

Simulation and Verification Software for Superconducting Electronic Circuits

by
Johannes Arnoldus Delport

*Dissertation presented for the degree of Doctor of Philosophy in Electrical and
Electronic Engineering in the Faculty of Engineering at Stellenbosch
University*



Supervisor:
Prof. Coenrad Johann Fourie
Electrical and Electronic Engineering
Stellenbosch University

April 2019

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

This dissertation includes four original papers published in peer-reviewed journals or books and zero unpublished publications. The development and writing of the papers (published and unpublished) were the principal responsibility of myself and, for each of the cases where this is not the case, a declaration is included in the dissertation indicating the nature and extent of the contributions of co-authors.

April 2019

Abstract

The dissertation presents simulation and verification software for both high- and low-level designs with emphasis on superconducting integrated circuits. A static timing analysis tool called SuperSTA is introduced as a method of analysing high-level superconducting designs in terms of the longest delay from any input to any output. This method of analysis provides metrics with which a design can be scrutinized and judged based on predefined criteria. A superconducting circuit simulator, JoSIM, is also introduced as a modified nodal analysis transient simulator which uses trapezoidal integration to solve a circuit netlist. A SPICE syntax circuit netlist is analysed and a matrix constructed, whereafter the $\mathbf{Ax} = \mathbf{b}$ linear algebra problem is solved using the KLU algorithm. JoSIM incorporates unique features such as a modified nodal phase analysis method, parametrization through expression parsing and alphanumeric node names. JoSIM includes a built-in graphical user interface for direct result verification. This is achieved using the cross-platform graphical library FLTK as well as Python's Matplotlib library. The design of the simulator and all the incorporated components are discussed in a step-by-step manner. JoSIM is compared to existing superconducting circuit simulators and judged in terms of accuracy and ability to simulate very large scale superconducting circuit designs. Results are investigated and recommendations for future improvements and optimizations to the simulator are discussed

Opsomming

Die dissertasie bied simulasi- en verifikasieprogrammatuur vir beide ho- en laevlakontwerpe, met die klem op supergeleidende gntegreerde stroombane. 'n Statiese tyd-analiese program genaamd SuperSTA word bekendgestel as 'n metode om hovlak supergeleidende ontwerpe te analiseer in terme van die langste vertraging van enige intree na enige uittree. Hierdie metode van analise verskaf maatstawwe waarteen 'n ontwerp ondersoek kan word en beoordeel word op grond van voorafbepaalde kriteria. 'n Supergeleidende stroombaan simuleerder, JoSIM, word ook bekendgestel as 'n aangepaste nodus tydsgebied analiese wat trapezoidal integrasie gebruik om 'n stroombaan op te los. 'n SPICE-sintaks stroombaan intree word geanaliseer en 'n matriks gebou, waarna die $\mathbf{Ax} = \mathbf{b}$ lineere algebra-probleem opgelos word met behulp van die KLU-algoritme. JoSIM bevat unieke eienskappe soos 'n aangepaste nodus fase analiese metode, parametrisering deur uitdrukking ontleding en alfanumeriese nodus name. JoSIM sluit 'n ingeboude grafiese gebruikerskoppelvlak in vir direkte resultaatverifikasie. Dit word behaal deur die grafiese biblioteek FLTK sowel as Python se Matplotlib-biblioteek te gebruik. Die ontwerp van die simuleerder en al die ingeslote komponente word stap-vir-stap bespreek. JoSIM word vergelyk met bestaande supergeleidende stroombaansimuleerders en beoordeel in terme van akkuraatheid en vermo om supergeleidende kringontwerpe te kan simuleer. Resultate word ondersoek en aanbevelings vir toekomstige verbeteringe en optimalisering aan die simuleerder word bespreek

Acknowledgements

I would like first and foremost thank my parents for all their years of support and motivation. They built the bridge on which I stand and for that I am forever grateful. Second, I would like to thank my soon to be loving wife for all the late night cups of coffee and words of support when I really needed. Third, I would like to extend my special gratitude to Dr Kyle Jackman and Mr Paul le Roux for their support and endless wisdom during times of mathematical strife and debugging nightmares. Last, I would like to thank Professor Coenrad Fourie for all his patience and wisdom when it was needed, without him all of this would not have been possible. My gratitude also extends to all of my research colleagues, past and present, who helped with testing and debugging of the software.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Background on Computing and Superconductivity	1
1.2.1	Digital Devices	1
1.2.2	Sensors and Filters	5
1.2.3	Simulation in computing	6
1.3	Objectives of Dissertation	7
1.3.1	Objectives	7
1.3.2	Document Layout	7
2	Electronic Design Automation in Superconductivity	9
2.1	Introduction	9
2.2	Design Process	9
2.2.1	High Level	9
2.2.2	Synthesis	10
2.2.3	Analogue Simulation	11
2.2.4	Optimization	12
2.2.5	Layout and Fabrication	12
2.2.6	Technology Computer Aided Design	13
2.3	Conclusion	13
3	High Level Verification	15
3.1	Introduction	15
3.2	SuperSTA	15
3.2.1	Design Flow	15
3.2.2	Pre-Placed	17
3.2.3	Post-Placed	17
3.3	Results	18
3.4	Conclusion	20
4	Analogue Simulation	21
4.1	Introduction	21
4.2	Josephson junction	21
4.3	JSIM	22
4.4	PSCAN	24
4.5	WRspice	25
4.6	Conclusion	27

5	JoSIM - Development	29
5.1	Introduction	29
5.2	Design Flow from Input to Output	30
5.2.1	Input	30
5.2.2	Matrix Setup	31
5.2.3	Solution Calculation	32
5.2.4	Output Handling	32
5.3	Components	33
5.3.1	Resistor	33
5.3.2	Inductor	34
5.3.3	Capacitor	34
5.3.4	Josephson Junction	35
5.3.5	Voltage and Current Sources	40
5.3.6	Lossless Transmission Line	44
5.3.7	Mutual Inductance	45
5.4	Control Commands	46
5.4.1	Parameters	46
5.5	Chicken and Egg	48
5.6	Phase Simulation	48
5.6.1	Phase Inductor	49
5.6.2	Phase Capacitor	49
5.6.3	Phase Resistor	50
5.6.4	Phase JJ	50
5.6.5	Phase Lossless Transmission Line	51
5.6.6	Phase Mutual Inductance	51
5.6.7	Phase Voltage Source	52
5.6.8	Phase Source	52
5.7	Conclusion	52
6	JoSIM - Results	53
6.1	Introduction	53
6.2	IV Curve	53
6.3	Small Simulations	54
6.4	Medium to Large Scale Simulations	57
6.5	Conclusion	60
7	Very Large Scale Design Simulation	62
7.1	Introduction	62
7.2	Data Structure Considerations	62
7.3	Parallel Processing	63
7.4	Optimizations in the Math Engine	64
7.5	Conclusion	64
8	Conclusion	66
	Bibliography	67
	Appendices	
	A	

B

C

D

E

List of Figures

1.1	CMOS logic using voltage levels	2
1.2	RSFQ pulse storage in superconducting loops	3
1.3	AQFP logic with a fluxon in the right loop	4
1.4	RQL design concept	4
1.5	A typical superconducting quantum interference device	5
1.6	An example of a superconducting quantum interference filter	6
2.1	Suggested standard design flow for superconducting circuit design	10
2.2	Schematic of a one bit full adder in single fanout design.	11
2.3	gSchem schematic editor by gEDA	11
2.4	Xic layout editing tool	13
3.1	Basic design flow of SuperSTA	16
3.2	4-bit KSA simulated as DUT at HDL level	19
3.3	4-bit KSA simulated with a clock speed higher than maximum identified	20
4.1	Basic example of a Josephson junction	22
4.2	IV curve of a typical JJ	22
4.3	V curves of the RCSJ model in JSIM	23
4.4	IV curve of the RCSJ model in JSIM without Rtype=0	23
4.5	Josephson junction IV curve created using phase based simulator PSCAN	24
4.6	Tunnel junction model IV curves created using PSCAN. Dirichlet coefficients used: (a) V. K. Semenov (b) MiTMoJCo	25
4.7	IV curves of the RCSJ model in WRspice. (a) With Rtype=0 (b) Without Rtype=0	26
4.8	Rtype=1 model comparison between WRspice and JSIM	27
4.9	Circuit used to find the IV curve of a JJ	27
5.1	Overview of the JoSIM design flow	30
5.2	Example of a standard netlist with a subcircuit	30
5.3	Resulting master netlist after substitution	31
5.4	Approximating the y_{n+1} value	31
5.5	A basic resistor element.	33
5.6	A basic inductor element	34
5.7	A basic capacitor element	34
5.8	A basic Josephson junction element	35
5.9	The resistively and capacitively shunted equivalent junction model	36
5.10	Element model of the Werthamer approximation	40
5.11	Sum and difference current calculation using the junction voltage	40
5.12	Sum and difference admittance equivalent circuits	40
5.13	A basic voltage source	40
5.14	A basic current source	41
5.15	Example of a PWL function depicting the relevant values	41

5.16	Example of a pulse function depicting the relevant values	42
5.17	Example of a sinusoidal function depicting the relevant values	43
5.18	Example of a custom function depicting a waveform line of [0 1 2 3 2 1 0] with periodicity enabled to create a triangle wave.	43
5.19	A basic transmission line element	44
5.20	A basic mutual inductance element	45
6.1	JoSIM IV curves	53
6.2	JoSIM IV curve of only rtype=1	54
6.3	IV curves of JSIM, JoSIM and WRspice	54
6.4	Basic JTL used to test small simulations	55
6.5	Results of the JTL simulation performed with JoSIM	56
6.6	Comparison of the results for the JTL between JSIM, JoSIM and WRspice . . .	56
6.7	Error percentage of JoSIM compared to JSIM and WRspice	57
6.8	Results of the 7th partial product	58
6.9	Output of the 4-bit KSA simulation	59
6.10	Execution time vs JJ count	61

List of Tables

3.1	Gate-delay timings for a RSFQ cell library	18
5.1	Shunting yard conversion to a RPN stack	47
5.2	RPN stack evaluation	47
6.1	General binary partial product generation	58
6.2	4-bit binary addition	59
6.3	Simple example of a large circuit	60
6.4	Summary of execution times for various simulation sizes	61

Chapter 1

Introduction

1.1 Motivation

With Moore's Law[1] approaching its inevitable limit, we look towards alternative technologies to replace complementary metal-oxide-semiconductors (CMOS) in the high-speed computing domain. This end is being approached from both an energy efficiency and a physical limitations point. As devices become smaller, a need to reduce the amount of components and materials used arises. When these components and materials are reduced or interchanged for others, trade-offs are made to the energy efficiency. This battle can only go on until one reaches a hard limit. We therefore aim our sights towards an entirely different technology.

One such technology is to use superconductors to produce a highly energy efficient form of computing that would far exceed the current limitations seen by CMOS[2]. The resistive property of a material is proportional to the temperature and physical area of the material. Resistance accounts for the majority of energy loss in electrical components. To therefore minimize the resistance of a material it would be crucial to reduce the temperature. When the temperature has been sufficiently reduced, the resistance becomes almost non-existent and the material enters a superconducting state[3]. Current that flows in a superconducting material can do so indefinitely as there is no resistance for it to lose energy through.

When two superconducting materials are brought very close to each other, a device called a Josephson junction (JJ) can be created wherein the current in one superconductor can be passed to another if the current exceeds a critical value[4]. We can equate this to the switching of a transistor in conventional CMOS. These JJs can be used to build logic devices, which have been shown to reach clock frequencies close to the terahertz range[5].

Unlike the well established design software found in the CMOS domain, the superconducting circuit (SC) domain lacks the necessary tools to design and fabricate complex computing designs. The development of SC simulation software is what motivates this work.

1.2 Background on Computing and Superconductivity

1.2.1 Digital Devices

Superconducting digital devices can be created to operate using numerous technologies such as rapid single flux quanta (RSFQ), adiabatic quantum flux parametron (AQFP) and reciprocal quantum logic (RQL). Each technology is unique in its own way and vastly different to CMOS.

1.2.1.1 CMOS

CMOS is a technology that was developed by Fairchild Semiconductor International, Inc. in the early 1960s. The complementary part of the name refers to the p-type and n-type metal-oxide-semiconductor field-effect-transistors (MOSFET) that almost all CMOS logic consists of. CMOS forms the basis of modern day computing.

In CMOS a logical 1 is defined as a state where the voltage measured at a point to ground is within the threshold of some value, usually 5V, and the logic 0 is defined when the voltage is close or equal to 0V. Any state between these threshold values is undefined and causes undefined behaviour. The rise and fall of the voltage between levels does not occur instantly and therefore creates a delay in the system which needs to be compensated for. This concept is illustrated in Figure 1.1.

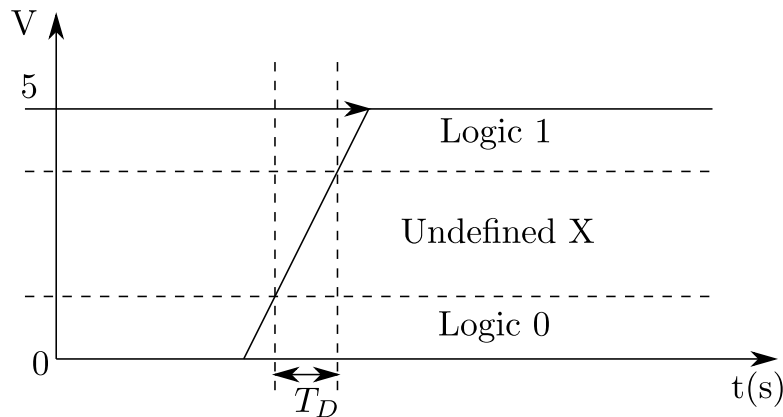


Figure 1.1: CMOS logic using voltage levels

1.2.1.2 RSFQ

RSFQ is a technology that was developed by Likharev's research group at Moscow State University circa 1985[6]. RSFQ circuits are driven by single flux quantum (SFQ) voltage pulses which are defined by an integration of voltage over time where the result is exactly 1 magnetic flux quantum (Φ_0) or rather a fluxon (1.1). These fluxons are stored in superconducting loops consisting of inductors and JJs.

$$\int V(t)dt = \Phi_0 \equiv \frac{\hbar}{2e} \approx 2.07\text{mV} \times \text{ps} \quad (1.1)$$

Each superconductive loop can store exactly an integer number of fluxons, and when configured with the correct amount of input and bias current, passes a fluxon on to the next superconductive loop. When a fluxon is present in a loop we call it a logic 1 and when a fluxon is absent a logic 0. Using this convention it is possible to build a pulse based family of logic gates.

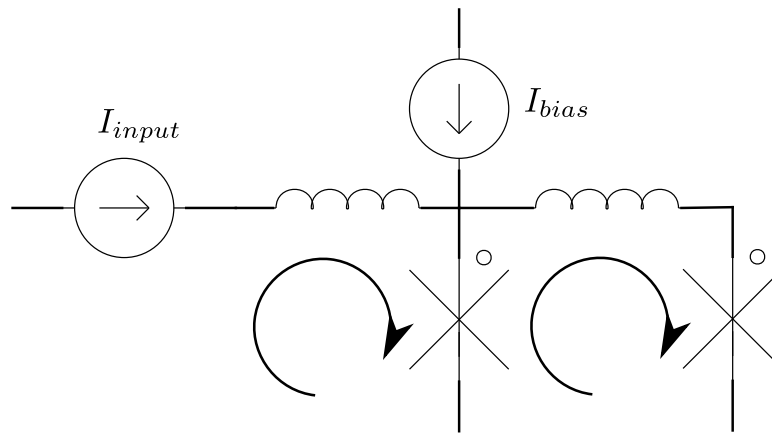


Figure 1.2: RSFQ pulse storage in superconducting loops

When an SFQ pulse arrives at the input in the circuit shown in Figure 1.2, it is stored in the initial loop until the bias current is greater than the critical current of the junction, causing a switch to occur and the pulse to be moved to the second loop.

RSFQ technology has shown the capability to be used for very-large-scale integration (VLSI) through a complete 8 bit microprocessor called CORE1. CORE1 was developed by Yokohama National University in collaboration with Nagoya University[7]. This design has been optimized quite a few times since the first release and is currently at the 4th iteration[8].

RSFQ is DC biased and therefore has static power dissipation which makes it less suitable for VLSI when compared to derivations like ERSFQ or ESFQ[9] and other technologies such as AQFP and RQL.

1.2.1.3 AQFP

AQFP was developed by Professor Yoshikawa's research laboratory at Yokohama National University circa 2013[10]. Similar to RSFQ fluxons are stored in superconducting loops which consist of inductors and JJs. These circuits are AC biased which results in extremely low power dissipation.

The most basic AQFP gate consists of 2 superconducting loops called a quantum flux parametron (QFP). The bias line, which consists of inductors, is mutually coupled to this QFP. When the bias is increased in the presence of a small input current, the fluxon moves between the two loops. The QFP represents a logic 0 when the fluxon is in the left loop and a 1 when it is in the right loop. We illustrate this concept in Figure 1.3.

Due to its AC nature, AQFP has almost zero power dissipation and can thus be scaled almost indefinitely for VLSI designs. This AC bias does however have the drawback of not being as fast as DC based RSFQ due to the limit of the microwave frequency generated using room-temperature electronics.

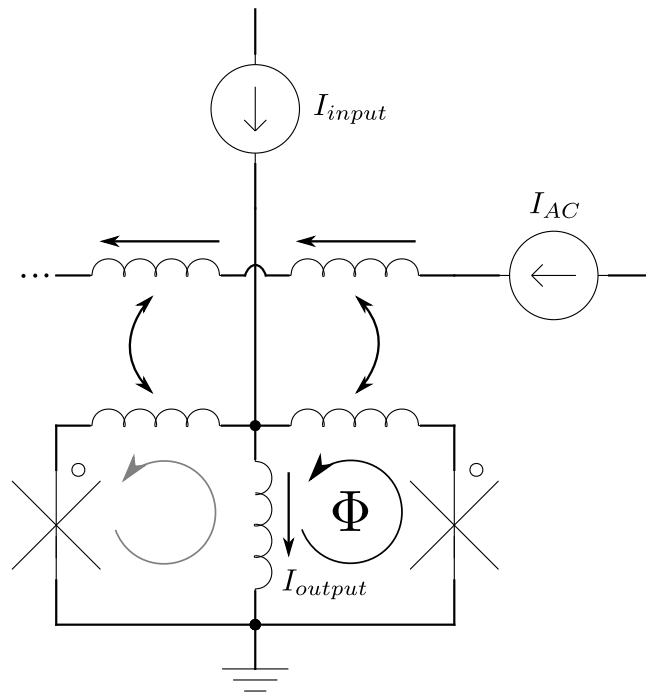


Figure 1.3: AQFP logic with a fluxon in the right loop

1.2.1.4 RQL

RQL is a low power superconductor logic family developed by Northrop Grumman Systems Corp. circa 2011[11]. Like AQFP it is an AC biased logic family, however the logic is a lot more similar to RSFQ. RQL was patented by Northrop Grumman and therefore very little external research is conducted on the applications of this technology.

In RQL the clock is generated using an AC signal which also acts as the bias for the JJs through the mutually coupled inductors. Similar to RSFQ a fluxon is stored in a superconducting loop and is transferred to the next loop when the bias exceeds the junction critical current. The basic idea behind RQL is demonstrated in Figure 1.4. This technology, like AQFP, is speed-limited to the maximum speed of the microwave frequency generating circuitry and due to the AC bias retains similar low power dissipation qualities.

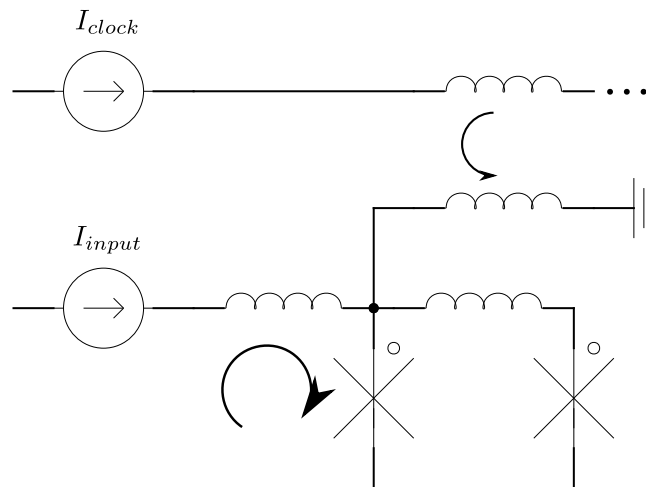


Figure 1.4: RQL design concept

1.2.2 Sensors and Filters

Superconductivity in digital electronics require very low temperatures, often close to absolute zero. At this temperature these devices become very susceptible to external interference such as external magnetic fields. This can be advantageous in some cases such as with superconducting quantum interference devices (SQUID) and superconducting quantum interference filters (SQIF).

1.2.2.1 SQUIDs

A SQUID is a device that consists of 2 JJs to form the superconductive loop depicted in Figure 1.5. SQUIDs are extremely sensitive to variations in current due to external magnetic fields and therefore are useful in systems where measurement accuracy is of utmost importance.

SQUIDs are mainly used in medical fields to measure neural activity as well as magnetic resonance imaging (MRI). SQUIDs can also be used in mining exploration to measure geothermal energy. These devices form the basis for some quantum computing applications[12].

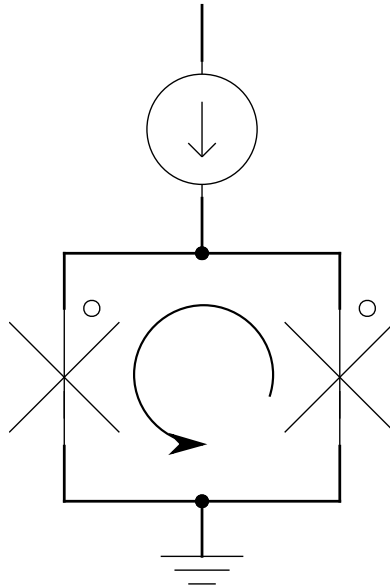


Figure 1.5: A typical superconducting quantum interference device

1.2.2.2 SQIFS

SQIFs are a relatively new application of the JJ where it is used in filter applications. Basic designs for SQIFs include multiple SQUID loops attached in an array with varying sizes[13]. We demonstrate a basic example of a SQIF array in Figure 1.6.

SQIFs work particularly well when used in sensitive magnetometry, due to its non-periodic voltage response and large negative spike at zero magnetic field. The use of SQIFs in the RF domain has also been shown in recent studies[14].

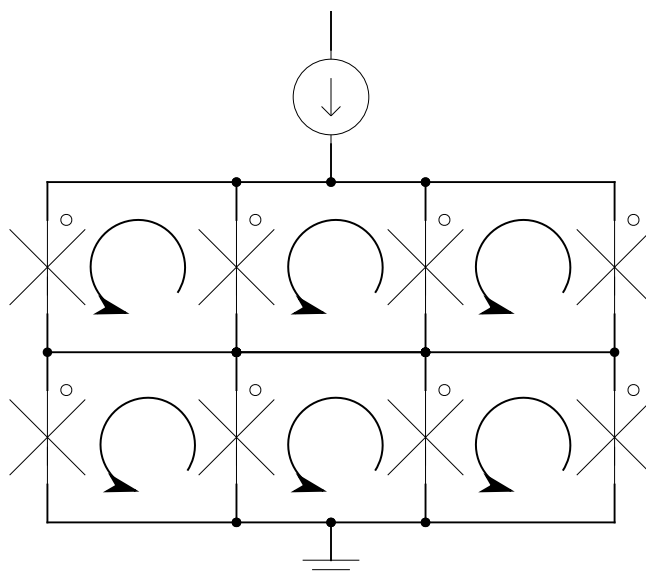


Figure 1.6: An example of a superconducting quantum interference filter

1.2.3 Simulation in computing

Any type of computing technology requires some form of computer aided simulation software that would enable the user to verify the device operation before physically fabricating the device. Smaller designs such as simplistic filters can be mathematically verified, however in complex electronic systems this can no longer be done by hand.

The purpose of any computer aided circuit simulation software is simply to solve a set of simultaneous linear equations which can be grouped as

$$\mathbf{Ax} = \mathbf{b} \quad (1.2)$$

where \mathbf{A} is a $m \times n$ matrix of coefficients, \mathbf{x} a vector of unknown variables and \mathbf{b} a vector of known values.

In the early 1970s, a research group at the University of California Berkeley created a simulation program with integrated circuit emphasis (SPICE)[15]. What SPICE allowed them to do was solve an electronic circuit of theoretically any complexity by applying nodal analysis, essentially Kirchoff's current law (KCL), to the design creating a matrix of conductances that is used to solve the voltage at every node within the design.

Since its conception SPICE has seen many variants[16][17], with the latest being SPICE3F whereafter it became open domain and the base for all modern day commercial and non-commercial circuit simulators.

With the advancement of computing systems, new methods of simulation were also developed. These simulation methods are based on a higher level of abstraction where it is no longer necessary to see what components are physically connected in the device but rather what the behavioural function of the device would be given a specific input. This method of simulation is called high level verification and can be done using hardware description language (HDL)[18][19]. This method of circuit design spawned two programming languages, very high speed integrated circuit (VHSIC) hardware description language (VHDL)[20] and Verilog HDL[21].

HDL allows the user to describe the function of a circuit on a higher level and verify operation through logic simulation. This is a lot faster than SPICE, however, it only verifies operation and does not account for any electrical properties in the design.

1.3 Objectives of Dissertation

1.3.1 Objectives

In light of the limited availability of advanced SC simulation software for both analogue and high level verification, we propose a new SPICE type simulation engine JoSIM and a static timing analysis tool (SuperSTA) for high level timing verification.

JoSIM is specifically aimed to overcome the challenges faced when attempting to simulate designs that incorporate superconducting elements such as the JJ. SuperSTA uses results obtained from analogue simulations to apply timing verification to SC designs in both analogue form as well as high level description.

The simulation engine and all algorithms mentioned in this work are written in C++ and compiled to be executable on three major operating systems Microsoft Windows, Apple Mac OS and Linux.

We now summarise the required objectives to ensure the success of this dissertation:

1. Investigation of the available electronic design automation software within the superconducting circuit design domain.
2. Creation of high level verification tool for static timing analysis.
3. Creation of a SPICE simulator that is capable of interpreting superconducting elements.
4. Implementation of faster numerical solvers to improve simulation time and solution accuracy.
5. Attempt implementation of non-conventional superconducting elements such as the microscopic tunnel junction.
6. Investigate and derive phase based simulation techniques for more accurate superconducting simulation results.
7. Simulation of very large scale circuits to allow verification of microprocessor level designs within reasonable time.

1.3.2 Document Layout

In Chapter 2 we discuss the state of electronic design automation (EDA) in the superconducting circuit domain and provide some insight into the steps involved when designing and creating a superconducting integrated circuit. Each step is discussed in detail with examples and what the equivalent would be within the existing CMOS EDA design flow. We take note of the large efforts made by research foundations to attempt to get the design flow for superconductivity to a level where it can compete with CMOS. We make some recommendations on fields that require investigation to help bolster the strength of EDA in superconductivity

High level verification and the effect of a higher level of abstraction is discussed in Chapter 3. We explain the need for timing analysis at this level and present the static timing analysis tool SuperSTA. The algorithms and design process for SuperSTA are discussed and results are presented. Recommendations are also made for future improvements to the tool.

In Chapter 4 we investigate the history of simulation engines for superconducting elements. Development decisions, results and features are discussed for each one and drawbacks are highlighted. Proposed improvements are mentioned and conclusions are made.

JoSIM development is discussed, design decisions are explained and supported with arguments in Chapter 5. Algorithms used for file input and output (IO) are walked through, comparing data structure methods for improved efficiency. Supported components within JoSIM

are shown with their corresponding input file descriptions as well as the generalized matrix entries that collate to form the conductance matrix. Methods for solving very sparse matrices in the most efficient way are investigated and the findings implemented.

The results of various different designs, each with unique elements are displayed in Chapter 6. These results are critically analysed and compared to previous simulation engines in both speed and accuracy. We further analyse features not present in other simulators and scrutinize the accuracy of the results.

In Chapter 7, we investigate improvements to the designs implemented in JoSIM to handle massively parallel processing to reduce simulation times. Data structures used are reconsidered to reduce the complexity of the algorithms used. Optimizations to the mathematical engines used are investigated. We experiment with the idea of very large scale circuits with up to one million JJs to evaluate the efficacy of analogue simulators on microprocessor scale design.

Conclusions and recommendations as well as ideas for future research are mentioned in the final chapter.

Appendix A contains a published article “*Superconducting digital circuit design with an open source and freeware tool Chain*”[22] wherein we investigate the entire design flow for superconducting circuit design using only non-commercially available tools.

Appendix B contains a published article “*Analysis of a Shielding Approach for Magnetic Field Tolerant SFQ Circuits*”[23]. In this article we utilize superconducting simulation engines to investigate the effect of external magnetic fields on SFQ circuits and explore ways for improvement to make designs more tolerant to magnetic fields.

Appendix C contains a published article “*A Static Timing Analysis Tool for RSFQ and ERSFQ Superconducting Digital Circuit Applications*”[24], wherein we announce SuperSTA and discuss its features and capabilities as well as shortcomings.

Appendix D contains an published article “*JoSIM – Superconductor SPICE Simulator*”[25], wherein JoSIM is presented and evaluated against other simulation engines.

Appendix E contains the user manual for JoSIM, which guides the user through the process required to run the simulation engine while discussing the features and input file requirements.

Chapter 2

Electronic Design Automation in Superconductivity

2.1 Introduction

This chapter will discuss the process that a designer would follow when designing a superconducting circuit. This process is called a design flow and is not necessarily the exact process that always needs to be followed, but rather the one suggested by the developers of the software used in the process[22].

We discuss each step of the process as well as provide general examples of what the input/output would look like. The software used is also mentioned where possible and possibilities for future research in software that does not exist yet are mentioned. A more comprehensive roadmap of the available state of EDA in SC design can be seen in [26] and [27].

In the CMOS domain, these design flows are well-established and large EDA companies such as Cadence and Synopsis have been founded to enhance these design flows as far as possible. Efforts in the superconductivity domain have been made through large government projects[28] in the attempt to kick start EDA companies specifically for this domain.

