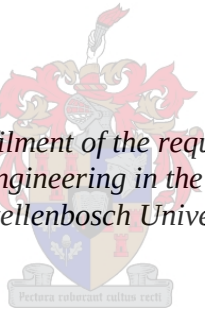
AUTOMATED PARAMETER EXTRACTION FOR SINGLE FLUX QUANTUM INTEGRATED CIRCUITS WITH LVS

by
Rebecca Mimi Catherina Roberts



Thesis presented in fulfilment of the requirements for the degree of Master of Science in Engineering in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof C. J. Fourie

March 2015

Stellenbosch University https://scholar.sun.ac.za

Declaration of Own Work

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Signed: …Rebecca..Roberts……

Copyright © 2015 Stellenbosch University
All rights reserved

Stellenbosch University https://scholar.sun.ac.za

Abstract

Thorough layout verification of superconductor integrated circuits goes beyond design rule checking and parameter value extraction. The former is used to verify adherence to process design rules, and the latter to determine the element values of components such as inductors and resistors and Josephson junction critical currents. Still, neither gives much warning against subtle layout errors that could result in unintended parasitic elements, or a circuit that does not reflect the original circuit topology.

A specialized implementation for Cadence Virtuoso allows layout-versus-schematic verification, but it is limited both to commercial software and in terms of its usefulness. Parameter extraction software such as InductEx is used to extract the component element values of a circuit from its layout if the circuit topology is provided as a netlist, which is mostly created by the designer. However, the element values are extracted for the supplied topology, even if a layout mistake such as creating a connection to the wrong node or a mistake in the netlist results in a model mismatch. After a failed verification, further diagnosis is required to determine whether the error is indeed in the layout or in the input topology - prolonging the verification process significantly.

Here we present a free-standing layout-versus-schematic verification toolkit for superconductive integrated circuits, and discuss its implementation after systematically considering the algorithms at its core. We demonstrate results of the layout-versus-schematic verification and how the layout-versus-schematic toolkit is used as a whole in conjunction with InductEx to perform automated parameter extraction for cell-level layout verification.

The current version of this toolkit provides the user with three stand-alone tools that are best used in conjunction with InductEx: A GDSII file flattener, a layout-to-schematic netlist extractor (with the option of viewing a pictorial reconstruction of the netlist and schematic) and a netlist comparison tool by which the user can determine whether a layout agrees with an input schematic.

We conclude that the netlist comparison and viewing tool provides a valuable method for expediting the layout verification process, making it more efficient and minimizing the chances of mistakes. In its current form the layout-to schematic tool is still limited in that it cannot yet fully support circuits with mutual coupling.

Although many improvements can still be made to this toolkit, the implemented version of these tools can already provide great benefit to Rapid Single Flux quantum (RSFQ) cell designers.

Stellenbosch University https://scholar.sun.ac.za

Opsomming

Deeglike uitleg verifikasie van supergeleier geïntegreerde stroombane strek verder as bloot die nasien van ontwerpreëls en die onttrekking van parameter waardes. Eersgenoemde word gebruik om vas te stel of daar voldoen word aan die proses se ontwerpreëls, en laasgenoemde om die waardes van komponente soos induktors en resistors en die kritiese strome van Josephson aansluitings te bepaal. Nogtans bied nie een van hulle veel waarskuwing teen subtiele uitlegfoute wat onbeplande parasitiese elemente kan veroorsaak nie, of teen ‘n stroombaan wat nie die oorspronklike stroombaan topologie weerspieël nie.

‘n Gespesialiseerde implementasie van Cadence Virtuoso maak LVS (layout-versus-schematic) verifikasie moontlik, maar dit is beperk tot kommersiële sagteware en ook beperk in terme van bruikbaarheid. Parameter onttrekking sagteware soos InductEx word gebruik om waardes van die komponent-elemente van ‘n stroombaan vanuit die uitleg te onttrek wanneeer die stroombaan topologie as ‘n netlist, wat meestal deur die ontwerper geskep is, voorsien word. Die elementwaardes word egter onttrek volgens die topologie wat verskaf is, al is daar uitlegfoute, soos byvoorbeeld wanneer ‘n koppeling met ‘n verkeerde node plaasvind, of wanneer daar netlist foute is wat modelteenstrydighede veroorsaak. Na ‘n mislukte verifikasie poging word verdere diagnostiese stappe gedoen om te bepaal of die fout in die uitleg lê, of in die spesifieke topologie wat verskaf is, wat natuurlik die verifikasieproses aansienlik verleng.

Hier stel ons ‘n vrystaande LVS verifikasie sagteware-pakket vir supergeleier geïntegreerde stroombane bekend, en bespreek, deur middel van die algoritmes wat die kern daarvan uitmaak, die implementering van hierdie sagteware-toestel. Ons bied die resultate van die LVS verifikasie aan en wys hoe die LVS sagteware toestel as geheel saam met InductEx gebruik kan word om automatiese parameter uittrekking vir sel-vlak uitleg verifikasie te berwerkstellig.

Die huidige weergawe van die pakket bied die verbruiker drie alleenstaande programme wat verkieslik saam met InductEx gebruik moet word: ‘n GDSII “file flattener”, ‘n uitleg-tot-schematiese diagram netlist ekstraktor (met die opsie om ‘n herkonstruktueerde beeld van netlist en skematiese diagram te besigtig) en ‘n netlist vergelyking toestel waarmee die verbruiker kan vasstel of ‘n uitleg met ‘n oorspronklike skematiese diagram ooreenstem.

Ons lei af dat die netlist vergelyking toestel ‘n waardevolle metode bied om die uitleg verifikasie proses te bespoedig en vergemaklik en die kanse van foute te minimaliseer. In sy huidige vorm is die uitleg-tot-skematiese diagram toestel beperk omdat dit nog nie stroombane met koppeling kan steun nie.

Stellenbosch University https://scholar.sun.ac.za

Alhoewel vele verbeteringe aan hiedie pakket nog gemaak kan word, kan die geimplementeerde weergawe reeds van groot waarde wees vir RSFQ (Rapid Single Flux quantum) sel ontwerpers.

Stellenbosch University https://scholar.sun.ac.za

Acknowledgements

- First and foremost, God. I trust Him with my life and am even more in awe of His complexity after finishing this thesis. God is, ultimately, the perfect engineer.
- Prof C. Fourie, my supervisor and Dr M. Volkmann for inspiration and encouragement. You are two of the most logical, brilliant people I know.
- Cameron Taylor, my amazing boyfriend and an exceptional engineer for his support and love.
- The Japanese professors and students at Yokohama University for answering my questions and supporting me while I spent time in Japan.
- Last and definitely not least importantly, my parents. For always praying for me and believing in me.

Stellenbosch University https://scholar.sun.ac.za

Table of Contents

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

List of Abbreviations

2D: Two Dimensional

3D: Three Dimensional

ALAS*: A Layout Analysis System*

BLEX: Block Extraction

CMOS: Complementary metal-oxide-semiconductor

DMOS: Diffusion Metal Oxide Semiconductor

DC: Direct current

eSFQ: energy-efficient Single Flux Quantum

ERSFQ: Energy-Efficient Rapid Single Flux Quantum

GDS: Graphic Data System

HEX: hexadecimal

IC: Integrated circuit

LSI: Large-Scale-Integration

LVS: Layout versus Schematic

MSI: Medium-Scale Integration

PCB: Printed circuit board

RSFQ: Rapid Single Flux Quantum

RQL: Reciprocal Quantum logic

SFQ: Single Flux Quantum

SIC: superconductor integrated circuit

SREF: Structure Reference

SSI: Small-Scale Integration

SVD: Singular Value Decomposition

SVG: Scalable Vector Graphics

Via: Vertical Interconnect Access

VLSI: Very-Large-Scale-Integration

Stellenbosch University https://scholar.sun.ac.za

List of Figures

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

List of Tables

Stellenbosch University https://scholar.sun.ac.za

Chapter 1
Introduction

Further thought led me to the conclusion that semiconductors were all that were really required... I also realized that, since all of the components could be made of a single material, they could also be made in situ interconnected to form a complete circuit

Jack S. Kilby, Physics Nobel Prize winner, 2000

Kilby's realization in 1958 that "semiconductors were all that were really required" [1] changed the face of electronic design. His concept of Integrated Circuits (ICs) has made the existence of portable, personal computers possible although this is just the tip of the iceberg of IC usage today. Our world has changed so radically in the last few decades that we virtually rely on ICs for the functioning of our society as demonstrated by the concept of the Internet of Things (IoT) [2].

One could speculate that Kilby would find it hard to imagine how dependent we have become on technologies that have been sparked by his invention and how even the design of today's ICs themselves can be possible only through the use of Computer Aided Design (CAD) and Electronic Design Automation (EDA).

In this thesis, the development of a key component in EDA for a specific family of ICs will be presented, namely Layout Versus Schematic for SFQ ICs.

1. 1 Layout Versus Schematic (LVS) as part of EDA

Before describing what LVS is and why it is needed, the terms *layout*, *schematic* and *netlist* will be defined.

The IC layout is drawn by a mask engineer using layout editing software (CAD software). The layout consists of mask layers that represent the actual layers of the IC that is to be fabricated. The layers contain polygons that either represents the presence of metal or the absence of isolation material (depending on the layer type). The layer information (such as the type of material, layer width etc.) is contained in a separate layer description file.

The *schematic* diagram of the IC (including the parameter values such as inductance, capacitance and resistance) is either captured by the design engineer using a schematic tool or is drawn by hand and then input into LVS software as a text file which is often then referred to as a netlist. In the netlist each component is named (e.g. L1, C1, R1). Next to each component name, the node numbers to which the component is connected are listed (e.g. L1 1 2).

The purpose of LVS software is to check that the IC layout implements the designed IC circuit schematic diagram – this process is also referred to as verification. The LVS check is one of the final steps in EDA and is highly important because mistakes often occur during the process of drawing the layout from the original circuit schematic. If these mistakes are not identified and eliminated before manufacture, the IC may be completely defective. In this thesis, we will focus on LVS for a specific,

Stellenbosch University https://scholar.sun.ac.za

namely Single Flux Quantum.

1. 2 Verification of Single Flux Quantum (SFQ) ICs

Various implementations of SFQ superconductive logic families have been identified as having great potential [27], [28]. Recent technology developments such as that of eSFQ [29], ERSFQ [30], RQL [31] and AQFP [32] have decreased the already low energy consumption (~10−19 J) [29], [33], attracting even more attention to SFQ.

To allow Large Scale Integration (LSI) of SFQ circuits with these evolving technologies, it is important that layout verification tools keep up with their CMOS verification counterparts [28],[34], [35],[36],[34]-[36]. Unlike for semiconductor technologies such as Complementary metal-oxide-semiconductor (CMOS) EDA there are currently no free, readily available LVS tools for even Small and Medium scale Integration of SFQ circuits.

In this thesis we present a toolkit that addresses this need by providing the IC designer with 4 stand-alone tools with which to conduct layout verification for Small and Medium Scale Integration. We focus on a cell-level approach which means that the user will perform LVS on the individual cells of the IC and not on the entire IC at once. This toolkit is comprised of a file flattener to pre-process the layout files before LVS, a layout-to-schematic netlist extractor to extract the underlying schematic model from the mask layout, a netlist comparator that does the final LVS check and finally InductEx [47], a well-known [36] parameter extraction tool that calculates the inductance and resistance values of the components in the extracted IC schematic by using the IC layout as an input. Before delving into the complexities of completing this task, a brief history of the IC industry will be given.

1. 3 Status of the IC Industry

The IC industry, having commenced in the early 1960's [1], is currently one of the fastest growing industries worldwide [3]. The optimisation of software tools that are used in IC design and development have therefore been a topic of great interest to the engineering and the commercial world alike. Vast improvements have been made to these IC design tools since the inception of large and very large scale integration (VLSI), but these are limited mainly to semiconductor technology [3] – more specifically the planar Complementary metal-oxide-semiconductor (CMOS) technology [4].

Planar CMOS has dominated the integrated circuit industry since the 1980's [5] and most IC development software has therefore been focused on semiconductor implementations.

The almost linear improvement in device feature size of semiconductor ICs occurring over the last four decades was foretold by the now famous Gordon Moore [6], whose prediction is now often referred to as Moore's Law. A succinct summary of this "law", as given in Thomson and Parthasarsthy's 2006 paper [5], is:

"Moore's law is the empirical observation that component density and performance of integrated circuits doubles every year, which was then revised to doubling every two years." [5]

This steady improvement in device feature size is, however, not sustainable [5] and power

Stellenbosch University https://scholar.sun.ac.za

consumption of super-computers needs to be addressed [7]. Due to the innate properties of the metal-oxide Field Effect Transistors (FET), the oxide layer has to shrink in proportion to its gate length [8]. As gate lengths shrink to reduce feature size, the oxide layers (only 10s of molecules wide) begin to fail [5], [8]. To continue down the desired path of feature size scaling, the microelectronics industry is faced with two main options: (1) modify the CMOS transistor or (2) find other technologies as alternatives to CMOS.

Although companies that focus on mainstream IC development such as IBM [9], Global foundries [10] and Intel [11] have opted for option 1 at this stage (and are yielding successful results at the 22, 20 and 14 nm nodes [11], [12]), emerging technologies are receiving their fair share of interest [13] and are gaining momentum.

1. 3. 1 More Moore, Beyond CMOS and More than Moore

The International Technology Roadmap for Semiconductors (ITRS) [7] plays a critical role in government, research and commercial decision making with regard to the IC industry; it is used to predict and benchmark emerging research devices. These devices can be categorized into three classes: More Moore, Beyond CMOS and More than Moore.

The first, More Moore [14], corresponds to option 1 given above: modify the CMOS transistor. Currently the two most popular options in this category are: (a), continue with planar technologies [8 Ahmed, K.], such as Ultra-Thin Body and Box (UTBB) fully depleted silicon-on-insulator (FDSOI) FETs [15], or (b), migrate to 3-D IC [16] such as FinFET [17], [18], [19] and TriGate [20]).

The second category, “Beyond CMOS”, corresponds to the second option (find alternatives to CMOS) and refers to technologies that aim to bypass the scalability and power usage problems that CMOS is currently facing and provide solutions for the “end of the roadmap”[21]. These technologies include Tunnel FETs, Spin Transistors, quantum electronics and nano-electronics (silicon nanowires (NWFETs), carbon nanotubes and graphene FETs) [22], [23].

Other nano-technologies such as nanoelectromechanical systems (NEMS) along with microelectromechanical systems (MEMS) fall into the final category: More than Moore (MtM) [24], [25]. MtM devices provide non digital micro- or nano-electronic functions to ICs – usually by 3D integration – and can extend both options 1 and 2. The prediction of growth of these devices is a challenge since they do not scale according to Moor’s law [13].

1. 3. 2 CAD and Electronic Design Automation (EDA)

Due to the wealth of IP that is captured in CMOS CAD software, design flow [26] and the IC designs themselves, much attention is currently being focused on technologies that are either CMOS compatible or can be easily interfaced with CMOS but provide diversification [24]. These MtM and beyond CMOS devices will ideally become part of so called “Extended CMOS”. It is important to note that none of these MtM or beyond CMOS technologies will be able to fully replace CMOS in the foreseeable future [13].

Stellenbosch University https://scholar.sun.ac.za

Certain categories of computing can, however, greatly benefit from these emerging technologies. In the field of supercomputing, an alternative to CMOS is highly desirable; with regard to high speed computing, the Emerging Research Device Summary of ITRS 2013 states that successful technologies may "eventually replace the CMOS gate as a new information processing primitive element" [13]. We are currently far from this point, but for progress to be made, it is imperative that CAD and EDA software support the design process.

Making even a relatively simple change such as changing the transistor structure from that of planar CMOS to a 3D transistor poses a challenge. Multiple stages in the design flow (such as layout editing, design-rule checking and parameter extraction) need to be modified to accommodate the new transistor structure. Furthermore, the IC layout designs with millions of transistors have to be changed; this cannot be done manually due to time constraints. Automation of this sort is not a trivial task and is currently being addressed for various processes and transistor types [4].

The complexity required to modify EDA for technologies that differ fundamentally from CMOS is a yet greater problem [4], [13]. Layout verification, also known as Layout Versus Schematic (LVS) forms a particularly challenging part of this problem. LVS for SFQ – the focus of this thesis – is effectively a LVS implementation for a highly promising Beyond CMOS technology.

In the next chapter we begin with a literature study and also provide the background information for concepts that are required to understand the rest of the thesis.

Stellenbosch University https://scholar.sun.ac.za

Chapter 2
Literature study and background

With the advent of the transistor and the work in semiconductors in general, it seems now possible to envisage electronics equipment in a solid block with no connecting wires. The block may consist of layers of insulating, conducting, rectifying and amplifying materials, the electrical functions being connected directly by cutting out areas of the various layers.

G.W.A Dummer, 1952.

In this chapter we will start by giving an overview of the IC design process in the general case and then zoom in to IC design for CMOS and SFQ technologies. We also discuss manufacturing processes and how different manufacturing processes affect EDA and more specifically LVS.

2. 1 The IC design Process

The semiconductor design process has progressed significantly since Dummer's prediction in 1952 [45] and the development of the first IC in 1958. The invention of the first IC was followed by small-scale integration (SSI) in the 1960s, medium-scale integration (MSI) in the late 1960's and eventually large and very-large-scale integration (LSI and VLSI) in the 1970's and 1980's respectively [46].

Since the inception of VLSI, much research effort has been spent on developing and improving software tools to aid IC design [38]. The IC design flow has grown in complexity and has been refined over the last decades [26], [37], [38]. As new technologies are born, the design toolkits for these new technologies are moulded to best fit new needs. Stand-alone tools can be used for each step in the design process flow, but integrated design systems or packages are more commonly used in industry today (such as Cadence [37], [12], Synopsys [26] and Mentor graphics [39]).

EDA software can be broken down into the following categories that often overlap [26], [34]-[36]:

- Synthesis, place and route (software that aids the mask designer to converting the schematic into a layout)
- Layout editing (CAD software that allows the user to modify the polygons on the various layers of the layout)
- Design rule checking (software that ensures that rules regarding minimum feature size and spacing of features are within allowable manufacturing tolerance levels)
- Logic simulation and circuit simulation
- Electrical rule checking and layout versus schematic
- Parameter extraction

The usage of these tools depend on the technology and design methodology. The two main categories of digital design methodology are *full-custom* and *semi-custom* [35], [49].

2. 1. 1 Full- and Semi-custom design environments

Stellenbosch University https://scholar.sun.ac.za

In a *full-custom* design environment, design is done from the cell level (in CMOS, for example, transistors are drawn individually). Most of the circuit optimisation and some of the verification is done manually – resulting in long lead times (between schematic or logical IC design dates and manufacturing dates). The full-custom methodology is usually employed either when performance is critical, or for newer technologies where automated tools are not yet fully developed or reliable [36]. This approach is better suited to small and medium scale integration.

In *semi-custom* design environments, a standard cell library is required. Hardware description languages (such VHDL) are used to describe the IC functionally; thereafter synthesis tools are used to convert the functional specification into an optimised (flat) netlist [35], [49]. Automatic place and route software is used after the synthesis process [49] to create the layout. Verification is mostly automated in a *semicustom* environment.

When cell-level design is required for LSI and VLSI, a hybrid approach between full- and semi-custom is usually implemented [35].

2. 1. 1. 1 Full-custom

The toolkit presented in this thesis has been developed for a full-custom environment. From a high level perspective, the full-custom design flow can be summarised as follows.

Initially, a schematic diagram of the cell's circuit is designed and drawn. The schematic is then converted to (or translated into) a machine-readable netlist (this can be done by hand, but when a schematic editor is used to draw the schematic, the conversion is usually automated [36]). This netlist is then simulated with respect to its functionality (does it meet the functional specifications of the cell with respect to logical input and outputs) as well as timing (are timing criteria met). Once the design criteria have been met, the schematic must be translated by a layout design engineer into a physical layout using layout editing software.

Parameter values of each component in this physical layout are highly sensitive to the area and shape of the polygons that describe them [36]. This makes drawing a layout that correctly represents the schematic a difficult task. Small changes to the dimensions and placement of the polygons can have a large effect on the parameter values and consequently on the functionality of the circuits on the chip [40]. Margins also have to be carefully observed since deviations in the manufacturing process will affect parameter values too. An incorrect parameter value can result in complete IC failure, therefore verification of the layout file is vital to the IC design process [35].

Unless verification is manually performed, layout-to-schematic (also referred to as netlist extraction) is a crucial part of this process. The extracted schematic that is generated from the layout is compared to the original, designed, schematic to check that the layout is equivalent, in model, to the schematic. This process of comparing the model of the extracted schematic with the original schematic is often considered the final step of layout-versus-schematic (LVS) verification [38]-[44]. Strictly speaking, however, parameter extraction should be included in LVS.

Stellenbosch University https://scholar.sun.ac.za

The extracted netlist will often contain parasitic components that were not originally included in the input schematic. Before parameter extraction, we do not know *how significant* the effects of these previously not included components will be. To determine this, these components can be back-annotated into the model of the original schematic [50] (which we will now call the *modified* input schematic).

Once the back-annotation process is complete, parameter extraction can be performed on the modified schematic; if the parameters of the original schematic and extracted parameters of the modified input schematic are sufficiently similar we say that LVS has been successfully performed: the modified input netlist (with extracted parameters) can now be simulated. If the desired simulation results are not achieved, the layout (or even sometimes the schematic design) should be modified. The verification process will then be repeated with the new design until satisfactory results have been achieved [34].

2. 1. 2 History of LVS

In the 1980's many papers were published about layout verification and netlist comparison methods and algorithms. "*Efficient Netlist Comparison Using Hierarchy and Randomness*" by J.D Tygar published in 1980 [48] briefly discusses the history of layout-to-schematic and netlist comparison software. Tygar also presents a LVS solution for basic CMOS and NMOS microprocessors. Restrictions are, however, placed on the design of layouts based on the specific semiconductor processes and logic elements. Erich Barker's 1984 paper, "*A Network Comparison Algorithm for Layout*" [43] extends some of these techniques and also provides a more comprehensive LVS solution by making use of block extraction techniques. The software, A Layout Analysis System (ALAS), applies an isomorphism implementation to check the equivalence of the graph representation of netlists by using information obtained by the Block Extraction (BLEX) algorithm.

Today, design suites often contain combinations of software tools which include LVS to provide the layout designer with a complete design solution [37],[26]. The details of current LVS techniques are not discussed in detail in publications, since this software is mostly proprietary. Incorporating multiple tools into a software package can reduce the complexity of the individual components of the software (and their core algorithms). For example, the designer can aid the component identification process by labelling sub circuits or cells with the same names in the schematic and layout file – greatly simplifying the layout verification process. This is especially useful in VLSI semi-custom environments where layout-driven-schematic [50] techniques are employed. Unfortunately these software packages are not affordable to smaller companies and research groups [26].

2. 1. 3 Superconductor electronic (SCE) design software

SCE design software can be categorised into (1) in-house tools that are not freely available, (2) freeware or open source tools and (3) commercial toolkits or suites (not commonly used by smaller companies or research groups). In the paper *"Status of Superconductor Electronic Circuit Design*

Stellenbosch University https://scholar.sun.ac.za

Software" [36], an investigation was conducted to compare the use of various SCE tools and software solutions. This investigation shows that no one set of tools currently dominates the SCE design software market [36].

In-house tools are commonly used for more *technology dependent functions* [36] (such as optimisation and margin/ yield analysis), whereas freely available tools are the most popular for inductance extraction and circuit simulation [36]. The Cadence software suite is the most popular of the third category and is used by many of the larger SIC companies for schematic capture, layout, logic simulation, DRC and LVS.

Cadence's SCE tools have been adapted and calibrated for RSFQ making it easier for CMOS IC designers to become familiar with SCE design. The learning curve for using Cadence packages is, however steep. XIC [51], a more lightweight toolkit, is the other popular software suite; tools for schematic capture, layout and circuit simulation are provided. Both of these commercial packages rely on the semiconductor markets for revenue.

From these results we can conclude that a freeware LVS toolkit will be beneficial to the SCE community.

2. 2 Digital superconducting technologies

Digital superconductive technologies can be subdivided into two main categories – voltage state and flux quantum [34]. The first generation of superconductive technologies were of the voltage state category.

Voltage state relies on the principle property of the Josephson junction to switch quickly to its resistive state as current increases above its critical value and then to remain in this state even when the current is no longer present [27]. The bias current, in the form of voltage pulses, is supplied to the junction at the clock frequency (Josephson voltage state latching logic is externally clocked). One pulse will therefore switch the junction to its resistive state (logical 1) and the next will return the junction to its superconductive state (logical 0). Junctions are naturally underdamped, resulting in a hysteretic VI curve [61]. We will later see the relevance of this to netlist extraction.

Josephson latching logic, unlike flux quantum logic, is similar to the logic used by semiconductor technologies; so similar that semiconductor software could in many cases be used for the IC design process. This allowed for design engineers to enthusiastically begin designing and manufacturing ICs. In 1980, IBM announced the goal of creating a superconducting supercomputer. This project was, however, abandoned in 1983. Since then, manufacturing techniques have improved, the Josephson junction's composition has changed, and Josephson Latching logic has been replaced with logics that detect the presence of flux pulses [27].

After recent developments in the superconducting digital logic field, it is safe to say [27], [33] that the failure of IBM to create a superconductive supercomputer, does not imply that this will be the fate of other projects such as the Cryogentic Computing Complexity (C3) program [52].

Stellenbosch University https://scholar.sun.ac.za

2. 2. 1 Rapid Single Flux Quantum (RSFQ)

RSFQ, the first of these pulse-base logics to attract widespread attention, became popular after Likarev and Semenov's 1991 review paper [27]. RSFQ is based on the same principle of using voltage pulses of quantized area to encode logic that was first employed in Resistive SFQ logic – the first technology to utilize the characteristic of overdamped Josephson junctions to produce flux quanta [62]. *Rapid* SFQ, (with name chosen partly to result in the same acronym as its predecessor), uses junctions instead of resistive interconnects, resulting in higher switching speeds and improved margins.

Josephson junctions are used control the movement of voltage pulses of quantized area:

$$\int V(t)dt = \Phi_0 = \frac{h}{2e}, \tag{2.1}$$

where h is Planck's constant and e is elementary charge. These pulses are generated from switching Josephson junctions. For a comprehensive introduction into RSFQ, the reader is encouraged to read flagship paper, *"RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock frequency digital systems"* [27].

2. 2 .2 Manufacturing processes

SICs, like CMOS ICs, are manufactured in the form of wafers; multiple chips are produced from one wafer. Each wafer is comprised of various layers of material that differ in width. These layers are either isolation layers (usually a form of oxide) or metal layers. Metal layers can be either superconductive or resistive.

Lithographic techniques (such as deposition, patterning and etching) are used to deposit the layers sequentially from the bottom up; metal layers are connected to one another either through holes in the isolation layers that separate them – which we call vias (vertical interconnect access) – or through tunnel junctions. Isolation layers and ground plane layers (metal) are usually negative layers: they are defined by where the isolation layer or metal layer is removed (or not deposited). The resistive and other superconductive layers are usually positive layers and therefore defined where the metal is deposited (by polygons in a layout file).

The number of metal layers and their widths, the minimum feature size, the type of materials used and the existence or lack of planarization depend on the specific manufacturing process. Each process also has a set of design rules that restricts the user from designing layouts that have a high chance of failure due to manufacturing failure. For example, vias are not allowed to be smaller than a set size.

These manufacturing processes are often specific to a research group, country or company. The details about many manufacturing processes are not freely available; proprietary information about these processes may not be divulged. Two popular manufacturing processes that are open are the 1 kA/cm^2 Nb RSFQ Fluxonics process from IPHT Jena [57] and the 4.5 kA/cm^2 process from Hypres [58], [59]. The software in this thesis has been written to support these processes although most of the algorithms have been written so as to be generalizable to other processes.

Stellenbosch University https://scholar.sun.ac.za

2. 2. 2. 1 The Fluxonics process

The Fluxonics process has a single ground plane layer as the lowest metal layer, followed by a superconductive layer (M1), the JJ layer (TRI), another superconductive metal layer (M2) and a resistive layer [60]. These metal layers are separated by isolation layers (except for M1 and the TRI layer which are connected directly) [60].

In Figure 2.1, we see a cross section of the Fluxonics 1 kA/cm^2 Nb RSFQ process. We can see that I2 is used to isolate R1 from M2 and that I1A and I1B are used to separate R1 from the layers beneath it. This fabrication process differs from the Hypres process (b) in this regard and allows for easier via identification.

Lines can be drawn at 45°, 90° and 135° [60], therefore polygon processing techniques that apply only to Manhattan polygons [56] (in this case polygons whose edges may only be are parallel to x or y axes) cannot be used for this process.

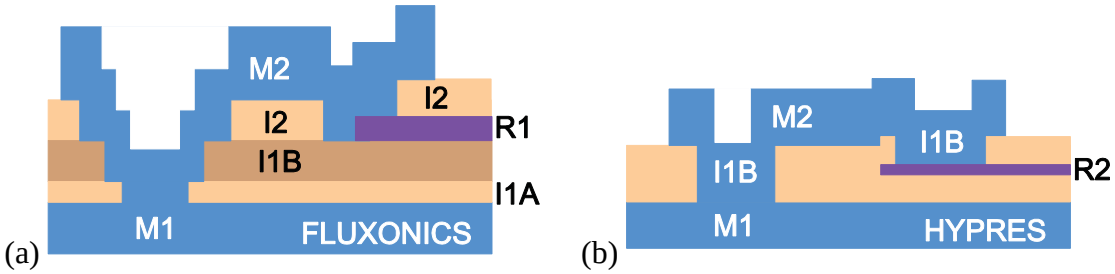


Figure 2.1: In (a) we see the relevant layers in a cross section of the Fluxonics 1 kA/cm^2 Nb RSFQ process. In (b) the layers between M1 and M2 of the 4.5 kA/cm^2 Hypres process are shown. Figure used with permission from C. J. Fourie [113].

2. 2. 2. 2 HYPRES process

The Hypres 4.5 kA/cm^2 Nb process is a slightly more complicated process than the Fluxonics process in terms of layout to schematic extraction. Although similar in that both processes have only one ground plane layer, JJ layer and resistive layer, the Hypres process has only one isolation layer defined between its lower superconductive layer (M1), the second superconductive metal layer (M2), the resistive layer and the JJ layer (as seen in Figure 2.1 (b)).

For both parameter extraction and layout to schematic extraction, an additional dummy layer needs to be introduced so as to determine the layer connectivity.

2. 2. 2. 3 Other multilayer process

Other manufacturing processes include (but are not limited to) newer Hypres processes technologies, technologies from D-wave Systems [63], MIT Lincoln Lab's 10 kA/cm^2 technology [52], and the standard and advanced Japanese processes. Time was spent with a Japanese group so as to understand their process sufficiently to adapt the LVS tools to support their processes in the future.

The two main processes currently used in Japan are the standard 2.5 kA/cm^2 process (SDP2) and the 10 kA/cm^2 advanced process (ADP2) [53]. The standard process (Figure 2.2) is an older process and is still the cheaper of the two processes. Adiabatic logic is currently implemented using the standard process, although there are plans to migrate this logic to the advanced process.

Stellenbosch University https://scholar.sun.ac.za

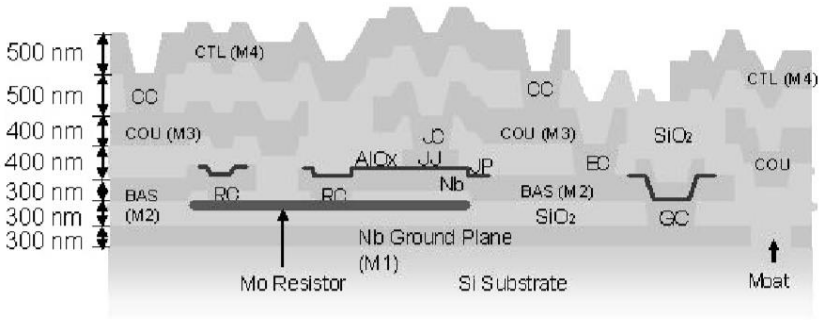


Figure 2.2: A cross section of a device fabricated using the standard process. Figure with permission. Source found in: "*Current status and future prospect of the Nb-based fabrication process for single flux quantum circuits*" [53].

As well as being a faster technology with a higher J_c (10 kA/cm^2) [54], the advanced process includes two separate passive transmission line (PTL) wiring layers that are each protected on both sides by ground planes. This has the advantage of allowing passive wiring to be done underneath the active logic circuit elements which results in the option of more effective chip real estate usage.

The bottom layers, from the DC power layer up to the main ground plane, are planarized [54]. The active layers include a resistive layer above the Main Ground Plane, the Base layer for biasing, the JJ layer and the Counter layer for JTL wiring. Pillars that connect layers are used for DC biasing. These bias currents originate from the bottom (DCP) layer.

2. 2. 3 Low power superconducting technologies

There are two main types of low power technologies: those using AC bias and those using DC bias. For the first type, the clock itself acts as the AC bias and is not generated on chip (these circuits are externally clocked). For DC biased low power technologies – when clocked – the clock is generated on the chip itself.

2. 2. 3. 1 AC biased low power technologies

The two main AC biased low power technologies AQFP [32] and RQL [31], [55]. Both these technologies are very promising and have already proved their worth by creating a variety of complex components (including adders) at impressive speeds with very attractive low power consumption [64], [65].

AQFP does not use the principle of flux pulses, but rather relies on the principle of slowly increasing current in a loop to represent its logic. Using the adiabatic principle results in very low power operation that is clock speed dependent [32].

The principle of mutual coupling is used extensively in AQFP logic, which poses a problem for the current version of the layout to schematic software that has been designed (this will be discussed in more detail later). All bias lines are connected by mutual coupling and data transfer is also done this way. There are no JTLs required. Currently no PTLs either, but in future they may be added. The SDP2 process is currently the process of choice for AQFP.

RQL relies on a different process of operation to any of the others. Pulses are used, but one pulse

Stellenbosch University https://scholar.sun.ac.za

does not represent one logic operation but rather a pair of pulses (a positive and negative pulse) [31].

This logic has the advantage of very low latency since more than one operation can be performed per clock cycle. Depending on the specific architecture, this can mean many more operations per GHz than the other technologies. The dynamic power dissipation, however, is not as attractive as for AQFP.

2. 2. 3. 2 DC biased low power technologies

ERSFQ [30] and eSFQ [29] are the two main DC biased low power technologies, although RSFQ with lowered driving voltages (LV-RSFQ) [66] has also proven to be a promising alternative (lower static power dissipation). Both ERSFQ and eSFQ are similar to RSFQ in structure, but modifications in the logic avoid the use of (highly dissipative) bias resistors, resulting in lower power consumption.

2. 3 CMOS LVS as opposed to SIC LVS

Graph theory and (more specifically graph and subgraph isomorphism), has been used since the 1980s to address the subcircuit matching problem. For background on this topic, see Appendix G. Because of the high computational complexity of subgraph matching, heuristic and probabilistic techniques have been developed improve execution time for CMOS LVS.

Since the developed toolkit has been designed for use in a full-custom environment (and not for VLSI), such techniques were not deemed within the scope. For a more in-depth look at recent solution to the CMOS graph matching problem, the user is advised to read *Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach* [107].

Other Advanced probabilistic techniques such as Probabilistic Graphic Models (PGMs) are also used in some layout verification tools for Beyond CMOS technologies [108]; such techniques could be incorporated in SFQ verification tools in the future.

2.3.1 CMOS netlists as Bipartite graphs

A common way of representing CMOS circuits is to represent every component and node as a vertex. The edges connect these vertices to one another in a way that replicates the topology of the circuit. Since components will always be connected to nodes and vice versa the graph will always be bipartite [107]. This differs from the approach used in the toolkit where components are represented as *edges*.

Because the inductance of the interconnects in CMOS are not of great concern to the designers, this graph representation can be used throughout the LVS process.

If this approach was used in the layout-to-schematic tool for SFQ circuits, we would not be able to use the existing MST libraries directly.

2.3.2 Possible use of alternative graph structure to allow for mutual coupling

Mutual coupling is not part of the scope of the developed toolkit. Should it be included in the future, an approach similar to that used for CMOS graphs could be used for the SFQ netlist comparison tool (not for layout-to-schematic tool).

Stellenbosch University https://scholar.sun.ac.za

Both the extracted and input netlists would be converted to a bipartite graph after layout-to-schematic extraction is complete. Thereafter, the mutual coupling "components" could be added. The coupling factor can be seen as a component that connects a component (inductor) to another directly (not through a node). The resulting graph would no longer be bipartite, since adding the mutual coupling "components" would connect component vertices to other components vertices directly.

With the currently implemented graph structure, we would have to connect edges to other edges directly through another edge. This is less conducive to viewing the graph graphically and when using existing algorithms.

We will now discuss algorithms that were implemented in the toolkit.

Stellenbosch University https://scholar.sun.ac.za

Chapter 3
Layout to Schematic
Algorithms and Implementation

In this chapter the core algorithms used in the layout to schematic tool will be discussed. First, an overview of the layout to schematic tool will be presented: we shall discuss the data flow, and required file formats.

3. 1 Overview and file formats

The information flow of the layout to schematic tool is presented in Figure 3.1. The only two input files required are the layer definition file (.ldf) and the layout file (.gds). The essential output file is the netlist file (.cir). This extracted netlist file can be used as an input to InductEx [113] (which also requires a layer definition file and layout file). Additional files shown in Figure 3.1 are the scalable vector graphics files (.svg) that show the vias, metal layer connections, user selection nodes and ports, as well as the output schematic. These will be discussed in more detail later in the chapters. The *fastout.out* file will also be mentioned later in this thesis.

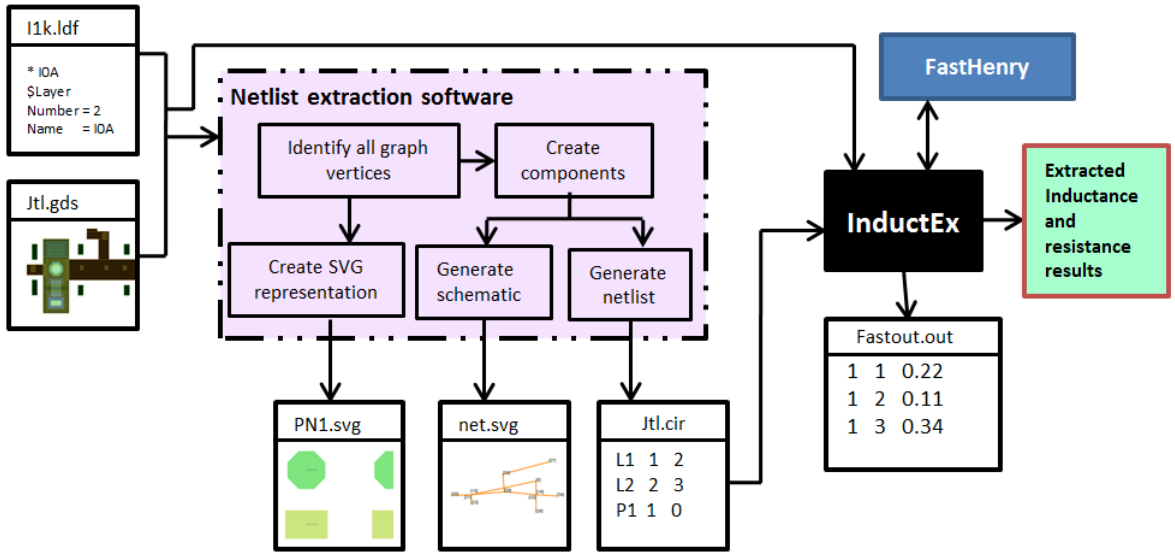


Figure 3.1: Figure showing the input and output files that interface with the netlist extraction software and how the output files can be used with the layer definition file and layout file to interface with InductEx.

3. 1. 1 Design approach

The top-down software design approach (Figure 3.2) was used; independent modules that are required to solve the problem were identified. This chapter discusses the methods and algorithms used to design and develop these modules. Although Python was used in the early stages for prototyping, the C++ language was used to implement the final algorithms.

