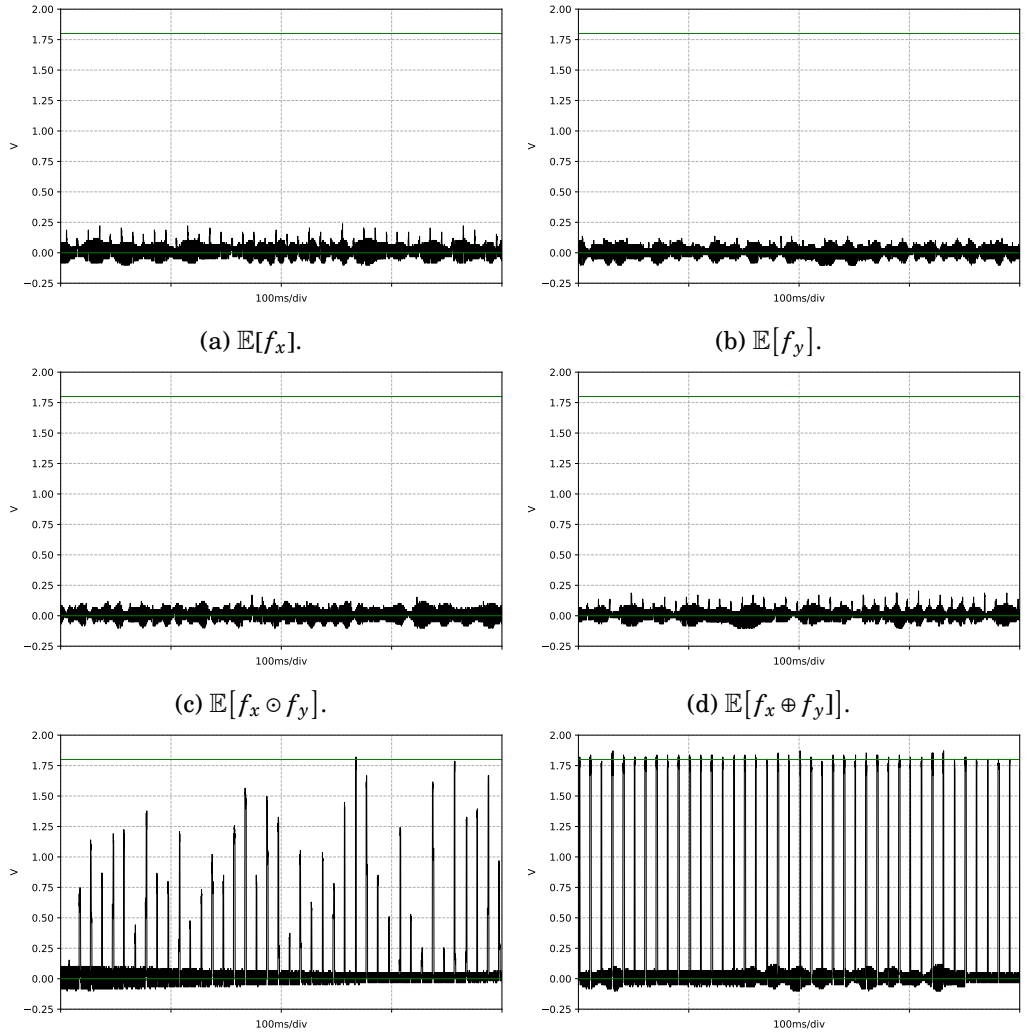


Figure 4.18: Experimental setup of case study 1 - Demonstrating correlator usage where results are presented via $\Delta\Sigma$ modulated GPIO pins. Blue highlights indicate custom built components and green highlights off-the-shelf components. The "USB/JTAG" cable is used only to configure the FPGA with a bitstream. The SD card in the VC707 contains a pre-compiled Linux image. The Linux OS running on the OpenPiton+Ariane system provides a serial terminal over the "USB/UART" cable. The "USB/correlator" cable is used to configure and interact with the correlator. Outputs from $\Delta\Sigma$ modulators exit the XM105 via the LPF to be measured by the oscilloscope.

workloads, produces different results showing changes in the way the tile's caches interact. Observing the plots in Figure 4.19, unremarkable values are seen in the individual counters ($\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$). Similarly, unremarkable values are observed for pairwise counters ($\mathbb{E}[f_x \odot f_y]$ and $\mathbb{E}[f_x \oplus f_y]$) which are difficult to discern from the noise floor. However, correlation results using either the $\dot{\text{C}}\text{ov}$ or $\dot{\text{D}}\text{ep}$ metrics display in sharp contrast the cache interface's periodic interactions.

Differences between metrics are apparent in the plots for $\dot{\text{C}}\text{ov}$ and $\dot{\text{D}}\text{ep}$ highlighting the utility of comparing different definitions of correlation. Figure 4.19f shows two signals which become highly dependent approximately every 10 ms for around 2 ms. As each spike in $\dot{\text{D}}\text{ep}$ is around the same level, close to 1, further periodic aspects of correlation cannot be seen clearly using only $\dot{\text{D}}\text{ep}$. In contrast,



(a) $\mathbb{E}[f_x]$.

(b) $\mathbb{E}[f_y]$.

(c) $\mathbb{E}[f_x \odot f_y]$.

(d) $\mathbb{E}[f_x \oplus f_y]]$.

(e) $\dot{\mathrm{C}}\mathrm{ov}(f_x, f_y)$.

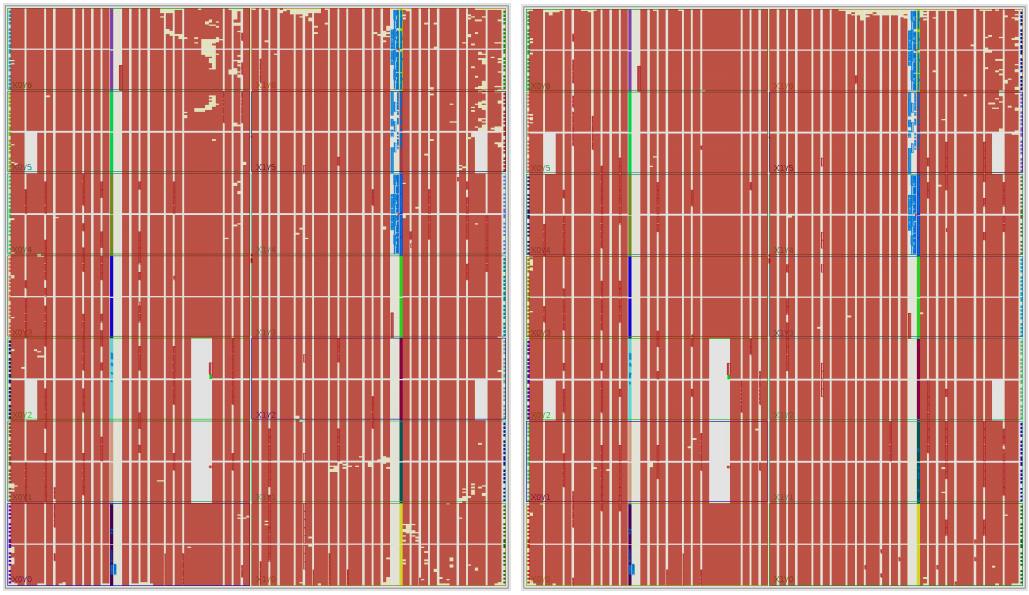
(f) $\dot{\mathrm{D}}\mathrm{ep}(f_x, f_y)$.

Figure 4.19: Oscilloscope plots visualizing counter values and associated correlations where f_x = `system.chip.tile_0_0_out_E_noc2_valid` and f_y = `system.chip.tile_0_1_out_E_noc2_valid`; i.e. eastern ports of NoC2 on the left and middle tiles. Green rulers highlight 0 V and 1.8 V levels. Correlator register configuration is the same for all six subplots: `sampleRate` = 48 MHz, `windowLength` = 2^{15}. Unremarkable signals, are seen on outputs for performance counters $\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$. Similarly, inconspicuous values are seen for intersection and symmetric difference counters ($\mathbb{E}[f_x \odot f_y]$ and $\mathbb{E}[f_x \oplus f_y]]$). In contrast, signals presenting correlation using $\dot{\mathrm{C}}\mathrm{ov}$ and $\dot{\mathrm{D}}\mathrm{ep}$ metrics show pronounced periodic voltage spikes. Spikes in the $\dot{\mathrm{D}}\mathrm{ep}$ results are consistently close to 1.8 V (i.e. a correlation result close to 1), whereas spikes in $\dot{\mathrm{C}}\mathrm{ov}$ are more variable.
These results show that the level of activity ($\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$), while similar to the level of coincidences ($\mathbb{E}[f_x \odot f_y]$ and $\mathbb{E}[f_x \oplus f_y]]$), is quite different from the level of interactions ($\dot{\mathrm{C}}\mathrm{ov}$ and $\dot{\mathrm{D}}\mathrm{ep}$) between these two cache interfaces.

Figure 4.19e shows that while these signals become highly covariant with the same time profile, there are differing levels of covariance between the spikes. A Linux OS is composed of many threads which run and suspend periodically, coordinated by a scheduler. Differences between the observed $\dot{\text{C}}\text{ov}$ and $\dot{\text{D}}\text{ep}$ may therefore be explained by differences in thread behaviour, but confirming this would require further exploration of the source code. The correlator device is intended to guide exploration of SoC behaviour rather than provide explicit explanations or reverse-engineering capabilities, and as such is meeting its design intentions.

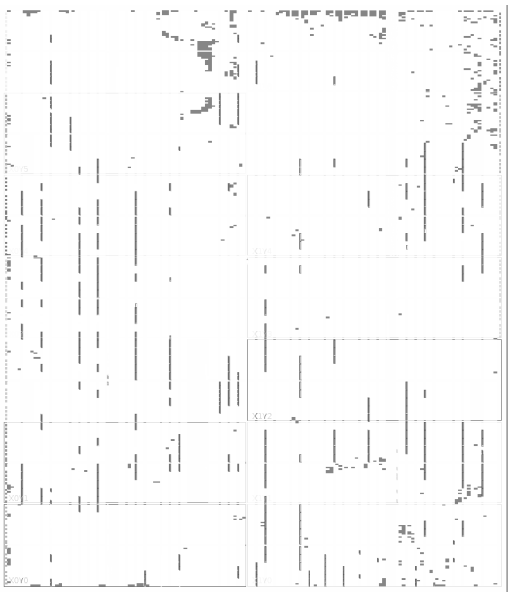
Evidence that the correlator design permits straightforward integration with a parent system running on an FPGA is shown in Figure 4.20. The correlator design is intended to avoid synthesis issues and placement issues by using only generic logic constructs, requiring few logic resources, and having loose timing constraints. Technology constraints are demonstrably avoided because the design is implemented in two very different FPGA technologies, described in Section 4.3.4. Placement issues are also shown to have been avoided in Figure 4.20c where it can be seen that the correlator logic has filled in "gaps" in the parent system. Timing constraints do not present an obstacle to synthesising the integrated system, with no changes required to the default f_{max} of the OpenPiton+Ariane system, 50 MHz.

This case study features the parent system OpenPiton+Ariane with an integrated on-chip correlator that passively monitors the cache interfaces between CPU tiles. Validation is shown that the correlator design can be readily integrated with a complex parent system. Insights into the system's behaviour can be gleaned by performing real-time on-chip correlation analysis and viewing a lossy representation of results with an oscilloscope. The example data shown in Figure 4.19 tells us that a pair of cache interfaces periodically interact in a highly dependent manner, but the difference between $\dot{\text{C}}\text{ov}$ and $\dot{\text{D}}\text{ep}$ suggests that one is not simply mirroring the other. Using the correlator as a tool to guide exploration of system behaviour, an engineer might therefore be motivated to investigate the reasons behind this behaviour, thus providing assistance in understanding their system.



(a) Without correlator.

(b) With integrated correlator.



(c) Highlighted differences.

Figure 4.20: Floorplans of OpenPiton+Ariane on VC707 with and without integrated correlator device. LUT utilisation is high at 81% without and 83% with the correlator. Highlighting differences (dark) in the floorplan shows that the design is very similar (mostly white). This confirms that adding the correlator has not forced Vivado to make drastically different synthesis or placement decisions because the correlator does not impose difficult constraints on timing or placement.

4.4.2 Case Study 2: Fast Datalink

In the previous section, a correlator is integrated into a parent system as a way of monitoring and analysing the behaviour of cache interfaces. However, correlation analysis is only one tool amongst many aimed at helping SoC engineers to understand and improve their systems. UltraSoC is the leading supplier of commercial SoC embedded analytics tools and the industrial sponsor of this research. Their product portfolio is centred around RTL modules for passive monitoring of specialised aspects of SoC behaviour, and the surrounding sub-system required for configuration and to extract data. These modules include a Bus Monitor consisting of performance counters for interconnect transactions (AXI, OCP, etc), a Static Instrumentation block to assist software profiling, and execution trace encoders for embedded CPUs (RISC-V, CEVA, ARM, etc). Missing from their offerings, and indeed the offerings of all major silicon IP suppliers, are modules supporting real-time correlation analysis. While cross-correlation can be calculated using a succession of performance counter values over many time windows, this is a different type of analysis from the single-window analysis considered in this work. In this case study, the correlator is implemented as part of an UltraSoC sub-system enabling use with UltraSoC's standard RTL and software set of tools. This piece of work, assisted by engineers at UltraSoC, demonstrates the viability of the correlator device in a commercial setting where flexibility, low resource usage, and toolchain compatibility are paramount.

The primary advantages of this case study over Section 4.4.1 are out-the-box software support for recording data and faster datalink; i.e. USB High Speed (480Mb/s) (USB-HS) vs USB-FS. An existing UltraSoC module, the Status Monitor [92], is augmented to include correlator engines alongside the existing features. The Status Monitor uses a flexible arrangement of comparison, sequencing, matching, and filtering steps to derive a number of "qualifiers" which are used as input to performance counters in place of the probe wires. Using the powerful qualifier logic, complex conditions between input wires can be created at runtime, instead of requiring a specific input wire for each condition. Integration of the correlator into an UltraSoC component makes use of the qualifier logic, treating each qualifier wire as a probe.

Figure 4.21 shows the internal organisation of the augmented UltraSoC Status Monitor which is used in place of the correlator structure in Figure 4.17 (drawn in black). The USB-FS datalink, control registers, and input crossbar are replaced with UltraSoC components (USB-HS datalink, message-based sub-system, and match/filter/qualifier logic). Implementation characteristics on the same FPGA,

such as logic resource requirement, layout flexibility, and achieveable $f_{\max}$ are very similar to the case study in Section 4.4.1.

Instead of using an oscilloscope to record data in analogue form, the faster datalink is used to retrieve counter values from correlator engines, thus enabling the offline calculation of any counter-compatible correlation metric. Section 4.3.2.1 explains how scaling at the point of increment enables the volume of data on the datalink to be dynamically adjusted by dropping the least significant bits. Transmitting only the most significant bits from the counters means that precision is lost, but that the mechanism for supporting a slower datalink is straightforward. While the counters in the UltraSoC-like correlator engines are 20 b wide, only the most significant 8 b are actually recorded.

Figure 4.22 plots the correlation between the same pair of signals as Figure 4.19 with the OpenPiton+Ariane system running a similar workload. Without additional circuitry in the correlator device, calculation of other metrics, such as $\dot{\mathrm{T}}$mt, from the output of a single engine is not possible. However, access to the recorded values from the correlation counters allows other metrics to be used. Figure 4.22 demonstrates this by calculating $\dot{\mathrm{T}}$mt and two of the FFNN-based metrics learned in Chapter 3.

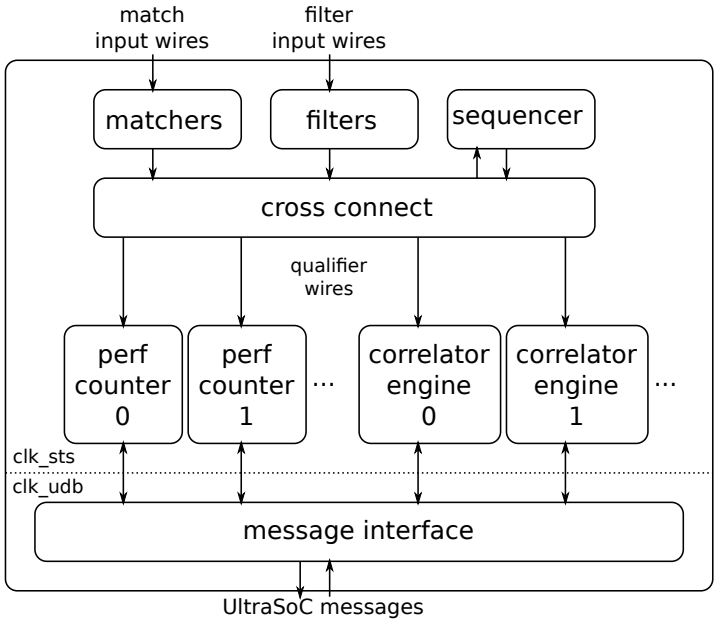
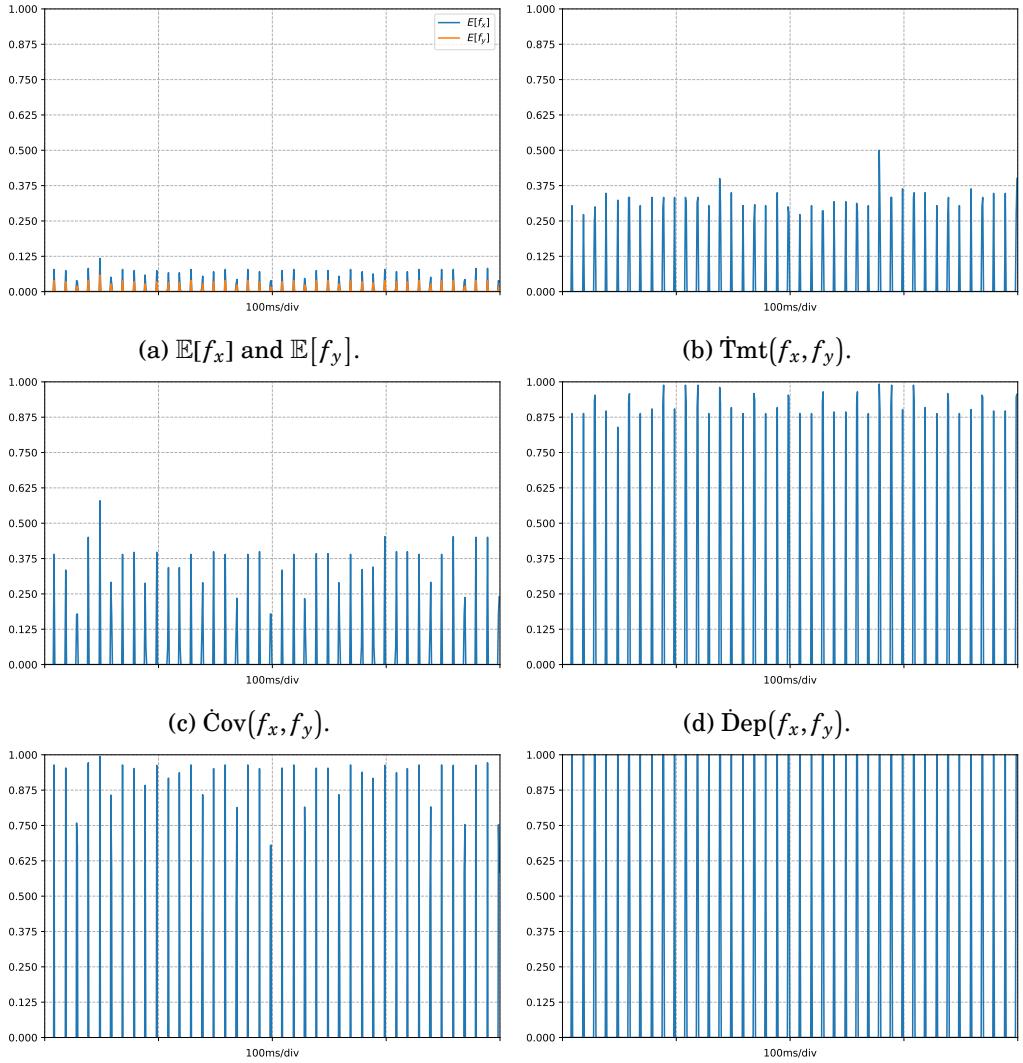


Figure 4.21: UltraSoC Status Monitor augmented with correlator engines. Other features of the Status Monitor including GPIO, data-tracer, accumulators, and interval timer are not shown here. Probe wires are replaced with qualifier wires to take advantage of the match, filter, and sequencing logic [92].



(a) $\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$.

(b) $\dot{\mathrm{T}}\mathrm{mt}(f_x, f_y)$.

(c) $\dot{\mathrm{C}}\mathrm{ov}(f_x, f_y)$.

(d) $\dot{\mathrm{D}}\mathrm{ep}(f_x, f_y)$.

(e) `withIsectSymdiff_4qsig_4qsig_sigm`(f_x, f_y). (f) `withAssist_8tanh_8tanh_sigm`(f_x, f_y).

Figure 4.22: Plots of data captured via the correlator and five calculated metrics of correlation. OpenPiton+Ariane system is running a similar workload and the same pair of signals are probed as in Figure 4.19. Values observed for the performance counters ($\mathbb{E}[f_x]$ and $\mathbb{E}[f_y]$) do show in-phase periodic low levels of activity which does hint towards correlation, but actual confirmation of correlation according to various interpretations is given by the other plots. Recorded counter values enable calculation of different correlation metrics, including the FFNN-based metrics trained in Chapter 3. As in Figure 4.19, $\dot{\mathrm{C}}\mathrm{ov}$ shows less consistent and lower values than $\dot{\mathrm{D}}\mathrm{ep}$. The ratio between union and intersection $\dot{\mathrm{T}}\mathrm{mt}$ is different again, highlighting the difference in the various interpretations of correlation. Vertical axes are labelled underneath each plot.

This "fast datalink" case study validates that the correlator design can provide data suitable for correlation analysis over a datalink capable of supporting the rate of results. Additionally, use of the correlator design is demonstrated to integrate well with an existing toolchain and product line from the market-leading vendor of embedded analytic solutions. Plots in Figure 4.22, particularly `withIsectSymdiff_4qsig_4qsig_sigm` and `withAssist_8tanh_8tanh_sigm`, demonstrate that the data retrieved via the correlator design is suitable for use with esoteric correlation metrics such as the FFNN-based ones discussed in Chapter 3.

4.5 Conclusion

With the ever-increasing complexity of modern SoCs, designers and engineers require increasingly sophisticated analysis tools. As explored in Chapter 3 correlation analysis provides a useful tool to fill knowledge gaps like "Why does X happen?" by rephrasing the problem to "What is X correlated with?". Capturing and monitoring correlations between SoC signals can be used to answer these questions, thus providing a means of guiding system understanding or detecting anomalous behaviours. Real-time estimation of correlation, by a variety of interpretations, is therefore a valuable tool for engineers attempting to understand how complex systems, composed of both hardware and software components, behave under different scenarios. With further understanding of nuanced system behaviour, SoCs designs can be further optimised to gain competitive advantage.

Chapter 3 uses a synthetic dataset based on a probabilistic model to assess the usefulness of various metrics based on different interpretations of correlation. This chapter examines how those metrics can be used to analyse RTL designs for both real-time on-chip monitoring and data collection for off-chip processing. Chapter 5 then explores how that collected data can be presented for effective visual analysis.

Key points and contributions of this chapter are thus:

1. Exposition on the issues around hardware, particularly RTL, support for correlation analysis. Special attention is paid to the case of on-chip real-time monitoring for embedded analytics.

2. Jittery sampling mechanism enabling low-cost analysis over a range of δ offsets.

3. Novel window function Logdrop defined in Equation (4.6), specifically designed for low-cost implementation in digital logic.

All RTL code used to design the circuitry, written in SystemVerilog (IEEE 1800-2005), is made available online. [7]

The objective of this chapter, to explore the issues around hardware support for correlation analysis of binary SoC signal, is achieved through the RTL design of an on-chip correlator device. The three issues of sampling, window functions, and integration to a parent system are first described in Section 4.2 before being addressed in Section 4.3 then validated in Section 4.4. The main issue with sampling in SoC correlation is that the question "Is X correlated with Y?" will be reasonably interpreted with the appendix "within a range of time offsets", leading to difficulties in scaling the required computation when the range may be thousands of cycles. By using a sampling strobe with a Gaussian-distributed period to statistically combine a range of δ-offset signal versions into one sample population, some accuracy can be sacrificed to gain a result which better answers the question. Window functions, as used pervasively in Fourier analysis, are useful to apply a time-dependent weighting to a set of samples for modelling the process of visual foveation; i.e. paying more attention to the centre of vision than the periphery. In Fourier analysis applications, the cost of additional hardware for sophisticated coefficient calculation or a memory is weighed against the benefit brought by a particular frequency response. However, in correlation analysis the cost/benefit calculation is different. In order to avoid the need for a memory to store coefficients, a window function is desired which permits low-cost calculation of each coefficient from the window index. A new window function Logdrop is presented which permits simple RTL implementation of this calculation. Logdrop is a novel window function well-suited to calculation with digital logic because all components can be implemented using multiplexers and zero-cost bit-shifts. The use of increment-scaling counters is explained to support runtime configuration of window length, and their use with rectangular and Logdrop window functions is illustrated. Practical integration with FPGA-based parent systems is seen as a critical design goal for useful on-chip monitoring hardware. Issues including technology, timing, and layout constraints are avoided by first implementing the correlator as a stand-alone device and performing characterisation using the methodology described in Appendix E, and examining the floorplans of a variety of FPGA configurations. The first case study confirms that the characterisation effort has resulted in a design which can be integrated with a complex parent system.

[7] All RTL is in a single git repository available at `https://github.com/DaveMcEwan/dmpvl`. The stand-alone correlator device, described in Section 4.3, corresponds to the sub-project `dmpvl/prj/correlator`. Integration with OpenPiton+Ariane including the UltraSoC module variation, described in Section 4.4, corresponds to the sub-project `dmpvl/prj/corrOPA`.

Also confirmed by the first case study is that the correlator device is suitable for real-time on-chip correlation monitoring, even when a fast datalink is not available. Correlation, as calculated via the $\dot{\text{C}}$ov and $\dot{\text{D}}$ep metrics, is presented in analogue form using an off-the-shelf oscilloscope which requires only a single GPIO pin. The second case study, integrating the correlator into an UltraSoC sub-system, demonstrates the utility as a data-collection tool and the viability of the design within the market-leading embedded analytics portfolio. To gain alternative views of system behaviour, collected data provides the ability to apply different metrics, including esoteric methods such as FFNN-based metrics. Chapter 5 extends the concept of alternative views quite literally to present novel visualisations of signal behaviour using collected data.

CHAPTER 5

VISUALIZING PAIRWISE CORRELATIONS

5.1 Aim

This chapter is concerned with the visual presentation of correlations to assist in understanding SoC behaviour. State-of-the-art techniques described in Chapter 2, correlation metrics explored in Chapter 3, and comparisons with existing tools are used to guide the development of presentation techniques. While Chapter 4 deals with real-time analysis and data collection from physical systems, this chapter focuses on presenting correlations in pre-collected data. Material from this chapter has previously appeared in [132].

5.1.1 Problem Description

Where traditional methods of SoC analysis such as formal assertions and runtime checks work well for individual modules and small systems with well-defined usage, large complex systems with dynamic behaviour present a significant challenge. Behaviour is a set of qualities which are difficult to quantify and assess due to their high-level and potentially vague nature. For example, “CPUs 1 and 2 should work together but separately from CPU 3” describes the intended behaviour of a multi-processor SoC supporting two concurrent tasks. Precisely defining the phrase “work separately” may be difficult and depend on presently undefined properties such as software state.

In the high-level design of a complex system, the designer might have an idea of what the behaviour should be for the intended application, but a precise

definition would require too much time and effort; i.e. a real-world cost in terms of resource usage and lost earnings due to a delay in time-to-market. An intuitive method of visualisation enables the designer to imagine how a well-behaved system should present. With this expectation, the designer is no longer required to explain detailed properties; Instead, they can simply contrast their mental picture with the one presented. This is similar to the advantages of using a logical schematic in digital design; i.e. an experienced digital design engineer will be able to quickly extract key features (deep logic, number of DFFs, combinatorial loops) by looking at a diagram more easily than reading pages of RTL code. Easing the mental burden therefore allows a more efficient workflow.

Each correlation metric, as defined in Chapter 3, gives results on the unit interval for each time window. Naturally, each pairwise correlation may be plotted like any other time-varying scalar, e.g. Figure 4.22, and multiple results may be drawn together in a multi-plot. As the number of interesting signals m grows, the number of pairwise correlation results (including auto-correlations) grows by $\frac{m^2+m}{2}$. The correlator device of Chapter 4 implements a number of computation engines, each of which calculates one pairwise correlation. For example, calculating a single metric for every pair within $m = 50$ signals would require 1275 correlator engines. Even in a very small example with $m = 5$ signals, 15 plot-lines are likely too much to keep track of mentally. Where hardware resources are plentiful and the number of correlation engines is large, the limiting factor is the usability of overcrowded multi-plots.

Effective visualisations present information with satisfactory precision and accuracy in a manner such that the viewer can easily (1) understand what features are desirable, (2) achieve an overview, (3) identify points of interest, and (4) selectively ignore parts which are less interesting [133]. The effectiveness and usability of information visualisation techniques are readily evaluated using heuristics [134], closely related to the field of HCI. Heuristics provide a set of issues which must be carefully addressed to design effective presentations [9][59]. Critical evaluation of the currently available tools and techniques for correlation analysis of binary SoC data, hint that this is an area ripe for improvement. For example, waveform viewers – specialised multi-plot viewers – adhere closely to the “B5 Information coding” heuristic [60] in the context of presenting when signals assert over the time domain. However, no realistic characteristic/technique is apparent to suggest how related those assertions are. Other analytical tools such as corrgrams [67] and source code annotations [133] are found to be similarly lacking.

The data required to be presented for effective correlation analysis has many dimensions, i.e. $\frac{m^2+m}{2}$ pairs over different metrics, time offsets, and time windows.

Devising methods to present these many different aspects is a difficult task, for which no adequate solution is currently available. This chapter aims to address deficiencies in currently available methods of visualisation for pairwise correlations. Currently available tools and techniques are addressed, then a set of techniques are proposed to improve upon the state-of-the-art.

5.1.2 Objective

Effective visualisations enable system engineers to better understand the behaviour of their systems, thereby allowing faster iteration over ideas and improvements. By speeding up the process of understanding the behaviours of a system, engineers can meet their design goals faster and provide a competitive edge.

The objective of this chapter is to examine the particularities of effective visualisation in correlation analysis of binary SoC data. A fuller knowledge of the requirements thus enables improvement upon the state-of-the-art and, ultimately, the design of more useful tools for SoC work.

5.1.3 Approach

Chapter 3 examines several interpretations of correlation in binary SoC data, and Chapter 4 explores the practicalities of extracting correlation data from a running system. Correlation analysis, used to understand SoC behaviour, requires not only mathematical foundations and practical methods of accessing results but also requires effective ways of putting that knowledge into an engineer's head. A discussion of common statistics and their relevance to binary SoC data is given in Section 5.2.1 to uncover the most useful information for presentation. For example, the use of gradient analysis is shown to be particularly intuitive due to the nature of clocked digital logic. Information visualisation heuristics, as discussed in Chapter 2, are used to critically evaluate proposed techniques based on network/graph and table data models. The widely used heuristics for HCI evaluation, first given by Nielsen [53], are not entirely applicable to this work, so these are compared with more recent sets of heuristics designed specifically for information visualisation. With a further selection of suitable heuristic sets [55, 56, 60] and knowledge of what information should be presented, a novel method for correlation visualisation is developed along with a browser-based tool [1] to produce examples of its application.

[1] All Python code is available in a git repository at `https://github.com/DaveMcEwan/dmppl/tree/master/dmppl/experiments/eva`.