We depict a suggested standard design flow for superconducting circuit design in Figure 2.1.

2.2 Design Process

2.2.1 High Level

A standard process to design a electronic circuit would start at a very high level of abstraction. This means that nothing is known about the physical properties of the circuit, only its function. This functionality would then be described in an HDL language of choice. An example of some behavioural description in Verilog HDL can be

```

module add(a, b, out, carry_in, carry_out)
  input a, b, carry_in;
  output out, carry_out;

  assign carry_out = (a & b) | (a & carry_in) | (b & carry_in);
  assign c = a ^ b ^ carry_in;
endmodule

```

where \wedge , $|$ and $\&$ are the logic operators XOR, OR and AND respectively. The example describes a single bit full adder operation in Verilog.

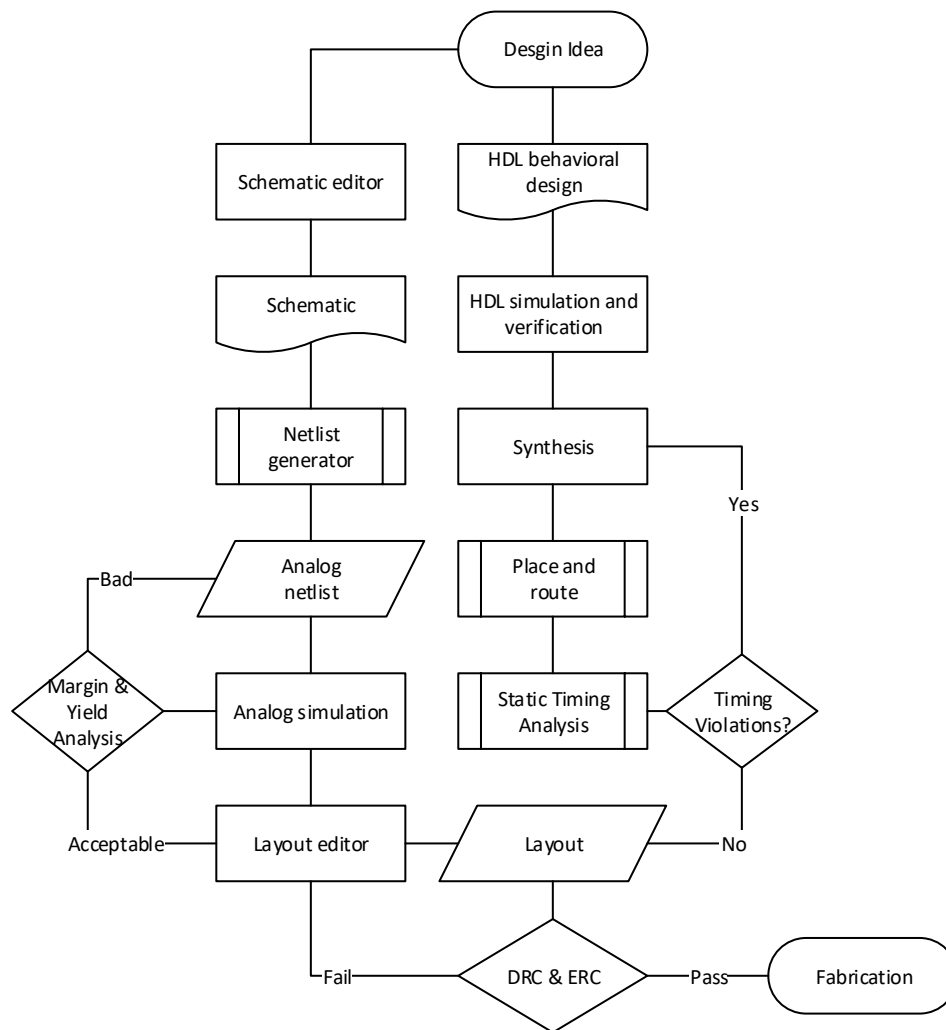


Figure 2.1: Suggested standard design flow for superconducting circuit design

Once the circuit behaviour has been described, it would be placed in an HDL testbench which will allow it to be simulated at a high level. Simulations can be done with commercial level tools or open source simulators such as Icarus Verilog[29]. Once simulated, the operation can be verified. This verification includes visual verification through interpretation of the results as well as static timing analysis to ensure there are no timing violations[24].

If the designer is satisfied with the device operation, the design can be synthesized down to an analogue netlist that can be simulated using an analogue simulation engine.

2.2.2 Synthesis

Synthesis is described as a process of combining a series of components or elements to form a single entity. In EDA, synthesis is used to describe the process of transition from a high level design to lower level. There are a few variants of synthesis and the definition of the term seems to change depending on who is asked.

In SC design the two main variants of synthesis involve either generating a SPICE circuit netlist or a fully placed and routed layout from a single HDL design. Both of these processes would require the designer to specify the type of technology the design is intended for as well as provide the synthesis engine with a library of analogue logic gates that will be used to formulate

the output.

There are however no known synthesis tools that support superconducting digital circuits fully due to their inherent synchronous design. There have been multiple efforts to alter existing tools or to create new ones[30][31], however none have been truly successful and thus the designer would have to create the analogue netlist either by hand or by creating a schematic in some schematic editor.

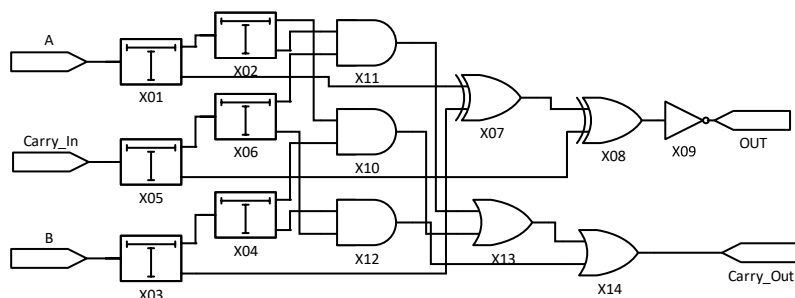


Figure 2.2: Schematic of a one bit full adder in single fanout design.

Schematic editors can be quite useful in providing visual feedback in terms of how everything is connected. Most schematic editors provide the user with the ability to convert the design into a analogue netlist which can then be used for simulation[32][33]. An example of a schematic editor can be seen in Figure 2.3.

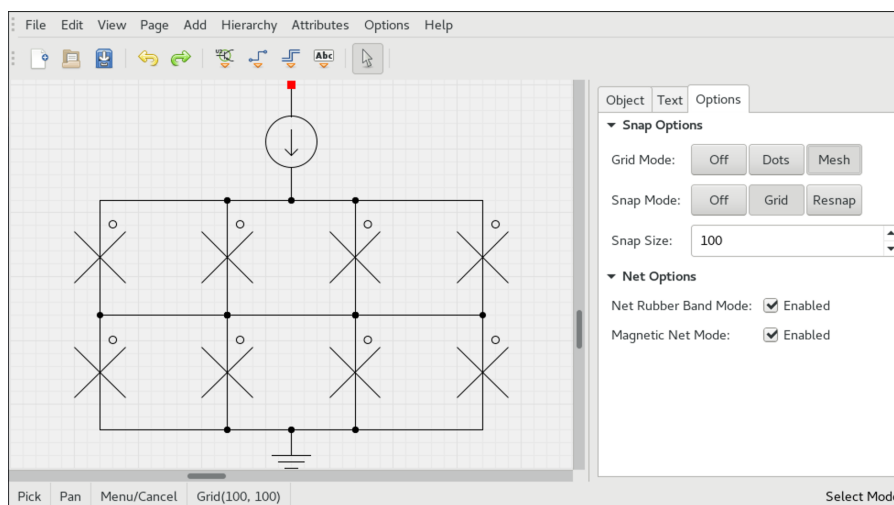


Figure 2.3: gSchem schematic editor by gEDA

2.2.3 Analogue Simulation

A technology independent description of the example adder in Figure 2.2 through SPICE notation can be presented as

```
.subckt ADDER 1 2 3 4 5
X01 SPLIT 1 6 7
X02 SPLIT 6 8 9
X03 SPLIT 2 10 11
X04 SPLIT 10 12 13
```

```

X05 SPLIT 4 14 15
X06 SPLIT 14 16 17
X07 XOR 7 11 22
X08 XOR 15 22 23
X09 NOT 23 3
X10 AND 8 12 18
X11 AND 9 16 19
X12 AND 13 17 20
X13 OR 18 19 21
X14 OR 21 20 5
.end ADDER

```

In SC design using SFQ pulse propagation, the maximum fanout that a single gate can have is one. To provide the same pulse to more than one gate a splitter cell is required, which essentially duplicates the SFQ pulse. This netlist coupled with the corresponding sub-circuits required can then once again be simulated by inserting it into a testbench with excitations at the relevant ports.

If the designer is satisfied with the results that the analogue simulation produces they can then proceed to creating a layout of the design. If however the designer is not satisfied, they then have the option to optimize the design or start from scratch. The design might produce the correct results but operation margins might make it less likely to succeed due to fabrication tolerances. Since the design operates as expected behaviourally the advised option would be to perform some form of optimization to attempt to increase operational margins and yield percentage.

2.2.4 Optimization

The process of optimization includes doing a margin analysis, which varies every component within the netlist over a specified range. Multiple simulations are done while varying these components to establish the operational margins of the design. This allows the designer to identify components that might cause the design to fail once fabricated if the fabrication tolerances are not tight enough.

Optimization of these identified components can be done by manually tweaking the values of the components to try and improve stability, however this can be quite a tedious and time consuming process if the design is quite large. Optimization can also be done by implementing various methods such as genetic algorithms[34], Monte Carlo analysis[35] and others.

There are entire research fields dedicated to establishing algorithms that provide the best form of optimization. It might be worthwhile to investigate these algorithms in an attempt to create a generalised optimization software that applies the best algorithm of optimization relevant to the design under scrutiny.

2.2.5 Layout and Fabrication

Once the designer is satisfied with the results of the analogue simulation, they can then proceed with the creation of the layout. A layout is the physical description of the device that will be used for fabrication. This includes the type of materials used for each layer, the width of the material, the electrical properties and the physical location of each part of the design as it would appear on a chip.

Since layouts rely strictly on the layer information, they can be done in almost any layout editor that accepts custom layers. Common layout editing software includes LayoutEditor, LASI, KLayout and Xic seen in Figure 2.4.

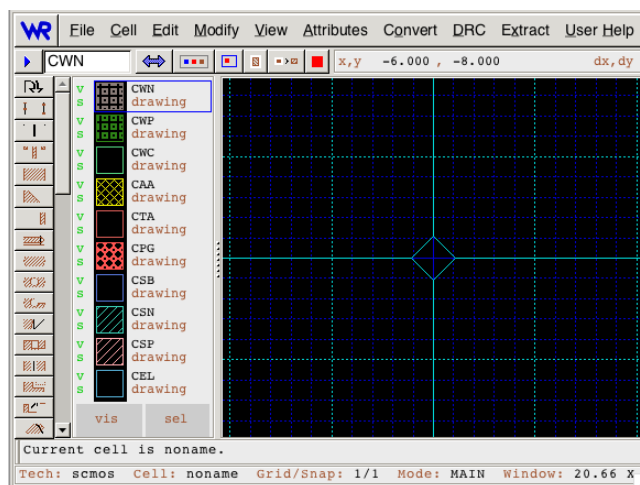


Figure 2.4: Xic layout editing tool

The process of laying out a design is not automated and requires that the designer have intricate knowledge on layout design. The process is very time consuming and tedious especially with large designs, though many layout editors have the ability to create sub-cells that aid the designer in the process. Software tools such as layout versus schematic (LVS) are under development to allow the user to verify the layout being created against the schematic or netlist source. LVS requires intense graph manipulating algorithms and is a very new field even in the CMOS domain.

The layout that is produced must adhere to the specific design rule checks (DRC) and electrical rule checks (ERC) of the specific technology the design was intended for. Once successful, tools such as Inductex[36] can be used to extract the impedances and repopulate the analogue netlist to ensure that the design still produces the expected results given the materials used and physical dimensions.

If the designer is completely satisfied with the results the layout level analogue simulation produces the layout can then be sent for fabrication at the very few superconducting digital circuit foundries that exist.

2.2.6 Technology Computer Aided Design

Technology computer aided design (TCAD) is a process whereby the fabrication process is simulated through equations that mimic the geometric extrusions[37]. This simulation method is completely separated from the design flow and does not have an established software tool for SC design as of yet. The equations that mimic the geometry extrusions are extremely complex and require a large effort from both engineering and physics fields.

2.3 Conclusion

Though far from complete, the design flow for superconducting electronics is reaching a point where VLSI circuits can be designed and simulated. Operation can be verified through the verification techniques discussed. These verified designs can be fabricated and tested physically, which will possibly and most probably introduce other unforeseen parts of the design flow that have not been looked at prior to the fabrication of VLSI circuits.

Software tools that spawn from projects like SuperTools[28] and similar will hopefully provide a good foundation to build forth from in the development of better methods for superconducting specific EDA tools. Once these tools reach a certain stability, companies may be founded from

them or simply acquired by larger, well-established companies such as Cadence or Synopsis. Such is the life cycle of many innovative software technologies.

Chapter 3

High Level Verification

3.1 Introduction

High level verification is a process that attempts to establish valid design operation at a higher level of abstraction. As the number of gates in a design increase, the time required to perform an analogue simulation also increases and as a result so does the time to verify successful operation. This time grows exponentially and soon verification through analogue simulation of VLSI designs no longer become viable. To reduce the necessity of performing multiple simulations, static timing analysis attempts to gauge the maximum performance of a design by evaluating all possible paths from input to output and identifying the critical path given a set of gate-delay times.

This chapter will discuss the design and implementation of a high level verification tool SuperSTA[24]. SuperSTA is a command-line interface, static timing analysis tool for SC applications used to identify critical paths through a design, maximum frequency of operation and possible slack in the design. Two timing values are presented to the user, one being a global clock which is the maximum clock at which the design when viewed as a black box would produce an output at every clock edge. The second clock value presented is the system clock, which is the maximum clock at which the design can receive inputs at when performing wave-pipelining[38].

SuperSTA presents the user with a few additional device metrics when the user provides a target time within which the design is intended to finish. These metrics include the slack within the design as well as mean path time and path time variance.

This tool was developed under the IARPA seedling project[39] to work in conjunction with TimeEx[40] and RSFQ mapper[41]. The aim of this project was to establish the possibility of high level design tools for RSFQ and ERSFQ circuits, and is the precursor to Supertools[28].

A macro view of the design flow implemented in SuperSTA is depicted in Figure 3.1.

3.2 SuperSTA

3.2.1 Design Flow

The static timing analysis tool SuperSTA takes as input either a SPICE netlist which is produced as an intermediate step within the synthesis process discussed in Section 2.2.2 or a device exchange format (DEF)[42] file which is the end result of the place and route synthesis process. The SPICE netlist input is referred to as the pre-placed STA and is discussed in detail in Section 3.2.2 where the DEF input is called the post-placed STA and discussed in Section 3.2.3.

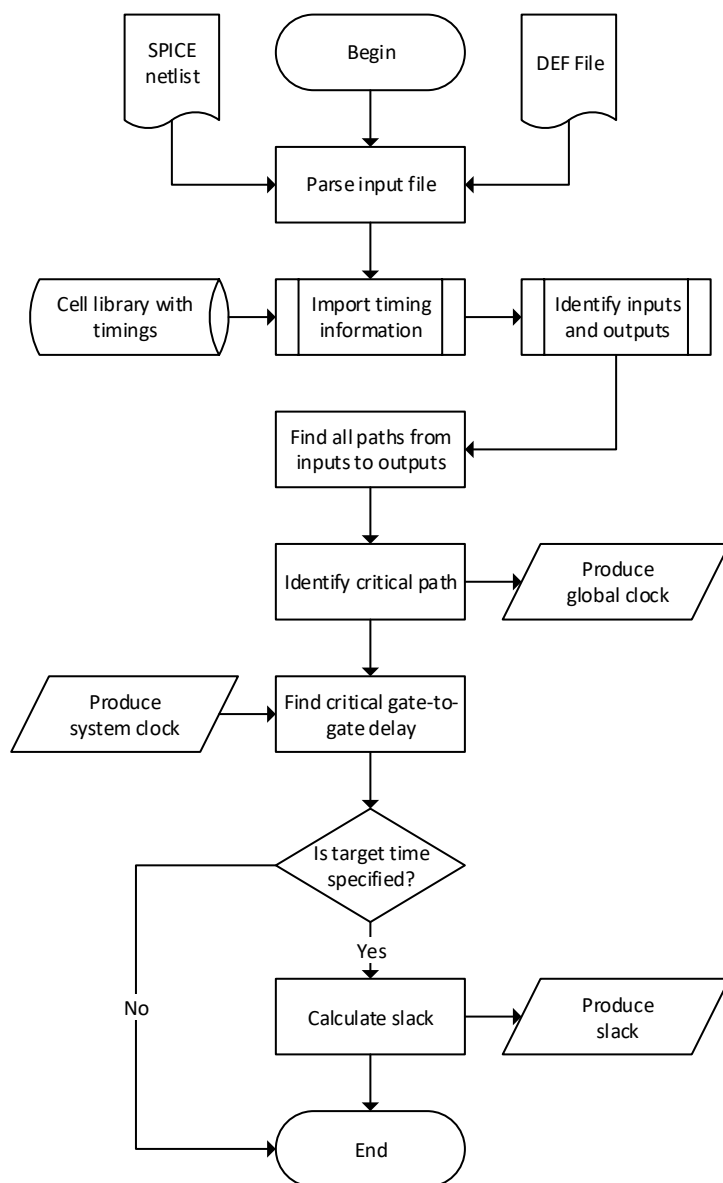


Figure 3.1: Basic design flow of SuperSTA

Either file type provided requires specification of the input and output nodes or locations, which, when not provided, will be determined algorithmically. SuperSTA parses each file sequentially to construct a graph of how the design is interconnected. Once all interconnects are established, gates that are not connected at both input and output are flagged as possible IO in the case where the IO is not specified. This method of IO detection is not always accurate as there is no way of determining if the identified point is an input or output. It is therefore much more advantageous for the user to specify these points.

This constructed graph is then traversed from every possible input to every possible output using a variation of the breadth-first search (BFS) method[43]. This method traverses the graph by choosing a single point of input and output finding the most direct path between them. Due to the single fanout nature of SFQ design the use of splitters is required, which in this case causes branching to occur whenever a splitter is found whilst traversing. A note is simply made of the location of every splitter found along the path and once the path reaches an output it returns to each splitter to find a new path to an output.

There are alternate methods of graph traversing such as depth-first search (DFS) [43], which explores all paths to their output first before traversing any branches. This method would essentially produce the same result, however BFS was ultimately chosen instead.

This method of finding all possible paths is however only possible if the design being analysed contains no feedback. If feedback is present, the search algorithm will loop infinitely. There are methods for circumventing this problem such as checking for repetition in the path being analysed.

Once all possible paths from input to output have been found these paths are analysed using a provided cell library containing gate-delay times to find the path that has the longest delay. This path is called the critical path. The cell library of gate-delays is merely a folder that contains HDL descriptions of each gate with accurate timing delays as characterized using the timing extraction tool TimeEx[40]. If this cell library is not provided, generic times will be used and results will not be accurate.

This critical path can be used to identify the maximum clock frequency at which the design would produce a result in every clock cycle. This clock frequency is called the global clock and is useful in the case where the design will be used as a subcircuit in a larger design and requires a single clock input. In addition to this, each path is inspected to find the largest inter-gate delay to identify the system clock, which is the maximum clock frequency at which the design can throughput data without error. This clock is useful in cases where the design would be used through wave-pipelining[38] of the inputs.

SuperSTA is also capable of determining whether the design meets certain requirements such as operating frequency. This can be accomplished by allowing the user to specify a time within which a result is expected at the output when given an input. This timing parameter is called the target time and is used to calculate slack in the design. Slack in design philosophy is a performance metric, where positive slack indicates a good design and negative slack a design that needs reconsideration. The user is also presented with other meaningful design statistics such as the mean path time and path time variance which should also be taken into consideration when negative slack is present.

3.2.2 Pre-Placed

During the process of synthesis whereby a high level design is transformed into a low level description, a SPICE-like netlist can be generated as an intermediate step. This netlist only contains information regarding the interconnection between gates as well as the types of gates used. It cannot be directly simulated and would require some form of testbench with relevant subcircuit definitions. This netlist also does not contain any wiring information, which would affect the delay within the design.

The purpose of allowing SuperSTA to evaluate this intermediate netlist is to allow the user to gauge relatively early in the design process whether the design would meet target time requirements or not. If the design does not meet the requirement in the netlist stage, then it will by no means improve through the addition of interconnect delays. Similarly, if the design just barely meets the set requirement, then it would generally not after adding interconnect delays.

The results produced for this method should not be used to accurately characterize the maximum operation speed of the design and should merely be used as an indication.

3.2.3 Post-Placed

The final result of the synthesis process would generally contain information regarding the precise coordinates of placement and lengths of the interconnect wires. Additionally, due to

the use of PTLs in SFQ circuit design (discussed in Section 5.3.6), the requirement to include the delay caused by the interconnection between different layers called vias also need to be considered.

This file is called the post-placed design and is presented in a DEF file format. This file is parsed to determine the interconnection between gates as well as the delays caused by these interconnections. To calculate the interconnect delays some information is required about the material the design will be fabricated with. These values are the data transmission speed and the via delay. The user has the option to specify these values as part of the command used to execute SuperSTA. If not specified, the default values will be used.

3.3 Results

The results of static timing analysis performed on a 4-bit Kogge-Stone adder (KSA)[44] is presented. The file analysed is of the DEF type (post-placed) and the cell library of gate-delays is presented in Table 3.1.

Table 3.1: Gate-delay timings for a RSFQ cell library

Gate	Out delay
AND	9.3ps
DFF	5.5ps
JTL	3.2ps
NDRO	7.0ps
NOT	9.2ps
OR	7.2ps
PTL	18.5ps
SPLITTER	5.0ps
XOR	5.7ps

Due to the extreme file length of the 4-bit KSA def we ultimately decided not to include it in this dissertation. SuperSTA was executed using the following command

```
SuperSTA -t 300E-12 -ppv 0.02 -lpp 110 -d 4bitksa_route.def
```

where *ppv* and *lpp* are the via delay and data transmission delay per micron. This command produces the following result

```
SuperSTA v2.1.1(Jan 3 2018). Copyright 2016 Johannes Delport.
```

```
This program comes with ABSOLUTELY NO WARRANTY.
```

```
This is free software, and you are welcome to redistribute it
under certain conditions; see the GNU GPL v3 for details.
```

```
File specified to analyse: "4bitksa_route.def"
```

```
Total possible paths in design: 61
```

```
Critical path time: 2.92225e-10s or rather 292.225ps
```

```
Global clock: 3.42202GHz
```

```
Critical path:
```

```
b1_Pad
```

```

Split_99_n222
xor2a_17_n17
Split_76_n199
Split_77_n200
and2_26_n26
Split_84_n207
and2_27_n27
or2_28_n28
xor2a_29_n29
DFF_67_sum2
sum2_Pad

```

Slack: $7.77455e-12s$

Total slack: $3e-10s$

Mean path time: $2.58092e-10s$ or rather 258.092ps

Path time variance: $4.77655e-22s^2$ or rather $4.77655e-10ps^2$

Path time standard deviation: $2.18553e-11s$ or rather 21.8553ps

Longest NET: and2_34_n34 -> and2_35_n35 => 570 micron

Largest inter-gate delay time: 50.7ps

System clock: 19.7239GHz

From this results we can see that the maximum clock that can be achieved internal to the design (system clock) is 19.7239GHz and note that the critical path is of depth 6 clocked gates. This means that when clocked at this frequency we expect the result to appear at the output after 6 clock cycles. When this clock is applied to the HDL test bench where the 4-bit KSA is the device under test (DUT) we observe this exact behaviour.

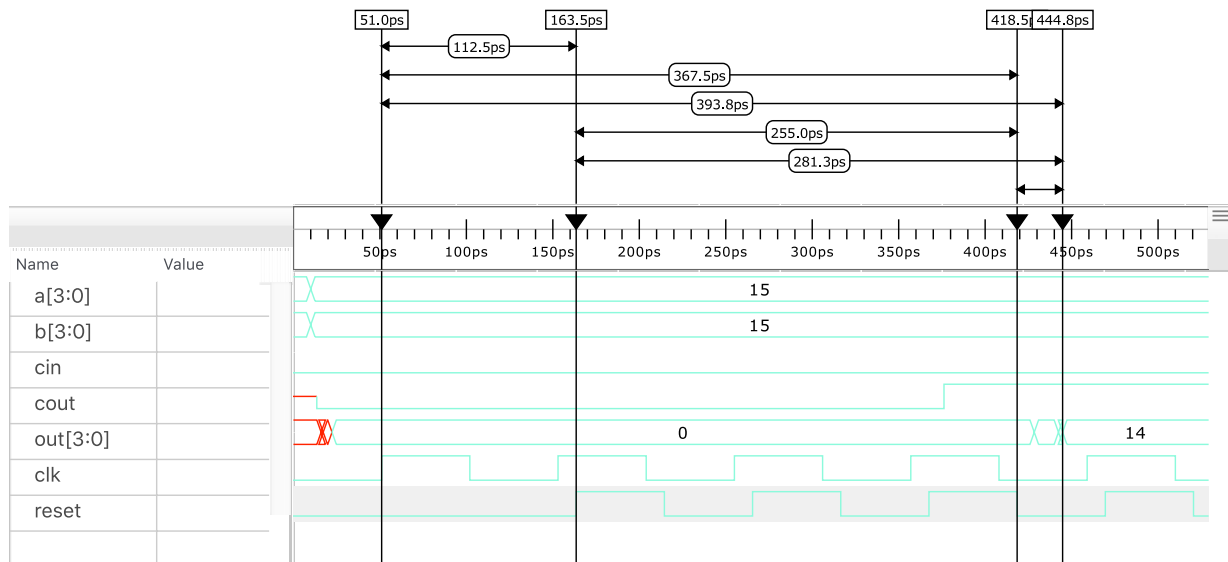


Figure 3.2: 4-bit KSA simulated as DUT at HDL level

The clock period in Figure 3.2 is determined through $t = \frac{1}{f}$, which in this case is 50.7ps. We adjust this to 51ps since the testbench does not accept fraction values for the clock. The time difference between the *clk* (clock input) and the *outclk* vectors is due to the delay created by the clocking circuitry. To simplify the implementation and legibility with SFQ HDL simulations we assume that every edge event is considered an SFQ pulse rather than attempting to mimic the width of the pulse. Noting this and observing the results we see that output is produced in the 6th clock cycle of the *outclk* vector.

If we alter the clock to 45ps, simply 6ps faster than the maximum speed calculated by SuperSTA we observe the result appearing in the wrong clock cycle violating the timing restrictions of the design. This can be seen in Figure 3.3

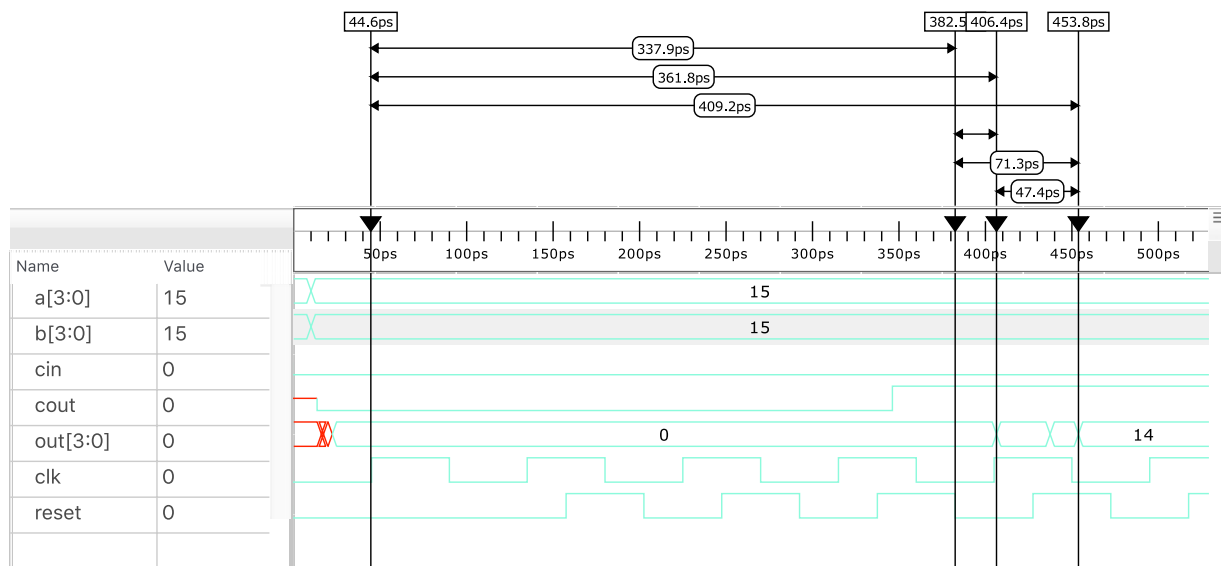


Figure 3.3: 4-bit KSA simulated with a clock speed higher than maximum identified

Additionally we note that the specified target time was met and that the slack presented is positive. This design therefore meets the set requirement of the designer and can therefore be moved further down the design process.

3.4 Conclusion

In this chapter a static timing analysis tool SuperSTA was developed and tested. An example was run through SuperSTA and the results scrutinized. Back-annotation into an HDL description of the example confirms the results of SuperSTA as correct. Results were also shown for the example when maximum clock speed is exceeded and the consequences thereof discussed.

The use of SuperSTA can greatly speed up the design process due to the ability to back-annotate results into a HDL simulation of the design. Future improvements to SuperSTA will include the analysis of a broader spectrum of design methodologies such as AQFP. We intend to continue development through inclusion of more elements that cause delay within design such as temperature, bias current effects and different clocking methods. Attempts to introduce parallel processing during the BFS stage of SuperSTA will also be investigated.

The unique contributions made through the implementation of a static timing analysis tool for superconducting circuit applications include a first of its kind tool as well as the definition of new types of delay factors involved in superconducting circuits. These contributions are further enforced through publication in an international journal[24] and is attached to this dissertation in appendix C.