Stellenbosch University https://scholar.sun.ac.za

Libraries were used, where possible, to simplify the process as well as to make the software more robust and re-useable. The standard C++ libraries were used extensively as well as the Boost libraries. Because complex polygon processing and graph libraries are required for layout to schematic, the boost::graph and boost::polygon were used although other polygon processing [111] graphing [112] libraries were investigated.

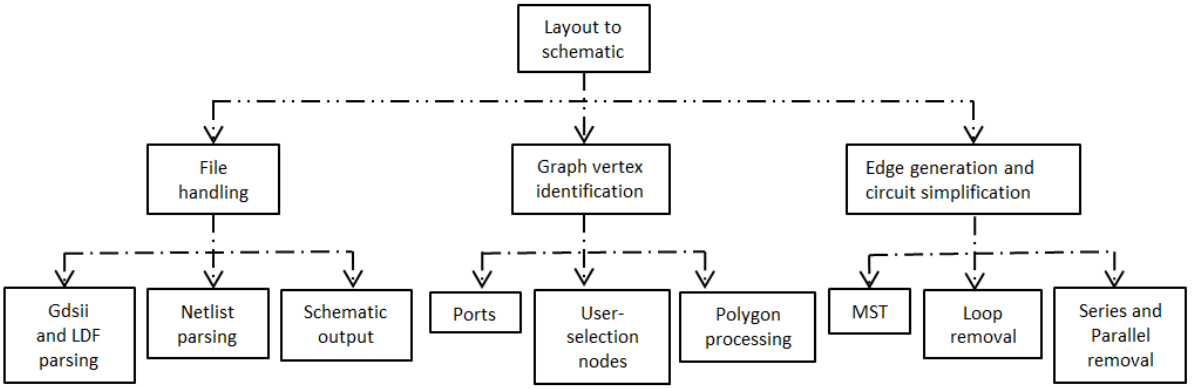


Figure 3.2: Graphic representation of the Top Down approach used to design the layout to schematic tool.

3. 1. 2 GDSII stream file format

As seen in Figure 3.2, graph vertex identification forms an important part of layout to schematic. To identify the geometries that will represent these graph vertices, one must traverse through polygons in the layout file and use the positions of these polygon elements relative to one another to perform polygon operations such as intersections, unions and subtractions. The co-ordinates of these components have to be represented in the same Cartesian co-ordinate system with the same origin as a reference point. If multiple co-ordinate systems are used within a library, pre-processing must occur before a netlist can be extracted.

3. 1. 2 .1 Flattening

As in most CAD type file formats, the GDSII file format uses the principle of structure reference elements. These are referred to as SREF records in the GDSII file format. When a SREF is defined, the location of the elements contained in the referenced structure will not be defined relative to the origin but rather relative to the origin of this referenced structure.

Structure reference elements are created when a cell (or combination of polygons on various layers) is drawn, saved and then later imported into another cell, circuit or entire integrated circuit. These imported cells can be translated, rotated and/ or scaled; this information is stored in element contents records following the SREF. The transformation performed on the coordinates of the cell will have to be reversed in the flattening process and the structure reference element will then be replaced by all these transformed elements.

A file flattener, *gds2GDSFlat*, was written in the C programming language and complied as a separately executable program that is used to pre-process GDSII files before netlist extraction. An explanation of how the parsing and flattening takes place can be seen in Appendix A.

Stellenbosch University https://scholar.sun.ac.za

3. 1. 2. 2 Parsing

The GDSII file header contains information about the entire cell, IC or wafer and this information must first be read in and stored. After this, the rest of the GDSII file can be processed.

When reading the GDSII file to perform layout to schematic, an object-orientated approach is used to store the initial polygon information. A state-machine type method is used to collect the required information for each layer and all the GDSII elements that represent a polygon on that specific layer are used to form Boost polygons. All boost polygons on a specific layer, as well as other information about that layer that is gathered from the GDSII and LDF file, are stored as a layer object. Layer objects are then sorted according to their order numbers (different to their GDSII numbers) and then stored in a vector of layers. Look up tables are created to keep track of the GDSII numbers, order numbers and the indices of the layer vector.

Text elements are handled differently and although one can dedicate a specific GDSII layer to text elements, designers have the freedom of adding text elements to other layers. This concept has to be taken into account in the state machine since port information is extracted from the nearest text element to each port object. It is important that all text elements be evaluated; text that is not in the port format specified in the InductEx user manual [114] will be ignored. A port *object* is simply a polygon on a dedicated port layer that is specified in the LDF file.

3. 1. 3. LDF file format

GDSII files specify the polygons and text elements found on each layer. Without additional information about the characteristics of each layer and the manufacturing process, it is impossible to perform netlist extraction.

As discussed in Chapter 2, negative and positive layers are used to describe the integrated circuit. Without knowing the mask of a layer, it is not possible to identify the connections between layers. This information is required to identify the vertices of the connectivity graph of the IC. An input file containing the process technology parameters is thus vital to the netlist extraction process. Any file format can be used to feed this information into the netlist extraction software but it should preferably be text based so as to make editing user-friendly.

Layer definition files (LDF) are used to configure InductEx. Since this file format contains all the necessary fields for netlist extraction, LDF files will be used in conjunction with GDSII files to perform netlist extraction.

Stellenbosch University https://scholar.sun.ac.za

3. 1. 4 Scalable vector graphics files

In the initial testing of the GDSII and LDF processing functions, Python's *matplotlib.pyplot* was used to plot images of polygons on specific layers and the intersections between layers.

Later, Boost's SVG interface for C++ was used to implement the schematic writer. Scalable vector graphics was chosen as a file format mainly because images can be mapped on top of one another and later be moved apart when using an SVG editor. This is useful because of the 3D nature of SICs. The scalability of SVG is also a crucial feature, especially when large, complex IC's are being plotted.

3. 1. 5 Netlists

Netlists are output in the same format similar to that of SPICE netlist and follow the InductEx user manual's format specifications since they are used primarily in conjunction with InductEx. A more detailed description on netlist parsing will be presented in Chapter 4.

3. 2 Graph vertex identification

The identification of the polygons that will eventually represent the netlist nodes is a key part of extracting the graph representation of an SIC. Nodes will be defined as graph vertices and the components as graph edges. The process of identifying these nodes involves polygon operations, two main operations being intersection and subtraction. The algorithms and implementation will be discussed in this section.

3. 2. 1 Polygon merging

The Boost library has a polygon merging method that takes two polygons and merges them if they overlap or intersect. The polygons have to have the same orientation (the points in a polygon can be defined in a clockwise or an anticlockwise order) and after the merging operation, the orientation of the polygon has to be checked and corrected if the merging operation caused the orientation to change.

The merge method provides the output in the form of a *multipolygon* which is a vector of Boost polygons. The reason for this is because both polygons will be returned in this vector if they do not touch. Only two polygons can be merged at once (not two *multipolygons*), so an algorithm is required to merge all the polygons on a layer. This concept was mentioned in 1975 by Baird [42] in discussions about layout to schematic solutions for semiconductor technologies. It is vital that polygons be merged, since connectivity between points within a layer is dependent on whether the points intersect the same polygon.

All the polygons on a layer are stored as a vector of polygons. These polygons all need to be compared with one another and merged if possible. As the polygons are modified, there are other polygons that they would now intersect that they did not previously intersect. This problem is solved by an outer while loop.

Stellenbosch University https://scholar.sun.ac.za

3. 2. 2 Intersection and Subtraction

When the intersection between two negative layers (isolation layers) that are situated between two positive layers (metal layers) is computed, the resulting region describes the region of connectivity between the two (outer) metal layers. This region of connectivity is often referred to as a via.

In the manufacturing process, metal layers physically touch at these vias. If a metal layer is found directly above another metal layer (for example the TRI layer above M1) then there is a direct connection between these two layers at the point of overlap. In such a case, the upper layer, prevents a via between the lower layer and a layer above it in the region where these two layers overlap and it therefore has to be subtracted from the lower layer to determine if connectivity with layers above the upper of the two layers is possible. This concept is depicted in Figures 3.3 and 3.4. The Fluxonics process is used as an example.

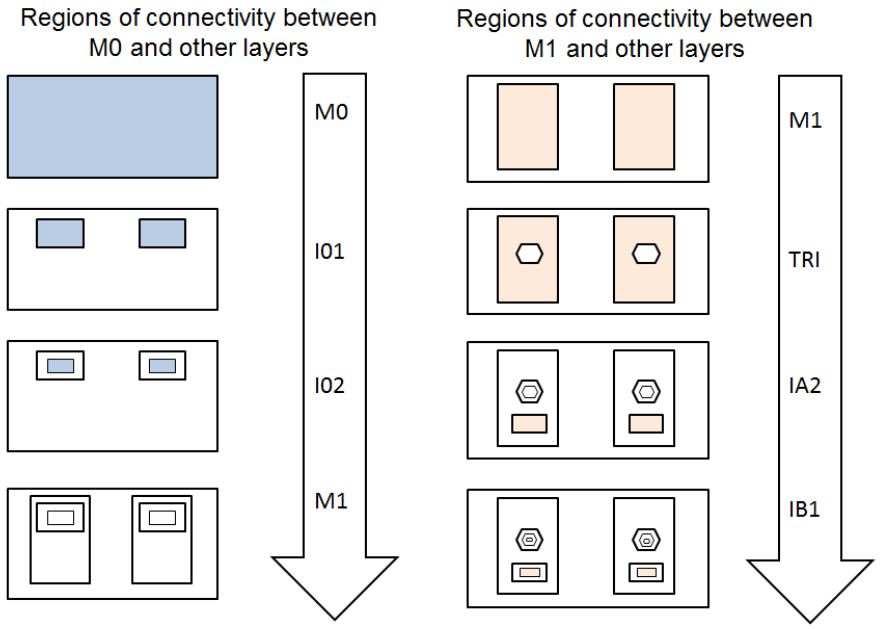


Figure 3.3: Figure depicting the process of identifying the regions of connectivity

Via polygons will be used to create initial graph vertices. It is important to note that the via its self has an inductance and is therefore a component (edge). These polygons (above and below the via) will be used for vertex identification. Regions where two metal layers are connected directly (for example M1 and the TRI layer in the IPHT process) are also used to identify vertices. An example of this can be seen in Fig 3.4. Later, port polygons and user selection node polygons (added to the layout file by the users) will be added as graph vertices.

Two algorithms are used to convert these vias and regions where metal layers connect into graph vertices. They have been named the *initial node finding algorithm* and the *projected node identification algorithm.* The first identifies all the polygons that represent inter-layer metal connectivity and the second maps or projects the appropriate nodes onto virtual layers that represent each metal layer. These nodes are classified as nodes that intersect this metal layer from below or

Stellenbosch University https://scholar.sun.ac.za

from above. SVG images are generated from the results of the *projected node identification algorithm* and can be used in the layout verification process by the user. This will be discussed in Chapter 4.

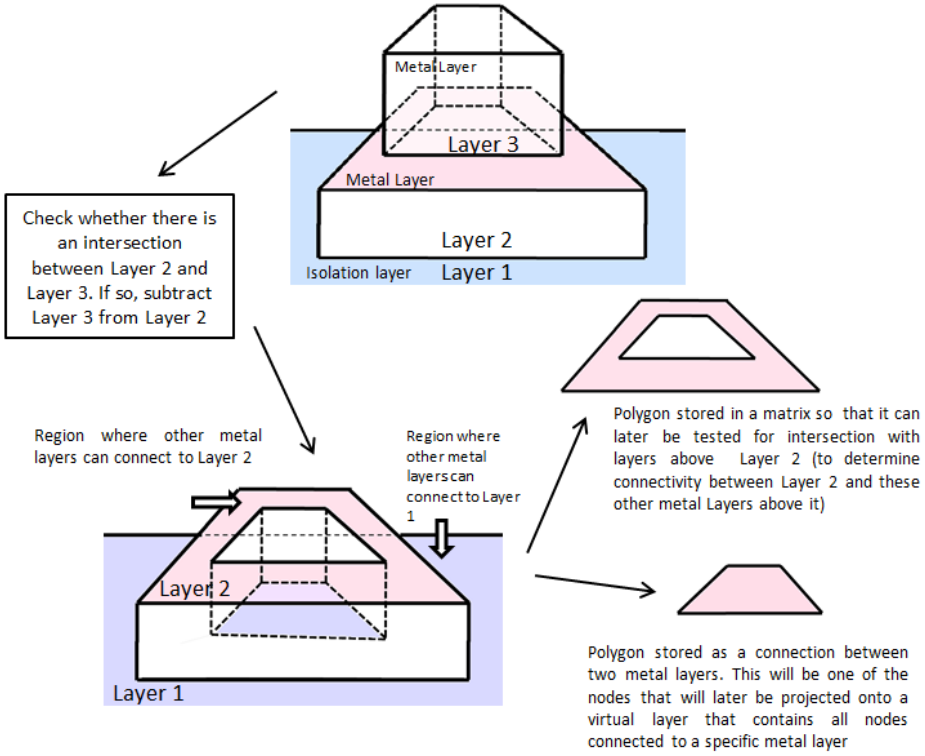


Figure 3.4: A graphical representation of the subtraction and intersection methods

3. 2. 3 Initial node finding algorithm

All intersections between layers are stored as vectors of polygons in a type of adjacency matrix structure. Since the matrix would be symmetric, only the upper triangular matrix entries are stored. To make the implementation more convenient, the i and j indices are chosen in such a way that the matrix can be represented by a column vector with a row vector at each index. The row vector lengths become shorter (by one entry) with every added column entry. The reason for this is that in the first entry of the column vector, the row vector entries represent connections with this entry and the layers above it. At each index of the row vector, a vector of polygons is stored.

When metal connections are identified, they are stored in a separate vector, S, which also keeps track of the direction from which the connection was formed (from a layer below or above). Vector S is the output of the algorithm, while the matrix is merely used to store the intermediate connections between all types of layers. The algorithm is given below:

Stellenbosch University https://scholar.sun.ac.za

```
M[][] = 0
S[] = 0
FOR all i Layers
  FOR all j Layers
    IF i == j
      M[i][j] = Layers[i]
    ELSE IF j<i
      IF Layer[j] is not a metal layer
        M[i][j] = intersect(M[i][j-1],Layer[j])
      ELSE
        M[i][j] = subtract(M[x][j-1],Layer[j]
        S.push(intersect(M[x][ j-1],Layer[j]))
  END
END
RETURN S
```

Note that the *intersect* function handles vectors of polygons. This means that each element of vector S contains multiple polygons and also information about the upper and lower layers that these polygons connect to.

3. 2. 4 Projected node identification algorithm

Before a graph can be constructed, nodes should be ordered into a vector of virtual layers. The term virtual is applied, since each layer represents one of the metal layers but does not contain the polygons found in this layer. Each virtual layer contains a vector of polygons (nodes) that intersect this virtual metal layer. The projection algorithm transforms the output of the initial node-finding algorithm (vector S) into this format.

While traversing though every element of vector S, the algorithm checks whether the lower layer from which each element was obtained is a metal layer or not. If this connection was obtained from a metal layer, every polygon in this vector of polygons will be projected onto the virtual layer of the lower metal layer from which the element was obtained (retaining information as to which two metal layers this node connects). Thereafter, the same technique is applied to the upper layer.

The output of this algorithm is a vector of the same length as the number of metal layers. Each entry of this vector will contain a vector of objects. Each object has a polygon attribute, a colour attribute and two connectivity attributes (*connected_to* and *connected_from*). Projected nodes that have been projected from a layer above the current metal layer are coloured in a darker green and vias from a lower layer are coloured lighter green. In Figures B.2 and B.2 we see examples projected nodes.

3. 2. 5 Vertex generation from projected nodes

Vertices are generated from projected nodes, ports and user-selection-nodes. All projected nodes will become vertices in the original graph representation of the layout (before simplification) unless they intersect a port node (in which case they will be replaced by the port node). Port nodes are defined in a specific GDSII layer (defined in the LDF file). User selection nodes are placed on various GDSII layers that each correspond to a metal layer (also defined in the LDF file).

Stellenbosch University https://scholar.sun.ac.za

We will first describe the vertex generation process overview. Thereafter, the purpose of user-selection-nodes (and how they can be used) will be discussed.

3. 2. 5. 1 Vertex generation process overview

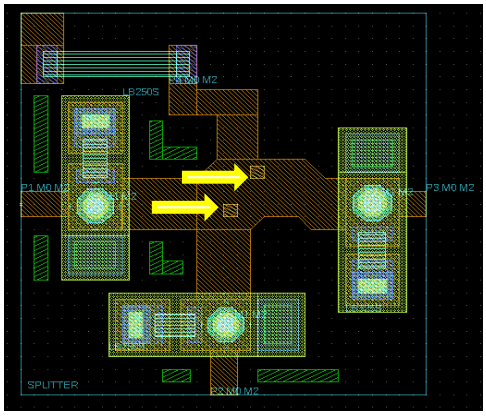
It is important to note that each node (a projected node, port node or user-selection-node) has a polygon attribute. Once these nodes have been identified, the vector that contains the multi-polygons on each metal layer is traversed. Each polygon on each metal layer is evaluated and intersected with the polygon attributes of firstly the projected nodes and then the ports nodes and user-selection-nodes that correspond to that specific metal layer.

If there is more than one intersection for a specific polygon, a component (or set of components) will be generated. If there are more than two components, the within-polygon-optimisation algorithm is used to connect the components in an optimal manner. Before we discuss this algorithm, more detail will be given about the user-selection-nodes.

3. 2. 5. 2 User-selection-nodes

The purpose of user-selection-nodes is to ensure that all the required components (components that the user needs to extract the parameters of) are in the extracted netlist. Modelling a polygon (piece of metal) as a combination of components (inductors) is not a trivial task. In many cases "correctness" of the model depends partially on the model (schematic) that the user had in mind when drawing the layout. If the model of one polygon is "incorrect", the entire extracted schematic will differ from the original schematic. The model can only be as good as the vertex choices within the polygon. We therefore allow the user to provide input into the within-polygon modelling process (as part within-polygon optimisation which will be discussed in the next section).

(a)



(b)

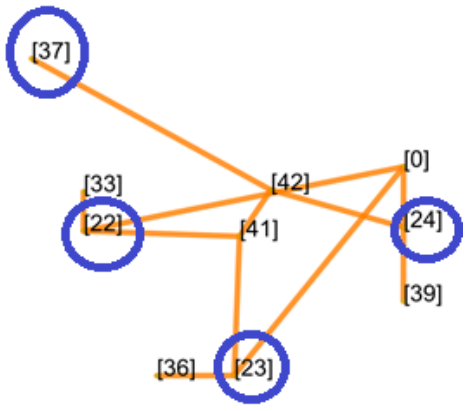


Figure 3.5: In (a), a layout is given where two user-selection-nodes are used. In (b) the extracted schematic (without ports or resistances) is given. The two user-selection-nodes in (a) correspond to nodes 41 and 42 in (b).

The user is, however, not always correct. There are cases when the original schematic does not model the layout in an effective manner. As part of effective LVS, we would like to be able to distinguish between cases where the extraction is incorrect and cases where the original schematic is incorrect.

Stellenbosch University https://scholar.sun.ac.za

In other words, if a certain inductor is included in the original schematic, the user would like to extract the inductance of this inductor; if this inductor is not included in the final extracted schematic (after layout-to-schematic is performed), either (1) the model that has been extracted is incorrect (user-selection nodes were not used correctly), (2) the model is equivalent to the original schematic but not isomorphically equal (see Appendix E) or (3) the original schematic does not model the layout correctly.

The advantage of using user-selection nodes is that the user knows which inductors he or she plans to extract. To use these nodes, the user is simply required to add nodes on either side of these inductors in certain circumstances. For example, in Figure 3.5 (b), the inductance between node 41 and 42 is required as part of the final netlist. Two user-selection-nodes (indicated by yellow arrows) are added to the layout in Figure 3.5 (a). Without these nodes, this inductance will not be extracted. More examples like this will be seen in the results and discussion chapter (Chapter 6). Only a few user selection nodes are usually required per RSFQ cell.

In Figure 3.6, one of the polygons from an SFQ-DC cell connected to a JTL can be seen. This image has been generated from information in an intermediate step in the layout–to-schematic process. The weights of each component (edge) can be seen in blue. These weights will be discussed in the next section. The small black rectangles all represent nodes. The nodes that are *inside* the bounds of the polygon (not on the border of the polygon) are *main* nodes and therefore will become the vertices of the initial graph. The two nodes that are indicated by red arrows are generated as a result of user-selection-nodes added to the layout.

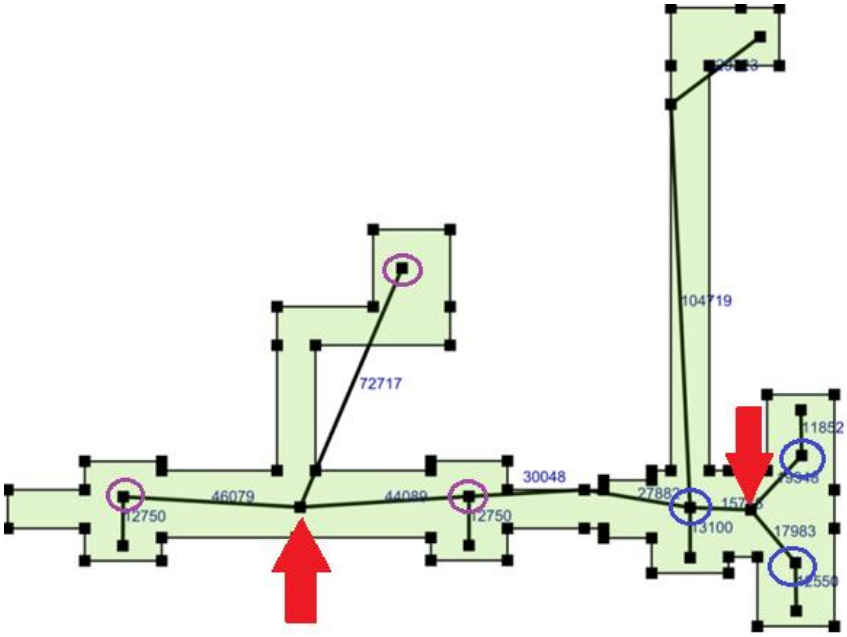


Figure 3.6: Main nodes (black squares that fall *within* the bounds of the polygon) and auxiliary nodes (black squares that fall on the border of the polygon) are connected by edges. The edge weights can be seen in blue. These edge weights are calculated by the method described in the within-polygon-optimisation algorithm. The

Stellenbosch University https://scholar.sun.ac.za

purple and blue circled nodes are the nodes that become connected to the user-selection-nodes that the arrows point to.

If these selection nodes are *not* added (as seen in Figure 3.7), the choice of components (the model) will differ from the model (in this case, configuration of edges and nodes) in Figure 3.6. It is evident that the set of generated components in Figures 3.6 and 3.7 are different.

Because user-selection-nodes play an important role in the layout-to-schematic tool, it is important to understand how and when they should be used. Selection nodes are most commonly required under the following circumstances:

- If there are 3 nodes clustered as *part* of a set of nodes on a polygon and these 3 nodes should be in a Wye configuration (Figure 3.6: purple and blue circled nodes)
- If there are 4 nodes clustered as *part* of the set of nodes on a polygon and these four nodes should be connected by 5 inductors (such as in Figure 3.5 (b): blue circled nodes).

Theoretically, these nodes are *never* required if there are 3 or fewer main nodes in a polygon. If there are only two nodes, we simply have a single component. If there are only three nodes, a Delta to Wye transform can be performed.

It is important to remember that ports and vias generate nodes – if nodes already exist on either side of a desired inductance, selection nodes are not needed to extract this component.

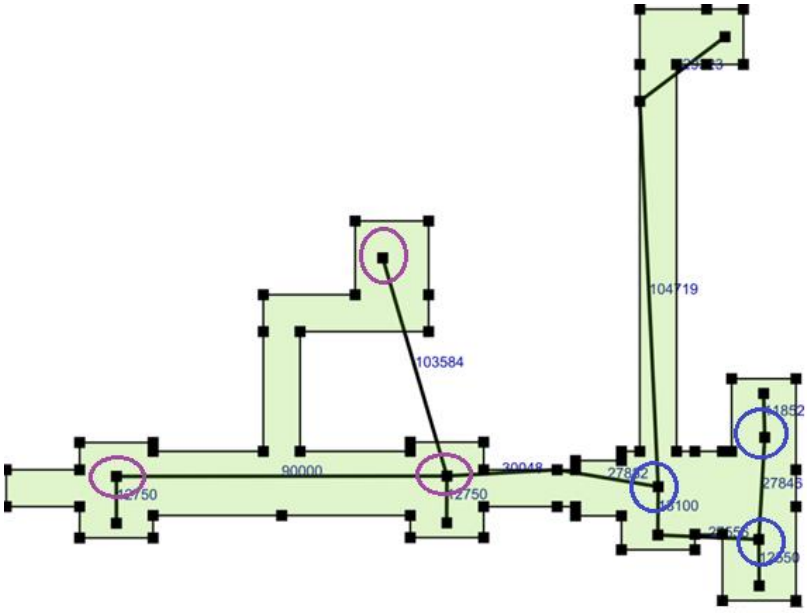


Figure 3.7: This figure is identical to Figure 3.6, except that the edges connecting blue and purple nodes (respectively) are connected differently to that of Figure 3.6 because user-selection-nodes have not been used.

3. 3 Edge (component) generation

Once all the required vertices within a polygon have been identified, the edges that connect these vertices are generated. As with any pattern recognition problem, tasks that can be easily accomplished by a *person* can be very difficult to *automate*. Connecting the edges in a way that best represents the components in the schematic is such a case.

Stellenbosch University https://scholar.sun.ac.za

The vertical components (vias, ports) are reasonably easy to automatically identify. To connect horizontal components (components that lie within a polygon on a specific layer), on the other hand, is a more difficult task. We call this task within-polygon-component-optimization. We will fist briefly discuss this problem and give the algorithm that was developed as a solution. We then explain how these horizontal components are joined by vertical components to create a graph of the entire layout.

3. 3. 1 Within polygon component selection

The Minimum Spanning Tree algorithm (MST) [101], [102], [103] is at the heart of within-polygon-component-optimization. For an introduction to the graph theory terms required to understand MST algorithms, see Appendix G. For a discussion on the execution time the MST algorithm used, see G.2.

Consider a set of vertices that are connected by a set of edges: the distance from each vertex to each other vertex (Euclidean) can be stored in a matrix. If certain sets of vertices are not connected by an edge, the corresponding matrix index for this pair of vertices will contain a 0 entry. Such a matrix is called a weighting matrix and is used in the minimum spanning tree algorithm.

In an initial version of the layout-to-schematic tool discussed in [115], the above approach was applied: Euclidean distances between these nodes were used to calculate a Minimum Spanning Tree of the graph. The edges of the MST were then converted into components.

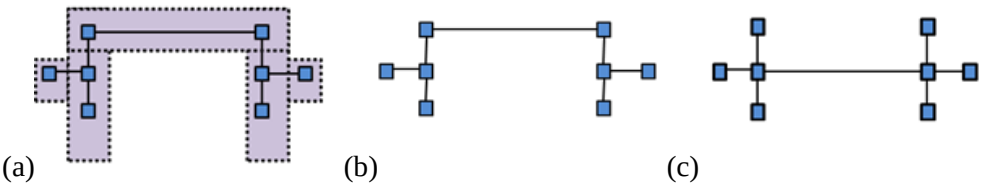


Figure 3.8: The polygon in (a) contains 8 vertices. If the edge weights of the polygon are chosen to simply be the Euclidean distances between nodes (the bounds of the polygon are not taken into consideration), (b) or (c) could be a correct MST of (a). If, however, the currently implemented algorithm is applied and only paths that remain within the bounds of the polygon are allowed, (b) is the only correct MST.

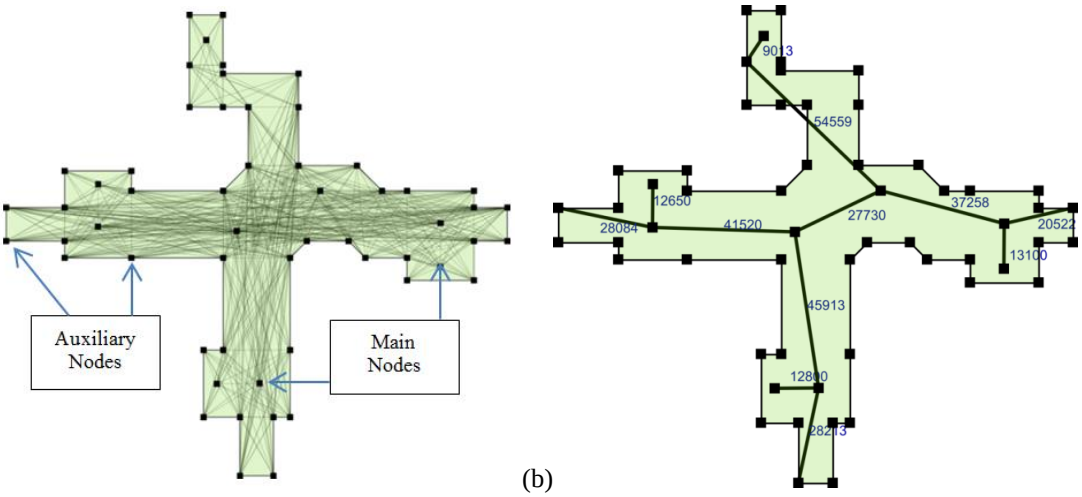
The method applied in [115] is effective for certain cells but, in general, requires many user-selection-nodes. In cases where a polygon folds back on itself, this method is particularly prone to failure because in such a cases, components are created between points that are close in absolute distance, but are often much further apart if the distance *within* the polygon is taken into account.

Furthermore, there are sometimes cases when multiple equivalent MSTs exist but only one of them remains within the bounds of the polygon (Figure 3.8). In Figure 3.8, the correct MST for the node set and bounding polygon in (a) is (b). The graph in Figure 3.8 (c) is also a valid MST, but the edges exit the bounds of the polygon. The Euclidean distance is a poor approximation of the actual current path between nodes in cases where the line representing the direct path leaves the polygon bounds.

To solve this problem, auxiliary nodes are introduced: each corner node is added to the set of auxiliary nodes (Fig 3.9). The set of all nodes includes main nodes and auxiliary nodes: this will be

Stellenbosch University https://scholar.sun.ac.za

the set of vertices for $G(E,V)$ where the edges represent components between every node and every other node. This approach is similar to the technique used by early semiconductor resistance extraction tools [116].



(a) (b)

Figure 3.9: A polygon (a) during and (b) after the within-polygon-optimisation algorithm can be seen in this figure. The auxiliary nodes and main nodes can be seen in (a). In (b), the cumulative edge weights can be seen in blue.

The algorithm applied to each polygon on each metal layer is as follows:

1. Add edges to graph by connecting each node in the set of all nodes to every other node in this set.
2. Remove all edges from G (Fig 3.8 (a)) where the path leaves the bounds of the polygon. This is done by analyzing the turning points.
3. Perform a shortest path algorithm between all vertices in G.
4. Use the shortest path distances as weights (edges) for a graph, G_w, (Fig 3.8 (b)) that only contains main nodes as vertices.
5. Perform MST on G_w. Note that the weights in Fig 3.8 (b) correspond to the cumulative edge weights on the shortest path shown in (a).
6. Create components from the MST.

The components generated from each polygon's MST, are added to the attributed graph (now a Minimum Spanning Forrest) that represents part of the extracted netlist. The edge attributes are the component names and the type of component: port, inductor or resistor. The vertex attributes are the node numbers and co-ordinates. These co-ordinates are those of the centroid of the polygon representing the via, port or user selection node.

Stellenbosch University https://scholar.sun.ac.za

3. 3. 2 Inter polygon component generation (vertical componets)

The vertices generated from the projected vias and used in the within-polygon-component-optimization algorithms are also used to create the vertical components. This is done in the same double four loop that compares each metal layer with each projected via.

Once the vertical components have been generated, we effectively have linked the Minimum Spanning Forest (that consisted of a MST for each metal polygon) to form a graph of the entire netlist. The netlist will then be simplified before it is output as a .cir file and the schematic of this circuit is generated. We will now discuss the required simplifications in the following chapter.

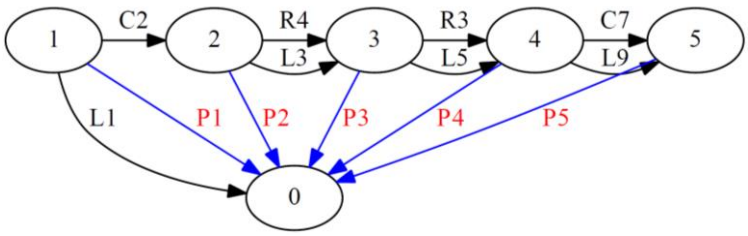
Stellenbosch University https://scholar.sun.ac.za

Chapter 4
Netlist Simplification
Algorithms and Implementation

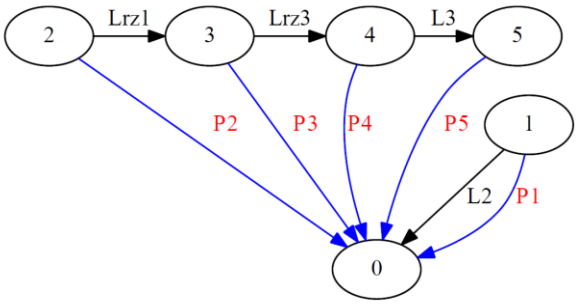
In this chapter the netlist simplification algorithms will be given and discussed. Although some of the netlist simplification is performed in Layout to Schematic tool, all simplifications will be described in this chapter and not Chapter 3.

Before parameter extraction can be performed, it is imperative that series and parallel components be simplified so as to reduce the number of unknowns. This is, however, not the only type of simplification required. Capacitance parameters are not currently a focus in parameter extraction of SFQ circuits [34] and therefore capacitors should be removed (short circuited). Other simplifications include the simplification of via components and the conversion of inductive components into impedances.

In Figure 4.1, we see the conversion of an RLC circuit (with ports) to an *InductEx*-friendly, simplified version of this circuit. The capacitors C2 and C7 are removed and the parallel and series combinations of resistors and inductors are simplified. The now inductive components are then re-named with the prefix "Lr" as specified in the *InductEx* user Manuel [114].



(a)



(b)

Figure 4.1: Figure showing an example RLC circuit (a) and its simplified version (b). In (b) we can see how impedances are named in accordance with the naming convention supplied in the *InductEx* user manual.

Stellenbosch University https://scholar.sun.ac.za

In this chapter and Chapter 5, the graph representation of this circuit (as opposed to the schematic representation seen in the previous chapter) is used to visualise the circuit. This format allows us to view any input netlist without prior knowledge about the layout that it represents.

Graphs can be represented by adjacency matricies. In this case, a type of hybrid adjacency list format is used to store the graph information. Instead of storing the number of edges connecting each vertext to each other vertex, a pointer to a list of edges containing edge atributes is stored. The polarity of the component is captured in the sign allocated to the edge weight. The upper and lower triangles of the matrix are equivalent with respect to the components that they represent and only differ with respect to the sign of the edge weights. The chosen polarity of each component will determine the half in which the positive version of of the component resides.

The via simplification problem will now be discussed, followed by parallel and series simplification. Finally, examples will be presented.

4. 1 Via simplification

The extracted netlist is initially a dense network of components – many of which are in series or parallel with one another. The complexity of this network can be reduced by simplifying the *via* components before applying the other simplification techniques.

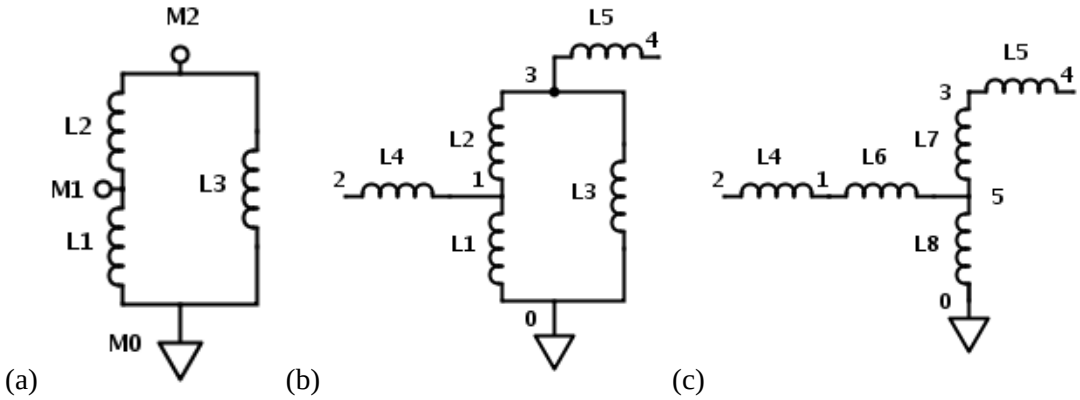
Each via is, at first, described by multiple inductive components. A typical grouping of via components can be seen in Figure 4.2 (a). When a via passes through three metal layers (M0, M1 and M2, for example), inductive components will be generated between M0 and M1, M0 and M2 and between M1 and M2. Should the vias intersect a polygon *only* on layer M2 (and not on M1), series and parallel simplifications are all is required to fully simplify the via. Given the situation in Fig 4.2 (a), L1 and L2 could be simplified in series and then simplified in parallel with L3 to provide an equivalent via inductance: L_{VIA}. This is, however, is rarely the case. In Figure 4.2 (b), we see the more common occurrence where both layers M1 and M2's via areas intersect polygons on their respective layers; some of the current flowing from M2 to M0 will typically flow along M1 (for example to another via on M1 or to a junction). In cases like this, we need an alternative to simply performing parallel and series simplifications.

Two solutions to this problem will now be explained and their implementations will be presented. Because L1, L2 and L3 form a Delta network, the Delta to Wye transform is an obvious choice. This can be seen in Fig 4.2 (c). Node 5 is added as the centre of the Wye configuration, and L1, L2 and L3 are transformed into L6, L7 and 8. After the transform, newly created series components can be simplified (in this case L4 and L6 as well as L7 and L5 can be combined).

The second solution is to remove via inductances completely and make all via nodes that correspond to the same via become a single node. In some cases, via inductances have to be re-added

Stellenbosch University https://scholar.sun.ac.za

later in the simplification process so that we never have a port without a series component in the same Kirchhoff loop. This is done as a final step before parameter extraction.



(a) (b) (c)

Figure 4.2: In (a) we see a typical via between three metal layers. If no other vias or nodes intersect the polygon in M1, L4 (as seen in (b)) will not exist: series parallel simplifications will be sufficient to simplify the via component. In (b), series parallel combinations are not sufficient to simplify the via. In (c) we see the result of the Delta to Why transform when applied to the via in (b). The series inductors L4 and L6 can now be simplified.

4. 1. 1 Delta to Wye simplification

The notation that will be used below to represent the matrix structure of the graphs is only possible for graphs or portions of graphs that do not contain parallel edges. We chose one of the edge attributes (component name or not edge weight) and display it at the matrix entry M[i][j].