This chapter focuses on scenarios where data has been collected in advance of analysis, such as through simulation or instrumentation. Examples from two case studies are used to demonstrate the described visualisation techniques. A simple SoC featuring a single AXI interconnect [33] is used to demonstrate visualisations of stationary behaviour. Presentation of behavioural changes over time are demonstrated using a dual-processor machine learning system with lightweight software instrumentation. These case studies reflect two very different types of system which are both applicable to modern SoC engineering, which demonstrates that the method is not limited to one particular level of abstraction or niche usage.

Working alongside a synergistic colleague who answers questions is a useful analogy to an effective analysis tool because they will understand the appropriate level of detail to provide. Providing too little detail leads to a frustrating amount of further questioning, but providing much detail up-front can swamp our attention, which is equally frustrating. Shneiderman [135] argues that the most effective tools for information analysis function as intelligent but subservient co-workers. Working with colleagues who provide the information we want with minimal mental effort required to filter out extraneous data is the desired experience in all fields of work. The subservient aspect is also crucial for effective use, because if a tool contains black-box functionality which cannot be rationalised by the user, then the user cannot feel in control of decisions based on that tool. As such, the visualisation techniques presented in the following sections work on the principle that they should relieve mental pressure while being fully explainable.

5.2 Visualisations for Binary SoC Data

A novel method for visualising correlations in binary SoC data is described through a tool's development in alignment with Shneiderman's mantra. The input format is VCD which is the lingua franca for SoC data, which is chosen to avoid preprocessing and import issues, as explained in Section 2.3.2. Output is presented via a browser interface, which goes some way to supporting several heuristics related to status, orientation, recovery, and interactivity. The web browser environment is familiar with standard methods of interaction such as the Uniform Resource Locator (URL) for absolute navigation, clickable hyperlinks for relative navigation, forward/back buttons for a history of views, and zooming. The familiarity of environment supports ten heuristics by running on a sufficiently powerful machine (N1, N3, N5, N7, N9, S2, S3, S6, GP1, FJ3, FJ4), i.e. inordinate effort would be required to violate these heuristics.

5.2.1 Statistics on Binary Signals

In other scientific fields, several derived measurements and statistics are commonly given, such as area under curve, arithmetic mean, variance, and gradient. The area under a binary curve is equivalent to the number of times where $f_i \in \mathbb{B}$ is asserted; i.e. $\sum_t f_i[t]$. However, as discussed in Section 4.3.2, using a window function gives a result focused on the window centre; i.e. weighted expectation $\mathbb{E}[f_i]$.

Where measurements are subject to significant noise, more robust statistics are also commonly given, such as the median, interquartile range, and uncertainty. In this work, focused on binary signals whose accurate values are gleaned from simulation or runtime instrumentation, robust methods are not considered necessary. It is straightforward to calculate the median from the expectation, and thereby see the limited value it provides:

$$f_i[t] \in \mathbb{B} \implies \text{Median}(f_i) = \begin{cases} 1 & : \frac{1}{2} < \mathbb{E}[f_i] \\ \frac{1}{2} & : \frac{1}{2} = \mathbb{E}[f_i] \\ 0 & : \text{otherwise} \end{cases}$$

The weighted arithmetic mean is equivalent to the weighted expectation $\mathbb{E}[f_i]$ which tells us the proportion of times where the signal is asserted. Variance of a signal f_i is equivalent to covariance with itself; i.e. $\dot{\text{Cov}}(f_i, f_i)$. For real-valued signals variance is a useful statistic, telling us how much a signal value varies from the usual, but for binary signals $\dot{\text{Cov}}(f_i, f_i)$ provides no additional information over the expectation:

$$\begin{aligned} \dot{\text{Cov}}(f_i, f_i) &= 4 \left| \mathbb{E}[f_i \odot f_i] - \mathbb{E}[f_i]\mathbb{E}[f_i] \right| \\ f_i[t] \in \mathbb{B} &\implies \mathbb{E}[f_i \odot f_i] = \mathbb{E}[f_i] \\ &\implies \dot{\text{Cov}}(f_i, f_i) = 4 \left| \mathbb{E}[f_i] - \mathbb{E}[f_i]^2 \right| \end{aligned}$$

The gradient of a binary signal is less straightforward but is used intuitively by referring to the rising and falling edges. For example, "`cpu.idle` *high* leads to lower power consumption" or "`cpu.idle` *rising* leads to increased cache activity". Both statements refer to the same signal `cpu.idle`, but it is unlikely that lower power consumption leads to increased cache activity. Indeed, it is so intuitive for SoC engineers to refer to the reflection, rising and falling edges, that waveform viewers often feature methods of showing these sibling signals [61]. This work uses the

following definitions for the reflection, rising edge, and falling edge, respectively.

$$\neg_i[t] := 1 - f_i[t] \tag{5.1}$$

$$\uparrow_i[t] := \max(0,\ f_i[t] - f_i[t-1]) \tag{5.2}$$

$$\downarrow_i[t] := \max(0,\ \neg_i[t] - \neg_i[t-1]) \tag{5.3}$$

The rising and falling edges are closely related to the first-order derivative. However, for binary signals, it does not make sense to define functions related to higher-order derivatives because $\uparrow_i[t] = 1 \implies \uparrow_i[t+1] = 0$. Taking the rising edge of a rising edge simply produces a copy, i.e. $f_j = \uparrow_i \implies \uparrow_j = \uparrow_i$. The constructions in Equation (5.2) and Equation (5.3) mean that $\mathbb{E}[\uparrow_i], \mathbb{E}[\downarrow_i] \leqslant \frac{1}{2}$.

Expectation of the rising edge $\uparrow_i[t]$ is a useful statistic which tells us how often $f_i[t]$ begins to assert. In addition to knowing how often a binary state asserts, it is also useful to know how long those assertions can be expected to last. The ratio of rising edges to assertions $\frac{\mathbb{E}[\uparrow_i]}{\mathbb{E}[f_i]}$, equivalent to the conditional expectation $\mathbb{E}[\uparrow_i | f_i]$, provides the expected length of assertions. A higher value means that assertions are of shorter duration where the maximum value $\mathbb{E}[\uparrow_i | f_i] = 1$ means that all assertions of f_i last only a single clock cycle. Where it is more intuitive to speak of the negated signal, $\mathbb{E}[\downarrow_i]$ and $\mathbb{E}[\downarrow_i | \neg_i]$ may be used instead to say how often f_i is de-asserted and how long it remains de-asserted.

Binary signals can be categorised by their intended behaviour based on specific design knowledge. Events are signals which are asserted to indicate the occurrence of something interesting at a specific discrete time, e.g. $f_{\texttt{cache\_miss}}$ indicating that a read was made to an uncached address. A binary state, shortened to "bstate", is asserted to indicate that some bit of a system is in a specific state, e.g. $f_{\texttt{rd\_waiting}}$ indicating that a requester is presently waiting for data. The difference between events and bstates is somewhat arbitrary, but the suggested guideline is: Signal names which are either an adjective or present participle refer to bstates, and signal names which are nouns refer to events. The purpose of this distinction is that when speaking naturally about signals, events are not typically discussed in terms of their reflection or edges. For example, in a single-CPU system, $f_{\texttt{cache\_miss}}$ would be expected to be asserted in pulses one cycle wide, so speaking of the rising edge of $f_{\texttt{cache\_miss}}$ is unnecessary because $f_{\texttt{cache\_miss}} = \uparrow_{\texttt{cache\_miss}}$. On the other hand, bstates are usefully discussed by how long assertions perpetuate.

Statistics on individual signals are useful in exploratory correlation analysis because they allow the viewer to frame the context of correlation results. Presenting these individual statistics alongside pairwise correlation results allows the viewer to gain an overview and make better reasoned decisions more quickly than

if these data require separate distinct steps. Avoiding mental overhead in this way is the basis of several usability heuristics including N7 (flexibility and efficiency of use), S5 (view relationship hips among items), and GP3 (fuse data). In Section 5.2, statistics are presented via the colour of graphical nodes or table cells, thereby allowing the user to quickly comprehend the context of accompanying correlation results.

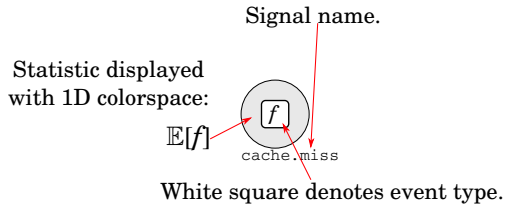
The application of a correlation metric is written as $\cdot(f_x, f_{y\langle\delta\rangle})$ where $\cdot$ is the metric's name, δ is a time offset, and both f_x and f_y are bit-vectors. As described in Section 5.2.1, it is natural to speak of a bstate f_i signal in terms of its sibling signals (f_i, $\neg_i$, $\uparrow_i$, and $\downarrow_i$), therefore a set of m signals may require analysis of up to $4m$ bit-vectors, depending on how many are bstates rather than events. Fully computing the results of n metrics therefore requires up to $n\Delta(8m^2+2m)$ calculations. For context, applying the 6 metrics identified in Chapter 3 to a system with 50 bstate signals and a maximum δ offset of 10 cycles requires $1,206,000$ calculations per time window. Given the large number of required calculations, effective visualisations must be selective in what is displayed to avoid overwhelming the viewer. For this reason, the number of metrics presented in any single visualisation is restricted to either $n = 1$, displayed with a greyscale colourspace, or $n = 2$ using the colourspace defined in Appendix I.

5.2.2 Netgraph View

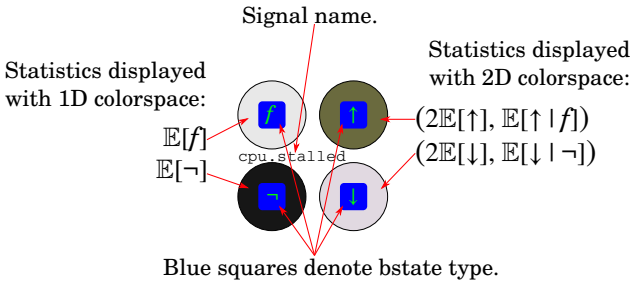
5.2.2.1 Statistics on Individual Signals

For the viewer to understand the context of signal correlations, an overview of relevant statistics on the individual signals must be known (S1). Relevant statistics are displayed through the colour of large circular nodes which can be hovered over to summon a pop-up display with more details. The basic layout is described in Figure 5.1.

Sibling nodes are always laid out in the same orientation with the signal's name below the f node (GP2, GP5). Consistent orientation layout of these circular nodes and the blue/white coloured squares helps the user to distinguish between event and bstate signal types, thus adhering to the relevant heuristics (N4, N6, N8, GP2, GP6, GP10, FJ1, FJ5, FJ6, FJ7). Using darker colours to represent more significant values is a useful principle because it allows the same diagrams to be reproduced on printed paper. The largeness of the nodes (relative to the font used for sibling and signal name) provides visual redundancy and enables the statistic values to be gleaned even in a zoomed-out view (S1, S2, FJ4). No heuristics are violated in presenting statistics on individual signals via coloured nodes.



(a) Visualisation of an event signal's statistics. Darkness of the circular node allows the viewer to quickly estimate how often the event occurs.



(b) Visualisation of a bstate signal's statistics. Darkness of the left-hand circles show the proportion of time the signal is asserted. Right-hand circles are coloured using the 2D colourspace described in Appendix I.

Figure 5.1

A bstate signal has six relevant statistics to present about its four sibling signals. Presenting the expected values of both $\mathbb{E}[f_i]$ and $\mathbb{E}[\neg_i]$ in greyscale provides redundancy (GP2, GP5, GP10). Additionally, presenting both the signal and its reflection allows the viewer to make natural connections with different thought processes, e.g. thinking in terms of `busy` vs `not busy`, thus adhering to N2 and FJ1. Appendix I describes a 2D colourspace useful for representing a pair of unit-interval values by a single colour. On each bstate, the colour of the upper-right node indicates both the number of rising edges, and the average length of assertion. Similar interpretations about the abundance and length of de-assertions are enabled by the $\downarrow$ nodes in the lower-right position. There are four extreme points in the colourspace:

white Both $\mathbb{E}[\uparrow]$ and $\mathbb{E}[\uparrow|f]$ are close to 0, i.e. few long-lived assertions.

green $2\mathbb{E}[\uparrow]$ close to 1 and $\mathbb{E}[\uparrow|f]$ close to 0, i.e. many long-lived assertions.

black Both $2\mathbb{E}[\uparrow]$ and $\mathbb{E}[\uparrow|f]$ are close to 1, i.e. many pulse-like assertions. The extreme case is a square wave with a minimal period.

purple $\mathbb{E}[\uparrow]$ close to 0 and $\mathbb{E}[\uparrow|f]$ close to 1, i.e. few short-lived assertions.

Combining the presentation of sibling statistics using the 2D colourspace supports heuristics related to minimalism and data fusion (N8, S7, GP3, GP4, GP8, FJ1, FJ10).

5.2.2.2 Identicons as Aids for Orientation

Intuitive interactive exploration depends on compatibility with zooming (S2) between an overview (S1) and a detailed view (S4). The text of signal names is not compatible with zooming out because it is not usable when viewed from far away. Allowing the user to maintain their mental orientation while changing views is enabled by coupling each signal with a unique and easily recognisable glyph. Identicons were first proposed by Park [136] as a method for providing simple, easily distinguishable user icons for online forums. Several prominent websites have since adopted similar methods of icon generation including Wordpress and Github, which informally validates their usefulness.

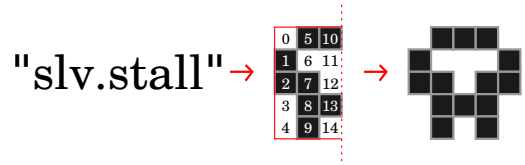


Figure 5.2: Identicon derived from signal name.

To create an identicon, 15 b are extracted from the signal name's MD5 hash, then used to fill a 3×5 grid of squares correspondingly. As humans, we find vertically symmetric figures easily recognisable, even the simple aliens of the computer game "Space Invaders", which is related to our ability to recognise faces [137]. Horizontally mirroring the 3×5 grid over the rightmost column produces a vertically symmetric identicon. As the resulting identicons often bear a resemblance to faces, bodies, castles, or other familiar shapes, assigning an identicon to each signal provides a strong visual aid for the viewer to quickly identify a signal (N6, FJ7). Tapping into the user's ability to efficiently recognise vertically symmetric shapes thus supports N6 (equivalent to FJ7).

The utility of identicons in exploratory analysis is further demonstrated in the next sections. Applying critical analysis to the association of signal names to identicons, it can be argued that their purpose may not be immediately apparent to new users, which would violate N2 and potentially GP4. To mitigate the effects of these potential usability issues, the presence of identicons must not interfere with the presentation of other data, i.e. a user confused by their purpose should be able

to completely ignore them with minimal effort. In the netgraph view described in the next section, identicons are placed outside the circle of signal nodes, thus easily ignored but available as a visual aid if desired.

5.2.2.3 Circular Graphical Layout

Extending the use of coloured circles to present statistics on all m signals simply involves placing all nodes (or node quads for bstates) on the same figure. A circular layout with each signal's position determined by alphabetical order is chosen instead of another layout (e.g. grid) for two reasons: (1) edges can be drawn between the nodes of any two signals; (2) visual redundancy is provided by the relative angles between nodes and the accompanying identicons, such that a user can identify a location on the circle even from a zoomed-in view containing only one node. For example, Figure 5.3 shows an overview of a system's correlations, and the position of the zoomed-in portion shown in Figure 5.4a is made clear via the relative angles of other signals and the identicon.

Correlations between metrics are represented by the colour and thickness of connecting edges, i.e. a heavier darker edge represents a higher correlation. This holds closely to Amar and Stasko's representational primacy and the related heuristics (N2, N6, N8, S5, GP6, FJ1, FJ5) because a logical connection between signals is presented with a visual connection. For a high-level understanding, knowing that a significant link exists is more important than knowing the exact values of correlation statistics, i.e. a rough estimate is often good enough. Using the 2D colourspace described in Appendix I either 1 or 2 metrics can be displayed at once, allowing the viewer to estimate an approximate value. If desired, then a precise value can be summoned using either the interactive features described in Section 5.2.2.4 or an alternative view described in Section 5.2.3.

A noticeable feature of Figure 5.3 is that the signal names are so small that they are practically unreadable on paper. However, viewed on a web browser these can be interactively zoomed in on, or hovered over to see precise details. This is in particular keeping with Shneiderman's mantra (S1, S2, S3, S4) in addition to GP8, because the familiar and ergonomic controls of a web browser enable a smooth coupling between what the user wants to know and what is displayed on screen.

The positioning of signals around the circle is given by arranging the signal names in alphabetical order, which is chosen for two reasons: (1) alphabetical order is simple for the user to rationalise; (2) alphabetical order is also useful for SoCs with regular naming schemes, e.g. hierarchical paths, because similarly grouped signals will be adjacent. The most interesting correlations are those

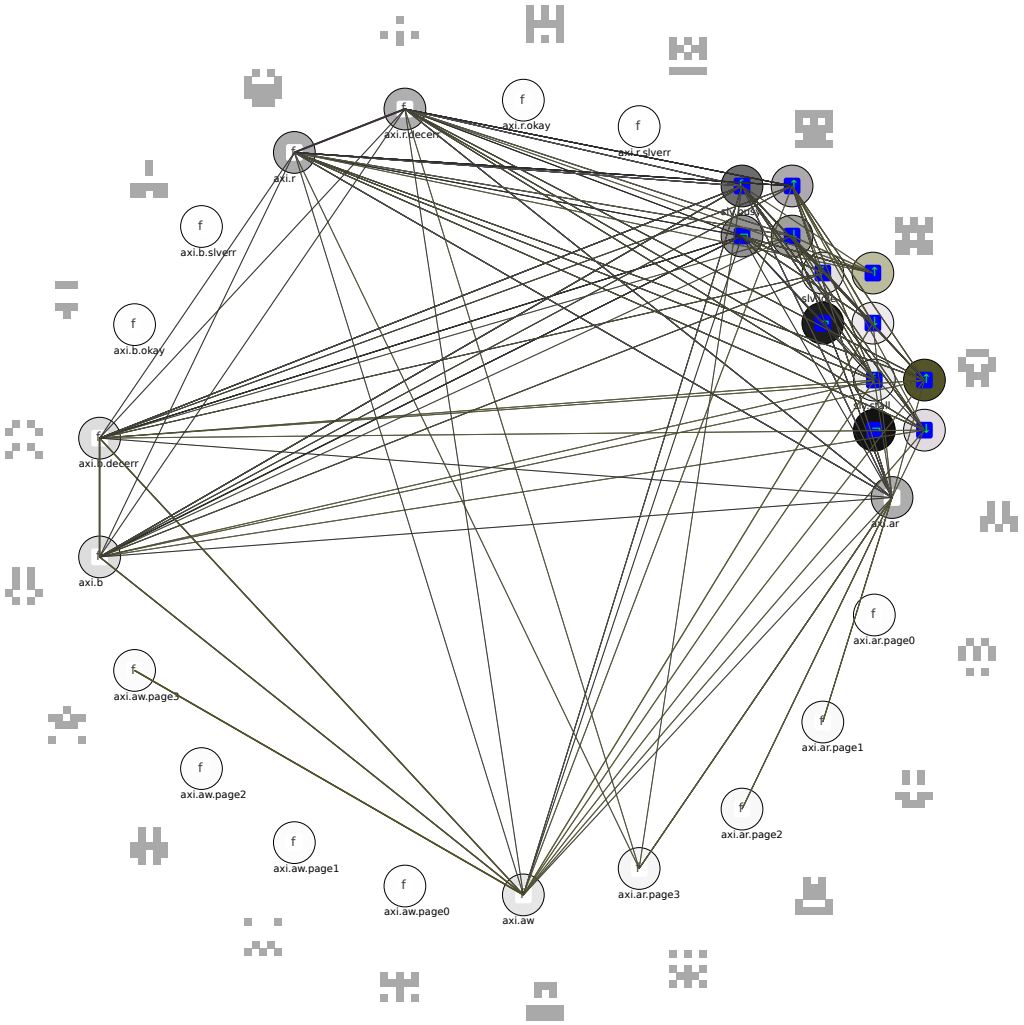


Figure 5.3: Netgraph visualisation with edges in 2D colourspace $(\dot{\mathrm{C}}\mathrm{ov}, \dot{\mathrm{D}}\mathrm{ep})$. Each node is coloured according to its statistics, and each signal is accompanied by an identicon to assist recognition and orientation. Correlations are represented by the weight of edges

between distinct sub-systems, and the least interesting results are between signals of the same sub-system because these relationships may be already understood via module-level verification. Alphabetical positioning places regularly named signals from the same sub-system beside each other so connecting edges will be short, thus contributing less visual impact than longer edges crossing the centre (N2, S5, GP6, FJ5).

Gansner and Koren [73] argue that circular graphs suffer from three main disadvantages: (1) longer edges, (2) visual regularity, and (3) following paths is difficult. In terms of visual impact, a longer edge crossing near the middle of the

figure is more prominent than a shorter edge chording near the periphery. Ganser and Koren's usage of circular graphs is primarily aimed at the layout of computer networks and system management where connections are binary. In contrast to Gansner and Koren's argument longer edges are seen as an advantage because, in combination with alphabetical positioning, they highlight connections between non-adjacent signals. The disadvantages of visual regularity distracting from less prominent edges and increased difficulty of following paths must be weighed against the advantages provided by a circular layout, i.e. circular layout is the simplest and easiest for a user to rationalise, in comparison to alternative layouts.

Support for the display of multiple δ offsets is limited in the netgraph view. Between a pair of signals, only the $\delta \in (-\Delta, 0]$ offset with the highest correlation is shown in order to avoid overwhelming the figure with too many lines (N8, GP3, FJ5, FJ9, FJ10). Section 5.2.3 describes alternative views which are suitable for investigating correlation over δ (cross-correlation).

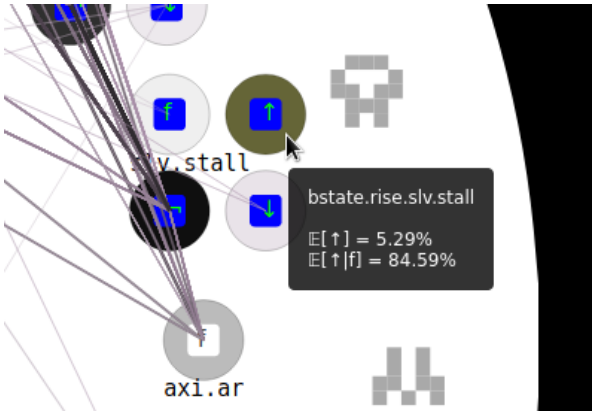
5.2.2.4 Interactive Features

Colours provide a fast visual approximation of various pertinent statistics, but it is difficult to realise exact values, so interactive features are available to uncover precise values. Figure 5.4 shows how the mouse pointer is used to examine the exact values of a node's statistics and metrics for a specific connection. Pointing with the mouse to specify a feature of particular interest, allows exact values to be obtained on request without adding unnecessary clutter to the overview (N8, N10, S4, S7, GP4, GP7, FJ4, FJ8).

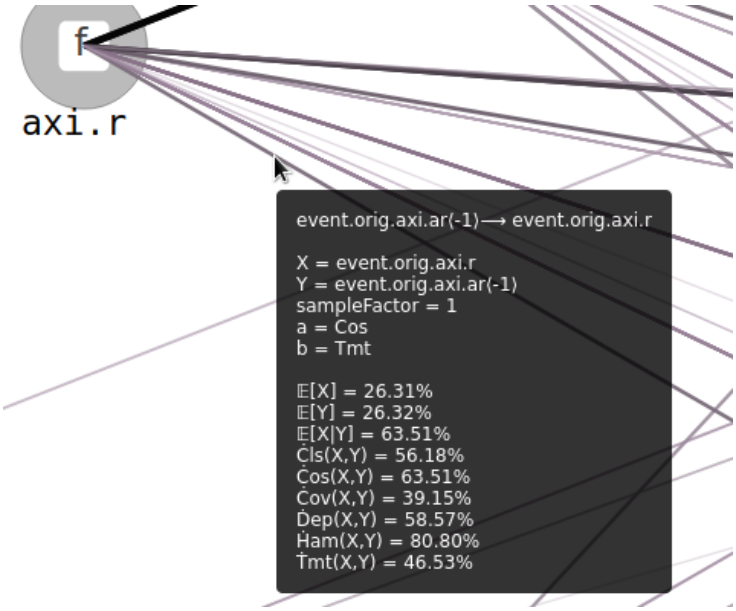
The netgraph view presents a full set of correlation results for a single time window, and relative navigation through time is enabled via clickable hyperlinks, i.e. to the previous or next window. Further detail on relationships with a specific signal or a specific δ offset is enabled with the tabdelta view.

5.2.3 Tabdelta View

Netgraph views present correlations between all signals for an overview of system behaviour during a specific time window, but support for different δ offsets is limited. The tabdelta partner view enables effective exploratory analysis by viewing differences in correlation over δ. There are three variations of the table-based visualisation, $\delta \times u$, $\delta \times x$, and $\delta \times y$ which present correlation alongside related statistics. In contrast, corrgrams (described in Section 2.3.2) present the variation $x \times y$ with fixed δ and u; i.e. constant δ and a single time window. Tables are a tried



(a) Hovering a pointer over a node displays a dialogue box containing precise values of relevant statistics. The strong green colour presents the low value of $\mathbb{E}[\uparrow_i]$ and high value of $\mathbb{E}[\uparrow_i | f_i]$ where i represents the signal name `slv.stall`.



(b) Hovering a pointer over an edge displays a dialogue box containing precise values of several statistics and correlation results. The purple tinge to the edge colour represents the correlation results $(\dot{\text{C}}\text{os} \approx 0.6351, \dot{\text{T}}\text{mt} \approx 0.4653)$. Redundant information, including the signal names and types, is included to help the user maintain orientation.

Figure 5.4: Hovering a pointer provides interactive details on demand.

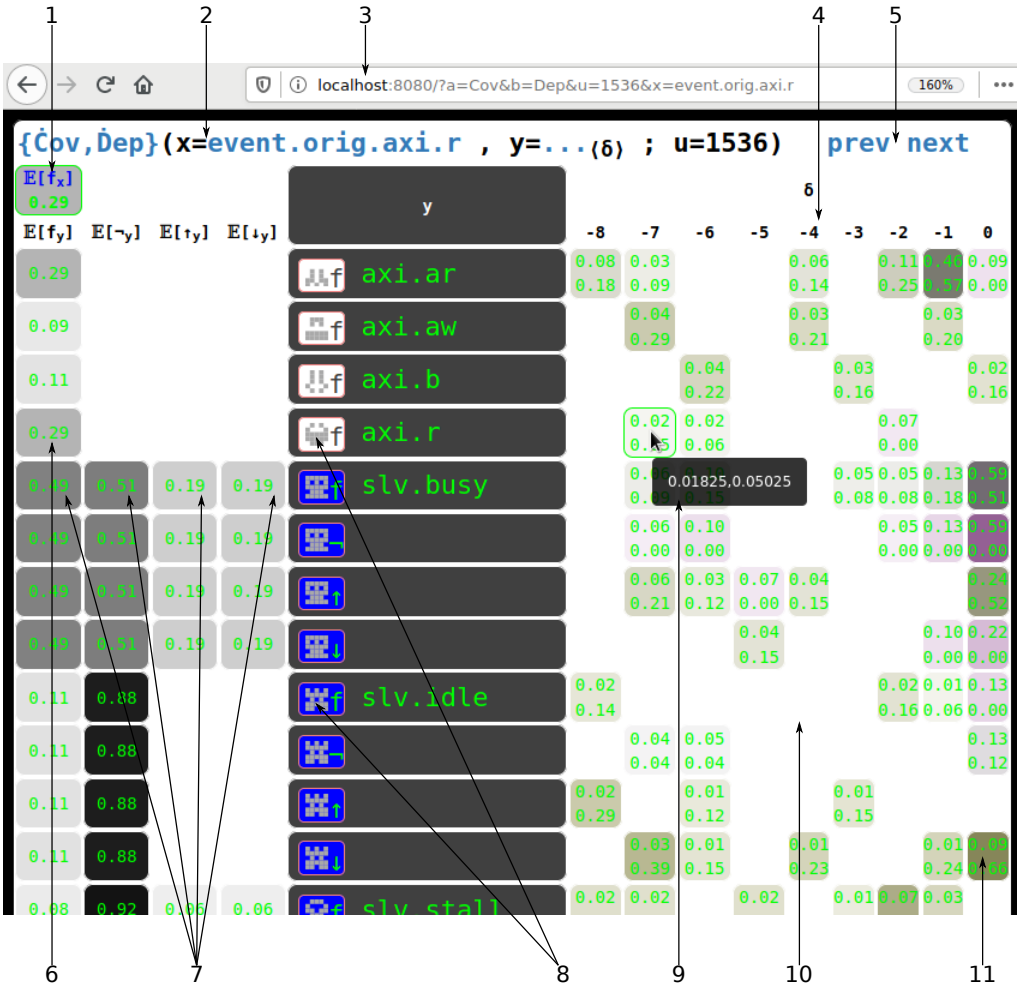


Figure 5.5: Features of tabdelta visualisation.

1. $\mathbb{E}[f_x]$ as colour and text.
2. Title of data view.
3. Browser-based navigation URL.
4. δ varies horizontally.
5. Links for navigation through time.
6. Where y is event, only show $\mathbb{E}[f_y]$.
7. Where y is bstate, show additional statistics.
8. Identicons on blue/white squares indicate type of y.
9. Precise value displayed by hovering a pointer.
10. Zero-valued results not displayed to avoid distraction.
11. Result cells use 1D or 2D colourspace background. Light-coloured text on light-coloured background reduces distraction from low values.

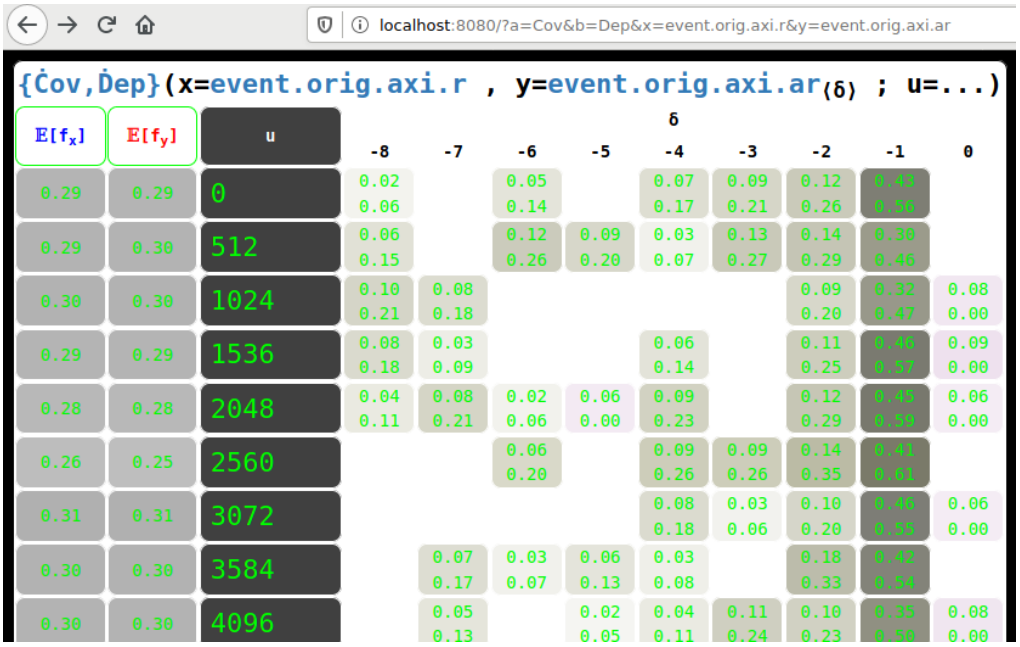


Figure 5.6: Tabdelta view from demonstration system "praxi", described in Section 5.4.1. Central dark grey column shows the first time index of each window. Correlation results are shown for different values of δ in each main column. Metric names are prominently displayed in the upper-left corner, and the URL is shown at the top.

and true method of presenting a 2D cross product with text in a grid – used since at least 120BCE by Hipparchus, and therefore do not require much evaluation.

The same 2D colourspace is used in the background colour of all data cells to provide a fast and intuitive method of visually identifying higher, more significant values. Identicons are clearly associated with each signal name, which simplifies the translation from a signal's position on a netgraph to a tabdelta row.