Chapter 4

Analogue Simulation

4.1 Introduction

When the design moves to a lower level of abstraction, the need to perform analogue simulations arise. This requires the designer to have intrinsic knowledge of the electrical properties of the design. An analogue simulation could also be described as an electrical simulation and is used to compute electrical circuits. This is trivial for smaller circuits however as the component count increases, it quickly becomes necessary to use computer aided simulation software such as SPICE[15].

SPICE simulation in superconductivity is a rather niche field due to the complexity of the JJ element and non-linearities created by it. Most simulators rely on approximations such as the resistively and capacitive shunted junction (RCSJ) to model the tunnel current effect of the Josephson junction[4]. Regardless of being approximations, the modeled effect is suitable for simulation purpose and near enough to practical results to be acceptable in most cases. The closest approximation to the Josephson junction that models the Josephson effect most accurately was done by Werthamer in 1966[45][46]. This approximation though has not seen implementation in a general simulation engine.

The first documented case of an attempt at the Josephson effect in SPICE was by Jewett at University of California Berkeley in 1982[47]. The JJ model was added to the existing SPICE 2G5 and allowed the user to choose one of 3 types of quasiparticle resistances (Rtype). This method was however rather slow due to the numerical method used by SPICE for accurate simulation of transistor type devices. The SPICE 2G5 with the implementation of the JJ was named JSPICE.

In 1991, S. Whiteley was consulted to incorporate the JJ into the then new SPICE3 simulator which offered enhanced simulation performance and the implementation of a graphical post-processor for result plotting[48]. Similar to the implementation of the JJ in the original SPICE, this version was named JSPICE3. JSPICE3 also included an implementation of margin analysis using pass/fail tests. JSPICE3 is the base upon which WRspice is built.

The rest of this chapter is dedicated to discussing the Josephson junction as well the capabilities and drawbacks of the available superconducting circuit simulators. This will highlight a few key areas on which improvement can be made.

4.2 Josephson junction

A Josephson junction is created by bringing two superconducting elements very close to each other, only separating them with a thin insulating barrier (figure 4.1). At absolute zero temperature the superconductor has no resistance and therefore a zero voltage drop. This allows DC current to persist within a superconductor forever. When the current through the JJ is

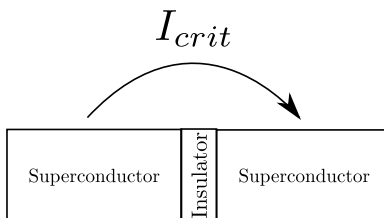


Figure 4.1: Basic example of a Josephson junction

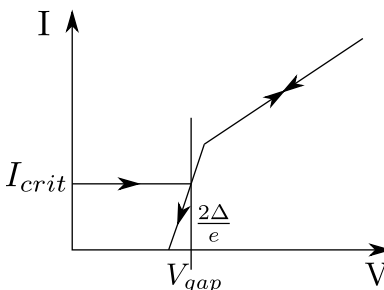


Figure 4.2: IV curve of a typical JJ

increased to some critical value, the voltage becomes non-zero and the electrons are said to tunnel across the insulator. This non-zero voltage is called the gap voltage and is presented using the equation

$$V_{gap} = \frac{2\Delta}{e} \quad (4.1)$$

where 2Δ is the binding energy of two electrons and e is the electron charge. When the current is further increased the junction approaches a linear resistance value. This resistance value is called the normal resistance and is associated with normal electron tunnelling. When the current is reduced, the curve follows the same path until it reaches the gap voltage after which it drops sharply to zero current [49]. The same occurs for negative currents the junction which creates a hysteresis curve. This phenomenon is called switching and is depicted in Figure 4.2.

4.3 JSIM

Josephson simulator (JSIM)[50], developed by Fang and Van Duzer in 1989, is a SPICE simulator dedicated to simulation of JJs and does not support any semiconductor devices. The simulator is very light weight due to the need to only support a few components. JSIM makes use of nodal analysis to compute solutions of large matrices.

JSIM, although being the most widely used superconducting circuit simulator, has received no update since the last release in 1992. It was well optimized at the time and remains so even on modern more powerful computing devices. However, many optimizations can be done to improve input methods and time to solution. The code was written with older memory architectures in mind using native C language where dynamic array sizes were hard to implement. This led to using pointers to form very long linked lists. These design choices make the alteration of the code and debugging to identify bottlenecks in large designs almost impossible.

JSIM, like JSPICE, is only capable of doing transient time domain simulations and cannot handle dc or ac simulations. What made JSIM unique to JSPICE is the implementation of the JJ requiring only the right hand side to be changed on every time step where JSPICE would change the \mathbf{A} matrix after each iteration. This gives JSIM a large speed improvement compared to JSPICE.

JSIM has only 2 implemented types of quasiparticle resistance models. $R_{type}=0$ meant

that the quasiparticle resistance was ignored and $R_{type}=1$ approximated the resistance as a piecewise linear function. When $R_{type}=1$ is specified the resistance used in the conductance matrix (A) for the Josephson junction would vary depending whether the junction voltage is below, in transition state or above the gap voltage.

The computing systems at the time of its creation were very limited in terms of memory and performance when compared to modern computers. This fact influences the size of simulations that JSIM can handle as it was written to be memory efficient. To further improve performance in JSIM, the time step size in the transient simulation is increased when the junction phase does not vary significantly within one time step. Similarly the time step is reduced when the phase difference becomes large and can also be reduced to less than the specified time step if the difference is greater than some threshold value.

The possibility of using JSIM to simulate VLSI designs, however, becomes impractical due to its failure in handling simulations that near 10 000 JJs. Regardless of this limitation JSIM is still regarded as one of the more popular simulators despite its age. We plot the current versus voltage (IV) curves for the JJ implementations in JSIM in Figure 4.3 and 4.4.

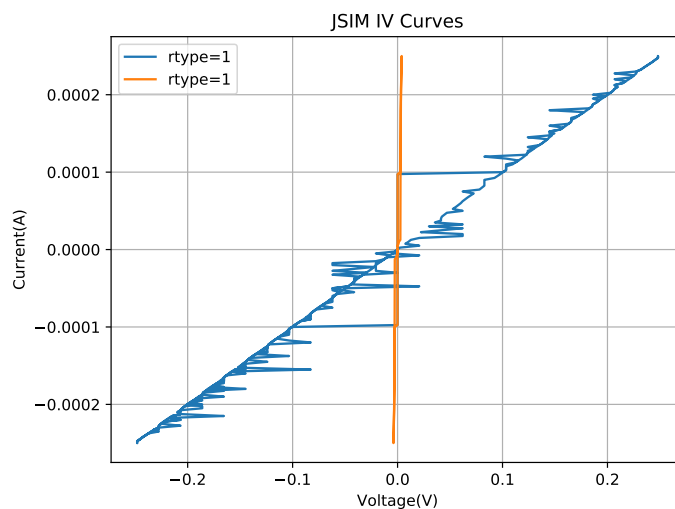


Figure 4.3: V curves of the RCSJ model in JSIM

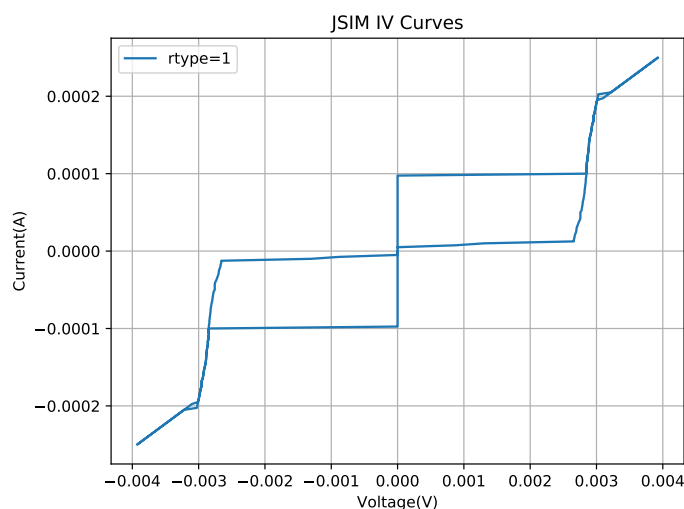


Figure 4.4: IV curve of the RCSJ model in JSIM without $R_{type}=0$

4.4 PSCAN

Personal superconductor circuit analyser (PSCAN) was written in 1991 by Polonsky at Moscow State University[51]. At the time only JSIM existed along with JSPICE2 and 3. All of which used the modified nodal voltage analysis approach to set up the linear equations. What PSCAN introduced was a solution that was labeled modified nodal phase analysis and made use of the relationship between the Josephson junction phase and voltage

$$v = \frac{\Phi_0}{2\pi} \frac{d\phi}{dt} \quad (4.2)$$

This method of simulation relies on satisfying London's quantization law for phase differences in a superconducting loop

$$\phi_1 + \dots + \phi_n = 0 \quad (4.3)$$

Unlike other SPICE simulators PSCAN utilizes a unique behavioural type of input language called SFQHDL[52]. This special input language made it possible for the designer to incorporate test vectors for self-verification within the input file. What this enabled was the incorporation of built-in margin optimization tools since the operation verification is done internally.

PSCAN was written in Fortran and was not available for testing publicly. PSCAN2 which was released in September 2016 by Pavel Shevchenko and builds on the development experience of the original PSCAN. PSCAN2 is written in Python and is available to the public at pscan2sim.org. Though the documentation indicate that it now accepts standard SPICE syntax, it however still requires a SFQHDL behaviour description file for which there seems to be no proper syntax guideline. We managed to get it up and running using the provided examples and used it to plot a simple JJ IV curve. This IV curve can be seen in Figure 4.5.

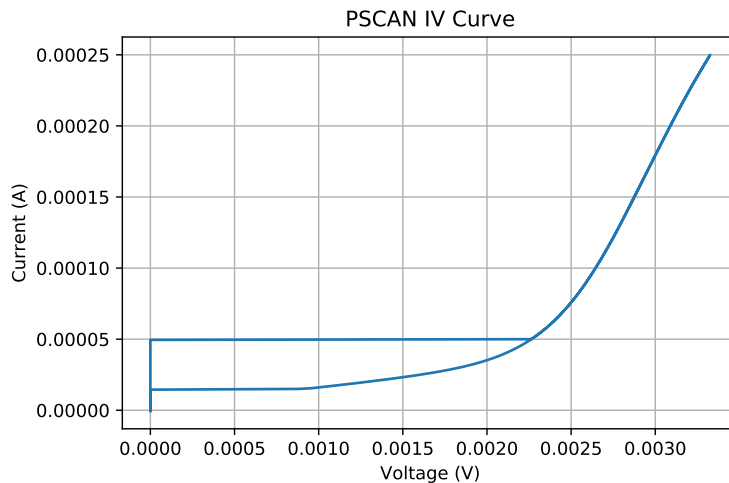
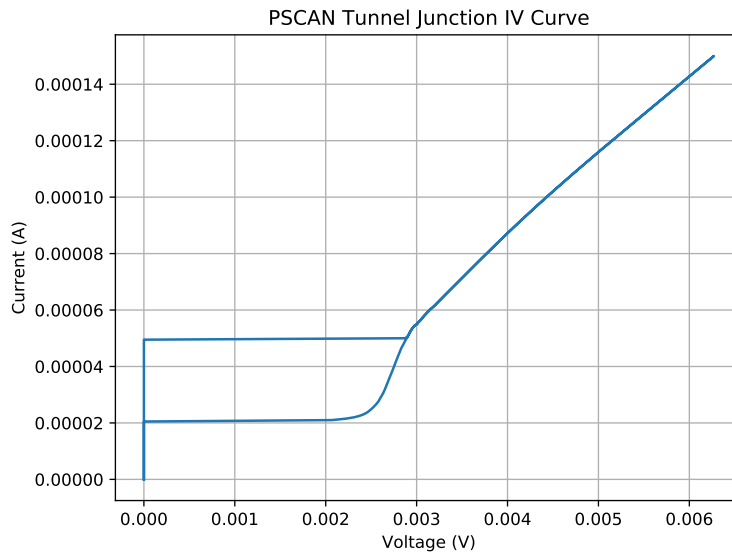
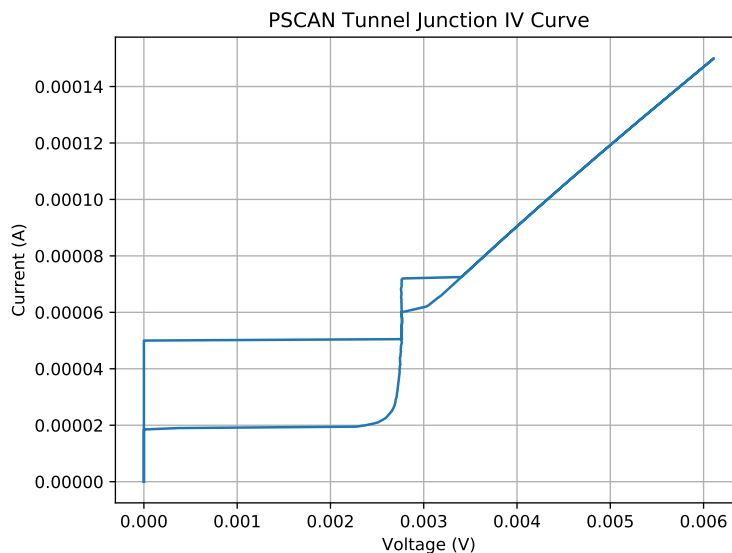


Figure 4.5: Josephson junction IV curve created using phase based simulator PSCAN

PSCAN2 includes an approximation to the microscopic tunnel junction as derived in [53] which uses a Dirichlet series approximations to model the tunnel junction behaviour. The Dirichlet series coefficients used by PSCAN2 can be set in one of the configuration files. We plot the two available tunnel junction model IV curves in Figure 4.6. One set of coefficients was provided by Prof Semenov and the other was borrowed from the tunnel junction coefficient generating tool MiTMoJCo[54].



(a)



(b)

Figure 4.6: Tunnel junction model IV curves created using PSCAN. Dirichlet coefficients used: (a) V. K. Semenov (b) MiTMoJCo

PSCAN2, unlike JSIM, includes a graphical interface window through which simulation options can be altered and results plotted. This GUI, like the rest of PSCAN2, requires the user to have in-depth knowledge of the PSCAN2 code since there is no user manual. Due to the lack of a proper syntax guideline and detrimental speed of Python, PSCAN2 is by no means a viable option to perform VLSI simulations.

4.5 WRspice

WRspice is a SPICE engine developed by Whiteley Research Incorporated in Sunnyvale, CA. Until October of 2017 it was a commercial SPICE engine and part of a toolset called XicTools, which included the layout package Xic. Development started as a project to rewrite JSPICE3 in C++ while maintaining full compatibility for older SPICE simulators.

WRspice unlike PSCAN and JSIM supports transistor components and was developed to

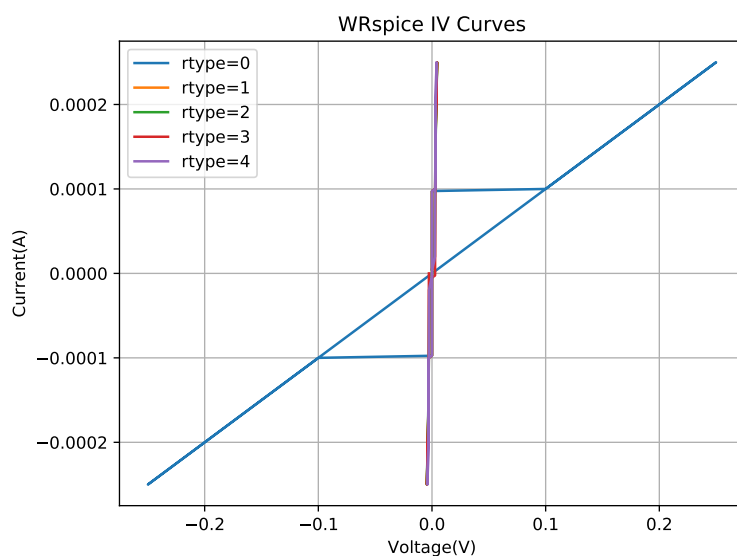
be fully graphical for ease of user interaction but still allows for command-line input. It is fully cross platform supporting Microsoft Windows, Linux and Apple MacOS.

WRspice is the most modern SPICE simulator available at present with capabilities such as network distribution, parallel processing and Monte Carlo analysis. It is well documented with an online help and manual system.

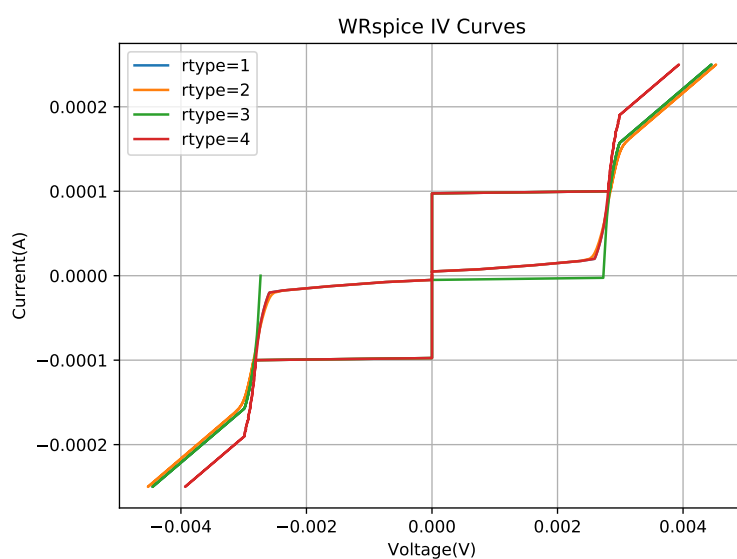
Similar to the JSPICE and JSIM the quasiparticle resistance type can be specified, however there are 5 different types unlike the 3 found in JSPICE and 2 in JSIM. WRspice supports transient analysis of any device as well as support for AC and DC (phase mode) simulations.

The integration with the layout editor Xic allows schematic editing and translation to the equivalent SPICE netlist for direct simulation. The ability for WRspice to view the simulation results directly through the graphical interface makes it the closest simulator in terms of features to what our project set out to achieve.

We compare the various implementations of JJ resistance types found in WRspice in Figure 4.7.



(a)



(b)

Figure 4.7: IV curves of the RCSJ model in WRspice. (a) With Rtype=0 (b) Without Rtype=0

4.6 Conclusion

In this chapter the Josephson junction as an element in analogue simulators was briefly examined. We discussed the various available circuit simulators, the features that make them unique as well as their downsides. The JJ element within each simulator was tested to form an IV curve for each type of resistance model.

To compare these simulators to each other the testing environment needs to be set up to perform the same simulation on all of the simulators. This becomes a tedious task due to the differences in syntax between them. It was not however possible to compare PSCAN2 to JSIM or WRspice due to the lack of a proper user guide and difficulty understanding the syntax. We were however able to compare JSIM to WRspice using the exact same circuit and resistance model. This comparison can be seen in Figure 4.8.

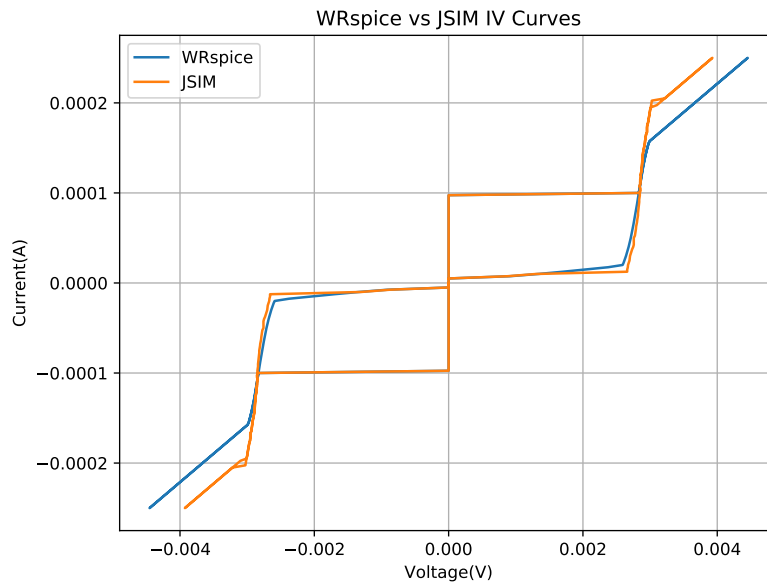


Figure 4.8: Rtype=1 model comparison between WRspice and JSIM

In Figure 4.8 we see that JSIM starts to let current through at a lower voltage and exits the sub-gap at a lower current level when compared to WRspice. This difference is largely due to the implementation each simulator uses to approximate the Josephson effect. This difference is further discussed in 5.3.4.2. This is achieved using the test circuit seen in Figure 4.9 and the following model for the JJ

```
.model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
```

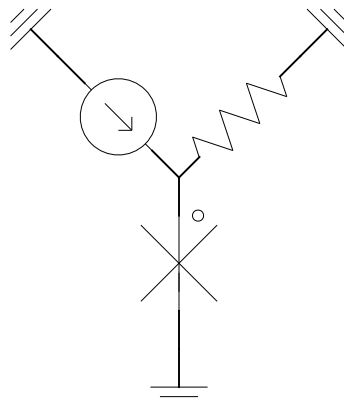


Figure 4.9: Circuit used to find the IV curve of a JJ

The IV curve is drawn by ramping up the current source to $2.5\mu\text{A}$ steadily until 50ps and then keeping it constant for the rest of the simulation. The simulation is run for 1ns and a step size of 0.1ps is used. The voltage across the JJ is then averaged for the last 500ps of the simulation and stored.

The current is increased by another $2.5\mu\text{A}$ and the simulation is repeated. This is done until the current reaches $250\mu\text{A}$, which is much more than the $100\mu\text{A}$ critical current of the JJ. The current is then ramped down back to zero incrementally and the same process is followed for the negative current.

This requires a total of 400 simulations to find an average voltage for each current value which when plotted presents the IV curve seen in Figure 4.8.

The same process is done for both WRspice and JSIM and the execution speeds of the 400 simulations are compared. In the test script we use to calculate the IV curve, the only variable is the simulator and the timing can therefore be directly compared. While using WRspice the script completed in 55.97s, where when using JSIM, it completed in 60.49s.

There are some slight differences between the piece-wise linear resistance models implemented in each simulator which could attribute to the difference in execution speed. WRspice does have the advantage of being written in C++ and using updated linear algebra methods such as KLU, but JSIM does not come in too far behind it despite its age.

This leads us to believe that a simulator written in C++ with updated linear algebra libraries that is exclusively designed for superconducting circuit simulation could be advantageous in the long run.

Chapter 5

JoSIM - Development

5.1 Introduction

In this chapter we will discuss the development decisions and design intricacies involved in the creation of the superconducting simulation engine JoSIM[25]. Due to JSIM and WRsice being the most common superconducting simulation engines at the start of this development process, the decision was made to utilize the same SPICE syntax used in both. This should allow simulation of existing netlists and direct comparison of results and simulation times.

JoSIM was initially developed as part of this project to help motivate the argument of this dissertation. It was intended to be used to simulate VLSI designs in reasonable time by making use of advanced linear algebra libraries and modern day computing systems with parallel processing. Through the course of development, the need for JoSIM was further enhanced through the US Government sponsored IARPA project Super Tools [28], wherein the purpose of JoSIM was altered to provide simulation results of more accurate approximations of the JJ behaviour through incorporation of more intricate models. Additional requirements were set for JoSIM to be able to elegantly plot the results of a simulation to some graphical form that should be publication ready such as scalable vector graphic (SVG) format.

JoSIM is developed in standard C++ to provide compatibility across multiple platforms and is open source at [JoSIM.git](https://github.com/JoSIM) with the MIT license. Code is also commented as far as possible to help legibility and aid future developers of the code. Documentation in the form of a user manual is also made available and can be found in Appendix E.

Code was initially written using a MSI GE72 6QF running Microsoft Windows 10 and Microsoft Visual Studio Enterprise with a student license. Development was later shifted to a Apple Macbook Pro 13" (2017) running mac OS High Sierra and Visual Studio Code. This change was made to eliminate the need for a nearby power outlet whenever inspiration struck.

This chapter is further divided into the different sections of JoSIM as a simulation process and is depicted in the design flow shown in Figure 5.1.

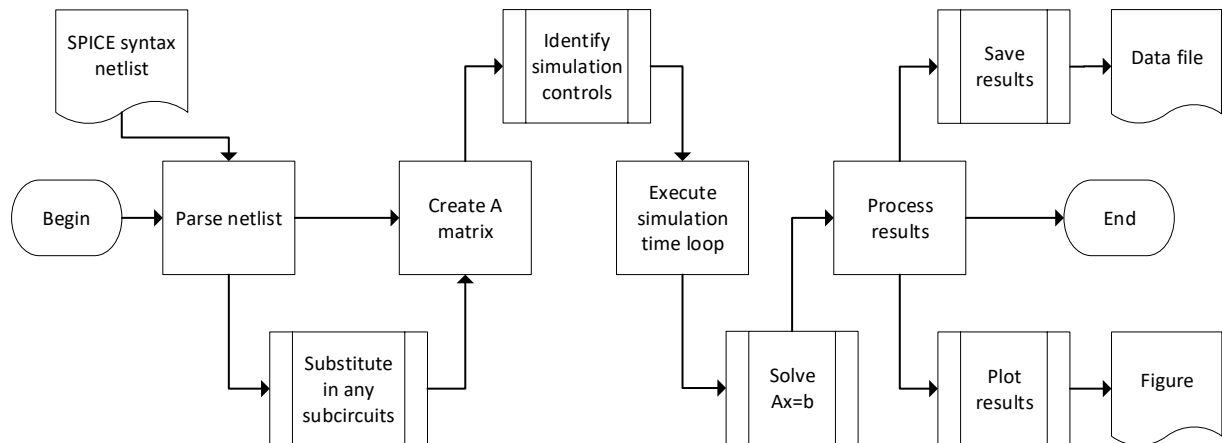


Figure 5.1: Overview of the JoSIM design flow

5.2 Design Flow from Input to Output

5.2.1 Input

As mentioned, input to JoSIM is aimed to provide as much compatibility with standard SPICE syntax and closely follows the syntax found in The SPICE Book[55]. The user would provide a netlist containing multiple lines of component definitions and interconnect information. Component definitions are further elaborated on in 5.3. These netlist files would also contain control lines that define what JoSIM has to do with the components it has read in. The types of control lines available in JoSIM are discussed in 5.4.

When a netlist is parsed, each line is read in and stored either as main design, controls or subcircuit under the corresponding subcircuit section. Once the entire file has been parsed we iterate through both the main design substituting the relevant subcircuit lines when a subcircuit definition is found. This process is recursive to included nested subcircuits. In JoSIM we allow alphanumeric node names as opposed to the standard numeric found in JSIM. This allows us to append the corresponding subcircuit label to the node names and labels of the lines being substituted. This process is illustrated in Figure 5.2 and 5.3.

```

.subckt RLC 1 4
R01 1 2 5
L01 2 3 10
C01 3 4 20
.ends RLC

V01 0 1 PWL(0 5 10 20)
R01 1 2 5
L01 2 3 10
X01 RLC 3 4
C01 4 0 20
.end
  
```

Figure 5.2: Example of a standard netlist with a subcircuit

```

V01 0 1 PWL(0 5 10 20)
R01 1 2 5
L01 2 3 10
X01 RLC 3 4
R01 | X01 3 2 | X01 5
L01 | X01 2 | X01 3 | X01 10
C01 | X01 3 | X01 4 20
C01 4 0 20
.end

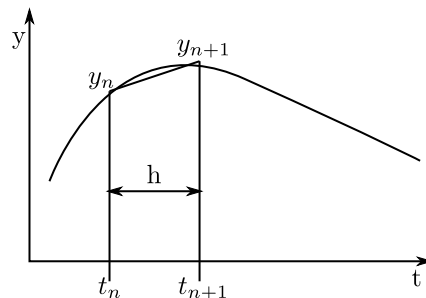
```

Figure 5.3: Resulting master netlist after substitution

Once this master netlist has been generated we now iterate through each individual line, deciding what needs to be done with it based on the first character of each line. These components are then transformed into their relevant MNA stamp.

5.2.2 Matrix Setup

When attempting to solve a set of linear equations to find the voltage at every node as in the modified nodal analysis (MNA), we need to first decide on an integration method that would approximate the current or voltage of non-linear components. The most basic of these methods is simply the backward Euler method, which interpolates the value based on the previous value.

Figure 5.4: Approximating the y_{n+1} value

From Figure 5.4 we can write backward Euler differential as

$$\frac{dy}{dt}_n = \frac{y_{n+1} - y_n}{h} \quad (5.1)$$

This method is however a first order method which does not model the behaviour accurately enough and is prone to error. We therefore opt to use a second order method such as the trapezoidal integration method. The trapezoidal integration method can be defined as

$$\frac{dy}{dt}_n = \frac{2}{h}(y_n - y_{n-1}) - \frac{dy}{dt}_{n-1} \quad (5.2)$$

where n is the current time step and h is the difference between the current time step and the next. The trapezoidal method is still prone to error if sharp spikes in y occur between t_n and t_{n+1} . There are other methods which approximate the non-linear curve more accurately however these methods require additional computation and have not been considered for implementation yet.

Using the trapezoidal integration method we can calculate a linear approximation to non-linear devices such as an inductor where the current cannot instantaneously change. Substituting the trapezoidal method into the standard voltage and current equations for every component allows us to create matrices called the MNA stamps[56]. These stamps are very sparse and when all the stamps for every component in the system have been collected and collated we are left with a very sparse \mathbf{A} matrix. The \mathbf{A} matrix is required to be square, which means the amount of rows equal the amount of columns. This \mathbf{A} matrix can then be used to solve the system $\mathbf{Ax} = \mathbf{b}$.