In equations (4.1) and (4.2), Fig 4.2 (b) and (c) are represented in such a manner. The attribute shown in (1) is simply the component name. In (2) the edge weight is chosen. In this case – since we are referring to vias – the edges represent inductors, but in the general case they are be impedances. The values in (2) (*v*, *x*, *y*, and *x*') are therfore complex. The via components, *v*, do not have a weight based on a distance measuremnet between co-ordiante points, but are allocated the number 1 or -1. The other components's distance paramentes (*x* and *y*) are determined by the algorithms described in Chapter 3.

$$\begin{bmatrix} 0 & L1 & 0 & L3 & 0 \\ L1 & 0 & L4 & L2 & 0 \\ 0 & L4 & 0 & 0 & 0 \\ L3 & L2 & 0 & 0 & L5 \\ 0 & 0 & 0 & L5 & 0 \end{bmatrix} \xrightarrow{Delta\ to\ Wye} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & L8 \\ 0 & 0 & L4 & 0 & 0 & L6 \\ 0 & L4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & L5 & L7 \\ 0 & 0 & 0 & L5 & 0 & 0 \\ L8 & L6 & 0 & L7 & 0 & 0 \end{bmatrix} \quad (4.1)$$

$$\begin{bmatrix} 0 & v & 0 & v & 0 \\ v & 0 & x & v & 0 \\ 0 & -x & 0 & 0 & 0 \\ v & v & 0 & 0 & y \\ 0 & 0 & 0 & -y & 0 \end{bmatrix} \xrightarrow{Delta\ to\ Wye} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & v' \\ 0 & 0 & x & 0 & 0 & v' \\ 0 & -x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & y & v' \\ 0 & 0 & 0 & -y & 0 & 0 \\ v' & v' & 0 & v' & 0 & 0 \end{bmatrix} \quad (4.2)$$

Stellenbosch University https://scholar.sun.ac.za

To identify a delta configuration of vias in a netlist, the pattern vias create in the adjacency matrix can be identified. One way of identfying Delta configurations is to traverse through the matrix from one component to the next until a loop of three components is found (one of the first component's vertices is shared by the third component). The algorithm applied specifically to vias is as follows.

1. Identify a via component at M[i][j] and store the pair (i, j) in stack *S*
2. Traverse to the oposite side of the M, namely M[j][i]
3. Traverse row j:
    a. If a via component in row j can be found, and add it to *S*.
    b. If another via in j does not exist, move to the next row and pop *S*
4. Repeat steps 2 and 3 or 4 once (three components should now be on *S*)
5. If S[0].i == S[2].j (first edge shares a vertex with last edge on the stack), a Delta configuration of via components exists. If not, repete steps 1 – 5 until entire net has been searched.

Once a delta configuration has been found, extra row and colums in *M* are added and the new connections are created by adding the appropriate edge information into the appropriate M[i][j] entries as seen in (4.2). The verticies that are connected to the new central vertex of the Wye network are stored in *V*, where $V = V_1, V_2, V_3$. These vertices are found by using *S*.

For the example above, looking at (4.2) we see that:

$$S = (0,1), (1,3), (3,0)$$

and

$$S[0][0] == S[2][1]$$

therefore

$$V_1 = S[0][0] = 0, \qquad V_2 = S[1][1] = 1, \qquad V_3 = S[2][0] = 3$$

By using this simple approach, the Delta network is transformed into a Wye topology. The simplification of the remaining series components will be explained in the next section.

4. 1. 2 Removal of via components and additional vertices

The via removal algorithm is similar to the series element removal algorithm. Instead of removing the vertex that connects two series components, the vertices that connect via components to one another are removed. The series component simplification algorithm was developed before the via removal algorithm and therefore the via removal algorithm could be implemented quickly by using already written functions.

One of these functions removes all connections to a certain vertex (*i)* from the graph and connects all vertices that were previously connected to this vertex to an alternative vertex (*j)*. The input vector, *V*, is a list of other indices that are connected to *i* and *M* is the matrix representation described above.

Stellenbosch University https://scholar.sun.ac.za

The implementation if this algorithm is more complicated than depicted below because of the parallel components and complex numbers, but this detail is ignored for the algorithm description.

```
FUNCTION connect(INDEX ind, VECTOR V, MATRIX M)
    FOR all x elements in V
        IF (V[x] != ind.j)
            // new connections made
            M[ind.j][V[x]] = M[ind.i][V[x]]
            M[V[x]][ind.j] = M[V[x]][ind.i]
        END IF
        // now obsolete connections removed
        M[ind.i][V[x]] = 0
        M[V[x]][ind.i] = 0
    END FOR
    RETURN M
END FUNCTION
```

The via removal algorithm that uses *connect* can be described as follows for $i = 0\ to\ N$

1. Identify edges that connect to i at $M[i][j]$ and add their j indicies to V
2. If there is a via component in V store the i, j indices of this edge in ind (done in conjunction with step 1).
3. Call connect

4. 2 Series simplification

Parameter extraction cannot be performed if the netlist contains series impedances. The series simplification algorithm that was implemented is based on identifying series components and calling the *connect* function (as seen above) to remove the vertex between the two components and add the component values.

Refer to the Wye form of the matrix in (4.2). Row 0, 2 and 4 of the matrix each have one component contained in them (one entry per row). This indicates that the vertices represented by these rows are only connected to one component. The example netlist in Figure 4.2 (and used in (2)) is a only part of a circuit; that is why we see these "floating" components. In reality, there will very rarely only be one component connected to a node (vertex). Usually more than two components will be connected to a vertex (seen in row 4).

If the degree of $v_k = 2$ and if e_x and e_y are adjacent to v_k, then e_x and e_y are series edges. This means that when only two components are connected to one node, these components are in series. In row 1 (which represents the edges connected to v_1), we see that edges $\{v_1, v_2\}$ and $\{v_1, v_2\}$ are in series. Row 3 contains series edges that are adjacent to v_3 . These vertices (v_1 and v_3) are the vertices that need to be removed to simplify these series components. To do so, the following simple steps are followed:

Stellenbosch University https://scholar.sun.ac.za

1. Traverse through all rows and identify a row where there are no parallel components and only two components are stored in this row (for example row 1)
2. Call *connect* which performs the following main functions:
    a. The edge attributes are combined for the two components (complex addition) and stored as a new edge which could possibly be in parallel with an existing edge.
    b. The vertex is removed (all elements in the row are deleted)
3. If the newly created combined component is now in parallel with a previously existing component, mark the parallel flag.

The complex addition step is the straight forward addition of impedances. An example of series simplification can be seen in Figures 4.3 and 4.4.

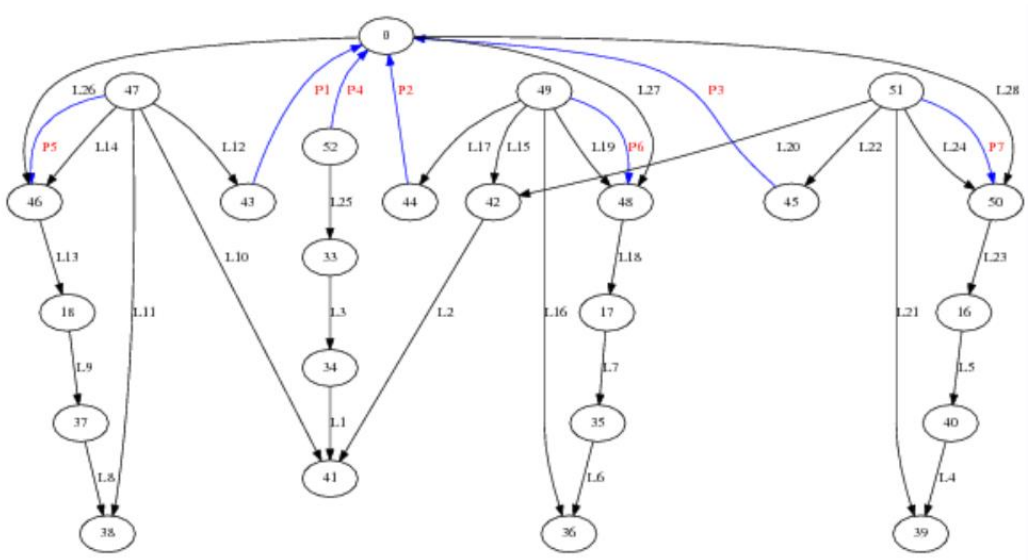


Figure 4.3: Figure showing a graph of a netlist before series simplification

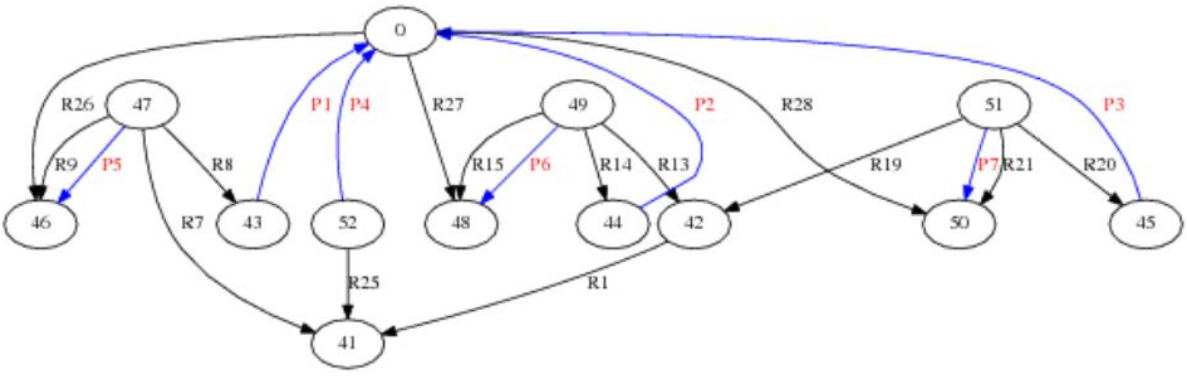


Figure 4.4: Figure showing a graph of the netlist in Figure 4.3 after series simplification but before parallel simplification.

Stellenbosch University https://scholar.sun.ac.za

4. 3 Parallel simplification

As mentioned in section 4.1, the graph of the netlist is represented by a matrix of pointers to a linked list of parallel edges. Each row represents a vertex (node) where the entries in that row represent the links (edges) to other vertices in the graph. The same can be said about the columns. The upper and lower triangles in the matrix are therefore equal except for polarity information. In Fig z we see how three parallel components of a circuit (between nodes *i* and *j*) can be stored in matrix *M*.

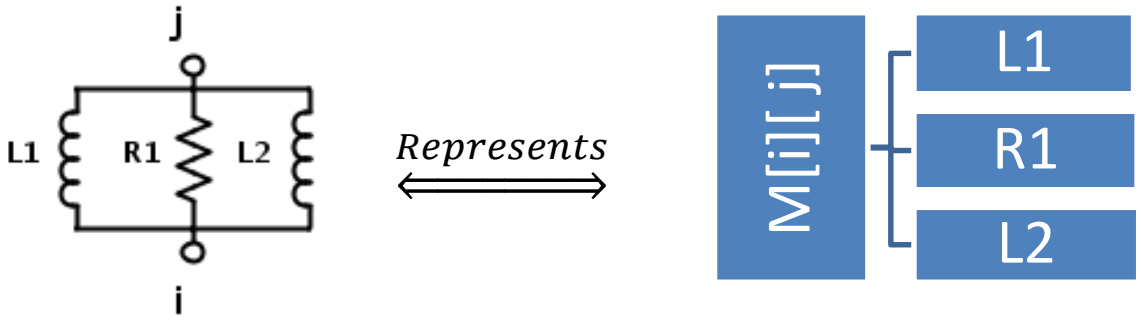


Figure 4.5: Figure showing the correspondence between the schematic and matrix representations.

From Figure 4.5 we can see that the solution to parallel component simplification is simple. If a row entry has more than one edge, these edges should be combined. Because there are usually only two components in parallel, the simplification is reduced to the following well-known equation for combining parallel impedances:

$$Z_1||Z_2 = \frac{Z_1 Z_2}{Z_1 + Z_2} \tag{4.3}$$

If more than two components are in series, the first two are combined. Thereafter, the next is combined with this combination in an iterative manner until parallel simplification is complete.

4. 4 Preparation for input to InductEx

To prepare the netlist for input into InductEx, ports in parallel with existing components replace these components. All components are then given the prefix Lr and any existing prefixes are removed. The graph of the netlist in Figure 4.5 after preparation for InductEx is seen in Figure 4.6.

Stellenbosch University https://scholar.sun.ac.za

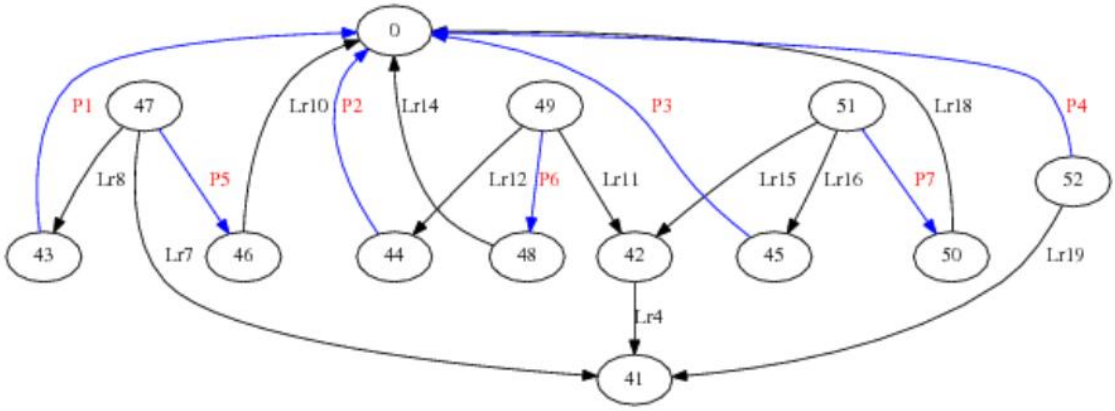


Figure 4.6: Graph of the netlist seen in Figures 4.4 and 4.5 after preparation for InductEx has been completed

Stellenbosch University https://scholar.sun.ac.za

Chapter 5
Netlist Comparison and parameter extraction
Algorithms and Implementation

Graph theory – more specifically (sub)graph isomorphism and monomorphism – can be used to solve the subcircuit matching problem. In this chapter we will focus on the implementation of these techniques.

5. 1 Subcircuit matching problem

It is well known that graph isomorphism falls into the *NP* class of problems, but it has not been proved to be *NP-complete*. Some special cases of graph isomorphism (which involve specific types of graphs, or graphs with specific restrictions), however, are always solvable in polynomial time (*P*).

Subgraph monomorphism and induced subgraph isomorphism, on the other hand, have been proved to be in *NP-complete* since they can be seen as generalisations of other *NP-complete* problems. Before reading this chapter, the reader is advised to read Appendix D if terms such as *NP*, and *NP-Complete* are unfamiliar. For an introduction to (sub)graph monomorphism and induced subgraph isomorphism, see Appendix G (specifically G.3).

In this chapter, we will first look at the exact graph matching problem of graph isomorphism. This approach can only be applied to graphs that have exactly the same number of vertices. When the vertex number differs, subgraph iso- or mono-morphism techniques are required. In this section we focus on graph isomorphism – the subgraph techniques and the implementation of these techniques will be presented in the next section in the same chapter.

5. 1. 1 Exact graph matching using graph isomorphism

The graph isomorphism problem can be solved in polynomial time for the following types of graphs that are of specific interest to us (there are many more – some of which are listed in Appendix D):

- planar graphs (such as series-parallel graphs (SPG))
- graphs with bounded degree (or valence)
- trees and k-trees with bounded *k*

Due to the structure of SFQ circuits, can we often – but not always – represent a circuit as a planar graph (but not as a SPG, since all series and parallel combinations are simplified before graph comparison). As part of the netlist comparison tool, the Boyer-Myrvold Planarity test [119] (supplied by the boost::graph library) is used prior to performing graph isomorphism. This test informs the user whether the graphs are planar or not (Figure 5.1).

Stellenbosch University https://scholar.sun.ac.za

To not limit ourselves to planar graphs but still be able to determine if two graphs are isomorphic in polynomial time, we have the option of limiting the vertex degree. SFQ cell netlists that are used as input for *only* inductance extraction (not resistance extraction) usually have a maximum degree of 4, and many have a maximum of 3 (when we exclude the ground vertex). When resistance parameters are included, some extracted circuits have vertices of degree 5 (excluding ground), but this is uncommon and usually means that an additional user selection node is required.

The ground node will often have a high degree since many components (especially port components) are connect to ground. For large graphs these ports can be removed from both graphs before the matching process is initiated.

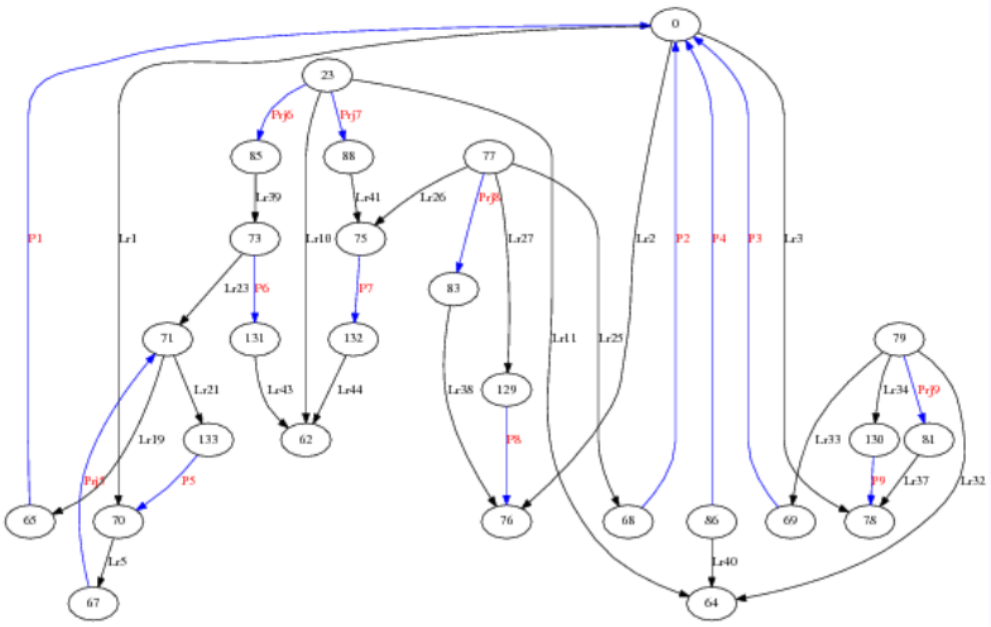


Fig 5.1: A planar graph that could easily be mistaken for a non-planar graph.

Another way to reduce the execution time could be to remove *all* ports in the attempt to transform the simplified graph into a tree or k-tree. If all simplification is done before the ports are removed, the graph will often become disjoint. This is because ports in parallel with components replace the entire parallel combinations (to comply with InductEx's input format requirements) and once the ports are removed, these parts of the graph can become disjoint. In Figure 5.2, we can see how the removal of ports that (1), replace junctions and (2), are in series with loops containing shunt resistors, cause the graph to separate into 4 separate trees (this graph is now a forest).

Stellenbosch University https://scholar.sun.ac.za

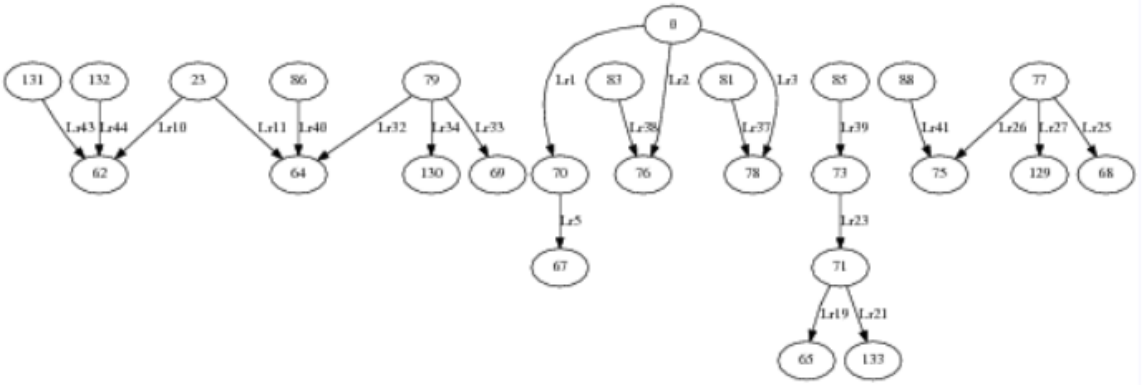


Figure 5.2: Via and series and parallel simplifications have been performed on the graph and the graph has been prepared to be used as an input InductEx: ports have replaced the componentes that were initially in parallel with them. Once the ports are removed, the graph becomes a forest containing 4 trees.

In Figure 5.3, the graph was not prepared for InductEx before ports were removed (ports in parallel with inductors had not replaced these inductors). In cases such as this, the graph will remain connected. To ensure that the graph is transformed into a tree or forest, all simplifications should be done prior to port removal.

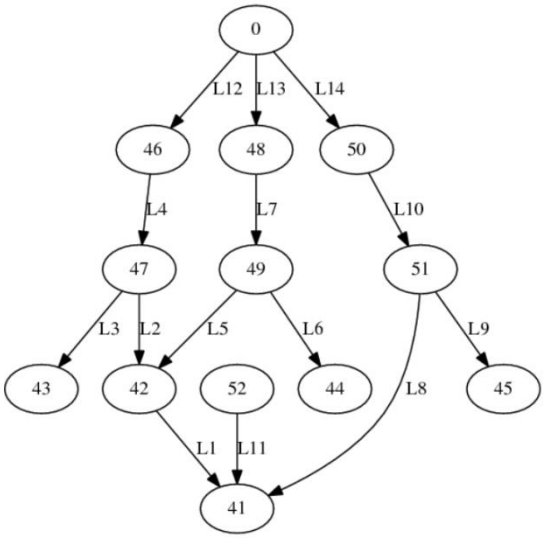


Figure 5.3: Via and SP simplification have been performed on the graph, but the graph has not been fully prepared to be used as an input InductEx: no components have been replaced by ports. Once the ports are removed, we are left with a graph containing cycles.

This approach has not been further investigated, since the graphs of most simple cells after simplification are planar or can be transformed to have bounded degree. Non-planar graphs with over 100 edges and vertices respectively, with a maximum degree of over 10 have been tested and the execution time is almost instantaneous.

The execution time for graph isomorphism (for matching typical SFQ cells) is not a limiting factor. For subgraph monomorphism and induced subgraph isomorphism, however, execution time plays a more significant role and will be discussed in the following section.

Stellenbosch University https://scholar.sun.ac.za

5. 1. 2 Subgraph isomorphsims and induced subgraph isomorphism

Subgraph monomorphism and induced subgraph isomorphism are more computationally complex than graph isomorphism [106]; they both fall into the NP-complete category even for planar graphs [106]. Subgraph monomorphism is a weaker (less constrained) morphism than induced subgraph isomorphism and is empirically harder than induced subgraph isomorphism[106] (it is NP-complete for some graph types for which induced subgraph isomorphism can be solved in polynomial time).

From an algorithm design point of view however, induced subgraph isomorphism is harder than subgraph monomorphism [105] – algorithms to solve the induced version are more complex to program.

5. 1. 3 Implementation

Once the two netlists have been converted into Boost graphs, they can be compared using Boost's *vf2_subgraph_iso* or *vf2_subgraph_mono* functions [110]. These functions can be modified to return the set of isomorphic and monomorphic mappings respectively (if such mappings exist).

Once the functions return all mappings, one of the mappings is chosen and the vertex index numbers of this mapping are used to construct either (1) a graph diagram of the subgraph that the smaller graph has been mapped to or (2) a modified version of the larger graph where the components that are not part of the chosen mapping are coloured in green (Figure 5.4). The second output is more intuitive to interpret and is therefore the default output.

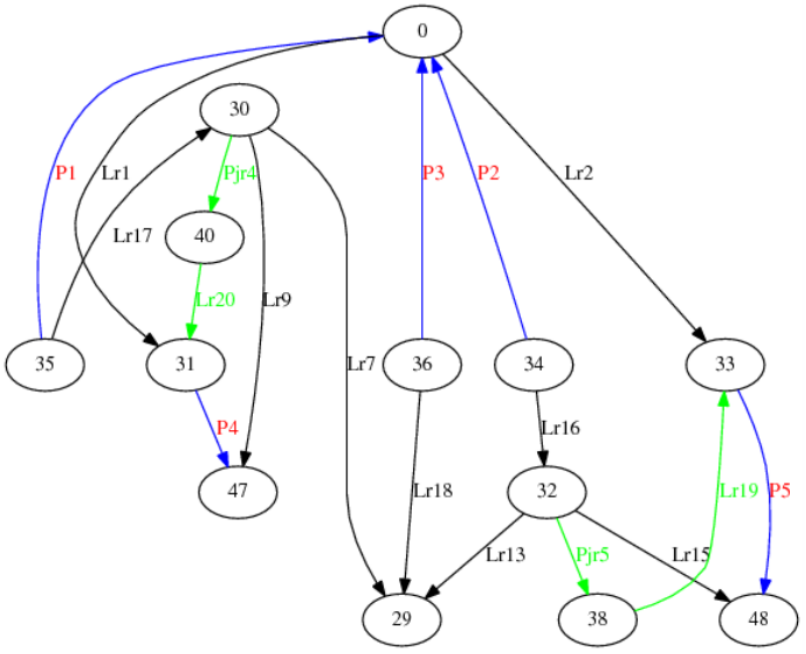


Figure 5.4: Graphic result of the netlist comparison between the netlist of an extracted JTL and the netlist of the same JTL, but where the shunt impedances (Lr19 and Lr20) in series with their respective ports (Pjr4 and Pjr5) have been removed. The blue and black components represent the chosen mapping between the two graphs after subgraph monomorphism or induced subgraph isomorphism have been performed.

Stellenbosch University https://scholar.sun.ac.za

Because the node numbers and component names will differ between the input schematic and extracted netlist in the general case, the choice of mapping is inconsequential. However, the tool has been programmed to search through all the mappings that have been found and select the mapping where the *indices* of the vertices match one another. It is important to note that we match the *indices* of the vertices and not the actual node numbers themselves.

This modification is useful when the components in the two netlists are defined in the same order (in their corresponding netlist text files). When this is the case, the mapping will show the *actual components* that were removed (in green) and not components that were not included in one of the other possible mappings. The user has the option to disable this feature to improve execution time for larger graphs.

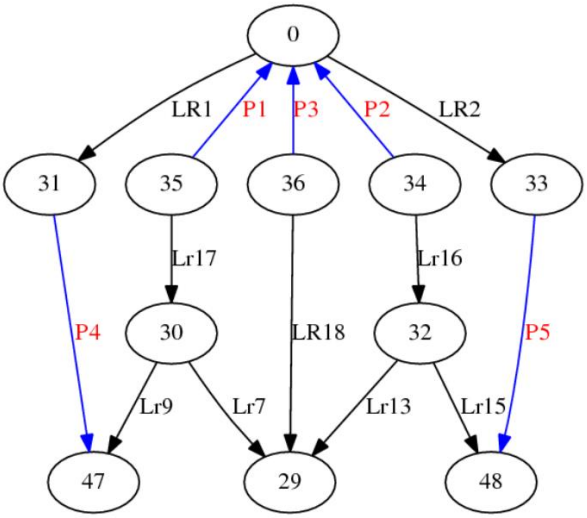


Figure 5.5: Figure showing a graph of a JTL where shunt resistors are not included. For further discussion this graph will be referred to as Graph B.

In Figure F.3 we see the graph representation of an extracted JTL: Graph A. Figure 5.5 shows the graph of a JTL but where shunt resistances (and their respective series ports) are excluded: Graph B. Figure 5.6 shows one of the induced subgraph mappings Graph B onto Graph A. We can see that the mapping that was chosen is – in fact – the 'correct' mapping (the names and node numbers correspond). This will not always be the case since the attributed characteristic of these graphs are not being exploited in the netlist comparison tool.

Even if the type of component (in this case port or impedance component) was included and attributed graph matching was performed, this would not always result in the *expected* mapping to be chosen. As mentioned earlier, this depends on the order of the components in the netlist and on the symmetry of the graphs.

Stellenbosch University https://scholar.sun.ac.za

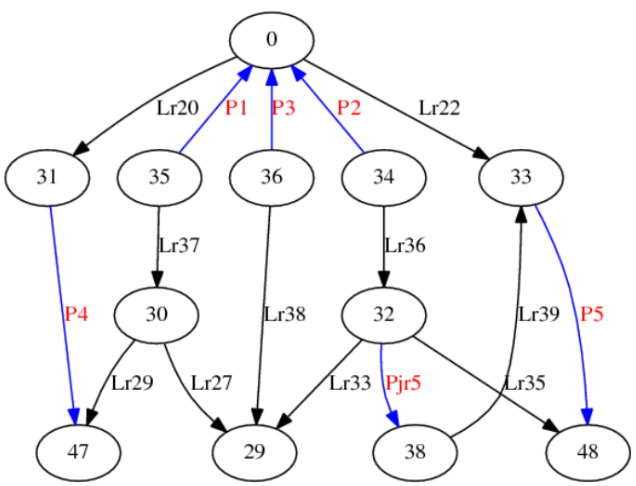


Figure 5.6: Graph C: identical to Graph A, but with the shunt impedance (Lr20) and series port (Prj4) removed.

The graphs in Figure 5.6 and 5.7 can be used to further explain this concept. In Figure 5.6 we see that only one of the junction's shunt resistor branch and corresponding port has been removed from Graph A, namely Prj4 and Lr20. In Figure 5.7 we see that Prj5 and Lr19 have been identified as the components that are not included in the mapping. This was a result of manually changing the order of the components in the smaller netlist. With the original order, Prj4 and Lr20 are identified as the components not included in the larger circuit.

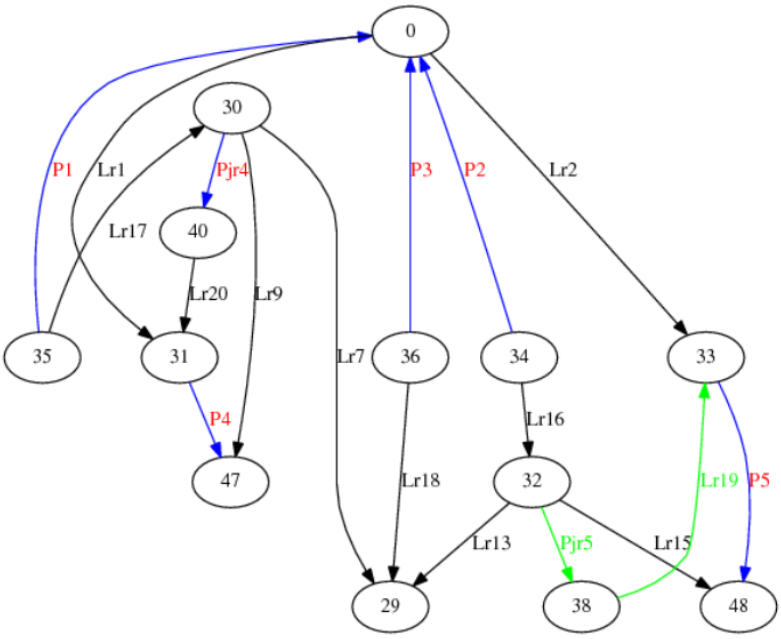


Figure 5.7: Graphic result of the netlist comparison between the netlist of an extracted JTL and the netlist of the same JTL, but where one of the shunt impedances (Lr20) and its respective series port (Pjr4) have been removed.

5. 1. 4 Subgraph monomorphism versus induced subgraph isomorphism

In the examples above, the results are the same whether subgraph monomorphism or induced subgraph isomorphism are used; this is not always the case. The theoretical difference between subgraph monomorphsim and induced subgraph isomorphism can be found extensively in graph

Stellenbosch University https://scholar.sun.ac.za

theory and computer science literature such as *Graph theory with applications to engineering and computer science* by Janet Barnett [82], *Introduction to graph theory* by Douglas West [89] and *Applied Graph Theory in Computer Vision and Pattern Recognition* [89]. We now look at the difference from a LVS netlist comparison point of view.

It is important to note that the smaller graph is often the graph generated from the original schematic since the extracted graph can contain parasitic that were not originally included in the schematic. If there are mistakes in the layout or input netlist, however, this will not necessarily be the case. If there has been an open or short circuit mistake in either the layout or schematic, either of the graphs could be the smaller one. We therefore do not make any assumptions but rather run the isomorphism and/or monomorphism checks so as to cater for either graph being the smaller graph.

To answer yes to the induced subgraph isomorphism decision problem, components in the original schematic all have to exist in the extracted netlist (even parallel edges). Furthermore, if no component connects two specific nodes in the original schematic, there cannot be a component between the corresponding nodes in the extracted netlist. In subgraph monomorphism we do not have as strong restrictions.

5. 1. 4. 1 User input

To perform netlist comparison, the user can select from the following options, depending on the netlist size and on his or her requirements.

1. to use the (a) induced subgraph isomorphism test, (b) the subgraph monomorphism test or (c) both
2. to (a) simply solve the *decision problem(s)* or to (b) calculate *all* morphisms
3. to (a) simply chose the *first* morphism to view graphically or to (b) chose the morphism that maps the original graph indices to one another in (possibly) the best way (and print all mappings to the text file)

Flags are used to allow the user to choose a netlist comparison strategy. For example: --1a --2a --3a, would be an appropriate selection for very large, complex graphs.

5. 1. 4. 2 Recommended usage strategy

If the graphs are planar and have fewer than 50 vertices (which is usually the case) the default selections are (c) for option 1 and (b) for option 2 and 3. This decision has been made because the execution time for planar graphs of this size will in most cases be only a number of seconds in total. The additional information (all mappings and the total number of mappings) can therefore be provided to the user without sacrificing noticeable execution time. The user can, if need be, over-ride these defaults by explicitly choosing the other selections for the three options.

If the graphs are not planar, the size of the graphs, as well as the graphs' characteristics, will give an indication to the user as to which selections are advisable. For non-planar graphs it is advisable to

Stellenbosch University https://scholar.sun.ac.za

use the induced subgraph isomorphism tests (selection (a) option 1) and search for only one mapping to view graphically (selection (a) for 2 and 3).

If the information is insufficient (no mappings are found), the subgraph monomorphism test should be used and these results should be viewed graphically. If the execution time takes longer than a minute, ports connected to the ground node should be removed from both graphs and the process should be repeated. Removing these ports connected to ground reduces the maximum vertex degree (as mentioned above) and lowers execution time.

If port components connected to ground are removed, nodes with a degree of 1 will be present in both graphs (in most cases) which means that the number of induced subgraph isomorphism mappings (if any) will usually be significantly less than the number of subgraph monomorphisms (if the graphs have a high level of symmetry). It is therefore advised to first attempt run the induced subgraph test and only resort to the subgraph monomorphism test if the information obtained is insufficient to perform meaningful LVS.

In the following chapter (Results and Discussion) we use examples to compare subgraph monomorphism to induced subgraph isomorphism. In cases where we have *one edge* faults such as short circuit or open circuit faults in either the layout or input schematic, induced subgraph isomorphism will not be able to identify any mapping (since the morphism is *vertex* induced). Subgraph monomorphism, however, will identify the edge that is included in the larger graph but not in the smaller graph. It is important to note that it may not be this specific edge that has been removed in cases where more than one mapping exists! Furthermore, if multiple errors exist in the layout *and* schematic, we will often not find any mappings.

This concept will be explained by the aid of examples in Chapter 6, section 4.

5. 4 Parameter extraction

Parameter extraction is the final step of LVS. This section will give a background to parameter extraction. For a more comprehensive overview, the following sources can be consulted [113], [120], [121], [122]. To understand the linear algebra concepts discussed, the book, *Numerical Linear Algebra* [129] is strongly advised.

The three parameters that are typically required in the design of SFQ circuits are junction critical current, inductance (especially of squid loops) and resistance [120]. Junction critical current and resistance are easy to calculate.

Junction area is given by the junction critical current, I_c, divided by the current density, J_c. In terms of parameter extraction, the actual junction critical current for each drawn junction can simply be calculated by calculating the junction area since J_c for a specific process is known [113].

Resistance can be calculated geometrically with reasonable accuracy for simple structures, but for more accurate and reliable results, boundary methods are used. In Chawla and Gummel's paper [123], the well-known Cauchy's integral is used to describe the complex potential function,$\varphi(Z)$, at a point

Stellenbosch University https://scholar.sun.ac.za

Z' in a region R and bounded by the boundary, C, as

$$\varphi(Z') = \frac{1}{2\pi i}\oint \frac{\varphi(Z)}{Z-Z'}\,dZ\,, \tag{5.1}$$

where Laplace's equation is satisfied in R, and $\varphi(Z)$ is defined on the boundary (and is analytical in C and R) [123]. The complex potential function has the potential function, $V(x,y)$ as the real part and the integral of current density, $U(x,y)$, as the imaginary part such that [123],

$$\varphi(Z) = V(x,y) + iU(x,y) \tag{5.2}$$

The extraction of inductance is harder than that of junction critical current and resistance – especially for modern SFQ circuits [113]. Inductance is sensitive to small changes in layout. Furthermore, inductance is a very important parameter since it not only affects margins directly, but also indirectly (changes in inductance result in changes to bias current distribution through circuit branches which in can affect the junction bias and margins).

5. 4. 1 Inductance extraction

Inductance in superconductive circuits can be divided into two categories, namely geometric (a function of all the return paths but mostly the shortest return path [120]) and kinetic (a function the current flowing through the cross section and also of the London penetration depth [124]). Using London and Maxwell's equations it follows that [125],

$$\left(\frac{\mu_0}{\lambda^2} - \nabla^2\right)\frac{d}{dt}\boldsymbol{H} = 0 \tag{5.3}$$

governs a perfect conductor where the penetration depth is written as [125],

$$\lambda \equiv \sqrt{\frac{\Lambda}{\mu_0}} \tag{5.4}$$

Inductance can be calculated analytically for ideal, or close to ideal, situations. In the ideal case, we have infinitely long micro-strip lines over an infinite ground plane. Practical SFQ circuits, especially the more complex LSI circuits that use current re-cycling and mutual coupling, do not fall into this category.

A numerical solution is therefore required for inductance extraction. FastHenry, a multi-pole accelerated 3D inductance extraction program, was made available by M. Kamon [126] in 1994 and adapted to support superconductivity in 1996 by S. Whitely [51]. FastHenry is not, however, easy to use for complex inductive networks.

InductEx, a pre- and post- processor to FastHenry, extracts inductance and resistance parameters in multi-terminal networks [113]. InductEx's superior accuracy, execution time an ease of use has caused it to become one of the most popular parameter extraction tools for SFQ circuits[29]. In the following section, the process of resistance and inductance extraction is explained as a final step in LVS.

Stellenbosch University https://scholar.sun.ac.za

5. 4. 2 Impedance extraction

A description of impedance extraction as performed by InductEx can be found in C. J. Fourie's 2014 paper entitled "*Full-gate verification of superconductive integrated circuit layouts with InductEx*" [113]. In the following section, we give a more detailed explanation of the linear algebra techniques used in InductEx as part of LVS.

Let us define n as the number of unknowns (components), c as the number of unique cycles and p as the number of ports. In this discussion ports are not seen as components, but rather as sources of a voltage. We have three sets, namely the set of components $= \{Z_1, Z_2, Z_3, \dots, Z_n\}$, the set of ports $= \{P_1, P_2, P_3, \dots, P_p\}$ and the set of cycles $= \{C_1, C_2, C_3, \dots, C_c\}$.

To calculate the parameter values for each component, FastHenry is used to calculate the port currents. This is done by energizing one port at a time with a voltage of 1V and shorting all of the other ports. The branch currents are then iteratively solved by InductEx using these port currents. For every port P_i, the port current is given through P_i and also all the other (shorted) ports, giving us *pxp* known port currents. We define the branch current I_{ij} as the branch current passing through Z_i when P_j is energized with 1V and all other ports are short circuited.

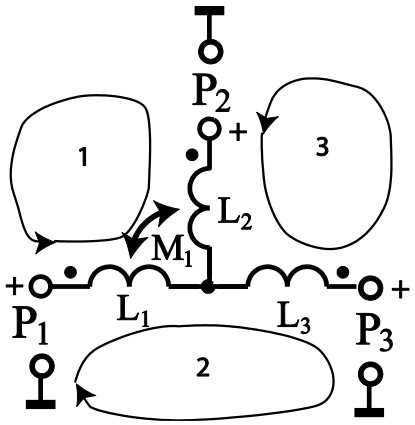


Figure 5.8: Diagram depicting the three cycles found in the in the example netlist discussed in this section.