URLs assign a unique locator to each view, thus enabling direct and absolute navigation. Hyperlinks enable relative navigation to a view which is different by only one variable, i.e. changing view by swapping or altering one of x, y, or u.

Figure 5.5 shows the main features of a tabdelta view using an example of the $\delta \times y$ variation, and Figure 5.6 shows an example of the $\delta \times u$ variation. In the centre, the dark grey column shows the value of the vertical-axis variable. Identicons and blue/white squares identify the signal types and aid navigation between different views. Left of the central column, expected values of the row's sibling signals are presented via both text and colour. The main result area is right of the central column, where δ is varied on the horizontal axis. Each table cell shows a correlation result as text with the background colour providing a fast

visual estimation of the value. To reduce visual distractions, a low-contrast font colour is used, and very low-valued results are simply not displayed.

5.3 Gestalt Evaluation Against Heuristics

Hvannberg et al. note that effective use of heuristic sets requires considering a whole set at a time, otherwise meaning and intention are lost [9]. Gerhardt-Powals' reasoning for the inclusion of both GP9 and GP10, despite sounding similar, further demonstrates this requirement. Without GP10, there may be a conflict between GP6 and GP8, so the set must be given holistic consideration. While individual elements of a visualisation or an interface can be evaluated separately, the real test of an interface applies to the whole experience. A gestalt evaluation ensures that the techniques described above enable greater usability as a whole than the sum of its parts.

5.3.1 Nielsen's Usability Heuristics

Nielsen's usability heuristics are designed to uncover problems in existing designs and are used here to identify potential usability issues in the combined techniques of the netgraph and tabdelta views. These heuristics were originally designed for evaluation of systems that take data from the user, e.g. a form on a website, so some heuristics must be re-interpreted for use with presentation-only interactive visualisations. Overall, Nielsen's heuristics do not highlight any serious usability issues.

N1 - Visibility of system status. The web-browser interface is a familiar environment for SoC engineers, and the built-in features support this heuristic. For example, the native page-loading animation tells the user that a view request is currently being processed, and the native page-not-found screen informs the user that the tool cannot meet a request. This heuristic is mostly aimed at interfaces used for control, but all applicable interpretations are adhered to.

N2 - Match between system and the real world. This heuristic is interpreted for visualisation in the same way as Amar and Stasko's representational primacy. It is natural to speak of correlated things being connected, and netgraph draws visual connections to represent correlations. Stronger connections are drawn with heavier darker lines, which is a good match with our intuition. The use of bstate sibling signals allows the user to relate their thought process directly to different

parts of the netgraph view whether they are thinking in terms of f_i, $\neg_i$, $\uparrow_i$, or $\downarrow_i$. This heuristic is well-supported by closely matching the display to the user's intuition.

N3 - User control and freedom. Again, N3 is aimed at control interfaces, but the interpretation used here is that the user should be able to navigate the result space smoothly. Hyperlinks enable relative navigation around the result space by changing one variable, and absolute navigation is achieved by entering a URL directly. With the support for familiar navigation methods, both relative and absolute, N3 is adhered to.

N4 - Consistency and standards. In a novel visualisation, the use of standards, i.e. what similar visualisations use, is not applicable. N4 is primarily supported by the consistent use of alphabetical ordering, regular layout, identicons, and the same colourspace in both netgraph and tabdelta views.

N5 - Error prevention. In the netgraph and tabdelta views, the only way to introduce an error to the interface is to request a view which doesn't exist or isn't supported, e.g. netgraph view for a window beginning at a negative time. Absolute navigation by directly entering a URL does not adhere to N5 because nothing prevents the user from requesting an unsupported view. However, in relative navigation, only valid hyperlinks are displayed which does support N5. No checks for potential errors are presented, e.g. "Are you sure?" because this would violate N3 (user freedom) and interfere with N7 (efficiency of use).

N6 - Recognition rather than recall. Using interactive zooming, the set of displayed results can be easily altered to avoid having to recall specific ones. N6 can also be interpreted as the ease of finding or maintaining orientation within the result space, which is supported through the consistent use of layout, identicons, and colourspace.

N7 - Flexibility and efficiency of use. A sufficiently powerful machine must be used to run the tool in order for the user to avoid waiting too long between making a request and receiving a view. To argue that this heuristic is supported it is noted that the demonstration tool is implemented in Python (known for low-speed performance) and primarily tested on a laptop with an i5 CPU. The low-specification machine computes a new netgraph view for the demonstration

systems of Section 5.4 in less than 2 s, which does not present a significant impediment. Additionally, as the tool is based on a web-server, it is straightforward to calculate a view on a powerful machine and display results on a personal machine. As long as sufficient computing power is available, N7 is adhered to.

N8 - Aesthetic and minimalist design. Several aspects of both the netgraph and tabdelta views work towards supporting N8. Statistics on individual signals and their siblings are presented with coloured circles, but without cluttering text to compete for attention. Correlations are presented with coloured lines, and weaker correlations are drawn in thinner, lighter lines, which allows attention to focus naturally on stronger correlations. In the tabdelta view, light-coloured text is specifically used in the cells so that weaker correlations with light-coloured backgrounds attract less attention. However, information is not lost because the interactive pop-up feature provides these details on demand, as shown in Figure 5.5. In the netgraph view, the correlation with only a single δ offset is displayed to avoid clutter, but a more detailed display of correlations over different δ values is available in the tabdelta view. In the combination of these techniques, N8 is partially supported, but may benefit from a survey of user opinions.

N9 - Help users recognise, diagnose, and recover from errors. In partial support of N5, error conditions are avoided except for absolute navigation by directly entering a URL, which also partially supports N9. In the proof-of-concept implementation, judicious error messages are not included, so N9 is not well-supported. However, this heuristic is implementation-specific, so work to support N9 consists of adding more helpful error messages rather than changing the proposed techniques. Further, this heuristic is argued to be unimportant for information visualisation because it is aimed specifically at interactive control interfaces dealing with user input.

N10 - Help and documentation. In information visualisation, the presence of help and documentation is orthogonal to the method of presentation. As the proof-of-concept tool only includes basic usage instructions it could be argued that N10 is not fully supported. However, the strong representational primacy of the netgraph view and strong analytical primacy of the tabdelta view can be used to argue that N10 is adhered to because additional help is unnecessary. Additionally, as a browser-based tool, the internet is also close to hand for extra support.

5.3.2 Shneiderman's Visual-Information-Seeking Tasks

Shneiderman's seven heuristics describe high-level tasks that users will expect to be easily and efficiently accomplished. Using the tool combining the netgraph and tabdelta techniques, all seven tasks can be accomplished efficiently and intuitively in the pursuit of exploratory correlation analysis of binary SoC data.

S1 - Overview. The most easily specified view is the netgraph view, requiring only the window's first time u, which means this should be the starting point for exploratory analysis; i.e. "overview first". In a zoomed out overview, the colours of nodes and edges remain visible, thus the netgraph view adheres to S1 because all pairwise correlations can be shown in one figure.

S2 - Zoom. Zooming is chiefly applicable to the netgraph view where the native browser controls adhere to S2. A close-up view reveals the exact pattern of closely-packed edges, and further details can be examined by summoning a dialogue box with the mouse pointer. The tabdelta view provides less scope for zooming, but using browser controls to fit more table cells onto the screen is a reasonable interpretation which also supports S2.

S3 - Filter. This heuristic is interpreted to mean that a clutter-free minimalist aesthetic is used. Figure 5.5 shows a filtering feature of the tabdelta view where zero-valued results are not displayed in order to avoid distracting clutter. It can be argued that including text in tabdelta cells constitutes distracting clutter which would go against S3, but this argument must be balanced by S2 and the ability to zoom in on exact values. The netgraph view reduces the width of lines showing weak correlations which are less important than strong correlations, thereby making it low-effort to visually filter less important results.

S4 - Details-on-demand. Details can be demanded from either the netgraph or tabdelta view by hovering a pointer, thus adhering to S4 on a per-view level. The ability to navigate from a netgraph overview to a detailed tabdelta view supports S4 on a per-use level.

S5 - Relate. By rendering correlations as connecting lines, strong representational primacy is achieved in the netgraph view which supports S5. Additionally, sibling results are shown concurrently, which allows the user to easily relate thought processes to different terms (value, reflected value, or rising or falling edge), also supporting S5.

S6 - History. Each view is specified by a URL and web-browsers typically keep an ordered history of URLs visited. S6 is thereby well-supported through use of a browser-based interface.

S7 - Extract. Extraction of query parameters is well-supported by looking at the URL. Support for extraction of sub-collections is limited to that provided by the features supporting S3 and S4. Further improvement may be possible to interactively extract correlation sub-graphs. Thus, S7 is supported but enhancements may be possible.

5.3.3 Gerhardt-Powals' Cognitive Engineering Principles

Gerhardt-Powals' heuristics are phrased as guidelines for the construction of user interfaces. All ten heuristics have been adhered to in the development of the tool combining netgraph and tabdelta views.

GP1 - Automate unwanted workload. Adherence to GP1 is the fundamental purpose of the novel visualisation techniques specialised towards binary SoC data. Through effective presentation of data, laborious mental calculations, estimations, and comparisons are avoided, thus freeing up cognitive resources for higher-level tasks. The netgraph view supports GP1 particularly well because the overview of correlations assists higher level thinking more than the detail-oriented tabdelta view.

GP2 - Reduce uncertainty. Availability of precise details, provided on demand, removes uncertainty about exact values. Due to the statistical nature of correlations analysis, there is always some uncertainty that parameters such as window length are ideal for identifying a particular behaviour or change in behaviour. However, the operator should be made aware of the limitations intrinsic to statistical analysis. The 2D colourspace is also designed to reduce uncertainty in operators with protanopia (also known as red/green colour-blindness) as well as those with normal colour vision. GP2 is supported through identification and addressing these three types of uncertainty.

GP3 - Fuse data. In information visualisation, as opposed to generic user interfaces, GP3 is interpreted in the same way as GP1 because the purpose of data fusion is to reduce cognitive load.

GP4 - Present new information with meaningful aids to interpretation. The use of identicons to aid mental orientation between views can be argued to violate GP4 because the glyphs do not carry meaning about the signal's definition or purpose. However, by keeping the identicons slightly separated from the presented results, they can be easily ignored or used only on a sub-conscious level, i.e. no substantial negative impact on GP4. In partnership with GP6, this heuristic alludes to the notion of representational primacy. As such, it is interpreted and supported in the same ways as N2.

GP5 - Use names that are conceptually related to function. The names "netgraph" and "tabdelta" are convenient names for the two views which describe their structure, and "tabdelta" also describes its function: to tabulate correlations over δ. "Netgraph" does not describe its function, but does not violate GP5 because the graphical structure implies the presence of connections or relationships. With the tool's intended purpose being correlation analysis, the implied purpose behind the name "netgraph" is apparent, and GP5 is not violated. Other text displayed includes the context-dependent title of a tabdelta view (2nd item of Figure 5.5) and the signal names in the both views. These textual labels help the user maintain a precise mental orientation, further supporting GP5.

GP6 - Group data in consistently, meaningful ways. Consistent presentation is achieved through alphabetical ordering. While alphabetical ordering may not provide the tidiest circular graphs [73], it is a simple scheme requiring little cognitive effort to understand. Meaning is attached to the consistent layout of sibling nodes by redundantly presenting bstate statistics in terms of their value, reflected value, or edges, which matches naturally spoken terms. Additionally, through an interpretation close to representational primacy, GP6 is supported by presenting correlations as connections.

GP7 - Limit data driven tasks. In an interface for visualisation rather than control, GP7 is interpreted as equivalent to GP1 and is therefore well-supported.

GP8 - Include in the displays only that information needed by the operator at a given time. Section 5.2.1 describes the use of some common statistics such as median and variance before showing why these are not relevant to binary signals. By excluding statistics which might be expected (as they are common in other tools) confusion and misunderstandings are avoided. Both netgraph and

tabdelta include only the information directly relevant to the view requested, thus adhering to GP8.

GP9 - Provide multiple coding of data. In the same way as GP2 and S4, this heuristic is partially supported by the interactive feature of summoning details using a pointer. The tabdelta view further supports GP9 by presenting correlation values via both colour and text.

GP10 - Practise judicious redundancy. Redundancy is practised in the netgraph presentation of individual statistics using large coloured nodes. Larger nodes provide more redundancy than smaller nodes, and can be more clearly seen in a zoomed-out view. This principle was devised to resolve possible conflicts between GP6 and GP8, but as no conflicts have been found between these principles for correlation visualisation, GP10 is supported according to the original intention.

5.3.4 Forsell and Johansson's Visualisation Heuristics

Forsell and Johansson's ten most important visualisation heuristics are found via meta-analysis of several sets of heuristics, including Nielsen's and Shneiderman's. Therefore, evaluation against each of these heuristics can be deferred to one of the other three heuristic sets above. The purpose of inclusion here is that these are found in the meta-analysis to be the most relevant heuristics specifically for information visualisation interfaces. Checking that visualisation techniques do not cause conflicts in Forsell and Johansson's set thereby ensures consistency of heuristic support when the methodology is considered as a whole.

FJ1 - Information coding. This heuristic is equivalent to the meaning of representational primacy encompassed by N2, S5, GP4, and GP6, i.e. well-supported.

FJ2 - Minimal actions. Navigation using URLs or hyperlinks, zooming using browser controls, and summoning details using a pointer enables tasks to be performed intuitively and with minimal actions. Thus the cognitive load is minimised for all actions in exploratory analysis and FJ2 is adhered to.

FJ3 - Flexibility. This is distinct from the flexibility term in N7, which is supported through the use of a browser-based interface on a sufficiently powerful machine. Instead, FJ3 refers to customisation to enable different exploratory strategies and workflows, and is aimed at making improvements to an interface with user feedback. Given that this evaluation is of a novel visualisation where

common habits and strategies have not yet been established, it is not possible to critically evaluate against this heuristic. However, the combination of the netgraph view, three variations of the tabdelta view, and relative navigation between them using hyperlinks goes some way to anticipating possible exploratory strategies.

FJ4 - Orientation and help. The meaning of this heuristic is very broad, encompassing documentation, a history of views, and visual orientation. While extensive documentation is not included in the proof-of-concept tool, this is straightforward to add and is orthogonal to the techniques the tool is based on. A history of views is maintained through the browser interface. Mental orientation is assisted through consistent layout (also supporting FJ6) and the use of identicons.

FJ5 - Spatial organisation. This heuristic is also very broad, but is adhered to as explained in evaluation against N2, N8, S1, S5, and GP6. As a rephrasing of N8 and GP8, this heuristic is well-supported.

FJ6 - Consistency. Similarly to N4 and GP6, this heuristic is supported through the use of alphabetical positioning and the same colourspace to represent values.

FJ7 - Recognition rather than recall. This is a repetition of N6 in both phrase and meaning, supported through the consistent use of layout, identicons, and colourspace.

FJ8 - Prompting. Available actions in any netgraph or tabdelta view include zooming, navigation to a different view, and hovering a pointer to demand precise details. Zooming does not require prompting because it uses the native controls of the web-browser. Hyperlinks for navigation are coloured differently from non-clickable text (seen at the top of Figure 5.5) which is native functionality for browser-based interfaces. Hovering a pointer does not require prompting because there are so few views to learn. This heuristic is aimed at control or data-input interfaces that have many types of views. However, in the presented method of only two views, hovering a pointer does not require prompting because learning all aspects of a view requires minimal effort.

FJ9 - Remove the extraneous. As a rephrasing of N8 and GP8, this heuristic is well-supported.

FJ10 - Dataset reduction. This heuristic is concerned with features to reduce a data-set for the purpose of focused analysis. As in the evaluation against S7, this heuristic is not fully supported because extracting correlation sub-graphs would require re-starting the tool with a smaller section of signals chosen for analysis. As such, this heuristic offers a potential area for improvement in future work.

5.3.5 Direct Comparison with State of the Art

Table 5.1 compares the existing techniques, as described in Chapter 2, with the proposed techniques for visualising correlations in binary SoC data. The SoC logic designer's most familiar visualisation, waveforms, is reviewed first. Waveforms, or other multi-plot tools, are very useful for visualising signal values over time, but are ill-suited to presenting relationships. Corrgrams are well-suited to presenting one, or perhaps two, correlation metrics for a single time window but not for finding the context around those correlations. Corrgrams are compared for different ordering methods; (1) sorted by signal identifier, e.g. alphabetical; (2) sorted by correlation value, e.g. SVD, as described by Friendly and Kwan [67, 68]. Sorting by value allows a corrgram to provide a zoomable overview because elements can be sensibly blurred together and still present an accurate depiction. Zapf and Kraushaar's Solar Correlation Map [69] excels at providing an intuitive overview of correlations, but only centred around a selected signal. However, the limitations around other features explain why their technique is not widely used. Printable tables are also compared because they are the basis of tabdelta, so the improvements made by the proposed techniques can be seen clearly.

A direct gestalt comparison shows the netgraph and tabdelta techniques to be overall preferable in comparison to existing tools and techniques, despite specific features being less desirable, e.g. density of corrgram versus netgraph.

Features supported	Waveform or other multi-plot	Corrgram sorted by identifier	Corrgram sorted by value	Solar map	Printable table	Netgraph only (this work)	Tabdelta only (this work)	Netgraph +Tabdelta (this work)
Zoomable timeline	✓	✗	✗	✗	✗	✗	½	½
Zoomable overview	✗	✗	½	½	✗	✓	½	✓
Details on demand	✗	✗	✗	✗	✗	✓	✓	✓
Result-space navigation	✗	✗	✗	✗	✗	✓	✓	✓
Many signals	✓	✓	✓	✗	✓	✓	✓	✓
Multiple metrics	✗	✓	✓	✗	½	✓	✓	✓
Contextual statistics	½	✗	✗	✗	✓	✓	✓	✓
Unordered values	✓	✓	✗	✓	✓	✓	✓	✓
δ offsets	½	✗	✗	✗	½	½	✓	✓
Familiar usage	✓	✗	✗	✗	✓	✓	✓	✓
Visible connections	✗	½	✓	½	✗	✓	✗	✓
Dense format	✓	✓	✓	✗	✓	✗	✓	✓

Table 5.1: Direct comparison of usability features between existing and proposed techniques for visualising correlations in SoC data. Cells containing ✓ indicate full support with intuitive and practical usage; ½ indicates partial support; ✗ indicates unsupported or impractical usage. The columns for netgraph and tabdelta demonstrate the greater usefulness of the proposed techniques in comparison to existing techniques.

5.4 Demonstration Case Studies

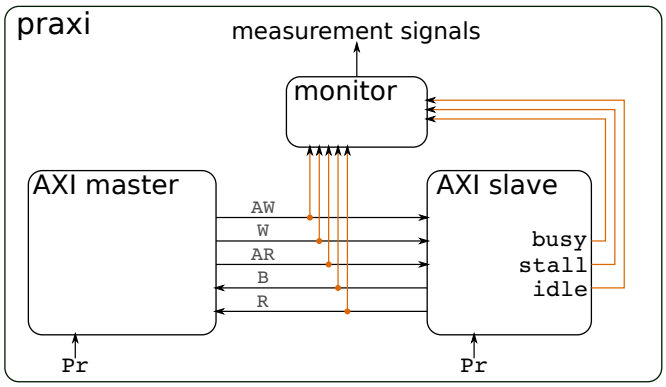
Two case studies are used to demonstrate the utility of the netgraph and tabdelta views. First, a simple system with unchanging behaviour is used to showcase how the netgraph view can uncover sub-systems operating in parallel. An example is included to show how the tabdelta view can expose behavioural features and match them to the relevant source code. Second, a complex system which performs a machine learning task is analysed to demonstrate how changes in behaviour appear in the netgraph overview. Finding the points in time where behaviour changes is also shown to be investigable with the tabdelta $\delta \times u$ view. In the examples of changed behaviour, the concept of a recognisable behaviour is made clear, which paves the way for future work in behaviour monitoring.

5.4.1 Case Study 1: Static Behaviours

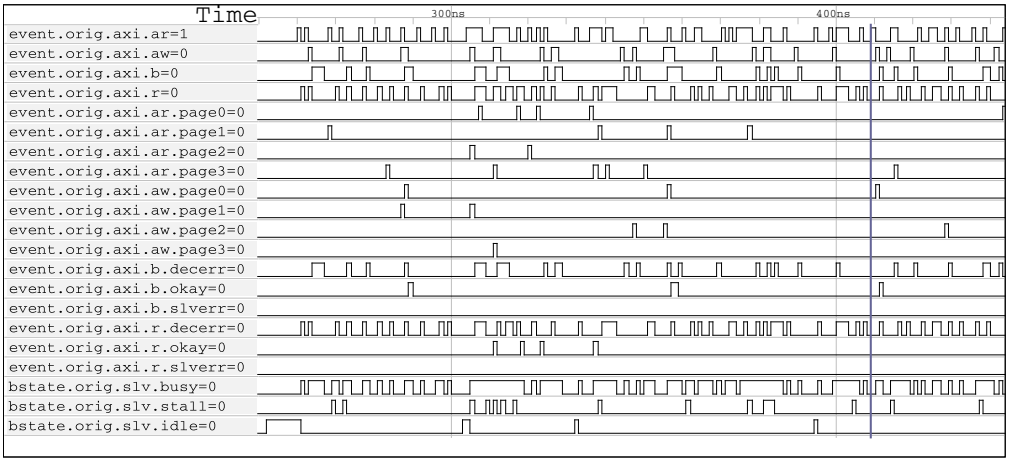
5.4.1.1 Description of praxi System

Visual analysis of static behaviour is demonstrated using a simple SoC named "praxi" is depicted in Figure 5.7, created from components usually used to verify the function of an AXI network. A master communicates pseudo-randomly with a slave, and signals are recorded via a passive monitor component. The AXI link has separate channels for write data (`W`), write address (`AW`), write reply (`B`), read address (`AR`), and read reply (`R`) that have independent handshaking mechanisms. Probability controls are used to determine aspects of the master and slave behaviour, including the likelihood of a request being generated by the master, and how likely the slave is to stall a response. These controls are held constant during the entire simulation, thereby keeping system behaviour static. Queues are implemented on each side of the AXI link so that multiple transactions may be outstanding.

Event signals are named like "`axi.ar`" which indicate that a transaction occurred on the `AR` channel, and similarly for the other four channels. On the AXI request address channels (`AR` and `AW`), four sub-events show where the request was addressed to; e.g. `axi.ar.page3` indicates a read request to page3. On the AXI reply channels (`R` and `B`), three sub-events show the reply status (`okay`, `slverr`, or `decerr`); e.g. `axi.b.decerr` indicates a decode error on the write reply channel. For additional familiar context, three bstate signals are provided on the slave module (`busy`, `stall`, and `idle`). The `busy` signal indicates that the slave has accepted requests which have not been replied to, `stall` indicates that the slave is currently stalling a reply, and `idle` is defined by the code snippet in Figure 5.8; i.e. `idle` is only asserted when no reply transactions have occurred in the previous four cycles. In total, the



(a) Logical topology of SoC used for demonstration system "praxi". The passive monitor uses AXI `*VALID` and `*READY` signals to determine when transactions occur on each channel [33]. Additional passive logic decodes the AXI sub-events, and no logic is required to decode the bstate signals (`busy`, `stall`, and `idle`).



(b) Sample of the praxi dataset, approximately 150 ns in length at clock frequency of 1 GHz. With probabilistic controls held constant, all signals transition at uniform rates, i.e. system behaviour is unchanging.

Figure 5.7: Demonstration system "praxi". All components are synthesisable allowing the system to run in either simulation or FPGA.

```
1  // ...snip...
2  parameter TIMEOUT = 3,
3
4  always @(posedge i_clk)
5    if      (bfifo_o_valid || rfifo_o_valid) idlecnt_q = TIMEOUT;
6    else if (idlecnt_q != 0)                 idlecnt_q = idlecnt_q-1;
7    else                                     idlecnt_q = idlecnt_q;
8
9  assign o_idle = !o_busy && (idlecnt_q == 0);
10 // ...snip...
```

Figure 5.8: System Verilog specifying behaviour of `slv.idle`. The relationship between $f_{\texttt{slv.idle}}$ and $f_{\texttt{slv.busy}\langle -3\rangle}$ is readily apparent from lines 2, 5, and 9.

praxi dataset is comprised of 21 binary signals which are collected over several thousand clock cycles. Figure 5.7a depicts the simple topology of the praxi system, and Figure 5.7b shows a sample of captured data presented with the GTKWave waveform analyser.

This static-behaviour system is used to demonstrate three visualisation features: (1) the netgraph view provides an overview of behaviour over one time window; (2) conceptual sub-systems can be identified with the netgraph view; (3) the tabdelta view can be rationalised to match source code.

5.4.1.2 Behaviour Overview in Netgraph View

The single AXI link in praxi can be thought of as three sub-systems: (1) write requests elicit write replies, (2) read requests elicit read replies, and (3) status signals indicate the state of the slave block. While the read and write sub-systems do not interact directly with each other, both interact with the status sub-system. Figure 5.9 augments Figure 5.3 to highlight these three sub-systems. Strong correlations can be seen within each sub-system, as well as interactions with the status signals. Expected results such as read replies being strongly correlated with read requests, and replies strongly correlated with the `slv.busy` status are readily apparent. It can also be seen that the read channel is not interacting with the write channel; i.e. expected behaviour is confirmed.

Figure 5.9 is therefore seen to demonstrate that the netgraph view confirms all the main behavioural aspects of this simple example system. By contrast, Figure 5.7b highlights the difficulty of identifying behavioural connections from a busy-looking waveform diagram. For a single time window, the system behaviour is presented as a graph that enables the viewer to quickly confirm hypotheses about correlations between signals. The pattern of correlations is clearly visible even at a zoomed-out perspective, thus an overview is effectively presented. Highly connected node clusters indicate sub-systems, but recognising these requires care-

ful inspection because circular layout is not optimal for identifying clusters [73]. However, visually confirming node clusters is straightforward, and the advantages of consistent circular layout (mental orientation, straight edges) outweigh the disadvantage of less prominent clusters.

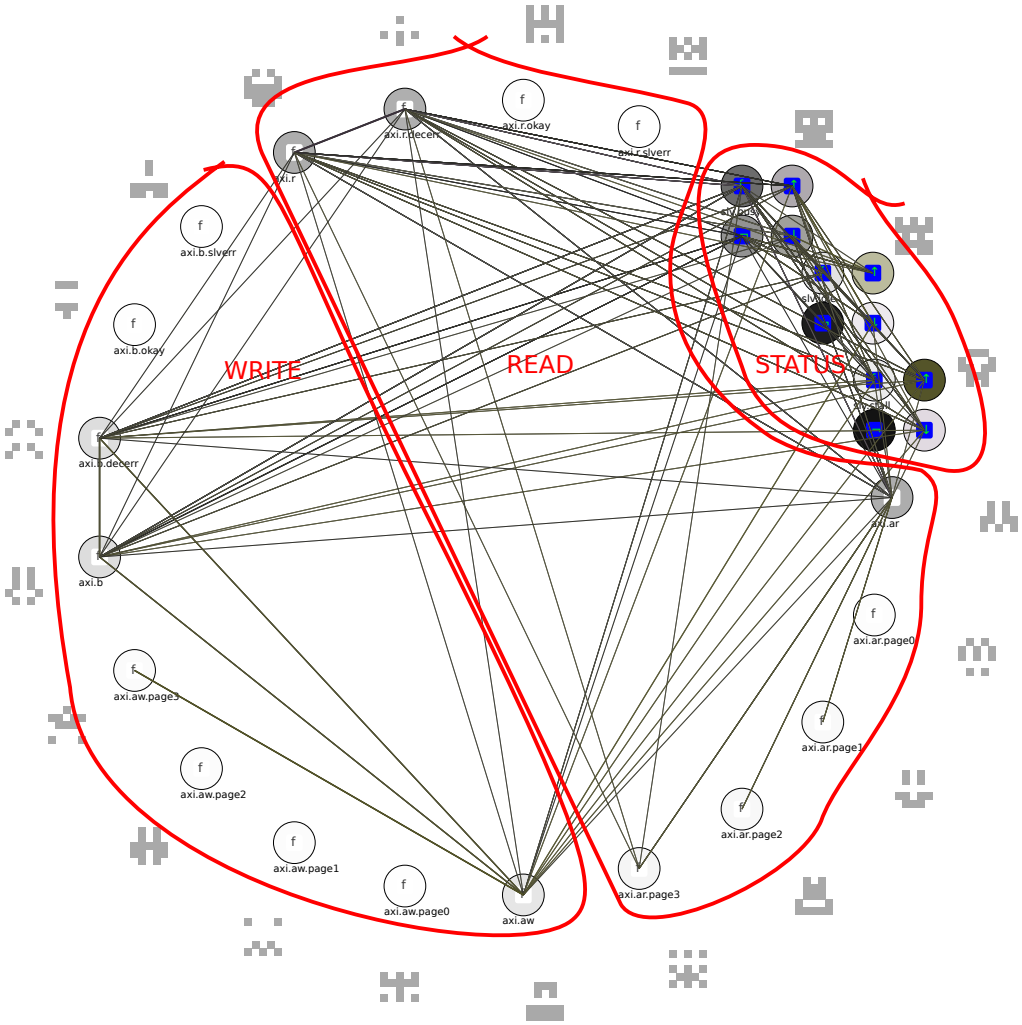
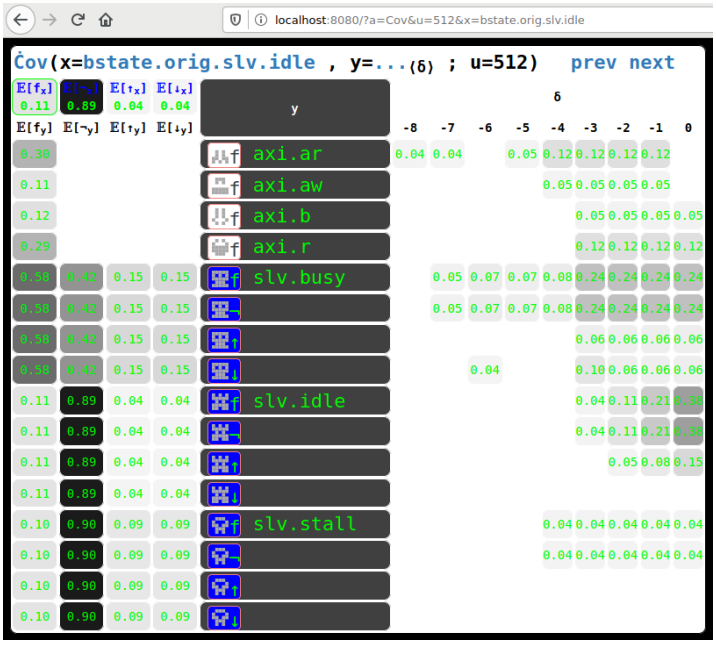


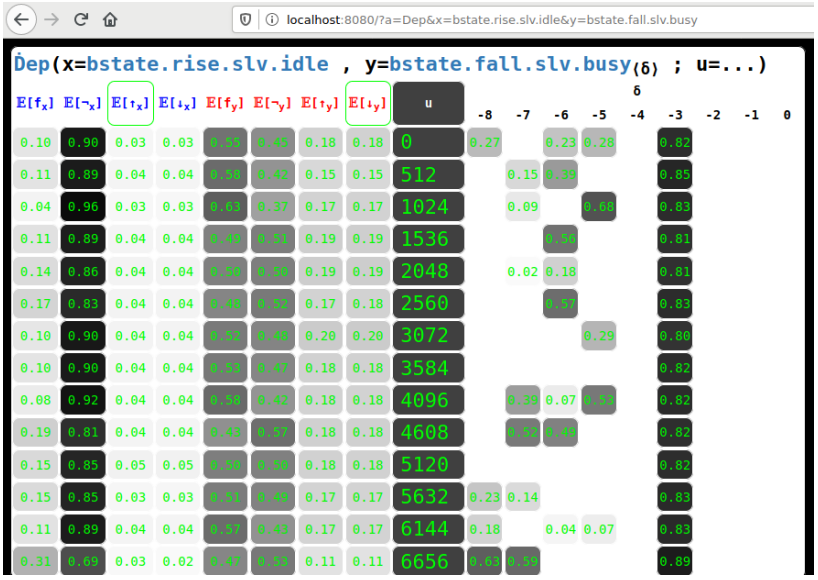
Figure 5.9: Identifiable behaviour sub-systems in praxi.