5.2.3 Solution Calculation

To calculate the voltage and current at the present time step we need to solve $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. There are various C++ libraries that do this very efficiently. We, however, chose KLU from the SuiteSparse linear algebra library[57]. For KLU to perform operations on the sparse \mathbf{A} matrix we need to convert it to compressed sparse row (CSR) format, which essentially consists of 3 vectors that completely describe a sparse matrix of any size. These 3 vectors are the nonzero vector, the count of nonzero elements in every row and the index of every nonzero value. This is probably better explained using an example.

$$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \quad (5.3)$$

The matrix in (5.3) requires 49 values to completely be described. However, if we convert this to CSR we find

$$A = [2, 5, 8, 9, 1, 2, 6, 7, 3, 5]$$

$$IA = [0, 1, 2, 4, 5, 7, 8, 10]$$

$$JA = [1, 3, 5, 6, 0, 2, 4, 3, 1, 6]$$

which completely describe the matrix in 5.3 using only 28 values, rather than the 49 required by the sparse matrix.

When we pass this CSR format to the KLU solver, a symbolic solution is first calculated where after the numerical solution is calculated at every time step. If at any point the \mathbf{A} matrix needs to be updated, a new symbolic calculation needs to be performed where after numerical solutions can again be calculated for every time step.

If there exists a row within the \mathbf{A} matrix that does not contain a single value, the matrix is deemed singular and no solution will be found for it. The simulator will then exit and provide the user with an appropriate error message.

5.2.4 Output Handling

Once all the unknown values are calculated for the entire simulation period we are able to present the results as either graphical results, numerical results or both. In the case of graphical presentation we make use of the fast light toolkit (FLTK)[58] to very crudely plot the results of a simulation. This library is cross platform and can be statically compiled into the executable of JoSIM. This plotting method however has no way of saving the produced results as well

as no way of identifying the values being plotted. The functionality of FLTK is theoretically endless and further effort will be put into developing this plotting method in the future.

Due to the crudeness of the FLTK plotting method we sourced alternative methods of displaying the results in a way that can be saved for publication. One such alternative identified is to use the Python library Matplotlib. This library has an interface to C++ that can be found at `matplotlib-cpp` which interfaces with the Python installed on the host machine. Although this plotting mechanism requires additional set up per individual system, the plots that are produced can be saved in a publication ready format.

Apart from the plotting methods, we also developed methods to write the results out to file in various formats. The intention is to use a universal standard such as comma separated value (CSV) which will enable the ease of use with multiple standard third-party plotting tools. In addition to this we also developed a method to produce a space separated value file which corresponds to the JSIM output format.

If no plotting or output command is specified the results are simply written to the standard output of the host, which in this case would be the command line.

5.3 Components

5.3.1 Resistor

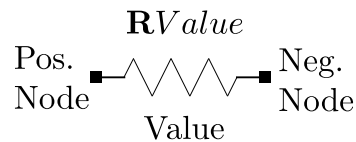


Figure 5.5: A basic resistor element.

The definition of a resistor in JoSIM is

RLabel Pos.Node Neg.Node Value

The voltage across a resistor is defined as

$$v(t) = Ri(t) \quad (5.4)$$

and written as a function of the current time step

$$V_n = RI_n$$

$$\frac{1}{R}V_n = I_n$$

Since we do not need to calculate the current we can leave it on the right hand side (RHS) and only use it if the value is known

$$\frac{1}{R}V_n^+ - \frac{1}{R}V_n^- = I_n \quad (5.5)$$

This can be written in matrix form as

$$\begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} V_n^+ \\ V_n^- \end{bmatrix} = \begin{bmatrix} I_n \\ -I_n \end{bmatrix} \quad (5.6)$$

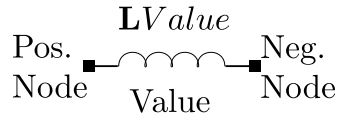


Figure 5.6: A basic inductor element

5.3.2 Inductor

A inductor in JoSIM has the following definition

LLabel Pos.Node Neg.Node Value

The voltage across an inductor can be described by

$$v(t) = L \frac{di(t)}{dt} \quad (5.7)$$

which can be written as a function of the current time step as

$$V_n = L \left(\frac{dI}{dt} \right)_n$$

$$V_n = L \left[\frac{2}{h_n} (I_n - I_{n-1}) - \left(\frac{dI}{dt} \right)_{n-1} \right]$$

Knowing that $L \left(\frac{dI}{dt} \right)_{n-1} = V_{n-1}$ we can write

$$V_n = \frac{2L}{h_n} I_n - \frac{2L}{h_n} I_{n-1} - V_{n-1}$$

$$V_n - \frac{2L}{h_n} I_n = -\frac{2L}{h_n} I_{n-1} - V_{n-1}$$

Since we want the voltages at each node and the branch current we can split the voltage as the difference between the two nodes.

$$V_n^+ - V_n^- - \frac{2L}{h_n} I_n = -\frac{2L}{h_n} I_{n-1} - (V_{n-1}^+ - V_{n-1}^-) \quad (5.8)$$

We can then neatly transform (5.8) to matrix form as

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{2L}{h_n} \end{bmatrix} \begin{bmatrix} V_n^+ \\ V_n^- \\ I_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L}{h_n} I_{n-1} - (V_{n-1}^+ - V_{n-1}^-) \end{bmatrix} \quad (5.9)$$

5.3.3 Capacitor

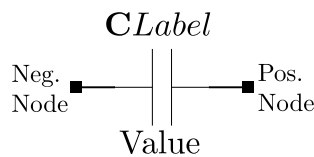


Figure 5.7: A basic capacitor element

A capacitor in JoSIM has the following definition

CLabel Pos.Node Neg.Node Value

The current through a capacitor is described by

$$i(t) = C \frac{dv(t)}{dt} \quad (5.10)$$

which when viewed as a function of the current time step

$$I_n = C \frac{dV_n}{dt}$$

$$I_n = C \left[\frac{2}{h_n} (V_n - V_{n-1}) - \left(\frac{dV}{dt} \right)_{n-1} \right]$$

The second differential is simply the current at the previous time step

$$I_n = \frac{2C}{h_n} V_n - \frac{2C}{h_n} V_{n-1} - I_{n-1}$$

$$V_n - \frac{h_n}{2C} I_n = -V_{n-1} - \frac{h_n}{2C} I_{n-1}$$

We now substitute the voltage as the difference between the two nodes

$$V_n^+ - V_n^- - \frac{h_n}{2C} I_n = - (V_{n-1}^+ - V_{n-1}^-) - \frac{h_n}{2C} I_{n-1} \quad (5.11)$$

We can now write (5.11) in matrix form as

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & \frac{h_n}{2C} \end{bmatrix} \begin{bmatrix} V_n^+ \\ V_n^- \\ I_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ - (V_{n-1}^+ - V_{n-1}^-) - \frac{h_n}{2C} I_{n-1} \end{bmatrix} \quad (5.12)$$

5.3.4 Josephson Junction

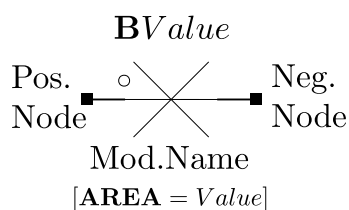


Figure 5.8: A basic Josephson junction element

A Josephson junction in JoSIM has the following definition

BLabel Pos.Node Neg.Node Mod.Name [**AREA**=Value]

5.3.4.1 RCSJ

The resistively and capacitively shunted junction model for the Josephson junction is shown in Figure 5.9 and has the following model description

.model Mod.Name
jj([**RTYPE**=Value],[**ICRIT**=Value],[**VG**=Value],[**RN**=Value],[**R0**=Value],[**CAP**=Value])

Both the component and model definition have specifiers that have default values. These values are:

AREA	1
RTYPE	0
ICRIT	1mA
VG	2.8mV
RN	5
R0	30
CAP	2.5pF

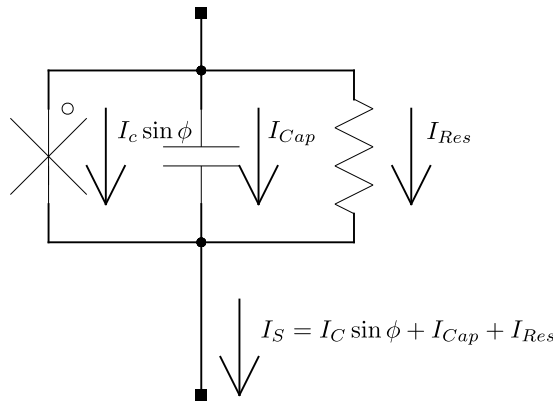


Figure 5.9: The resistively and capacitively shunted equivalent junction model

The equations that describe the voltage and current of a RCSJ are given by

$$i(t) = I_c \sin \phi(t) + C \frac{dv(t)}{dt} + \frac{1}{R} v(t) \quad (5.13)$$

$$\dot{\phi}(t) = \frac{2e}{\hbar} v(t) \quad (5.14)$$

where I_c the junction critical current, ϕ the junction phase, e the electron charge and \hbar Plank's constant. If we write these equations in terms of the current time step we find

$$I_n = I_c \sin \phi_n + \frac{1}{R} V_n + C \left[\frac{2}{h_n} (V_n - V_{n-1}) - \left(\frac{dV}{dt} \right)_{n-1} \right]$$

$$I_n - \left(\frac{1}{R} + \frac{2C}{h_n} \right) V_n = I_c \sin \phi_n - \frac{2C}{h_n} V_{n-1} - C \left(\frac{dV}{dt} \right)_{n-1}$$

$$\left(\frac{1}{R} + \frac{2C}{h_n} \right) V_n - I_n = -I_c \sin \phi_n + \frac{2C}{h_n} V_{n-1} + C \dot{V}_{n-1}$$

When written with the junction voltage as the difference between the two node voltages

$$\left(\frac{1}{R} + \frac{2C}{h_n} \right) (V_n^+ - V_n^-) - I_n = -I_c \sin \phi_n + \frac{2C}{h_n} (V_{n-1}^+ - V_{n-1}^-) + C \dot{V}_{n-1} \quad (5.15)$$

The phase equation is handled similarly

$$\begin{aligned}
\dot{\phi}_n &= \frac{2e}{\hbar} V_n \\
\frac{2}{h_n} (\phi_n - \phi_{n-1}) - \dot{\phi}_{n-1} &= \frac{2e}{\hbar} V_n \\
-\frac{h_n}{2} \frac{2e}{\hbar} V_n + \phi_n &= \phi_{n-1} + \frac{h_n}{2} \dot{\phi}_{n-1} \\
-\frac{h_n}{2} \frac{2e}{\hbar} V_n + \phi_n &= \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} V_{n-1} \\
-\frac{h_n}{2} \frac{2e}{\hbar} (V_n^+ - V_n^-) + \phi_n &= \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} (V_{n-1}^+ - V_{n-1}^-)
\end{aligned} \tag{5.16}$$

If we now combine (5.15) and (5.16) into a single matrix, we find

$$\begin{bmatrix} \frac{1}{R} + \frac{2C}{h_n} & -\frac{1}{R} - \frac{2C}{h_n} & 0 \\ -\frac{1}{R} - \frac{2C}{h_n} & \frac{1}{R} + \frac{2C}{h_n} & 0 \\ -\frac{h_n}{2} \frac{2e}{\hbar} & \frac{h_n}{2} \frac{2e}{\hbar} & 1 \end{bmatrix} \begin{bmatrix} V_n^+ \\ V_n^- \\ \phi_n \end{bmatrix} = \begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} (V_{n-1}^+ - V_{n-1}^-) \end{bmatrix} \tag{5.17}$$

Where the current value on the RHS is defined as

$$I_s = -I_c \sin \phi_n^0 + \frac{2C}{h_n} (V_{n-1}^+ - V_{n-1}^-) + C\dot{V}_{n-1} \tag{5.18}$$

The RHS of the matrix in (5.17) requires that we . We therefore define the phase guess (ϕ^0) as

$$\phi_n^0 = \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} (V_{n-1} + v_n^0) \tag{5.19}$$

with

$$v_n^0 = V_{n-1} + h_n \dot{V}_{n-1} \tag{5.20}$$

The only caveat in guessing the next voltage is when the previous value differs quite significantly within a time step (large $\frac{d}{dt}$) the difference between the phase guess and the previous phase value becomes extremely large causing unwanted behaviour. This occurs especially when initial conditions are zero and the previous two voltage values are therefore zero. We combat this by keeping the phase guess zero until around the third or fourth time step.

5.3.4.2 RCSJ resistance models

JoSIM, like JSIM, has only 2 types of resistance models implemented for the RCSJ model. These are namely Rtype=0 and 1, and will be discussed in further detail in this subsection.

Rtype=0 quite simply refers to the fact that there is no conductance present from the resistance branch in (5.17) and thus theoretically infinite resistance. We do however in the case of Rtype=0 in JoSIM simply leave this as the subgap resistance (R_0) since in practice there it is impossible to reach a temperature of absolute 0 and thus there will always be a resistance.

The implementation of Rtype=1 requires a bit more thought since it represents a piecewise linear conductance surrounding the voltage gap. JSIM defines an area that specifies 2 regions wherein the conductance transitions between the subgap conductance and normal conductance. These regions are $V_{gap} + \Delta V$ and $V_{gap} + 2\Delta V$ where ΔV is the gap transition voltage and is set to 0.1mV by default, but can be specified in the junction model.

As the junction current increases and approaches the critical current, the voltage guess (v^0) is monitored. If the the voltage for the next step is guessed to cross the voltage gap, the junction conductance on the LHS in (5.17) is altered to a weighted value described as

$$\frac{2C}{h_n} + G_T \quad (5.21)$$

where G_T is the transition conductance

$$G_T = \frac{\frac{V_{gap} + \Delta V}{R_n} - \frac{V_{gap}}{R_0}}{\Delta V} \quad (5.22)$$

This change requires that an LU decomposition be performed again using the new \mathbf{A} matrix. This alone however does not accurately describe the path the voltage follows and thus more needs to be taken into consideration. An additional current value is added to the RHS I_s to compensate for this transition period and can be specified as

$$I_T = V_{gap} \left(\frac{1}{R_0} - G_T \right) \quad (5.23)$$

while $V_{gap} < V^0 \leq V_{gap} + \Delta V$ and

$$I_T = V^0 \left(\frac{1}{R_n} - G_T \right) \quad (5.24)$$

while $V_{gap} + \Delta V < V^0 \leq V_{gap} + 2\Delta V$.

This minimizes the amount of LU decompositions required as it only now needs to be done 3 times during a junction switch instead of at every time step during the switch.

This method does however not take into account the ratio of critical current to quasiparticle step height as seen in WRspice. With this parameter being user tunable it allows for more accurately characterized junction behaviour above the subgap region. Similar to JSIM, the transition is defined as a region surrounding the gap voltage with a transition conductance. This region is defined as $V_{gap} - \frac{1}{2}\Delta V$ and $V_{gap} + \frac{1}{2}\Delta V$ where the conductance is

$$G_T = \frac{I_c}{(I_{cfact} \times \Delta V)} \quad (5.25)$$

with I_{cfact} being the ratio of critical to gap current with a default value of $\frac{\pi}{4}$. When the junction voltage enters the region surrounding the gap voltage the junction conductance in the \mathbf{A} matrix is changed and a constant current is added to the RHS. This constant current is the same as in (5.23).

When the junction voltage exits above this region the transitional current is however not set to 0 as in JSIM but given a constant value as defined in (5.26).

$$I_T = \frac{I_c}{I_{cfact}} + \left[\frac{1}{R_0} \times \left(V_{gap} - \frac{1}{2}\Delta V \right) \right] - \left[\frac{1}{R_n} \times \left(V_{gap} + \frac{1}{2}\Delta V \right) \right] \quad (5.26)$$

This method of handling Rtype=1 is preferred and is the method used in JoSIM as it matches measured results from fabrication processes such as MITLL.

Though WRspice includes more quasiparticle resistance models such as a analytic exponentially-derived approximation (Rtype=2) and a fifth order polynomial expansion model (Rtype=3), these have yet to be investigated for implementation in JoSIM.

5.3.4.3 CPR

The current phase relation (CPR) is defined through the sinusoidal supercurrent equation

$$I_s = I_c \sin(\phi) \quad (5.27)$$

which holds true for most cases of electron tunneling between two superconducting materials. However, an investigation by Haberkorn [59] shows that this CPR can be non-sinusoidal, especially when considering ballistic tunnelling.

This non-sinusoidal effect is shown in the following equation.

$$I_s = \frac{\pi\Delta}{2eR_N} \frac{\sin\phi}{\sqrt{1 - \bar{D} \sin^2\left(\frac{\phi}{2}\right)}} \tanh\left[\frac{\Delta}{2k_B T} \sqrt{1 - \bar{D} \sin^2\left(\frac{\phi}{2}\right)}\right] \quad (5.28)$$

The equation in 5.28 introduces the temperature dependency through Δ , which is defined as

$$\Delta_0 = 1.76k_B T_c \quad (5.29)$$

$$\Delta = \Delta_0 \sqrt{\cos\left[\frac{\pi}{2} \left(\frac{T}{T_c}\right)^2\right]} \quad (5.30)$$

with T , the boiling point of liquid Helium, being 4.2K and T_c , the critical temperature of Niobium, being 9.1K. k_B is Boltzmann's constant for average kinetic energy of particles.

The resistance value R_N is defined as

$$R_N = \frac{\pi\Delta}{2eI_c} \tanh\left(\frac{\Delta}{2k_B T}\right) \quad (5.31)$$

This allows us to change the characteristics of the tunnel current by simply altering the value of \bar{D} . For values of $\bar{D} \ll 1$ the equation becomes the normal sinusoidal equation whereas for large values of \bar{D} it becomes the non-sinusoidal ballistic tunneling equation.

Implementation of this equation introduces temperature dependence and inches us ever closer to a more accurate representation of the Josephson junction dynamics through simulation.

5.3.4.4 MTJ

The microscopic tunnel junction (MTJ) in [46], is described by Kratz as being a much more accurate approximation of the tunnel current phenomenon seen in the Josephson junction. Kratz calls this the Werthamer approximation after the Werthamer theory seen in [45]. We therefore aspire to incorporate this more accurate approximation into JoSIM.

Other simulators such as PSCAN[51] and MiTMoJCo[54] approximate the MTJ through using a set of coefficients for a Dirichlet series. This, however, is not the method of implementation described in [46], where the MTJ is represented by the element model in Figure 5.10.

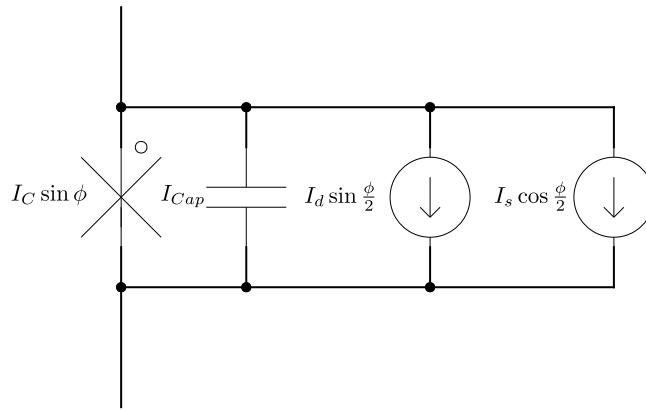


Figure 5.10: Element model of the Werthamer approximation

The two current sources in Figure 5.10 are the sum and difference currents created using the element model in Figure 5.11.

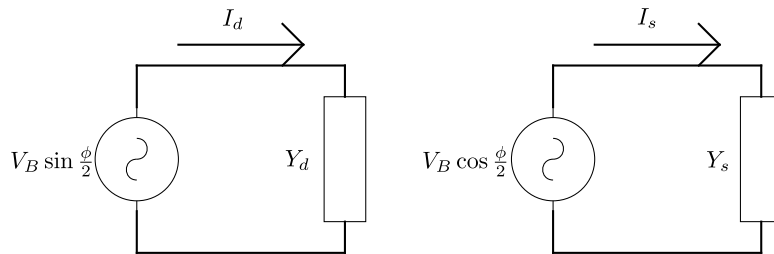


Figure 5.11: Sum and difference current calculation using the junction voltage

The sum and difference admittances seen in Figure 5.11 are represented by the equivalent circuits in Figure 5.12.

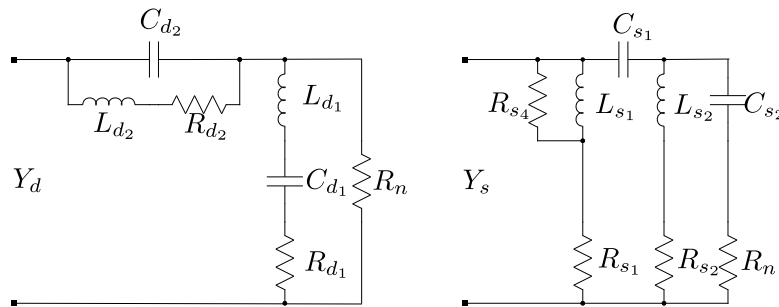


Figure 5.12: Sum and difference admittance equivalent circuits

These admittances are complex and therefore require a discrete Fourier transform on every time step. This, however, is very time consuming and will require many changes to the core engine of JoSIM to see implementation.

5.3.5 Voltage and Current Sources

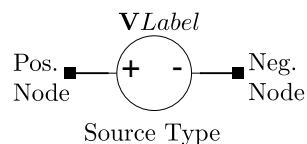


Figure 5.13: A basic voltage source

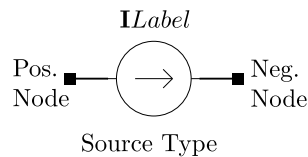


Figure 5.14: A basic current source

Voltage sources in JoSIM have the definition

$$\mathbf{VLabel} \quad \text{Pos.Node} \quad \text{Neg.Node} \quad \text{Source Type}$$

and similarly current sources

$$\mathbf{ILabel} \quad \text{Pos.Node} \quad \text{Neg.Node} \quad \text{Source Type}$$

with the voltage causing predefined values at the specified nodes on the RHS and current causing predefined values at the corresponding nodes on the LHS. The source types could be either piecewise linear (PWL), pulse or sinusoidal. The option to include DC sources was considered, but ultimately deemed unnecessary, since in the case of performing a transient analysis, the differential in components such as inductors and capacitors will always be zero. If a DC operating point simulation method is to be implemented in the future, the DC source will be added.

5.3.5.1 PWL

PWL source types have the following description

$$\mathbf{PWL}(t_0 \ v_0 \ t_1 \ v_1 \ \dots \ t_n \ v_n)$$

where t is the time point and v is the value at the corresponding time point. We always assume that the initial time and initial value is zero, as this is the start of the simulation. Given that t_k is the current time step we create a linear interpolation from v_{k-1} to v_k in the time period t_{k-1} to t_k . Doing this for all n time values specified builds a complete set of values that define the piecewise linear function.

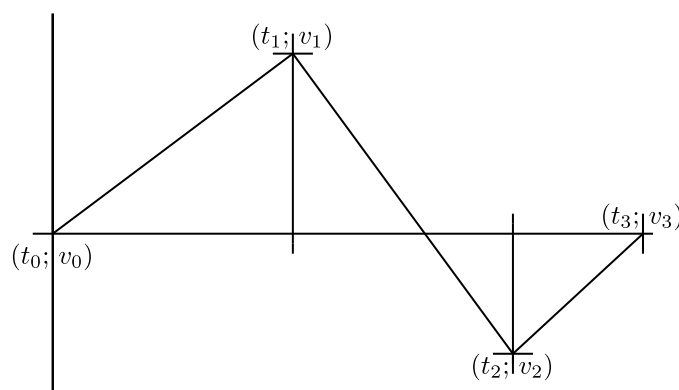


Figure 5.15: Example of a PWL function depicting the relevant values

5.3.5.2 Pulse

Pulse source types have the following description

$$\mathbf{PULSE}(V_1 \ V_2 \ [T_D \ [T_R \ [T_F \ [PW \ [PER]]]])$$

where V_1 and V_2 are the two voltage levels. T_D , T_R , T_F , PW and PER are the delay time, rise time, fall time, pulse width and period respectively. The values from time delay to period are optional but require the preceding value to function, as they are read in a specific order. If not specified, these values have the following default values.

T_D	0
T_R	Time step size
T_F	Time step size
PW	Stop time
PER	Stop time

The time step size and stop time are that of the transient simulation being run. Pulse functions are implemented in a similar way to PWL functions in that the time points and values are generated using the relevant parameters, where after the same procedure is followed to generate the function values.

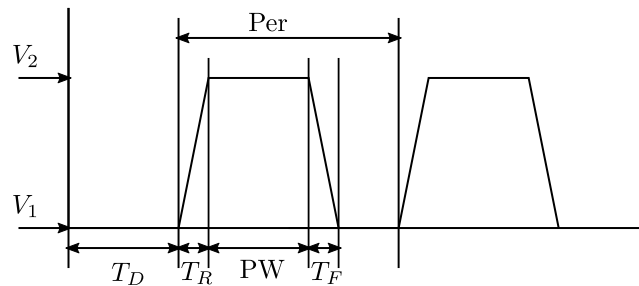


Figure 5.16: Example of a pulse function depicting the relevant values

5.3.5.3 Sinusoidal

Sinusoidal source types have the following description

$$\text{SIN}(V_O \ V_A \ [\text{FREQ} \ [T_D \ [\text{THETA}]]])$$

where V_O is the offset, V_A the amplitude, $FREQ$ the frequency and T_D the time delay. We include the damping factor $THETA$ for completeness as this is used to modulate the signal's amplitude if need be. $FREQ$ to $THETA$ can be left out upon which default values will be assumed. The parameters are consecutively read in and therefore need to be provided in the correct order.

$FREQ$	1/Stop time
T_D	0
$THETA$	0

The general function for a sinusoidal function would then be written as

$$f(t) = V_O + V_A \sin(2\pi FREQ(t - T_D)) e^{-THETA(t - T_D)} \quad (5.32)$$

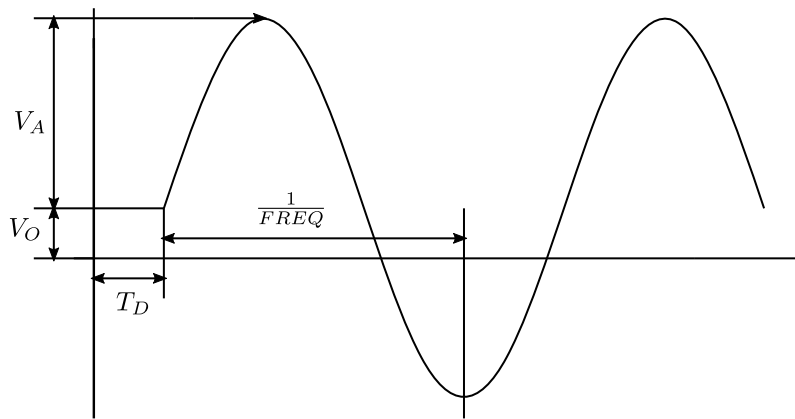


Figure 5.17: Example of a sinusoidal function depicting the relevant values

5.3.5.4 Custom Waveform

Custom waveform source types have the following description

$$\text{CUS}(\text{wavefile } T_S \text{ SF IM } [T_D \text{ PER}])$$

where T_S is the step size, SF the scaling factor, IM the interpolation method, T_D the time delay and PER the periodicity. T_D and PER can be left out upon which default values will be assumed. The parameters are consecutively read in and therefore need to be provided in the correct order.

T_S	Time step size
SF	1
IM	1
T_D	0
PER	0

The custom waveform source type allows the user to provide a single line, space separated waveform file from which the amplitudes of each point of the wave is created and scaled using the scaling factor (SF). The values between the points are interpolated using either no interpolation (0), linear (1), cubic (2) or spline (3). The function can become periodic if PER is set to 1, whereby the pattern is repeated for the entire simulation

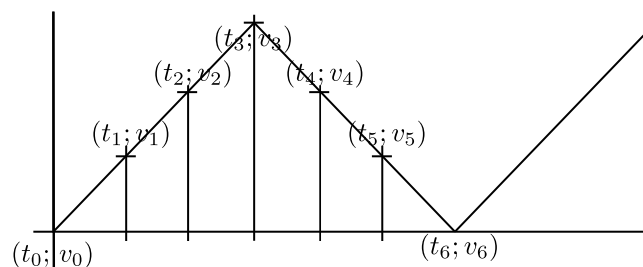


Figure 5.18: Example of a custom function depicting a waveform line of [0 1 2 3 2 1 0] with periodicity enabled to create a triangle wave.

5.3.6 Lossless Transmission Line

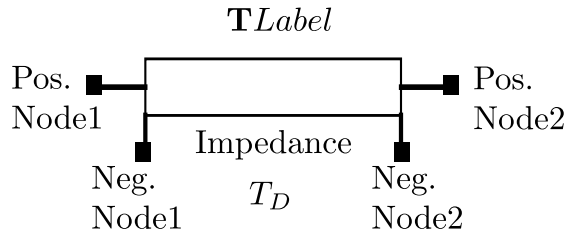


Figure 5.19: A basic transmission line element

A lossless transmission line in JoSIM is defined through

TLabel Pos.Node1 Neg.Node1 Pos.Node2 Neg.Node2 [**TD**=Value **Z0**=Value]

with T_D the time delay and Z_0 the line impedance. Both are optional and have default values

T_D	0
Z_0	10

The high speed of superconducting electronics makes interconnection between logic gates slightly problematic. The energy content of a SFQ pulse can quite quickly diminish to the point where it no longer resembles a pulse and thus not transfer the correct data. This can be corrected through pulse regeneration circuits such as the Josephson transmission line (JTL). JTLs need to be biased and in the case of RSFQ consume static power. This makes them a costly component if the signal needs to propagate over quite a distance. Additionally, each JTL adds roughly a 5ps delay, which also slows performance in large designs.