In Figure 5.8, we see a network with three inductors, three ports and one mutual inductance. There are three cycles in this example. In equation (5.5), the branch current matrix as a result of P_1 being energised is given. Each row represents the contribution of one cycle. In (5.6) we see that the inner product of I_{P1} and Z (the impedance matrix) gives the vector of voltages (all either 1 or 0) to satisfy Kirchhoff's Voltage Law (KVL). Each row in the branch current matrix represents a cycle; I_{Pi} will therefore always be a cxn matrix.

$$I_{P1} = \begin{bmatrix} I_{11} & -I_{21} & 0 & (I_{21} - I_{11}) \\ I_{11} & 0 & I_{31} & I_{21} \\ 0 & I_{21} & -I_{31} & I_{11} \end{bmatrix}, \tag{5.5}$$

$$x = I * Z\,, \tag{5.6}$$

We can combine these branch current matrices into a matrix, I, that contains all the branch

Stellenbosch University https://scholar.sun.ac.za

current matrices. For this example, we have,

$$I_{tot} = \begin{bmatrix} I_{P1} \\ I_{P2} \\ I_{P3} \end{bmatrix}, \tag{5.7}$$

where I_{tot} is an $(cxp)xn$ matrix.

The system of linear equations is shown below, where x_{PiCj} represents the voltage contribution (either 1 or 0) of P_i to the KVL equation for cycle C_j.

$$\begin{bmatrix} x_{P1C1} \\ x_{P1C2} \\ x_{P1C3} \\ x_{P2C1} \\ \vdots \\ x_{P3C3} \end{bmatrix} = \begin{bmatrix} 1+j0 \\ 1+j0 \\ 0 \\ -1+j0 \\ \vdots \\ -1+j0 \end{bmatrix} = \begin{bmatrix} I_{11} & -I_{21} & 0 & (I_{21}-I_{11}) \\ I_{11} & 0 & I_{31} & I_{21} \\ 0 & I_{21} & -I_{31} & I_{11} \\ I_{12} & -I_{21} & 0 & (I_{22}-I_{12}) \\ \vdots & \vdots & \vdots & \vdots \\ 0 & I_{23} & I_{33} & I_{13} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \tag{5.8}$$

Let m be the total number of equations in the system. As shown before, the number of equations is the number of ports times the number of cycles: $m = pxc$. Also, the I_{tot} matrix will be re-named A.

In this example, since we have more equations than unknowns, we have an over-determined system. SVD is then performed on matrix A. When the model (input netlist) and the layout agree, we have exactly n singular values. These values are not close to 0 and therefore the A matrix is of full rank. In cases when the netlist does not model the layout sufficiently, some of the singular values are very close to 0. This indicates that the matrix is almost singular and not of full rank. We can therefore use the singular values or condition of the matrix to given us additional LVS information as a final check after netlist comparison has been performed.

This concept is currently under further investigation, and we aim to include additional LVS information to the user as part of InductEx's output.

If the model has, however, been correctly confirmed by the netlist comparison tool, InductEx will output the resistance and (more importantly) inductance values for the components in the input netlist. InductEx also gives an indication of the variation from the estimated input values. If the variation is sufficiency small, and the circuit functions correctly when simulated with the extracted values, we say that LVS has been successfully completed.

Stellenbosch University https://scholar.sun.ac.za

Chapter 6
Results and Discussion

In this chapter, results and outputs for some commonly used RSFQ cells will be given and discussed.

In the first section of this chapter, Section A, we will limit our discussion to cells for which the extracted netlist and the schematic agree (the graphs of the netlists are isomorphic). We will start by discussing the results for a Josephson Transmission Line cell – one of the most prevalent RSFQ cells. The results for two SFQ Splitter cells (one Hypres and one Fluxonics) will then be discussed. The Hypres cell will be used to point out the limitation of this toolkit with respect to mutual coupling. Hereafter, the results for a Fluxonics Confluence Buffer cell will be given and discussed.

In the second half of the chapter, Section 2, the success of the netlist comparison tool will be discussed (with respect to execution time and expediency) using examples where subgraph matching is required.

6. 1. Section A: Examples where graphs are isomorphic

6. 1. 1 Josephson Transmission Line (JTL)

The layout to schematic as well as netlist comparison results will be given for a standard Fluxonics JTL. The extracted netlists differ depending on whether resistance extraction is required in addition to inductance extraction. When resistance extraction is required, the port placement on the layout itself also differs from pure inductance extraction. We will first look the inductance extraction case and then at resistance plus inductance extraction.

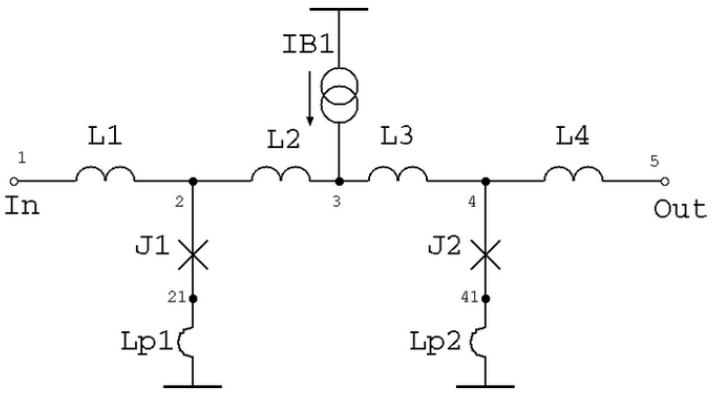


Figure 6.1: Standard JTL schematic – as seen on the Fluxonics website.

It is important to note that InductEx is not primarily a resistance extraction tool but an inductance extraction tool. The advantage of extracting resistance as well as inductance by using the layout to schematic tool in conjunction with the netlist comparator and Inductex, however, is that layout errors (such as omitting a required via) will quickly be picked up by the netlist comparison tool even if these errors are in a resistive branch of the circuit [113].

Stellenbosch University https://scholar.sun.ac.za

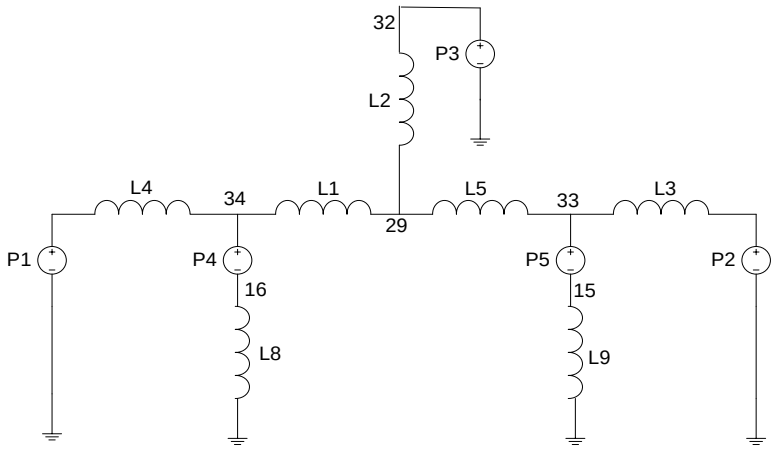


Figure 6.2: Modified schematic to account for ports that replace Josephson Junctions.

6. 1. 1. 1 Inductance extraction (shunt impedance ignored)

In Figure 6.1, we see a typical JTL schematic (as given on the Fluxonics website[127]). To perform inductance extraction, the schematic needs to be modified to include ports (as seen in Figure 6.2).

The layout of the JTL can be seen in Figure 6.3. A user selection node has been added to one of the metal layers (M2) so that the within-polygon-optimisation algorithm discussed in 3.3.1 can function correctly.

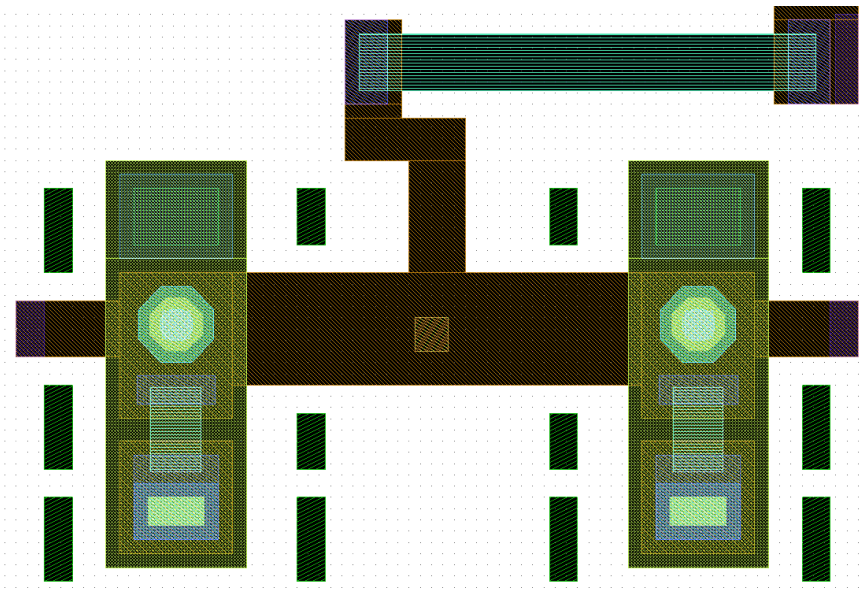


Figure 6.3: Layout of standard Fluxonics JTL. One user selection node has been added to the centre of the polygon on the metal layer M2.

The extracted un-simplified schematic (after via simplification but before other simplifications) can be seen in Figure F.1. The graph representation of the same netlist can be seen below it in Figure F.2. The series components are not evident in the schematic representation but can be clearly seen in the graph representation. Once the other simplifications have been performed, the netlist can be compared to the original input netlist.

Stellenbosch University https://scholar.sun.ac.za

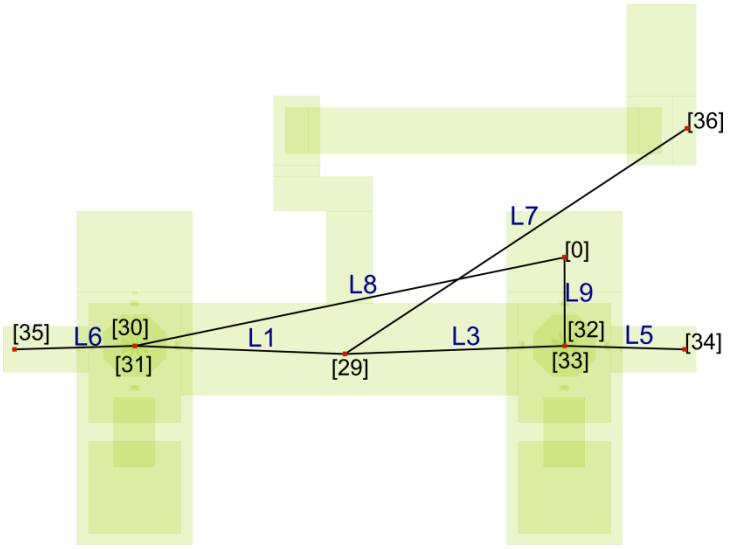


Figure 6.4: Simplified schematic of JTL. This schematic was extracted from the layout in Figure 6.3.

The simplified schematic and netlist can be seen in Figure 6.4 and 6.5. The original input netlist is shown in Figure 6.6. In Figure 6.5, port P1 is connected between ground and node 35 while port P4 connects node 30 to ground. Ports P1 and P4 represent the JJs and therefore should ideally either both be on the ground side of the inductor that they are in series with *or* both connected to node 30 (in the same format as the input netlist in Figure 6.6). Because graph attributes (such as type of component) are not used in the isomorphism check, this plays no role in netlist comparison.

When the netlists are compared, the solution file states that both graphs are planar and that the graphs are isomorphic. The solution file also tells us that there are 8 automorphisms for these two 9 vertex graphs (that are effectively the same graph).

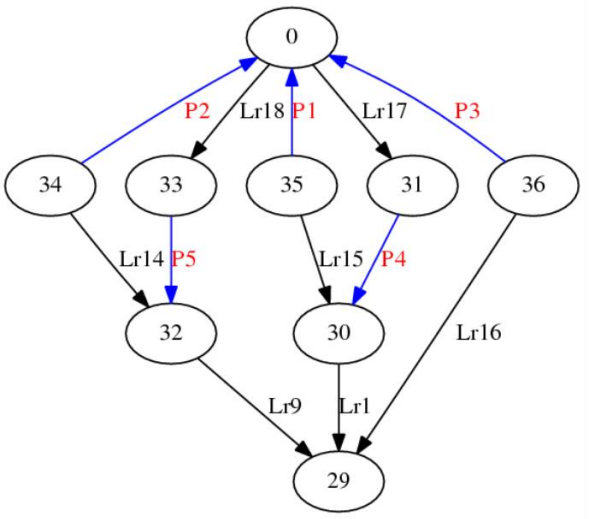


Figure 6.5: Graph representation of the simplified netlist extracted from the layout in Figure 6.3. This is the graph representation of the JTL's schematic in Figure 6.4.

Stellenbosch University https://scholar.sun.ac.za

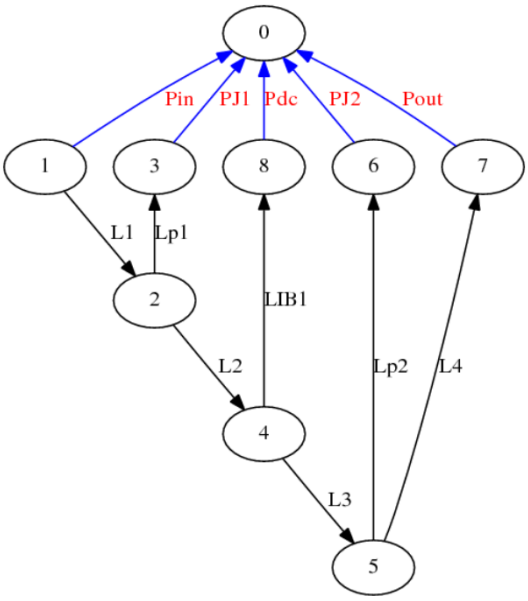


Figure 6.6: Graph representation of original input netlist. The extracted netlist's graph is isomorphic to this graph. Both have 8 automorphisms.

6. 1. 1. 2 Inductance plus resistance extraction

The schematic representation of the extracted netlist (Figure 6.7) looks similar to the schematic in Figure 6.2. The graph representation, however, reveals the hidden horizontal ports in Figure F.3.

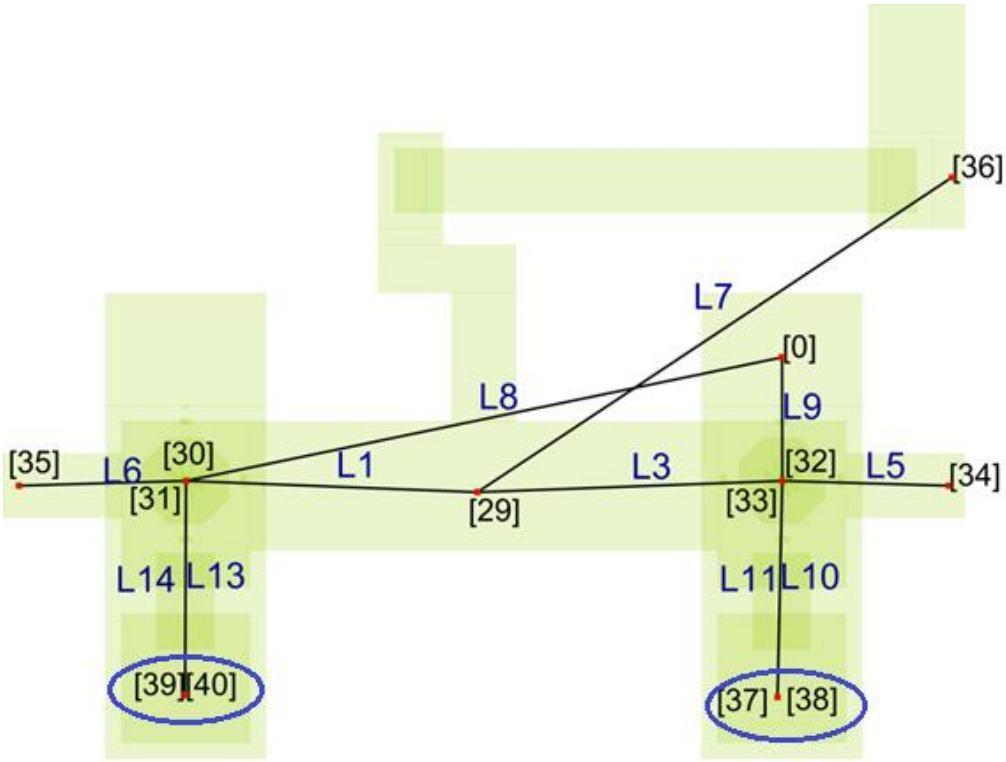


Figure 6.7: Simplified schematic of JTL extracted from layout in Figure 6.3. Ports placed on the vias between M1 and M2 (circled) so as to add a port to the cycle containing the shunt resistor.

The input schematic can be seen in Figure F.4. The input and extracted schematic are isomorphic and also have a high level of symmetry (as in the previous example). The entire netlist comparison

Stellenbosch University https://scholar.sun.ac.za

process (including layout to schematic) takes under a second.

We are now assured that the layout and schematic agree and can use the original netlist to perform parameter extraction with InductEx. The parameter extraction results for a slightly improved JTL layout (but with an equivalent netlist) can be seen in section F.2.1.

The parameter values have been back-annotated into the formal schematic and can be seen in Figure F.5.

6. 1. 2 RSFQ Splitter cells

The layout to schematic results for two SFQ Splitter cells will now be given and discussed. The first Splitter cell has been designed using the Fluxonics process and the second using a Hypres process. In the first example we extract the netlist with respect to inductive components alone and in the second (Hypres) example we include the resistance parameters.

The extraction results for the Fluxonics cell where the resistance plus inductance components are included results in successful LVS and the graphs look similar to that of the Fluxonics JTL (when resistance and inductance parameters are included in the model). We will therefore not discuss these results but rather focus on the Hypres cell that includes mutual coupling.

The schematic of both SFQ Splitter cells are of the same form when resistances are ignored; this schematic can be seen below in Figure 8.

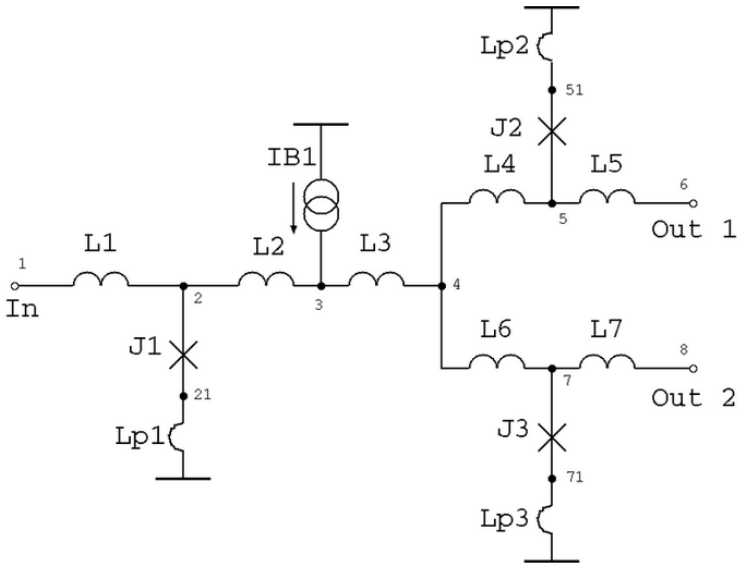


Figure 6.8: Standard SFQ Splitter schematic – as seen on the Fluxonics website.

6. 1. 2. 1 Inductance extraction of Fluxonics SFQ Splitter cell

The layout of the SFQ Splitter cell that has been tested can be seen in Figure 6.9. Two user selection nodes have been included so as to extract the inductance between these two nodes. This inductance corresponds with L3 in the standard schematic seen in Figure 6.8 and the extracted inductance L1 seen in the output schematic in Figure 6.10 (a). When the two user selection nodes are added, the extracted netlist agrees with the input netlist (resulting in 16 automorphisms).

This cell has been chosen as an example because the placement of user selection nodes plays an

Stellenbosch University https://scholar.sun.ac.za

important role in correct LVS (when using this toolkit). If no user selection nodes are used, the extracted netlist is completely incorrect. In Figure 6.10 (b) we see the extracted schematic for this cell when no user selection nodes have been included. If such a model is input into InductEx, the A matrix used in SVD is almost singular (the condition of the matrix is poor).

In cases where the condition of the A matrix is poor, results for all the other inductors in the model cannot be guaranteed to be even nearly correct. When the condition of the matrix is poor, the user should be warned to re-look at how user selection nodes are used (if the extracted netlist is used directly as an input to InductEx). The next version of InductEx will include such a warning.

If only one user selection node is used, the results will also differ slightly from the model where two are used, since L3 (Figure 6.8) will not be included in the model. Because L3 is "small", the effect on the other inductors will not be too great (about 10%). This is an example where four nodes connected to one inductor gives a less accurate result than two Wye connections.

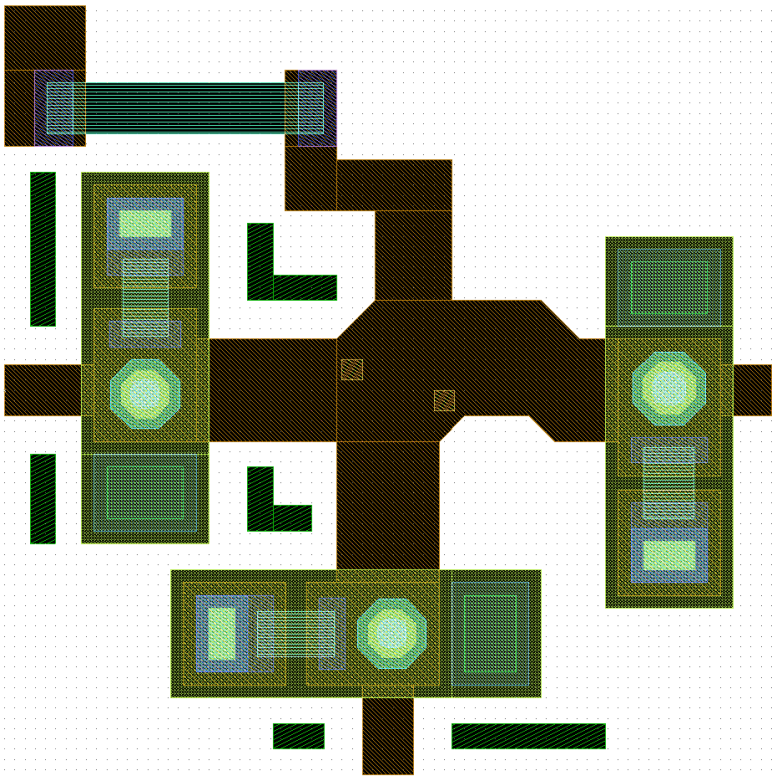
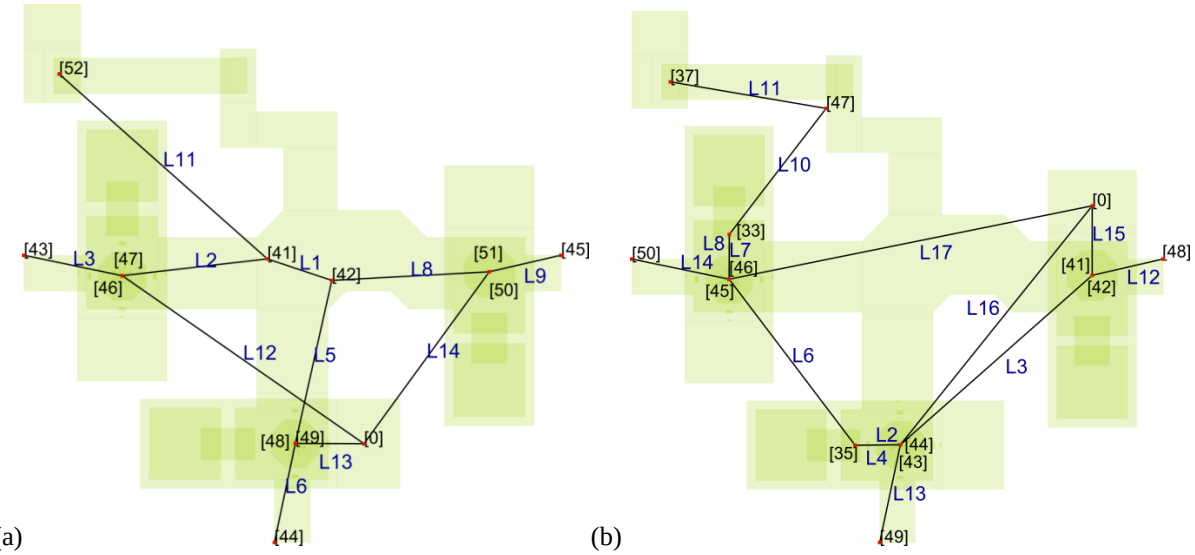


Figure 6.9: Layout of standard Fluxonics SFQ Splitter cell. Two user selection nodes are added to the centre of the polygon on the metal layer M2.

Stellenbosch University https://scholar.sun.ac.za



(a) (b)

Figure 6.10: The simplified schematic of an extracted SFQ Splitter cell with correct user selection nodes can be seen in (a). In (b) we see the result of layout to schematic if user selection nodes are not used. The layout of this cell can be seen in Figure 6.9.

6. 1. 2. 2 Inductance and resistance extraction of HYPRES SFQ Splitter cell

This specific Hypres Splitter cell layout (Figure 6.11) has been chosen because each of the three Josephson Junctions has been grounded differently. This example allows us to test our layout to schematic tool using three different ways of modelling junction and shunt resistance branches.

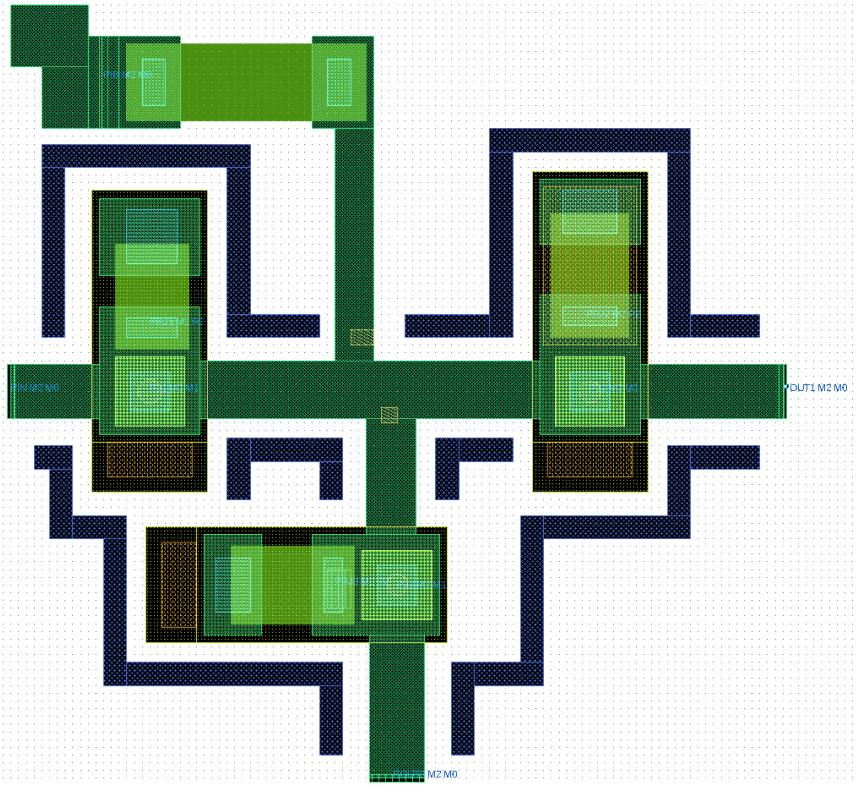


Figure 6.11: Layout of a Hypres splitter cell where two user selection nodes have been added. Ports for resistance and inductance extraction have been included.

Stellenbosch University https://scholar.sun.ac.za

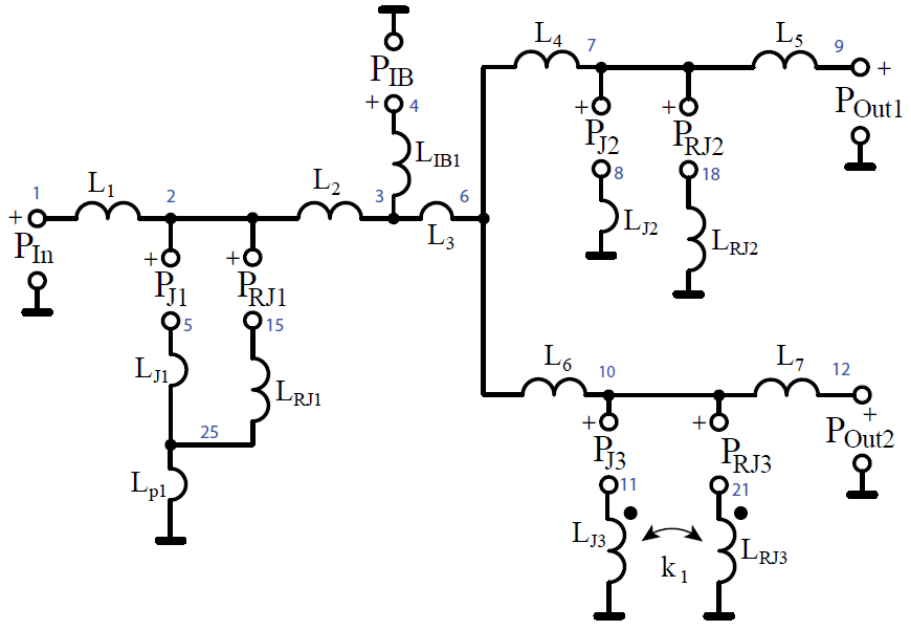


Figure 6.12: Schematic representation of the specific Hypres SFQ splitter cell seen in Figure 6.11. Mutual coupling can be seen between two of the inductors.

The first junction is grounded close to the junction itself (it is grounded just below (to the South of) the junction when looking at the layout from the top). The shunt resistor is laid out above (to the North of) the junction. Each branch (the junction branch and shunt resistor branch) has an impedance that can be modelled separately (L_{J1} and L_{RJ1}respectively in Figures 6.12 and 6.13). These impedances are connected at node 25 (Figures 6.12 and 6.12) and are connected to ground via an inductance L_{p1}.

For the second junction, the junction and shunt resistor are both grounded by a hole in the isolation layer underneath the shunt resistor. Each branch (junction and shunt resistor branches) can be modelled separately ($L_{J2}$and L_{RJ2}). They are connected directly to ground unlike for the first junction.

For the third junction, there are two ways that the grounding can be modelled. The model included in the example includes strong mutual coupling between the two branches (L_{J3} and L_{RJ3}). Alternatively, this junction could be modelled in the same way as the first junction. In Appendix E we see the equivalence between these two models.

In Figure 6.13, the circled components (L_{J3} and L_{RJ3}) are coupled. Unfortunately this graph representation of the netlist (as well as the internal graph model) does not support the viewing of mutual coupling. This is one of the current drawbacks of this toolkit (as discussed in Section 2.5.3.2). There are, however, ways of working around this problem for specific circuits.

Stellenbosch University https://scholar.sun.ac.za

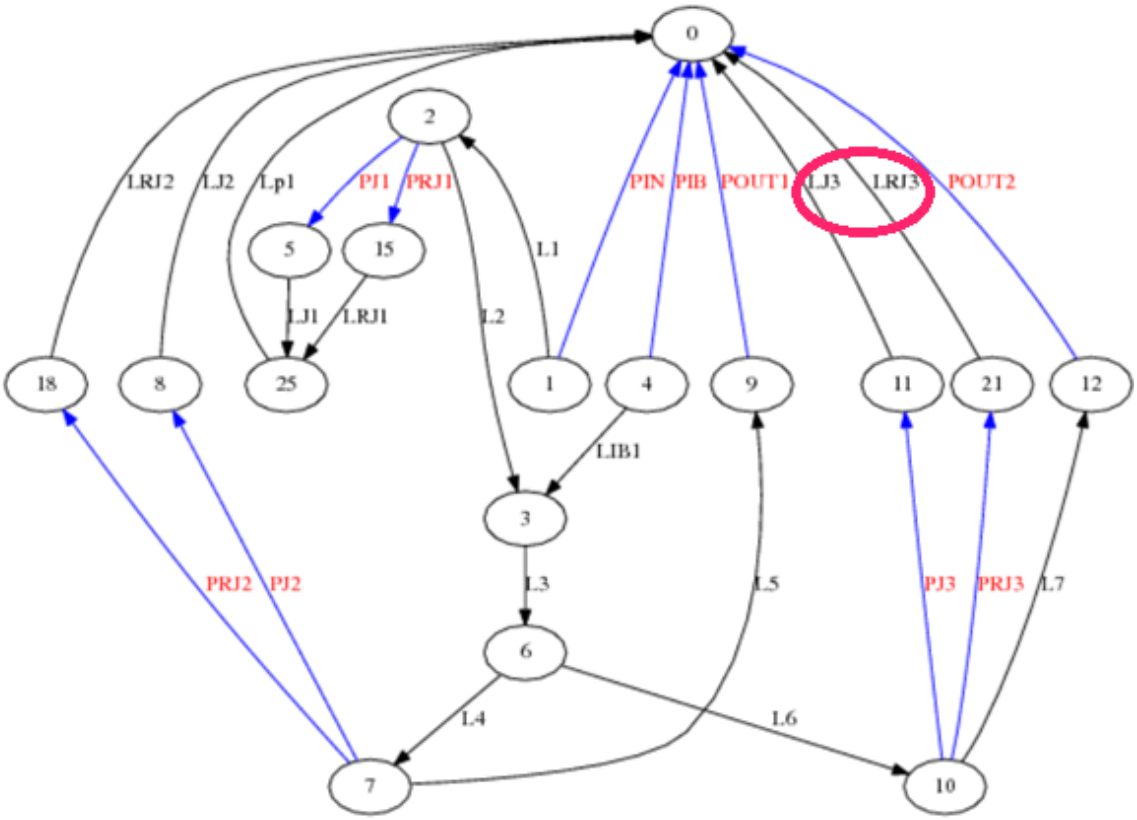


Figure 6.13: Schematic representation of the input Hypres SFQ Splitter. Mutual coupling exists between the components circled by the pink oval (LJ3 and LRJ3). Representing mutual coupling graphically is not part of the scope for this thesis.

The first option is to exclude the mutual coupling in the input schematic and manually modify the extracted netlist to ground the two branches directly. This would require the following modifications to the pink circled region of Figure 6.14:

- node 22 should be removed
- 77 and 52 should be connected directly to ground via Lr31 and Lr6 respectively.

Netlist comparison can then be performed between the two modified netlists. This task may seem tedious, but the fact that the extracted ports will be labelled the same as the input netlist's ports will simplify the process.

The second option is to transform the input netlist *before* using the toolkit. All mutual coupling branches can be modelled by using Why configurations. Layout to schematic and netlist comparison can then be performed directly and if the netlists agree, the original netlist (now with the original mutual coupling included) can be used as an input to InductEx.

Where the second option *can* be used (the extracted model should be visually inspected first to be sure of this), it is simpler to apply than the first option. Neither of the two options will work with all mutual coupling cases (and never in AQFP-type cells). For the example above, however, both techniques have been applied and result in successful LVS.

Stellenbosch University https://scholar.sun.ac.za

The un-simplified extracted netlist and the simplified schematic can be seen in Figures F.6 and F.7 for reference. The input netlist and solution can be found on the InductEx website under the advanced example.

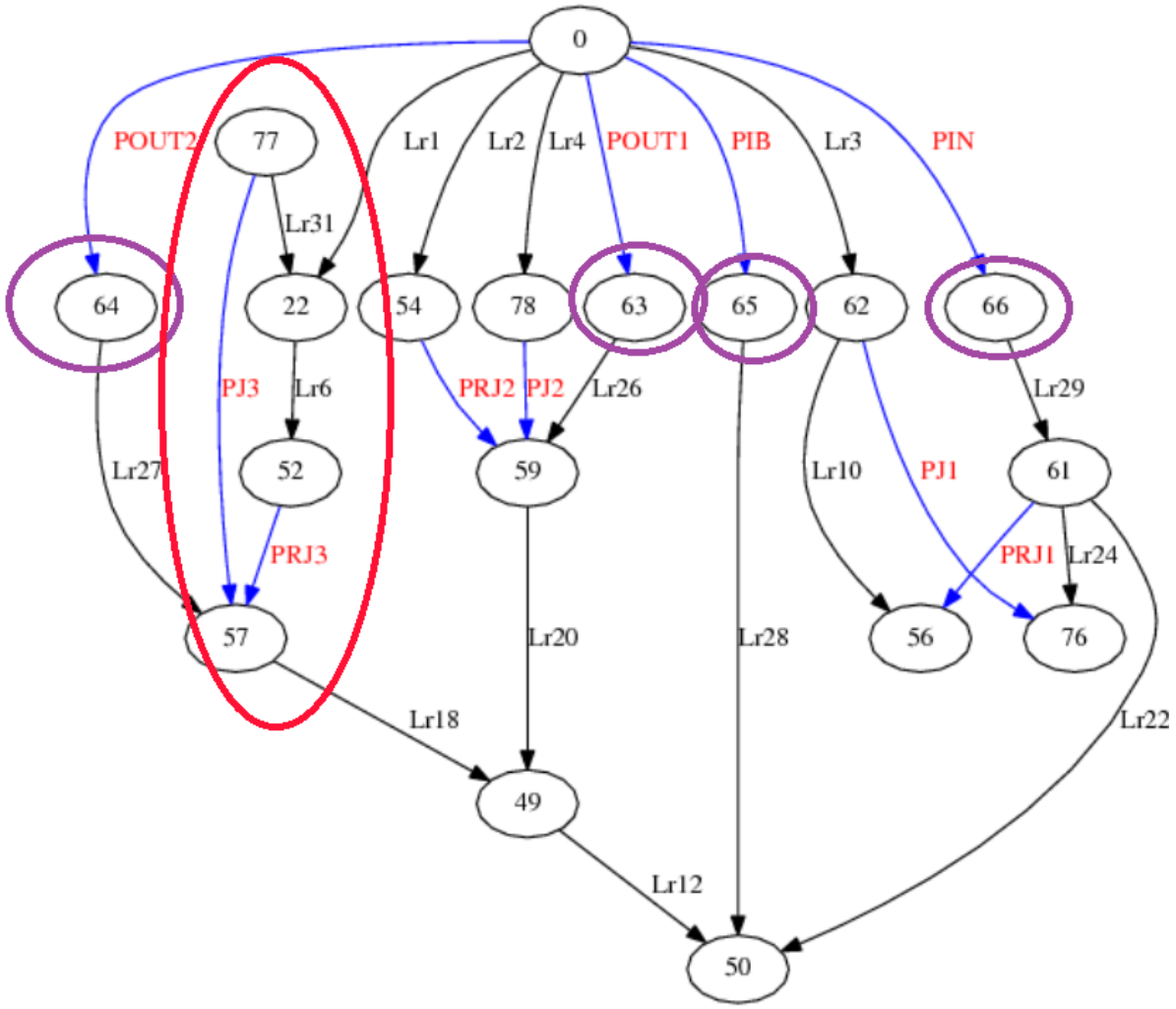


Figure 6.14: Figure showing a fully simplified Hypres splitter cell without mutual coupling. The input and output ports are circled in purple so that they can be identified in conjunction with the schematic representation in Figure 6.13.

Mutual coupling is in general beyond the scope of this thesis since the underlying graph structure should ideally be modified as mentioned in Chapter 2.

6. 1. 3 Confluence buffer

LVS in conjunction with inductance (not resistance) extraction for the confluence buffer is similar to that of the Fluxonics JTL and SFQ Splitter cells. One user selection node is required on M1 (the metal layer above ground) between the two junctions and the connection to ground (identified by an arrow on Figure 6.15).