5.4.1.3 Behaviour Overview in Tabdelta View

Figure 5.10 demonstrates, in the scenario of investigating interactions with `slv.idle`, how specific source code features manifest in the tabdelta view. Correlations with all other signals are shown in Figure 5.10a over $\delta \in [-8, 0]$. The darkest result cells are auto-correlations that are explained in Section 5.2.1 to offer no new information over $\mathbb{E}[f_i]$. The next most significant visual features include the correlation of `slv.idle` with `slv.busy`, particularly for $\delta \in [-3, 0]$. Figure 5.10b



(a) Correlations in the praxi system between `slv.idle` and other signals.



(b) Correlations between `slv.idle` and `slv.busy` over time.

Figure 5.10: Specific behaviour of praxi `idle` signal visible in tablar view.

then shows a closer look at how this pairwise relationship evolves over time. As expected for a static-behaviour system, results are consistent over time (u on the vertical axis). A strong result for Ḋep with a δ-offset of 3 cycles matches with the behaviour expected from Figure 5.8. This example therefore demonstrates how a tabdelta view can be rationalised against source code.

5.4.2 Case Study 2: Changing Behaviours

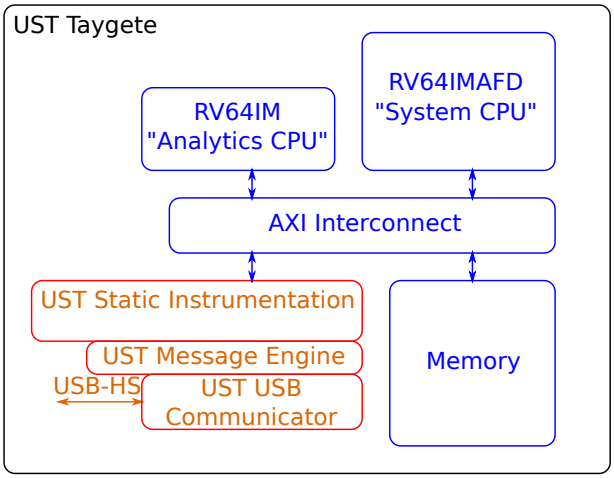
5.4.2.1 Description of tinn System

In this second case study, a more complex system is used to demonstrate the visualisation techniques. Where the praxi dataset is based on simulation of a simple SoC, the "tinn" dataset is recorded from instrumented software running a machine-learning task on a FPGA-based system, depicted in Figure 5.11. The system's main function is to recognise handwritten numerical digits from the Semeion dataset [140, 141]. Tinn software is based on a FOSS project implementing a tiny neural network [139] that has been modified to run on UltraSoC's dual-CPU system Taygete [138]; i.e. to support RISC-V embedded environment, allocate tasks between two CPUs, and draw on an LCD screen. The smaller, less performant processor `A_CPU` collects batches and passes them to the `S_CPU` which has floating-point hardware support. Instrumenting Tinn's C code uses the `-finstrument-functions` flag of the GCC compiler to insert the `__cyg_profile_func_enter/exit()` standard profiling functions at the entry and exit points of other C functions, such that each function call produces an analytics message containing the CPU name, function entered (or exited), and timestamp. Operation is split into two phases: First, the training phase where a FFNN is trained using stochastic gradient descent. Second, the prediction phase where images of handwritten numerical digits are analysed by the FFNN to predict which digit they represent.

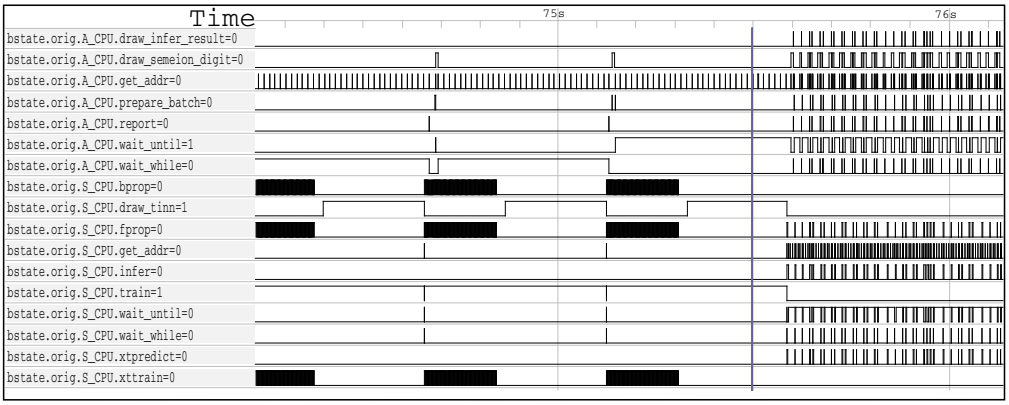
This known change in behaviour is used to demonstrate three points: (1) behaviour changes are apparent in a series of netgraph views; (2) the netgraph view provides recognisable images of specific behaviours; (3) the tabdelta view can be used to find approximate points in time where behaviour changes.

5.4.2.2 Behaviour Changes in Netgraph View

Figure 5.12 shows netgraph views for three time windows corresponding to the training and prediction phases, as well as a window which straddles the transition point between them. Prior to this point, no other views of tinn's correlations have been shown but it is clear that Figure 5.12 represents a different system

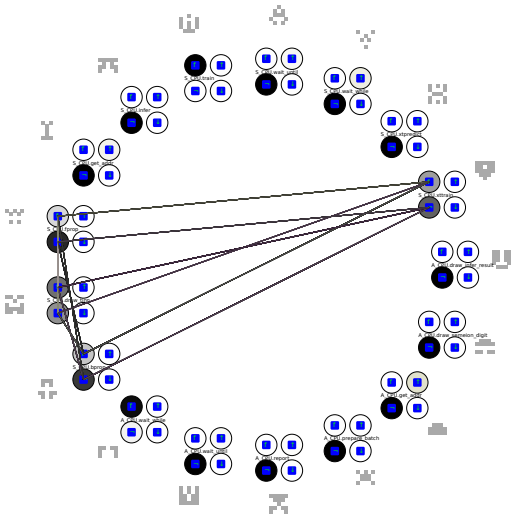


(a) UltraSoC's Taygete [138] SoC which runs the Tinn [139] software.

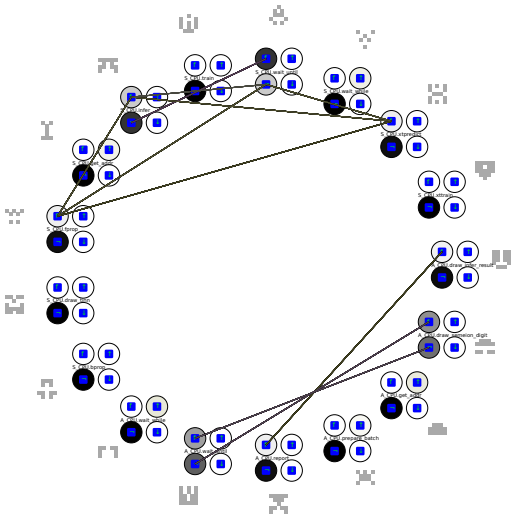


(b) Extract of tinn dataset of approximately 2 s in length, capturing the entry and exit times of 17 C functions across 2 CPUs. A step change in behaviour can be observed at around 75.6 s when operation moves from the training to the prediction phase.

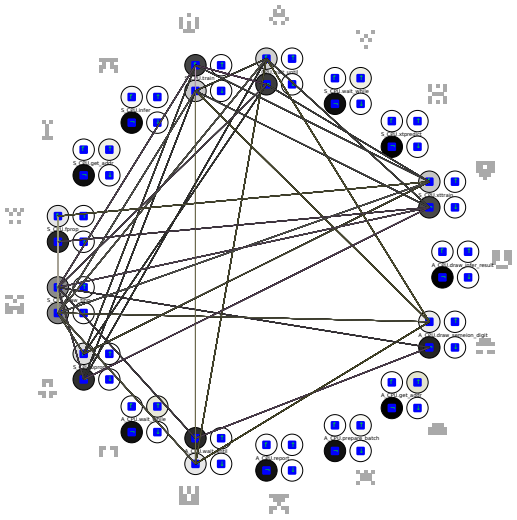
Figure 5.11: Demonstration system "tinn". A small software FFNN, running across 2 CPUs, is trained to recognise handwritten digits from the Semeion dataset [140, 141].



(a) Training phase, $u = 70\,\mathrm{s}$.



(b) Prediction phase, $u = 77\,\mathrm{s}$.



(c) Transition from training to prediction phase, $u = 74.9\,\mathrm{s}$.

Figure 5.12: Changing behaviour patterns visible in netgraph for $(\dot{\mathrm{C}}\mathrm{ov}, \dot{\mathrm{D}}\mathrm{ep})$ perspective of tinn. Correlation metrics are calculated over a window of $N = 2^{20}$ samples, corresponding to around 1 s. At this zoomed-out scale all text is too small to be readable, but the different patterns of edges between Figure 5.12a and Figure 5.12b clearly convey that system behaviour has changed between the training and prediction phases. Using a time window which straddles the transition between phases, as in Figure 5.12c, produces a mixed pattern of edges while system behaviour stabilises.

because the identicons are different. Another clear difference is that all signals are bstates, shown by the blue square in each node. Even without much context it is straightforward to see that these diagrams refer the same (previously unseen) system.

It is clear from the visualisations that system behaviour changes between the training phase (Figure 5.12a) and the prediction phase (Figure 5.12b) by the pattern of edges. A time window which straddles the transition in phase presents as a mix of many lighter edges (weaker correlations) than either training or prediction, e.g. Figure 5.12c. Other windows in the training and prediction phases present as similar patterns of edges as Figures 5.12a and 5.12b respectively.

This example demonstrates that the netgraph view presents a specific behaviour as a specific and recognisable pattern of edges. As behaviour changes, the pattern of edges changes recognisably, thus providing an effective overview of the system's behaviour through time.

5.4.2.3 Behaviour Changes in Tabdelta View

Finding specific points in time where the behaviour transitions is sometimes possible in a waveform analyser, e.g. Figure 5.11b shows a distinct change at approximately 75.6 s. However, it is unclear from the waveform how interactions between the signals change. Waveform diagrams may show that a change has occurred, but do not offer much help deciphering what the system was doing on either side of the change. Similar functionality is offered by the tabdelta view as exemplified in Figure 5.13. Two indicators of the behaviour change at around 75.6 s are visible: (1) in the $\mathbb{E}[f_y]$ column on the left, the cell colour changes from light grey to white at the row for $t \in [75.24\,\text{s}, 76.29\,\text{s})$; (2) on the same row, an increase in $\dot{\text{D}}\text{ep}$ and $\dot{\text{C}}\text{os}$ produces darker cells for low values of δ. After seeing these features, the user knows that netgraph views before, after, and during this time are worth further investigation.

The advantages of the tabdelta view over a waveform view are: (1) easy relative navigation to a netgraph view or other tabdelta views, and (2) correlations are presented directly, rather than the simple change in density of occurrences visible. However, a waveform view has the advantage of presenting all signals in one figure, so effectively searching for the most interesting points in time requires a blended approach of waveform and tabdelta views. Therefore, the presented visualisation techniques in this work should be used in conjunction with traditional waveform analysers, rather than replacing them.

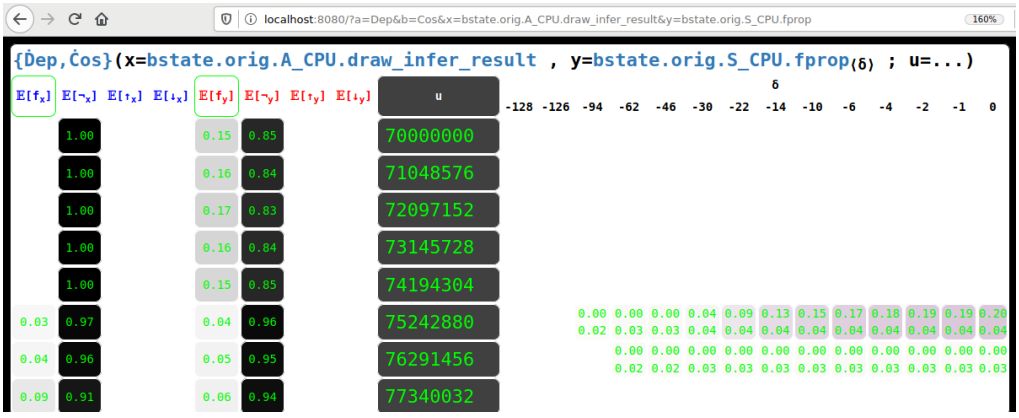


Figure 5.13: Tabdelta view of tinn over time. Change in behaviour can be seen around 75 s.

5.5 Conclusion

Better methods to visualise correlations in binary SoC data are desirable to assist engineers in their process of building up a mental picture of system behaviour. On the basis that an understanding of system behaviour can be achieved via knowledge of correlations between signals, visualisations are used to effectively present this information. A methodology comprised of two different views, netgraph and tabdelta, has been presented to visualise SoC correlations at different levels of abstraction. The netgraph view provides a visual overview of all pairwise correlations in a single time window, and the tabdelta view provides a detailed view of correlations across a range of δ offsets. Each aspect of these views has been evaluated against heuristic sets from Nielsen, Shneiderman, Gerhardt-Powals, and Forsell and Johansson to provide confidence that the proposed techniques improve upon state-of-the-art visualisations for correlations in binary SoC data. Further heuristic evaluation is applied to the methodology gestalt to ensure that the techniques in the netgraph at tabdelta views combine to support a fluid and effective workflow.

Existing tools and methods are shown in Section 2.3.2 to be insufficient for effective behaviour analysis in binary SoC data. The novelty of this work is in the development, heuristic evaluation, and demonstration of visualisations aimed specifically at SoC engineers looking to understand their system's behaviour. All example figures are created using screenshots from a browser-based tool which has been created to implement these techniques and function as a practical tool. The two systems used for demonstration represent realistic scenarios in modern SoC development; i.e. praxi demonstrates the methodology at a low level of abstraction (AXI channels), and tinn demonstrates the methodology at a software-based higher level abstraction. This combination establishes that the visualisation techniques described in this chapter are not restricted to only toy examples but are applicable to real-life systems with significant complexity.

CHAPTER 6

CONCLUSIONS

6.1 Summary of Thesis

The basis of this thesis is that engineers and system designers can better understand how and why their system exhibits interesting attributes through the use of correlation analysis. The use of correlation analysis for understanding behaviour in SoCs has been explored with a view to enhancing the development of complex systems in the increasingly competitive silicon marketplace. Chapter 1 outlines the purpose and scope of this research, then Chapter 2 provides the required context by reviewing state-of-the-art literature in the field.

Various interpretations of the vague term "correlation" are presented in Chapter 3 along with context surrounding their use in other scientific fields. Using a probabilistic model of binary SoC data, correlation metrics aimed at uncovering the existence of inter-signal logical relationships are characterised. Six correlation metrics are defined based on the concepts of set theory, geometry, and probability:

$\dot{\text{H}}\text{am}$	Hamming Distance, reflected to form the Hamming Similarity, and normalised to the unit interval.
$\dot{\text{T}}\text{mt}$	Tanimoto Coefficient (also known as the Jaccard Index).
$\dot{\text{C}}\text{ls}$	Euclidean Distance, reflected to form the Euclidean Closeness.
$\dot{\text{C}}\text{os}$	Cosine Similarity.
$\dot{\text{C}}\text{ov}$	Covariance. Rectified and normalised to the unit interval.
$\dot{\text{D}}\text{ep}$	Statistical dependence.

These six metrics are compared for usefulness using a Monte-Carlo approach based on a probabilistic model of binary SoC data. The model is composed of four probability distributions defining the number of signals in a system, how often those signals assert, how many other signals each is connected to, and the type of logic operations forming their connections. Evaluating the performance of these six metrics, using PDFs of several methods of scoring binary classifiers, reveals that the most useful metrics for uncovering connected behaviours are $\dot{\text{C}}$ov and $\dot{\text{D}}$ep.

Further metrics are constructed and investigated using a machine learning approach to produce metrics which perform statistically well, but lack explainability. A series of FFNN models are trained to perform as correlation metrics using various configurations and sets of input data. While the unexplainable nature of metrics based on NNs makes them unattractive for engineering use, the results obtained show that practical and useful hardware can be constructed based on sets of counters.

Hardware implementations using the results of Chapter 3 are explored in Chapter 4. Real-time detection of correlation between signals with a slight (but potentially variable) offset in time presents a particular challenge for practical hardware due to the potential costs of buffering large amounts of data. This is tackled by introducing some randomness to the sampling process, effectively combining multiple time-offsets of a signal into one defined by a Gaussian probability distribution. A low-cost approach to windowing functions results in the contribution of a novel windowing function Logdrop which facilitates analysis focused on a particular region of time without requiring memory to store coefficients. Counters supporting jittery sampling and specific windowing functions are designed to collect the data found in Chapter 3 to offer the best value in terms of correlation information.

Hardware demonstrators are first implemented and characterised on the Lattice iCE40LP technology using the development process described in Appendix E. Subsequent implementations on FPGAs in the Xilinx 7-Series technology family are used to demonstrate practical usage of the correlator device. Correlations between the cache interfaces of an OpenPiton+Ariane SoC running Symmetric Multi-Processor (SMP) Linux are plotted in real-time using an oscilloscope. Incorporation with a commercial product line is also demonstrated using UltraSoC's message-based infrastructure for embedded analytics. This commercial integration is used to capture data from a running system, then several metrics are plotted, including metrics based on FFNNs described in Chapter 3.

Ergonomic visualisations are developed in Chapter 5 to ease the process of transferring correlation results, and thus knowledge of behaviour, into the minds

of engineers. The techniques developed are critically evaluated using 4 sets of visualisation heuristics and compared against existing tools and techniques. Two systems are used to demonstrate the visualisation techniques: (1) “praxi” for specific behaviours in a static context, (2) “tinn” for changes in behaviour. The netgraph technique combines correlations between all signals and their sibling signals into a circular network/graph form where the weight and colour of each edge/connection represents the level of correlation. Extensive use is made of the 2D colourspace described in Appendix I, because of the dense presentation of values it permits. The other presented visualisation technique, tabdelta, is used to navigate conveniently around a result space with a focus on highlighting correlations with specific time offsets.

6.2 Achieved Objectives and Contributions

The over-arching goal of this research is to improve the development process of complex SoCs which is achieved via a multi-pronged approach to behaviour analysis and the correlations comprising behaviour.

Chapter 3 makes three novel contributions:

1. A probabilistic model for “Binary SoC Data” which provides a basis for comparison between correlation metrics.

2. A quantitative evaluation of six interpretations of the term “correlation“. Interpretations based on the concepts of covariance (Ċov) and independence (Ḋep) are shown to be the most useful.

3. An evaluation of counter information for correlation metrics. A minimum set of counters equivalent to $\{\mathbb{E}[f_x], \mathbb{E}[f_y], \mathbb{E}[f_x \odot f_y], \mathbb{E}[|f_x - f_y|]\}$ is shown to provide most valuable information.

The objective of Chapter 3 is to characterise simple but effective metrics for estimating the presence of a logical relationship between binary SoC signals. Through the three novel contributions listed above, this objective has been achieved, with the results laying the foundations of the work in Chapter 4 and Chapter 5.

Chapter 4 also makes three novel contributions as it explores the practical issues surrounding the implementation of on-chip support for correlation (and thus behaviour) analysis.

1. An exposition on the issues around RTL implementation of circuits to support real-time monitoring for embedded analytics.

2. A jittery sampling mechanism which enables low-cost analysis over a configurable range of δ offsets.

3. A window function Logdrop designed specifically for low-cost implementation in digital logic.

Chapter 4 culminates in the design and demonstration of a correlator device which can be used as either a stand-alone tool or embedded into a larger SoC. The objective of Chapter 4, to explore the issues surrounding hardware support for correlation analysis of binary SoC signals, is achieved through the listed contributions and evidenced via demonstration of the correlator device.

Metrics evaluated in Chapter 3 are also used in Chapter 5, which focuses on presenting the results in an effective manner to efficiently communicate correlations and context to human viewers. Two novel techniques for visually presenting correlations are contributed:

1. Netgraph – for visualizing all inter-signal correlations in a single time window in a network/graph format.

2. Tabdelta – for visualizing a 2D slice of a result space using a table form, particularly the difference in correlation over different δ offsets.

The contributed techniques are subjected to heuristic evaluation using four sets of heuristics both in terms of individual features and in the gestalt.

Each contribution in Chapter 4 and Chapter 5 is accompanied by a reference implementation [1] and worked examples through case studies.

6.3 Future Work

This thesis presents several avenues for valuable future work to further the goal of improving SoC behaviour analysis using correlations.

- The probabilistic model of what constitutes "Binary SoC Data" may be improved using heuristics gathered from datasets of real systems. The model used in Chapter 3 has been shown to be sufficient for vastly different example systems (OpenPiton+Ariane, praxi, tinn). A probabilistic model is used in

[1] Hardware examples and systems including the correlator, OpenPiton+Ariane integration, and praxi are provided in a git repository at `https://github.com/DaveMcEwan/dmpvl/`, all written in SystemVerilog. Similarly, software tools are provided in a separate git repository at `https://github.com/DaveMcEwan/dmppl/` which includes the "relest" experiment used in Chapter 3 and the "eva" tool used in Chapter 5, all written in Python.

this work due to the lack of available datasets; however, a model based on the actual observed behaviour of a large collection of real SoCs may provide the basis for developing new and interesting correlation metrics.

- Developing case studies and example systems which are non-trivial was a significant challenge for this work, primarily due to commercial sensitivities around high-value SoC designs. A general trend towards more open hardware has been observed in recent years, exemplified by the rise of the openly developed RISC-V instruction set and its many free implementations. Access to more open hardware, especially that for performing complex workloads, provides the opportunity to refine and advance the analytic hardware and visualisation techniques described here.

- Section 3.5 trained a series of FFNN models as correlation metrics. Implementing metrics similar to these in hardware has the potential to provide high-quality real-time correlation analysis at a low resource cost. Further investigation of hardware-friendly activation functions, such as qsig in Equation (3.30), and power-efficient numerical bit-formats (e.g. posits [118, 119, 120]) are compelling directions for hardware implementations which are low-cost, low-power, real-time, and high-quality.

- Section 5.2.2 describes how different types of behaviour manifest as recognisable patterns in netgraph visualisations. Using machine learning techniques, these behaviours could be classified by a further level of correlation analysis. Behaviour recognition of this type may be interesting to the field of functional validation, particularly where low-level hardware probes are used to detect high-level software behaviours.

APPENDIX A

LIST OF PUBLICATIONS

- David McEwan and Jose Nunez-Yanez, "Relationship Estimation Metrics in Binary SoC Data," in *Machine Learning, Optimization, and Data Science (LOD)*, Sep 2019. [8]
- David McEwan, Marcin Hlond, and Jose Nunez-Yanez, "Visualizations for Understanding SoC behaviour," in *15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, Jul 2019. [132]

APPENDIX B

PARAMETER SETS

RTL modules are often parameterised in order to allow reuse across multiple projects. For example, a timer module might have a parameter to control the range that can be representable, thus limiting the number of inferred DFFs. Design parameters are used at all levels of SoC development and can be split according to two partitions shown as a linear set diagram [142] in Figure B.1.

Design parameters can first be partitioned into behavioural and implementation subsets.

- Behavioural parameters - Affect logical function; e.g. Depth of a buffer, width of cache line, cache eviction policy, etc. Effects of behavioural parameters may be determined using purely logic models.
- Implementation parameters - Do not affect logical function; e.g. Ripple carry adder vs carry-lookahead adder, circular vs linear buffer, clock distribution strategy, etc.

Or, alternatively, into logical and physical subsets.

- Logical parameters - Affect logical specification; e.g. Depth of buffer, circular vs linear buffer, ripple carry adder vs carry-lookahead adder etc.
- Physical parameters - Affect only the physical implementation; e.g. Cell placements, foundry process node, clock distribution strategy, etc.

Behavioural parameters are fundamental to the design because their effects are visible to higher levels such as software, whereas implementation parameters are

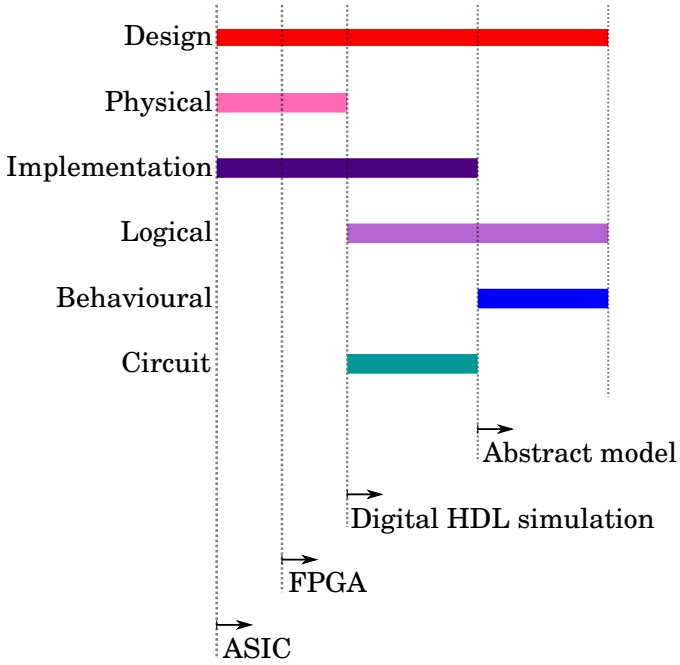


Figure B.1: Linear set diagram of SoC parameters. Labels underneath show which design levels each subset is visible to.

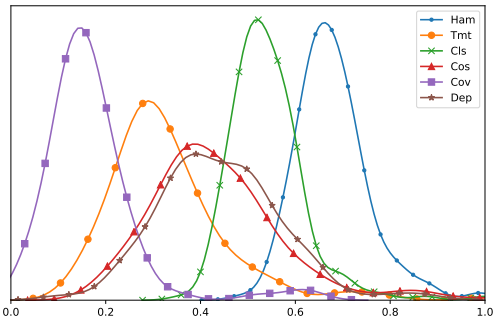
used for fine-grained tweaking intended to be functionally transparent. Similarly, logical parameters are more fundamental than physical parameters because they are specified by the frontend rather than the backend. Circuit parameters affect only the circuit being programmed into a FPGA (or etched into an ASIC), but not the behaviour of the system. The effects of circuit parameters cannot be measured using logical simulations because they require a hardware implementation to be characterised. As such, circuit parameters make the most subtle design contributions and are the most difficult to analyse using simulations. At the prototype stage, a circuit parameter choosing between a ripple adder or a carry-lookahead adder is less important than a behavioural parameter which chooses the adder's result width.

This thesis is concerned only with logical parameters, all of which may be used in Hardware Description Language (HDL) source such as 4-state SystemVerilog [37] or 9-state VHDL [143].

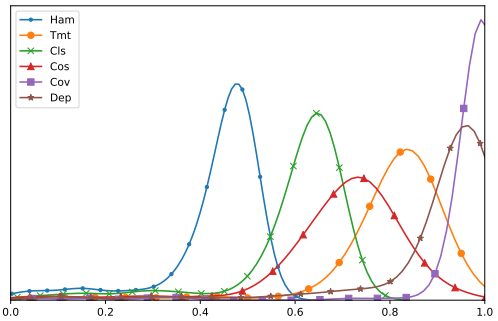
APPENDIX C

RESULTS FROM SECTION 3.4 BY SYSTEM TYPE

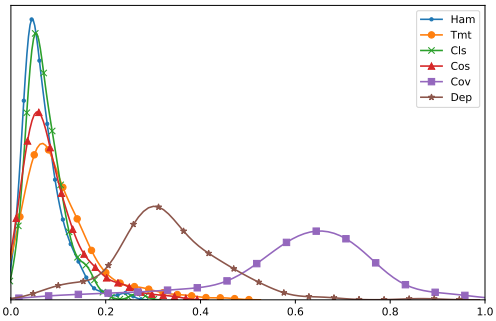
KDE plots of accuracy PDF by system type. More weight on the right-hand side is always better. X-axis is accuracy from 0 (never correct) to 1 (always correct). Y-axis is the probability density that the metric will have that score (unitless).



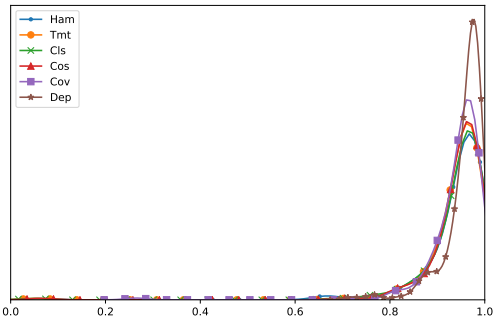
(a) True Positive Rate (Sensitivity).



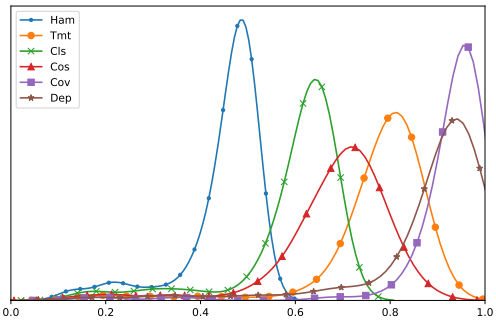
(b) True Negative Rate (Specificity).



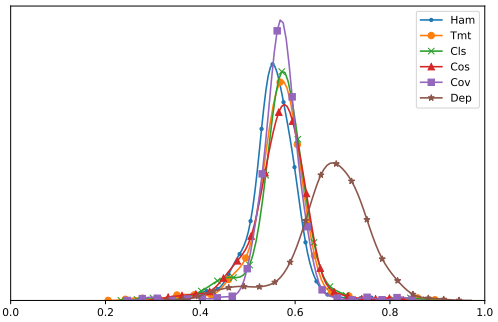
(c) Positive Predictive Value (Precision).



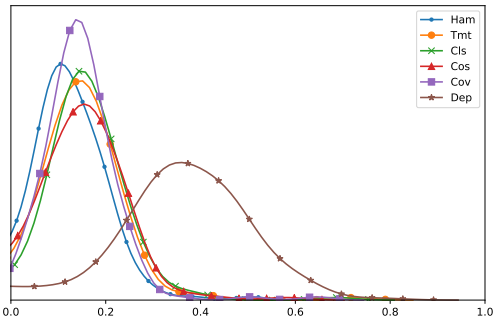
(d) Negative Predictive Value.



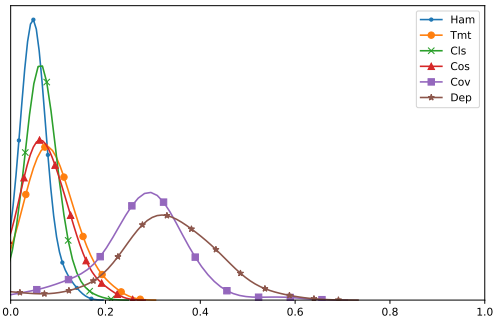
(e) Accuracy.



(f) Balanced Accuracy.

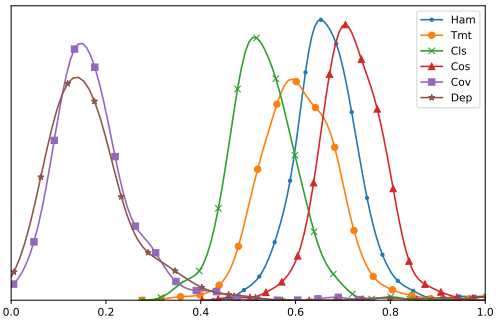


(g) Book-Maker's Informedness.

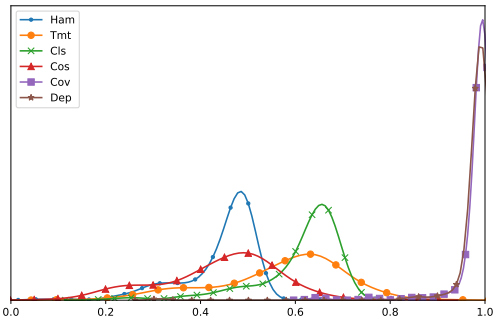


(h) Matthews Correlation Coefficient.