To combat the use of JTLs, designers of superconducting electronics make use of passive transmission lines (PTLs) through which SFQ pulses can be transferred at near light speed with minimal loss of energy[60][61]. The equations that govern this transmission are described by

$$v_1(t) - Z_0 i_1(t) = v_2(t - T_D) + Z_0 i_2(t - T_D) \quad (5.33)$$

$$v_2(t) - Z_0 i_2(t) = v_1(t - T_D) + Z_0 i_1(t - T_D) \quad (5.34)$$

The time delay in terms of the current time step can be written as

$$t - T_D \equiv n - \frac{T_D}{h_n} = n - k$$

Rewriting the equations in (5.33) and (5.34) as a function of the time step

$$(V_1)_n - Z_0 (I_1)_n = (V_2)_{n-k} + Z_0 (I_2)_{n-k}$$

$$(V_2)_n - Z_0 (I_2)_n = (V_1)_{n-k} + Z_0 (I_1)_{n-k}$$

As a function of the voltages at both nodes we can write these equations as

$$(V_1)_n^+ - (V_1)_n^- - Z_0 (I_1)_n = (V_2)_{n-k} + Z_0 (I_2)_{n-k}$$

$$(V_2)_n^+ - (V_2)_n^- - Z_0 (I_2)_n = (V_1)_{n-k} + Z_0 (I_1)_{n-k}$$

We can now write the equations for the transmission line in matrix form as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -Z_0 & 0 \\ 0 & 0 & 1 & -1 & 0 & -Z_0 \end{bmatrix} \begin{bmatrix} V_1^+ \\ V_1^- \\ V_2^+ \\ V_2^- \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ Z_0 (I_2)_{n-k} + (V_2)_{n-k} \\ Z_0 (I_1)_{n-k} + (V_1)_{n-k} \end{bmatrix} \quad (5.35)$$

5.3.7 Mutual Inductance

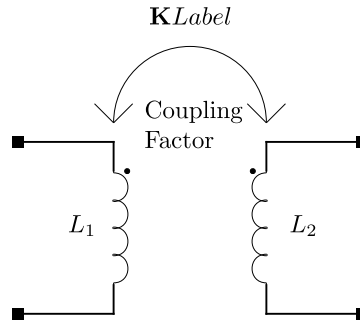


Figure 5.20: A basic mutual inductance element

Mutual inductance in JoSIM is defined as

KLabel **LLabel1** **LLabel2** **Coupling Factor**

Where the inductances being coupled will now have a revised equation

$$v_1(t) = L_1 \frac{di_1(t)}{dt} + M \frac{di_2(t)}{dt} \quad (5.36)$$

$$v_2(t) = M \frac{di_1(t)}{dt} + L_2 \frac{di_2(t)}{dt} \quad (5.37)$$

The coupling factor (k) determines the mutual coupling (M) through the equation with $-1 \leq k \leq 1$

$$M = k\sqrt{L_1 L_2} \quad (5.38)$$

By rewriting (5.36) and (5.37) in terms of the current time step we find

$$\begin{aligned} (V_1)_n &= \frac{2L_1}{h_n} (I_1)_n - \frac{2L_1}{h_n} (I_1)_{n-1} + \frac{2M}{h_n} (I_2)_n - \frac{2M}{h_n} (I_2)_{n-1} - (V_1)_{n-1} \\ (V_1)_n - \frac{2L_1}{h_n} (I_1)_n - \frac{2M}{h_n} (I_2)_n &= -\frac{2L_1}{h_n} (I_1)_{n-1} - \frac{2M}{h_n} (I_2)_{n-1} - (V_1)_{n-1} \\ (V_1^+)_n - (V_1^-)_n - \frac{2L_1}{h_n} (I_1)_n - \frac{2M}{h_n} (I_2)_n &= -\frac{2L_1}{h_n} (I_1)_{n-1} - \frac{2M}{h_n} (I_2)_{n-1} - (V_1)_{n-1} \end{aligned} \quad (5.39)$$

and

$$\begin{aligned} (V_2)_n &= \frac{2M}{h_n} (I_1)_n - \frac{2M}{h_n} (I_1)_{n-1} + \frac{2L_2}{h_n} (I_2)_n - \frac{2L_2}{h_n} (I_2)_{n-1} - (V_2)_{n-1} \\ (V_2)_n - \frac{2M}{h_n} (I_1)_n - \frac{2L_2}{h_n} (I_2)_n &= -\frac{2M}{h_n} (I_1)_{n-1} - \frac{2L_2}{h_n} (I_2)_{n-1} - (V_2)_{n-1} \\ (V_2^+)_n - (V_2^-)_n - \frac{2M}{h_n} (I_1)_n - \frac{2L_2}{h_n} (I_2)_n &= -\frac{2M}{h_n} (I_1)_{n-1} - \frac{2L_2}{h_n} (I_2)_{n-1} - (V_2)_{n-1} \end{aligned} \quad (5.40)$$

When collating (5.39) and (5.40) into one matrix we arrive at the MNA stamp

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -\frac{2L_1}{h_n} & -\frac{2M}{h_n} \\ 0 & 0 & 1 & -1 & -\frac{2M}{h_n} & -\frac{2L_2}{h_n} \end{bmatrix} \begin{bmatrix} (V_1^+)_n \\ (V_1^-)_n \\ (V_2^+)_n \\ (V_2^-)_n \\ (I_1)_n \\ (I_2)_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -\frac{2L_1}{h_n} (I_1)_{n-1} - \frac{2M}{h_n} (I_2)_{n-1} - (V_1)_{n-1} \\ -\frac{2M}{h_n} (I_1)_{n-1} - \frac{2L_2}{h_n} (I_2)_{n-1} - (V_2)_{n-1} \end{bmatrix} \quad (5.41)$$

5.4 Control Commands

The commands used to control JoSIM are placed within the netlist and usually start with a single period followed by a command. These commands include the type of simulation to be done, any models required by components and all the plotting commands.

The most basic of these commands, and a requirement for the simulator to do anything until other simulation methods are implemented is a transient analysis. A transient analysis in JoSIM is defined using the command

```
.tran T.Step T.Stop [T.Start]
```

which will perform a transient analysis that runs from time start (T.Start) to time stop (T.Stop) with the time step value of time step (T.Step). The number of simulation time steps would then be

$$n = \frac{T.Stop - T.Start}{T.Step} \quad (5.42)$$

where T.Start has a value of 0 unless specified. In the case that n is not an integer number, the amount of simulations to be run will be rounded down due to the nature of casting a double to an integer in C++.

The model specifier for JoSIM is discussed in Section 5.3.4.1, as the only model parameter JoSIM accepts at present is the Josephson junction model. Additional models will become available as the software matures.

The plotting commands used by JoSIM can take any of the following forms

```
.print Pr.Type Device or Node
.plot Pl.Type(Node or device)0 ... Pl.Type(Node or device)n
```

These commands both essentially do the same thing, however, the way the commands are formatted differs slightly. In the *print* command each separate print needs to be stated on a new line whereas the *plot* command can have multiple commands to plot in one line.

The *Pr.Type* can be either NODEV, DEVV, DEVI or PHASE. *NODEV* requests the node voltage for the specified node. *DEVV* and *DEVI* request the device voltage and current for the specified device. The voltage is defined as the difference between the two nodes the device is connected to and the current would be this difference divided by the device impedance. *PHASE* would simply print out the phase value (ϕ) for the specified junction.

When performing phase-based analysis the *PHASE Pr.Type* can be used for any component and an additional *Pr.Type NODEP* is introduced that allows plotting the nodal phase.

The *Pl.Type* can be any of the V, I or P. Where when *V* is the voltage across the device, but nodes could also be specified separated by a comma to find the voltage between two nodes. The *I* and *P* produce the device current and phase respectively.

Any of the control commands listed prior can be positioned within a control block of which there can only be one per circuit netlist. These commands would be enclosed in a section that starts with a *.control* command and ends with a *.endc* command. The lines enclosed in such a block would not require the a period at the start to indicate that it is a control command.

To specify whether the enclosed lines are part of a subcircuit the *.subckt* and *.ends* commands are used.

5.4.1 Parameters

A unique feature which is implemented in JoSIM is the expression parser which allows the user to set variables and do calculations within the circuit netlist. The definition of an expression within JoSIM is

.param Var.Name = Expression

The expression parser implements a variant of the Dijkstra's shunting yard algorithm[62] whereby the expression to be parsed is read in and converted to a reverse polish notation (RPN) stack which is then evaluated to return a value.

An example of this would be

$$R01 = 5E-3 * 1000 * \sin(PI/2) + 5$$

Table 5.1: Shunting yard conversion to a RPN stack

Token	RPN Stack	Operator Stack	Note
5E-3	5E-3		5E-3 is a value
*	5E-3	*	* is a high precedence operator
1000	5E-3 1000	*	1000 is a value
*	5E-3 1000 *	*	* has equal precedence to *
sin	5E-3 1000 *	sin *	sin has higher precedence than *
(5E-3 1000 * PI	(sin *	(is of the highest precedence
PI	5E-3 1000 PI	(sin *	PI is a value
/	5E-3 1000 * PI	/ (sin *	/ is within the bracket
2	5E-3 1000 * PI 2	/ (sin *	2 is a value
)	5E-3 1000 * PI 2 / sin *) found, pop stack
+	5E-3 1000 * PI 2 / sin *	+	+ has low precedence
5	5E-3 1000 * PI 2 / sin * 5	+	5 is a value
	5E-3 1000 * PI 2 / sin * 5 +		no more tokens, pop stack

The table in 5.1 depicts the algorithm steps in a very simplified manner. It does however show that certain operators have precedence over others and if operators of the same importance are found the operator is pushed to the RPN stack. Once all the tokens have been exhausted the RPN stack can then be evaluated using an algorithm that applies the operator to the preceding stack value(s) until there is only one item left in the stack

Table 5.2: RPN stack evaluation

RPN Stack	Expression Evaluation	Note
5E-3 1000 * PI 2 / sin * 5 +	5E-3 1000 *	
5 PI 2 / sin * 5 +	5 PI	No operator found
5 PI 2 / sin * 5 +	PI 2 /	
5 PI/2 sin * 5 +	5 PI/2 Sin	Sin takes one operator
5 PI/2 sin * 5 +	PI/2 Sin	
5 1 * 5 +	5 1 *	
5 5 +	5 5 +	
10		

This value is then stored using the *R01* label as key. This value can then be used within the netlist in various ways such as component values, model definitions or even in other parameter values. This is particularly useful when some form of scaling needs to be applied to values, since changing a single value then changes all values. Parameters can be unique to a subcircuit or defined in main netlist making them global and accessible within subcircuits. If a parameter

makes use of another parameter within the expression that needs parsing, the required parameter needs to be defined prior to the evaluation due to the sequential read in of files within JoSIM. The sequential read in and caviates related to that are discussed in Section 5.5.

The types of expressions that can be evaluated at present are only limited to very basic algebra and trigonometric functions, however this can be expanded to encompass any possible function.

The *param* control can also be enclosed within the *control* block whereby the period can be omitted.

5.5 Chicken and Egg

During the development of JoSIM and the process used to read in a netlist various cases arise which we call a ‘*chicken and egg*’ situation. The name derives from the age old question ‘*What came first? The chicken or the egg?*’[63] and is so named because these situations often required parts of the netlist which had not been read in at that point which results in causality dilemmas.

The most apparent of these dilemmas is that of subcircuit definitions. Where the subcircuit used within the main netlist does not exist at the time of read in. We combat this by first reading in the entire netlist, splitting it into a main part and subcircuit sections, and finally expanding the main part into a master netlist using all the relevant subcircuits sections. This also solves the problem when subcircuits are used within subcircuits.

Another situation which only became apparent during implementation was that of mutual inductance, where the inductors being coupled need to be defined before they can be mutually coupled. It would not be good practice to force this requirement onto the user and we therefore solve it in a similar way to the previous dilemma. We create the entire \mathbf{A} matrix by looping through the master netlist, thereby defining all the inductors and only take note of mutual coupling. Once through the master netlist we then iterate through the mutual inductances, if any, and apply them to the matrix separately. This dilemma will make a return when discussing mutual inductance in Section 5.6.

The solutions decided upon do impact the performance negatively, however the alternative would require shifting the responsibility to the user which would not be good practice. The key to a successful design is to give the user exactly what they want while requiring as little possible effort from their side.

5.6 Phase Simulation

The JJ is largely a phase-based element, and the direct calculation of the phase would thus be more sensible. With phase mode, we present a direct relation between voltage and phase which can be substituted into any voltage dependent equation. The act of derivation implies extraction of information from a source and thus retaining data that is less informative than the original. This can be seen through the constant derivation of voltage to obtain phase error accumulation.

It therefore becomes necessary to do the entire simulation in phase, since each component affects the phase of the entire circuit. We already have a well established MNA system that handles calculation of voltage and current in a circuit, and it would be practical to simply adapt this method to calculate phase.

This process is started by substituting the phase-voltage relation (5.44) into every component equation and reducing it to find a modified nodal phase analysis (MNPA) matrix.

$$\phi = \frac{2\pi}{\Phi_0} \int v dt \quad (5.43)$$

$$v = \frac{\Phi_0}{2\pi} \frac{d\phi}{dt} \quad (5.44)$$

The standard voltage-based equations for basic analogue circuit components are presented as

$$v = L \frac{di}{dt} \quad (5.45)$$

$$i = C \frac{dv}{dt} \quad (5.46)$$

$$i = \frac{v}{R} \quad (5.47)$$

We now substitute (5.44) into each of (5.45), (5.46) and (5.47) respectively to find the phase-based equations and eventually the MNPA stamps. These MNPA stamps are obtained using the trapezoidal integration method.

$$\left(\frac{dx}{dt} \right)_n = \frac{2}{h_n} (x_n - x_{n-1}) - \left(\frac{dx}{dt} \right)_{n-1} \quad (5.48)$$

5.6.1 Phase Inductor

When (5.44) is substituted into (5.45) and we apply (5.2), we find

$$\frac{\Phi_0}{2\pi} \frac{d\phi}{dt} = L \frac{di}{dt} \quad (5.49)$$

Performing integration on both sides and assuming initial values as 0

$$\begin{aligned} \frac{\Phi_0}{2\pi} \phi &= LI_n \\ \phi^+ - \phi^- - \frac{2\pi L}{\Phi_0} I_n &= 0 \end{aligned} \quad (5.50)$$

We are then able to create the MNPA stamp as

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{2\pi L}{\Phi_0} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (5.51)$$

5.6.2 Phase Capacitor

When (5.44) is substituted into (5.46) and we apply (5.2), we find

$$i = \frac{C\Phi_0}{2\pi} \frac{d^2\phi}{dt^2} \quad (5.52)$$

$$\begin{aligned} I_n &= \frac{C\Phi_0}{2\pi} \left[\frac{2}{h_n} \left(\frac{d\phi}{dt}_n - \frac{d\phi}{dt}_{n-1} \right) - \left(\frac{d^2\phi}{dt^2} \right)_{n-1} \right] \\ I_n &= \frac{C\Phi_0}{\pi h_n} \left(\frac{d\phi}{dt}_n - \frac{d\phi}{dt}_{n-1} \right) - I_{n-1} \\ I_n &= \frac{2C\Phi_0}{\pi h_n^2} \phi_n - \frac{2C\Phi_0}{\pi h_n^2} \phi_{n-1} - \frac{2C\Phi_0}{\pi h_n} \dot{\phi}_{n-1} - I_{n-1} \\ \frac{\pi h_n^2}{2C\Phi_0} I_n - \phi_n &= -\phi_{n-1} - h_n \dot{\phi}_{n-1} - \frac{\pi h_n^2}{2C\Phi_0} I_{n-1} \end{aligned}$$

$$\begin{aligned}\phi_n - \frac{\pi h_n^2}{2C\Phi_0} I_n &= \phi_{n-1} + h_n \dot{\phi}_{n-1} + \frac{\pi h_n^2}{2C\Phi_0} I_{n-1} \\ \phi_n^+ - \phi_n^- - \frac{\pi h_n^2}{2C\Phi_0} I_n &= \phi_{n-1} + h_n \dot{\phi}_{n-1} + \frac{\pi h_n^2}{2C\Phi_0} I_{n-1}\end{aligned}\quad (5.53)$$

Which leaves us with a MNPA stamp

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{\pi h_n^2}{2C\Phi_0} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi h_n^2}{2C\Phi_0} I_{n-1} + \phi_{n-1} + h_n \dot{\phi}_{n-1} \end{bmatrix}\quad (5.54)$$

We use the trapezoidal method to calculate the derivative of phase, which is done at the end of every time step for the next time step.

$$\dot{\phi}_{n-1} = \frac{2}{h_n} (\phi_{n-1} - \phi_{n-2}) - \dot{\phi}_{n-2}\quad (5.55)$$

5.6.3 Phase Resistor

When (5.44) is substituted into (5.47) and we apply (5.2), we find

$$i = \frac{\Phi_0}{2\pi R} \frac{d\phi}{dt}\quad (5.56)$$

$$\begin{aligned}I_n &= \frac{\Phi_0}{2\pi R} \left[\frac{2}{h_n} (\phi_n - \phi_{n-1}) - \left(\frac{d\phi}{dt} \right)_{n-1} \right] \\ I_n &= \frac{\Phi_0}{\pi h_n R} \phi_n - \frac{\Phi_0}{\pi h_n R} \phi_{n-1} - I_{n-1} \\ \phi_n^+ - \phi_n^- - \frac{\pi h_n R}{\Phi_0} I_n &= \frac{\pi h_n R}{\Phi_0} I_{n-1} + \phi_{n-1}\end{aligned}\quad (5.57)$$

which leaves us with a MNPA stamp

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{\pi h_n R}{\Phi_0} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi h_n R}{\Phi_0} I_{n-1} + \phi_{n-1} \end{bmatrix}\quad (5.58)$$

5.6.4 Phase JJ

The JJ in phase remains the same since the phase was already calculated in the the voltage-based method. The voltage now becomes a virtual node only required to estimate the phase for the next time step and to identify which mode of operation the junction will be in.

The MNPA stamp for the JJ can be written as

$$\begin{bmatrix} 0 & 0 & \frac{2C}{h_n} + \frac{1}{R} \\ 0 & 0 & -\frac{2C}{h_n} - \frac{1}{R} \\ 1 & -1 & -\frac{h_n}{2} \frac{2e}{\hbar} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ V \end{bmatrix} = \begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} V_{n-1} \end{bmatrix}\quad (5.59)$$

where the RHS current is defined as

$$I_s = -I_c \sin \phi_n^0 + \frac{2C}{h_n} V_{n-1} + C \dot{V}_{n-1}\quad (5.60)$$

Since this method is equivalent to the one used in the voltage-based method, we retain the same equations for the voltage and phase guess as in (5.19) and (5.20).

5.6.5 Phase Lossless Transmission Line

A lossless transmission line has the voltage base equations

$$v_1 - Z_0 i_1 = Z_0 i_2(t - T_D) + v_2 \quad (5.61)$$

$$v_2 - Z_0 i_2 = Z_0 i_1(t - T_D) + v_1 \quad (5.62)$$

When 5.61 and 5.62 are represented in terms of phase they become

$$\begin{aligned} \frac{\Phi_0}{2\pi} \dot{\phi}_{1n} - Z_0 I_{1n} &= Z_0 I_{2n-k} + \frac{\Phi_0}{2\pi} \dot{\phi}_{2n-k} \\ \frac{\Phi_0}{2\pi} \dot{\phi}_{2n} - Z_0 I_{2n} &= Z_0 I_{1n-k} + \frac{\Phi_0}{2\pi} \dot{\phi}_{1n-k} \end{aligned}$$

which we can then write in terms of the current time step as

$$\phi_{1n}^+ - \phi_{1n}^- - \frac{\pi h_n Z_0}{\Phi_0} I_{1n} = \frac{\pi h_n Z_0}{\Phi_0} I_{2n-k} + \phi_{1n-1} + \frac{h_n}{2} (\dot{\phi}_{1n-1} + \dot{\phi}_{2n-k}) \quad (5.63)$$

$$\phi_{2n}^+ - \phi_{2n}^- - \frac{\pi h_n Z_0}{\Phi_0} I_{2n} = \frac{\pi h_n Z_0}{\Phi_0} I_{1n-k} + \phi_{2n-1} + \frac{h_n}{2} (\dot{\phi}_{2n-1} + \dot{\phi}_{1n-k}) \quad (5.64)$$

We are now able to write the equations in (5.63) and (5.64) in matrix form as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -\frac{\pi h_n Z_0}{\Phi_0} & 0 \\ 0 & 0 & 1 & -1 & 0 & -\frac{\pi h_n Z_0}{\Phi_0} \end{bmatrix} \begin{bmatrix} \phi_1^+ \\ \phi_1^- \\ \phi_2^+ \\ \phi_2^- \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ VT1 \\ VT2 \end{bmatrix} \quad (5.65)$$

where $VT1$ and $VT2$ are the RHS of equations (5.63) and (5.64) respectively.

This implementation of the lossless transmission line requires that the differential for each phase node be calculated for each time step.

5.6.6 Phase Mutual Inductance

If we substitute the phase equation into that of the mutual inductance we find

$$\frac{\Phi_0}{2\pi} \frac{d\phi_1}{dt} = L_1 \frac{di_1}{dt} + M \frac{di_2}{dt} \quad (5.66)$$

$$\frac{\Phi_0}{2\pi} \frac{d\phi_2}{dt} = M \frac{di_1}{dt} + L_2 \frac{di_2}{dt} \quad (5.67)$$

to which we can apply integration on both sides and assume initial conditions.

$$\frac{\Phi_0}{2\pi} \phi_1^+ - \frac{\Phi_0}{2\pi} \phi_1^- = L_1 I_1 + M I_2 \quad (5.68)$$

$$\frac{\Phi_0}{2\pi} \phi_2^+ - \frac{\Phi_0}{2\pi} \phi_2^- = M I_1 + L_2 I_2 \quad (5.69)$$

(5.68) and (5.69) can then be written in matrix form as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -\frac{2\pi L_1}{\Phi_0} & -\frac{2\pi M}{\Phi_0} \\ 0 & 0 & 1 & -1 & -\frac{2\pi M}{\Phi_0} & -\frac{2\pi L_2}{\Phi_0} \end{bmatrix} \begin{bmatrix} \phi_1^+ \\ \phi_1^- \\ \phi_2^+ \\ \phi_2^- \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (5.70)$$

5.6.7 Phase Voltage Source

A voltage source can also be transformed into a phase based voltage source using the direct transformation

$$v = \frac{\Phi_0}{2\pi} \frac{d\phi}{dt} \quad (5.71)$$

$$V_n = \frac{\Phi_0}{2\pi} \left[\frac{2}{h_n} (\phi_n - \phi_{n-1}) - \left(\frac{d\phi}{dt} \right)_{n-1} \right] \quad (5.72)$$

$$\phi_n^+ - \phi_n^- = \phi_{n-1} + \frac{\pi h_n}{\Phi_0} (V_n + V_{n-1}) \quad (5.73)$$

When we write (5.73) in matrix form it becomes

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi h_n}{\Phi_0} (V_n + V_{n-1}) + \phi_{n-1} \end{bmatrix} \quad (5.74)$$

5.6.8 Phase Source

A phase source in JoSIM is defined as

PLabel Pos.Node Neg.Node Value

The purpose of the phase source is to apply a constant phase at the specified nodes. This source type is only active when performing a phase mode analysis and will present the user with an unknown component when attempting to use this in voltage mode.

In matrix form a phase source is written as

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \phi_n \end{bmatrix} \quad (5.75)$$

5.7 Conclusion

JoSIM was developed as a SPICE syntax circuit simulator capable of simulating the superconducting effects of the Josephson junction. A netlist is fed to JoSIM as input and read in sequentially. Once read in the netlist is parsed into a matrix, and the simulation controls are identified. The controls define the actions that need to be executed on the matrix. The matrix is solved using the KLU linear algebra library and the solution is presented to the user as output in the manner defined by the controls.

JoSIM utilizes a voltage and phase based MNA approach to create the matrix and we therefore discussed the circuit elements available in JoSIM including how they fit into the MNA matrix. The integration methods used to approximate non-linear elements were discussed. However, other methods that are less prone to error will be investigated and possibly provided as optional to the user in future versions of the simulator.

The control commands in JoSIM were discussed and further expansion of simulation methods such as fixed point DC and frequency sweep AC will be investigated for implementation in future versions.

Chapter 6

JoSIM - Results

6.1 Introduction

In this chapter we will discuss the results of simulations performed using JoSIM. We divide this chapter into three parts namely the IV curve, small simulations and medium to large simulations. We compare the results from each simulation to results produced by other simulators, where possible, to critically analyse the success of JoSIM.

In the small simulations section we simulate a basic Josephson transmission line (JTL). We compare simulation speed, accuracy as well as the ease of visually representing the results.

In the medium to large simulations section we simulate designs that use multiple subcircuits, as well as designs that include nested subcircuits. These simulations are quite computationally intense and the room for error to occur becomes larger. We analyse execution speed, accuracy and resource intensity compared to existing simulators.

6.2 IV Curve

To evaluate the performance and accuracy of JoSIM as a superconducting circuit simulator we run it through the same testbench used to calculate the IV curves in Section 4.6. The results of this testbench can be seen in Figure 6.1 and 6.2.

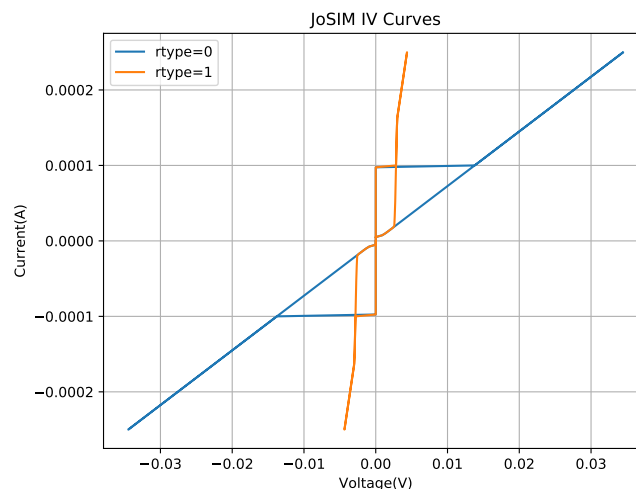


Figure 6.1: JoSIM IV curves

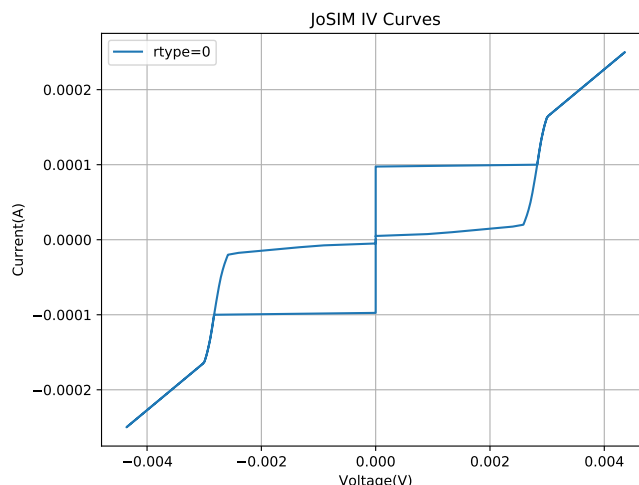


Figure 6.2: JoSIM IV curve of only rtype=1

The IV curves generated by JoSIM are clearly in line with the theoretical expectation of the RCSJ. The voltage remains zero until the current reaches the junction critical current value whereafter it changes to the the gap voltage and when pushed further it approaches a linear zone. When the current is brought back down, the voltage follows the same linear trend until it reaches the gap voltage where the current drops sharply to a value close to zero.

If we now compare these results to that of JSIM and WRspice we can see through Figure 6.3 that the junction implementation closely matches that of WRspice. This is expected due to the same procedure for implementation being followed as in [50].

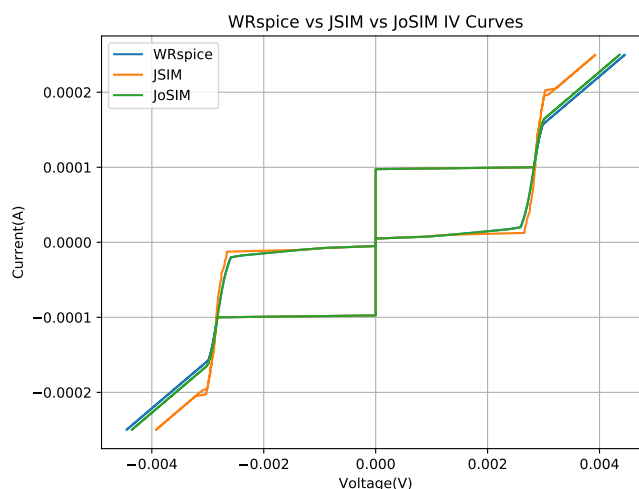


Figure 6.3: IV curves of JSIM, JoSIM and WRspice

JoSIM completes the testbench of 400 simulations and post-processing in 57.4s, which is almost identical to that of JSIM seen in Section 4.6. The improvement hereof is discussed in Chapter 7.

6.3 Small Simulations

We now simulate a very basic example to test that the modified nodal voltage analysis method implemented in JoSIM functions as expected and compare these results along with execution times to that of JSIM and WRspice. The circuit we will use to conduct this small scale test is the JTL, of which the schematic can be seen in Figure 6.4. This simulation tests the capability

of JoSIM to handle resistors, inductors, JJs and both types of input sources. It additionally tests the ability to handle devices in parallel as can be seen with the junctions being shunted by resistors in parallel.