Stellenbosch University https://scholar.sun.ac.za

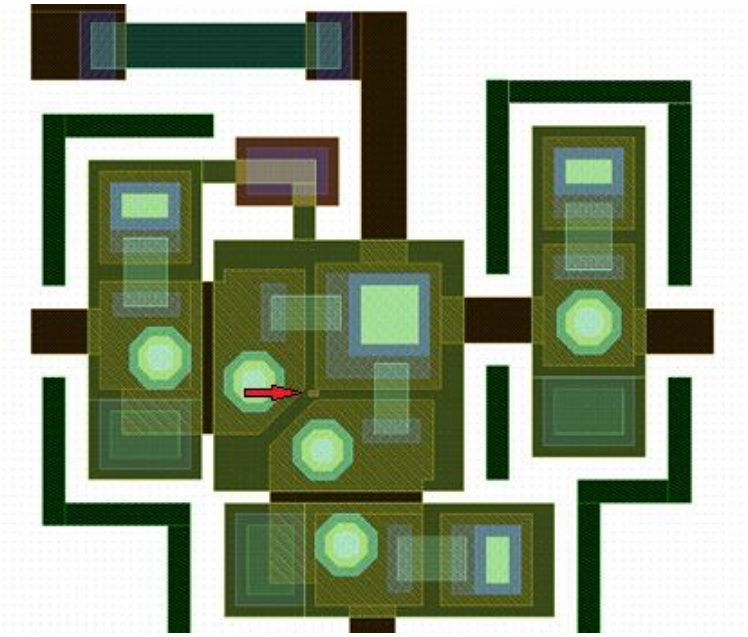


Figure 6.15: Layout of a Fluxonics Confluence Buffer cell where a user selection nodes has been added to M1 (see red arrow).

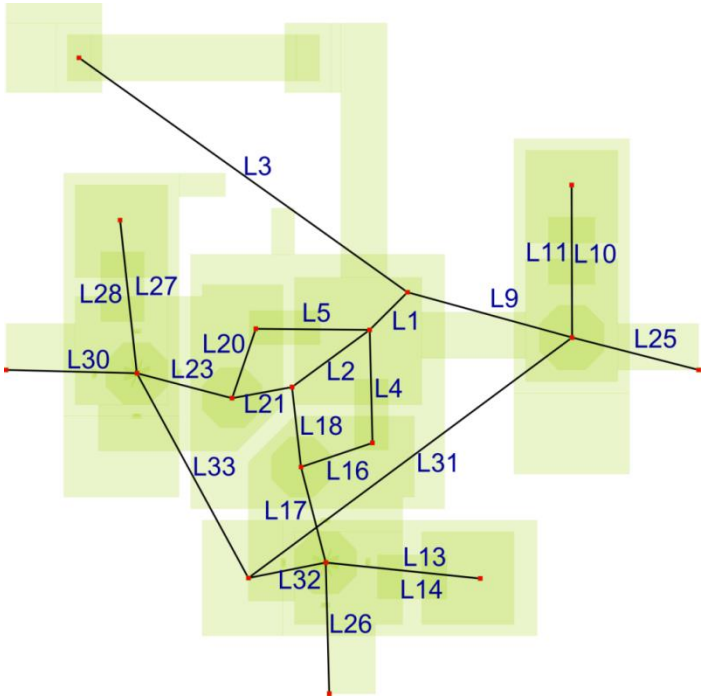


Figure 6.16: Extracted schematic for a Confluence Buffer cell. The node numbers can be excluded for cells where there is too much overlap between component names and node numbers. This option has been selected for this output.

The toolkit was successfully tested on the Confluence buffer cell (inductance and resistance included) and could identify the correct of input netlist.

The extracted schematic (node numbers excluded) can be seen in Figure 6.16 (and with node numbers in Figure F.8). This schematic's graph can be seen in Figure 6.18 and is isomorphic to the graph of the input netlist (derived from the schematic in Figure 6.17). LVS has been successfully performed on this cell.

Stellenbosch University https://scholar.sun.ac.za

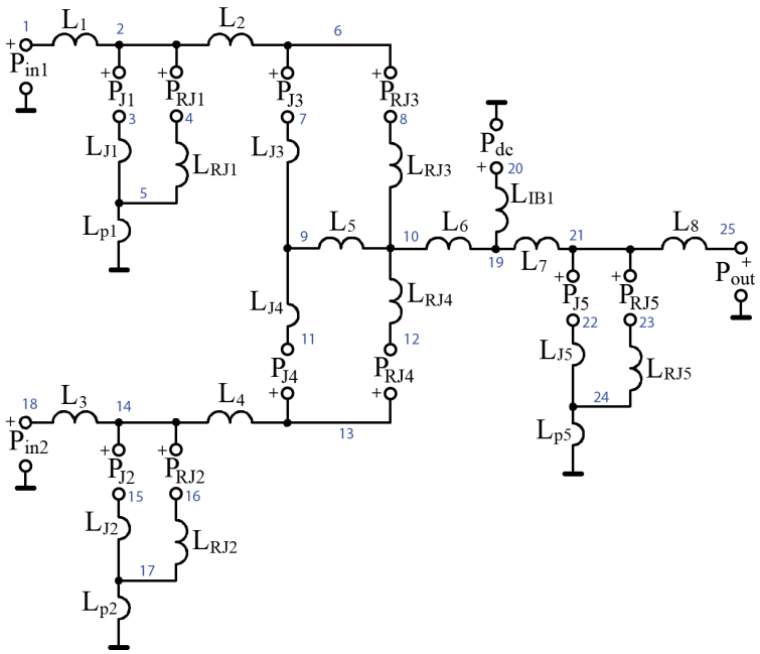


Figure 6.17: Input schematic for a confluence buffer cell where resistance and inductance extraction is required.

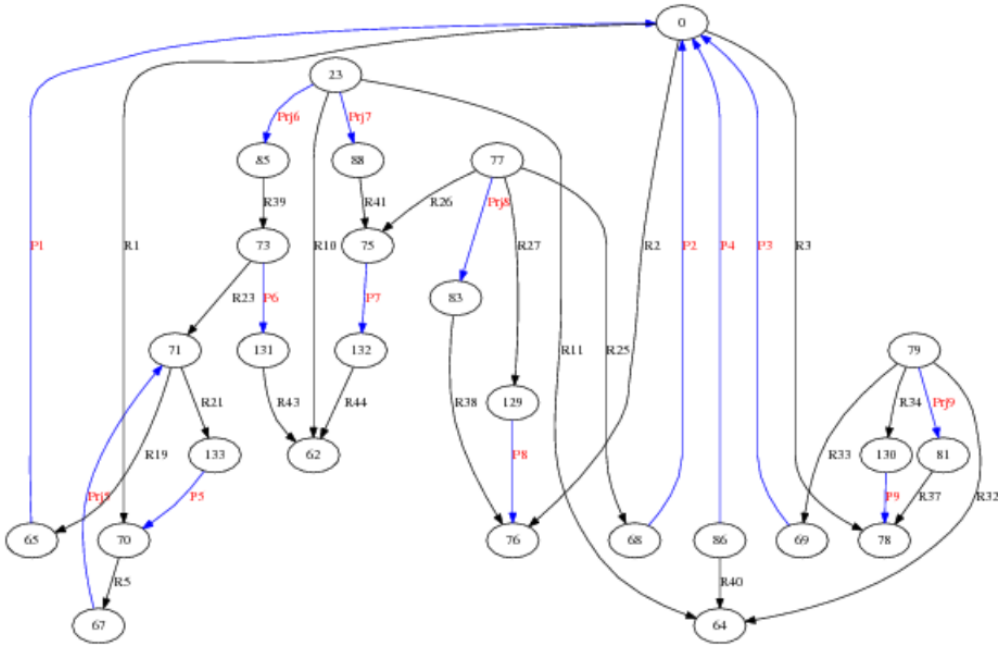


Figure 6.18: Graph view of the simplified extracted netlist for a confluence buffer cell. This graph representation corresponds to the extracted schematic in Figure 6.16 and is isomorphic with the netlist of the schematic in Figure 6.17.

6. 2 Section B: Examples where graphs are not isomorphic

We now address the problem of comparing graphs that have a different number of vertices and/ or edges. In such cases, as mentioned in Chapter 5, the two netlist comparison tests can be run:

1. The induced subgraph isomorphism test
2. The subgraph monomorphsim test

Stellenbosch University https://scholar.sun.ac.za

In this section, examples are used to graphically depict how the results from the induced subgraph isomorphism and subgraph monomorphism tests differ. We also discuss how and when to use each of these tests. In many cases the results will be identical. We will first discuss two examples for which this is the case. We then discuss examples where the results differ between tests.

6. 2. 1 Examples where subgraph monomorphism and induced subgraph isomorphism are equivalent

In Figure 6.19, we present Graph A – the graph representation of an extracted JTL. This will be used as the large graph for the first part of this discussion.

Figure 5.5 represents graph B – identical to graph A, but without the shunt impedances and their respective series ports. Figure 5.4 shows the 4 components (in green: Lr19, Lr20, Pjr4 and Pjr5) that are not found in the chosen mapping. When subgraph monomorphism is applied to Graphs A and B, 8 mappings are found. The same 8 mappings are found when induced subgraph isomorphism is applied.

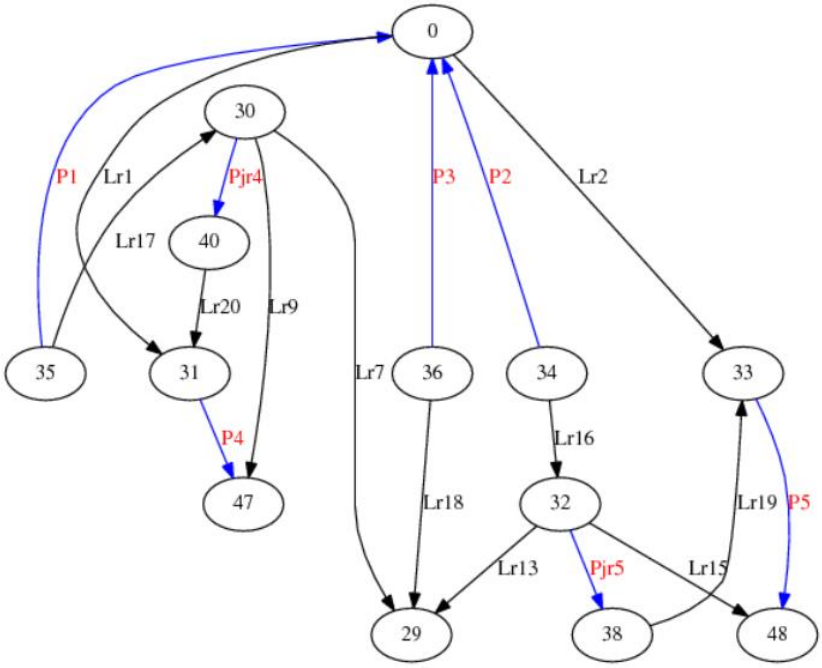


Figure 6.19: Graph A – the graph of the netlist extracted from a Fluxonics JTL where resistance is included. This graph will serve as the larger graph to which other graphs in this section will be compared.

Figure 6.19 shows a smaller graph: Graph D. When subgraph monomorphism and induce subgraph isomorphism are applied to Graphs A and D, the resultant mappings are equivalent; both functions yield 32 mappings. Figure F.10 and F.11 show two of the mappings.

Stellenbosch University https://scholar.sun.ac.za

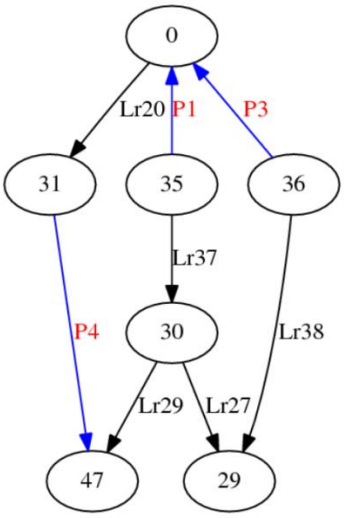


Fig 6.19: Graph D: an arbitrary small graph that contains two loops. This graph was originally in the form of Graph A, but many of the components were removed.

6. 2. 2 Examples where results for subgraph monomorphism and induced subgraph isomorphism differ

As mentioned in Chapter 5, there are cases when the information gained by applying subgraph monomorphism and induced subgraph isomorphism differs. Different results can be further broken down into (1) cases when both give a 'yes' result to the decision problems but the number of mappings differ and (2) cases there is at least one subgraph monomorphism mapping but no induced subgraph isomorphism mappings. For readability we will refer to (1) as "Type A problems" and (2) as "Type b problems". In the following section will first look at Type A and then Type B problems.

6. 2. 2. 1 Type A problems

We will start with a small example.

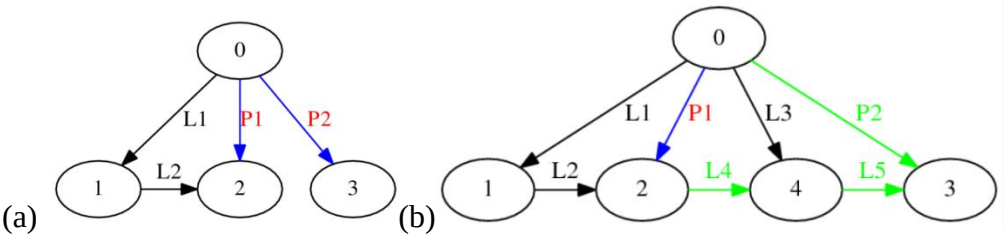


Figure 6.20: The smaller graph, Graph E, is given in (a). In (b) we see one of the induced subgraph isomorphism mappings of graph E onto graph F (the larger graph). The two green components surround the vertex that is not part of the chosen mapping.

In Figure 6.20 (a) the smaller graph (Graph E) is given. The larger graph (Graph F) can be seen in Figure 6.20 (b) if one ignores the colouring of the components. When the subgraph monomorphism test is applied to Graphs E and F, the decision problem is answered 'yes'. There are 20 mappings found – one of the mappings is represented by a selection of the components L1, L2, L3 and P1 (the red and black components) in Figure 6.20 (b).

Stellenbosch University https://scholar.sun.ac.za

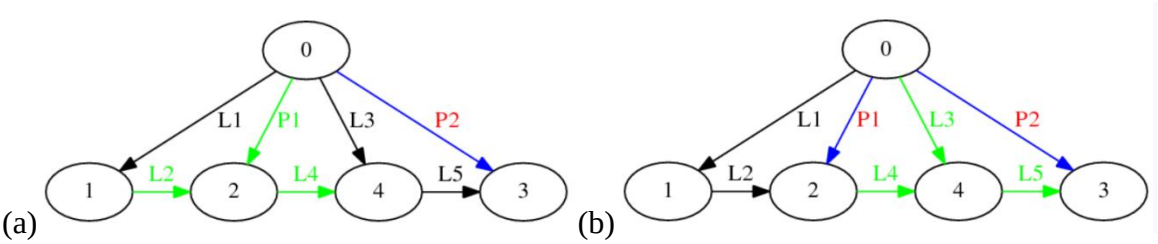


Figure 6.21: This figure is a graphic representation of results to the subgraph monomorphism test. Two of the four mappings can be seen. The other two mappings look identical (graphically) to (a) and (b) respectively, but the nodes 4 and 5 (a) and 1 and 2 (b) are swapped for the other two mappings.

In Figure 6.21, two of the mappings from the subgraph monomorphism test can be seen. Figure 6.22 gives the actual mappings between Graphs E and F as seen in the solution output file: the first two mappings correspond to Figure 6.21 (a) and the second two to (b). The format of the mapping representation is as follows:

$$(smaller\ graph\ vertex\ u, larger\ graph\ vertex\ v),$$

where u and v are the edges.

```
Induced Subgraph Isomorphism Test:
There are 4 permutations for this 4 vertex graph!
(0, 0)(1, 4)(2, 3)(3, 1)
(0, 0)(1, 3)(2, 4)(3, 1)
(0, 0)(1, 1)(2, 2)(3, 3)
(0, 0)(1, 2)(2, 1)(3, 3)
```

Figure 6.22: Figure showing the mappings from the solution text file for netlist comparison between graph E and Graph F. The first two mappings correspond to figure 6.21 (a) and the following two to Figure 6.21 (b). Use the node numbers in Figure 6.20 (a) as a reference when interpreting the above results.

We now look at a slightly larger example. Graph G is given below in Figure 6.23. In this graph, both parasitic branches are excluded and the edges connected to vertex 29 have been removed.

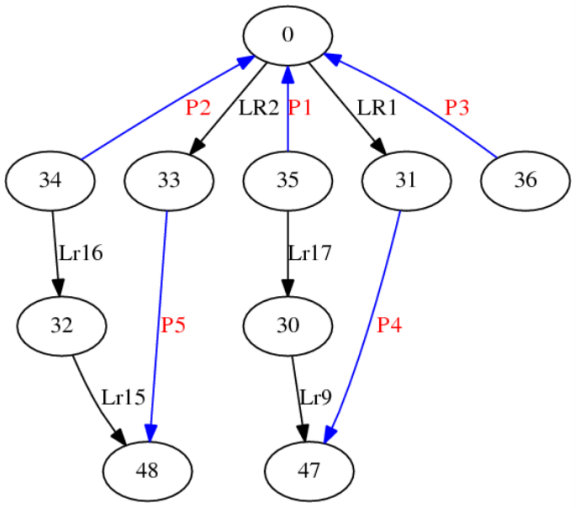


Figure 6.23: Graph G: a small graph that was derived from graph A by removing components. A vertex of order 1 (node 36) can be seen.

Stellenbosch University https://scholar.sun.ac.za

When the subgraph monomorphism test is performed on Graphs G and A, 64 mappings are found. Figure 6.24 depicts one of the mappings. This is not the 'correct' mapping, however, since the components that were actually removed are not those highlighted in green in Figure 6.24. This is due to the matrix symmetry and the fact that the two graphs differ so fundamentally. In larger, less symmetric graphs this poses less of a problem but it is, however, an inherent limitation to this toolkit.

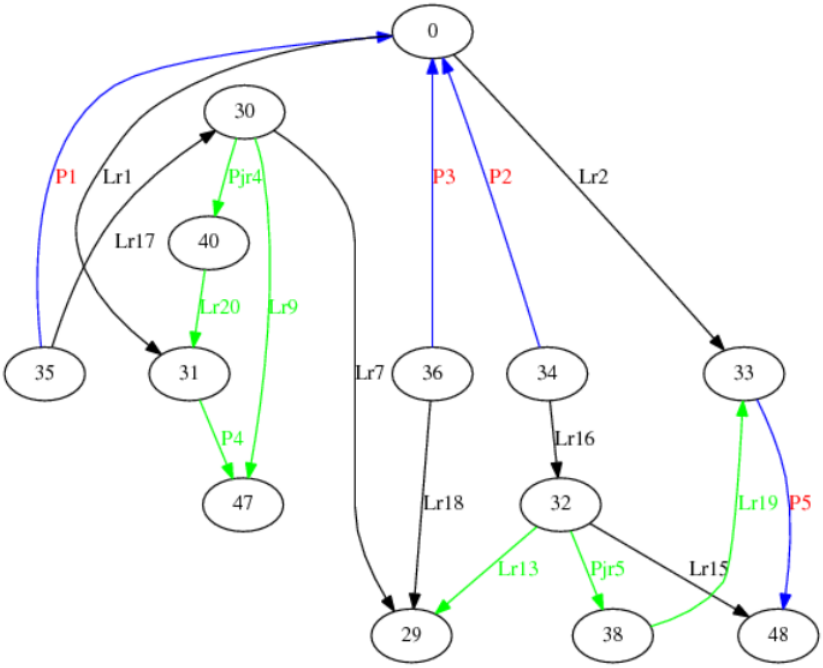


Figure 6.24: Graphic result of the netlist comparison (subgraph monomorphsim) between Graphs G and A. Components Lr19, Lr20, Lr13, Pjr4, Pjr5, Pr4 and Lr9 are not part of the chosen mapping.

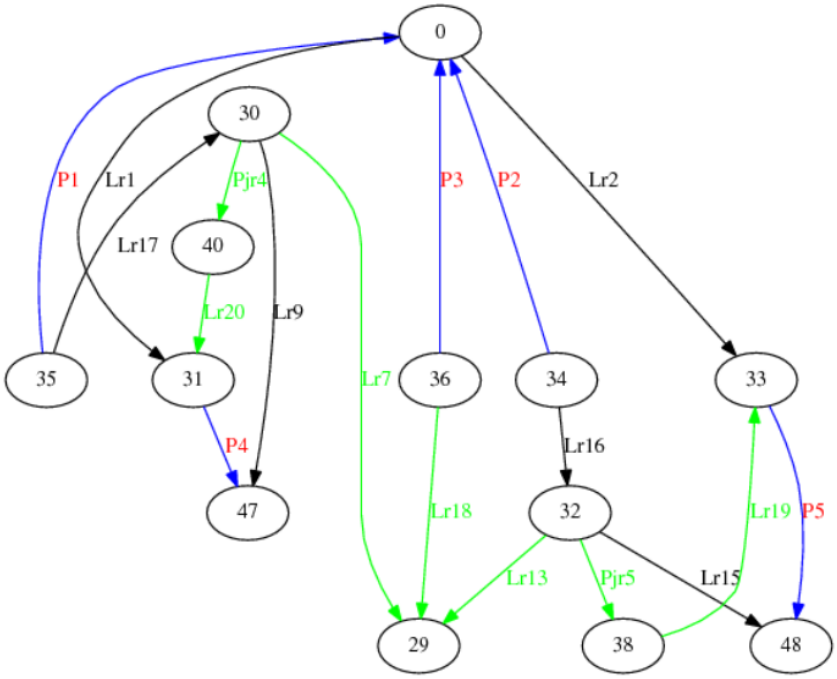


Figure 6.25: Graphic result of the netlist comparison (induced subgraph isomorphism) between Graphs G and A. Components Lr19, Lr20, Pjr4, Pjr5, Lr18, Lr7 and Lr13 are not part of the chosen mapping. These are the components that were originally removed from Graph A to give us Graph G: this is the 'correct' mapping.

Stellenbosch University https://scholar.sun.ac.za

In an example like this, it would be completely unnecessary to perform the subgraph monomorphism test. The induced subgraph isomorphism test has a faster execution time since it finds fewer mappings.

We will now look at an example where the subgraph monomorphism test would be required to give additional information to the user to perform netlist comparison.

6. 2. 2. 2 Type B problems

These problems only (but not always) occur when leaf vertices are found in one of the two graphs that are being compared. We will first look at a small example and then the results for some larger examples will be given.

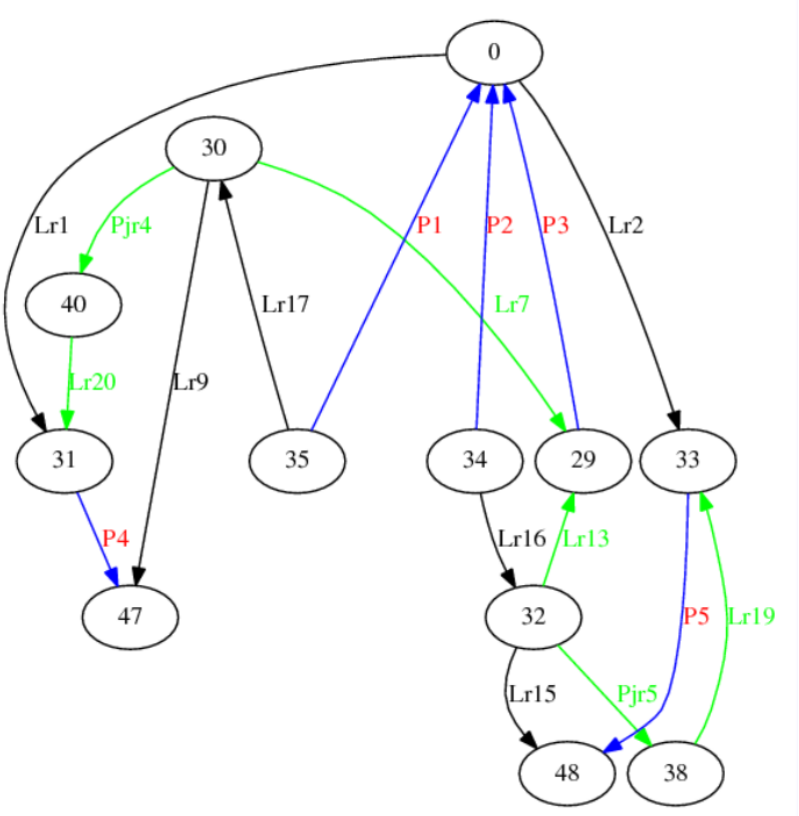


Figure 6.26: Figure showing Graph H (a modified version of graph A (vertex 36 is removed)) and the mapping of Graph G onto graph H.

In Figure 6.26 we are presented with Graph H. Graph H is a modified version of graph G: vertex 36 and Lr18 were removed. The test: induced_subgraph(G,H), results in no mappings. The subgraph_mono(G,H) test, however, results in 96 mappings.

The question why there are no induced subgraph isomorphism mappings but are subgraph monomorphism mappings will be discussed in in the section below.

6. 2. 2. 3 Discussion: identification of open and closed circuits using (induced)subgraph iso- or mono-morphism

Stellenbosch University https://scholar.sun.ac.za

In cases where there are no induced subgraph isomorphsims mappings but *are* subgraph monomorphism mappings we can make one of three conclusions:

a. The input schematic is incorrect
b. The layout is incorrect
c. The automated layout to schematic tool has been used incorrectly (ports or user-selection-nodes placed incorrectly) or used on an un-supported layout type.

Because subgraph monomorphism mappings *do* exist, the "mistake" should not be too difficult to find. The fact that there are no induced subgraph isomorphisms means that there is most likely a leaf vertex that should have been connected to ground but has not been (either in the input schematic or layout) or that there is an open circuit or closed circuit fault (an edge between two vertices either exists when it should not or doesn't exist when it should exist either in the input schematic or layout).

Schematic faults of this type are usually typing errors. If the fault is in the layout itself (b), it will usually be either an incorrect connection or a via that has been left out.

With regards to use to the layout to schematic tool (c), it is highly unlikely that incorrect selection node placement will result in monomorphsim mappings but no induced isomorphsims mappings. If selection nodes have been use incorrectly, there will usually be no isomorphism/monomorphism mappings at all. It is much more likely that port placement is the cause of the problem. If ports are removed from the netlist (manually) to improve execution time, leaf vertices will exist in the modified netlist and this will increase the chances of there being no induced isomorphism mappings, but monomorphism mappings.

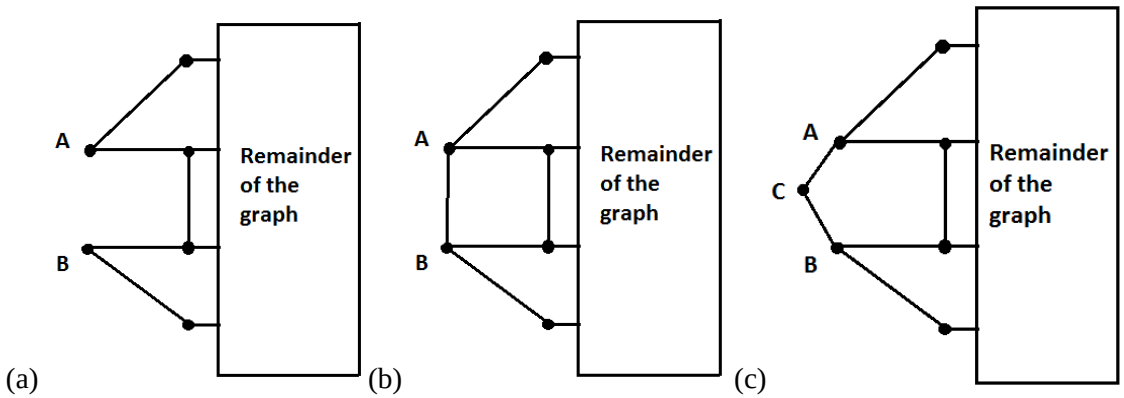


Figure 6.27: In (a) we see a portion of a graph where there are no edge between vertices A and B. In (b) there is a vertex between A and B. When performing the subgraph monomorphsim test, the graph in (a) is a subgraph of (b) (and vice versa) if the remainder of the graphs are equivalent. The graphs in (a) is, however, not an *induced* subgraph of (b) but it is an induced subgraph of (c). The graph in (b) is not an induced subgraph of (c).

In Figure 6.27 three graphs can be seen. The part of the graph between vertices A and B is the only part of the graph that differs between these three graphs. The graphs in (a) and (b) represent short

Stellenbosch University https://scholar.sun.ac.za

and open circuit faults. The graph in (c) represents an entire branch that has been either added or omitted (usually the case with parasitic).

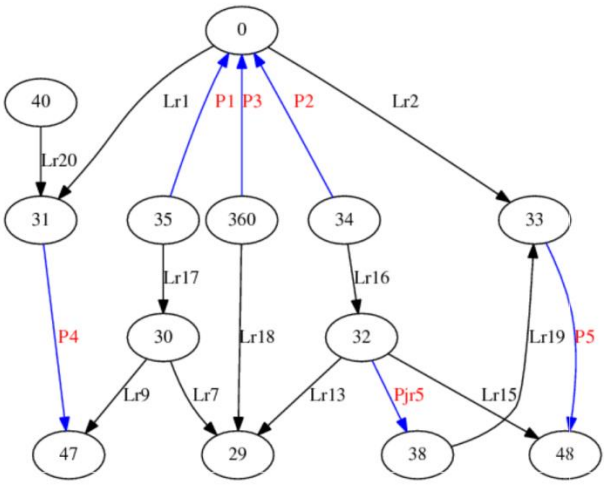


Figure 6.28: Graph A, but with Pjr4 removed from the netlist. One of the graphs used in an open circuit test.

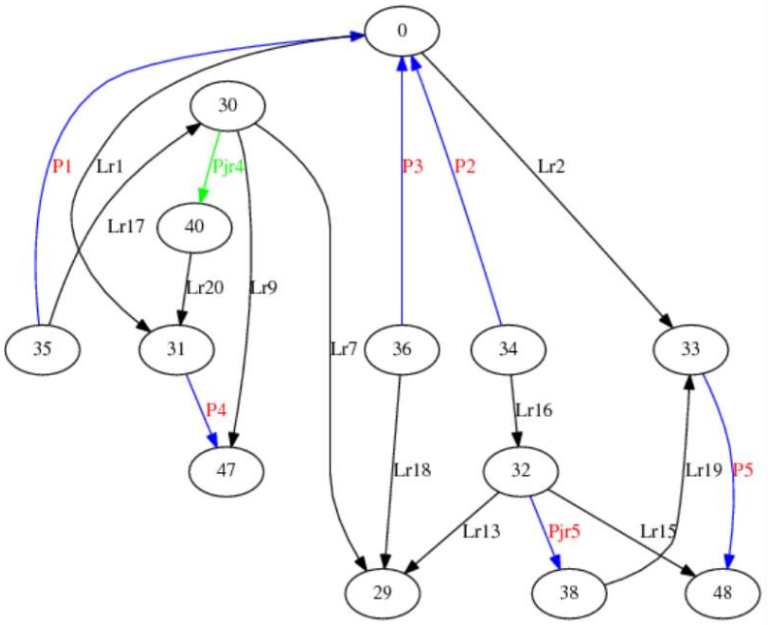


Figure 6.29: Resultant mapping of the graph in Figure 6.28 onto Graph A. In this mapping, component Pjr4 has been correctly identified as the incorrect component.

Actual open and closed circuit faults have been simulated on practical cells. Netlists were modified by adding and removing components. One of these netlists can be seen in Figure 6.28.

For this specific test, the component Pjr4 was removed from the netlist of Graph A, but still included in the layout itself as a port. The subgraph monomorphism test was applied to the extracted netlist (Graph A) and the modified input netlist (Figure 2.28); a resultant mapping can be seen in Figure 6.29.

Unfortunately there are 8 possible mappings. Depending on the chosen mapping, components Ljr4, Lr20, P4, Lr9, Ljr5, Lr19, P5 or Pr19 could have been highlighted as the removed component. In

Stellenbosch University https://scholar.sun.ac.za

cases such as these, the mapping output text file can be viewed and analysed so that all possible faults can be investigated.

6. 2. 2. 3 Larger graph: execution time investigation

Larger graphs that also fall into this category were tested so as to determine the execution times of larger problems. For these tests un-simplified netlist were used since these resulted in graphs with many more vertices and edges (parallel and series components were not removed). One of the unplanar graphs that were tested was an un-simplified version of an SFQ to DC cell with one JTL attached to it.

The original graph contained 79 vertices and over 100 components. For testing purposes, components were removed manually from the original graph and the subgraph monomorphism test was performed. We can usually assume (especially when there is a high order of symmetry between the graphs) that there will be many more mappings for a small graph into a much larger graph than for two graphs of a similar size.

After 15 components were removed from the original, 512 mappings were found. This took under a second for the entire process (including printing the printing of all the mappings). Removing an additional 5 component from the smaller graph did not noticeably change the execution time, but the total number of mappings increased to 1536.

The execution time increased significantly when the smaller graph was reduced by 40 edges. Most of this time was, however, spent on sorting and printing the 67968 mappings.

- When the default settings were used, the entire process took 33 seconds.
- When the option not to list the mappings was selected the netlist comparison completed in under 4 seconds. One of the mappings is shown in Figure 6.30.

Three additional edges were then removed and this resulted in an additional 2 seconds of execution time (without printing the mappings): a total of 107648 mappings were found.

Figure 6.30: Graph of one of the mappings described in the test above.

When small graphs are mapped into significantly larger graphs, many ‘incorrect’ mappings may be found (mappings that do not highlight the *actual* difference between the circuits). Fortunately, the

Stellenbosch University https://scholar.sun.ac.za

extracted and original netlists are often of a similar size and because the user can visually inspect the output schematic, the actual mistake should not be too difficult to find.

Stellenbosch University https://scholar.sun.ac.za

Chapter 7
Conclusion and Recommendations

A toolkit has been developed to simplify the current full-custom superconductive IC design process. The toolkit is suitable for SFQ IC designers who require LVS tools to verify their cell layouts. The current version of this toolkit provides the user with three stand-alone tools that are best used in conjunction with InductEx:

- A GDSII file flattener
- A layout-to-schematic netlist extractor (with the option of viewing the schematic and netlist)
- A netlist comparison tool

The GDSII file flattener is used as a pre-processor for the layout-to-schematic tool and can be replaced by any GDSII file flattener. Conclusions and recommendations will be provided separately for the layout-to-schematic and netlist comparison tools.

7. 1 Layout-to-schematic tool

The layout to schematic tool currently (fully) supports only cells that do not use mutual coupling, thus greatly limiting its scope. We recommend that the underlying graph structure be changed to support mutual coupling in future versions (as discussed in Section 2.5.3.2). Once support for mutual coupling is included, technologies (such as AQFP) that rely heavily on mutual coupling can also be supported.

The execution time of the layout-to-schematic tool is suitable for cell-level layout-to-schematic extraction. Should the tool be adapted for VLSI applications, it is recommended that the following modifications be made:

- A hierarchical approach should be used: cell boundaries must be identified and layout-to-schematic should be performed per cell
- More advanced algorithms should be implemented to replace the current path finding and MST algorithms (heuristics could be used)
- An alternative to the user-selection-nodes should, if possible, be found

For the small scale integration environment, however, this tool can greatly simplify the IC designer's task. The schematic and graph viewers are extremely useful representations of the extracted netlist. By viewing the schematic superimposed onto the cell layout, the user can quickly see how the layout maps to the extracted schematic; this can provide valuable assistance when searching for a mistake in the layout.

Stellenbosch University https://scholar.sun.ac.za

7. 2 Netlist comparison tool

The netlist comparison tool has already proved its worth in the testing process. When cell layouts are modified and improved, the user can easily either forget to update the netlist or may use an incorrect (older) netlist as an input for parameter extraction. This was the case in one of the example test circuits. The layout-to-schematic's output netlist did not correspond with the given netlist. After further investigation, it was determined that the given input netlist was, in fact, a netlist that corresponded to an older version of the cell. By using the netlist comparison tool before inductance extraction, the risk of extracting parameters of an incorrect model can be severely minimised.

The option of viewing the netlists graphically is especially useful. Because many of the IC designers using InductEx enter the components into the netlist files manually, mistakes are hard to avoid completely. By examining the graph view of the input netlist, the user can often identify mistakes at a glance (especially when circuits are small).

When comparing two netlists, visually inspecting the mapping of the one netlist onto the other netlist is a very effective way of finding differences between two circuits. The underlying principle of using graph isomorphism to find the mappings does, however, result in the problem of finding multiple possible differences between the circuits (multiple mappings are often generated). For complex circuits that have a low level of symmetry, this is less of a problem. The text file can be used besides the graphical representation of the mapping to provide additional assistance.

We recommended that attributed graph matching be used for later versions of this tool so as to minimise the symmetry. Another option could be to find the maximum common subgraph of the two netlists and provide the user with the mapping of this graph onto both the extracted and original netlists.

It is evident that the symmetry of the graphs play an important role in how many mappings are found and consequently also in the execution time. For small simplified cells, however, execution time is almost negligible – especially since inductance extraction itself takes so much longer.

7. 3 Summary

The aim of this thesis was to investigate the algorithms required to perform stand-alone LVS for SFQ cells. The task of extracting a schematic or netlist from a layout without any additional information is a highly challenging task; further research is required to fully automate this process (i.e., without any user input such as user-selection-nodes). The layout-to-schematic tool is currently the limiting part of the toolkit, given the lack of full support for circuits with mutual coupling.

The netlist comparison and viewing tool, on the other hand, is highly valuable in its current state and can be used separately from the layout-to-schematic extraction tool.

Stellenbosch University https://scholar.sun.ac.za

Further investigation into the use of machine-learning and linear algebra techniques applied to the branch current matrix to predict the accuracy of the input netlist *before* inductance extraction is recommended.

Although many improvements to this toolkit can still be made, the implemented version of these tools can already provide great benefit to RSFQ cell designers. It is further recommended that the condition number be used as an alternative verification check in parameter extraction.