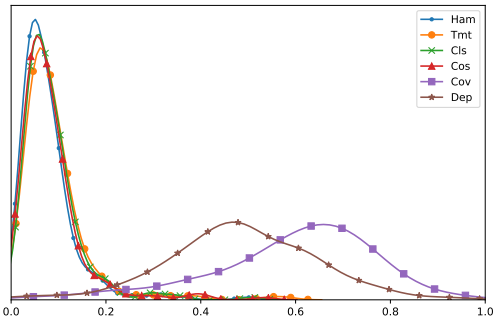
Figure C.1: KDE plots of score PDFs for systems with AND-only logical relationships. More weight on the right-hand side is always better.



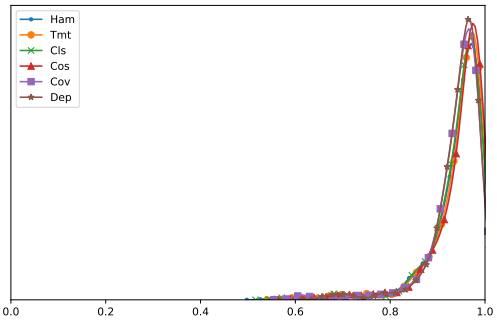
(a) True Positive Rate (Sensitivity).



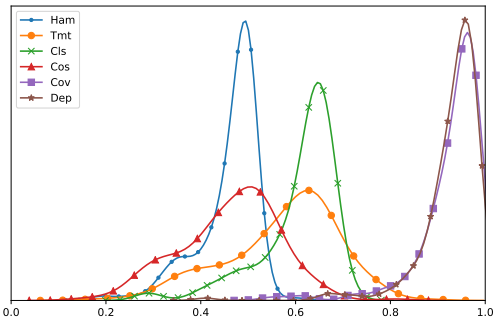
(b) True Negative Rate (Specificity).



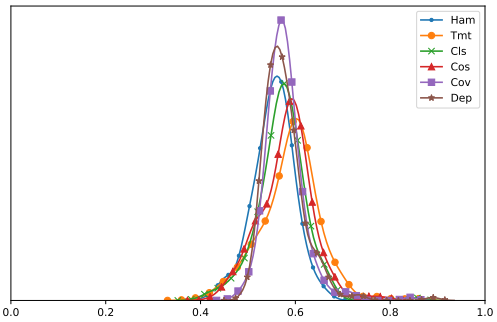
(c) Positive Predictive Value (Precision).



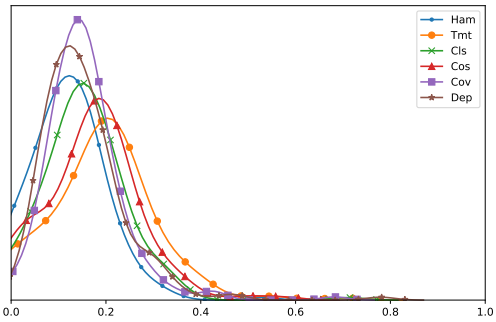
(d) Negative Predictive Value.



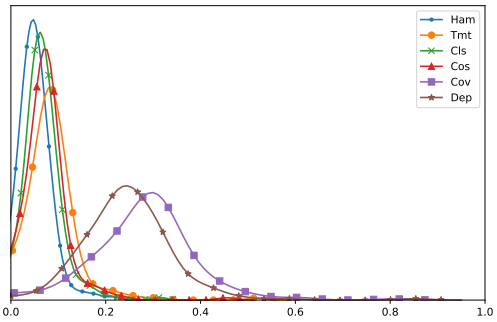
(e) Accuracy.



(f) Balanced Accuracy.

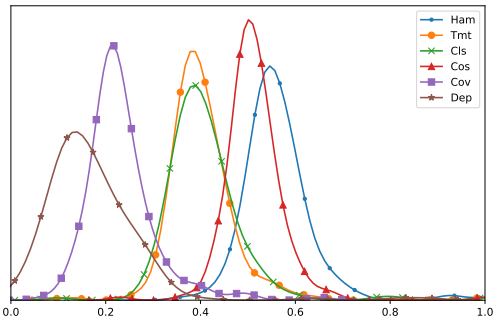


(g) Book-Maker's Informedness.

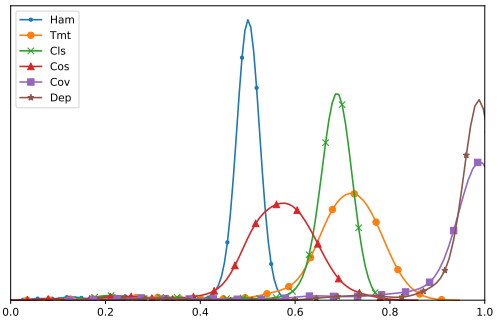


(h) Matthews Correlation Coefficient.

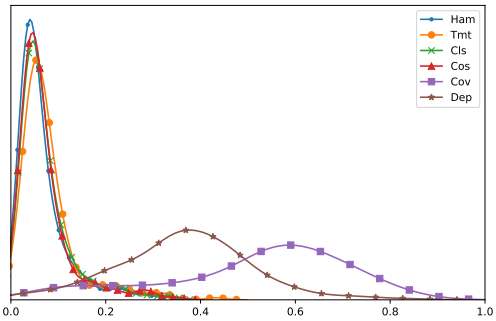
Figure C.2: KDE plots of score PDFs for systems with OR-only logical relationships. More weight on the right-hand side is always better.



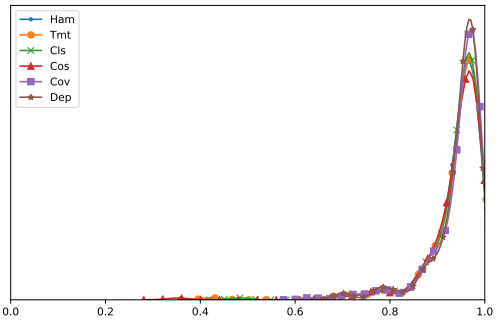
(a) True Positive Rate (Sensitivity).



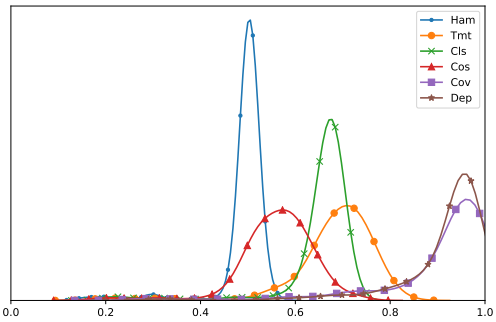
(b) True Negative Rate (Specificity).



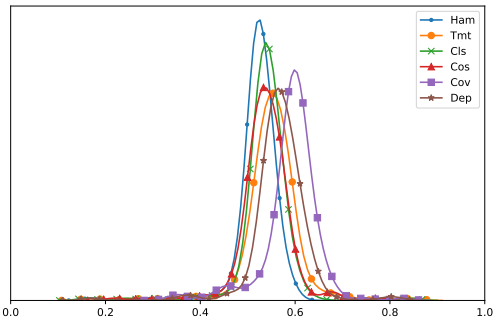
(c) Positive Predictive Value (Precision).



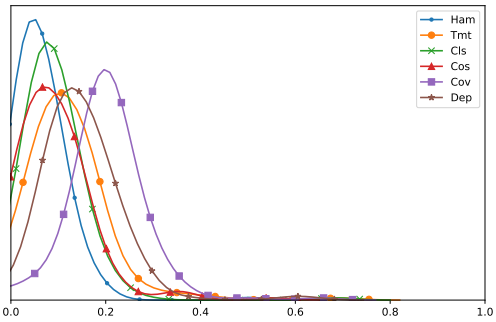
(d) Negative Predictive Value.



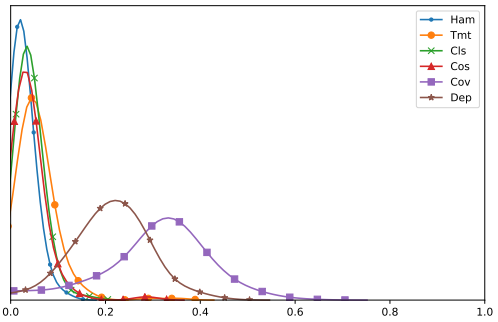
(e) Accuracy.



(f) Balanced Accuracy.

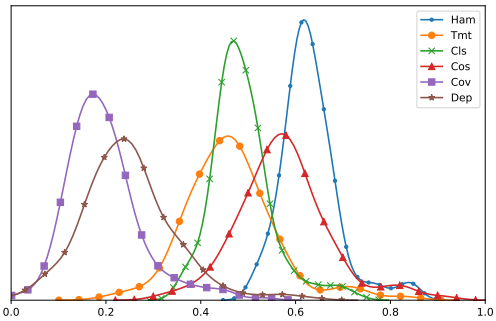


(g) Book-Maker's Informedness.

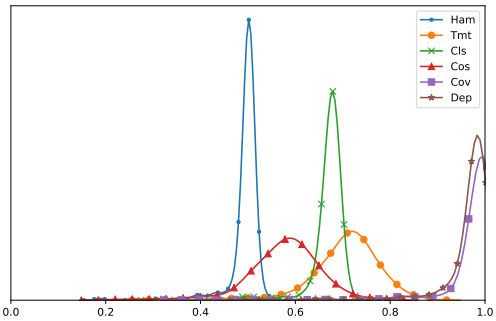


(h) Matthews Correlation Coefficient.

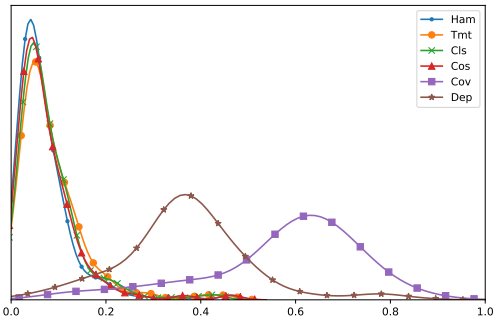
Figure C.3: KDE plots of score PDFs for systems with XOR-only logical relationships. More weight on the right-hand side is always better.



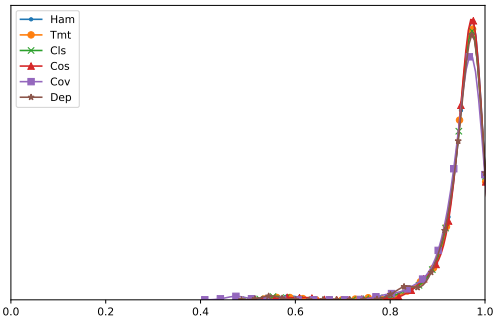
(a) True Positive Rate (Sensitivity).



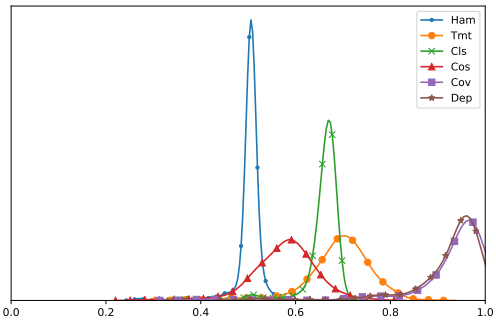
(b) True Negative Rate (Specificity).



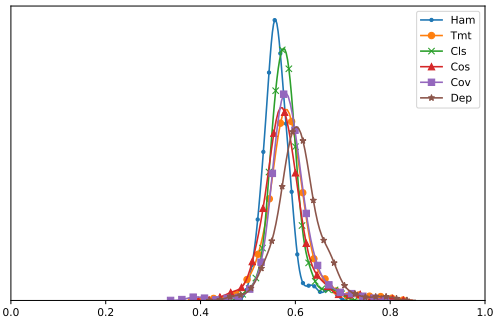
(c) Positive Predictive Value (Precision).



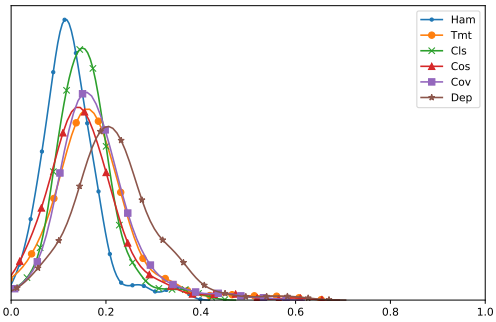
(d) Negative Predictive Value.



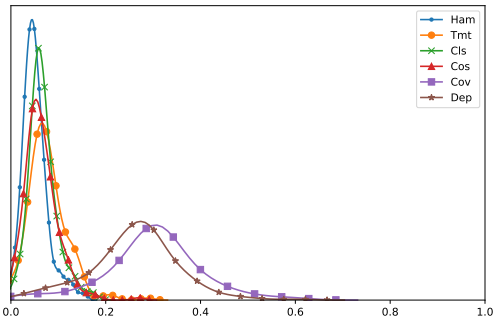
(e) Accuracy.



(f) Balanced Accuracy.

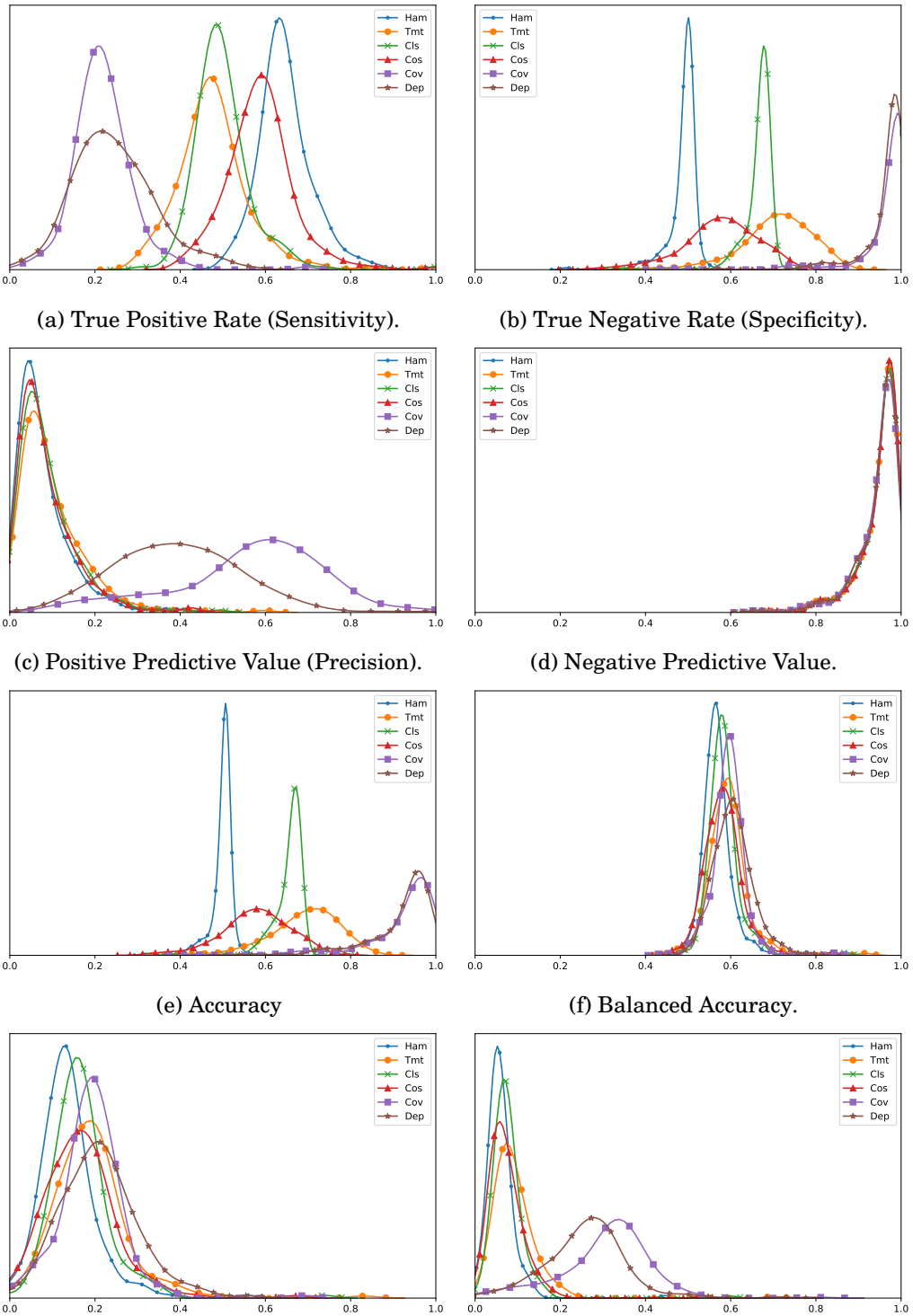


(g) Book-Maker's Informedness.



(h) Matthews Correlation Coefficient.

Figure C.4: KDE plots of score PDFs for systems with a mix of AND-only, OR-only, and XOR-only logical relationships.



(a) True Positive Rate (Sensitivity).

(b) True Negative Rate (Specificity).

(c) Positive Predictive Value (Precision).

(d) Negative Predictive Value.

(e) Accuracy

(f) Balanced Accuracy.

(g) Book-Maker's Informedness.

(h) Matthews Correlation Coefficient.

Figure C.5: KDE plots of score PDFs for systems with logical relationships defined by left-hand-associative functions composed of AND, OR, XOR operations.

APPENDIX D

Model Summaries for FFNN-based Metrics

D.1 Counter Inputs Combination `perfCntrs`

```
Model: "perfCntrs_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_54 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_56 (Dense)             (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
EVAL 8 perfCntrs_2qsig_0_sigm loss=-0.0002 acc=0.0445 mse=0.8861
```

```
Model: "perfCntrs_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_20 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_57 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_58 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_59 (Dense)             (None, 1)                 3
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
EVAL 9 perfCntrs_2qsig_2qsig_sigm loss=-0.0013 acc=0.9555 mse=0.0443
```

```
Model: "perfCntrs_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_48 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_49 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_50 (Dense)             (None, 1)                 5
=================================================================
Total params: 37
Trainable params: 37
Non-trainable params: 0
_________________________________________________________________
EVAL 6 perfCntrs_4qsig_4qsig_sigm loss=-0.0004 acc=0.9555 mse=0.0442
```

```
Model: "perfCntrs_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_15 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_42 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_43 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_44 (Dense)             (None, 1)                 5
=================================================================
Total params: 37
Trainable params: 37
Non-trainable params: 0
_________________________________________________________________
EVAL 4 perfCntrs_4sigm_4sigm_sigm loss=-0.0002 acc=0.0445 mse=0.6579
```

```
Model: "perfCntrs_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_16 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_45 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_46 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 5
=================================================================
Total params: 37
Trainable params: 37
Non-trainable params: 0
_________________________________________________________________
EVAL 5 perfCntrs_4tanh_4tanh_sigm loss=-0.0165 acc=0.3076 mse=0.6577
```

```
Model: "perfCntrs_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_30 (Dense)             (None, 8)                 24
_________________________________________________________________
dense_31 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_32 (Dense)             (None, 1)                 9
=================================================================
Total params: 105
Trainable params: 105
Non-trainable params: 0
_________________________________________________________________
EVAL 0 perfCntrs_8qsig_8qsig_sigm loss=-0.0003 acc=0.0445 mse=0.9356
```

```
Model: "perfCntrs_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_39 (Dense)             (None, 8)                 24
_________________________________________________________________
dense_40 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 9
=================================================================
Total params: 105
Trainable params: 105
Non-trainable params: 0
_________________________________________________________________
EVAL 3 perfCntrs_8relu_8relu_sigm loss=-0.0652 acc=0.9555 mse=0.0440
```

```
Model: "perfCntrs_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_33 (Dense)             (None, 8)                 24
_________________________________________________________________
dense_34 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_35 (Dense)             (None, 1)                 9
=================================================================
Total params: 105
Trainable params: 105
Non-trainable params: 0
_________________________________________________________________
EVAL 1 perfCntrs_8sigm_8sigm_sigm loss=-0.0003 acc=0.0445 mse=0.8416
```

```
Model: "perfCntrs_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 2)]               0
_________________________________________________________________
dense_36 (Dense)             (None, 8)                 24
_________________________________________________________________
dense_37 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 9
=================================================================
Total params: 105
Trainable params: 105
Non-trainable params: 0
_________________________________________________________________
EVAL 2 perfCntrs_8tanh_8tanh_sigm loss=-0.0149 acc=0.2467 mse=0.7244
```

D.2 Counter Inputs Combination `withAssist`

```
Model: "withAssist_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_59 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_174 (Dense)            (None, 2)                 16
_________________________________________________________________
dense_176 (Dense)            (None, 1)                 3
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
EVAL 8 withAssist_2qsig_0_sigm loss=-0.3776 acc=0.9555 mse=0.0434
```

```
Model: "withAssist_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_60 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_177 (Dense)            (None, 2)                 16
_________________________________________________________________
dense_178 (Dense)            (None, 2)                 6
_________________________________________________________________
dense_179 (Dense)            (None, 1)                 3
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
EVAL 9 withAssist_2qsig_2qsig_sigm loss=-0.4129 acc=0.9555 mse=0.0438
```

```
Model: "withAssist_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_57 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_168 (Dense)            (None, 4)                 32
_________________________________________________________________
dense_169 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_170 (Dense)            (None, 1)                 5
=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
EVAL 6 withAssist_4qsig_4qsig_sigm loss=-0.4333 acc=0.9543 mse=0.0413
```

```
Model: "withAssist_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_55 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_162 (Dense)            (None, 4)                 32
_________________________________________________________________
dense_163 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_164 (Dense)            (None, 1)                 5
=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
EVAL 4 withAssist_4sigm_4sigm_sigm loss=-0.4120 acc=0.9520 mse=0.0443
```

```
Model: "withAssist_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_56 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_165 (Dense)            (None, 4)                 32
_________________________________________________________________
dense_166 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_167 (Dense)            (None, 1)                 5
=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
EVAL 5 withAssist_4tanh_4tanh_sigm loss=-0.4748 acc=0.9506 mse=0.0462
```

```
Model: "withAssist_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_51 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_150 (Dense)            (None, 8)                 64
_________________________________________________________________
dense_151 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_152 (Dense)            (None, 1)                 9
=================================================================
Total params: 145
Trainable params: 145
Non-trainable params: 0
_________________________________________________________________
EVAL 0 withAssist_8qsig_8qsig_sigm loss=-0.4534 acc=0.9512 mse=0.0456
```

```
Model: "withAssist_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_54 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_159 (Dense)            (None, 8)                 64
_________________________________________________________________
dense_160 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_161 (Dense)            (None, 1)                 9
=================================================================
Total params: 145
Trainable params: 145
Non-trainable params: 0
_________________________________________________________________
EVAL 3 withAssist_8relu_8relu_sigm loss=-0.4738 acc=0.9538 mse=0.0447
```

```
Model: "withAssist_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_52 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_153 (Dense)            (None, 8)                 64
_________________________________________________________________
dense_154 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_155 (Dense)            (None, 1)                 9
=================================================================
Total params: 145
Trainable params: 145
Non-trainable params: 0
_________________________________________________________________
EVAL 1 withAssist_8sigm_8sigm_sigm loss=-0.4326 acc=0.9524 mse=0.0444
```

```
Model: "withAssist_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_53 (InputLayer)        [(None, 7)]               0
_________________________________________________________________
dense_156 (Dense)            (None, 8)                 64
_________________________________________________________________
dense_157 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_158 (Dense)            (None, 1)                 9
=================================================================
Total params: 145
Trainable params: 145
Non-trainable params: 0
_________________________________________________________________
EVAL 2 withAssist_8tanh_8tanh_sigm loss=-0.4660 acc=0.9514 mse=0.0467
```

D.3 Counter Inputs Combination fullAssist

```
Model: "fullAssist_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_24 (Dense)             (None, 2)                 24
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 3
=================================================================
Total params: 27
Trainable params: 27
Non-trainable params: 0
_________________________________________________________________
EVAL 8 fullAssist_2qsig_0_sigm loss=-0.3828 acc=0.9547 mse=0.0393
```

```
Model: "fullAssist_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 11)]              0
_________________________________________________________________
dense_27 (Dense)             (None, 2)                 24
_________________________________________________________________
dense_28 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 3
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
EVAL 9 fullAssist_2qsig_2qsig_sigm loss=-0.4144 acc=0.9520 mse=0.0440
```

```
Model: "fullAssist_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_18 (Dense)             (None, 4)                 48
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 5
=================================================================
Total params: 73
Trainable params: 73
Non-trainable params: 0
_________________________________________________________________
EVAL 6 fullAssist_4qsig_4qsig_sigm loss=-0.4118 acc=0.9544 mse=0.0411
```

```
Model: "fullAssist_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 48
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 5
=================================================================
Total params: 73
Trainable params: 73
Non-trainable params: 0
_________________________________________________________________
EVAL 4 fullAssist_4sigm_4sigm_sigm loss=-0.4176 acc=0.9531 mse=0.0425
```

```
Model: "fullAssist_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_15 (Dense)             (None, 4)                 48
_________________________________________________________________
dense_16 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 5
=================================================================
Total params: 73
Trainable params: 73
Non-trainable params: 0
_________________________________________________________________
EVAL 5 fullAssist_4tanh_4tanh_sigm loss=-0.4270 acc=0.9536 mse=0.0446
```

```
Model: "fullAssist_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense (Dense)                (None, 8)                 96
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
EVAL 0 fullAssist_8qsig_8qsig_sigm loss=-0.4356 acc=0.9536 mse=0.0442
```

```
Model: "fullAssist_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_9 (Dense)              (None, 8)                 96
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
EVAL 3 fullAssist_8relu_8relu_sigm loss=-0.4756 acc=0.9543 mse=0.0445
```

```
Model: "fullAssist_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 96
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
EVAL 1 fullAssist_8sigm_8sigm_sigm loss=-0.4198 acc=0.9535 mse=0.0434
```

```
Model: "fullAssist_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
dense_6 (Dense)              (None, 8)                 96
_________________________________________________________________
dense_7 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
EVAL 2 fullAssist_8tanh_8tanh_sigm loss=-0.4615 acc=0.9524 mse=0.0458
```

D.4 Counter Inputs Combination `withIsect`

```
Model: "withIsect_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_39 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_114 (Dense)            (None, 2)                 8
_________________________________________________________________
dense_116 (Dense)            (None, 1)                 3
=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
EVAL 8 withIsect_2qsig_0_sigm loss=-0.0162 acc=0.9555 mse=0.0436
```

```
Model: "withIsect_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_40 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_117 (Dense)            (None, 2)                 8
_________________________________________________________________
dense_118 (Dense)            (None, 2)                 6
_________________________________________________________________
dense_119 (Dense)            (None, 1)                 3
=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
EVAL 9 withIsect_2qsig_2qsig_sigm loss=-0.0080 acc=0.9555 mse=0.0433
```

```
Model: "withIsect_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_37 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_108 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_109 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_110 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 6 withIsect_4qsig_4qsig_sigm loss=-0.1311 acc=0.9555 mse=0.0440
```

```
Model: "withIsect_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_35 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_102 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_103 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_104 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 4 withIsect_4sigm_4sigm_sigm loss=-0.0328 acc=0.8777 mse=0.0938
```

```
Model: "withIsect_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_36 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_105 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_106 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_107 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 5 withIsect_4tanh_4tanh_sigm loss=-0.2503 acc=0.9555 mse=0.0426
```

```
Model: "withIsect_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_31 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_90 (Dense)             (None, 8)                 32
_________________________________________________________________
dense_91 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_92 (Dense)             (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 0 withIsect_8qsig_8qsig_sigm loss=-0.0548 acc=0.9555 mse=0.0445
```

```
Model: "withIsect_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_34 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_99 (Dense)             (None, 8)                 32
_________________________________________________________________
dense_100 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_101 (Dense)            (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 3 withIsect_8relu_8relu_sigm loss=-0.3273 acc=0.9546 mse=0.0440
```

```
Model: "withIsect_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_32 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_93 (Dense)             (None, 8)                 32
_________________________________________________________________
dense_94 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_95 (Dense)             (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 1 withIsect_8sigm_8sigm_sigm loss=-0.1980 acc=0.9555 mse=0.0441
```

```
Model: "withIsect_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_33 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_96 (Dense)             (None, 8)                 32
_________________________________________________________________
dense_97 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_98 (Dense)             (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 2 withIsect_8tanh_8tanh_sigm loss=-0.3363 acc=0.9546 mse=0.0441
```

D.5 Counter Inputs Combination withSymdiff

```
Model: "withSymdiff_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_49 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_144 (Dense)            (None, 2)                 8
_________________________________________________________________
dense_146 (Dense)            (None, 1)                 3
=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
EVAL 8 withSymdiff_2qsig_0_sigm loss=-0.0909 acc=0.9400 mse=0.0514
```

```
Model: "withSymdiff_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_50 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_147 (Dense)            (None, 2)                 8
_________________________________________________________________
dense_148 (Dense)            (None, 2)                 6
_________________________________________________________________
dense_149 (Dense)            (None, 1)                 3
=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
EVAL 9 withSymdiff_2qsig_2qsig_sigm loss=-0.1844 acc=0.9555 mse=0.0425
```

```
Model: "withSymdiff_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_47 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_138 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_139 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_140 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 6 withSymdiff_4qsig_4qsig_sigm loss=-0.2471 acc=0.9555 mse=0.0437
```

```
Model: "withSymdiff_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_45 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_132 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_133 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_134 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 4 withSymdiff_4sigm_4sigm_sigm loss=-0.2485 acc=0.9555 mse=0.0445
```

```
Model: "withSymdiff_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_46 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_135 (Dense)            (None, 4)                 16
_________________________________________________________________
dense_136 (Dense)            (None, 4)                 20
_________________________________________________________________
dense_137 (Dense)            (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
EVAL 5 withSymdiff_4tanh_4tanh_sigm loss=-0.2860 acc=0.9546 mse=0.0431
```

```
Model: "withSymdiff_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_41 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_120 (Dense)            (None, 8)                 32
_________________________________________________________________
dense_121 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_122 (Dense)            (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 0 withSymdiff_8qsig_8qsig_sigm loss=-0.2576 acc=0.9555 mse=0.0444
```

```
Model: "withSymdiff_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_44 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_129 (Dense)            (None, 8)                 32
_________________________________________________________________
dense_130 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_131 (Dense)            (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 3 withSymdiff_8relu_8relu_sigm loss=-0.3156 acc=0.9546 mse=0.0438
```

```
Model: "withSymdiff_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_42 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_123 (Dense)            (None, 8)                 32
_________________________________________________________________
dense_124 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_125 (Dense)            (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 1 withSymdiff_8sigm_8sigm_sigm loss=-0.2757 acc=0.9555 mse=0.0445
```

```
Model: "withSymdiff_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_43 (InputLayer)        [(None, 3)]               0
_________________________________________________________________
dense_126 (Dense)            (None, 8)                 32
_________________________________________________________________
dense_127 (Dense)            (None, 8)                 72
_________________________________________________________________
dense_128 (Dense)            (None, 1)                 9
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
EVAL 2 withSymdiff_8tanh_8tanh_sigm loss=-0.2651 acc=0.9555 mse=0.0440
```

D.6 Counter Inputs Combination `withIsectSymdiff`

```
Model: "withIsectSymdiff_2qsig_0_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_29 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_84 (Dense)             (None, 2)                 10
_________________________________________________________________
dense_86 (Dense)             (None, 1)                 3
=================================================================
Total params: 13
Trainable params: 13
Non-trainable params: 0
_________________________________________________________________
EVAL 8 withIsectSymdiff_2qsig_0_sigm loss=-0.1613 acc=0.9555 mse=0.0432
```

```
Model: "withIsectSymdiff_2qsig_2qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_30 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_87 (Dense)             (None, 2)                 10
_________________________________________________________________
dense_88 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_89 (Dense)             (None, 1)                 3
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
EVAL 9 withIsectSymdiff_2qsig_2qsig_sigm loss=-0.2176 acc=0.9555 mse=0.0444
```

```
Model: "withIsectSymdiff_4qsig_4qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_27 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_78 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_79 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_80 (Dense)             (None, 1)                 5
=================================================================
Total params: 45
Trainable params: 45
Non-trainable params: 0
_________________________________________________________________
EVAL 6 withIsectSymdiff_4qsig_4qsig_sigm loss=-0.2196 acc=0.9555 mse=0.0426
```

```
Model: "withIsectSymdiff_4sigm_4sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_72 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_73 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_74 (Dense)             (None, 1)                 5
=================================================================
Total params: 45
Trainable params: 45
Non-trainable params: 0
_________________________________________________________________
EVAL 4 withIsectSymdiff_4sigm_4sigm_sigm loss=-0.2486 acc=0.9555 mse=0.0444
```

```
Model: "withIsectSymdiff_4tanh_4tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_75 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_76 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_77 (Dense)             (None, 1)                 5
=================================================================
Total params: 45
Trainable params: 45
Non-trainable params: 0
_________________________________________________________________
EVAL 5 withIsectSymdiff_4tanh_4tanh_sigm loss=-0.2573 acc=0.9555 mse=0.0425
```

```
Model: "withIsectSymdiff_8qsig_8qsig_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_60 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_61 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_62 (Dense)             (None, 1)                 9
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
EVAL 0 withIsectSymdiff_8qsig_8qsig_sigm loss=-0.2627 acc=0.9555 mse=0.0445
```

```
Model: "withIsectSymdiff_8relu_8relu_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_24 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_69 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_70 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_71 (Dense)             (None, 1)                 9
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
EVAL 3 withIsectSymdiff_8relu_8relu_sigm loss=-0.2965 acc=0.9555 mse=0.0445
```

```
Model: "withIsectSymdiff_8sigm_8sigm_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_22 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_63 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_64 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_65 (Dense)             (None, 1)                 9
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
EVAL 1 withIsectSymdiff_8sigm_8sigm_sigm loss=-0.2590 acc=0.9555 mse=0.0432
```

```
Model: "withIsectSymdiff_8tanh_8tanh_sigm"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_23 (InputLayer)        [(None, 4)]               0
_________________________________________________________________
dense_66 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_67 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_68 (Dense)             (None, 1)                 9
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
EVAL 2 withIsectSymdiff_8tanh_8tanh_sigm loss=-0.3374 acc=0.9542 mse=0.0441
```

APPENDIX E

QUANTIFIED CONFIDENCE IN f_{MAX} ROBUSTNESS

Chapter 4 develops an RTL design (correlator) which is implemented first as a stand-alone device, then as a passive integrated monitoring component in a larger external SoC (OpenPiton+Ariane). The following design aims were followed for the correlator device, but are also applicable for developing general RTL designs suitable for easy integration with larger projects.