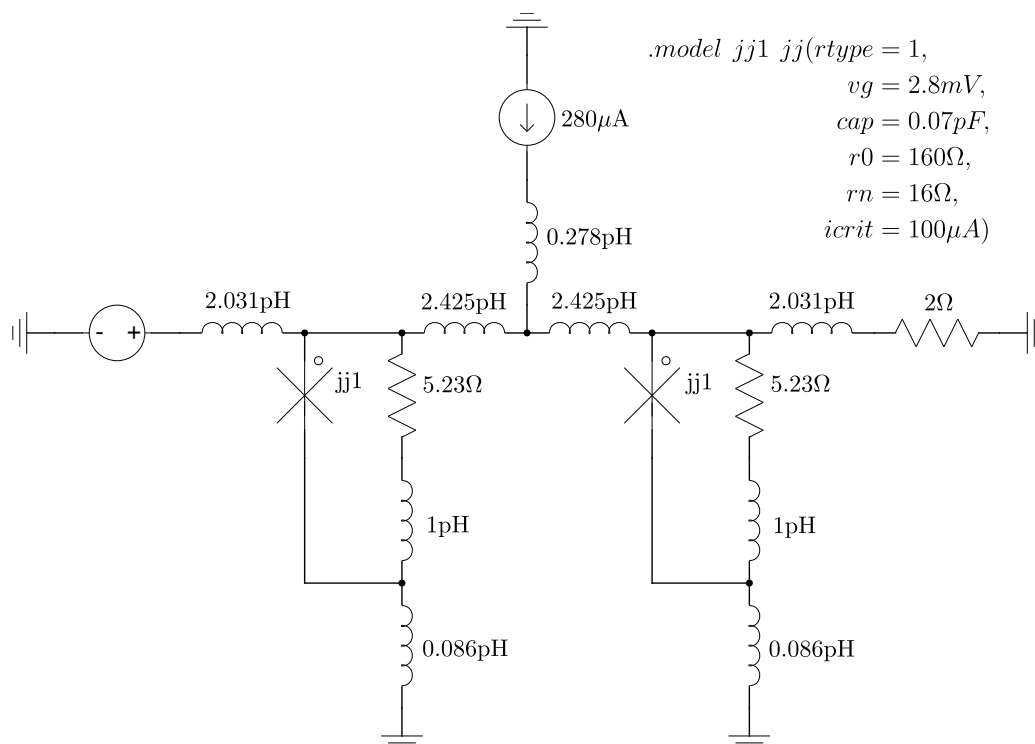


Figure 6.4: Basic JTL used to test small simulations

We excite the circuit with a SFQ pulse at 300ps and 600ps by providing a PWL voltage representation of an SFQ pulse. We set the simulation controls to perform a transient analysis of 0.1ps step size and simulation time of 1ns. Additionally we request that the voltage across the input source, current through the output resistor and phases of both JJs be plotted. The netlist representation of circuit in Figure 6.4 is run through JoSIM with the following command

```
JoSIM -g -m basic_jtl.dat basic_jtl.js
```

This presents us with the Python Matplotlib result window depicted in Figure 6.5. The same circuit is run through JSIM as well as WRspice. JoSIM completes execution in 0.071s, where JSIM does in 0.06s and WRspice in 0.219s. We now compare the results of all 3 simulators on one graph in Figure 6.6. With the execution time somewhere between JSIM and WRspice, while still being heavily unoptimized can be considered not bad for the first attempt at creating a superconducting circuit simulation engine.

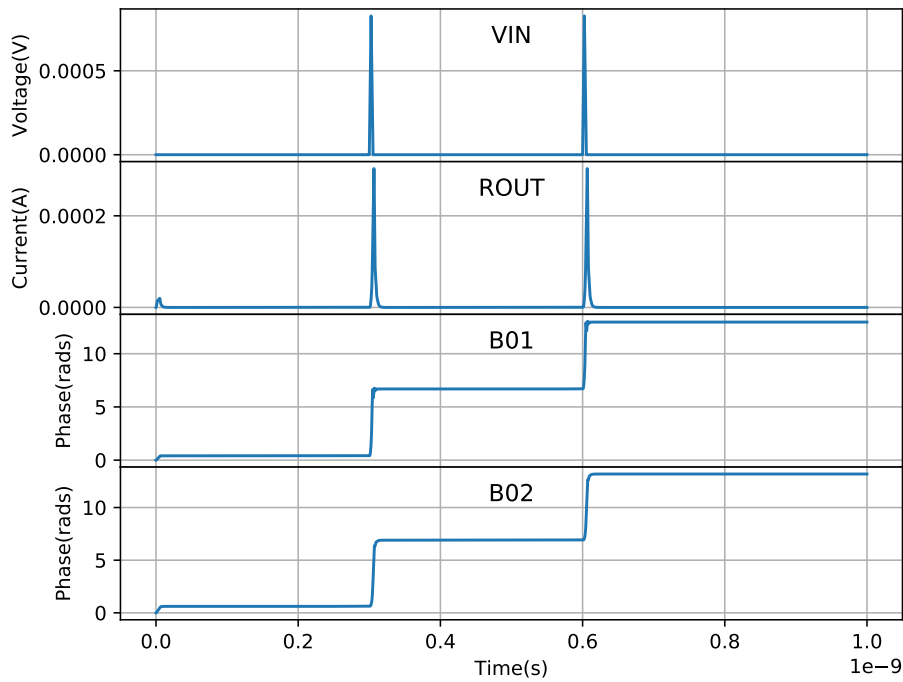


Figure 6.5: Results of the JTL simulation performed with JoSIM

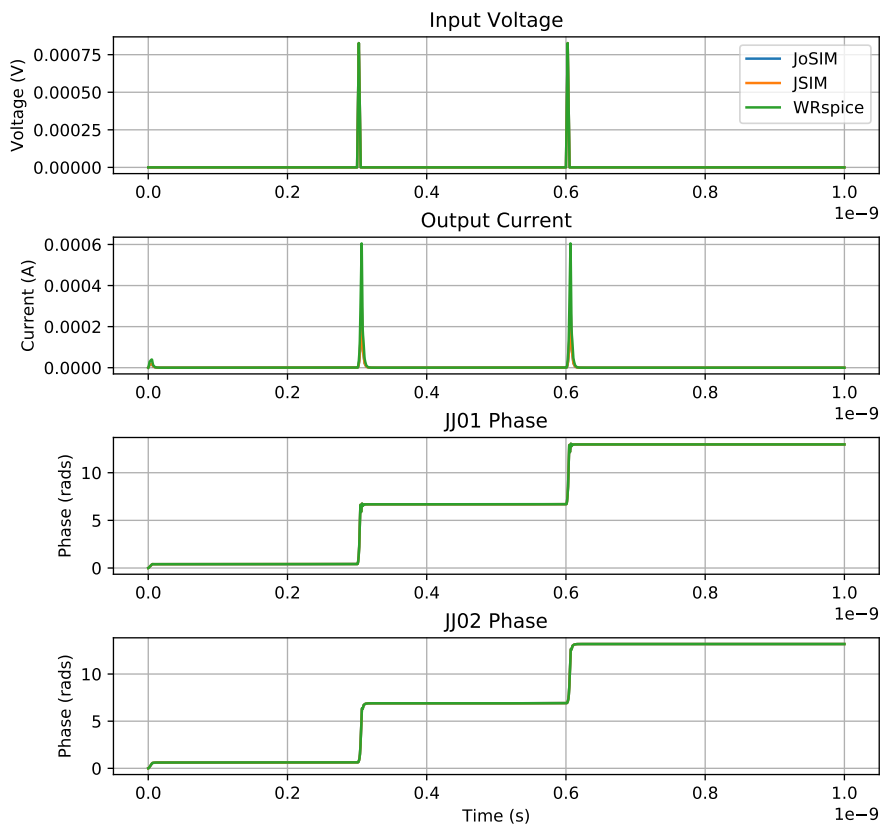


Figure 6.6: Comparison of the results for the JTL between JSIM, JoSIM and WRspice

The results in Figure 6.6 verify the correct operation of JoSIM through a practical example. The difference is, however, nearly impossible to notice and we therefore plot the difference between the signals as an error percentage on a log scale. This comparison is done on the phase of the first JJ and can be seen in Figure 6.7. The slight deviations seen between the error percentages of the simulators can be narrowed down to the variation in the implementation of

the $R_{type}=1$ resistance model seen in Figure 6.3. This difference is fairly small and can be considered negligible as it does not affect the accuracy of the results.

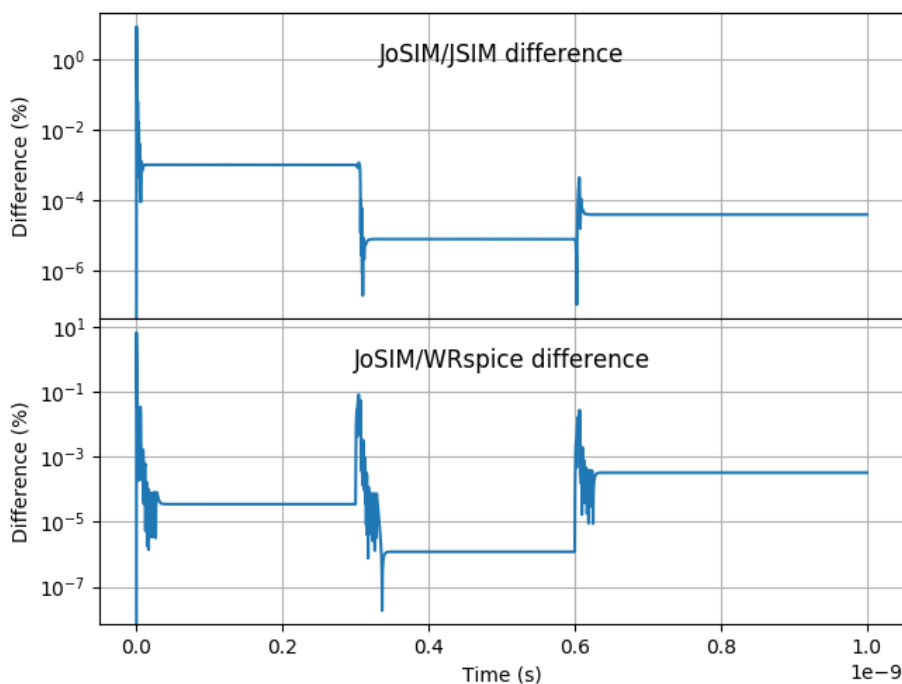


Figure 6.7: Error percentage of JoSIM compared to JSIM and WRspice

Small scale examples are however not a particularly good method of determining performance metrics due to the KLU algorithm claiming performance bonus only on heavily sparse matrices. We therefore need to experiment with larger simulations that produce matrices of greater sparsity to truly test the performance of JoSIM.

6.4 Medium to Large Scale Simulations

We consider medium simulations as simulations that range between a 1000 and 5000 JJs, with large scale designs being anything over 5000 JJs. To test JoSIM on medium to large scale designs, we simulate designs that include multiple subcircuits as well as nested subcircuits as this is the simplest way to increase complexity and matrix sparsity.

For the medium example we choose a design that calculates the partial products of an 8-bit multiplier created by N. Muchuka and R. Bakolo in 2016. This simulation consists of 2896 JJs and a total of 18392 components. This equates to an \mathbf{A} matrix of 36868×36868 , with only 112386 of the 1359249424 entries filled. This is an extremely sparse matrix which is only 0.00826% filled. JSIM completes this simulation in 1m33s. JoSIM completes this simulation in 34s. WRspice completes this simulation in 1m04s. This clearly demonstrates the power of the KLU algorithm when handling very sparse matrices as well as the advantage presented by JoSIM over WRspice and JSIM.

The operation of the partial products generating circuit is shown in Table 6.1 where each bit of B is multiplied with every bit of A individually using the logic *AND* operation. This creates the 8 partial product rows of which each column is then summed using an 8-bit full adders to generate the product. The multiplication of two 8-bit binary numbers results in a 16-bit product as seen in Table 6.1.

We demonstrate the output of the 7th partial product through simulation results shown in Figure 6.8.

Table 6.1: General binary partial product generation

									1	1	1	0	1	1	1	1	A
								×	1	1	1	1	1	1	1	1	B
									1	1	1	0	1	1	1	1	PP_0
									1	1	1	0	1	1	1	1	PP_1
					1	1	1	0	1	1	1	1	1	1			PP_2
				1	1	1	0	1	1	1	1	1					PP_3
			1	1	1	0	1	1	1	1	1						PP_4
		1	1	1	0	1	1	1	1	1							PP_5
	1	1	1	0	1	1	1	1	1								PP_6
+	1	1	1	0	1	1	1	1	1								PP_7
	P_{15}	P_{14}	P_{13}	P_{12}	P_{11}	P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	

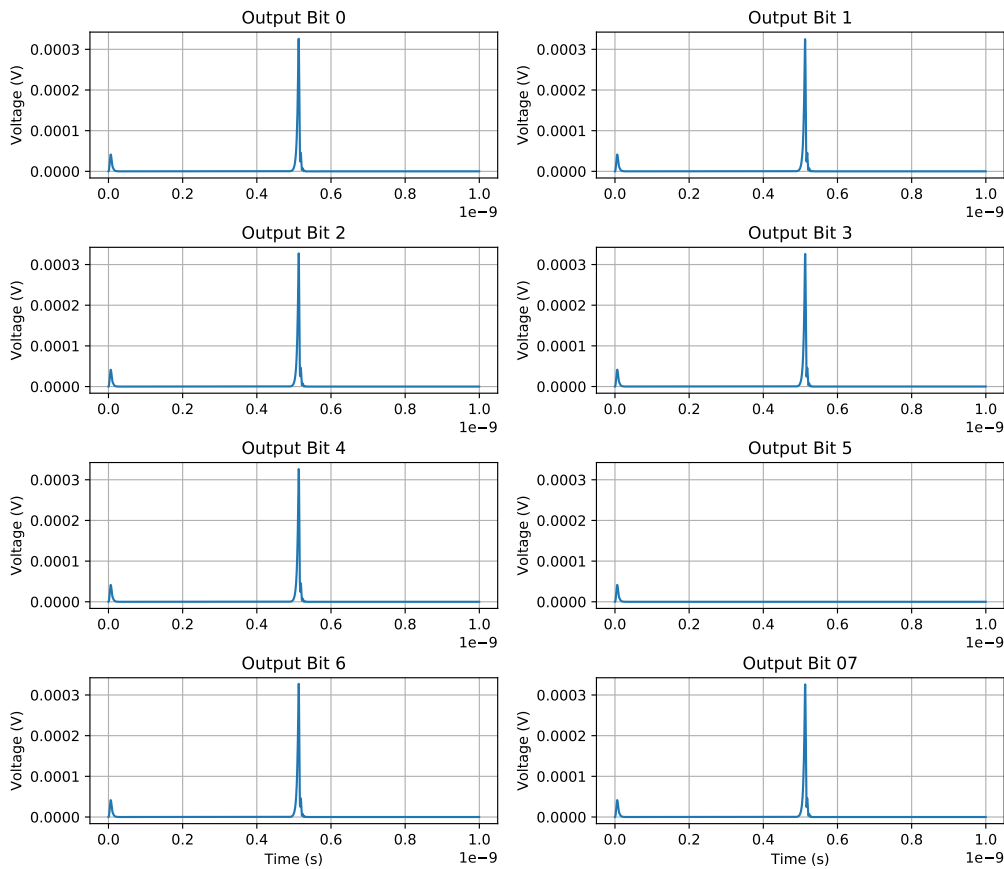


Figure 6.8: Results of the 7th partial product

As an additional example we simulate a 4-bit Kogge Stone added generated through attempts by the University of South California to create a synthesis tool. This example takes as input 2 sets of 4 bit inputs in parallel along with carry in bit. The bits are added using adder logic and an output of 4 bits is produced a long with a carry out bit. The entire design is and the interconnects between gates are modelled using transmission lines of variable time delay. JoSIM completes this simulation in 26.69s. JSIM completes the same simulation in 1m13.88s which is almost 3 times slower than JoSIM. WRspice did not finish this simulation and crashed after 51m stating that a non convergence problem was detected

The 2 4-bit inputs shown in Table 6.2 are used as input along with the a 0 carry in bit. The 4-bit output is plotted along with the carry out. These results can be seen in Figure 6.9 and verified as correct when compared to the expected result. These results also correspond with

Table 6.2: 4-bit binary addition

	1	1	1	1	A
	1	1	1	1	B
+				0	C_{IN}
	1	1	1	0	SUM
				1	C_{OUT}

the HDL simulation of the 4-bit KSA in Figure 3.2.

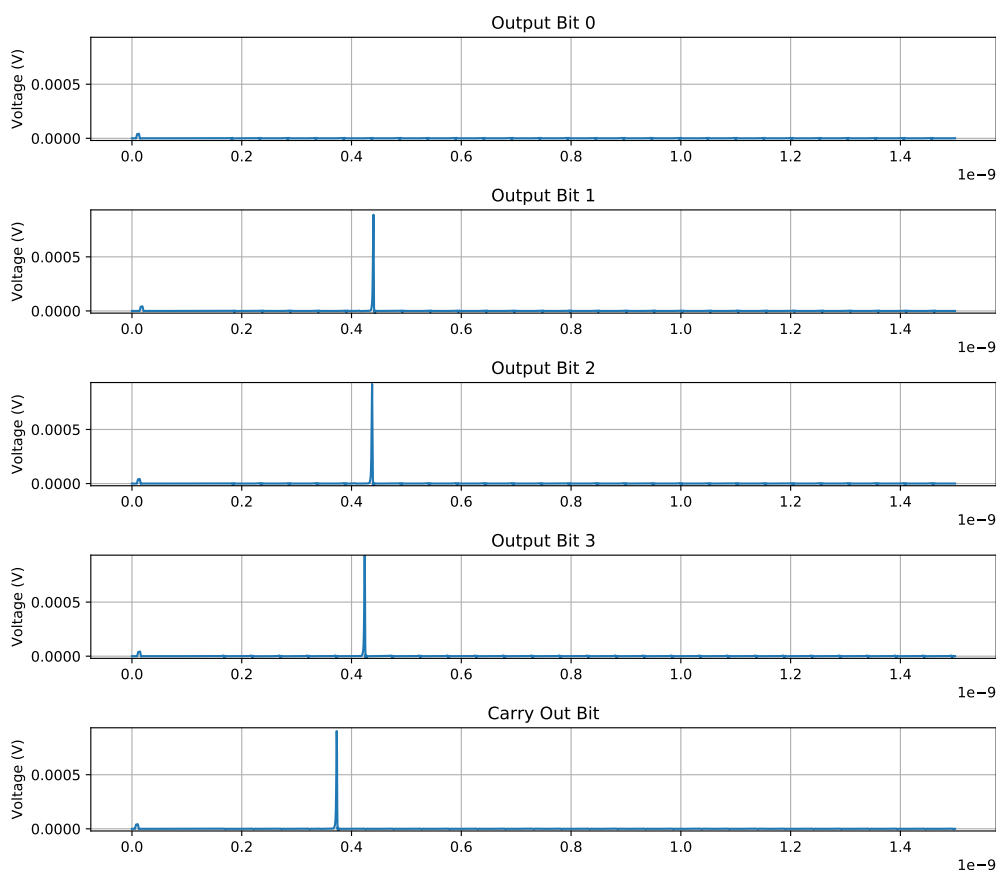


Figure 6.9: Output of the 4-bit KSA simulation

The implementation of a large scale simulation becomes quite time consuming and complex as it not only requires the design to be created in a schematic editor but also the verification of the correct operation results. We therefore artificially create a large simulation that simply strings together JTLs to boost the component and JJ count to a range where the simulation can be considered large. This operation becomes quite simple when utilizing the ability to create nested subcircuits. If we simply nest 10 JTLs in a subcircuit block called JTL10 and then further nest 10 of these in a block called JTL100 and so forth we can through nesting easily reach a simulation with 10 000 JJs. This nested netlist is shown in Table 6.3.

Table 6.3: Simple example of a large circuit

```

subckt JTLSTRING10 1 11      .subckt JTLSTRING1000 1 11
X01 JTL 1 2                  X01 JTLSTRING100 1 2
X02 JTL 2 3                  X02 JTLSTRING100 2 3
X03 JTL 3 4                  X03 JTLSTRING100 3 4
X04 JTL 4 5                  X04 JTLSTRING100 4 5
X05 JTL 5 6                  X05 JTLSTRING100 5 6
X06 JTL 6 7                  X06 JTLSTRING100 6 7
X07 JTL 7 8                  X07 JTLSTRING100 7 8
X08 JTL 8 9                  X08 JTLSTRING100 8 9
X09 JTL 9 10                 X09 JTLSTRING100 9 10
X10 JTL 10 11                X10 JTLSTRING100 10 11
.ends JTLSTRING10            .ends JTLSTRING1000

.subckt JTLSTRING100 1 11    IA 0 1 pwl(0 0 50p 0 56p 600u 62p 0)
X01 JTLSTRING10 1 2          X01 DCSFQ 1 2
X02 JTLSTRING10 2 3          X02 JTLSTRING1000 2 3
X03 JTLSTRING10 3 4          X03 JTLSTRING1000 3 4
X04 JTLSTRING10 4 5          X04 JTLSTRING1000 4 5
X05 JTLSTRING10 5 6          X05 SINK 5
X06 JTLSTRING10 6 7          .tran 0.25p 1000p 0 0.25p
X07 JTLSTRING10 7 8          .print nodev 1 0
X08 JTLSTRING10 8 9          .print nodev 2 0
X09 JTLSTRING10 9 10         .print nodev 3 0
X10 JTLSTRING10 10 11        .print nodev 4 0
.ends JTLSTRING100           .print nodev 5 0
                              .end

```

This example was run through JoSIM which completed it in 2m13s, which when run through WRspice and JSIM completes it in 2m39s and 4m36s respectively. To attempt to plot the results of the simulation will not provide much useful information as the output of the first JTLSTRING1000 lies outside of the bounds of the simulation time due to the delay caused by 1000 JTLs.

If we add one more 1000 JTL string to the simulation in Table 6.3 JSIM fails to converge and crashes after about 4m30s. JoSIM consumes a large amount of memory due to the way it was implemented and therefore starts to swap memory to the non-volatile memory of the system. We further discuss how to solve these problems in JoSIM in Chapter 7.

6.5 Conclusion

In this chapter we performed various simulations and compared JoSIM to existing simulators. We further demonstrated the speed improvement that JoSIM presents and contemplated the possibility of performing VLSI design simulations within reasonable time. This task requires further investigation and reconsideration of design implementation at the core of the JoSIM engine.

Comparisons were drawn between JoSIM, JSIM and WRspice in terms of execution speed and are tabulated in Table 6.4. We graph these results to attempt to extrapolate simulation times for larger circuits. This graph can be seen in Figure 6.10.

Table 6.4: Summary of execution times for various simulation sizes

Simulation	JJ count	Execution time (s)		
		JSIM	WRspice	JoSIM
Basic JTL	2	0.06	0.219	0.071
400 Simulation IVcurve	400	60.5	56	54.87
4-bit KSA	2095	73.9	<i>DNF</i>	23.49
General Partial Products	3904	93	64	20.9
3000 JTL String	6006	276	159	91.88
4000 JTL String	8006	>3600	232.7	130.87
5000 JTL String	10006	<i>DNF</i>	263.8	169.81

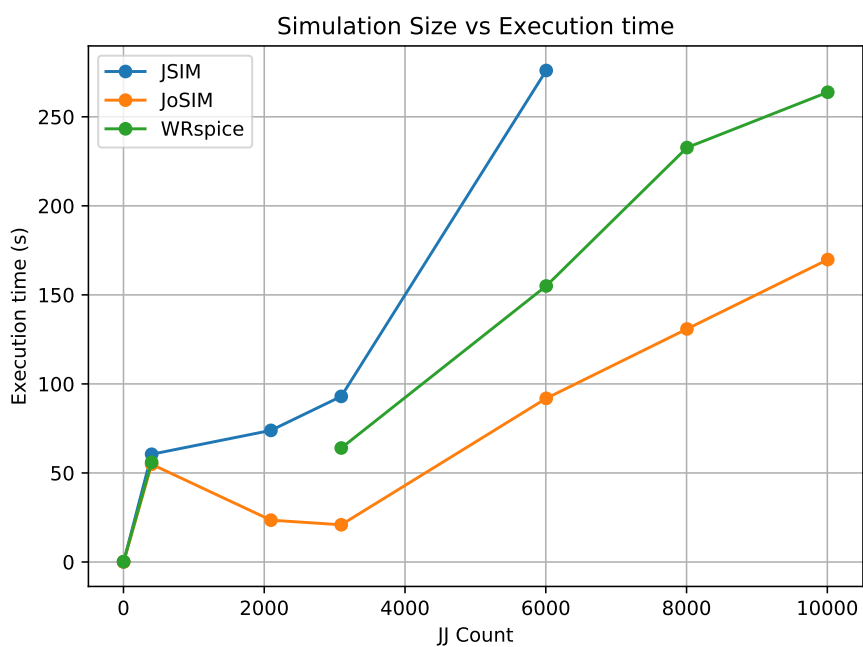


Figure 6.10: Execution time vs JJ count

Chapter 7

Very Large Scale Design Simulation

7.1 Introduction

JoSIM has shown significant performance improvements over previous superconducting circuit simulators, however due to the JoSIM's unoptimized implementation method it consumes large amounts of memory and quickly reaches the limit of available system memory. These issues, as well as experimentation with multi-threading and math engine optimizations, are discussed in this chapter.

Each of the possible bottlenecks that plague JoSIM are discussed in each of the various sections and possible solutions to solve these bottlenecks are presented. Implementation of these solutions, however, require large refactoring and rewriting efforts and are left to be done at a later stage.

7.2 Data Structure Considerations

The greatest bottleneck that JoSIM faces and also one of the many areas that will provide significant improvement in future releases is the way that data structures are implemented.

JoSIM reads in a netlist and stores each individual line as a *std::string* in a *std::vector*. Once all of these lines have been read, this vector of strings is iterated, identifying the different parts of the design such as main netlist, subcircuits and controls. Each of these parts are stored in a new *std::vector* of *std::strings*, essentially duplicating the initial vector, but as smaller vectors.

The main part is then iterated identifying any subcircuits that it might contain and substituting these subcircuits into the main netlist in a new vector of strings called the master netlist. Already the inefficiency of this method becomes apparent as multiple duplicates of the original netlist are created. These duplications can easily be reduced by rewriting the algorithm to alter the netlist in place instead.

This master netlist is essentially the main netlist with all of its subcircuits expanded. During the expansion of subcircuits, track needs to be kept of the interconnections, as well as the uniqueness of the device labels. Since JoSIM allows alphanumeric node numbers, this task can be simplified by simply appending the parent subcircuit call label to each of the labels and nodes of the subcircuit lines being substituted using some delimiter. This ensures the uniqueness of all node numbers and labels while also maintaining the ability to identify to which subcircuit the component belongs. When nesting of subcircuits occur, the appended label can become quite lengthy and since this is stored as a string the memory required to store this significantly increases.

The master netlist is then iterated, performing the relevant actions based on the type of component each line represents. Since the MNA solves nodal voltage/phase and branch current, each of the nodes create a unique row and column within the \mathbf{A} matrix. These nodes

are prepended with a $R_$ or $C_$ and also stored in a string vector. This is used to identify new matrix entries, since if a component is read and the row and column already exists for it, then the conductance of the component is simply added to the relevant matrix entry. Though, since nested subcircuits append the name of its parent to the node numbers as well, these row and column vectors can also consume quite a large amount of memory in large circuits.

A matrix element is created for each circuit element that stores the conductance value along with the row and column name as well as the row and column index. Once all of these elements are created, a full \mathbf{A} matrix is created of size $N \times N$, where N is the size of the row and column vectors.

This \mathbf{A} matrix can become extremely large even in smaller simulations since each of the N^2 matrix elements is allocated the size of a *double* which is 8 bytes even if it is empty (0.0). As an example, we look at the case of the general partial products simulation where $N=36868$. This means that $36868^2 \times 8 = 10873995392$ bytes or rather ≈ 10 GB of memory is required to simply store this colossal \mathbf{A} matrix. This is an extreme inefficiency and needs to be rectified.

Once this \mathbf{A} matrix is created, it is iterated and the 3 CSR vectors are created which do not even consume 0.1% of the memory required by the \mathbf{A} matrix in the general partial products example. These CSR vectors are then used by the KLU algorithm to solve the linear equations.

All of these inefficiencies and large memory consuming data structures can be minimized, and we list some of the proposed solutions that will be implemented.

1. Create only one master netlist free of the duplicates
2. Create an object for each component that stores all the relevant information and simply create pointers where necessary instead of duplicates
3. Directly create the CSR format vectors from the component objects instead of an \mathbf{A} matrix

The direct implementation of the CSR format allowed us to completely mitigate the large memory impact presented by creating the \mathbf{A} matrix in its totality. This was done by simply keeping track of the positions of each non-zero element, iterating through the rows as this vector is already in the correct order and simply adding the non-zero element to the corresponding CSR vector. The other two vectors are created by simply keeping track of the amount of non-zero elements thus far each time one is added to the vector as well as the column index of the said element.

This implementation allows us to simulate circuits of much larger size without the concern of running out of memory. We can still investigate other data structure inefficiencies to further reduce the memory footprint of JoSIM, however, at this point it might be necessary to shift focus to reduce execution speeds for large designs.

7.3 Parallel Processing

Another area that can be looked at for improvement in terms of execution speed is to multi-thread certain parts of JoSIM. We need to, however, be very careful when choosing areas for parallel processing as data dependency issues might occur if the wrong area is chosen. One might think that the largest part of the simulation is the time loop and that it would be a perfect candidate for parallel processing, however one would be mistaken. The time loop part (transient analysis) of the process is extremely dependant on the results of the previous iteration and thus attempting to run this in parallel will cause incorrect results. Data dependency issues occur not only in loops but can also occur when threads are not synchronized at the end of a parallel processed section.

There are, however, other areas that can be multi-threaded very well and should also see major performance improvement in large circuits. One such area is the analysis of each line of the master netlist. If the relevant functions that need to be performed for each component of the netlist is distributed across multiple threads the parsing of the master netlist should see major speed improvement. There should be no issues with data dependency in parallel processing this part of the process if the data structures are altered as mentioned before.

Multi-threading the solving of $\mathbf{Ax} = \mathbf{b}$ is a rather complex task and will not yield significant performance gain unless performed on extremely large simulations where the number of unknowns exceed roughly a few hundred thousand. We thus rather seek alternate numerical solution algorithms that already implement multi-threading and apply them once the unknowns exceed a certain threshold.

Implementation of parallel processing is not a trivial task and great consideration needs to be taken in whether the overhead created to manage the parallel threads would outweigh the speed improvement gained. This is especially true for small designs where parallel processing actually becomes detrimental to performance due to this overhead. We would rather then implement some threshold size that the design needs to exceed before there would be a benefit from multi-threading.

7.4 Optimizations in the Math Engine

The use of different linear algebra methods to solve the $\mathbf{Ax} = \mathbf{b}$ problem could potentially also improve solution time. Different algorithms have different features and are not always a one stop solution to improve simulation speed. In most cases certain trade-offs need to be made to allow for an increase in performance. There are quite a few linear algebra libraries available for C++ and we therefore discuss what makes them unique as well as consider the implementation in JoSIM. Since the values in the \mathbf{A} matrix in a circuit simulator are not symmetric and neither are they positive definite. We thus have to refine the available algorithms to those that solve unsymmetric matrices, which reduces this list to only KLU, SuperLU and UMFPACK.