Stellenbosch University https://scholar.sun.ac.za

Bibliography

1. Kilby, J.S., "Invention of the integrated circuit," *Electron Devices, IEEE Transactions on* , vol.23, no.7, pp.648,654, Jul 1976
2. Yokotani, T., "Application and technical issues on Internet of Things," *Optical Internet (COIN), 2012 10th International Conference on* , vol., no., pp.67,68, 29-31 May 2012
3. Gargini, P., "The International Technology Roadmap for Semiconductors (ITRS): "Past, present and future"," *GaAs IC Symposium, 2000. 22nd Annual* , vol., no., pp.3,5, 5-8 Nov. 2000
4. Wimer, S., Planar CMOS to multi-gate layout conversion for maximal fin utilization. Integration 47:115–122 , 2014
5. S. E. Thompson and S. Parthasarathy, "Moores law: The future of Si microelectronics," Mater. Today, vol. 9, no. 6, pp. 20–25, Jun. 2006.
6. Moore, Gordon E., "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," *Solid-State Circuits Society Newsletter, IEEE* , vol.11, no.5, pp.33,35, Sept. 2006
7. International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://public.itrs.net/ 2013.
8. Ahmed, K.; Schuegraf, K., "Transistor wars," Spectrum, IEEE , vol.48, no.11, pp.50,66, November 2011
9. Dr. Sani Nassif , "Fins on Transistors Change Processor Power and Performance", [Online]. Available at: http://ibmresearchnews.blogspot.co.il/2012/07/fins-on-transistors-change-processor.html, July 2012.
10. GlobalFoundries intros 14 nm Process with FinFET Transistors, http://www.digitimes.com-/news/a20120921PR200.html, September 2012.
11. Intel Discloses Newest Microarchitecture and 14 Nanometer Manufacturing Process Technical Details. [online]. Available at: http://newsroom.intel.com/community /intel-_newsroom/blog/2014 /08/11/ intel-discloses- newest-microarchitecture-and-14-nanometer-manufacturing-process-technical-details, Aug 2014
12. Samsung 14nm FinFET Design with Cadence Tools. [online]. Available at: https://www.semiwiki.com/forum/content/3866-samsung-14nm-finfet-design-cadence-tools.html, Sep 2014
13. International technology roadmap for semiconductors 2013 edition emerging research devices. [Online]. Avaliable: http://www.itrs.net/Links/2013ITRS/2013Chapters/2013ERD-_Summary.pdf

Stellenbosch University https://scholar.sun.ac.za

14. Heinig, A; Dietrich, M.; Herkersdorf, A; Miller, F.; Wild, T.; Hahn, K.; Grunewald, A; Bruck, R.; Krohnert, S.; Reisinger, J., "System integration — The bridge between More than Moore and More Moore,"*Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014* , vol., no., pp.1,9, 24-28 March 2014

15. Liu, Q.; Vinet, M.; Gimbert, J.; Loubet, N.; Wacquez, R.; Grenouillet, L.; Le Tiec, Y.; Khakifirooz, A; Nagumo, T.; Cheng, K.; Kothari, H.; Chanemougame, D.; Chafik, F.; Guillaumet, S.; Kuss, J.; Allibert, F.; Tsutsui, G.; Li, J.; Morin, P.; Mehta, S.; Johnson, R.; Edge, L.F.; Ponoth, S.; Levin, T.; Kanakasabapathy, S.; Haran, B.; Bu, H.; Bataillon, J.-L.; Weber, O.; Faynot, O.; Josse, E.; Haond, M.; Kleemeier, W.; Khare, M.; Skotnicki, T.; Luning, S.; Doris, B.; Celik, M.; Sampson, R., "High performance UTBB FDSOI devices featuring 20nm gate length for 14nm node and beyond," *Electron Devices Meeting (IEDM), 2013 IEEE International* , vol., no., pp.9.2.1,9.2.4, 9-11 Dec. 2013

16. Davis, W.R.; Wilson, J.; Mick, S.; Xu, J.; Hao Hua; Mineo, C.; Sule, AM.; Steer, M.; Franzon, P.D., "Demystifying 3D ICs: the pros and cons of going vertical," *Design & Test of Computers, IEEE* , vol.22, no.6, pp.498,510, Nov.-Dec. 2005

17. Bin Yu; Chang, L.; Ahmed, S.; Haihong Wang; Bell, S.; Chih-Yuh Yang; Tabery, C.; Chau Ho; Qi Xiang; Tsu-Jae King; Bokor, J.; Chenming Hu; Ming-Ren Lin; Kyser, D., "FinFET scaling to 10 nm gate length," *Electron Devices Meeting, 2002. IEDM '02. International* , vol., no., pp.251,254, 8-11 Dec. 2002

18. Amat, E.; Almudever, C.G.; Aymerich, N.; Canal, R.; Rubio, A, "Suitability of the FinFET 3T1D Cell Beyond 10 nm," *Nanotechnology, IEEE Transactions on* , vol.13, no.5, pp.926,932, Sept. 2014

19. Z Chang, J.B.; Guillorn, M.; Solomon, P.M.; Lin, C-H; Engelmann, S. U.; Pyzyna, A; Ott, J.A; Haensch, W.E., "Scaling of SOI FinFETs down to fin width of 4 nm for the 10nm technology node," *VLSI Technology (VLSIT), 2011 Symposium on* , vol., no., pp.12,13, 14-16 June 2011

20. Radosavljevic, M.; Dewey, G.; Basu, D.; Boardman, J.; Chu-Kung, B.; Fastenau, J.M.; Kabehie, S.; Kavalieros, J.; Le, V.; Liu, W. K.; Lubyshev, D.; Metz, M.; Millard, K.; Mukherjee, N.; Pan, L.; Pillarisetty, R.; Rachmady, W.; Shah, U.; Then, H. W.; Chau, R., "Electrostatics improvement in 3-D tri-gate over ultra-thin body planar InGaAs quantum well field effect transistors with high-K gate dielectric and scaled gate-to-drain/gate-to-source separation," *Electron Devices Meeting (IEDM), 2011 IEEE International* , vol., no., pp.33.1.1,33.1.4, 5-7 Dec. 2011

21. Hutchby, J.A; Bourianoff, G.I; Zhirnov, V.V.; Brewer, J.E., "Extending the road beyond CMOS," *Circuits and Devices Magazine, IEEE* , vol.18, no.2, pp.28,41, Mar 2002

22. P. Russer and N. Fichtner "Nanoelectronics in radio-frequency technology", *IEEE Microwave Mag.*, vol. 11, no. 3, pp.119 -135 2010

Stellenbosch University https://scholar.sun.ac.za

23. Esch, J., "Prolog to The Future of Integrated Circuits: A Survey of Nanoelectronics," *Proceedings of the IEEE* , vol.98, no.1, pp.8,10, Jan. 2010

24. Arden, W., Brillouët, M., Cogez, P., Graef, M., Huizing, B., Mahnkopf, R., "More-than-Moore White Paper", International Technical Roadmap for Semiconductors, 2010,

25. Casale-Rossi, M.; De Micheli, G.; Aitken, R.; Domic, A; Horstmann, M.; Hum, R.; Magarshack, P., "Panel: Emerging vs. established technologies, a two sphinxes' riddle at the crossroads?," *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014* , vol., no., pp.1,4, 24-28 March 2014

26. E. Lyons , V. Ganti , R. Goldman , V. Melikyan and H. Mahmoodi "Full-custom design project for digital VLSI and IC design courses using synopsys generic 90 nm CMOS library", *Proc. IEEE Int. Conf. Microelectron. Syst.Edu.*, pp.45 -48 2009

27. K. K. Likharev and V. K. Semenov, "RSFQ logic/memory family: A new josephson-junction technology for sub-terahertz-clock frequency digital systems", *IEEE Trans. Appl. Supercond.*, vol. 1, pp. 1-28, Mar. 1991.

28. H. Hayakawa, N. Yoshikawa, S. Yorozu and A. Fujimaki, "Superconducting digital electronics," *Proc. IEEE* , vol.92, no.10, pp. 1549-1563, Oct. 2004.

29. M. H. Volkmann, A. Sahu, C. J. Fourie and O. A. Mukhanov, "Implementation of energy efficient single flux quantum digital circuits with sub-aJ/bit operation," *Supercond. Sci. Technol.*, vol. 26, 015002, 2013.

30. O. A. Mukhanov, "Energy-Efficient Single Flux Quantum Technology," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 760-769, Jun. 2011.

31. O. T. Oberg, Q. P. Herr, A. G. Ioannidis, A. Y. Herr, "Integrated Power Divider for Superconducting Digital Circuits," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 571-574, Jun. 2011.

32. N. Takeuchi, K. Ehara, K. Inoue, Y. Yamanashi and N. Yoshikawa, "Margin and Energy Dissipation of Adiabatic Quantum-Flux-Parametron Logic at Finite Temperature," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, 1700304, Jun. 2013.

33. D. S. Holmes, A. L. Ripple, M. A. Manheimer, "Energy-Efficient Superconducting Computing—Power Budgets and Requirements," *IEEE Trans. Appl. Supercond.*, vol.23, no.3, 1701610, Jun. 2013

34. K. Gaj, Q. P. Herr, V. Adler, A. Krasniewski, E. G. Friedman and M. J. Feldman, "Tools for the computer-aided design of multigigahertz superconducting digital circuits," *IEEE Trans. Appl. Supercond.*, vol.9, no.1, pp. 18-38, Mar. 1999.

35. K. Gaj, Q. P. Herr, V. Adler, D. K. Brock, E. G. Friedman and M. J. Feldman, "Toward a systematic design methodology for large multigigahertz rapid single flux quantum circuits," *IEEE Trans. Appl. Supercond.*, vol.9, no.3, pp. 4591-4606, Sept. 1999.

Stellenbosch University https://scholar.sun.ac.za

36. C. J. Fourie and M. H. Volkmann, "Status of Superconductor Electronic Circuit Design Software," *IEEE Trans. Appl. Supercond.*, vol.23, no.3, 1300205, Jun. 2013
37. Cadence Design Systems, 2655 Seely Ave., San Jose, CA 95134, USA.
38. L. M. Rosenberg, "The Evolution of Design Automation to Meet the Challenges of VLSI," in *Proc. 17th Design Automation Conf.*, pp.3-11, 1980.
39. Anderson, Leigh C., et al. "Integrated verification and manufacturability tool." U.S. Patent No. 6,415,421. 2 Jul. 2002.
40. Y. Wang , D. Overhauser and M. Basel "Accurate parasitic resistance extraction for interconnect analysis", *Proc. IEEE Custom Integr. Circuits Conf.*, pp.255 -258 1995
41. Z. Nian, D. C. Wunsch, "Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach," *IEEE Trans. Fuzzy Syst.*, vol. 14, pp. 728-737, Dec. 2006.
42. H. S. Baird and Y. E. Cho. "An artwork design verification system." in *Proc. 12th Design Automation Conf.*, 1975.
43. E. Barke, "A network comparison algorithm for layout verification of integrated circuits," *IEEE Trans. CAD*, vol. CAD-3, pp. 135-141, 1984.
44. M. Ohlrich, C. Ebeling, E. Ginting and L. Sather, "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm," *30th Design Automation Conf.* pp. 31-37, Jun. 1993.
45. Dummer, G. W. A, "Electronic components in Great Britain," *Electrical Engineering* , vol.72, no.2, pp.167,169, Feb. 1953
46. Yu, A, "The future of microprocessors," Micro, IEEE , vol.16, no.6, pp.46,53, Dec 1996
47. C. J. Fourie, O. Wetzstein, T. Ortlepp and J. Kunert, "Three-dimensional multi-terminal superconductive integrated circuit inductance extraction," Supercond. Sci. Technol., vol. 24, 125015, Nov. 2011.
48. J. D. Tygar and R. Ellickson "Efficient Netlist Comparison Using Hierarchy and Randomization", *22nd Design Automation Conf.*, pp.702 -708 1985
49. Muller, L.C.; Fourie, C.J., "Automated State Machine and Timing Characteristic Extraction for RSFQ Circuits," Applied Superconductivity, IEEE Transactions on , vol.24, no.1, pp.3,12, Feb. 2014
50. Or, C.-Y.; Franca, J.E., "Layout driven macromodel of an operational amplifier," *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on* , vol.6, no., pp.302,305 vol.6, 31 May-3 Jun 1998
51. Liconix, Sunnyvale. "Software available at http://www. wrcad. com/freestuff. html, Whiteley Research Inc., Sunnyvale." No. 94086. 1981.
52. V. Bolkhovsky, S. Tolpygo, M. A. Gouker, L. M. Johnson, and W. D. Oliver, "Government-furnished superconducting electronics fabrication," IARPA Proposers' Day

Stellenbosch University https://scholar.sun.ac.za

for the Cryogenic Computing Complexity (C3) program, Washington, D.C, March 12, 2013; http://www.iarpa.gov/Programs/sso/C3/solicitation_c3.html M.

53. Hidaka et al., "Current status and future prospect of the Nb-based fabrication process for single flux quantum circuits," Supercond. Sci. Technol., vol. 19, pp. S138-142, 2006.
54. T. Satoh et al., "Improvement of Fabrication Process for 10-kA/cm2 Multi -Layer Nb Integrated Circuits," IEEE Trans. Appl. Supercond., vol. 12, pp. 169-172, 2007.
55. Q. P. Herr, A. Y. Herr, O. T. Oberg, and A. G. Ioannidis, "Ultra-low-power superconductor logic," *J. Appl. Phys.*, vol. 109, no. 10, p. 103903, 2011.
56. Lotter, P. Parameter Extraction of Superconducting Integrated Circuits. 2006.
57. S. Anders *et al.*, "European roadmap on superconductive electronics–status and perspectives," *Physica C*, vol. 470, pp. 2079-2126, 2010
58. *HYPRES Nb Process Design Rules*, HYPRES Inc., Elmsford, NY, USA, Dec. 12, 2012, Revision #25.
59. D. Yohannes, S. Sarwana, S. K. Tolpygo, A. Sahu, Y. A. Polyakov and V. K. Semenov, "Characterization of HYPRES's 4.5 kA/cm2 & 8kA/cm2 Nb/AlOx/Nb fabrication processes," *IEEE Trans. Appl. Supercond.*, vol. 15, no. 2, pp. 90–93, Jun. 2005.
60. Rapid Single Flux Quantum – Design Rules for Nb/Al2O3-Al/Nb Process Version 22.06.2007 at IPHT Jena. Available: http://www.ipht-jena.de/forschungsabteilungen/quantendetektion/fluxonics-foundry-the-foundry-of-the-european-fluxonicsnetwork.html
61. D.K. Brock, E.K. Track, and J.M. Rowell, "Superconductor ICs: The 100 GHz second generation", *IEEE Spectr.*, vol. 38, pp.40 -46 2000
62. K. K. Likharev, O. A. Mukhanov and V. K. Semenov, "Resistive single flux quantum logic for the Josephson-junction technology," in: *SQUID'85*, Berlin, Germany, 1985, pp. 1103-1108.
63. Bumble, B.; Fung, A.; Kaul, A.B.; Kleinsasser, Alan W.; Kerber, G.L.; Bunyk, P.; Ladizinsky, E., "Submicrometer $Nb/Al-AlO_x/Nb$ Integrated Circuit Fabrication Process for Quantum Computing Applications," *Applied Superconductivity, IEEE Transactions on* , vol.19, no.3, pp.226,229, June 2009
64. Dorojevets, M.; Chen, Z.; Ayala, C.L.; Kasperek, A.K., "Towards 32-bit Energy-Efficient Superconductor RQL Processors: The Cell-Level Design and Analysis of Key Processing and On-chip Storage Units,"*Applied Superconductivity, IEEE Transactions on* , vol.PP, no.99, pp.1,1
65. Inoue, K.; Takeuchi, N.; Yamanashi, Y.; Yoshikawa, N., "Simulation and implementation of an 8-bit carry look-ahead adder using adiabatic quantum-flux-

Stellenbosch University https://scholar.sun.ac.za

parametron," *Superconductive Electronics Conference (ISEC), 2013 IEEE 14th International* , vol., no., pp.1,3, 7-11 July 2013

66. Tanaka, M.; Kitayama, A.; Koketsu, T.; Ito, M.; Fujimaki, A., "Low-Energy Consumption RSFQ Circuits Driven by Low Voltages," *Applied Superconductivity, IEEE Transactions on* , vol.23, no.3, pp.1701104,1701104, June 2013
67. Stewart, Gilbert W. "On the early history of the singular value decomposition."*SIAM review* 35.4 (1993): 551-566.
68. S. Athloen and R. McLaughlin, Gauss-Jordan reduction: A brief history, American Mathematical Monthly 94 (1987) 130-142.
69. A. Tucker, The growing importance of linear algebra in undergraduate mathematics, The College Mathematics Journal, 24 (1993) 3-9.
70. Trefethen, Lloyd N., and David Bau III. *Numerical linear algebra*. Vol. 50. Siam, 1997.
71. Larson, Ron. *Elementary linear algebra*. Cengage Learning, 2012.
72. Miller, Ken. "Linear Algebra for Theoretical Neuroscience (Part 1)."
73. Miller, Ken. "Linear Algebra for Theoretical Neuroscience (Part 2)."
74. Miller, Ken. "Linear Algebra for Theoretical Neuroscience (Part 3)."
75. Hogben, Leslie, ed. *Handbook of linear algebra*. CRC Press, 2006.
76. Lyche T, University of Oslo Norway, http://heim.ifi.uio.no/~tom/svdslides.pdf
77. MacAusland, Ross. "The Moore-Penrose Inverse and Least Squares." (2014).
78. J. Stensby, course notes: EE448/528,http://www.ece.uah.edu/courses/ee448/chapter6.pdf, http://www.ece.uah.edu/courses/ee448/chapter7.pdf
79. M.T. Heath, University of Illinois ,Scientific Computing: An Introductory Survey, http://web.engr.illinois.edu/~heath/scicomp/notes/chap03_8up.pdf
80. Barnett, Janet Heine. "Early writings on graph theory: Euler circuits and the Königsberg bridge problem." (2005).
81. Shirinivas, S. G., S. Vetrivel, and N. M. Elango. "Applications of graph theory in computer science an overview." International Journal of Engineering Science and Technology 2.9 (2010): 4610-4621.
82. Deo, Narsingh. Graph theory with applications to engineering and computer science. PHI Learning Pvt. Ltd., 2004.s
83. Hopkins, Brian, and Robin J. Wilson. "The Truth about K6nigsberg." The College Mathematics Journal 35.3 (2004): 198-207.
84. L. Euler, Solutio problematis ad geometriam situs pertinentis,C ommentarii AcademiaeS cientiarum Imperi-alis Petropolitanae 8 (1736) 128-140 = Opera Omnia (1) 7 (1911-56), 1-10.
85. W. W. Rouse Ball, Mathematical Recreations and Problems of Past and Present Times, 1st ed. (later entitled Mathematical Recreations and Essays), Macmillan, London, 1892.

Stellenbosch University https://scholar.sun.ac.za

86. Kirchhoff, Studiosus. "Ueber den Durchgang eines elektrischen Stromes durch eine Ebene, insbesondere durch eine kreisförmige." Annalen der Physik 140.4 (1845): 497-514.
87. Weinberg, Louis. "Kirchhoff's' Third and Fourth Laws'." Circuit Theory, IRE Transactions on 5.1 (1958): 8-30.
88. Cayley, Arthur. "XXVIII. On the theory of the analytical forms called trees." The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 13.85 (1857): 172-176.
89. Kandel, Abraham, Horst Bunke, and Mark Last, eds. *Applied Graph Theory in Computer Vision and Pattern Recognition*. Vol. 1. Berlin: Springer, 2007
90. West, Douglas Brent. *Introduction to graph theory*. Vol. 2. Upper Saddle River: Prentice hall, 2001.
91. Dasgupta, Sanjoy, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.
92. DIJKSTRA, E W. A note on two problems in connexion with graphs. Numer. Math. 1, 269-271., 1959
93. R.E. Bellman, On a routing problem, Quart. Appl. Math., 16, pp. 87–90, 1958
94. Robert W. Floyd, Algorithm 97: Shortest path, Communications of the ACM, v.5 n.6, p.345, June 1962
95. Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* 4 (2): 100–107.
96. Tiernan, James C. "An efficient search algorithm to find the elementary circuits of a graph." *Communications of the ACM* 13.12 (1970): 722-726.
97. Tarjan, Robert. "Enumeration of the elementary circuits of a directed graph."*SIAM Journal on Computing* 2.3 (1973): 211-216.
98. Szwarcfiter, Jayme L., and Peter E. Lauer. "A search strategy for the elementary cycles of a directed graph." *BIT Numerical Mathematics* 16.2 (1976): 192-204.
99. Johnson, Donald B. "Finding all the elementary circuits of a directed graph."*SIAM Journal on Computing* 4.1 (1975): 77-84.
100. Ferreira, Rui, et al. "Optimal Listing of Cycles and st-Paths in Undirected Graphs." *arXiv preprint arXiv:1205.2766* (2012).
101. Graham, R.L.; Hell, Pavol, "On the History of the Minimum Spanning Tree Problem," *Annals of the History of Computing* , vol.7, no.1, pp.43,57, Jan.-March 1985
102. Karger, David R., Philip N. Klein, and Robert E. Tarjan. "A randomized linear-time algorithm to find minimum spanning trees." *Journal of the ACM (JACM)*42.2 (1995): 321-328.

Stellenbosch University https://scholar.sun.ac.za

103. Siek, Jeremy G., Lie-Quan Lee, and Andrew Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001.

104. Zampelli, Stéphane. "A constraint programming approach to subgraph isomorphism." *Doktorarbeit, Universite catholique de Louvain* (2008).

105. Skiena, Steven S. "The Algorithm Design Manual. 2008."

106. Gao, Yong, and Nathalie Japkowicz, eds. Advances in Artificial Intelligence: 22nd Canadian Conference on Artificial Intelligence, Canadian AI 2009, Kelowna, Canada, May 25-27, 2009 Proceedings. Vol. 5549. Springer, 2009.

107. Z. Nian, D. C. Wunsch, "Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach," *IEEE Trans. Fuzzy Syst.*, vol. 14, pp. 728-737, Dec. 2006.

108. Ming Gu; Yang Liu; Chakrabartty, S., "FAST: A simulation framework for solving large-scale probabilistic inverse problems in nano-biomolecular circuits," *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* , vol., no., pp.3160,3163, May 30 2010-June 2 2010

109. N. Zhang and D. C.Wunsch, II, "A fuzzy attributed graph approach to subcircuit extraction problem," in *Proc. IEEE Int. Conf. Fuzzy Systems*, St. Louis, MO, May 25–28, 2003, pp. 1063–1067.

110. Cordella, Luigi Pietro, et al. "An improved algorithm for matching large graphs." *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*. 2001.

111. Murta, Alan. "General polygon clipper library." The University of Manchester, United Kingdom (2010).

112. Dezso, Balázs, Alpár Jüttner, and Péter Kovács. "LEMON–an Open Source C++ Graph Template Library."

113. Fourie, C., "Full-gate verification of superconductive integrated circuit layouts with InductEx," Applied Superconductivity, IEEE Transactions on , vol.PP, no.99, pp.1,1, doi: 10.1109/TASC.2014.2360870

114. C. J. Fourie, "Inductex User's Manual" Stellenbosch University, Oct. 2012. [Online] http:www.inductex.info 100.

115. R. M. C. Roberts and C. J. Fourie, "Layout-to-schematic as a step towards layout-versus-schematic verification of SFQ integrated circuit layouts," in *Proc IEEE AFRICON*, Mauritius, pp. 898-902, 2013.

116. L. Ladage, R. Leupers, "Resistance Extraction Using a Routing Algorithm," *30th Design Automation Conf.*, pp.38-42, Jun. 1993.

117. Clark, David. "A Note On The Pseudoinverse.", 2009

118. Muller, Neil, Lourenço Magaia, and B. M. Herbst. "Singular value decomposition, eigenfaces, and 3D reconstructions." *SIAM review* 46.3 (2004): 518-545.

Stellenbosch University https://scholar.sun.ac.za

119. Boyer, John M., and Wendy J. Myrvold. "On the Cutting Edge: Simplified O (n) Planarity by Edge Addition." *J. Graph Algorithms Appl.* 8.2 (2004): 241-273.

120. Fourie, Coenrad Johann. *A tool kit for the design of superconducting programmable gate arrays*. Diss. Stellenbosch: University of Stellenbosch, 2004.

121. Fourie, Coenrad J., Mark H. Volkmann, and Thomas Weighill. "Parallel processing to speed up FastHenry for dense multilayer layouts."

122. Tolpygo, Sergey K., et al. "Inductance of circuit structures for MIT LL superconductor electronics fabrication process with 8 niobium layers." *arXiv preprint arXiv:1408.5828* (2014).

123. Chawla, Basant R., and Hermann K. Gummel. "A boundary technique for calculation of distributed resistance." *Electron Devices, IEEE Transactions on*17.10 (1970): 915-925.

124. Meyers, N.H., "Inductance in Thin-Film Superconducting Structures," *Proceedings of the IRE* , vol.49, no.11, pp.1640,1649, Nov. 1961

125. Sep 15, 2003 - Massachusetts Institute of Technology. 6.763 2003 Lecture 4. Lecture 4: *London's Equations*. Outline. 1. Drude Model of Conductivity. 2.

126. Kamon, Mattan, Michael J. Tsuk, and Jacob K. White. "FASTHENRY: A multipole-accelerated 3-D inductance extraction program." *Microwave Theory and Techniques, IEEE Transactions on* 42.9 (1994): 1750-1758.

127. *FLUXONICS Foundry for Design and Fabrication of RSFQ Circuits*, [online] Available: http://www.fluxonics-foundry.de

128. R. M. C. Roberts and C. J. Fourie, “Layout-versus-schematic verification for superconductive integrated circuits,” IEEE Trans. App. Supercond., accepted for publication.

129. Trefethen, Lloyd N., and David Bau III. Numerical linear algebra. Vol. 50. Siam, 1997.

Stellenbosch University https://scholar.sun.ac.za

Appendix A

Further information regarding the GDSII file format

This appendix comes from the 4th year engineering project done by the same author.

A. 1 GDSII record types

Record types can be grouped into categories which represent the hierarchical structure of the GDSII stream file.

Each of these categories will be briefly discussed below.

A. 1. 1 Header records

The highest level records are the file header records which are not to be confused with the header portion of each individual record. Each file can contain multiple libraries which could, for example, each represent a microchip on a wafer. The header records hold library information such as the date that the library has been modified etc. These records play a crucial part in netlist extraction since it is imperative to know when a library begins. Ideally each library will be represented by a netlist.

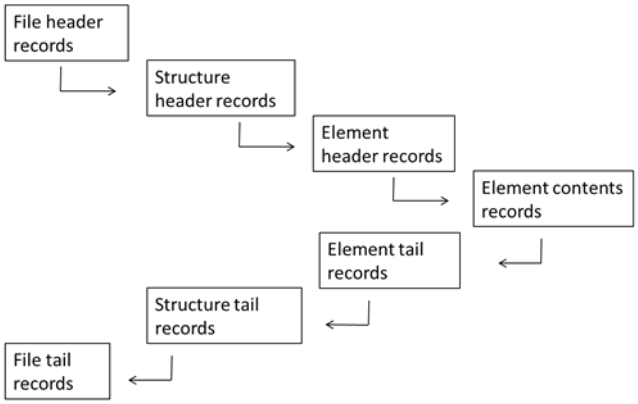


Figure A.1: Hierarchy of record types in GDSii stream file format.

A. 1. 2 Structure Header and Tail records

Structures header and tail records are used to indicate the beginning and end of a structure. These structures are comprised of groups of elements which together fully describe an object. There are various types of objects such as Text, Path or Box objects.

A. 1. 3 Element Header, Tail and Contents records

Element contents records are the core of GDSII files and are required to extract polygon and layer information. As with structure header and tail records, the element header and tail records indicate the beginning and end of a record.

Stellenbosch University https://scholar.sun.ac.za

A. 1. 4 Tail file record

This record type merely indicates the end of a library. This is not always at the same location as the end of file since some files are zero-padded.

In later chapters, GDSII file processing and parsing will be discussed. For consistency, consecutive records of the same hierarchical level will be called siblings and a record will be called a child record when it directly follows from a record of higher hierarchical position (Figure 52).

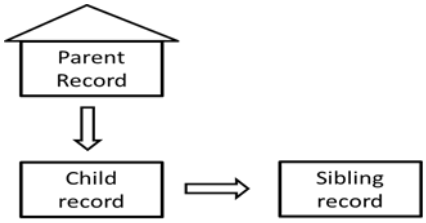


Figure A.2: Relationship between Parent, Child and Sibling records

Figure A.3 gives an example of the structure of the first part of a library in a GDSII file. The structure records are on the same hierarchical level and are therefore sibling records when in the same library. Boundary and text records, for example, are also sibling records when they represent elements a specific structure. Layer records are always children of element records and can have a variety of siblings – the most important one being the XY co-ordinate record which indicates the location of the element in relation to the defined origin.

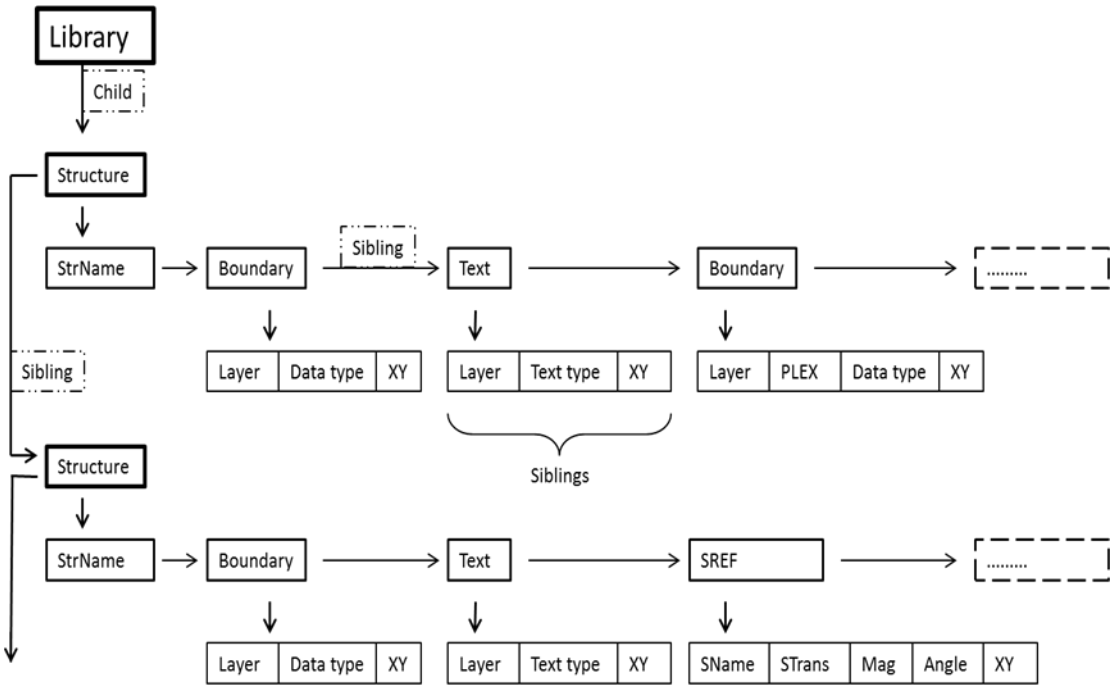


Figure A.3: Relationship between Parent, Child and Sibling records within a Library as part of a GDSii file.

Stellenbosch University https://scholar.sun.ac.za

A. 2 GDSII File Flattening

The gdsii stream file format has been designed to be read and processed sequentially. It is important to note that a structure will not be referenced if it has not yet been described earlier in the file. One can therefore read the entire file in one pass through the file. The file will be flattened in memory after it is read and stored. The flattened gdsii file will then be printed. The process is discussed in the following section.

A. 2. 1 File reading and storage structure

Each record's header must be processed and then a decision can be made as to how the record should be stored or if other records should be modified as a result of the record. The records in a gdsii file are all categorised as records that can be followed by child records and those that cannot. This information is used to determine how the record will be connected to the tree in memory. Child and sibling pointers are handled in such a way that each child is the start of a linked list of siblings.

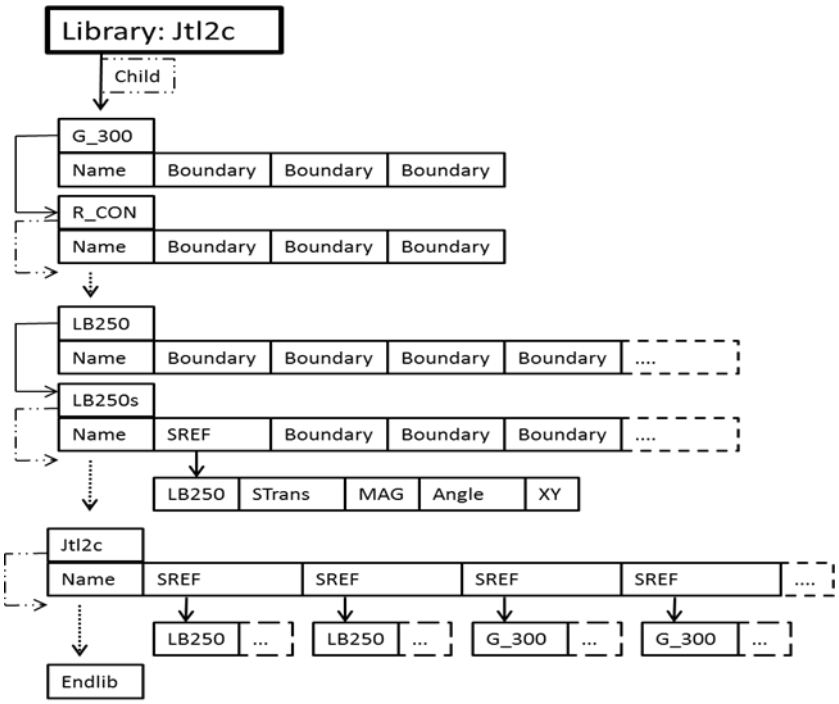


Figure A.4: Hierarchical structure of part of a JTL cell.

An example of part of a file that is read into memory in a tree structure is shown in Figure A.4, where the dotted arrows indicate other structures that are also found in the tree but are not shown in the diagram. The Structures (G_300, R_CON etc.) are on the same hierarchical level and stored as siblings. The Name elements are their children and the Boundaries and SREFs are the Name elements' children.

Stellenbosch University https://scholar.sun.ac.za

A. 2. 2 The Flattening process

The traversal of the tree is done recursively since recursion provides an elegant flattening algorithm. The Figure A.5 shows one of the final steps in flattening Jtl2c.gds file.

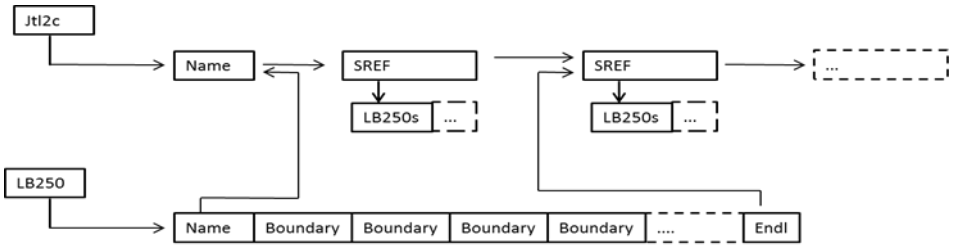


Figure A.5: Graphical representation of the flattening process. This figure is used in conjunction with the figure above.

A. 2. 2. 1 Two-dimensional special transformations

Between extracting the structure that is referenced and inserting it into the correct place in the file, all the transformations to this structure by the layout engineer have to be undone (Figure A.6). The STRANS record provides information as to if transformations have been done or not. The following transformation need to be provided for.

Flipping: This is done around the x axis, so the sign of the 'y' component of each co-ordinate pair is changed.

Rotation:
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Translation: Each point is defined from the origin, so xo and yo are simply added to the each co-ordinate pair of the sub-cell.

Scaling: This is done last. Each the x and y components of each point are multiplied by the scaling factor.

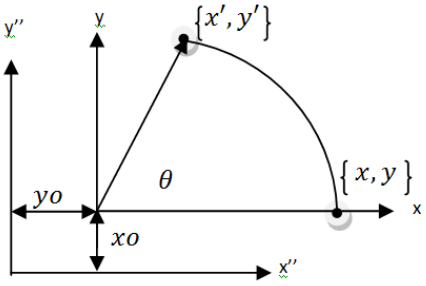


Figure A.6: Co-ordinate systems used in the flattening process.

Stellenbosch University https://scholar.sun.ac.za

A. 2. 2. 1 Code extract

```
int flatten_structure (Node *ptr_start_of_saved_structure){    // calls the recursive function below
        int replacements = 0;
        replacements = find_replace_srefs_new(&ptr_start_of_saved_structure, &ptr_start_of_saved_structure);
        return replacements;
}
int find_replace_srefs_new (Node **ptr_current,Node **ptr_start_of_saved_structure){
        int replacements = 0;
        Node *new_structure = NULL;
        Node *current_next = NULL;
        unsigned short int header_type;
        header_type = (short int)((*ptr_current)->header->record_type << 8)|((*ptr_current)->header->data_type);
        short int str_length = 0;
        int i = 0;
        // The following code traverses through the entire tree in search of SREFS
        if (header_type == SREF){ //This is a structure reference element that has to be transformed and inserted
                // 1: get copy of structure, and place in new_structure
                new_structure = get_copy_of_structure(*ptr_current, ptr_start_of_saved_structure);
                if (new_structure == NULL)
                        return -1;    // This is if a certain structure cannot be found. Gdsii_flatten exits
                // 2: store the current node's sibling in current_next, for future reference
                current_next = (*ptr_current)->sibling;          // this will be used later to re-connect the tree (step 6)
                // 3: free the current ptr's child: this would be the srefs's children (like rotation angle etc)
                memory_free_node((*ptr_current)->child);
                // 4: pointer to the new structure gets saved in the current pointer (ptr)
                (*ptr_current) = new_structure;
                // 5: traverse to the end of the new structure (all the elements in between will not be srefs)
                if ((*ptr_current)->sibling != NULL){    // have to check that it does not only have one element in it
                        header_type = (short int)((*ptr_current)->header->record_type << 8)|((*ptr_current)->header->data_type);
                        while (header_type != ENDSTR){       // traverses to end of structure (ENDSTR = last child)
                                if ((*ptr_current)->sibling != NULL){
                                        ptr_current = &((*ptr_current)->sibling);
                                        header_type = (short int)((*ptr_current)->sibling->header->record_type << 8)|((*ptr_current)-
>sibling->header->data_type);
                                }
                                else
                                        break;
                        }
                }
                // 6: we can now connect the previously stored current_next to the current pointer
                (*ptr_current)->sibling = current_next;
                replacements++;          //counts the number of replacements done
        }
        else
                if ((*ptr_current)->child != NULL)       // recursive function call for all children
                        replacements += find_replace_srefs_new (&((*ptr_current)->child), ptr_start_of_saved_structure);

        if ((*ptr_current)->sibling != NULL)     // recursive function call for all siblings
                replacements += find_replace_srefs_new (&((*ptr_current)->sibling), ptr_start_of_saved_structure);

        return replacements;
}
```

Stellenbosch University https://scholar.sun.ac.za

Appendix B

Layout to Schematic

Supporting screenshots, circuit and figures

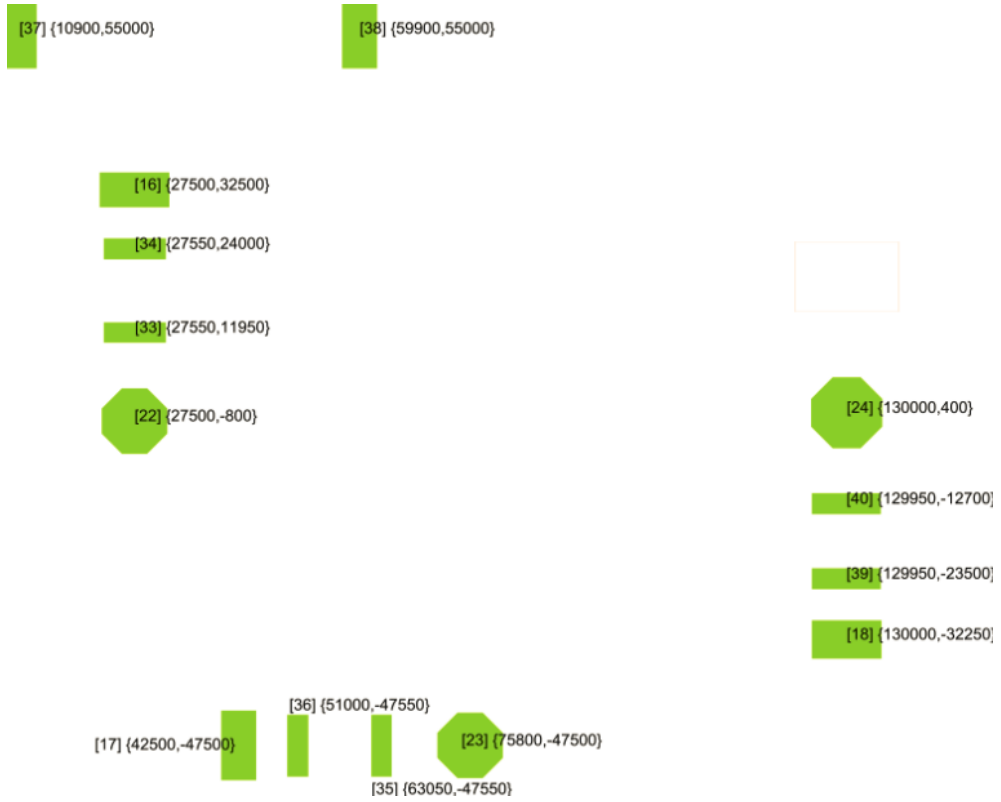


Figure B.1: Figure showing projected vias onto M2 of the Fluxonics process.