1. Compatibility with industry-standard RTL toolchains - Relying only on widely-supported tool features ensures that integration with external projects is straightforward, as demonstrated via case studies.

2. Portability to a variety of logic technologies - Although it is impractical to manufacture ASICs to confirm a design's portability to different process nodes, FPGAs are routinely used commercially to prototype digital logic systems. Successful implementation of the design on multiple FPGA technologies gives confidence in portability to ASIC process nodes.

3. Low resource requirements - For integration to another project, a large increase in resource requirements, such as LUTs, DSP slices, or memory blocks, is likely to affect the synthesis and PnR process.

4. Low power requirements - An integration target which includes any power-dependent behaviours such as Dynamic Voltage and Frequency Scaling (DVFS) cannot be passively analysed if the action of analysis consumes so much power that behaviour is significantly altered.

5. Parameterised design - Allowing an integrator the flexibility to make trade-off decisions using parameters is preferable to forcing onerous modifications to RTL logic.

6. High f_{max} - Deep logic and the associated restrictions on clock speed are undesirable because this would restrict target integrations to those with low clock speeds.

Standard digital logic implementation workflows read in RTL code (e.g. written in VHDL or Verilog), synthesise a netlist (e.g. formatted as BLIF [144], EDIF [145], or a restricted subset of Verilog), then assign the elements of that netlist to real on-chip components with a PnR tool. Synthesis tools operate using an sequence of "passes" to transform the abstract syntax tree from RTL into a network of connections between primitive components. The function and order of these passes determine what the resulting netlist looks like and there are many different ways to produce a netlist which conforms to the same RTL specification. Finding the ideal result of the PnR process is a hard problem with no known globally optimal solution, so heuristic methods are employed. Using a seed value, which may be chosen pseudo-randomly, an initial placement is created, then improved upon using an iterative process such as simulated annealing [30]. This means that it is possible to influence the resulting bitstream by setting the seed value such that some values produce better solutions than others in terms of f_{max} or power consumption.

E.1 Confidence In A Single Design

The guidelines of portability and high f_{max} are adhered to by characterising individual design components in isolation. It is therefore useful to examine the build process of intermediate prototype devices which implement only a minimal set of design features. An advantage of developing on the Lattice iCE40 family is that there is a high-quality FOSS toolchain from RTL to bitstream. This means that many experiments may be run in parallel and on many different machines without license restrictions. Exploiting this streamlined workflow allows designs to be characterised by running PnR many times, using the same netlist but different seed values for the initial placement. Plotting the PMF of the achieved f_{max} then provides a quantification of how robust a design is against variations in the PnR step.

On a modern workstation PnR for the Lattice iCE40 technology takes around 5 s to 20 s to complete the process of converting a netlist to a bitstream using either

arachne-pnr [128] or *nextpnr* [146] on a single CPU core. This type of Multiple Place-and-Route (Multi-PnR) experiment is performed using 1000 seed values to obtain a statistically significant visualisation of how likely a design is likely to achieve a particular value of f_{max}.

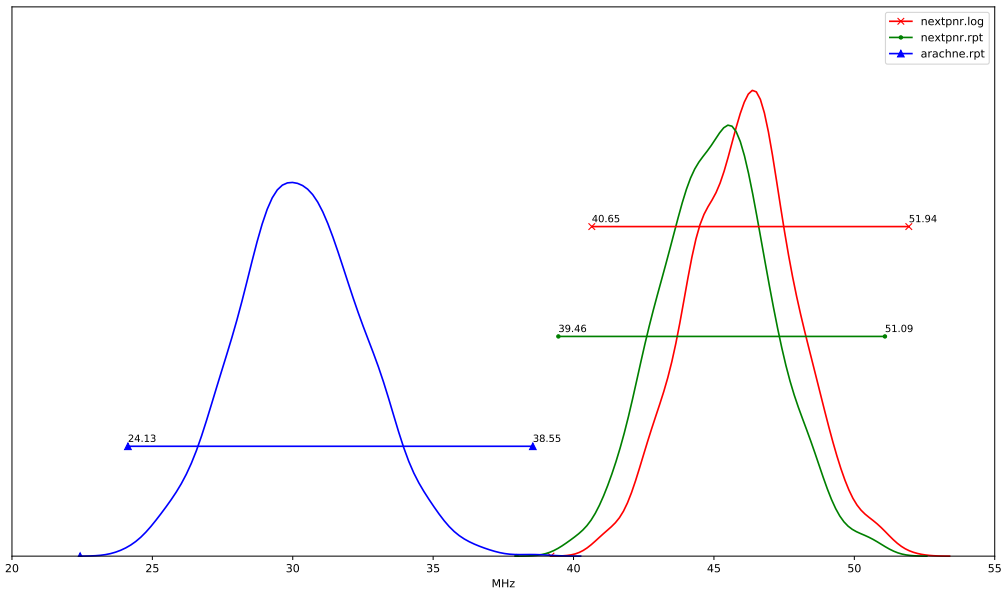


Figure E.1: PMF of Multi-PnR results of a prototype design on a Lattice iCE40LP8K FPGA implementing a 100 B memory accessed via BytePipe over USB-FS. A legacy PnR tool *arachne-pnr* which does not attempt to optimise for timing is seen (in blue) to achieve the slowest solutions. Results are reported by analysing the netlists with *icetime*. A new PnR tool *nextpnr* produces faster solutions, also analysed by icetime (shown in green). Results reported directly by nextpnr (shown in red) are seen to be slightly more optimistic.

Figure E.1 shows the results from a prototype used to characterise an implementation of a memory-mapped register block over the BytePipe protocol (described in Appendix G). Immediately obvious is that using nextpnr provides consistently better results than arachne. The blue plot refers to bitstreams produced with arachne, and the value of f_{max} found using an external bitstream analysis tool *icetime*. The red and green plots both refer to the same bitstreams but use different methods to extract the value of f_{max}. Red shows the f_{max} reported directly by nextpnr, and green is the value reported using icetime which makes it directly comparable to blue. Icetime uses an algorithm which is intended to be pessimistic and can be seen by the green showing a most likely f_{max} of around 45 MHz and red showing a most likely f_{max} of around 47 MHz.

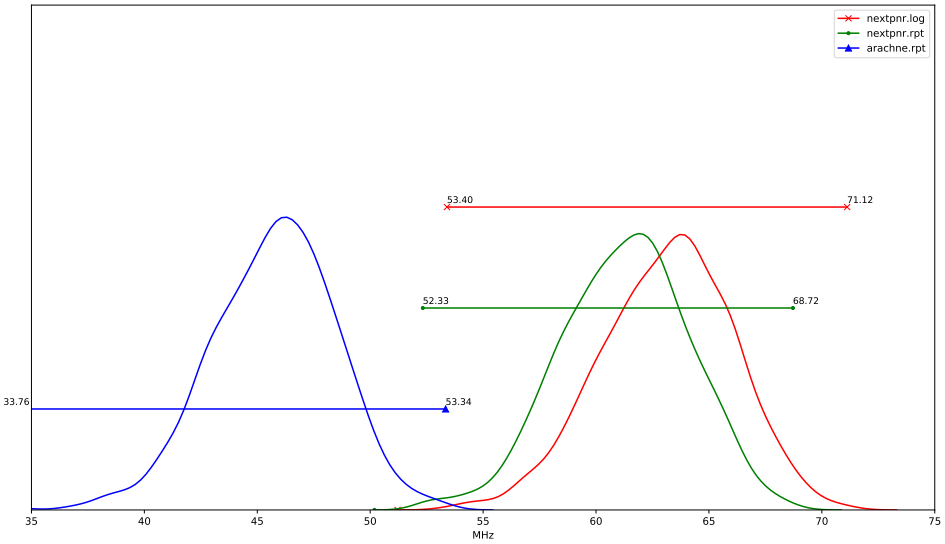
Statistical analysis of the pseudo-random PnR process provides a quantifiable

level of confidence that the design will meet or exceed a particular f_{max} goal. For example, from Figure E.1 it can be seen that the probability that a randomly chosen PnR seed will produce a bitstream capable of functioning correctly at 30 MHz is approximately $\frac{1}{2}$ when using arachne, and approximately 1 when using nextpnr.

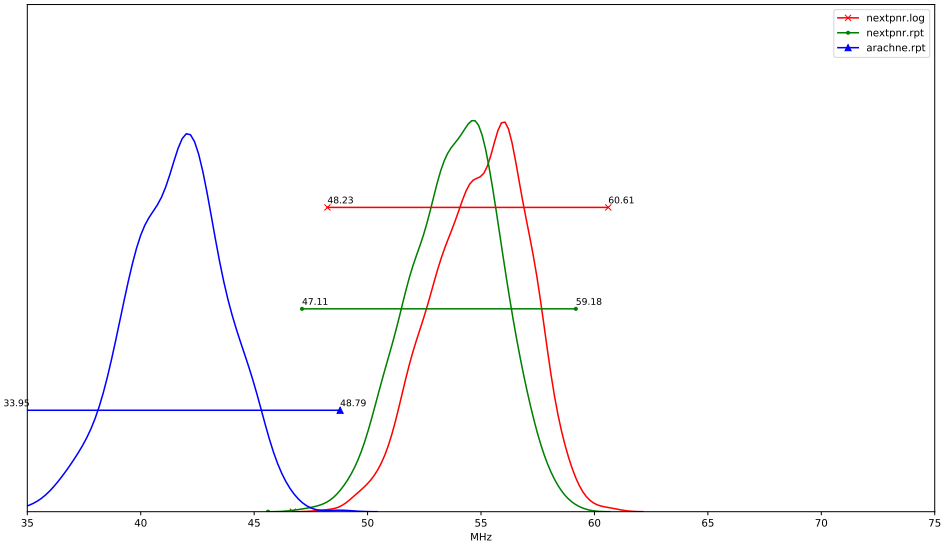
E.2 Comparison of Competing Designs

The shape of these f_{max} PMF plots also provides useful information to the engineer evaluating a design. A shorter, fatter, flatter plot indicates that a design is quite dependent on the initial placement in the PnR process, whereas a taller thinner plot suggests a design which is less affected by PnR. Figure E.2 shows a comparison of two functionally equivalent designs with different implementations. It can be seen that Figure E.2a indicates a design which is likely to be easier to integrate with other components due to the higher f_{max} and lower LUT requirement.

An ideal plot for a stand-alone design would be a tall thin spike on the right-hand side showing that the design will reliably produce a fast bitstream regardless of the PnR process used. Alternatively, an ideal plot for a design intended to be integrated as part of a larger system may be a rather low bump towards the right-hand side. A flattish right-trending bump indicates that the design is highly dependent on physical placement and therefore may require some seed experimentation to fit around other placement-restricted components in a larger design. A plot with multiple peaks suggests that the algorithm used by the PnR tool has found multiple local minima in the solution space, which demonstrates the importance of running this kind of analysis to find a good seed value for a final production bitstream.



(a) Multi-PnR results of a prototype design on a Lattice iCE40LP8K FPGA implementing a USB-FS serial port gadget which simply echoes bytes. Requires 762 LUTs.



(b) Multi-PnR results of a functionally equivalent design based on work by Luke Valenty, Lawrie Griffiths, and David Williams. Requires 1015 LUTs.

Figure E.2: Multi-PnR comparison of functionally equivalent design choices. Figure E.2b is based on the bootloader design shipped with the TinyFPGA-BX, whereas Figure E.2a is a functionally equivalent design produced as part of this research.

E.3 Comparing Similar Configurations

A statistical approach to the PnR design step is also useful for understanding the physical limitations of a design. The results plotted in Figure E.3 provide a compelling demonstration where six similar designs are compared using the $f_{\max}$ reported by a single tool.

Marsaglia [147] first introduced the xorshift family of PRNG algorithms as an alternative to the popular Mersenne Twister algorithms which use a large state of 2.5 KiB in the most popular variant MT19937. An analysis by Brent [148] notes that xorshift PRNGs are equivalent to well-understood Linear Feedback Shift Register (LFSR) algorithms which are known to fail statistical tests in the standard DIEHARD and TestU01 [149] suites. Vigna provided a thorough exploration [150, 151] of the xorshift family and proposed using addition or multiplication operations to scramble the output in order to improve statistical qualities. The xoroshiro family of PRNG algorithms developed by Vigna and Blackman were originally designed to produce pseudo-random sequences of 64 b or 32 b numbers which pass randomness tests and execute quickly in software. The components of the naming prefix (xo, ro, shi) indicate the number of XOR, rotate, and shift bitwise operations, the number (64, 128, 256) is the size of state in bits, and symbols (+, *) indicate arithmetic operations. From the practical hardware perspective of implementing these algorithms in digital logic, the operations involved are attractive due to their simplicity and low cost. Rotate and shift operations are zero-cost as they translate to a static arrangement of wires between D-type flip-flops which implement the state. XOR operations can be implemented using a single gate or LUT, and may be combined with other logic using Boolean reduction. Addition and multiplication operations usually require significantly more logic with the carry signal being the slowest to propagate.

The xoroshiro128+ algorithm is chosen as a reference point because it has middle values for its attributes such as 128 b state and a single 64 b addition. Four other algorithms from the xoroshiro family are examined which have similar attributes which can be neatly compared. Other algorithms of this family are likely to have implementation problems on a slow FPGA process such as Lattice iCE40LP because of either a large amount of state (causing congestion problems), or deep logic (causing timing problems).

The experimental platform used is in the form of a USB-FS gadget using the TinyFPGA-BX development board and the Multi-PnR method described in Appendix E. A baseline design was implemented containing the logic required for the USB Abstract Control Module subsystem and provides a simple memory-map

protocol which consumes 894 LUTs, 745 of which are the USB subsystem. The number of LUT cells reported in Table E.1 is that given by the PnR tool (NextPnR), minus the 894 of the baseline implementation. While it is possible for the synthesis tool (Yosys) to optimise away unnecessary logic, the baseline implementation has been designed to avoid these effects as much as possible. The PnR tool is also used to report $f_{\max}$ using the same flow for all algorithms. The PnR process contains random elements such as generating an initial layout which the tool attempts to improve on, and the pseudo-random process is controlled using a seed value. One thousand PnR solutions of each algorithm's netlist are created to see a fuller picture of what value of $f_{\max}$ should be expected. By plotting the distribution of reported $f_{\max}$ values in Figure E.3 the shape of each plot highlights characteristics of the design which are heavily affected by the pseudo-random PnR process.

Algorithm	Comments wrt. xoroshiro128+	#LUTs	$f_{\max}$ (MHz)
xoroshiro128+		381	60.70
xoshiro128++	2× 32 b add vs. 1× 64 b add	312	59.69
xoshiro128+	32 b vs. 64 b add	272	68.26
xoshiro256+	256 b vs. 128 b state	472	60.70
xoroshiro64*	32 b constant multiply vs. 64 b add	910	48.96

Table E.1: Comparison of PRNG implementations on Lattice iCE40LP.

Examples of different Multi-PnR plot shapes are shown in Figure E.3 where artefacts of the FPGA technology are visible such as faster routing paths for adders, including their hard limit, as well as restrictions from routing congestion, and the deep logic of a large multiplier. Artefacts of the FPGA technology are discerned by the shape of the Multi-PnR plots:

- *xoroshiro128+*, requiring 128 b of state and a single 64 b adder, is a middle-valued member of this family of PRNGs providing a point of comparison for the other four algorithms. The tall spike at the maximum achievable $f_{\max}$ of 60.70 MHz indicates that initial layout has little effect on $f_{\max}$ because the 64 b carry chain is a fundamentally limiting factor.

- *xoshiro128++* also requires 128 b of state, but needs two sequential 32 b adders instead of one 64 b adder. The deeper logic of two smaller adders, compared to one adder, prevents carry chain optimisation, which leads to the slightly lower $f_{\max}$ limit of 59.69 MHz. The limit on achievable $f_{\max}$ appears to be softer than that of xoroshiro128+, presumably because the PnR tool has more options on how to place two smaller adders compared to one large adder.

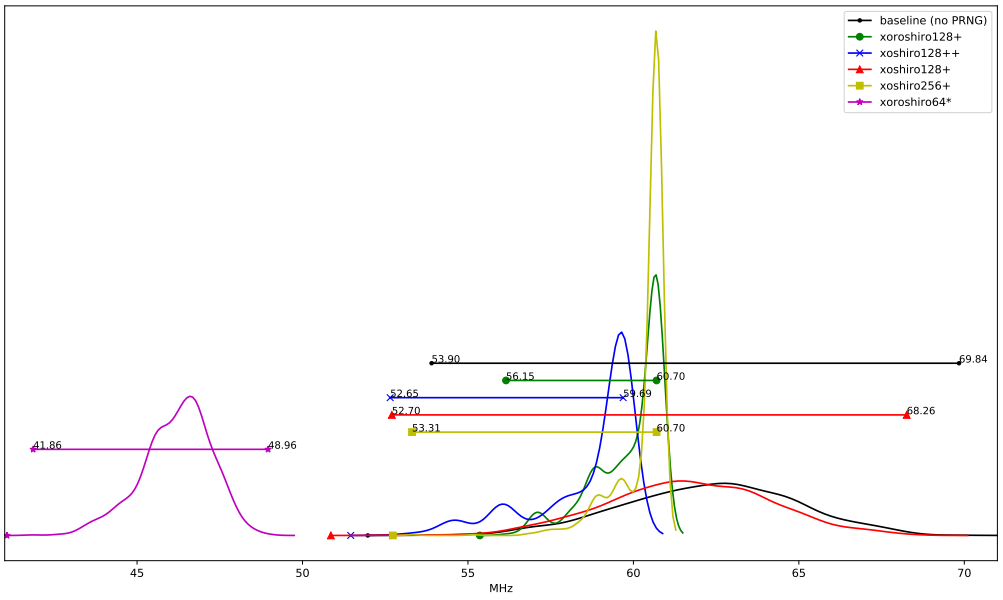


Figure E.3: Multi-PnR results of a prototype design on a Lattice iCE40LP8K FPGA implementing different PRNGs from the Xoroshiro family. All results are those directly reported by nextpnr. A baseline design which returns a constant number (shown in black) demonstrates that the underlying USB-FS subsystem is highly dependent on the PnR process but may achieve up to 69.84 MHz.

- *xoshiro128+*, requiring only a single 32 b adder, produces PnR solutions achieving $f_{\max}$ nearly as fast as the baseline. The difference is presumed due to the extra congestion of 272 LUTs which constitutes around 30% of the netlist

- *xoshiro256+* requires twice the amount of state compared to xoroshiro128+ which, due to congestion, appears to limit the feasible PnR options for the 64 b adder. The resulting narrow spike of the PMF indicates that this configuration is not very dependent on initial layout, but may present problems when integrated with another system if other parts restrict PnR layout.

- *xoroshiro64** requires a 64 b multiplier. The Lattice iCE40 technology does not provide specific cells for multipliers, so this must be synthesised using LUT cells, resulting in deep logic paths that are unable to operate at high frequency. The deep logic paths can be seen in timing reports to be the prominent factor limiting $f_{\max}$ to 48.96 MHz.

APPENDIX F

MODELLING THE JITTERY STROBE COUNTER CIRCUIT

Chapter 4 presents a circuit which produces a single-cycle assertion on `o_strobe` at nearly regular intervals. See Figure 4.5 for the logical circuit design which is based around a counter c. The nominal (mean) period is configurable via `ctrlPeriod`, as is the amount of jitter (variance) via `ctrlJitter`. For brevity, let s be the non-negative integer held in `ctrlPeriod`, and let j be the unit-normalised real held in `ctrlJitter` in Q0.8 format; i.e. $s \in [0, 2^{20})$ and $j \in [0, 1)$.

A PRNG is used to introduce (pseudo-)randomness to the strobe interval via two random variables $J \sim \text{Bernoulli}(j)$ and $K \sim \text{Bernoulli}\left(\frac{1}{2}\right)$. On each clock cycle, J determines whether to alter the period, then K determines whether that alteration is an extension or a contraction. At the beginning of an interval (in the cycle immediately following an assertion of `o_strobe`) the counter is initialised $c = s - 1$. The down-counter process is such that on each clock cycle a contraction decrements c by 2, no alteration decrements c by 1, and an extension leaves c unchanged; i.e. with no contractions or extensions, c reaches 0 in exactly s cycles. In a practical circuit the concept of negative c is removed; i.e. the contraction paths which could lead to $c = -1$ and $c = -2$ are absent, as shown in Figure F.1

This process is a variant of the simple random walk on $\mathbb{Z}$, therefore the deviation from an unaltered down-counter after n cycles ($s - n - c$) can be modelled by the binomial distribution $B\left(n, \frac{j}{2}\right)$. Each cycle makes one step with $\Pr = j$, and that step is $+1$ instead of -1 with $\Pr = \frac{1}{2}$. The Cumulative Density Function (CDF) of the deviation after n cycles is given in Equation (F.1). However, this approximation is not completely accurate because of the two missing contraction paths.

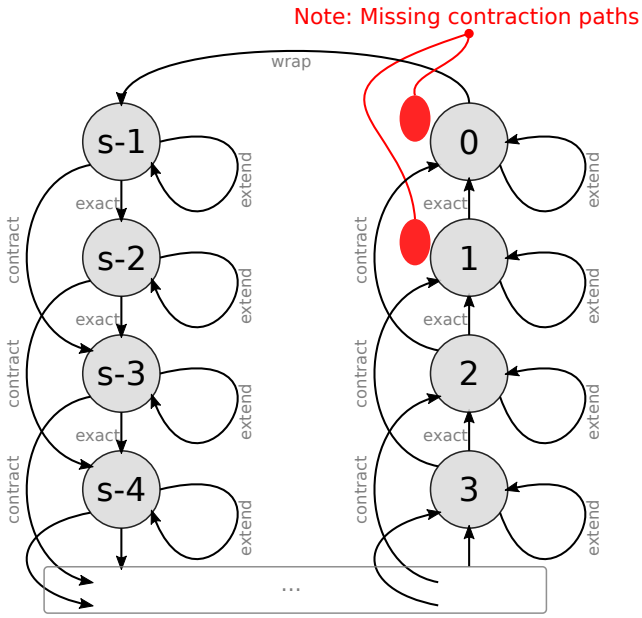


Figure F.1: State machine of strobe generation down-counter c.

$$\Pr(c \geqslant s \mid n) \approx \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{n-s}{\sqrt{sj}\sqrt{2}}\right)\right] \tag{F.1}$$

Let $q_0 = q_2 = \frac{j}{2}$ and $q_1 = (1-j)$ refer to the probability of the counter decrementing by 0, 2, or 1 respectively. Let $p_{(c,n)}$ be the probability of c reaching exactly 0 in n cycles, as defined in Equation (F.2). The Cumulative Mass Function (CMF) of the counter's deviation is given in Equation (F.3).

$$p_{(c,n)} = \begin{cases} q_c & : n = 1, c \in \{0,1,2\} \\ q_0 p_{(c,n-1)} & : c = 0 \\ q_0 p_{(c,n-1)} + q_1 p_{(c-1,n-1)} & : c = 1 \\ q_2 p_{(c-2,n-1)} & : c = 2n \\ q_1 p_{(c-1,n-1)} + q_2 p_{(c-2,n-1)} & : c = 2n-1 \\ q_0 p_{(c,n-1)} + q_1 p_{(c-1,n-1)} + q_2 p_{(c-2,n-1)} & : \text{otherwise} \end{cases} \tag{F.2}$$

$$\Pr(c \geqslant s \mid n) = \begin{cases} 0 & : s > 2n \\ \sum_{d \in [s,2n]} p_{(d,n)} & : \text{otherwise} \end{cases} \tag{F.3}$$

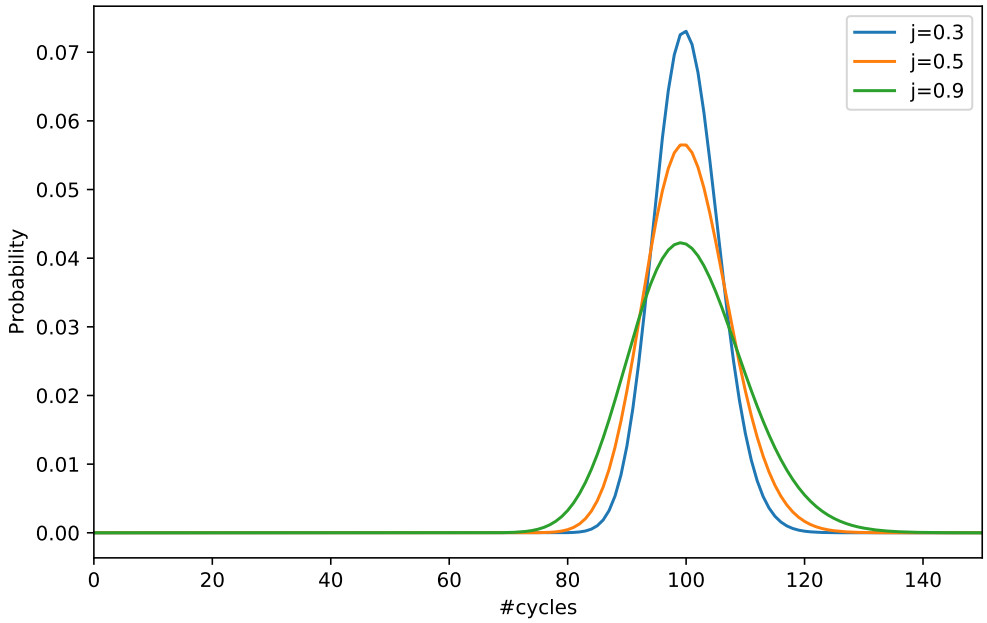


Figure F.2: PMF of number of cycles in a period with $s = 100$.

Equation (F.3) can be used for precise modelling of the intervals between assertions of `o_strobe`, or in case s is small. The discrete first derivative of Equation (F.3) gives the PMF of period length, as plotted in Figure F.2. Where s is sufficiently large, the PMF of a Binomial distribution approximates the PDF of a Gaussian Normal distribution. The recursive nature of Equation (F.3) requires intensive computation where s is large, but Equation (F.1) provides a reasonable approximation.

Python code demonstrating that the explicit recursive approach (Equation (F.3)) matches the continuous approximation (Equation (F.1)) is provided at:
`https://github.com/DaveMcEwan/dmpvl/blob/master/misc/strobe/plotStrobePeriodPMF.py`.

APPENDIX G

BytePipe Protocol

The purpose of the BytePipe protocol is to provide a memory map of up to 127 B, using an underlying "base" protocol capable of flow-controlled 8 b Flow Control Digits (flits). Each byte on the 127 B map is individually accessible for both read and write requests. BytePipe is designed to be simple to implement and low-cost in terms of power and LUT/area. It is assumed that the base protocol uses a handshaking mechanism to source an ordered stream of bytes losslessly, and this assumpion allows BytePipe to be implemented with shallow logic circuits. Common base protocols which are suitable include UART, USB, Unix TTY, and anything else which may be represented as a pair of 8b-wide FIFOs flowing in opposite directions.

Although BytePipe can be used to implement a 127 B Random Access Memory (RAM), its intended purpose is to provide a structure and control mechanism for SoC peripheral registers. The behaviour of each addressable byte is user-defined for addresses 1 to 127, but address 0 is reserved for use controlling burst behaviour. All burst transactions have an overhead of 5 B. The maximum efficiency of a burst read is $\frac{255}{255+5}$, just over 98%. The maximum efficiency of a burst write is $\frac{256}{256+5}$, also just over 98%.

Each byte location may be used in any of the following suggested modes, although this is not an exhaustive list.

- RW (Read and write)
  - Single byte of RAM, where reads return the last value written.
  - Reads and writes return and affect separate bits of state.

  - Data pushed/pulled to/from a First In First Out (FIFO).

- R (Read-only) Writing has no effect.
  - Non-programmable constant, such as an identifier like "core number 5".
  - Non-constant value, such as a status like “number of entries currently in queue”.
  - Data pulled from FIFO.

- W (Write-only) Reading returns a constant, or an unspecified value.
  - Initiation such as "begin processing when any value written here".
  - Set alias or clear alias to another set of bits.
  - Data pushed to FIFO.

Flow control uses `valid`/`ready` handshaking with the same semantics as AXI.

1. A byte has been transmitted when both `valid` and `ready` are asserted in the same clock cycle. The byte being “transmitted” means that the receiver has taken a valid sample of `data` and the sender should move onto sending another byte.
2. Sender may not change `data` until both `valid` and `ready` have been used to indicate the byte has been transmitted.
3. Sender must not wait for `ready` before asserting `valid` when there the sender has data to send.
4. Receiver must not wait for `valid` before asserting `ready`, so during a period of inactivity `ready` will be continually asserted.

There are a small number of simple rules governing the 17 mandatory bits of state comprising the protocol FSM (`addr` : 7 b, `burst` : 8 b, `rd` : 1 b, `wr` : 1 b). A transaction is initiated by the host sending the device one byte, while the device is not already processing a transaction. The `addr` register is updated on the initiation of each transaction. All single transactions cause the device to produce a 1 B response. The `burst` register may be written using address 0 to indicate that the next transaction is “burst”, rather than “single”.

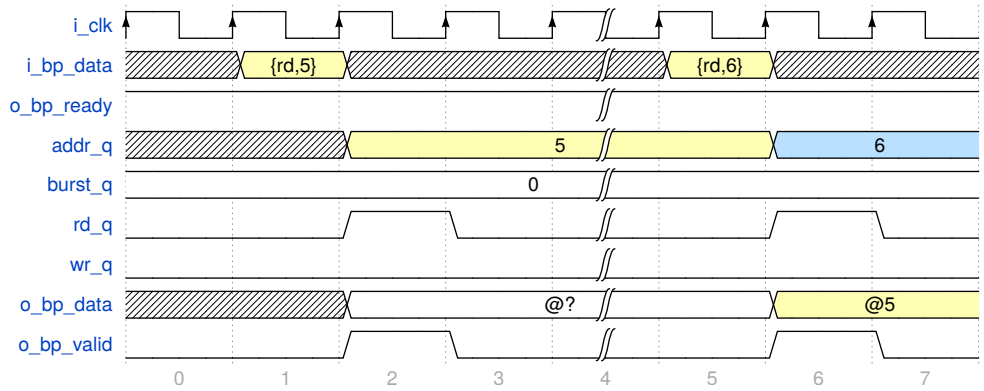


Figure G.1: A single read transaction is initiated by the host sending a single byte with the top bit clear; i.e. unsigned integer < 128. The device immediately returns a byte containing the value at whatever address was previously in `addr`, then updates `addr`. For example, if `addr` contains `0x12`, and the host begins a single read transaction by sending `0x55`, then the device will return the contents of location `0x12` and update `addr` to be `0x55`.