The current implementation of JoSIM utilizes the KLU solver from the SuiteSparse library created by Tim Davis. This solver was chosen due to the specific applications in electric circuit solving, as well as the ease of implementation and speed advantages over alternatives.

SuperLU was also considered for implementation due to its ability to perform multi-threaded solving as well as the relative ease of implementation. SuperLU is included in various libraries, but can also be installed as stand alone. This library will be added to JoSIM to increase the parallel processing capabilities at a later stage.

UMFPACK was also created by Tim Davis, and it is was designed to allow for multi-frontal re-ordering of the matrix and this gives it a great speed improvement when moving to very large matrices. It is relatively complex to implement when compared to KLU and SuperLU, but it will become necessary as we attempt to simulate VLSI designs.

At present the algorithm best suited for nodal circuit solutions is by far KLU and until improvements are made to this method it will remain the chosen method for implementation within JoSIM.

7.5 Conclusion

In this chapter we critically examined the prospects for JoSIM to simulate VLSI designs. We highlighted key flaws in the JoSIM design and provided various solutions to improve speed, as well as memory management. While these issues exist it will not be able to simulate VLSI designs, however prospects to do so remain and efforts to achieve this are actively being worked

on. We aim to one day simulate microprocessor scale designs in reasonable time which would allow for very accurate timing and result verification.

JoSIM's capability to simulate circuits with junctions exceeding 10,000 within reasonable time, where others start to falter, shows great promise towards this goal. Attempts were made to simulate a circuit with over 200,000 JJs, but due to memory limitations the simulation terminated after 4 hours on a machine with 64GB RAM. The simulation had not yet completed the step of parsing each line at time of termination. This at least provides some hint as to where to start our investigation for optimization.

Chapter 8

Conclusion

This dissertation investigates the requirements for design of large scale superconducting digital circuits. A toolchain and design flow for superconducting circuit design is proposed and discussed. The importance of verification through simulation is highlighted and SuperSTA is introduced.

SuperSTA is a static timing analysis tool developed to verify high-level designs as well as ensure operation at intended throughput speed. SuperSTA algorithmically determines all possible paths from input to output of design. These paths are evaluated using supplied delay timings, which allows SuperSTA to determine the largest delay through a design. Additional metrics are also provided to aid the user in determining the design quality.

The need for simulation software in the superconducting circuit domain was investigated. Existing software such as JSIM, WRspice and PSCAN was tested and scrutinized. A new analogue circuit simulation engine JoSIM was introduced as a modified nodal voltage analysis simulator that utilizes trapezoidal integration to solve linear equations. The development of JoSIM was discussed and design decisions explained. Unique features, such as modified nodal phase based analysis, parametrization through expression parsing and alphanumeric node names are amongst the features present in JoSIM.

The implementation of the modified nodal phase analysis method allows the user to simulate entirely in phase through the voltage-phase relationship present in the JJ. This removes the time dependent nature of the inductor, enabling the application of external magnetic fields to the design.

The initial intended aim of JoSIM was to simulate very large scale superconducting circuits and it was therefore tested for this capability along with existing simulators. Results clearly show a significant speed advantage in using JoSIM while retaining the same level of accuracy. The advantages became more apparent while moving to larger simulations as other simulation engines started failing.

Very large scale simulations with JJs exceeding 100,000 were tested and deemed impossible to simulate with JoSIM in the current state due to memory and execution speed bottlenecks. Various methods of optimization were investigated and some were implemented to show drastic improvement in memory utilization. Alternative optimization techniques are yet to be investigated and implemented to allow JoSIM to perform very large scale simulations.

Through its development it became evident that the JJ model implemented within JoSIM and others is not quite adequate in the representation of the electron tunneling and quasiparticle effect. More accurate models such as the Werthamer model were investigated however deemed far too complex to implement at present. Improvements to the existing model were made to allow temperature dependence and alteration between normal and ballistic electron tunneling.

Continued development on JoSIM through the open source repository will enable public contributions towards the goal of creating a simulator capable of truly simulating the full

nature of the Josephson effect as well as very large scale designs.

Though this dissertation was written with the emphasis on JoSIM and the creation of a circuit analysis tool it may also serve as a reference guide to the creation of any graph-based physics simulation engines. The emphasis being on the ability to create a set of linear equations where the state of two nodes are defined by what happens between them. Using the set of linear equations to set up matrices to solve the states and interpreting the results.

Bibliography

- [1] G. E. Moore, “Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff.,” *IEEE solid-state circuits society newsletter*, vol. 11, no. 3, pp. 33–35, 2006.
- [2] “Superconducting Technology Assessment,” NSA report, August 2005.
- [3] J. Bardeen, L. N. Cooper, and J. R. Schrieffer, “Theory of superconductivity,” *Physical review*, vol. 108, no. 5, p. 1175, 1957.
- [4] B. D. Josephson, “Possible new effects in superconductive tunnelling,” *Physics letters*, vol. 1, no. 7, pp. 251–253, 1962.
- [5] K. K. Likharev and V. K. Semenov, “Rsfq logic/memory family: A new josephson-junction technology for sub-terahertz-clock-frequency digital systems,” *IEEE Transactions on Applied Superconductivity*, vol. 1, no. 1, pp. 3–28, 1991.
- [6] K. K. Likharev, O. A. Mukhanov, and V. K. Semenov, “Resistive single flux quantum logic for the Josephson-junction technology,” *Superconducting Quantum Interference Devices and their Applications*, 1985.
- [7] N. Nakajima, F. Matsuzaki, Y. Yamanashi, N. Yoshikawa, M. Tanaka, T. Kondo, A. Fujimaki, H. Terai, and S. Yorozu, “Design and implementation of circuit components of the sfq microprocessor, core1,” *Superconductor Science and Technology*, vol. 17, no. 3, p. 301, 2004.
- [8] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki, “Design and demonstration of an 8-bit bit-serial rsfq microprocessor: Core e4,” *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 5, pp. 1–5, 2016.
- [9] O. A. Mukhanov, “Energy-efficient single flux quantum technology,” *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 760–769, 2011.
- [10] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, “An adiabatic quantum flux parametron as an ultra-low-power logic device,” *Superconductor Science and Technology*, vol. 26, no. 3, p. 035010, 2013.
- [11] Q. P. Herr, A. Y. Herr, O. T. Oberg, and A. G. Ioannidis, “Ultra-low-power superconductor logic,” *Journal of applied physics*, vol. 109, no. 10, p. 103903, 2011.
- [12] G. Chen, D. A. Church, B.-G. Englert, C. Henkel, B. Rohwedder, M. O. Scully, and M. S. Zubairy, *Quantum computing devices: principles, designs, and analysis*. Chapman and Hall/CRC, 2006.
- [13] J. Oppenlander, T. Trauble, C. Haussler, and N. Schopohl, “Superconducting multiple loop quantum interferometers,” vol. 11, pp. 1271–1274, IEEE, 2001.

- [14] O. V. Snigirev, M. L. Chukharkin, A. S. Kalabukhov, M. A. Tarasov, A. A. Deleniv, O. A. Mukhanov, and D. Winkler, "Superconducting quantum interference filters as rf amplifiers," vol. 17, pp. 718–721, IEEE, 2007.
- [15] L. W. Nagel and D. O. Pederson, "SPICE-Simulation Program with Integrated Circuit Emphasis," tech. rep., 1973.
- [16] W. N. Laurance, "SPICE2: A Computer Program To Simulate Semiconductor Circuits," 1975.
- [17] T. Quarles, "The SPICE3 Implementation Guide," *Ucberl M*, 1989.
- [18] I. Getreu and D. Teegarden, "An introduction to behavioural modelling," *Microelectronics journal*, vol. 24, no. 7, pp. 708–716, 1993.
- [19] C.-L. Huang and S. Su, "Approaches for computer-aided logic/system design using hardware description language," in *Proceedings of International Computer Symposium 1980*, pp. 772–790, 1980.
- [20] IEEE Computer Society, *IEEE Standard VHDL Language Reference Manual*. 2009.
- [21] S. Sutherland, *Verilog HDL quick reference guide*. Sutherland HDL Consulting, 1995.
- [22] M. N. Muchuka, J. A. Delpont, and C. J. Fourie, "Superconducting digital circuit design with an open source and freeware tool chain," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 8, pp. 1–8, 2016.
- [23] R. S. Bakolo, J. A. Delpont, P. Febvre, and C. J. Fourie, "Analysis of a shielding approach for magnetic field tolerant sfq circuits," *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–5, 2017.
- [24] J. A. Delpont and C. J. Fourie, "A static timing analysis tool for rsfq and ersfq superconducting digital circuit applications," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 5, pp. 1–5, 2018.
- [25] J. A. Delpont, K. Jackman, P. Le Roux, and C. J. Fourie, "Josim-superconductor spice simulator," *IEEE Transactions on Applied Superconductivity*, 2019.
- [26] C. J. Fourie and M. H. Volkmann, "Status of superconductor electronic circuit design software," *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, pp. 1300205–1300205, 2013.
- [27] C. J. Fourie, "Digital superconducting electronics design tools status and roadmap," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 5, pp. 1–12, 2018.
- [28] O. of the Director of National Intelligence, "Proposers' Day Notification for SuperTools," January 2016.
- [29] S. Williams, "Icarus Verilog." <http://iverilog.icarus.com>, January 2017. Accessed: 2018-08-28.
- [30] N. Katam, A. Shafaei, and M. Pedram, "Design of complex rapid single-flux-quantum cells with application to logic synthesis," in *2017 16th International Superconductive Electronics Conference (ISEC)*, pp. 1–3, IEEE, 2017.

- [31] N. M. Muchuka, *Hardware description language modelling and synthesis of superconducting digital circuits*. Stellenbosch : Stellenbosch University, 2017.
- [32] “gEDA Project Wiki.” <http://wiki.geda-project.org/>, February 2018. Accessed: 2018-08-28.
- [33] S. R. Whiteley, “Xic Reference Manual.” <http://www.wrcad.com/manual/xicmanual/xicmanual.html>, August 2018. Accessed: 2018-08-28.
- [34] C. J. Fourie and W. J. Perold, “Comparison of genetic algorithms to other optimization techniques for raising circuit yield in superconducting digital circuits,” vol. 13, pp. 511–514, IEEE, 2003.
- [35] M. Jeffery, W. J. Perold, Z. Wang, and T. Van Duzer, “Monte carlo optimization of superconducting complementary output switching logic circuits,” *IEEE transactions on applied superconductivity*, vol. 8, no. 3, pp. 104–119, 1998.
- [36] C. J. Fourie and W. J. Perold, “Simulated inductance variations in rsfq circuit structures,” vol. 15, pp. 300–303, IEEE, 2005.
- [37] M. E. Law and S. M. Cea, “Continuum based modeling of silicon integrated circuit processing: An object oriented approach,” *Computational Materials Science*, vol. 12, no. 4, pp. 289–308, 1998.
- [38] W. P. Burlison, M. Ciesielski, F. Klass, and W. Liu, “Wave-pipelining: a tutorial and research survey,” *IEEE Transactions on very large scale integration (vlsi) systems*, vol. 6, no. 3, pp. 464–474, 1998.
- [39] C. J. Fourie, N. K. Katam, J. A. Delport, S. N. Shahsavani, T. R. Lin, K. Jackman, and M. Pedram, “Design methodologies and tools for SFQ logic circuits: precursor project to SuperTools,” *IEEE Transactions on Applied Superconductivity*, 2019. To be submitted.
- [40] C. J. Fourie, “Extraction of dc-biased sfq circuit verilog models,” *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 6, pp. 1–11, 2018.
- [41] S. N. Shahsavani, T.-R. Lin, A. Shafaei, C. J. Fourie, and M. Pedram, “An integrated row-based cell placement and interconnect synthesis tool for large sfq logic circuits,” *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–8, 2017.
- [42] Cadence, *LEF/DEF language reference*, November 2009.
- [43] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms , Second Edition*. 2001.
- [44] S. Knowles, “A family of adders,” pp. 30–34, 1999.
- [45] N. Werthamer, “Nonlinear self-coupling of josephson radiation in superconducting tunnel junctions,” *Physical Review*, vol. 147, no. 1, p. 255, 1966.
- [46] H. Kratz and W. Jutzi, “Microscopic simulation model of josephson junctions for standard circuit analysis programs,” *IEEE Transactions on Magnetics*, vol. 23, no. 2, pp. 731–734, 1987.
- [47] R. Jewett, “Josephson junctions in spice 2g5,” *Electronics Research Lab internal memorandum, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA*, p. 94720, 1982.

- [48] S. Whiteley, “Josephson junctions in spice3,” *IEEE Transactions on Magnetics*, vol. 27, no. 2, pp. 2902–2905, 1991.
- [49] T. P. Orlando and K. A. Delin, *Foundations of applied superconductivity*, vol. 8. Addison-Wesley Reading, MA, 1991.
- [50] E. S. Fang and T. Van Duzer, “A Josephson integrated circuit simulator (JSIM) for superconductive electronics application,” in *Extended Abstracts of 1989 Intl. Superconductivity Electronics Conf. (ISEC 1989)*, 1989.
- [51] S. Polonsky, V. Semenov, and P. Shevchenko, “Pscan: personal superconductor circuit analyser,” *Superconductor Science and Technology*, vol. 4, no. 11, p. 667, 1991.
- [52] S. Polonsky, P. Shevchenko, A. Kirichenko, D. Zinoviev, and A. Rylyakov, “Pscan’96: New software for simulation and optimization of complex rsfq circuits,” 1997.
- [53] A. Odintsov, V. Semenov, and A. Zorin, “Specific problems of numerical analysis of the josephson junction circuits,” *IEEE Transactions on Magnetics*, vol. 23, no. 2, pp. 763–766, 1987.
- [54] D. R. Gulevich, *User Guide for MiTMOJCo (Microscopic Tunneling Model for Josephson Contacts)*, August 2017.
- [55] A. Vladimirescu, *The SPICE book*. John Wiley & Sons, Inc., 1994.
- [56] U. Wali, R. Pal, and B. Chatterjee, “On the modified nodal approach to network analysis,” *Proceedings of the IEEE*, vol. 73, no. 3, pp. 485–487, 1985.
- [57] T. A. Davis and E. Palamadai Natarajan, “Algorithm 907: Klu, a direct sparse solver for circuit simulation problems,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 37, no. 3, p. 36, 2010.
- [58] F. Costantini, D. Gibson, M. Melcher, A. Schlosser, B. Spitzak, and M. Sweet, *FLTK 1.3.4 Programming Manual*, 2016.
- [59] W. Haberkorn, H. Knauer, and J. Richter, “A theoretical study of the current-phase relation in josephson contacts,” *physica status solidi (a)*, vol. 47, no. 2, pp. K161–K164, 1978.
- [60] Y. Hashimoto, S. Yorozu, Y. Kameda, and V. K. Semenov, “A design approach to passive interconnects for single flux quantum logic circuits,” vol. 13, pp. 535–538, IEEE, 2003.
- [61] Y. Hashimoto, S. Yorozu, Y. Kameda, A. Fujimaki, H. Terai, and N. Yoshikawa, “Design and investigation of gate-to-gate passive interconnections for sfq logic circuits,” *IEEE transactions on applied superconductivity*, vol. 15, no. 3, pp. 3814–3820, 2005.
- [62] E. W. Dijkstra, “Algol-60 Translation,” 1961.
- [63] P. of Chaeronea, “The Symposiacs Question III,” 1st Century CE.

Appendices

Appendix A

Journal Paper - Superconducting Digital Circuit Design With an Open Source and Freeware Tool Chain

- M. N. Muchuka, J. A. Delpont, and C. J. Fourie, "Superconducting digital circuit design with an open source and freeware tool chain," *IEEE Transactions on Applied Superconductivity*, 2016.[22]

In this paper we discuss the availability of open source tools to design superconducting integrated circuits. The paper is headed by Dr Muchuka, who's suggestion it was to investigate the topic. Personal contributions to this paper amounts to large portions of the research, testing of examples using the software as well as providing most of the figures. Copyright for this paper is held by IEEE Transactions on Applied Superconductivity.

Superconducting Digital Circuit Design With an Open Source and Freeware Tool Chain

Maguu N. Muchuka, Johannes A. Delpont, and Coenrad Johann Fourie, *Member, IEEE*

Abstract—Superconducting digital circuits have been around for decades, but recent projects that exploit such circuits for low-power, high-performance computing are rapidly maturing superconducting circuit technologies. As a result of increasing circuit complexity, there is renewed focus on superconducting digital circuit design tools. Until now, most computer-aided design (CAD) tool development for superconducting electronics (SCE) circuit design has been based on calibrating semiconductor tools, rather than creating new technology-specific tools for the SCE circuit design. The recent development shows bias toward large and expensive CAD tools. These tools, or modules developed for commercial semiconductor CAD tools, require users to have access to an expensive commercial semiconductor CAD software, which places it out of the reach of new entrants or small research groups in SCE. In this paper, the open-source tools that can be used as alternatives to form an SCE design tool chain are described. All the design stages in an SCE integrated circuits design flow, from circuit specification down to layout design, are noted in this paper. Each stage is discussed and the available open-source tools are described. The inherent disadvantages of these open-source tools are also pointed out. A design example is then provided to demonstrate a complete open source and freeware tool chain.

Index Terms—Circuit design tool chain, circuit simulators, open-source tools, single flux quantum (SFQ) circuits.

I. INTRODUCTION

SUPERCONDUCTING electronics (SCE) design has become a key topic of research in the quest to develop faster yet low-power beyond-CMOS alternatives for power-hungry supercomputers and data centers [1]. As the barrier of physical limitations on semiconductor electronics approaches, the SCE technology is an increasingly viable option for the next major leap in high-performance computing [2]. Several superconducting logic families have emerged in the process, such as rapid single flux quantum (RSFQ) [3], energy-efficient RSFQ [4], energy-efficient SFQ [5], reciprocal quantum flux [6], and adiabatic quantum flux parametron (AQFP) [7].

As the SCE research community grows, many new researchers are facing challenges in selecting computer-aided design (CAD) tools. In most reported work, expensive commercial tools and/or in-house tools are utilized. The cost of these tools or/and accessibility may limit their acquisition, especially in the

developing countries. This makes it difficult for some novice researchers to reproduce the existing knowledge, and their impact on the technology is delayed.

An alternative is the use of circuit design tools that are either open source or freeware. In the case of open-source tools, users are free to use, alter, and redistribute the software, whereas the users of freeware tools have a free license with no limitations. This is not to be confused with shareware, which has limitations that can be removed by buying a full license.

Open source and freeware tools have existed for decades and have been used extensively in semiconductor-based technology. In the case of supported technology features, these tools provide results that are sufficiently similar to those of commercial tools to be practically useful for most research and development applications of budget-constrained groups. Although open-source tools lack regular updates and proper documentation, they allow the users to extend their technology support by modifying the source code. A drawback of open source or freeware tools is that these are often more difficult to use or concatenate than commercial tools. Moreover, some require the user to build the program from source code, which necessitates expertise in software development tools and porting across different operating systems.

Comprehensive studies on the SCE circuit design software, their performance, and accuracy were done in 1999 [8] and 2013 [9]. In [9], mention of open-source tools and the availability thereof was made. However, this was not the purpose of the study.

There is a good future for the open-source approach in CAD tools, with electronic design automation (EDA) giants such as Synopsys participating in OpenMAST [10], a tool that combines various open-source tools. Similarly, a complete tool chain for SCE can be formulated by evaluating and selecting the suitable open-source tool for every stage in the SCE design flow. A challenge is faced when selecting the right tool, which provides the most functionality and best results.

In this paper, a selection of open source and freeware tools for the SCE circuit design, which can be combined to construct useful tool chains, is described. This is done in three steps. First, simplified design flows are proposed for small-, medium-, and large-scale SCE circuits. Second, the available open source and freeware tools are discussed. Finally, the selection of a tool chain is presented with the aid of a design example.

II. DESIGN FLOW

With a standard design flow for SCE circuits not yet established, different design flows have been used in the reported literature. The design flow presented in this paper is not nec-

Manuscript received June 2, 2016; revised September 20, 2016; accepted October 3, 2016. Date of publication October 27, 2016; date of current version November 15, 2016. This paper was recommended by Associate Editor O. Mukhanov.

The authors are with the Department of Electrical and Electronic Engineering, Stellenbosch University, Stellenbosch 7602, South Africa (e-mail: maguunic@yahoo.co.uk.com; joeydelp@gmail.com; coenrad@sun.ac.za).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TASC.2016.2622623

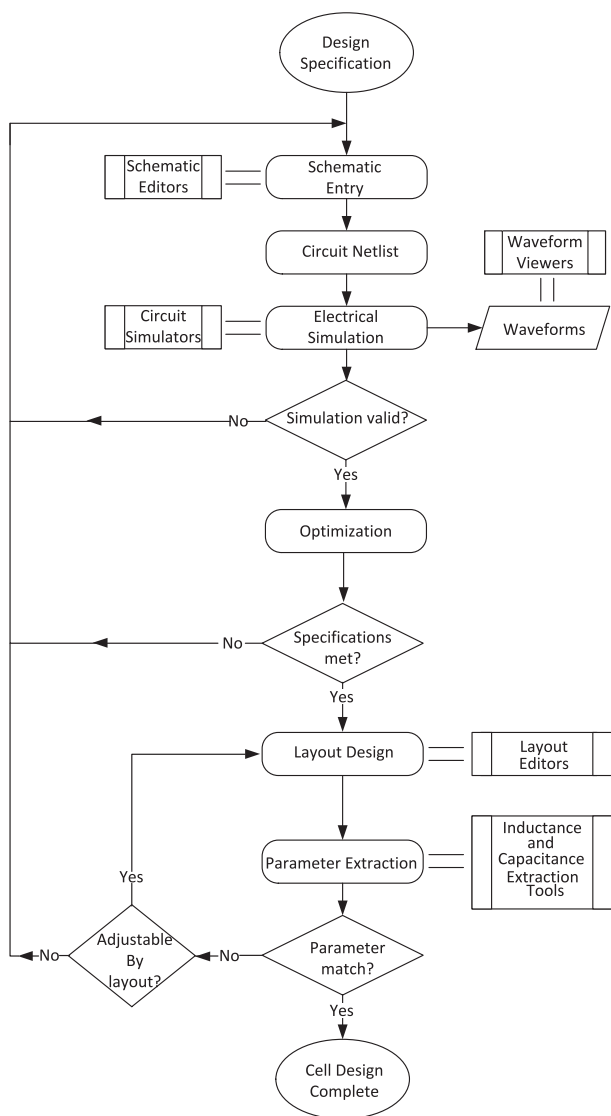


Fig. 1. Small-scale circuit design flowchart for SCE digital circuits.

essarily an instruction but rather a guideline. Two approaches based on circuit complexity are presented, the first being small-scale circuit design and the other being medium- and large-scale circuit design.

A typical small-scale circuit design flow is depicted in Fig. 1. Initially, a target SCE logic family and a fabrication process are identified. The design rules associated with them then guides the designer through the design process. The process begins with circuit specification, followed by logical design from which a suitable topology is obtained and the initial design values estimated. A schematic representation of the circuits is then built using a schematic capture tool, after which the circuit netlist is generated. The netlist may be hand edited to include simulator directives for the required circuit analysis. At the electrical simulation stage, a circuit simulator runs the netlist supplying it with the excitation signals necessary to test the circuit operation. In cases where the circuit does not simulate correctly, the design parameters are modified and the simulation repeated iteratively until a valid simulation is observed. Design optimization is then carried out to ensure that, despite the parameter spread during

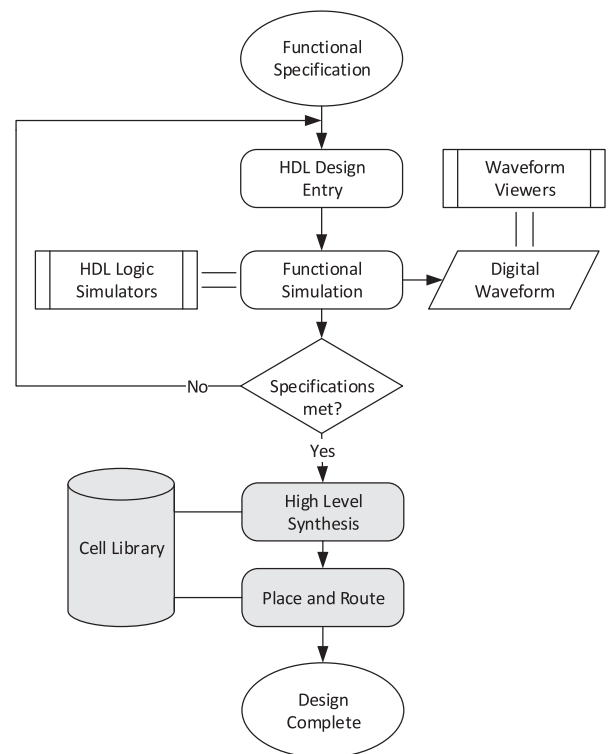


Fig. 2. Medium- and large-scale design flowchart for SCE digital circuits.

the fabrication process and variation of bias current, the circuit operates properly. If an optimal set of circuit parameters cannot be obtained, the design is modified at the design entry/netlist. Using the optimal circuit elements values, a geometrical representation of the circuit is then laid down using a layout editor. The resulting layout is used by a parameter extraction tool to compute the circuit parameters. The extracted parameters are then compared with the netlist parameters. If the former deviates from the latter beyond the required tolerance, the layout dimensions are adjusted and verified until the extracted values are within the required range. In the worst case scenario, where the design rules of the target process cannot allow further adjustment of element dimensions, the design process has to be restarted. The final layout is then taped out as a test cell and added to a reliable cell library repository if it is measured successfully with sufficient operating margins.

Designing medium- and large-scale circuits, on the other hand, employs a semicustom methodology based on cell library. This approach is adopted from semiconductor technology. However, much needs to be done to automate it in superconducting technology. With this methodology, a medium- or large-scale circuit is built from a collection of well optimized and tested small-scale circuits existing as a cell library. Custom interconnecting circuits are then built to route the signals to and from the standard cells. This method is demonstrated in [11] and [12].

A simplified medium- and large-scale design flowchart is presented in Fig. 2. The gray-shaded blocks of the design flow represent the parts that have not yet been developed in the open-source domain. The design is achievable at a high level of abstraction using hardware description languages (HDL). From the functional specification, a design is captured with HDL,

such as VHDL or Verilog. The resulting behavior or/and structural circuit description is then simulated using logic simulators. Simulation output can be viewed as waveforms using waveform viewer tools. If the simulation results do not match the specification, the HDL description is modified and simulated until the desired specifications are achieved.

The design should then be synthesized to obtain its circuit representation. This is a weak link in the general superconducting logic circuits design flow. To the best of our knowledge, there are no SCE cell libraries, syntheses, or place and route tools in the open-source domain. Nonetheless, cell libraries exist in various research institutions. A proprietary synthesis, place and route tool has been demonstrated [13], but it is applicable to only a specific cell layout and clock distribution strategy. Thus, the designer will have to manually complete the design process or develop a tool to handle it for generic processes and cell libraries.

For other than the instances noted, every stage in these design flows has associated tools in the open-source community.

A. Schematic Capture

Schematic capture tools allow the designers to draw a circuit on a digital canvas. This provides the visualization of the design and documentation, with some tools allowing schematics to be exported in a publication-quality graphics format. Drawing a schematic for a circuit can be easy; however, a challenge is faced when using nonstandard components such as a Josephson junction (JJ). Most open source and freeware schematic tools do not have a JJ built-in. They require the user to create the JJ symbol and define its attributes. This can be time-consuming especially if the knowledge of the tool is limited. Thus, many designers prefer schematic tools with a built-in JJ. Tools like these are few but work quite well. The popular open-source schematic tools are gSchem in gEDA [14], Sced in JSpice3 [15], Fritzing [16], KiCad [17], KTechlab [18], and Xcircuit [19]. Among them, only the first two include the JJ model.

gSchem is part of the gEDA tool suite, which is available in binary and source code for POSIX systems. The JJ model is incorporated in gSchem's component library, making the tool easy to use for new researchers. The tool also supports scheme language scripts, which allow the extension of its functionality without recompiling the source code. This makes the tool attractive to experienced users.

The program is easy to use. It provides an add-component dialogue and a grid layout, which makes component placing simple. However, a *.model* directive is required for the JJs used in the schematic. The JJ model program [20] is useful for new or inexperienced researchers who need to generate junction models.

The JSpice3 schematic editor Sced is a keyboard driven editor. Sced has a JJ model built into its component library. However, it offers only fundamental properties for schematic capture, making it difficult to use as the component count increases.

KiCad's Eeschema, Fritzing, KTechLab, and Xcircuit are extensively used by the semiconductor circuit designers. They do not have JJ in their component libraries, but provide means

of adding new components. By adding a JJ model to these tools SFQ circuits can be captured. In addition, KTechLab and Xcircuit have in-built netlist generators. Alternatively, schematic editors built-in SPICE suites such as LTSPice can be used.

B. Netlist Generation

Once the schematic has been created, a circuit netlist specifying all the components, their node numbers, and their values, is generated. Most netlist generator tools are embedded in schematic capture tools. gNetlist in gEDA suite is so far the only open-source netlist generator tool used with superconducting circuits. It is important to note that the generated netlist should be compatible with the available circuit simulator, although translation tools are easily scripted and are supported by gNetlist. The gNetlist tool in conjunction with an in-house modified version of the SPICE database file is used to generate a Josephson-junction simulator (JSIM) compatible netlist at Stellenbosch University. The designer may opt to use the schematic as a guide and generate the netlist by hand. This results in a more compact and easy to debug netlist; however, it is time-consuming.

C. Circuit Simulator

JSPICE [21] was the first SCE circuit simulator built by adding the JJ model into SPICE2; JSPICE can only handle circuits with a limited number of JJs. Motivated by the limitations of JSPICE, the JSIM was developed [22]. The JSIM was built focusing on reducing computation effort per time step in circuits with JJ as the only nonlinear device. This made the JSIM faster than its counterparts and currently the most commonly used open-source circuit simulator. JSIM generates an output data file, which can then be plotted using third-party plotting tools such as GNU Plot [23]. JSIM can be found on the Whiteley Research, Inc.'s, repository of free software tools [24]. PSCAN [25] [26] is another circuit simulator dedicated to SCE circuit simulation. This simulator has been extensively used for SCE circuit design, and is still popular in the USA. The developers consider PSCAN open to the public, but it lacks support [27] and has limited accessibility. It is not currently possible to download a copy of PSCAN from a publicly accessible repository. Csim [28] is also an open-source SCE simulator built from scratch. It uses scripting language similar to C, which can be attractive to many designers. However, it has not received much attention from the digital SCE community, and to our knowledge an SFQ circuit simulation has not yet been demonstrated in Csim. Csim is also slow compared to JSIM and its documentation is limited.