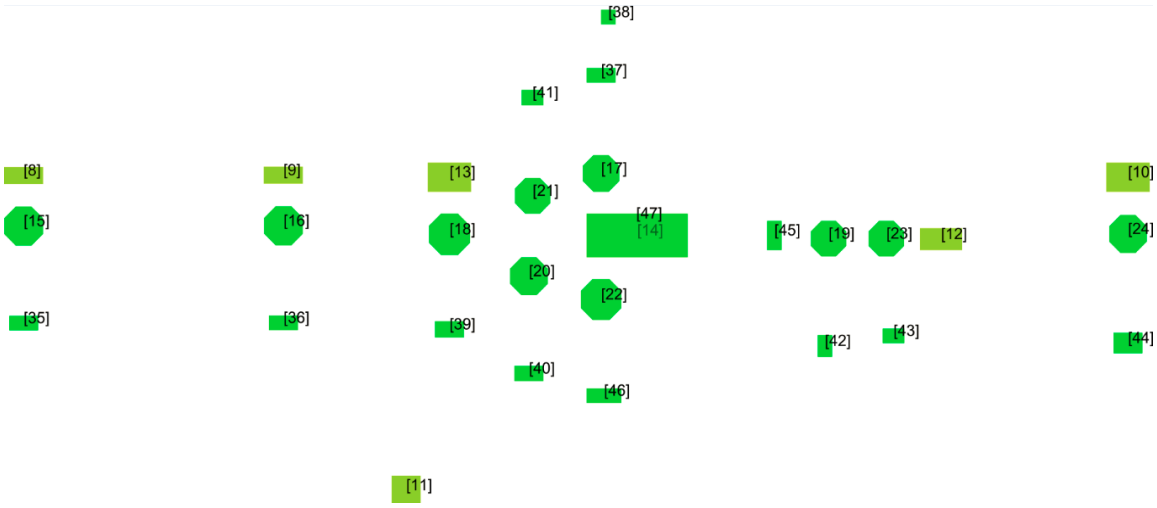


Figure B.2: Figure showing projected vias onto M1 of the Fluxonics process for a SFQ-DC and JTL.

Stellenbosch University https://scholar.sun.ac.za

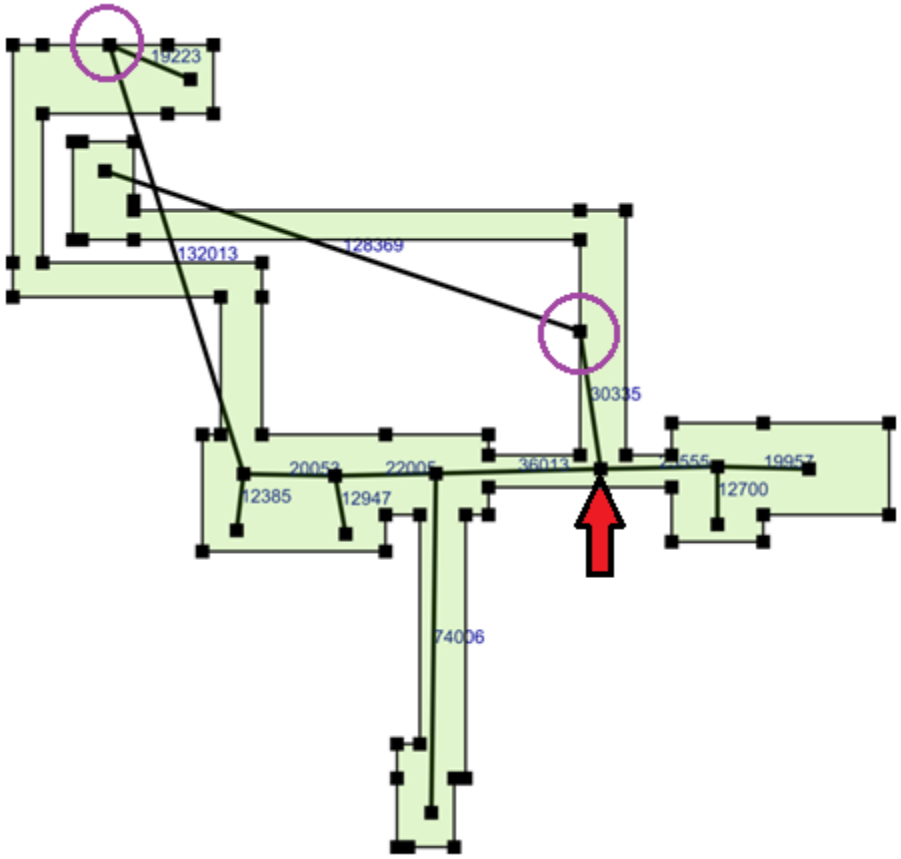


Figure B.3 Figure showing how components have been chosen for a specific polygon. The arrow points to a user selection node and the circled nodes are port nodes. This layout was extracted only with respect to inductance and not resistance – the port placement is therefore done to not include the bias resistors.

Stellenbosch University https://scholar.sun.ac.za

Appendix C

Circuit simplification examples

C. 1 Example 1: without ports

Given this arbitrary RLC netlist,

```
L1      0       1       1
C2      1       2       0.002
L3      2       3       3
R4      2       3       4
R3      3       4       4
L5      3       4       10
C7      4       5       0.002
L9      4       5       1
.end
```

we output this schematic:

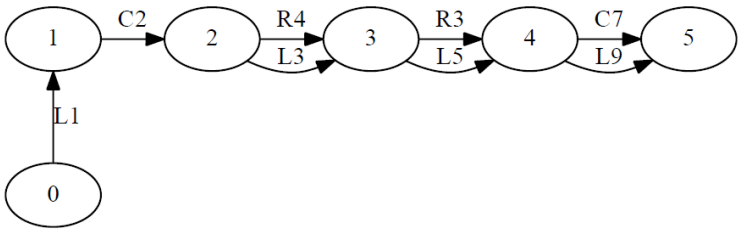


Figure C.1: Graph view of RLC netlist above

After Capacitors are removed and resistive components are converted to impedances (with no imaginary component) we have the following netlist,

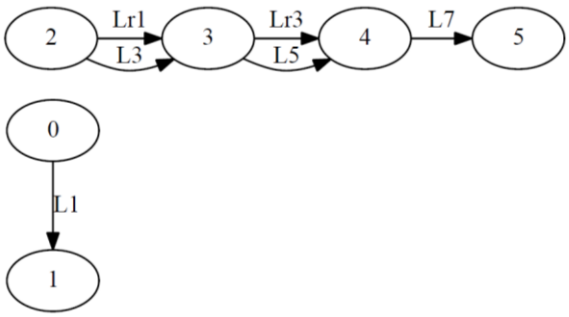


Figure C.2: Graph view of netlist in Figure C.1 after capacitors are removed.

which simplifies to an impedance and an inductor when parallel and series simplification is performed. This ficticious example is used simply to show the functionality of the netlist simplification tool.

Stellenbosch University https://scholar.sun.ac.za

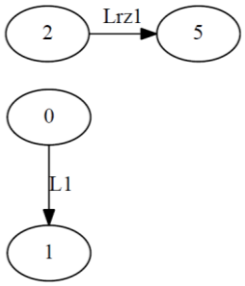


Figure C.3: Graph of completely simplified netlist in Example 1.

C. 2 Example 2: ports included

Input netlist

L1	0	1	1
C2	1	2	0.002
L3	2	3	3
R4	2	3	4
R3	3	4	4
L5	3	4	10
C7	4	5	0.002
L9	4	5	1
*Ports			
P1	1	0	
P2	2	0	
P3	3	0	
P4	4	0	
P5	5	0	
.end			

The visualization of the input netlist can be seen below:

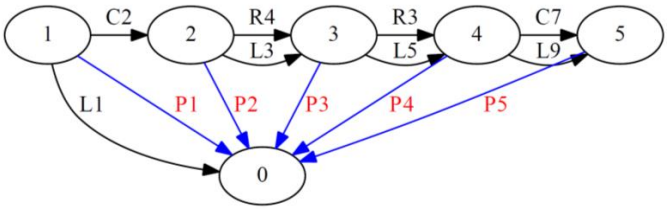


Figure C.4: Graph view of netlist for Example 2.

The capacitors (series and parallel) are removed (short circuited) and the resistors are converted to impedances and re-named.

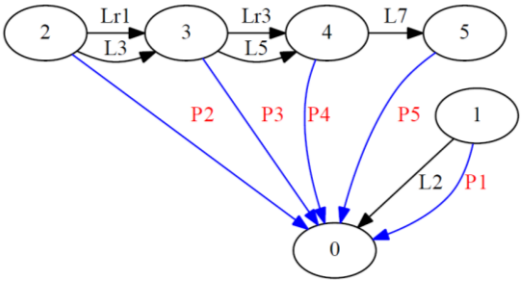


Figure C.5: Graph view of netlist for Example 2 after initial simplificaitons.

Stellenbosch University https://scholar.sun.ac.za

Parallel and series impedances are combined. The resulting impedances will typically have resistive and inductive components.

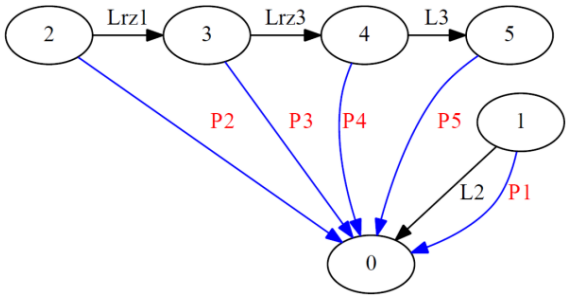


Figure C.6: Graph view of netlist for Example 2 after final simplifications.

C. 3 Example 3: components in parallel with ports removed

In the example below, all the simplification steps are performed. The inductor L5 is then removed since it is not required for parameter extraction.

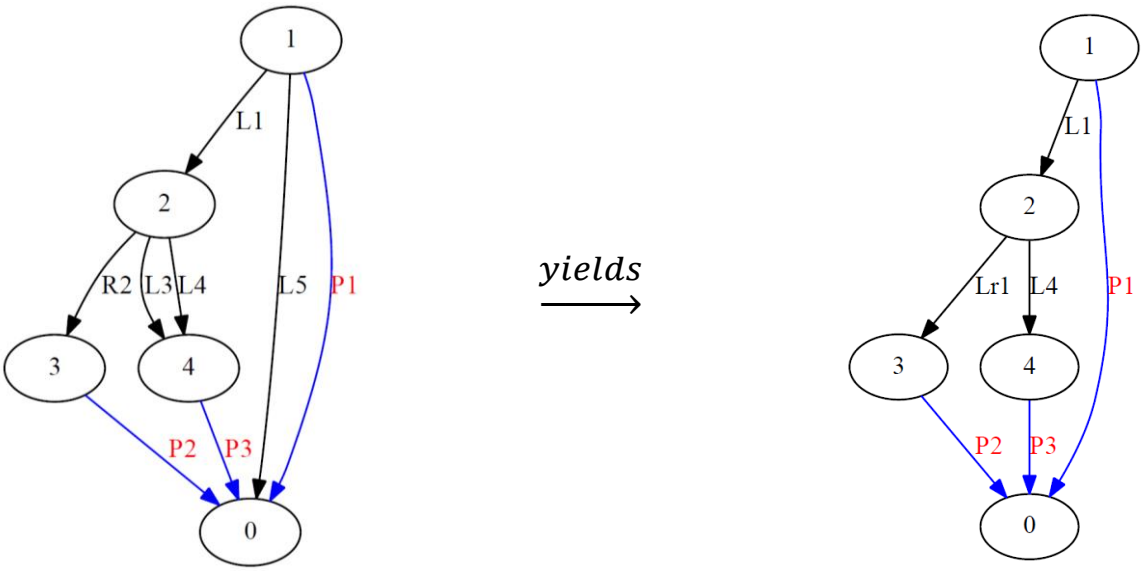


Figure C.8: Figure showing how port components replace components in parallel with these ports.

Should L5 be required, an extra inductance would need to be added in series with P1. This would require the addition of an extra node (vertex) between node 1 and 0.

Stellenbosch University https://scholar.sun.ac.za

Appendix D

Big O notation and Complexity classes

Both algorithm analysis and complexity theory are complex fields. Here I give a brief summary of only what is necessary from these fields to understand this thesis. The source for this section is the Textbook *Algorithms* [91].

Before implementing an algorithm, it is important to know how well it scales with respect to the input size, n. We would like to be able to answer the question: for which inputs will an algorithm be able to produce an output in a reasonable amount of time without exceeding the memory allowance? Big O notation can be used to give an approximate answer to this question by removing the complexity of taking processor speeds and architectures into account. An explanation Big O notation will first be given. Thereafter the relevant complexity classes will be discussed.

D.1 Big O notation

Big O notation provides us with a convenient way of benchmarking either the time or space complexity of an algorithm. We will introduce the notation by referring to time complexity, but the same definitions apply to space complexity.

Take, for example algorithm *X* and algorithm *Y*: *X* takes n operations to execute in the worst case and *Y* takes $4n$ operations. We could use a processor with quadruple the processing power to run *Y* in the same time that the original processor took to run *X* (here we assume that the same programming language was used and that memory requirements do not play a role). However large the input, we would still be able to solve both versions of the algorithm in *roughly* the same time. If, however, we introduce algorithm *Z* that takes n^2 operations, processor speed becomes less and less important as the input size increases – we wouldn't be able to compensate for the algorithm's complexity by improving the clock speed for the general case.

If we describe the complexity of a problem as a polynomial $c_0 n^{a_0} + c_1 n^{a_1} + \dots$, where $a_0 > a_1 > a_n$, then the only term that interests us is n^{a_0}: the constants $(c_0, \dots, c_n)$ and lower order polynomial terms are asymptotically dominated by the largest term, n^{a_0}. A simple list of 'rules' for big O notation is given below:

- Constants can be omitted
- Higher order terms in polynomials dominate those of lower order
- All exponentials dominate polynomials
- Polynomials dominate logarithms

Stellenbosch University https://scholar.sun.ac.za

D. 1. 1 Definitions

Given the two functions $f(n)$ and $g(n)$, if a constant, c_0, exists such that $f(n) \leq c_0 g(n)$, we can say that f grows no faster that g. In big O notation this is written as,

$$f = O(g)$$

This means that after applying the above rules, g provides an asymptotic upper bound for f. This is similar to saying $f \leq g$, but allows us to ignore the constant. The Big O notation analog to $g \leq f$ is

$$f = \Omega(g)$$

If both $f = O(g)$ and $f = \Omega(g)$ apply to f and g, we say,

$$f = \theta(g),$$

which is equivalent to

$$g = \theta(f)$$

D.2 Complexity classes

The complexity of an algorithm can be described as a function of its input. When we have a problem that needs to be solved by an algorithm, we would ideally like to find an algorithm which has an execution time that grows at most linearly with the input size, namely with $O(n)$ (keep in mind that constant time $O(1) = O(n)$ but $O(1) \neq \Omega(n)$). With more complex problems, linear algorithms are not always possible. We do, however, try to find algorithms that run in polynomial time: namely $O(n^k)$ where k is any positive integer. For an algorithm where k is very large, the algorithm could become unpractical for large inputs. However, this is still better than having an exponential algorithm that executes in $O(m^n)$ time. Even for small values of m such as 2 or 3, we would rather chose any polynomial – however large the k – because asymptotically the exponential will dominate.

D. 2. 1 Polynomial time

We say that decision problems (ones for which the answer is either 'yes' or 'no') that can be written in the form $O(n^k)$ are in P: they can be executed in polynomial time by a deterministic Turing machine (the computers we have today). A non-deterministic Turing machine is infinitely parallelizable (truly quantum computers would be classified as such).

D. 2. 2 Non-Deterministic Polynomial time

Decision problems that can be executed in in polynomial time by a *non-deterministic* Turing machine are said to be in NP (which stands for non-deterministic polynomial time) or alternatively $coNP$ (the class of problem whose complements lies in NP).

Another way of describing problems in NP is that if given the answer, 'yes' and a *certificate* (or proof), you can verify that the solution is correct in polynomial time (although you cannot necessarily generate the solution or certificate in polynomial time). If you can verify the answer, 'no' (when given a proof or certificate), in polynomial time, the problem falls into the $coNP$ class. Graph isomorphism

Stellenbosch University https://scholar.sun.ac.za

falls into the NP class: if we are provided with a bijective mapping between $G_1$and G_2, we can verify that it is correct in polynomial time.

All problems in P are also in NP, however, the reverse has not been proven (and many believe that this will never be proven: $N = NP$? is one of mathematics and computer science's greatest unanswered questions).

D. 2. 3 NP-complete and NP-hard

NP-complete problems are *the hardest problems* in NP. For algorithm X to be NP-complete, any algorithm in NP has to be reducible to X in polynomial time.

NP-hard problems are *at least as hard* as the hardest problems in NP. Problems that are in NP are by definition decision problems (such as, in our case, "are two graphs isomorphic?" or "is there a subgraph of G_1 in G_2?"). NP-hard problems, however, do not have to be decision problems (but can be). All NP-complete problems are therefore NP-hard but the converse is not true.

An example of an NP-hard problem is the problem of finding all embeddings of G_2 in G_1(subgraph isomorphism, but not in its decision problem form). This version of the subgraph isomorphism problem is *harder* than the decision problem (which is NP-complete), since *all* isomorphisms need to be found, not only one.

Any NP-complete problem can be reduced to any NP-hard in polynomial time. If one was to find a polynomial time solution for any NP-hard problem, you would be able to do so for all problems in NP-complete and would prove $P = NP$ (which, as mentioned before, has not been done).

D. 2. 4 GI, GI-complete and GI-hard

Because the graph isomorphism decision problem has not yet been proven to be solved in polynomial time, but is unlikely to be NP-complete, the complexity class GI has been defined as the set of problems that can be reduced to the graph isomorphism decision problem in polynomial time. The classes GI-complete and GI-hard can be difined similarly to their NP counterparts.

Some example of GI-complete and GI-hard problems are summarised in the table below:

GI-complete	P
Directed graphs (arks, not edges)	Planar graphs
Multigraphs (more than one edge per vertex pair)	Graphs with bounded valence
Hypergraphs (one edge may connect more than two vertices)	Permutation graphs
Bipartite graphs	k-trees
Complete graphs	circular arc graphs

Stellenbosch University https://scholar.sun.ac.za

Appendix E
Pi to Wye transform

In this appendix we first show the equivalence between the Pi model (using mutual coupling) and the Wye model. We then give the results of using this transform on a SFQ Splitter circuit.

E. 1 Pi to Wye

Here we show the equivalence of the Pi and Why model with respect to mutual coupling. We start by giving both models and the Kirchhoff voltage equations for the Wye model. The equations for the Pi model are then given. Finally the equations that link the two models are derived.

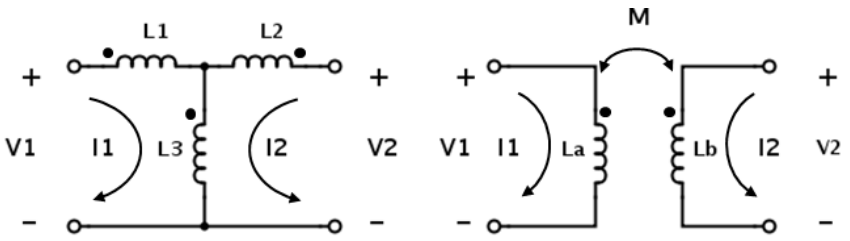


Figure E.1: The Wye model (left) and Pi model (right) are given. Polarities are defined according to the dot convention.

$$V_1 = Z_1 I_1 + Z_3(I_1 + I_2) = I_1(Z_1 + Z_3) + I_2 Z_3$$
$$V_2 = Z_2 I_2 + Z_3(I_2 + I_1) = I_1 Z_3 + I_2(Z_2 + Z_3)$$

In matrix form these equations can be written as

$$\boldsymbol{V} = \boldsymbol{Z}\boldsymbol{I}, \tag{9}$$

where $Z_i = j\omega L_i$ and,

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} Z_1 + Z_3 & Z_3 \\ Z_3 & Z_2 + Z_3 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}, \tag{9}$$

giving us the L matrix of the Wye model:

$$\boldsymbol{Z} = \begin{bmatrix} Z_1 + Z_3 & Z_3 \\ Z_3 & Z_2 + Z_3 \end{bmatrix} \tag{9}$$

For the Pi model , we can write,

$$\begin{aligned} V_1 &= Z_a I_1 - M I_2, \\ V_2 &= Z_b I_2 - M I_1 \text{ and,} \\ \boldsymbol{Z} &= \begin{bmatrix} Z_a & -M \\ -M & Z_b \end{bmatrix}. \end{aligned} \tag{9}$$

If we consider both $\boldsymbol{Z}$ matrices and solve for Z_a, Z_b and M, we have

$$Z_a = Z_1 + Z_3$$
$$Z_b = Z_2 + Z_3$$
$$M = -Z_3$$

E. 2 SFQ Splitter example

In this example we use the Hypres splitter layout (Figure 6.11) and circuit file that is provided as part of the advanced example on the InductEx website. As mentioned in Chapter 6 section 6. 1. 2, each

Stellenbosch University https://scholar.sun.ac.za

junction is grounded differently. Here we focus on the third junction's grounding which can be modelled in two ways: in a Pi configuration (using mutual coupling) or in a Wye configuration.

In the example netlist that is available on the InductEx website, the Pi configuration has been chosen to model this junction: the inductors L_{rj3} and L_{r3} are coupled with a coupling factor of K_1. For readability, we use the symbols L_a and L_b to represent these two inductors respectively. The symbol M is used to represent the extracted mutual coupling *inductance* (different to the M above). The parameter extraction results (using InductEx) for the entire netlist are given section F.5; the extracted values for the Pi model are given again in Table 1 (a).

The original netlist is modified to resemble the netlist extracted by the layout to schematic tool: the Pi configuration is transformed to a Wye configuration by (1) removing the coupling component (K_1), (2) adding an additional node (node 100), (3) connecting L_{r3} and L_{rj3} to node 100 and (4) connecting node 100 to ground via an additional component, L_{add}. These inductors are given the symbols L_1, L_2 and L_3 respectively (as seen in Table 1 (b)).

Table 1: Extracted results of inductors in (a) a Pi configuration (mutual coupling included) and (b) a Wye configuration.

(a)

symbol	given name	Extracted value
L_a	Lrj3	0.64964
L_b	Lr3	1.0058
M	M extracted	0.5029

(b)

symbol	given name	extracted value
L_1	Lr3	0.14673
L_2	Ljr3	0.50397
L_3	Ladd	0.50292

Table 2: Calculated values of the inductors in the (a) Wye configuration and (b) Pi configuration by the extracted results in Table 1 (a) and (b) respectively (by substituting into the equations derived in the previous section).

(a)

symbol	Calculated value using Pi to Wye
L_1	0.14674
L_2	0.5029
L_3	0.5029

(b)

symbol	Calculated value using Wye to Pi
L_a	0.64965
L_b	1.00689
M	0.50292

In Table 2 (a), the values of the inductors in the Wye configuration have been calculated by substituting the values from Table 1 (a) into the equations in the above section (Appendix F, 1.1). The same has been done for the Pi configuration (Table 2 (a)).

In Table 5 the percentage deviation from the actual values (with respect to the Pi configuration) are given. From the very small percentage deviation between the inductance values in the two models, we can conclude that either of these models can be used for junctions grounded in such a manner.

Table 3: Percentage deviation of the transformed netlist (using transform equations to calculate Pi inductances manually) from the extracted values of the original example netlist.

symbol	actual	calculated	percentage deviation
L_a	0.64964	0.64965	-0.002%
L_b	1.0058	1.00689	-0.108%
M	0.5029	0.50292	-0.004%

Stellenbosch University https://scholar.sun.ac.za

Appendix F
Results and discussion (A)
Supporting screenshots, circuit and figures

F. 1 Example 1: JLT without resistance extraction

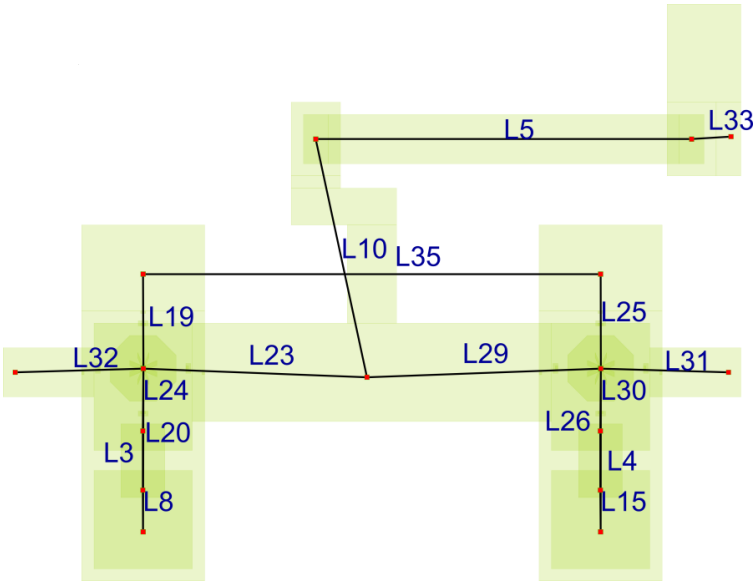


Figure F.1: Un-simplified schematic of a Fluxonics JTL cell that does not include the appropriate ports for shunt resistor extraction. The inductances and the bias resistors have been extracted.

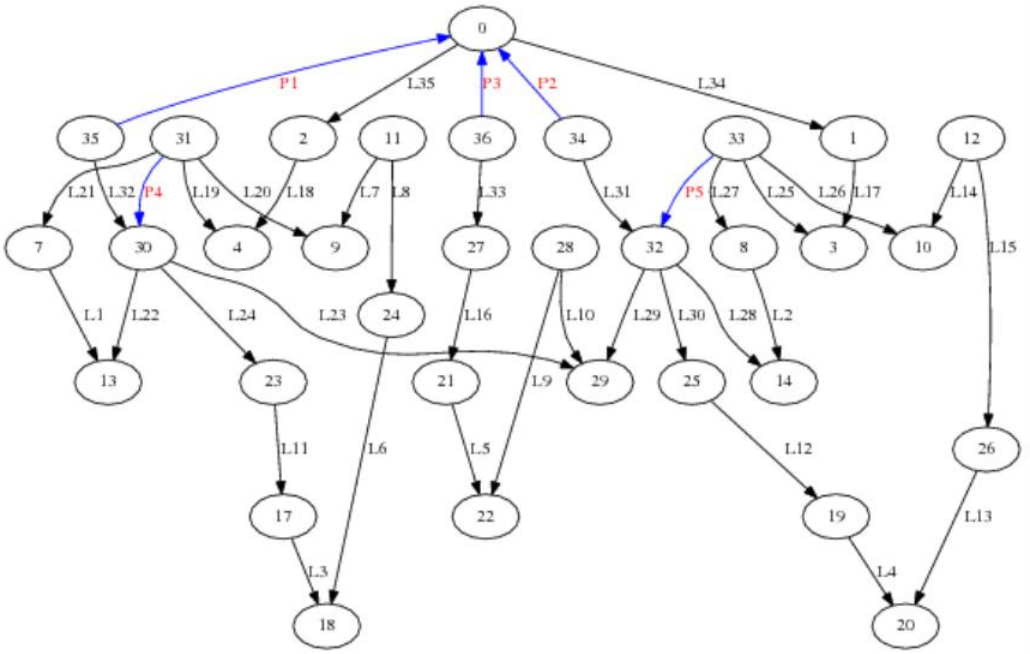


Figure F.2: The graph representation of the above JTL. Via simplifications have been performed but not series and parallel simplifications.

Stellenbosch University https://scholar.sun.ac.za

F. 2 Example 2: JLT with inductance and resistance extraction

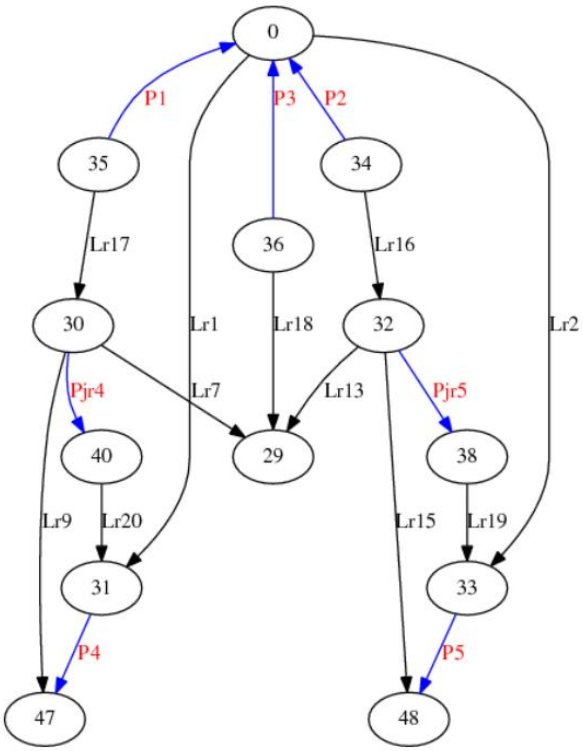


Figure F.3: Graph representation of the schematic in Figure 6.3. Ports Pjr4 and Pjr5 are in series with the shunt impedance.

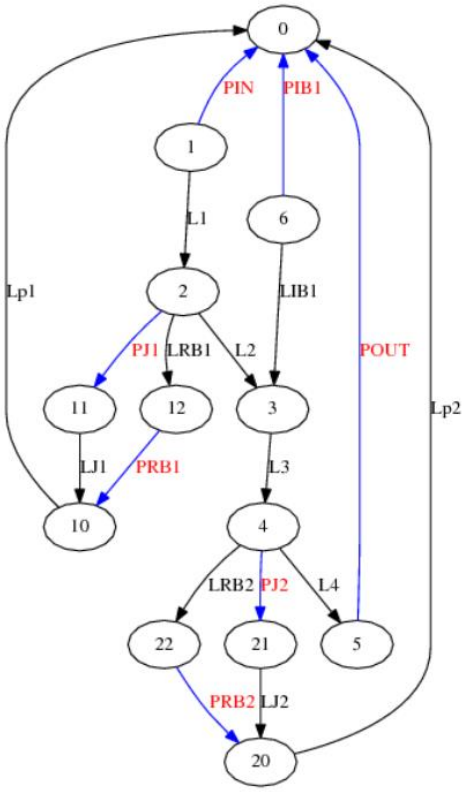


Figure F.4: Graph representation of original input schematic. Port PRB1 and PRB2 correspond to the Prj4 and Prj5 from the extracted schematic in Figure F.3.

Stellenbosch University https://scholar.sun.ac.za

F. 2. 1 *InductEx* solution to JTL with resistance included

```
Impedance    Inductance [pH]      Resistance [Ohm]   AbsDiff    PercDiff
Name         Design    Extracted  Design    Extracted (L only)   (L only)
L1           2.00000   2.02587    --        --        +0.0258    +1.29%
L2           2.00000   2.02828    --        --        +0.0282    +1.41%
L3           2.00000   2.02987    --        --        +0.0298    +1.49%
L4           2.00000   2.02358    --        --        +0.0235    +1.18%
Lj1          --        0.06201    --        --        +0.0620    --%
Lj2          --        0.06130    --        --        +0.0613    --%
Lrb1         1.00000   0.90318    --        1.204     -0.0968    -9.68%
Lrb2         1.00000   0.90290    --        1.204     -0.0971    -9.71%
Lp1          0.13000   0.16016    --        --        +0.0301    +23.20%
Lp2          0.13000   0.16037    --        --        +0.0303    +23.36%
Lib1         --        12.6456    --        7.391     +12.646    --%

Ports        Design    Extracted  AbsDiff   PercDiff
Pj1          250.000   251.240
Pj2          250.000   251.240
Prb1         --        1500.00
Prb2         --        1500.00
Deallocating memory.
Cycles found in 0.015 seconds.
SVD solution in 0.078 seconds.
Job finished in 47.315 seconds.
```

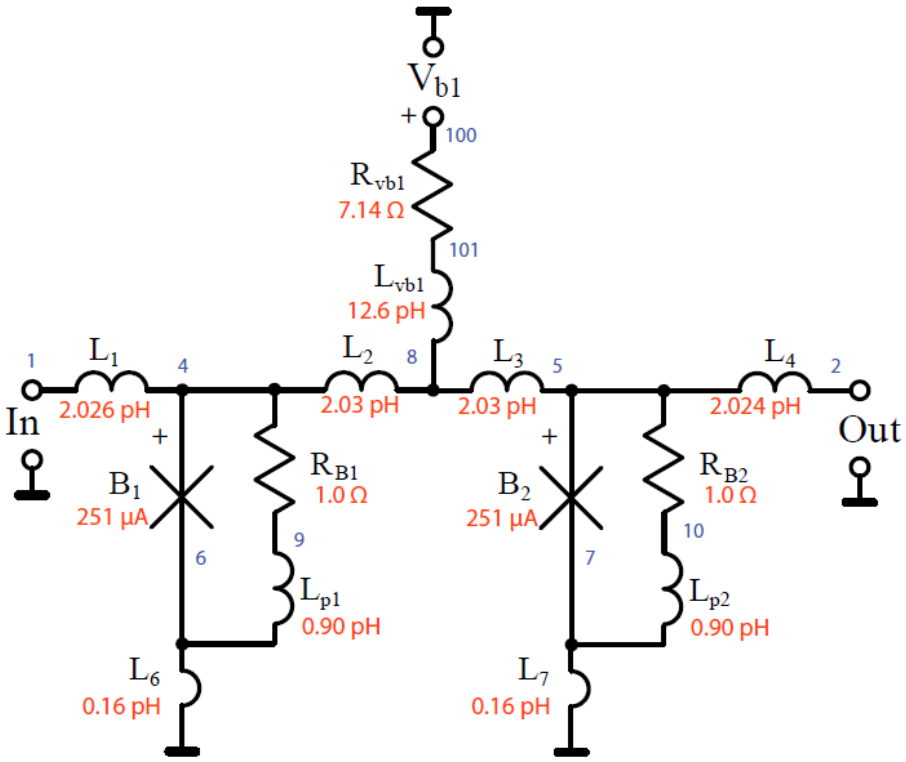


Figure F.5: Back-annotated schematic of a JTL. Schematic includes resistance and inductance parameters as calculated by InductEx.

Stellenbosch University https://scholar.sun.ac.za

F. 3 Example 4: Hypres SFQ Splitter with inductance and resistance extraction

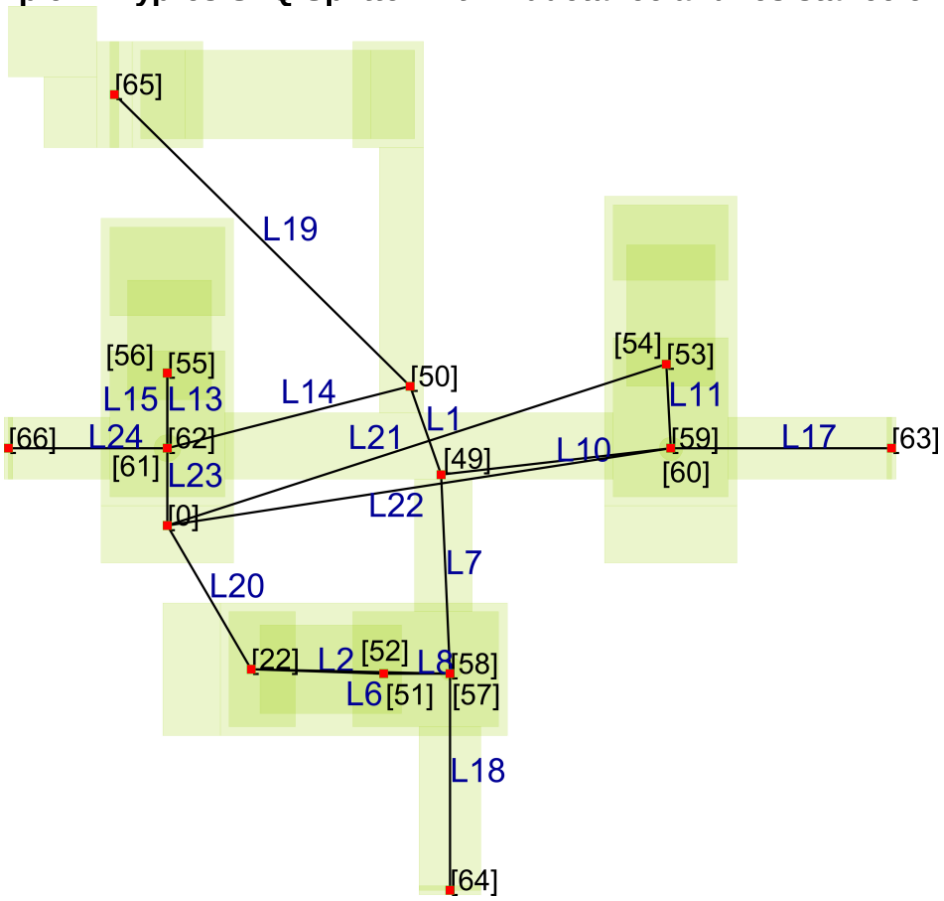


Figure F.6: Schematic representation of an extracted Hypres splitter cell.

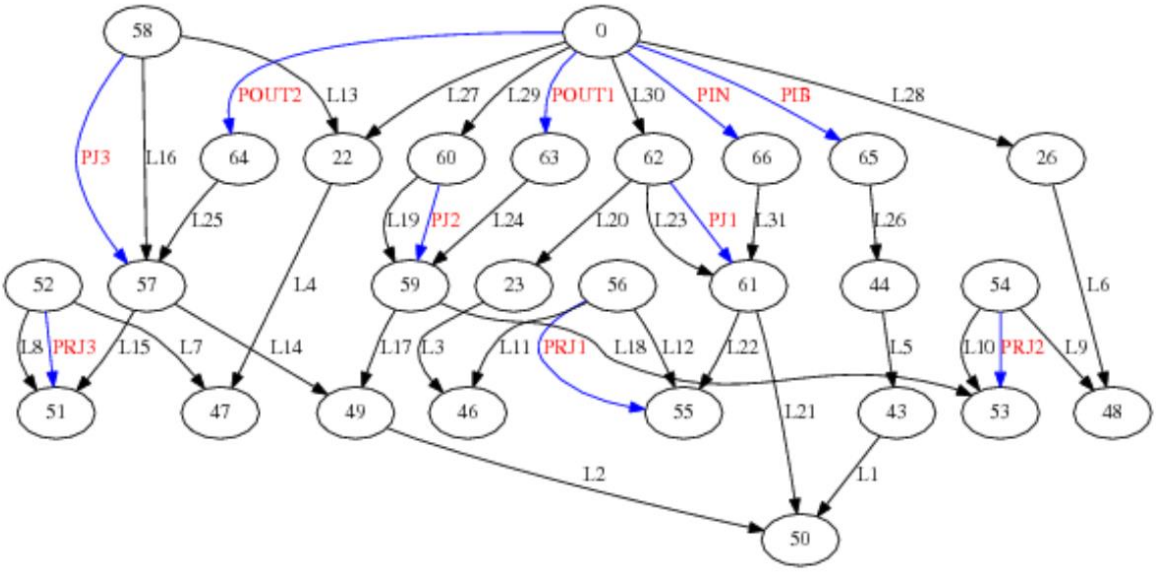


Figure F.7: Not fully simplified netlist of a Hypres splitter cell. Via components have been removed, but other simplifications are still required. This is the graph representation of the schematic in the figure above.