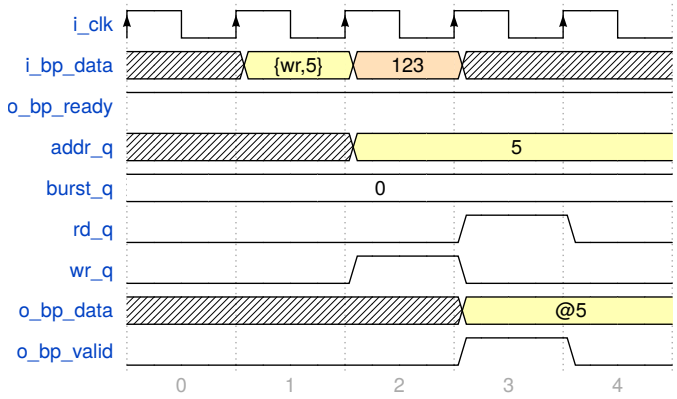


Figure G.2: A single write transaction is initiated by the host sending a single byte with the top bit set; i.e. unsigned integer > 127. The device immediately begins waiting for the data byte, and updates `addr`. The next byte received by the device is then written to that the location pointed to by `addr`. This second byte causes the device to return 1 B containing the value which is now overwritten, in effect a write acknowledgement. For example, if the host begins a single write transaction by sending `0xD5`, then the device will update `addr` to `0x55`. When the host sends the next byte, `0xAA` the device will write the value `0xAA` to address `0x55`.

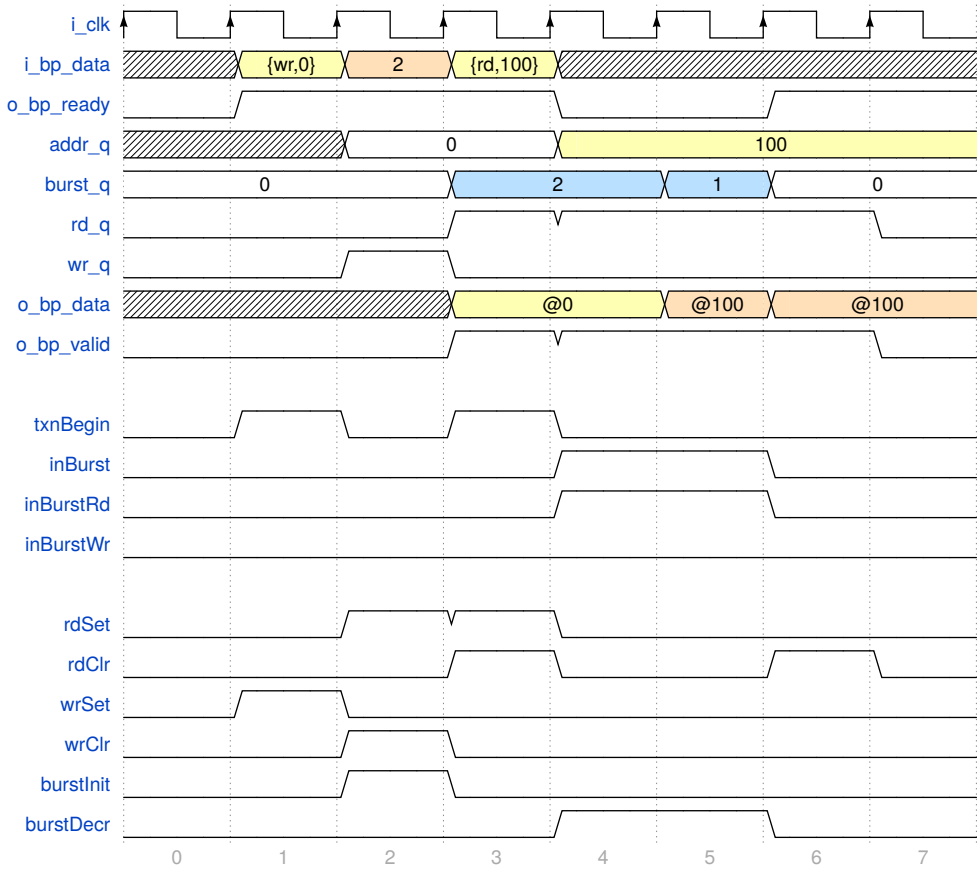


Figure G.3: A read burst is initiated by the host by performing a single write transaction to address 0 where the value is the number of bytes desired. Therefore, a read burst may be used to receive up to 255 B. The host then sends another byte with the top bit clear and the address in the lower 7 b, as with a single read. The first byte returned is the value at address 0, which is implementation-specific. Each subsequent byte returned by the device decrements `burst`, and when `burst` = 0 the transaction is complete.

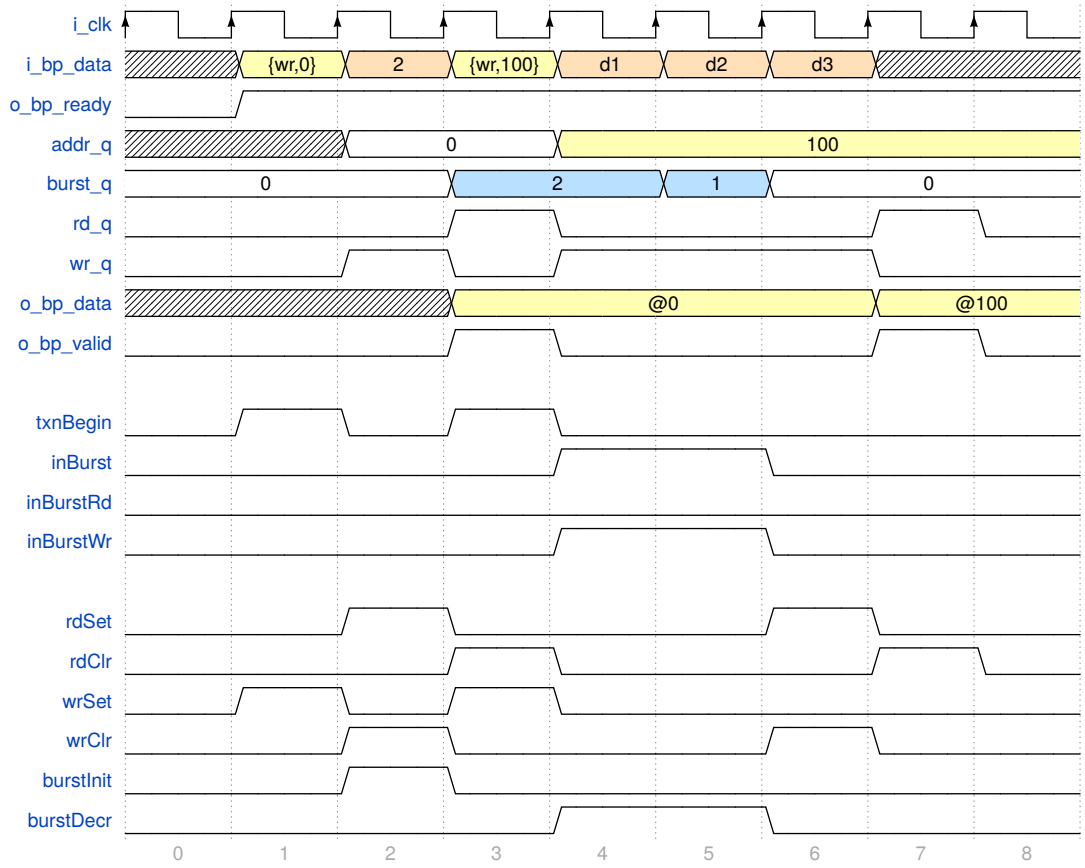


Figure G.4: A write burst is initiated by the host by performing a single write transaction to address 0 where the value is one less than the number of bytes desired. Therefore, a write burst may be used to send up to 256 B. The host then sends another byte with the top bit set and the address in the lower 7 b, as with a single write. Each subsequent byte received by the device decrements `burst`, and when `burst` = 0 the transaction is complete.

APPENDIX H

USB-FS Electrical Interface Over Five Pins

On the TinyFPGA-BX development board there is an existing USB socket, complete with termination resistors and GPIO pins operating at 3.3 V. This makes developing a single USB Full Speed interface on the TinyFPGA-BX a simple affair, requiring only RTL logic to drive the appropriate GPIO pins.

The USB-FS specification requires the data lines `D+`,`D-` to be driven with a low state of 0 V to 0.3 V and a high state of 2.8 V to 3.6 V. A 1.5 kΩ pull-up resistor connecting `D+` to 3.3 V is used to indicate USB-FS mode. Conveniently, this allows 3.3 V GPIO pins to interface directly with a USB-FS host without additional components [152].

In order for the same RTL logic to function on FPGA technologies which drive GPIO pins at different voltage ranges, an external electrical interface Printed Circuit Board (PCB) is required. The Xilinx 7-Series, for example, operates GPIO at 1.8 V.

Using this external circuitry, it is practical to add a USB-FS interface to the lower-voltage Xilinx 7-Series development boards. Alternatively, this external circuitry enables the addition of extra USB-FS interfaces to the higher-voltage TinyFPGA-BX.

Full KiCAD [153] schematics and PCB layout files in Gerber format are available at: `https://github.com/DaveMcEwan/dmpvl/tree/master/misc/extUsb`.

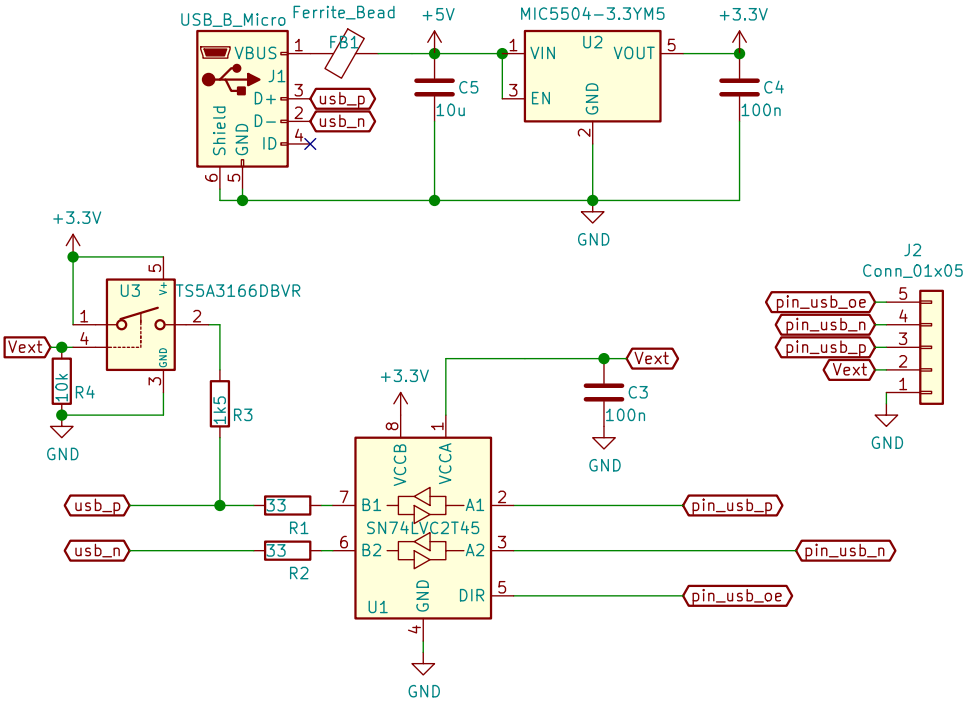
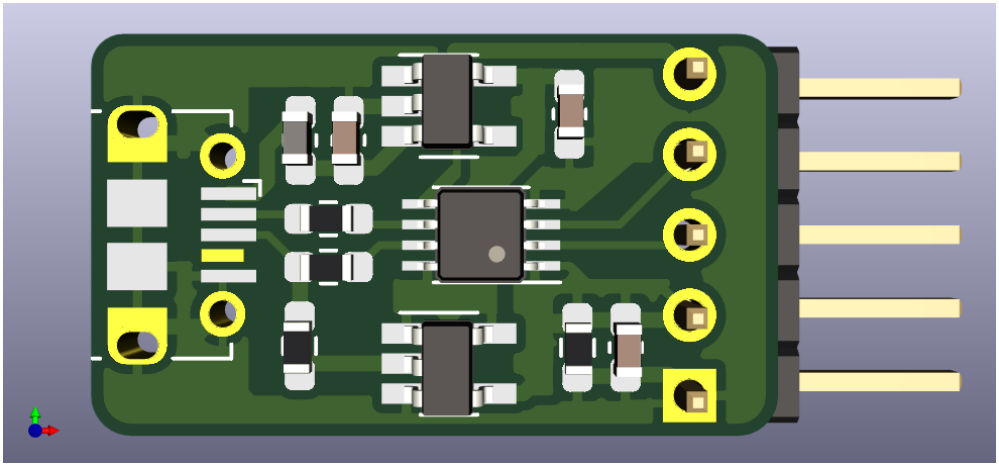
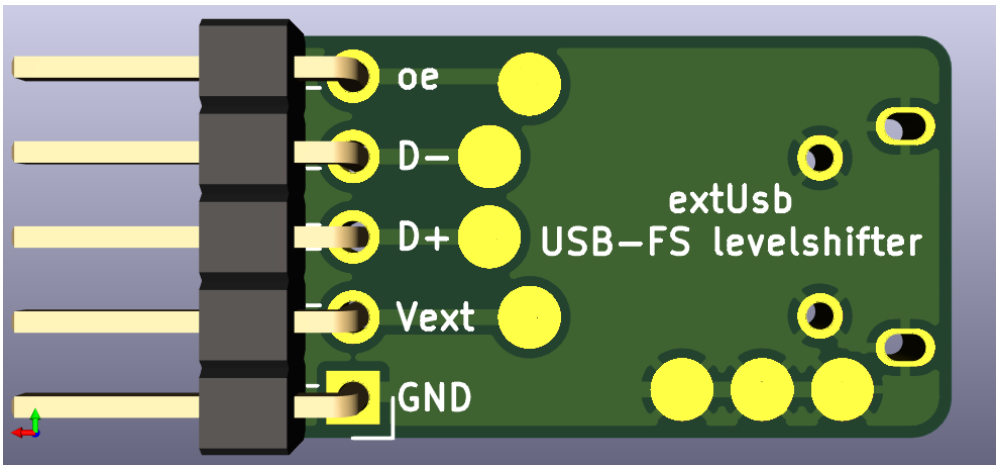


Figure H.1: Schematic for external USB-FS electrical interface. The 2 b level-shifter (`U1`) and low-dropout regulator (`U2`) allow FPGAs with GPIO voltages in the range 1.65 V to 5.5 V to safely use the USB lines `D+` and `D-` which operate at 3.3 V. The analogue switch (`U3`) connects the 1.5 kΩ pull-up (`R3`) when a voltage greater than 1.65 V is applied between `Vext` and `GND`. This allows a USB host to detect when the FPGA connected to `J2` is undergoing re-programming. The host will then be able to automatically disconnect, re-connect, and re-enumerate once programming has completed. Series termination resistors (`R1` and `R2`) provide some protection from electrical noise introduced from both the transceiver switching circuitry and environmental effects on the USB cable. The capacitors and ferrite bead are used to stabilise the constant voltage nodes (`+5V`, `+3.3V`, `Vext`) and suppress the propagation of noise into the USB cable.



(a) Top view.



(b) Bottom view.

Figure H.2: PCB rendering of external USB-FS electrical interface. Outline dimensions are 12.75 mmx22.75 mm. USB Micro B header is not rendered.

APPENDIX I

COLOURSPACE FOR BOUNDED 2D DATA

Not everybody perceives the same colours in the same way, even among different people with no diagnosed colour-blindness. This was first demonstrated by Ishihara [154] where a series of tests was developed to quantify and classify colour vision. As our perception of colour comes from a combination of psychological phenomena and the densities of various types of cell in the retina [101], the Ishihara tests are a vast simplification of a higher-dimensional space [155].

Despite differences in perception between individuals, colour is used pervasively to present multi-attribute datasets in a dense and uncluttered manner. Greater ability to distinguish additional dimensions of data reduces the amount of time spent browsing between different plots. In turn, this reduces the mental load on a viewer by giving them more time to think about higher things. Despite differences in human perception, colour is found to be extremely useful for presenting rich datasets in a dense and uncluttered manner. This appendix defines and demonstrates a colourspace which is perceived similarly by both those with normal colour vision and those with protanopia, the most common form of colour-blindness, often called red-green colour-blindness.

Binary SoC data is defined in Chapter 3 with the domain $\mathbb{B} = \{0,1\}$, and each of the correlation metrics has both a domain and codomain of $[0,1]$ [1] ; i.e. all values can be expressed as a percentage bounded between 0% and 100%. A 1D value on the unit interval is easily represented using greyscale with the extreme cases $0 \rightarrow$ `white` and $1 \rightarrow$ `black`, or vice-versa. This appendix contributes a colourspace

[1] This thesis only explores the usefulness of metrics with binary SoC data, $f_i[t] \in \mathbb{B}$, but the definitions also cover the more general case of normalised data, $f_i[t] \in [0,1]$.

which is convenient to represent 2D values of the form $(a,b) \in [0,1]^2$.

This novel colourspace has been developed to allow the viewer to quickly discern the approximate location of a point in 2D space by its colour. Naturally, due to physiological differences between individuals, it is not possible to ensure that all viewers perceive the exact same value. Instead, as with all colourspace schemes, only an imprecise estimation is possible. Imprecise estimations are still very useful, particularly when searching for values in a range, identifying trends, or filtering out uninteresting points. Examples of values for which this colourspace can be used to represent include $\big(\dot{\mathrm{C}}\mathrm{ov}, \dot{\mathrm{D}}\mathrm{ep}\big)$, $\big(\mathbb{E}[f_x], \mathbb{E}[f_y]\big)$, (`bandwidth::%`, `debugEnabled::bool`), or any combination thereof.

Equation (I.1) thru Equation (I.6) define the mapping from $[0,1]^2$ to w-bit Red/-Green/Blue (RGB) colours as depicted in Figure I.1a. Figure I.1b thru Figure I.1d show simple variations on the defining equations.

$$\theta = \left(1 - \frac{\sqrt{a^2+b^2}}{\sqrt{2}}\right)^{\gamma} \tag{I.1}$$

$$\phi = \arctan\left(\frac{a}{b}\right) \tag{I.2}$$

$$\mathrm{color}_{\mathrm{RGB}} = \left\lfloor (2^w - 1)\theta^{(\phi_r,\ \phi_g,\ \phi_b)} \right\rfloor \tag{I.3}$$

$$\phi_r = 1 \tag{I.4}$$

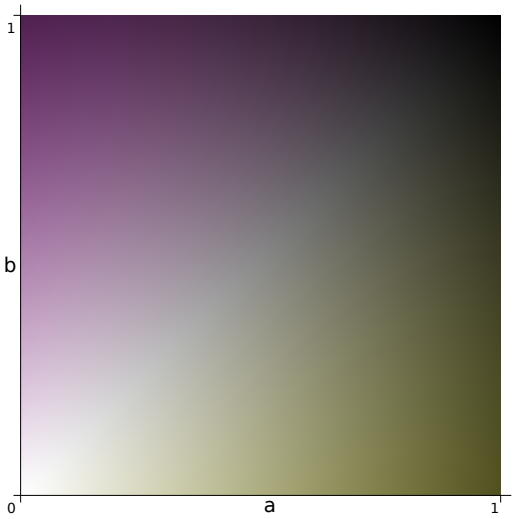
$$\phi_g = 1 + \max\left(0,\ \phi - \frac{\pi}{4}\right) \tag{I.5}$$

$$\phi_b = 1 + \max\left(0,\ \frac{\pi}{4} - \phi\right) \tag{I.6}$$

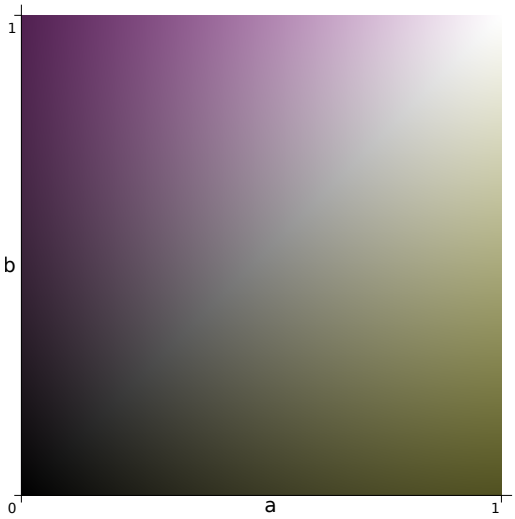
Equation (I.1) calculates the magnitude θ (Euclidean distance from the origin), and Equation (I.2) calculates the angular offset ϕ from the main diagonal. Where $a = b$, then $\phi_r = \phi_g = \phi_b = 1$, meaning the resulting colour is greyscale. Equation (I.3) has a codomain of $[0, 2^w)^3$ with the 3-tuple components representing red, green, and blue pixel brightness. A worked example is given in Figure I.2. Many modern computer applications use $w = 8$ allowing each component to be represented by an unsigned integer $\in [0, 256)$ in one byte.

Covering a printed page in too much ink is undesirable so this colourspace uses less ink for points with small values on both axes. Alternatively, for a dataset containing mostly large values, Figure I.1b shows a variation with an inverted origin, created by removing the "1−" from Equation (I.1). Inverting the origin, i.e. using brighter colours for larger values, may also be suitable for display on screen where a "dark theme" is desired to reduce eye fatigue.

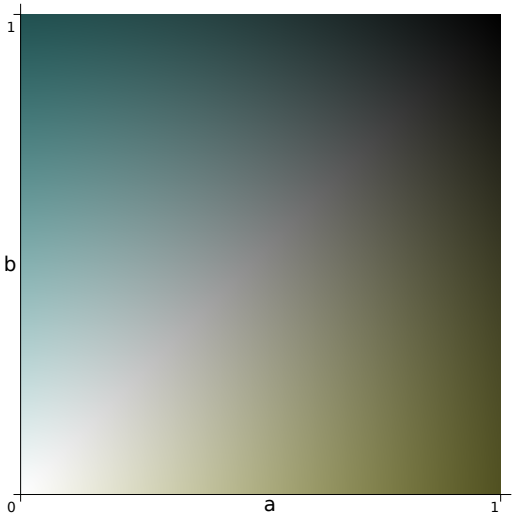
Protanopia, the most common form of colour-blindness affecting up to 10% of Caucasian men, including this author, has been considered in developing this colorspace. To avoid confusion and mis-communication, perceived colours should be similar between protans and those with normal colour vision. Figure I.1c and



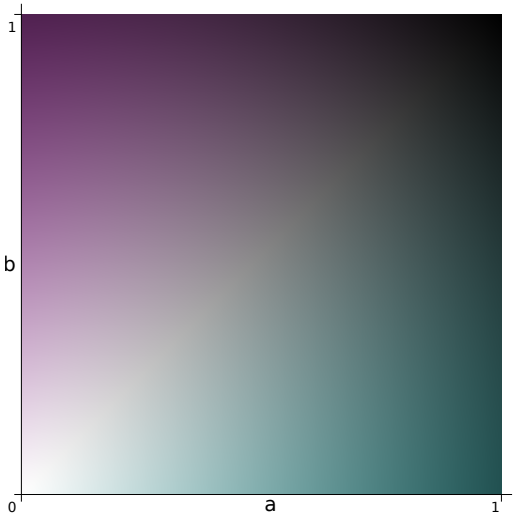
(a) Bounded 2D Colorspace defined by Equation (I.1) thru Equation (I.6).



(b) Inverted origin (where darker means smaller) doesn't work well on printed paper.



(c) Using green instead of red on the magnitude (Equation (I.4)) produces cyan in upper-left corner, which is indistinguishable from grey for propantopes.



(d) Using blue instead of red on the magnitude (Equation (I.4)) produces cyan in lower-right corner, which is indistinguishable from grey for propantopes.

Figure I.1: Plot of the full colourspace in Figure I.1a, i.e. the cross product ($a \times b$). Variations on the defined colourspace are shown alongside for comparison.

Figure I.1d demonstrate variations on the definition which are not suitable for protans because cyan is essentially indistinguishable from grey.

A worked example of mapping a point in bounded 2D space to a colour is given in Figure I.2, and a usage example is given in Figure I.3 where a system designer can visually process the results of many experimental configurations. Distinct colours in each corner (white, green, purple, and black) allow the designer to create goals with visual analogies, e.g. “Greener is better”, thus facilitating intuitive optimisation over two parameters at once.

Aim: Represent the data point ($\texttt{performance} = 70\%, \texttt{bandwidth} = 45\%$) with colour.

Step 1: Calculate magnitude and angular offset. Note that a gamma correction of $\gamma = 1$ is used here for simplicity. Different presentation media should use appropriate gamma correction values.

$$\begin{aligned} \theta &= \left(1 - \frac{\sqrt{0.7^2 + 0.45^2}}{\sqrt{2}}\right) && = 0.41157 \\ \phi &= \arctan\left(\frac{0.7}{0.45}\right) && = 0.99946 \end{aligned}$$

Step 2: Calculate exponents for each colour component.

$$\begin{aligned} \phi_r &= 1 \\ \phi_g &= 1 + \max\left(0,\ 0.99946 - \frac{\pi}{4}\right) && = 1.21406 \\ \phi_b &= 1 + \max\left(0,\ \frac{\pi}{4} - 0.99946\right) && = 1 \end{aligned}$$

Step 3: Let $w = 8$ and calculate colour components.

$$\begin{aligned} \text{color}_\text{R} &= \lfloor 255 \times 0.41157^{1} \rfloor && = 104 \\ \text{color}_\text{G} &= \lfloor 255 \times 0.41157^{1.21406} \rfloor && = 86 \\ \text{color}_\text{B} &= \lfloor 255 \times 0.41157^{1} \rfloor && = 104 \end{aligned}$$

Step 4: Convert to hexadecimal RGB format and display.

$$(104, 86, 104) = (\text{0x68}, \text{0x56}, \text{0x68}) \rightarrow \texttt{\#685668}$$

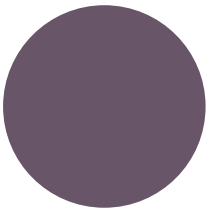
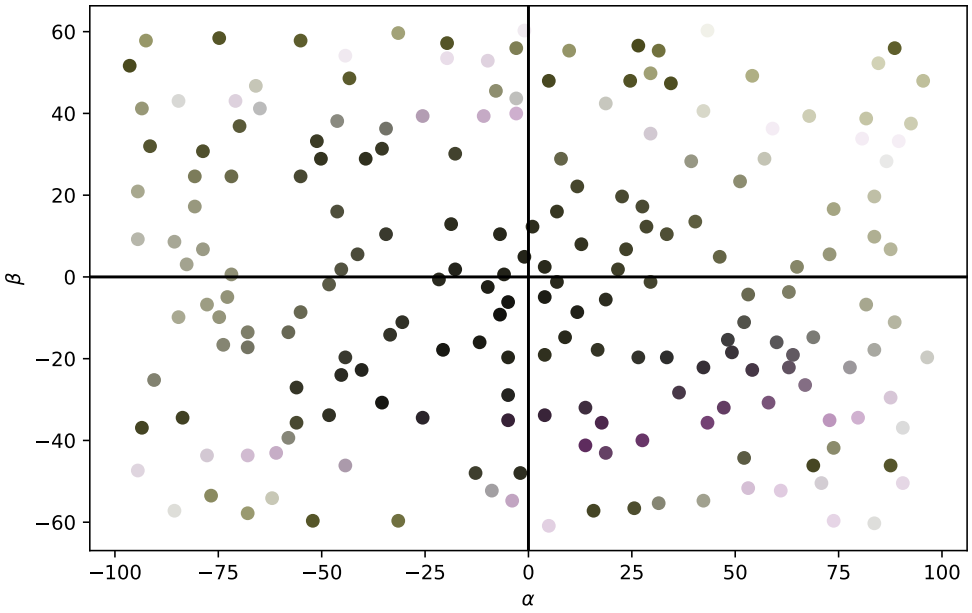
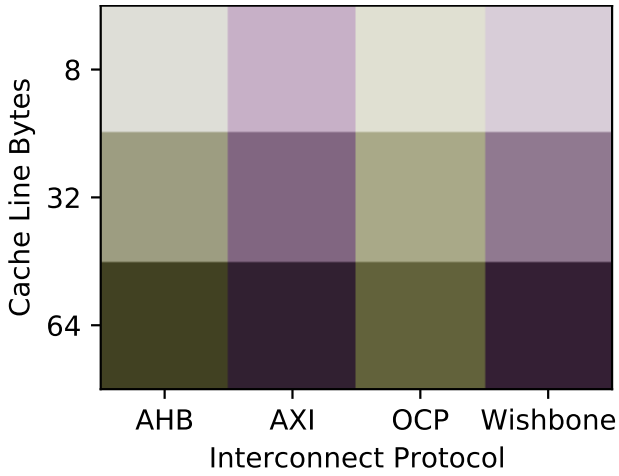


Figure I.2: Worked example of colour calculation.



(a) Variants of algorithm, parameterised by α and β. The sweet-spot (most purple area) appears around $\alpha = 30$, $\beta = -40$.



(b) Variants of SoC configuration with different interconnect protocols and cache-line widths. The sweet-spot (darkest area) appears as either AXI/64 or Wishbone/64.

Figure I.3: Example SoC optimisation problem assisted by use of this colourspace. Colors are used to represent a = performance vs b = bandwidth utilisation over many variants of algorithm and interconnect configuration. Firstly, interconnect configuration is fixed and many variations of the algorithm are tested, where the system designer aims for best performance with least bandwidth utilisation, i.e. purple is best. Secondly, the algorithm parameters are fixed, and various interconnect configurations are tested where the system designer makes the trade-off between performance and cache width to reduce hardware cost, i.e. black is best.

BIBLIOGRAPHY

[1] E. H. Lee, M. M. Krell, A. Tsyplikhin, V. Rege, E. Colak, and K. W. Yeom, "NanoBatch DPSGD: Exploring Differentially Private Learning on ImageNet with Low Batch Sizes on the IPU," *arXiv Computer Science*, Sep 2021. doi: arXiv:2109.12191

[2] P. A. Buitrago, J. Uran, and N. A. Nystrom, "System Integration of Neocortex, a Unique, Scalable AI Platform," in *Practice and Experience in Advanced Research Computing (PEARC)*, no. 37, Jul 2021. doi: 10.1145/3437359.3465604

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, Nov 2016. ISBN 978-1-931971-33-1

[4] M. Morris and Icera Inc, "Data Transmission," Great Britain Patent 091 038.8, 2009, A method of transmitting data from a first module to addressable storage devices in a second module.

[5] *nRF52832 Product Specification*, Nordic Semiconductor), 2021, Section 22 PPI - Programmable Peripheral Interconnect.

[6] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985. ISBN 0-13-153271-5

[7] *UL-000566-TR-4-An Introduction to the UltraSoC Architecture*, UltraSoC Technologies Ltd, Dec 2018.

[8] D. McEwan and J. Nunez-Yanez, "Relationship estimation metrics in binary SoC data," in *Proceedings of 5th International Conference on Machine Learning, Optimization, and Data Science (LOD2019)*, Sep 2019. doi: 10.1007/978-3-030-37599-7_11

[9] E. T. Hvannberg, E. L.-C. Law, and M. K. Larusdottir, "Heuristic Evaluation: Comparing Ways of Finding and Reporting Usability Problems," *Interacting With Computers*, vol. 19, pp. 225–240, 2007. doi: 10.1016/j.intcom.2006.10.001

[10] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.

[11] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *From Data Mining to Knowledge Discovery: An Overview*. American Association for Artificial Intelligence, 1996, pp. 1–34. ISBN 0262560976

[12] Z. W. Ellerby and R. J. Tunney, "The Effects of Heuristics and Apophenia on Probabilistic Choice," *Advances in Cognitive Psychology*, vol. 13, no. 4, pp. 280–294, 2017. doi: 10.5709/acp-0228-9

[13] R. J. Hilderman and H. J. Hamilton, "Knowledge Discovery and Interestingness Measures: A Survey," University of Regina, Tech. Rep., 1999. doi: 10.1.1.588.4570

[14] J. F. Roddick and S. Rice, "What's Interesting About Cricket? - On Thresholds and Anticipation in Discovered Rules," *SIGKDD Explorations*, vol. 3, pp. 1–5, 2001. doi: 10.1145/507533.507535

[15] J. Y. Chen, "Transformation of VLSI Technologies, Systems and Applications; The Rise of Foundry and its Ecosystem," in *International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, 2013, pp. 1–2. doi: 10.1109/VLSI-TSA.2013.6545604

[16] W.-S. Liao and P.-A. Hsiung, "FVP: a formal verification platform for SoC," in *IEEE International [Systems-on-Chip] SoC Conference Proceedings*, 2003, pp. 21–24. doi: 10.1109/SOC.2003.1241454

[17] C. Cummings, "Clock Domain Crossing (CDC) Design and Verification Techniques using SystemVerilog," *SNUG2008*, 2008. [Online]. Available: http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

[18] L. Cao, "In-depth behavior understanding and use: The behavior informatics approach," *Information Sciences*, vol. 180, no. 17, pp. 3067–3085, 2010. doi: https://doi.org/10.1016/j.ins.2010.03.025

[19] C. Wang, L. Cao, and C.-H. Chi, "Formalization and Verification of Group Behavior Interactions," *IEEE Transactions on Systems, Man,*

and Cybernetics: Systems, vol. 45, no. 8, pp. 1109–1124, 2015. doi: 10.1109/TSMC.2015.2399862

[20] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Pearson, 2013. ISBN 978-93-325-3503-9

[21] B. Sklar, *Digital Communications Fundamentals and Applications*, 2nd ed. Pearson Education, 2016. ISBN 978-0-13-4724058

[22] K. Pearson, "Note on Regression and Inheritance in the Case of Two Parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240–242, Jun 1895. doi: 10.1098/rspl.1895.0041

[23] M. C. Abounaima, F. Z. el Mazouri, L. Lamrini, N. Nfissi, N. el Makhfi, and M. Ouzarf, "The Pearson Correlation Coefficient Applied to Compare Multi-Criteria Methods: Case the Ranking Problematic," in *International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, 2020, pp. 1–6. doi: 10.1109/IRASET48871.2020.9092242

[24] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2002. ISBN 0521642981

[25] J. L. Rodgers and W. A. Nicewander, "Thirteen Ways to Look at the Correlation Coefficient," *The American Statistitian*, vol. 42, no. 1, pp. 59–66, Feb 1988. doi: 10.1080/00031305.1988.10475524

[26] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[27] D. Rossi, I. Loi, A. Pullini, C. Muller, A. Burg, F. Conti, L. Benini, and P. Flatresse, "A Self-Aware Architecture for PVT Compensation and Power Nap in Near Threshold Processors," *IEEE Design And Test*, vol. 34, no. 6, pp. 46–53, 2017. doi: 10.1109/MDAT.2017.2750907

[28] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-ready 1.7GHz 64bit RISC-V Core in 22nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019. doi: 10.1109/TVLSI.2019.2926114

[29] A. Waterman and K. Asanovic, *The RISC-V Instruction Set Manual Volume I: Unpriviliged ISA*, 20191213th ed., RISC-V Foundation, Dec 2019. [Online]. Available: https://riscv.org/specifications/

[30] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: an Open Source Framework from Verilog to Bitstream for Commercial FPGAs," *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, no. 27, pp. 1–4, Apr 2019.

[31] A. Torralba and A. A. Efros, "Unbiased Look at Dataset Bias," in *CVPR 2011*, 2011, pp. 1521–1528. doi: 10.1109/CVPR.2011.5995347

[32] L. Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories," in *Conference on Computer Vision and Pattern Recognition Workshop*, 2004, pp. 178–178. doi: 10.1109/CVPR.2004.383

[33] ARM Limited, *AMBA AXI and ACE Protocol Specification*, 2011. [Online]. Available: http://arm.com

[34] Accellera, *Open Core Protocol Specification 3.0*, 2013. [Online]. Available: http://www.eda.org/downloads/standards/ocp

[35] N. Frohlich, B. M. Riess, U. A. Wever, and Q. Zheng, "A New Approach for Parallel Simulation of VLSI Circuits on a Transistor Level," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, no. 6, pp. 601–613, 1998. doi: 10.1109/81.678468

[36] J. Djigbenou, T. V. Nguyen, C. W. Ren, and D. S. Ha, "Development of TSMC 0.25 µm standard cell library," in *IEEE Proceedings of International Conference on Sensing, Communication and Networking*, Jun 2007, pp. 566–568. doi: 10.1109/SECON.2007.342966

[37] *IEEE 1800-2017 Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language*, Design Automation Standards Commitee of the IEEE Computer Society. doi: 10.1109/IEEESTD.2018.8299595

[38] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, and J. Koenig, "The Rocket Chip Generator," Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[39] *IEEE 1364-1995 Standard Hardware Description Language Based on the Verilog Hardware Description Language*, IEEE Computer Society. doi: 10.1109/IEEESTD.1996.81542

[40] S. Lagraa, "New MP-SoC Profiling Tools Based on Data Mining Techniques," Ph.D. dissertation, L'Université de Grenoble, 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01548913

[41] *UL-001174-TR-3B-Static Instrumentation User Guide*, UltraSoC Technologies Ltd, Dec 2018.

[42] E. Edgar and T. Newman, *RISC-V External Debug Support*, 1st ed., SiFive Inc, 2021. [Online]. Available: https://riscv.org/specifications/

[43] *IEEE The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface*, IEEE Industry Standards and Technology Organization, 2012.

[44] G. Zellweger, D. Lin, and T. Roscoe, "So many performance events, so little time," in *APSys16 Proceedings of the 7th ACM SIGOPS Workshop on Systems*, Aug 2016. doi: 10.1145/2967360.2967375. [Online]. Available: https://people.inf.ethz.ch/troscoe/pubs/zellweger_apsys_2016.pdf

[45] *ChipScope Pro Software and Cores User Guide*, Xilinx), 2012, UG029.

[46] *Quartus Programmable Logic Development Software SignalTap User's Guide*, Altera Corporation (Intel Corporation), 1999, P25-04733-01.

[47] *UL-000247-TR-19I-Bus Monitor User Guide*, UltraSoC Technologies Ltd, Dec 2018.

[48] S. K. Roy, "Top Level SOC Interconnectivity Verification using Formal Techniques," in *IEEE Eighth International Workshop on Microprocessor Test and Verification*, 2008, pp. 63–70. doi: 10.1109/MTV.2007.22

[49] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and Trends in Modern SoC Design Verification," *IEEE Design And Test*, vol. 34, no. 5, pp. 7–22, 2017. doi: 10.1109/MDAT.2017.2735383

[50] R. S. Mitra, "Strategies for Mainstream Usage of Formal Verification," in *Proceedings of the 45th Annual Design Automation Conference*. Association for Computing Machinery, 2008, pp. 800–805. doi: 10.1145/1391469.1391674. ISBN 9781605581156

[51] D. Gentner and K. J. Holyoak, "Reasoning and Learning by Analogy," *American Psychologist*, vol. 52, no. 1, pp. 32–34, Jan 1997. doi: 10.1037/0003-066x.52.1.32

[52] D. Gentner and A. B. Markman, "Structure Mapping in Analogy and Similarity," *American Psychologist*, vol. 52, no. 1, pp. 45–56, Jan 1997. doi: 10.1037//0003-066x.52.1.45

[53] J. Nielsen, *Usability Engineering*. Academic Press, 1993. ISBN 978-0125184069

[54] ——, "Enhancing The Exploratory Power of Usability Heuristics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Apr 1994, pp. 152–158. doi: 10.1145/191666.191729

[55] B. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," in *Proceedings IEEE Symposium on Visual Languages*, 1996, pp. 336–343. doi: 10.1109/VL.1996.545307

[56] J. Gerhardt-Powals, "Cognitive Engineering Principles for Enhancing Human-Computer Performance," *International Journal of Human-Computer Interaction*, vol. 8, no. 2, pp. 189–211, 1996. doi: 10.1080/10447319609526147

[57] R. Amar and J. Stasko, "A Knowledge Task-Based Framework for Design and Evaluation of Information Visualizations," in *IEEE Symposium on Information Visualization*, 2004, pp. 143–150. doi: 10.1109/INFVIS.2004.10

[58] T. Zuk, L. Schlesier, P. Neumann, M. S. Hancock, and S. Carpendale, "Heuristics for Information Visualization Evaluation," in *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, May 2006, pp. 1–6. doi: 10.1145/1168149.1168162. ISBN 1595935622

[59] A. Tarell, A. Fruhling, R. Borgo, C. Forsell, G. Grinstein, and J. Scholtz, "Toward Visualization-Specific Heuristic Evaluation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 9, pp. 110–125, Nov 2014. doi: 10.1145/2669557.2669580

[60] C. Forsell and J. Johansson, "An Heuristic Set for Evaluation in Information Visualization," in *ACM Proceedings of the International Conference on Advanced Visual Interfaces*, 2010, pp. 199–206. doi: 10.1145/1842993.1843029. ISBN 9781450300766

[61] T. Bybell, *GTKWave 3.3 Wave Analyzer User's Guide*, GTKWave project, Jul 2018. [Online]. Available: http://gtkwave.sourceforge.net/gtkwave.pdf

[62] Synopsys, "VCS." [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html

[63] J. Gautier, P.-A. Davoine, and C. Cunty, "Helical Time Representation to Visualize Return-Periods of Spatio-Temporal Events," in *19th AGILE*

International Conference on Geographic Information Science, no. hal-01609156, Jun 2016.

[64] D. Loudon and M. H. Granat, "Visualization of Sedentary Behaviour Using an Event Based Approach," *Measurement in Physical Education and Exercise Science*, vol. 19, pp. 148–157, 2015. doi: 10.1080/1091367X.2015.1048342

[65] M. Desnoyers and EfficiOS Inc, "Common Trace Format (CTF) Specification (v1.8.3)," https://diamon.org/ctf/v1.8.3/, May 2020.

[66] Eclipse Foundation, "Trace compass." [Online]. Available: https://www.eclipse.org/tracecompass/

[67] M. Friendly, "Corrgrams: Exploratory Displays for Correlation Matrices," *The American Statistician*, vol. 56, no. 4, pp. 316–324, Aug 2002. doi: 10.1198/000313002533

[68] M. Friendly and E. Kwan, "Effect Ordering for Data Displays," *Computational Statistics and Data Analysis*, vol. 43, pp. 509–530, 2003. doi: 10.1016/S0167-9473(02)00290-6

[69] S. Zapf and C. Krausharr, "Solar Correlation Map: A New Visualization to Beautifully Explore Correlations," Jan 2017. [Online]. Available: https://github.com/Zapf-Consulting/solar-correlation-map

[70] S.-T. Yeh and GlaxoSmithKline, "Exploratory Visualization of Correlation Matrices," in *NESUG 2007*, 2007.

[71] E. R. Gansner and S. C. North, "An Open Graph Visualization System and its Applications to Software Engineering," *Software - Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[72] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, *Graphviz and Dynagraph - Static and Dynamic Graph Drawing Tools*. Springer, 2004, pp. 127–148. doi: 10.1007/978-3-642-18638-7_6. ISBN 978-3-642-18638-7

[73] E. R. Gansner and Y. Koren, "Improved Circular Layouts," *Graph Drawing*, vol. 30, pp. 386–398, 2007. doi: 10.1007/978-3-540-70904-6_37

[74] G. V. Loo, *BCM2836 ARM Quad-A7*, 2014. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/QA7_rev3.4.pdf

[75] S.-S. Choi, S.-H. Cha, and C. C. Tappert, "A Survey of Binary Similarity and Distance Measures," *Systems, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

[76] G. Maggiora, M. Vogt, D. Stumpfe, and J. Bajorath, "Molecular Similarity in Medicinal Chemistry," *Journal of Medicinal Chemistry*, vol. 57, pp. 3186–3204, Oct 2013. doi: 10.1021/jm401411z

[77] M. Gheradi and P. Rotondo, "Measuring Logic Complexity can Guide Pattern Discovery in Empirical Systems," *Complexity*, vol. 21, no. S2, pp. 397–408, Sep 2018. doi: 10.1002/cplx.21819

[78] D. Lo, S.-C. Khoo, and C. Liu, "Mining past-time temporal rules from execution traces," *ACM Workshop On Dynamic Analysis*, pp. 50–56, Jul 2008. doi: 10.1145/1401827.1401838

[79] G. Tkacik and W. Bialek, "Information Processing in Living Systems," *Annual Review of Condensed Matter Physics*, vol. 7, pp. 89–117, Dec 2014. doi: 10.1146/annurev-conmatphys-031214-014803

[80] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[81] P. Jaccard, "The Distribution of the Flora in the Alpine Zone," *The New Phytologist*, vol. 11, no. 2, pp. 37–50, Feb 1919. doi: 10.1111/j.1469-8137.1912.tb05611.x

[82] D. J. Rogers and T. T. Tanimoto, "A Computer Program for Classifying Plants," *Science*, vol. 132, no. 3434, pp. 1115–1118, Oct 1960. doi: 10.1126/science.132.3434.1115

[83] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," *Science*, vol. 315, pp. 972–976, 1950. doi: 10.1126/science.1136800

[84] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *International Conference on Learning Representations*, Jan 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[85] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic Cosine Similarity," in *Proceedings of the 7th ICAST Seoul*, Oct 2012.

[86] J. P. Mower, "PREP-Mt: Predictive RNA Editor for Plant Mitochondrial Genes," *BMC Bioinformatics*, vol. 6, no. 96, Apr 2005. doi: 10.1186/1471-2105-6-96

[87] B. C. Csaji, "Approximation with artificial neural networks," Ph.D. dissertation, Faculty of Sciences, Eotvos Lorand University, 2001. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf

[88] D. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *The 3rd International Conference on Learning Representations*, 2015. doi: arXiv/1412.6980

[89] L. Lu, Y. Shin, Y. Su, and G. Karniadakis, "Dying ReLU and Initialization: Theory and Numerical Examples," *Communications in Computational Physics*, vol. 28, no. 5, pp. 1671–1706, 11 2020. doi: 10.4208/cicp.OA-2020-0165

[90] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, *Universal Serial Bus Specification 2.0*, Apr 2000.

[91] D. Lyon, "The Discrete Fourier Transform, Part 6: Cross-Correlation," *Journal of Object Technology*, vol. 9, no. 2, pp. 17–22, Apr 2010. doi: 10.5381/jot.2010.9.2.c2

[92] *UL-000238-TR-27C-Status Monitor User Guide*, UltraSoC Technologies Ltd, Dec 2020.

[93] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge University Press, 1998. ISBN 0-521-59292-5

[94] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x

[95] D. P. Mitchel and A. N. Netravali, "Reconstruction Filters in Computer-Graphics," in *15th Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, 1988, pp. 221–228. doi: 10.1145/54852.378514

[96] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing Principles, Algorithms and Applications*, 4th ed. Pearson, 2007. ISBN 978-0131873742

[97] K. Buchanan, T. Adeyemi, C. Flores-Molina, S. Wheeland, and D. Overturf, "Sidelobe Behavior and Bandwidth Characteristics of Distributed Antenna Arrays," in *2018 United States National Committee of URSI National Radio Science Meeting*, Jan 2018, pp. 1–2. ISBN 978-1-5386-5031-8

[98] G. Marsaglia and W. W. Tsang, "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software*, vol. 5, no. i08, 2000. doi: 10.18637/jss.v005.i08

[99] R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed. Dover, 1973. ISBN 978-0-486-65241-6

[100] A. H. Nuttall, "Some Windows with Very Good Sidelobe Behavior," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, no. 1, pp. 84–91, Feb 1981. doi: 10.1109/tassp.1981.1163506

[101] D. R. Bull, *Communicating Pictures, A Course in Image and Video Coding*. Academic Press, 2014. ISBN 978-0-12-405906-1

[102] L. C. Loschky and G. S. Wolverton, "How Late Can You Update Gaze-contingent Multi-resolutional Displays Without Detection," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 3(4), no. 7, pp. 1–10, Dec 2007. doi: 10.1145/1314303.1314310

[103] *ARM Architecture Registers Armv8*, ARM Ltd, 2020.

[104] L. Semiconductor, *Lattice Semiconductor iCE40 LP/HX Family Data Sheet*, Lattice Semiconductor Corporation, Sep 2018.

[105] Xilinx, *Xilinx 7 Series FPGAs Data Sheet DS180*, Xilinx Inc, Sep 2020.

[106] I. Corporation, "Intel tick tock model," https://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html.

[107] K. Vorwerk, A. Kennings, and J. W. Greene, "Improving Simulated Annealing-Based FPGA Placement With Directed Moves," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 179–192, Jan 2009. doi: 10.1109/TCAD.2008.2009167

[108] R. Shi, L. Ma, and T. Watanabe, "Efficient Simulated Annealing-Based Placement Algorithm for Island Style FPGAs," *International Journal of Machine Learning and Computing*, vol. 8, no. 6, pp. 542–548, Dec 2018. doi: 10.18178/ijmlc.2018.8.6.743

[109] J. A. Espejo, L. Entrena, E. S. Millan, and E. Olias, "Logic Restructuring for MUX-Based FPGAs," in *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, 1999, pp. 161–168. doi: 10.1109/EURMIC.1999.794462

[110] X. Shen, X. Wu, J. Lu, and L. Qin, "Hybrid DPWM with Analog Delay Locked Loop," *Lecture Notes in Engineering and Computer Science*, vol. 2181, pp. 1279–1281, 2010.

[111] T.-L. Chu, S.-H. Yu, and C.-S. Hwang, "High-Accuracy Programmable Timing Generator with Wide-Range Tuning Capability," *Advanced VLSI Design Methodologies for Emerging Industrial Multimedia and Communication Applications*, no. 803616, 2013. doi: 10.1155/2013/803616

[112] V. Yousefzadeh, T. Takayama, and D. Maksimovi, "Hybrid DPWM with Digital Delay-Locked Loop," in *2006 IEEE Workshops on Computers in Power Electronics*, 2006, pp. 142–148. doi: 10.1109/COMPEL.2006.305666

[113] M. Kumm, H. Klingbeil, and P. Zipf, "An FPGA-Based Linear All-Digital Phase-Locked Loop," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 9, pp. 2487–2497, 2010. doi: 10.1109/TCSI.2010.2046237

[114] D. Blackman and S. Vigna, "Scrambled Linear Pseudorandom Number Generators," *ACM Transactions on Mathematical Software*, 2019. [Online]. Available: http://vigna.di.unimi.it/ftp/papers/ScrambledLinear.pdf

[115] G. F. Lawler and V. Limic, *Random Walk: A Modern Introduction*, 1st ed. Cambridge University Press, Jun 2010. doi: 10.1017/CBO9780511750854. ISBN 978-0511750854

[116] A. E. Hughes, R. V. Southwell, R. V. Gilchrist, and D. J. Tolhurst, "Quantifying Peripheral and Foveal Perceived Differences in Natural Image Patches to Predict Visual Search Performance," *Journal of Vision*, vol. 16(10), no. 10, pp. 1–17, Aug 2016. doi: 10.1167/16.10.18

[117] *IEEE 754-2019 Standard for Floating-Point Arithmetic*, IEEE Microprocessor Standards Committee. doi: 10.1109/IEEESTD.2019.8766229

[118] D. Monnaiux, "The Pitfalls of Verifying Floating-Point Computations," *ACM Transactions on Programming Languages and Systems*, vol. 30, no. 3, pp. 1–41, May 2008. doi: 10.1145/1353445.1353446

[119] J. L. Gustafson and I. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, Jun 2017. doi: 10.14529/jsfi170206

[120] V. Gohil, W. Sumit, M. Joycee, and A. Manu, "Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, p. 1, 2021. doi: 10.1109/TCSII.2021.3072217

[121] V. Mansur and R. B. Shettar, "Design and Development of Interval Arithmetic Library Using Q Format Data Representation: A FPGA Approach," in *2015 International Conference on Computing, Communication and Security (ICCCS)*, 2015, pp. 1–7. doi: 10.1109/CCCS.2015.7374176

[122] E. L. Oberstar, "Fixed-Point Representation and Fractional Math," *Oberstar Consulting*, Aug 2007.

[123] J. E. Volder, "The Birth of CORDIC," *Journal of VLSI Signal Processing*, vol. 25, pp. 101–105, Jun 2000. doi: 10.1023/A:1008110704586

[124] G. E. Walters, "Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs," *MDPI Computers*, vol. 5, no. 20, Sep 2016. doi: 10.3390/computers5040020. [Online]. Available: https://www.mdpi.com/2073-431X/5/4/20

[125] M. Langhammer and G. Baeckler, "High Density and Performance Multiplication for FPGA," in *IEEE 25th Symposium on Computer Arithmetic (ARITH)*, 2018, pp. 5–12. doi: 10.1109/ARITH.2018.8464695

[126] D. Yeager, D. Chiu, and G. Lemieux, "Congestion Estimation and Localization in Fpgas: A Visual Tool for Interconnect Prediction," in *ACM International Workshop on System Level Interconnect Prediction (SLIP07)*, 2007, pp. 33–40. doi: 10.1145/1231956.1231963. ISBN 9781595936226

[127] C. Yu and Z. Zhang, "Painting on Placement: Forecasting Routing Congestion using Conditional Generative Adversarial Nets," in *ACM Proceedings of the 56th Annual Design Automation Conference (DAC19)*, 2019. doi: 10.1145/3316781.3317876. ISBN 978-1-4503-6725-7

[128] C. Seed, "Arachne-PnR," 2015. [Online]. Available: https://github.com/YosysHQ/arachne-pnr

[129] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton: An Open Source Manycore Research Framework," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 217–232. doi: 10.1145/2872362.2872414. ISBN 978-1-4503-4091-5

[130] K. Lim, J. Balkind, and D. Wentzlaff, "JuxtaPiton: Enabling Heterogeneous-ISA Research with RISC-V and SPARC FPGA Soft-cores," *Princeton University*, Nov 2018. doi: arXiv/1811.08091

[131] D. Wentzlaff, *OpenPiton Microarchitecture Specification*, Princeton University, 2016. [Online]. Available: https://parallel.princeton.edu/openpiton/docs/micro_arch.pdf

[132] D. McEwan, M. Hlond, and J. Nunez-Yanez, "Visualizations for Understanding SoC Behaviour," in *Proceedings of 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME2019)*, Jul 2019. doi: 10.1109/PRIME.2019.8787837

[133] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," in *Proceedings of the 24th International Conference on Software Engineering*, May 2002, pp. 467–477. doi: 10.1145/581339.581397

[134] B. S. Santos, B. Q. Ferreira, and P. Dias, "Using Heuristic Evaluation to Foster Visualization Analysis and Design Skills," *IEEE Computer Graphics and Applications*, vol. 36, no. 1, pp. 86–90, 2016. doi: 10.1109/MCG.2016.7

[135] B. Shneiderman, "Design Lessons From AI's Two Grand Goals: Human Emulation and Useful Applications," *IEEE Transactions on Technology and Society*, vol. 1, no. 2, pp. 73–82, 2020. doi: 10.1109/TTS.2020.2992669

[136] D. Park, "Visual Security: 9-block IP Identification," http://www.docuverse.com/blog/donpark/2007/01/18/visual-security-9-block-ip-identification, Jan 2007.

[137] L. Sirovich and M. Kirkby, "Low-Dimensional Procedure for the Characterization of Human Faces," *Journal of the Optical Society of America*, vol. 4, no. 3, pp. 519–524, Mar 1987. doi: 10.1364/josaa.4.000519

[138] *UL-00231-TC-F-Taygete Prototype*, UltraSoC Technologies Ltd, Dec 2018.

[139] G. Louw, "Tinn the tiny neural network library," https://github.com/glouw/tinn.

[140] Semeion Research Center of Sciences of Communication and Tattile Srl, "Semeion handwritten digit data set." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/semeion+handwritten+digit

[141] M. Buscema, "MetaNet: The theory of independent judges," *Substance Use and Misuse*, vol. 33, no. 2, pp. 439–461, Feb 1998. doi: 10.3109/10826089809115875

[142] P. Rodgers, G. Stapleton, and P. Chapman, "Visualizing sets with linear diagrams," *ACM Transactions on Computer-Human Interaction*, vol. 22(6), no. 27, pp. 1–39, Sep 2015. doi: 10.1145/2810012

[143] *IEEE 1076-2019 Standard VHDL Language Reference Manual*, IEEE Design Automation Standards Commitee. doi: 10.1109/IEEESTD.2019.8938196

[144] N. Drakos and R. Moore, *Berkeley Logic Interchange Format*, University of California Berkeley, 1992. [Online]. Available: http://www.cs.columbia.edu/~cs6861/sis/blif/index.html

[145] H. Kahn, D. Systems, M. Graphics, Motorola, N. Semiconductor, Tektronix, T. Instruments, U. of California Berkeley, and U. of Manchester, *Electronic Design Interchange Format*, EDIF Standards Organization, 1983.

[146] D. Shah, "NextPnR," 2018. [Online]. Available: https://github.com/YosysHQ/nextpnr

[147] G. Marsaglia, "Xorshift RNGs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003. doi: 10.18637/jss.v008.i14. [Online]. Available: https://www.jstatsoft.org/v008/i14

[148] R. Brent, "Note on Marsaglia's Xorshift Random Number Generators," *Journal of Statistical Software*, vol. 11, no. 5, pp. 1–5, Aug 2004. doi: 10.18637/jss.v011.i05

[149] P. L'Ecuyer and R. Simard, "TestU01: A C Library for Empirical Testing of Random Number Generators," *ACM Transactions on Mathematical Software*, vol. 33, no. 4, p. 22, 2007. doi: 10.1145/1268776.1268777. [Online]. Available: http://simul.iro.umontreal.ca/testu01/tu01.html

[150] S. Vigna, "An Experimental Exploration of Marsaglia's Xorshift Generators, Scrambled," *ACM Transactions on Mathematical Software*, vol. 42, no. 4, 2016. [Online]. Available: https://arxiv.org/abs/1402.6246

[151] ——, "Further Scramblings of Marsaglia's Xorshift Generators," *Journal of Computational and Applied Mathematics*, vol. 315, pp. 175–181, 2016. doi: 10.1016/j.cam.2016.11.006

[152] D. Williams, L. Griffiths, and L. Valenty, "tinyfpga_bx_usbserial," 2019. [Online]. Available: https://github.com/davidthings/tinyfpga_bx_usbserial

[153] J.-P. Charras, "KiCad EDA," 1992. [Online]. Available: https://www.kicad.org/

[154] S. Ishihara, *Tests for Colour-Blindness*. Kanehara Shuppan Co, 1972.

[155] A. D. Logvinenko, "The Geometric Structure of Color," *Journal of Vision*, vol. 15, no. 1, pp. 1–9, January 2015. doi: 10.1167/15.1.16

GLOSSARY

AXI Advanced/ARM eXtensible Interface. 10, 13, 56, 91, 99, 122, 124, 132, 180

ASIC Application Specific Integrated Circuit. 15, 53, 55, 61, 141, 167

CDF Cumulative Density Function. 175

CMF Cumulative Mass Function. 176

CPU Central Processing Unit. 1, 2, 14, 47, 61, 84, 89, 91, 97, 102, 113, 127, 168

CSV Comma Separated Values. 19

CTF Common Trace Format. 21

DFF D-Type Flip-Flop. 13, 56, 60, 64, 78, 84, 97, 141

DSP Digital Signal Processing. 57, 167

DVFS Dynamic Voltage and Frequency Scaling. 167

FFNN Feed-Forward Neural Network. 43, 45, 46, 47, 48, 47, 51, 92, 94, 95, 127, 134, 137

flit Flow Control Digit. 179

FIFO First In First Out. 179, 180

$f_{\mathbf{max}}$ maximum operating clock frequency. 79, 80, 84, 89, 91, 168, 169, 170, 172, 173, 174

FSM Finite State Machine. 77, 78, 180

FOSS Free Open Source Software. 12, 127, 168

FPGA Field Programmable Gate Array. 12, 14, 15, 53, 55, 59, 60, 61, 63, 64, 66, 67, 68, 71, 77, 78, 79, 80, 85, 89, 91, 95, 122, 127, 134, 141, 167, 169, 170, 172, 173, 185

GPIO General Purpose Input Output. 14, 77, 85, 91, 95, 185

HCI Human-Computer Interaction. 16, 98, 99

HDL Hardware Description Language. 142

IP Intellectual Property. 54

ISA Instruction Set Architecture. 12

JTAG Joint Test Action Group. 15

KDD Knowledge Discovery in Databases. 7, 9

KDE Kernel Density Estimation. 34, 39, 143

ML Machine Learning. 43

Multi-PnR Multiple Place-and-Route. 168, 169, 170, 172, 173

MSB Most Significant Bit. 75

NoC Network-on-Chip. 22, 85, 84

LED Light Emitting Diode. 62, 63, 77, 85

LFSR Linear Feedback Shift Register. 172

LHA Left Hand Associative. 34, 36

LPF Low-Pass Filter. 77, 84, 85

LUT Look Up Table. 60, 79, 89, 167, 170, 172, 173, 174

NN Neural Network. 29, 30, 43, 45, 134

OCP Open Core Protocol. 13, 91

OS Operating System. 76, 84, 85, 87

PCB Printed Circuit Board. 185

PDF Probability Density Function. 34, 36, 39, 65, 67, 68, 133, 143, 176

PMF Probability Mass Function. 36, 67, 79, 168, 169, 170, 174, 176

DLL Delay-Locked Loop. 66

PLL Phase-Locked Loop. 66

PnR Place and Route. 59, 61, 79, 80, 84, 167, 168, 169, 170, 172, 173, 174

PRNG Pseudo-Random Number Generator. 67, 172, 173, 175

RAM Random Access Memory. 179

ReLU Rectified Linear Unit. 77

RGB Red/Green/Blue. 190

RTL Register Transfer Language. 13, 15, 53, 54, 55, 59, 60, 61, 62, 63, 64, 70, 72, 78, 80, 84, 91, 94, 95, 94, 97, 135, 141, 167, 168, 185

SMP Symmetric Multi-Processor. 134

SoC System-on-Chip. i, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 20, 22, 23, 25, 27, 28, 29, 30, 31, 32, 33, 34, 36, 39, 41, 42, 43, 45, 51, 52, 53, 54, 55, 56, 57, 56, 59, 61, 62, 69, 77, 78, 84, 85, 87, 91, 94, 95, 97, 98, 99, 100, 101, 106, 112, 114, 116, 120, 121, 122, 127, 132, 133, 134, 135, 136, 137, 141, 167, 179, 189, 192

SVD Singular Value Decomposition. 22, 120

SVM Support Vector Machine. 12

URL Uniform Resource Locator. 100, 108, 111, 113, 114, 115, 116, 118

USB Universal Serial Bus. 54, 62, 76, 185

USB-FS USB Full Speed (12Mb/s). 55, 62, 79, 84, 85, 91, 169, 170, 172, 173, 185

USB-HS USB High Speed (480Mb/s). 91

VCD Value Change Dump. 12, 13, 19, 100

TP True-Positive. 36, 37, 38

TN True-Negative. 36, 37, 38

FP False-Positive. 36, 37

FN False-Negative. 36, 37

TPR True Positive Rate (Sensitivity). 37, 39, 143

TNR True Negative Rate (Specificity). 37, 39, 143

PPV Positive Predictive Value (Precision). 38, 39, 143

NPV Negative Predictive Value. 38, 39, 143

ACC Accuracy. 38, 39, 143

BACC Balanced Accuracy. 38, 39, 143

MCC Matthews Correlation Coefficient. 38, 39, 143

BMI Book-Maker's Informedness. 38, 39, 45, 47, 51, 143