Stellenbosch University https://scholar.sun.ac.za

F. 4 Example 5: Confluence buffer with inductance and resistance extraction

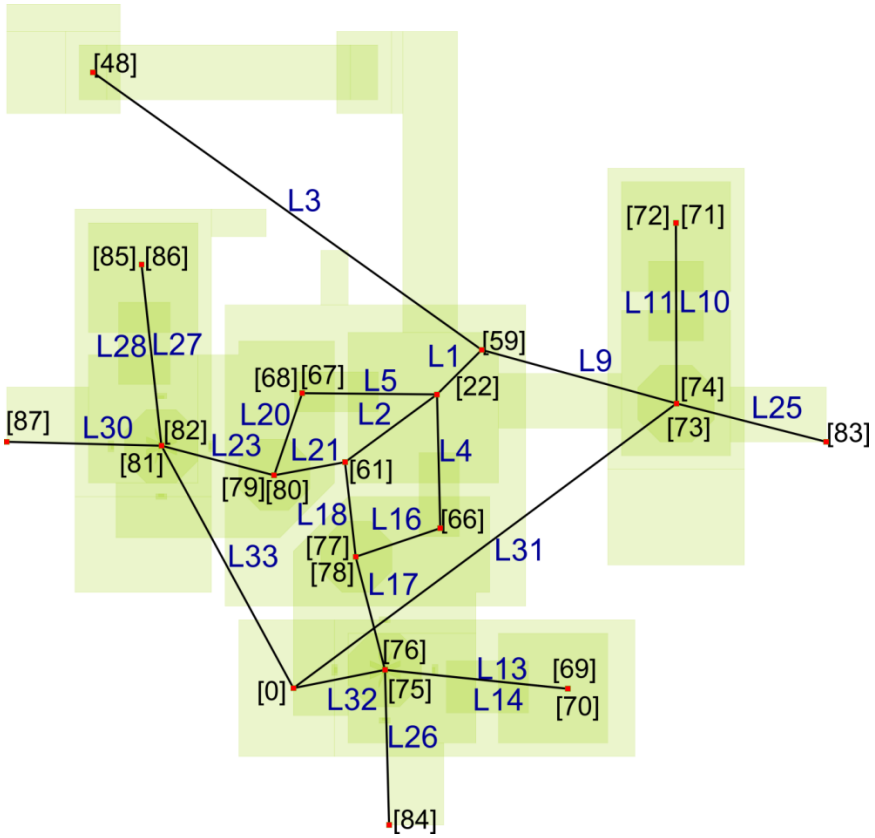


Figure F.8: Simplified (but not prepared for InductEx) schematic representation of an extracted Fluxonics confluence buffer cell. The node numbers and component names (for horizontal components) have been included.

Stellenbosch University https://scholar.sun.ac.za

F. 5. Example in Appendix E: mutual coupling

F. 5. 1 Without mutual coupling

F.5.1.1 Input netlist

```
* spice netlist for InductEx4
L1      1     2     1.36440
L2      2     3     1.89540
L3      3     6     0.33626
L4      6     7     1.85410
L5      7     9     1.94170
L6      6     10    1.84370
L7      10    12    1.98320
LJ1     5     25    0.03566
LRJ1    15    25    0.73352
Lp1     25    0     0.07540
LJ2     8     0     0.03154
LRJ2    18    0     0.61608
LJ3     11    0     0.64964
LRJ3    21    0     1.00580
LIB1    4     3     5.52520
K1      LJ3   LRJ3  0.3
* Ports
PIN     1     0
PJ1     2     5
PRJ1    2     15
PIB     4     0
PJ2     7     8
PRJ2    7     18
POUT1   9     0
PJ3     10    11
PRJ3    10    21
POUT2   12    0
.end
```

F.5.1.2 InductEx solution

Impedance	Inductance [pH]		Resistance [Ohm]		AbsDiff	PercDiff
Name	Design	Extracted	Design	Extracted	(L only)	(L only)
L1	1.36440	1.36835	--	--	+0.0039	+0.29%
L2	1.89540	1.90259	--	--	+0.0071	+0.38%
L3	0.33626	0.34234	--	--	+0.0060	+1.81%
L4	1.85410	1.85433	--	--	+0.0002	+0.01%
L5	1.94170	1.94609	--	--	+0.0043	+0.23%
L6	1.84370	2.04610	--	--	+0.2024	+10.98%
L7	1.98320	2.06564	--	--	+0.0824	+4.16%
Lj1	0.03566	0.03459	--	--	-0.0010	-3.01%
Lrj1	0.73352	0.72730	--	2.009	-0.0062	-0.85%
Lp1	0.07540	0.07314	--	--	-0.0022	-3.00%
Lj2	0.03154	0.02849	--	--	-0.0030	-9.68%
Lrj2	0.61608	0.60588	--	2.449	-0.0102	-1.66%
Lj3	0.64964	0.55225	--	--	-0.0973	-14.99%
Lrj3	1.00580	0.41761	--	2.159	-0.5881	-58.48%
Lib1	5.52520	5.50825	--	4.950	-0.0169	-0.31%

Ports	Design	Extracted	AbsDiff	PercDiff
Pj1	--	320.600		
Pj2	--	250.130		
Pj3	--	250.130		

Stellenbosch University https://scholar.sun.ac.za

```
Deallocating memory.
Cycles found in 0.031 seconds.
SVD solution in 0.718 seconds.
Job finished in 1.092 seconds.
```

F. 5. 2 Mutual coupling modelled in a Wye configuration

F. 5. 2.1 Input netlist

```
* spice netlist for InductEx4
L1      1     2     1.36440
L2      2     3     1.89540
L3      3     6     0.33626
L4      6     7     1.85410
L5      7     9     1.94170
L6      6     10    1.84370
L7      10    12    1.98320
LJ1     5     25    0.03566
LRJ1    15    25    0.73352
Lp1     25    0     0.07540
LJ2     8     0     0.03154
LRJ2    18    0     0.61608
LJ3     11    100     0.64964
LRJ3    21    100     1.00580
Ladd    100   0     1
LIB1    4     3     5.52520
* Ports
PIN     1     0
PJ1     2     5
PRJ1    2     15
PIB     4     0
PJ2     7     8
PRJ2    7     18
POUT1   9     0
PJ3     10    11
PRJ3    10    21
POUT2   12    0
.end
```

F. 5. 2.1 InductEx solution

```
Solution
Port                    Positive terminal     Negative terminal
Pin                     M2,   line along y;   M0,   same as "+" terminal.
Pj1                     M2,   polygon;        M1,   same as "+" terminal.
Prj1                    M2,   polygon;        R2,   same as "+" terminal.
Pib                     M2,   line along y;   M0,   same as "+" terminal.
Pj2                     M2,   polygon;        M1,   same as "+" terminal.
Prj2                    M2,   polygon;        R2,   same as "+" terminal.
Pout1                   M2,   line along y;   M0,   same as "+" terminal.
Pj3                     M2,   polygon;        M1,   same as "+" terminal.
Prj3                    M2,   polygon;        R2,   same as "+" terminal.
Pout2                   M2,   line along x;   M0,   same as "+" terminal.
Minimum filaments in FastHenry = 15342

Impedance    Inductance [pH]      Resistance [Ohm]  AbsDiff    PercDiff
Name       Design    Extracted Design    Extracted (L only)   (L only)
L1         1.36440   1.36435   --        --        -0.0000    -0.00%
L2         1.89540   1.89536   --        --        -0.0000    -0.00%
L3         0.33626   0.33626   --        --        -0.0000    -0.00%
```

Stellenbosch University https://scholar.sun.ac.za

```
L4          1.85410    1.85409    --         --         -0.0000    -0.00%
L5          1.94170    1.94173    --         --         +0.0000    +0.00%
L6          1.84370    1.84368    --         --         -0.0000    -0.00%
L7          1.98320    1.98321    --         --         +0.0000    +0.00%
Lj1         0.03566    0.03566    --         --         -0.0000    -0.01%
Lrj1        0.73352    0.73350    --         2.006      -0.0000    -0.00%
Lp1         0.07540    0.07540    --         --         -0.0000    -0.00%
Lj2         0.03154    0.03154    --         --         +0.0000    +0.01%
Lrj2        0.61608    0.61606    --         2.443      -0.0000    -0.00%
Lj3         0.64964    0.14673    --         --         -0.5029    -77.41%
Lrj3        1.00580    0.50397    --         2.426      -0.5018    -49.89%
Ladd        1.00000    0.50292    --         --         -0.4970    -49.71%
Lib1        5.52520    5.52517    --         4.939      -0.0000    -0.00%

Ports       Design     Extracted  AbsDiff    PercDiff
Pj1         --         320.600
Pj2         --         250.130
Pj3         --         250.130
Deallocating memory.
Cycles found in 0.047 seconds.
SVD solution in 0.312 seconds.
Job finished in 1.576 seconds.
```

F. 6 Results from Section B, Chapter 6

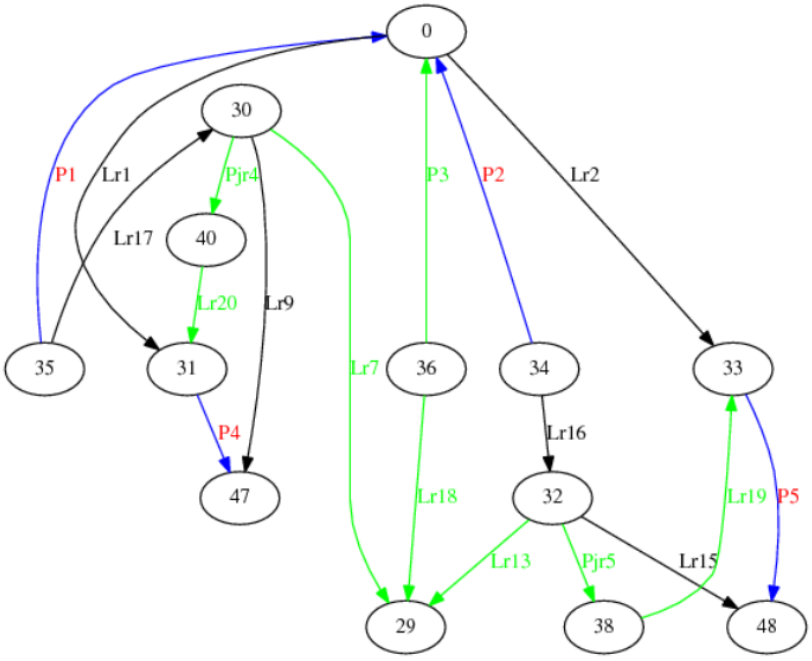


Figure F.9: Graphic result of the netlist comparison between Graph A and Graph D.

Stellenbosch University https://scholar.sun.ac.za

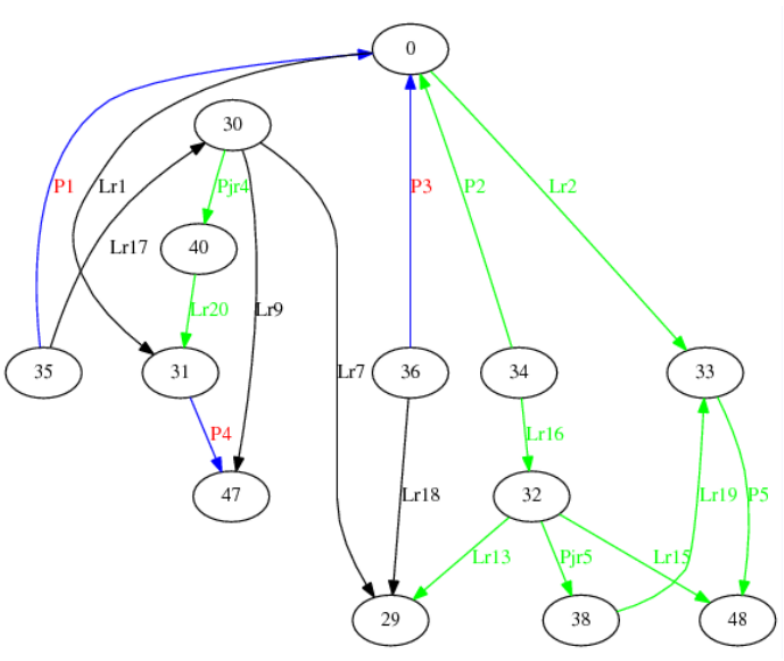


Figure F.10: Description identical to that of Figure 7: another one of the 32 mappings between Graph A and Graph D is shown.

```
Induced Subgraph Isomorphism Test:
There are 32 permutations for this 10 vertex graph!
(0, 0)(31, 31)(33, 33)(30, 30)(47, 47)(32, 32)(48, 38)(34, 34)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 33)(30, 30)(47, 40)(32, 32)(48, 48)(34, 34)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 33)(30, 30)(47, 40)(32, 32)(48, 38)(34, 34)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 34)(30, 30)(47, 47)(32, 48)(48, 32)(34, 33)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 34)(30, 30)(47, 47)(32, 38)(48, 32)(34, 33)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 34)(30, 30)(47, 40)(32, 48)(48, 32)(34, 33)(35, 35)(36, 36)
(0, 0)(31, 31)(33, 34)(30, 30)(47, 40)(32, 38)(48, 32)(34, 33)(35, 35)(36, 36)
(0, 0)(31, 33)(33, 31)(30, 32)(47, 48)(32, 30)(48, 47)(34, 35)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 31)(30, 32)(47, 48)(32, 30)(48, 40)(34, 35)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 31)(30, 32)(47, 38)(32, 30)(48, 47)(34, 35)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 31)(30, 32)(47, 38)(32, 30)(48, 40)(34, 35)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 35)(30, 32)(47, 48)(32, 47)(48, 30)(34, 31)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 35)(30, 32)(47, 48)(32, 40)(48, 30)(34, 31)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 35)(30, 32)(47, 38)(32, 47)(48, 30)(34, 31)(35, 34)(36, 36)
(0, 0)(31, 33)(33, 35)(30, 32)(47, 38)(32, 40)(48, 30)(34, 31)(35, 34)(36, 36)
(0, 0)(31, 34)(33, 31)(30, 48)(47, 32)(32, 30)(48, 47)(34, 35)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 31)(30, 48)(47, 32)(32, 30)(48, 40)(34, 35)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 31)(30, 38)(47, 32)(32, 30)(48, 47)(34, 35)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 31)(30, 38)(47, 32)(32, 30)(48, 40)(34, 35)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 35)(30, 48)(47, 32)(32, 47)(48, 30)(34, 31)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 35)(30, 48)(47, 32)(32, 40)(48, 30)(34, 31)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 35)(30, 38)(47, 32)(32, 47)(48, 30)(34, 31)(35, 33)(36, 36)
(0, 0)(31, 34)(33, 35)(30, 38)(47, 32)(32, 40)(48, 30)(34, 31)(35, 33)(36, 36)
(0, 0)(31, 35)(33, 33)(30, 47)(47, 30)(32, 32)(48, 48)(34, 34)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 33)(30, 47)(47, 30)(32, 32)(48, 38)(34, 34)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 33)(30, 40)(47, 30)(32, 32)(48, 48)(34, 34)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 33)(30, 40)(47, 30)(32, 32)(48, 38)(34, 34)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 34)(30, 47)(47, 30)(32, 48)(48, 32)(34, 33)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 34)(30, 47)(47, 30)(32, 38)(48, 32)(34, 33)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 34)(30, 40)(47, 30)(32, 48)(48, 32)(34, 33)(35, 31)(36, 36)
(0, 0)(31, 35)(33, 34)(30, 40)(47, 30)(32, 38)(48, 32)(34, 33)(35, 31)(36, 36)
```

Figure F.11: Figure showing the mappings from the solution text file for netlist comparison between graph G and graph A. Vertex 36 (the leaf node in graph G) is always mapped to vertex 36 in graph A.

Stellenbosch University https://scholar.sun.ac.za

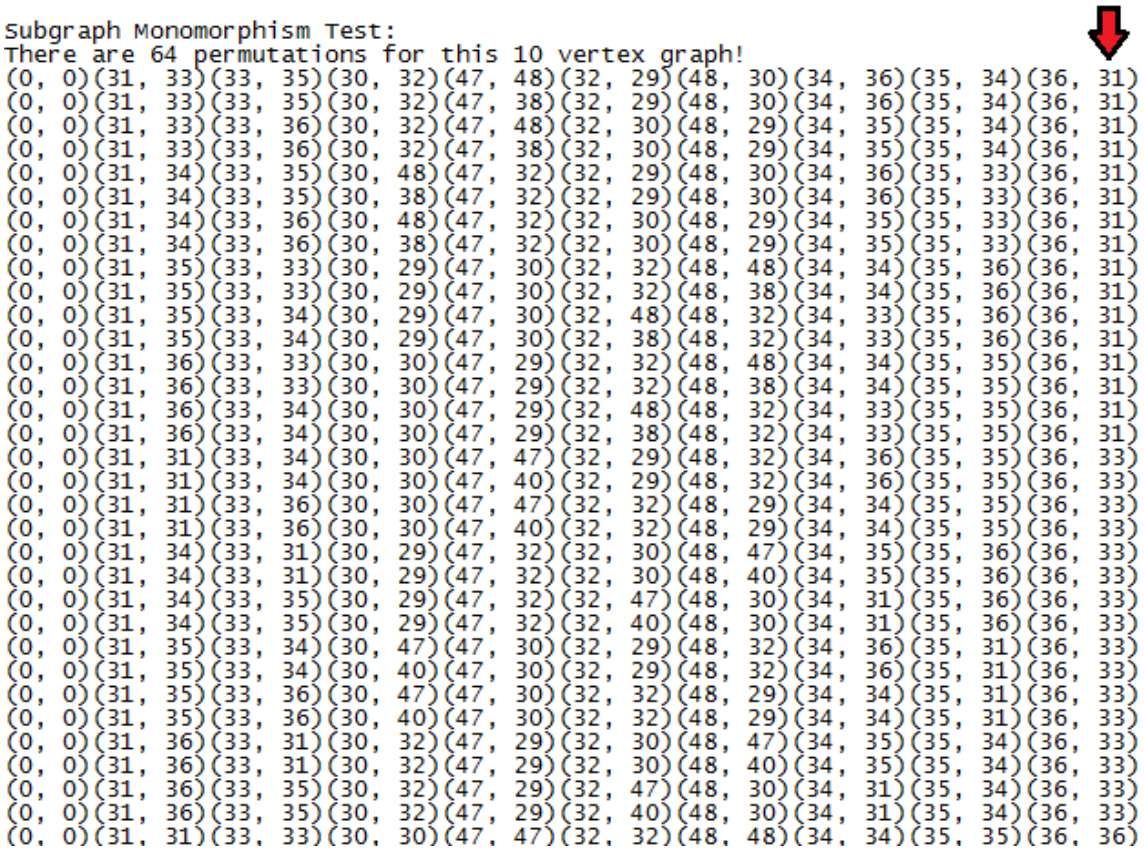


Figure F.12: Figure showing *a selection* of the mappings from the solution text file for netlist comparison between Graph G and Graph A. Vertex 36 (the leaf node in graph G) can be mapped to vertices 31, 33 or 36.

Stellenbosch University https://scholar.sun.ac.za

Appendix G
Graph theory
An overview of graph theory concepts used in this thesis

I am not content with algebra, in that it yields neither the shortest proofs nor the most beautiful constructions of geometry. Consequently, in view of this, I consider that we need yet another kind of analysis, geometric or linear, which deals directly with position, as algebra deals with magnitude.

- Gottfried W. Leibniz, 1670

Leibniz's study of position (also known as the field of 'topology'[80]) is believed to have sparked the beginnings of graph theory. It took more than half a century, however, for principles similar to that of Leibniz to be applied to a practical problem. Leonhard Euler is now considered by some to be the father of graph theory [81], [82] although he made no mention of vertices or edges, neither did he draw anything resembling the graphs we know today [83]. Euler solved the long-standing Konigsberg bridge problem in his 1736 paper [84] by proving that there is no way of reaching each land mass by crossing each bridge in Konigsberg only once. His universal solution to the bridge crossing problem for any number of bridges can be generalized to what we now know as an Eulerian Graph [81] although the actual graph drawing of the Konigsberg bridge problem defining the land masses as vertices and bridges as edges was only published in the late 1800s [85].

By the mid-1800s graph diagrams had already become increasingly popular. In 1847 Gustav Robert Kirchhoff extended Euler's work by developing the theory of trees [86]. He applied this theory to electrical networks [81] and developed the widely known voltage and current laws [87]. Soon after this, Arthur Cayley – now also well known for his work in the field of linear algebra – independently discovered trees [88]. In 1936 the first graph theory book was published and there were many to follow by the mid-1900s.

Although the early stages in graph theory's development were encouraged largely by puzzles and games (the four-colour conjecture, Hamilton's dodecahedron) [81], the introduction of computers have made it clear how useful graph theory can be to solve many practical problems.

In the following section, some basic graph theory principles will be presented as well as the terms that are required to follow the algorithms presented in later chapters. The books *Graph theory with applications to engineering and computer science* by Janet Barnett [82], *Introduction to graph theory* by Douglas West [89] and *Applied Graph Theory in Computer Vision and Pattern Recognition* [89] were used throughout this chapter. The information from these sources was combined to give a brief introduction to the graph theory concepts required to understand the remainder of this thesis.

Stellenbosch University https://scholar.sun.ac.za

G. 1 Terms and definitions

A graph, also called a linear graph, can be described by the *set* of vertices (V) and the *multiset* of edges (E) where $V = \{v_1, v_2, v_3, \dots, v_n\}$ and $E = \{e_1, e_2, e_3, \dots, e_m\}$. Each edge is made up of an unordered pair of vertices such that $e_k = (v_x, v_y)$. Because more than one edge may connect v_x to v_y, E is called a *multiset* or *collection.*

The diagrammatic representation of a graph $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ can be seen in Figure G.1, where the vertices (also called *nodes*) and edges (also referred to as *elements* or components) are labelled as numbers and letters respectively (although vertices can as be labelled as letters). Graph G_1 is a simple graph while G_2 is not. A simple graph contains no *loops* (also called *self-loops*) and does not have multiple edges per vertex pair (parallel edges). Although G_2 does not have any parallel edges, it contains a loop (e_9).

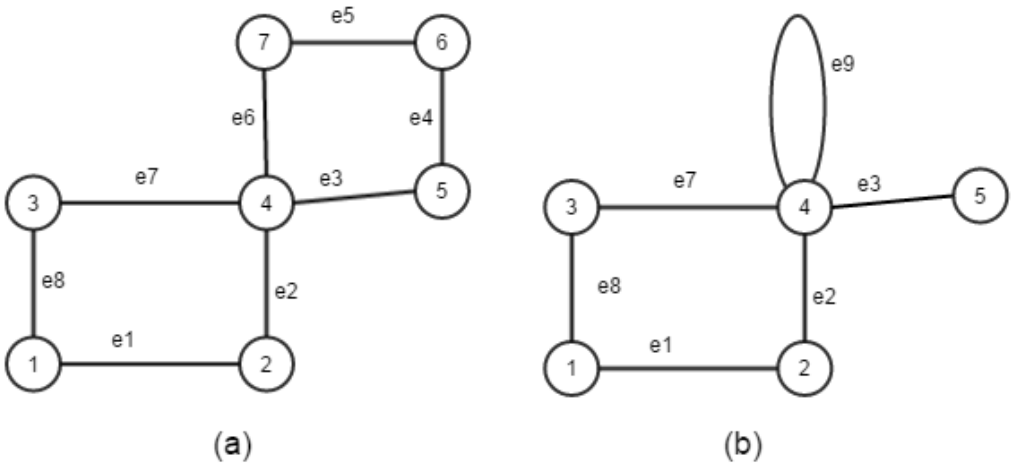


Figure G.1: (a) Graph $\boldsymbol{G_1 = (V_1, E_1)}$ is a simple graph containing two distinct cycles. (b) Graph $\boldsymbol{G_2 = (V_2, E_2)}$ contains 6 edges (one of which, $\boldsymbol{e_9}$, is a loop). Without $\boldsymbol{e_9}$, $\boldsymbol{G_2}$ would be a subgraph of $\boldsymbol{G_1}$.

The *degree* of a vertex v_k, denoted by $d(v_k)$, is the number of edges that have v_k as one of their end vertices. If two edges share *only* v_k, they are adjacent. If they share both end vertices, the edges are parallel edges. A vertex such as v_5 in G_2 has a degree of 1 and is called a *leaf*.

A list of other important terms and their descriptions is now given.

G. 1. 1 Graph traversal definitions

Graph traversal is the problem of visiting all the vertices in a graph in a certain way. Checks or updates to the edge parameters can as part of this process. Here are a few definitions with regards to graph traversal that are required to understand concepts in this thesis.

- A *walk* is a finite sequence consisting of alternating vertices and edges. It starts at v_{i0} and ends at v_{ik}. If $v_{i0} \neq v_{ik}$ the walk is open. If not, it is a closed walk.
- A walk that does not traverse an edge more than once is a *trail.*

Stellenbosch University https://scholar.sun.ac.za

- A walk that does not consider a vertex more than once (other than the initial vertex if it is a closed walk) is a *path* (also called a *simple path*).
- A *trail* where $v_{i0} = v_{ik}$ is a *circuit* and a closed *path* is a *cycle*. Note that a *loop* in circuit theory (a Kirchhoff current loop for example) is actually a *cycle* in graph theory. In G_1 an example of a circuit is $v_1, e_1, v_2, e_2, v_4, e_3, v_5, e_4, v_6, e_5, v_7, e_6, v_4, e_7, v_3, e_8, v_1$where v_4 is repeated (disqualifying it from being a cycle). An example of a cycle would simply be v_1, v_2, v_4, v_3, v_1(the edges may be left out for simplicity when there are no parallel edges).
- Two cycles are called distinct if one is not a cyclic permutation of the other. In circuit theory, we are most interested in distinct cycles that are also unique with respect to direction (reverse ordered cyclic permutations are also considered the same cycle).

G. 1. 2 Terms that describe types of graphs

Some important graph type definitions are given below.

- A *multigraph* is a graph with no self-edges (loops) but may have parallel edges.
- A *directed graph* $G_d = (V, A)$ is a graph where the set of edges is replaced with a set of *arcs* which contains *ordered* pairs of vertices. Arcs are often represented graphically by a line with an arrow.
- A *series-parallel* graph (SPG) is a directed multigraph graph that can be reduced to a loop by performing series and parallel operations on the graph. A series operation deletes a vertex of order two and replaces the vertices on either side of it with an edge. A parallel operation removes the edges between two vertices and replaces them with one edge.
- A *weighted* graph has a *weight function*, w: $E \rightarrow \boldsymbol{R}$ which assigns a weight to each edge.
- A *bipartite* graph is a graph where V is partitioned into two sets, V_1 and V_2. For edge $(u, v) \in E$, either $u \in V_1$ *and* $v \in V_2$ or $u \in V_2$ *and* $v \in V_1$.
- A graph where each vertex is of the same degree is a *regular* graph.
- A regular graph where $d(v_k) = 2$ is called a *cycle*. *Cycle* graphs contain a single cycles and are given the names $C_3, C_4, C_5, \ldots, C_i$ where i is the number of vertices.
- A *simple* graph with N vertices contains every possible edge (between all N vertices in V) is called a *complete* graph, denoted K_N.
- A *subgraph* $G' = (V', E')$, of $G = (V, E)$ exists when $V' \subseteq V$ and every $e_i' \subseteq E$.
- In a *disconnected* graph, there is not a path between every vertex and every other vertex; in a *connected* graph there is.
- A *tree* is a connected simple graph which contains no cycles (*acyclic*).
- A *spanning tree* is a subgraph of G that contains every vertex in G and is a tree.
- A set of disjoint trees is a *forest.*

Stellenbosch University https://scholar.sun.ac.za

G. 2 Important existing algorithms

Specific existing algorithms that are used in later chapters will be presented in this section. The main source for this section is the book *Algorithms* [91]. A brief introduction to graph traversal (as defined above) is first presented.

Graph traversal is usually part of a more complex problem such as looking for the best path from one edge to another or finding the minimum spanning tree. The aim of graph traversal, it self, is usually to find all the reachable vertices from a given root vertex or set of vertices. Two graph traversal algorithms that form the basis of many other algorithms are breath first search (BFS) and depth first search (DFS). In BFS traversal, all the vertices that share one common edge with a source vertex (the source vertex's neighbours) are traversed first and thereafter all the neighbour's neighbouring vertices are traversed until finally all vertices are *finished* (in BFS a vertex is *finished* once all its neighbours are *discovered*). In DFS, the traversal is performed from the source node to a child node of the source and then to a child of this node (and so forth) until a *leaf* has been reached or a cycle has been found. The algorithm then back-tracks to a node that it has not finished exploring and repeats the process until the entire graph has been traversed.

Both BFS and DFS are uninformed searches and can be implemented to take, at worst $O(|e| + |v|)$. The type of problem we would like to solve (and type of graph: tree or cyclic, directed or undirected, dense or sparse etc) determines which approach is superior for a given situation. For example: the principle of BFS is commonly used when finding the shortest path between a vertex and all the other vertices, while DFS is good for finding cycles in a graph. Both can be used to find spanning trees, but depending on the type of spanning tree, the choice of approach will differ. In this thesis, we use three algorithms that rely heavily on graph traversal:

- the *shortest path algorithm* between a vertex and all the other vertices in an undirected, weighted graph
- an algorithm for finding all the cycles in an undirected graph
- the minimum spanning tree algorithm

These three algorithms will be briefly discussed here, followed by an introduction to the concept of graph, subgraph and induced subgraph isomorphism in the next section.

G. 2. 1 Shortest path algorithms

One of the most well-known shortest path algorithms is Dijkstra's algorithm [92]. We can think of Dijkstra's shortest path algorithm as a more general solution of BFS for weighted graphs. An optimised implementation of Dijkstra's algorithm using a priority queue is available from boost::graph's C++ library [103]: this is the implementation that is used in the layout to schematic tool.

Stellenbosch University https://scholar.sun.ac.za

Other shortest path algorithms exist, such as the Bellman–Ford [93], Floyd–Warshall [94], and the A^* algorithm [95]. The Bellman–Ford and Floyd–Warshall algorithms can be thought of as even more general than Dijkstra's algorithm since they can support graphs with negative weights. Although they have been designed for directed graphs; for undirected graphs, the edges can simply be replaced by two arks in opposite directions.

Bellman–Ford is slower than Dijkstra's algorithm, and because all weights in our graphs are positive, Dijkstra's algorithm is superior to Bellman–Ford for our application. The Floyd–Warshall algorithm is often used to find the shortest path between all vertices (not between the source and all vertices as in Dijkstra). This is, in fact, what we need to calculate (this will be explained in Chapter 3). Execution time plays a role in algorithm choice (see Appendix D): since Floyd–Warshall is $O(|\mathrm{v}^3|)$ and Djikstra is $O(|\mathrm{e} * \mathrm{v}| + |\mathrm{v}|\log|\mathrm{v}|)$ but needs to be calculated V times, resulting in $O(|\mathrm{e} * \mathrm{v}| + |\mathrm{v}^2|\log|\mathrm{v}|)$, Djikstra is still be faster for the worst case scenario.

Another option is to use the A^* algorithm (which relies on a heuristic as well as the weight). The effectiveness of this algorithm is based heavily on the choice of heuristic and the algorithm is more popular for cases where the shortest path between *one pair* of vertices is required. An example of this would be choosing the heuristic to be the Euclidean distance between two vertices. In cases where there is no direct path (there is an obstacle in the way), the actual path weight between the vertices is *larger* than the Euclidean distance between them: the heuristic would guide the path finding algorithm towards the vertex that gives the shortest path *as the crow flies*. In our case, we calculate the shortest paths between the source vertex and all the other vertices which renders this solution to be less useful.

We can think of Dijkstra as a special case of A^* where there is no heuristic. Any time gain we could win by implementing A^* would most likely not be worth the effort of finding a suitable heuristic for our application – especially because we need to find all the shortest paths.

G. 2. 2 Cycle-finding algorithms

A DFS like approach is useful for finding the cycles in a graph. One of the first cycle-finding algorithms to use this approach was Tiernan's algorithm[96] (which is exponential with respect to the number of vertices). Shortly after Tiernan's publication, Tarjan [97] improved Tiernan's algorithm (by including an efficient pruning strategy) so as to give $O(\mathrm{e}(\mathrm{c}+1))$ for a graph of v vertices, e edges and c cycles [98].

Two years later, B. Johnson [99] built on the work of Tieman and Tarjan's and published his algorithm that can compute all elementary cycles in $O((\mathrm{n}+\mathrm{e})(\mathrm{c}+1))$. For 40 years, no vast improvements were seen to Johnson's algorithm. Finally, in 2013, R. Ferreira et al presented an asymptotically optimal solution for undirected graphs that can be completed in $O(\mathrm{e}+(\mathrm{c}+1)\mathrm{e})$ [100].

Stellenbosch University https://scholar.sun.ac.za

As mentioned in the previous section, finding all elementary cycles is required as part of the parameter extraction process. A simple recursive implementation, similar to that of Tarjan, was programmed in C and although a better implementation strategy could have been used, the time taken to calculate the cycles is much less than that of the other steps in parameter extraction. The implemented cycle finding algorithm (Appendix A) could be improved at a later stage if ever necessary.

G. 2. 3 Minimum Spanning Tree (MST) algorithms

Since the first minimum spanning tree algorithm was documented in 1926 [101], mathematicians and algorithm developers have been attempting to reduce the computational complexity of this problem. For undirected graphs, Kruskal and Prim's algorithms are currently the two most popular polynomial time algorithms used to solve the minimum spanning tree problem. Other MST algorithms can be almost linear, but are slightly more complicated to implement [102] (and can involve bit operations of edge weights and randomisation techniques).

Kruskal's algorithm is the easier of the two to implement; it was implemented in the first version of the layout to schematic tool, but when my own algorithm implementations were replaced with the boost::graph library's functions [103], Prim's algorithm could be used.

Kruskal can run in $O(|e|\log|e|)$ time and relies on the principle of adding the edge with the lowest weight to the minimum spanning forest of the graph until all vertices are spanned. Prim, on the other hand, can run in $O(|e|+|v|\log|v|)$ if a Fibonacci heap (a sophisticated data structure that is difficult to implement[91]) is used. Instead of (like Kruskal) growing several MSTs and combining them, Prim "grows" one MST by using an approach similar to DFS.

When there are many more edges than vertices, Prim's algorithm (even without a Fibonacci heap), performs better than Kruskal, giving $O(|e|\log|v|)$. The graphs of which the MST is required are typically dense and in many cases will have an edge between each vertex and each other vertex. Prim's algorithm is therefore a better choice.

G. 3 Graph matching and Isomorphisms

A morphism or homomorphism is a structure preserving map that needs to be defined for specific applications. Two specific type of morphisms that we are interested in to perform netlist comparison is that of graph isomorphism and subgraph isomorphism. (Sub)graph isomorphism is one of the most interesting and famous problems in graph theory. Mathematicians and computer scientists alike have spent many man-hours in search of a polynomial time solution to the graph isomorphism problem. Many have also tried to prove that such solutions do not exist for the general graph isomorphism case, but to date have failed to do so.

Before the problem is further discussed, some important definitions are given below with respect to the simple, undirected graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. In this section, the sources that

Stellenbosch University https://scholar.sun.ac.za

have been used extensively for definitions and theory are the dissertation *A constraint programming approach to subgraph isomorphism* [104] and the textbooks *The Algorithm Design Manual* [105] and *Advances in Artificial Intelligence: 22nd Canadian Conference on Artificial Intelligence* [106]:

- An *isomorphism* between the, G_1 and $G_2 = (V_2, E_2)$ is a bijective mapping, $f: V_1 \rightarrow V_2$, where $(v_1, v_2) \in E_1 \Leftrightarrow \big(f(v_1), f(v_2)\big) \in E_2$. G_1 and $G_2$are isomorphic if and only if such an isomorphism between $G_1$and G_2exists.
- An *automorphism* can be thought of as a measure of symmetry for a graph. It is a specific case of isomorphism but from one graph to itself.
- A *subgraph isomorphism* from (V_1, E_1) to (V_2, E_2) is an injective mapping $f: V_1 \rightarrow V_2$ such that for any $u, v \in V_1$, $\big(f(u), f(v)\big) \in E_2$ if $(u, v) \in E_1$. General subgraph isomorphism is also commonly referred to as subgraph monomorphism [104]
- An *induced subgraph isomorphism* from (V_1, E_1) to (V_2, E_2) is an injective mapping $f: V_1 \rightarrow V_2$ such that for any $u, v \in V_1$, $\big(f(u), f(v)\big) \in E_2$ if and only if $(u, v) \in E_1$.

To avoid ambiguity, non-induced subgraph isomorphism will be referred to as subgraph monomorphism for the remainder of this thesis.

The difference between isomorphism and the subgraph morphisms is that graph isomorphism only applies to graphs of the same size. The difference between subgraph isomorphism (or monomorphism) and induced subgraph isomorphism is that if an edge *does not* exist between two vertices in the smaller graph but does in the subgraph of the larger graph the following can be said:

- The answer to the induced subgraph decision problem is no.
- The answer to the subgraph isomorphism (or monomorphism) problem is yes.

In layout versus schematic, we are interested in not only the general graph isomorphism problem but also in the subgraph monomorphism and induced subgraph isomorphism problems. In general graph isomorphism we look for an exact match between the extracted schematic's graph and the input schematic's graph. Because the simplified (and unsimplified) versions of these graphs are usually different in size, a subgraph approach is normally required.

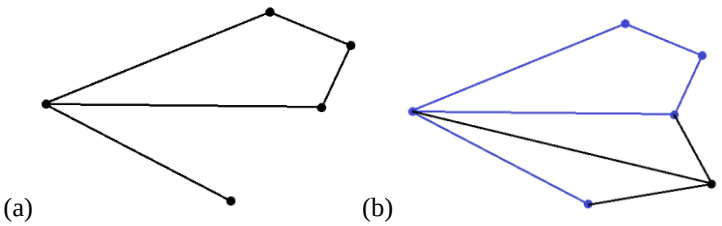


Figure G.2 Figure showing that graphs (a) and (b) are subgraph monomorphic and induced subgraph isomorphic. The smaller graph (a) can be mapped onto the larger graph (b). In (b) the blue edges and vertices represent an induced subgraph isomorphism of (a) onto the larger graph (which is also a subgraph monomorphism.

Stellenbosch University https://scholar.sun.ac.za

In Figure G.2 we see a subgraph monomorphic mapping between graphs in (a) and (b) (as seen by the blue edges and vertices). We see how the green edge in Figure G.3 (b) prevents [89] the induced subgraph isomorphism seen in Figure G.2 (b) from being present in Figure G.3 (b). There is still, however a subgraph monomorphic mapping between (a) and (b) in Figure G.3. In fact, if the green edge was added to (a) and not to (b) in Figure G.3, the results would remain the same: there would still be a subgraph monomorphism present (the red vertices and edges) but not an induced subgraph isomorphism.

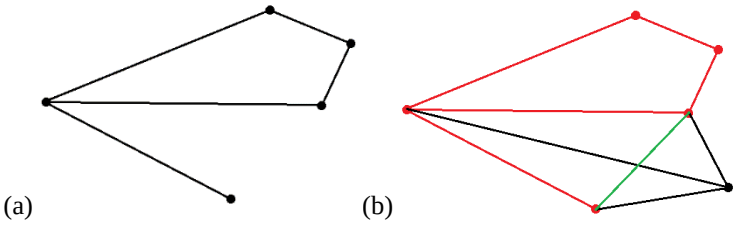


Figure G.3: Figure showing the difference between subgraph monomorphism and induced subgraph isomorphism. In (a) we are given the smaller of the two graphs. The larger graph (b) differs from that in Figure G.2 (b) since a green edge has been added. In (b) the red edges and vertices represent a subgraph monomorphism of (a) onto the larger graph. The additional green edge in (b) prevents there from being any induced subgraph isomorphism between the two graphs. If the green edge was added to (a) and not to (b) there would still be no induced subgraph isomorphism and there and the subgraph monomorphism would also remain the same (the red vertices and edges).

Both types of subgraph morphisms are useful to us in netlist comparison. The type of graph, among other factors, determines which type of morphism is more desirable. This will be discussed in more detail in Chapter 5 with the aid of examples.