LTSPice [29] and Ngspice can successfully simulate JJ-based SCE circuits, such as a Josephson transmission line (JTL) and D flip-flop (DFF) when the JJ model subcircuit is included, but ngspice fails to run simulations with step sizes smaller than 0.5 ps. Thus, by including the JJ-model subcircuit in any spice derivative, it is possible to simulate the small-scale SCE circuits. This makes the use of freeware modules such as LTSpice more convenient for beginners because they include a schematic editor.

D. Optimization Tools

Circuit optimization tools are used to ensure that the selected set of circuit parameters provide wide operation margins and high circuit yields. The available optimization tools for SCE have been built from scratch. Among them is MALT [30], an open-source yield optimization tool based on inscribed hyperspheres algorithm, and COWBOY [26] developed to work with PSCAN. COWBOY suffers similar problems as PSCAN mentioned in Section II-C. Other tools reported in the literature are proprietary. The limited number of optimization tools, however, may not constrain the designer when using free tools in the SCE circuit design. The designer can as well implement the optimization methods described in [26], [31] and [32]. This can be achieved with a scripting language or a programming language. A plethora of such languages with open-source compilers are available.

E. Circuit Layout and Verification

Layout editors are used to lay down the physical dimensions of the circuit elements. There are several open source and free-ware projects addressing the issues of integrated circuit (IC) layouts in semiconductor technology, such as Magic [33], Graal [34], Toped [35], KIC [36], and LASI [37]. Among them, Magic [38] and LASI [39] have been successfully adopted in superconducting IC. This is achieved by creating a technology file for superconducting circuits and setting it as the active technology file in the tool used. Thus, semiconductor tools that allow the changing of the target technology file can be adapted in superconductor technology.

The existing layout editors do not have design rule check and layout versus schematic (LVS) verification features for superconducting circuits. Thus, to aid layout verification, parameter extraction tools, commonly inductance extraction tools in the case of SCE, are used. Commonly used inductance tools are Lmeter [40], 3D-MLSI [41], FastHenry [42], and InductEx [43]. Among them, InductEx provides a model-size limited version to public access but L-meter and FastHenry are available in full for free.

F. HDL Logic Simulators

Researchers have shown that industry standard HDLs, such as Verilog and VHDL, have the ability to support superconducting logic circuit modeling [44]–[47]. Cell libraries containing HDL models have been built to support the logic-level simulation of large-scale circuits. However, these cell libraries, such as CONNECT library [44], SBU VHDL library [45], RSFQ64 [46], among others are based on commercial simulators. An attempt to use open-source simulators in modeling superconducting circuits was first reported in [47]. The authors successfully modeled AQFP gates and simulated using an open-source simulator iVerilog [48] and FreeHDL [49]. At Stellenbosch University, we have been evaluating various open-source logic simulators for superconducting circuit simulation. Virtually any logic simulator can be adopted for use in superconducting technology. However, the choice needs to be made keeping in mind the

TABLE I
OPEN-SOURCE HDL SIMULATORS EVALUATED

Simulator	Language	Standard supported	
		Full	Partial
GHDH	VHDL	87	93
FreeHDL	VHDL	93	-
NVC	VHDL	93	2008
iVerilog	Verilog	2001	2005
Verilator	Verilog	2001	2005

various supported language standards, for example, FreeHDL supports VHDL 93 standard, whereas Verilator [50] supports Verilog 2001 standard and partially 2005 standard. This limits the flexibility of the designer; however, the new modeling features described in the new HDL standards, such as VHDL 2008 and Verilog 2005, can be performed in earlier versions but not as succinctly. Thus, with the supported language features, a designer can successfully model SCE circuits. On the other hand, unlike the commercial tools, the user can extend the tool to cover the required language construct. Table I shows a summary of the simulators we have tested.

G. Synthesis

Two synthesis approaches have so far been demonstrated in SFQ technology. In [51]–[53], a binary decision diagram-based synthesis is presented, whereas in [54] a transduction-based framework is used. In the transduction method, the authors suggest that SIS of UC Berkeley can be utilized. The two methods, however, handle Boolean function synthesis. High-level synthesis (HLS) behavior of superconducting logic circuits has not been reported in the literature. The HLS steps involving, code optimization, allocation, and scheduling in superconducting and semiconducting technologies are similar. A major difference will be in the binding step. Therefore, calibrating open-source semiconductor tools can cut down the tool development time significantly. HLS open-source semiconductor tools such as, Yosys [55] and Odin II [56] are potential candidates for calibration in superconducting technology.

H. Layout Versus Schematic

To verify that the geometrical representation of a circuit in the layout is logically equivalent to its schematic, an LVS tool is necessary. An attempt toward general LVS in SFQ circuits is reported in [57], where the authors presented an algorithm for LVS and successfully verified a JTL and a pulse splitter. In the open-source community, no such tools have been developed nor adapted from the semiconductor technology for SFQ circuits. Thus, an open source or freeware tool chain needs to rely on manual LVS at this stage.

III. DESIGN EXAMPLE USING OPEN-SOURCE TOOLS

We demonstrate the design examples of superconducting circuits by describing the design stages presented in the previous section. The main objective of these examples is to present a complete open source and freeware toolset that can be used in

SCE design process. A summary of these tools is presented in Table II at the end of this section.

Listing 1. JSIM compatible netlist generated by gEDA's gNetlist from schematic shown in Fig. 3.

```

*****
* Begin .SUBCKT model *
* spice-sdb ver 4.28.2007 *
*****
.SUBCKT DFF7 4 13 6
*Begin SPICE netlist of main design
B1 1 0 jj1 area=0.25
B2 2 3 jj1 area=0.25
B3 3 0 jj1 area=0.2
B4 9 0 jj1 area=0.175
B5 11 9 jj1 area=0.15
B6 12 0 jj1 area=0.25
B7 7 0 jj1 area=0.25
IB1 0 5 pwl (0 0 5p 0.00025)
IB2 0 8 pwl (0 0 5p 0.0001)
IB3 0 10 pwl (0 0 5p 0.000175)
IB4 0 12 pwl (0 0 5p 0.00014)
L1 4 1 2.2e-012
L2a 1 5 1e-012
L2b 5 2 1.7e-012
L3 3 8 1.1e-012
L4 8 9 8.3e-012
L5a 9 10 3.8e-012
L5b 10 7 9e-013
L6 7 6 2.2e-012
L7 11 12 3e-012
L8 12 13 2.3e-012
RB1 1 0 1.02
RB2 2 3 1.02
RB3 3 0 1.275
RB4 9 0 1.45714
RB5 11 9 1.7
RB6 12 0 1.02
RB7 7 0 1.02
.model JJ1 jj(icrit=1m, cap=5p, rn=90, rtype=0)
.ends DFF7
*****

```

A. Small-Scale Design Example

A DFF is used as an example of small-scale circuits. Once the specifications of a DFF had been analyzed, a suitable circuit topology to implement it was identified. The seven-junction DFF shown in Fig. 3 is captured with gSchem from gEDA. A publication-quality graphic was then generated from gSchem for documentation. Using gNetlist of gEDA, a JSIM compatible netlist was generated to be used in the electrical simulation. Listing III shows the DFF subcircuit netlist generated from gSchem. Appropriate inputs from a test bench were given to the DFF circuit and simulated with the JSIM circuit simulator. A data file is generated by JSIM as the simulation output. GNUplot, an open-source plotting tool, was used to generate the graphical view of the results shown in Fig. 4(a). Once the required operation was obtained, the DFF was optimized for widest critical margin.

TABLE II
SUMMARY OF THE OPEN-SOURCE TOOLS USED IN EACH DESIGN STEP

Category	Tools
Schematic editor	gEDA's gschem
Netlist generator	gEDA's gnetlist
Circuit simulator	JSIM
Analog waveform viewer	GNUplot
Optimization	Custom using C++
Layout editor	LASI
Parameter extraction	InductEx (free version)
HDL description entry	Notepad ++
HDL simulators	IVerilog and FreeHDL
Digital waveform viewer	GTKwave

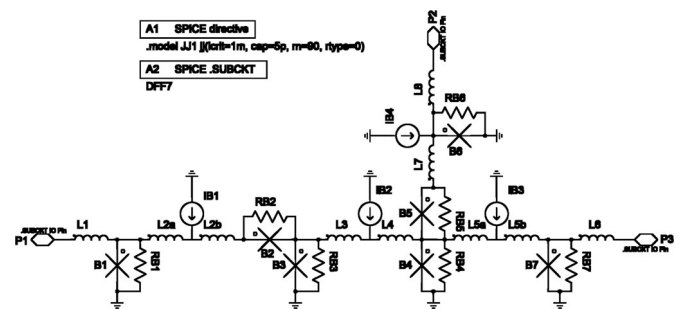


Fig. 3. Schematic capture of seven-junction DFF captured with gSchem.

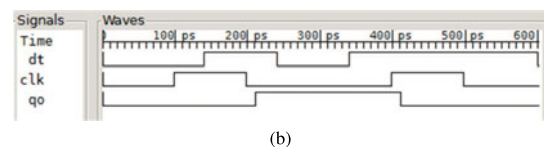
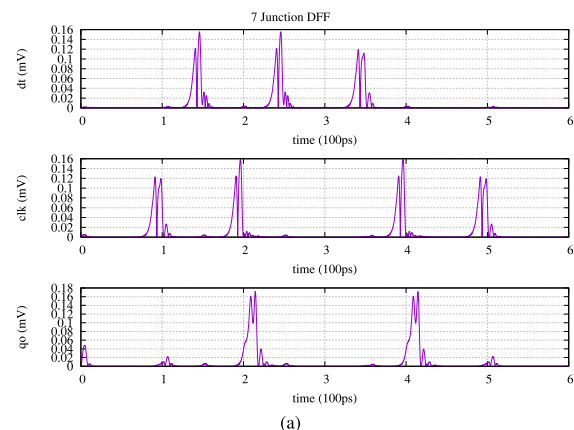


Fig. 4. Timing diagrams. (a) Electrical simulation output of DFF obtained from JSIM and plotted using GNUplot. (b) VHDL model simulation waveforms carried out with FreeHDL with an SFQ pulse encoded as level transition [45] and viewed with GTKwave. dt and clk are data and clock input, respectively, whereas qo is the output.

To do so, an optimization program was written on the basis of the centering method [31] and the critical margin as the figure of merit. Fig. 5 shows the parameter margins of the optimized DFF. Using the optimal values, the circuit layout was designed based on the IPHT process design rules and implemented in LASI. For this to be performed, an IPHT layer definition file had to be integrated into LASI. Fig. 6 shows the DFF layout

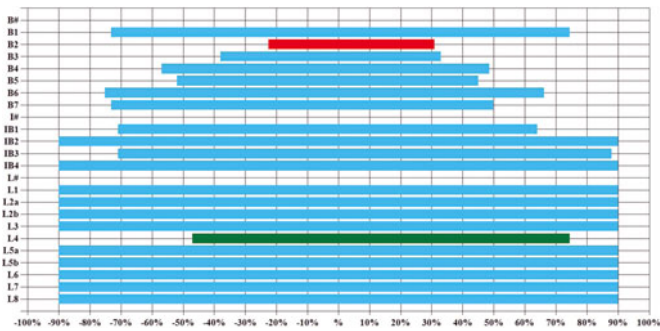


Fig. 5. Margin analysis for the optimized DFF circuit.

designed in LASI. The layout was then converted from LASI default file format to GDSII. Using the free version of InductEx, inductance values were extracted from the GDSII layout and manually verified.

B. Medium- and Large-Scale Design Example

With the current state of open-source tools, only high-level logic simulation can be performed on medium- and large-scale Listing 2. VHDL behavior model for DFF.

```

library ieee;
use ieee.std_logic_1164. all;
entity DFF is
  port
  (
    clk : in std_logic;
    d   : in std_logic;
    q   : inout std_logic;
  );
end entity DFF;
architecture bhv of DFF is
  begin
    process(clk)
      variable x1,x2: time := 0 ps;
      begin
        if(clk'event) then
          x1:=x2;
          x2:=NOW;
          assert(d' last_event > minimum_data2clock_
            seperation)
            report "Data to clock seperation time violated"
            severity error;
          if(d' last_event < (x2-x1)-clock2output_delay) then
            q <= transport (not q) after clock2output_delay;
          end if;
        end if;
      end process;
    end architecture bhv;
  
```

SCE circuits. The structural HDL modeling approach was used to build medium- and large-scale design in this paper. This was begun by creating an HDL description cell library of RSFQ circuit primitives. These primitives were then used as instances in a large design. Listing III-B and Fig. 4(b) show the VHDL model and logic simulation waveforms of the DFF described

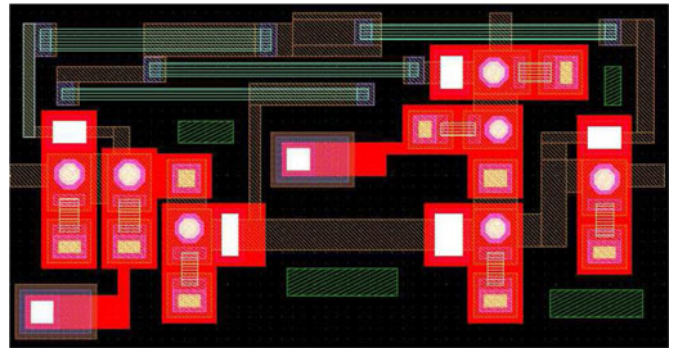


Fig. 6. Seven-JJ DFF layout prepared using LASI based on the IPHT design rules. The layout is implemented in a $150 \mu\text{m} \times 75 \mu\text{m}$ area.

earlier in this section. FreeHDL and iVerilog open-source HDL logic simulators were used to simulate the behavior description. Using the HDL cell library, a structural HDL description of an 8-b-Wallace-tree multiplier was implemented. While the arithmetic logic section was implemented with the help of generate keyword in Verilog HDL and VHDL, the clock section was implemented by hand coding all the necessary module instantiations that provide the required delay for each arithmetic stage. The circuit was then simulated to verify its operation. The HDL model simulation took 68 s, whereas the JSIM counterpart took 8168 s on the same platform.

IV. CHIP-SCALE DESIGN

Current SCE technologies limit chip design and layout to about 1000-10 000 gates, which could be synthesized, but is currently placed and routed by hand. Thus freeware/open-source tools can be used to design the chip-scale systems, as individual gates are placed and routed by hand-in layout editors such as LASI.

V. CONCLUSION

This paper brings out the awareness of open source and freeware tools, which can be used in superconducting circuit technology. Successfully tested tools and those which can be modified for use in SCE are pointed out. Based on the design flow proposed in this paper, an example is used to demonstrate a complete tool chain for small-scale circuits. The use of an open-source HDL simulators for medium- and large-scale circuits is also demonstrated. This would serve as a guideline for novice researchers in SCE. From a review of the availability of open source or freeware tools for SCE design, it is evident that the focus for open-source development to really push progress in SCE design should be on synthesis tool from high-level behavioral descriptions, as well as the associated place and route tools. These tools require close interaction with the circuit design community, because successful tool implementation, even if it is done with repurposed open-source tools from semiconductor circuit design, will require industry-defined constraints on cell library description and logic levelization (for synthesis), layout design (for place and route) and clock synthesis techniques. We would further like to point out that open-source

tools can be as accurate as the commercial tools; however, due to the continuous upgrading of commercial tools, commercial tools are recommended for large-scale commercial applications.

REFERENCES

- [1] D. C. Brock, "The NSA's frozen dream," *IEEE Spectr.*, vol. 53, no. 3, pp. 54–60, Mar. 2016.
- [2] M.I.T Lincoln Laboratory, "Forecasting superconductive electronics technology," *Next Wave*, vol. 20, no. 3, pp. 3–11, 2014.
- [3] 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, no. 1, pp. 3–28, Mar. 1991.
- [4] D. Kirichenko, S. Sarwana, and A. Kirichenko, "Zero static power dissipation biasing of RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 776–779, Jun. 2011.
- [5] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 760–769, Jun. 2011.
- [6] 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, 2011, Art. no. 103903.
- [7] K. Inoue, N. Takeuchi, K. Ehara, Y. Yamanashi, and N. Yoshikawa, "Simulation and experimental demonstration of logic circuits using an ultra-low-power adiabatic quantum-flux-parametron," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, Jun. 2013, Art. no. 1301105.
- [8] 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.
- [9] C. J. Fourie and M. H. Volkmann, "Status of superconductor electronic circuit design software," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, Jun. 2013, Art. no. 1300205.
- [10] Synopsys, "OpenMAST." [Online]. Available: <http://news.synopsys.com/index.php?item=122764>, Accessed on: Nov. 2015.
- [11] M. Dorojevets, C. L. Ayala, and A. K. Kasperek, "Data-flow microarchitecture for wide datapath RSFQ processors: Design study," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 787–791, Jun. 2011.
- [12] F. Matsuzaki, N. Yoshikawa, M. Tanaka, A. Fujimaki, and Y. Takai, "A behavioral-level HDL description of SFQ logic circuits for quantitative performance analysis of large-scale SFQ digital systems," *Physica C, Supercond.*, vols. 392–396, pp. 1495–1500, 2003.
- [13] Y. Kameda, S. Yorozu, and Y. Hashimoto, "Automatic single-flux-quantum (SFQ) logic synthesis method for top-down circuit design," *J. Phys., Conf. Ser.*, vol. 43, no. 1, pp. 1179–1182, 2006. [Online]. Available: <http://iopscience.iop.org/1742-6596/43/1/287>
- [14] gEDA Project. [Online]. Available: <http://www.geda-project.org>, Accessed on: Nov. 2015.
- [15] Jspice3. [Online]. Available: <http://www.wrcad.com/jspice3.html>, Accessed on: Nov. 2015.
- [16] Fritzing. [Online]. Available: <http://www.fritzing.org/home>, Accessed on: Nov. 2015.
- [17] Kcad. [Online]. Available: <http://www.kicad-pcb.org>, Accessed on: Nov. 2015.
- [18] KtechLab. [Online]. Available: <https://github.com/ktechlab/ktechlab/wiki>, Accessed on: Nov. 2015.
- [19] Xcircuit. [Online]. Available: <http://opencircuitdesign.com/xcircuit>, Accessed on: Nov. 2015.
- [20] S. R. Whiteley. [Online]. Available: <http://www.wrcad.com/ftp/pub/jjmodel.c>, Accessed on: Nov. 2015.
- [21] S. R. Whiteley, "Josephson junctions in SPICE3," *IEEE Trans. Magn.*, vol. 27, no. 2, pp. 2902–2905, Mar. 1991.
- [22] J. E. Fang and T. V. Duzer, "A Josephson integrated circuit simulator (JSIM) for superconductive electronics application," in *Proc. Extended Abstr. 2nd Int. Supercond. Electron. Conf.*, 1989, pp. 407–410.
- [23] GNUplot. [Online]. Available: <http://www.gnuplot.info/>, Accessed on: Sep. 2015.
- [24] [Online]. Available: <http://wrcad.com/freestuff.html>, Accessed on: Aug. 2016.
- [25] S. Polonsky, V. Semenov, and P. Shevchenko, "PSCAN: Personal superconductor circuit analyser," *Supercond. Sci. Technol.*, vol. 4, no. 11, pp. 667–670, 1991.
- [26] S. Polonsky, P. Shevchenko, A. Kirichenko, D. Zinoviev, and A. Rylyakov, "PSCAN'96: New software for simulation and optimization of complex RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 7, no. 2, pp. 2685–2689, Jun. 1997.
- [27] V. Semenov, personal communication, Jul. 2016.
- [28] A. Dewes, "A tool to simulate superconducting circuits, comparable to spice." [Online]. Available: <https://github.com/adewes/superconductor>, Accessed on: Jul. 2016.
- [29] Linear Technology Spice. [Online]. Available: <http://www.linear.com/designtools/software>, Accessed on: Oct. 2015.
- [30] Q. P. Kerr and M. J. Feldman, "Multiparameter optimization of RSFQ circuits using the method of inscribed hyperspheres," *IEEE Trans. Appl. Supercond.*, vol. 5, no. 2, pp. 3337–3340, Jun. 1995.
- [31] T. Harnisch, J. Kunert, H. Toepfer, and H. F. Uhlmann, "Design centering methods for yield optimization of cryoelectronic circuits," *IEEE Trans. Appl. Supercond.*, vol. 7, no. 2, pp. 3434–3437, Jun. 1997.
- [32] M. Jeffery, W. J. Perold, Z. Wang, and T. van Duzer, "Monte Carlo optimization of superconducting complementary output switching logic circuits," *IEEE Trans. Appl. Supercond.*, vol. 8, no. 3, pp. 104–119, Sep. 1998.
- [33] Magic VLSI Layout Tool. [Online]. Available: <http://opencircuitdesign.com/magic/>, Accessed on: Nov. 2015.
- [34] Graal Layout Editor. [Online]. Available: http://www.vlsitechnology.org/html/linux_help3.html, Accessed on: Nov. 2015.
- [35] Toped IC Layout Editor. [Online]. Available: <http://www.toped.org.uk/>, Accessed on: Nov. 2015.
- [36] K. Keller, "Kic : A graphical editor for integrated circuit." [Online]. Available: <http://www.wrcad.com/freestuff.html>, Accessed on: Jul. 2016.
- [37] Layout System for Individuals. [Online]. Available: <http://www.lasihomesite.com/>, Accessed on: Jul. 2015.
- [38] P. M. Xiao, E. Charbon, A. Sangiovanni-Vincentelli, T. Van Duzer, and S. R. Whiteley, "INDEX: An inductance extractor for superconducting circuits," *IEEE Trans. Appl. Supercond.*, vol. 3, no. 1, pp. 2629–2632, Mar. 1993.
- [39] R. S. Bakolo and C. J. Fourie, "Development of a RSFQ cell library for the University of Stellenbosch," in *Proc. AFRICON*, 2011, pp. 1–5.
- [40] P. I. Bunyk and S. V. Rylov, "Automated calculation of mutual inductance matrices of multilayer superconductor integrated circuits," in *Proc. Extended Abstr. Int. Supercond. Electron. Conf.*, 1993, vol. 93, p. 62.
- [41] M. M. Khapaev, A. Y. Kidiyarova-Shevchenko, P. Magnelind, and M. Y. Kupriyanov, "3D-MLSI: Software package for inductance calculation in multilayer superconducting integrated circuits," *IEEE Trans. Appl. Supercond.*, vol. 11, no. 1, pp. 1090–1093, Mar. 2001.
- [42] M. Kamon, M. J. Tsuk, and J. K. White, "FastHenry: A multipole-accelerated 3-D inductance extraction program," *IEEE Trans. Microw. Theory Techn.*, vol. 42, no. 9, pp. 1750–1758, Sep. 1994.
- [43] Inductex. [Online]. Available: <http://www0.sun.ac.za/ix/?q=home>, Accessed on: Jan. 2016.
- [44] S. Yorozu, Y. Kameda, H. Terai, A. Fujimaki, T. Yamada, and S. Tahara, "A single flux quantum standard logic cell library," *Physica C, Supercond.*, vol. 378, pp. 1471–1474, 2002.
- [45] C. L. Ayala, "Energy-efficient wide datapath integer arithmetic logic units using superconductor logic," Ph.D. dissertation, Stony Brook Univ., Stony Brook, NY, USA, Dec. 2012.
- [46] S. Intiso, I. Kataeva, E. Tolkacheva, H. Engseth, K. Platov, and A. Kidiyarova-Shevchenko, "Time-delay optimization of RSFQ cells," *IEEE Trans. Appl. Supercond.*, vol. 15, no. 2, pp. 328–331, Jun. 2005.
- [47] N. Maguu and C. J. Fourie, "Modeling of AQFP logic gates with HDL using multivalued logic approach," in *Proc. Int. Supercond. Electron. Conf.*, Nagoya, Japan, Jun. 2015, DS-P14.
- [48] Icarus Verilog. [Online]. Available: <http://iverilog.icarus.com/>, Accessed on: Nov. 2015.
- [49] FreeHDL. [Online]. Available: <http://freehdl.seul.org/>, Accessed on: Nov. 2015.
- [50] Verilator Verilog Simulator. [Online]. Available: <http://www.veripool.org/projects/verilator/wiki/Installing>, Accessed on: Nov. 2015.
- [51] N. Yoshikawa, H. Tago, and K. Yoneyama, "A new design approach for RSFQ logic circuits based on the binary decision diagram," *IEEE Trans. Appl. Supercond.*, vol. 9, no. 2, pp. 3161–3164, Jun. 1999.
- [52] N. Yoshikawa and J. Koshiyama, "Top-down RSFQ logic design based on a binary decision diagram," *IEEE Trans. Appl. Supercond.*, vol. 11, no. 1, pp. 1098–1101, Mar. 2001.
- [53] J. Koshiyama and N. Yoshikawa, "A cell-based design approach for RSFQ circuits based on binary decision diagram," *IEEE Trans. Appl. Supercond.*, vol. 11, no. 1, pp. 263–266, Mar. 2001.
- [54] S. Yamashita, K. Tanaka, H. Takada, K. Obata, and K. Takagi, "A transduction-based framework to synthesize RSFQ circuits," in *Proc. Asia South Pac. Conf. Des. Autom.*, 2006, pp. 43–48.

- [55] C. Wolf and J. Glaser, "Yosys open synthesis suite," in *Proc. Autochip*, 2013.
- [56] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin II—An open-source Verilog HDL synthesis tool for CAD research," in *Proc. 18th IEEE Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2010, pp. 149–156.
- [57] R. Roberts and C. J. Fourie, "Layout-to-schematic as a step towards layout-versus-schematic verification of SFQ integrated circuit layouts," in *Proc. AFRICON*, 2013, pp. 1–5.

Maguu N. Muchuka received the B.Sc. degree in physics, maths, and electronics from Dr. Babasaheb Ambedkar Marathwada University, Aurangabad, India, in 1998, and the M.Sc. degree in electronics from the University of Mysore, Mysore, India, in 2000. He is currently working toward the Ph.D. degree in electronics engineering at Stellenbosch University, Stellenbosch, South Africa.

Since 2007, he has been a Lecturer in Egerton University, Njoro, Kenya. His research interests include electronic design automation tools design, superconducting electronics, and micro- and nanosystems.

Johannes A. Delpport received the bachelor's degree in electronics engineering, in 2014, from Stellenbosch University, Stellenbosch, South Africa, where he is currently working toward the master's degree in electronics engineering.

His research interests include superconducting electronics, digital systems simulation, and superconducting software design.

Coenrad Johann Fourie (M'01) received the B.Eng. degree in electronics engineering and the Ph.D. degree in superconductive electronics from Stellenbosch University, Stellenbosch, South Africa, in 1998 and 2003, respectively.

In 2001, he joined Stellenbosch University as a Lecturer, where he is currently a Professor in the Department of Electrical and Electronic Engineering. His research interests include superconducting electronics, superconducting quantum interference device sensor applications, and the development of parameter extraction software for superconductive integrated circuits.

Appendix B

Journal Paper - Analysis of a Shielding Approach for Magnetic Field Tolerant SFQ Circuits

- R. S. Bakolo, J. A. Delport, P. Febvre and C. J. Fourie, "Analysis of a Shielding Approach for Magnetic Field Tolerant SFQ Circuits," *IEEE Transactions on Applied Superconductivity*, 2017.[23]

This paper discusses the effect of magnetic fields in various directions and how shielding can be effectively applied to minimize the effect of these fields. All the credit for this paper goes to Dr Bakolo for his excellent work. Personal contributions are in the form of simulations and margin calculations using in-house tools developed in the process of attaining this degree. Copyright for this paper is held by IEEE Transactions on Applied Superconductivity.

Analysis of a Shielding Approach for Magnetic Field Tolerant SFQ Circuits

Rodwell S. Bakolo, Johannes A. Delpont, Pascal Febvre, *Member, IEEE*, and Coenrad J. Fourie, *Member, IEEE*

Abstract—The operating margins of unshielded SFQ circuits are influenced by external magnetic fields, and earlier research showed experimental results of operating region versus bias current for circuits with in-plane and perpendicularly applied magnetic fields. Here, we report a method that can be used to analyze shields to protect SFQ circuits from external magnetic fields. To validate the approach, we investigated a grid-patterned shield of varying spacing. The analysis was done with cell layouts made according to the *Hypres*' 4.5 kA/cm² process, in which the top-most layer, M3, was used to implement the shields. It was calculated that a grid shield of 2.5 μm grid bar width and spacing of 5 μm offered a good compromise at both providing shielding and causing a relatively small drift in circuit inductance. In order to make SFQ circuits more tolerant to magnetic fields, we have simulated with circuit parameter alterations to realize the best bias and higher operating field margins, due to external magnetic fields. The external magnetic fields are modeled through three orthogonal coils that generate roughly a uniform magnetic field density throughout the cell under test.

Index Terms—SFQ, magnetic fields, shielding, magnetic field tolerant.

I. INTRODUCTION

SINGLE flux quantum (SFQ) based electronics are known to have high switching speeds and ultra low energy consumption figures [1]. However, SFQ circuits suffer operating margin drift and failure when exposed to magnetic fields as low as 15 μT [2], [3]. Further, very large scale integration issues are often impeded as complex circuits require large bias currents. In large circuits, the bias currents can reach several amperes [3]–[5]. These currents create design challenges including heating in feed lines, but of great concern are the magnetic fields generated by these currents. These fields, together with those from external sources, such as the Earth, further degenerate the mostly narrow operating margins of SFQ circuits.

Test results have shown fields parallel to SFQ circuit's plane have more negative impact on the operating margins than perpendicular ones [2]. It is therefore, imperative that SFQ circuits are shielded from external magnetic fields, especially

Manuscript received September 6, 2016; accepted February 9, 2017. Date of publication February 15, 2017; date of current version February 24, 2017. This work was supported in part by the French-South African Partenariat Hubert Curien PROTEA under Grant 33944VG, in part by the South African National Research Foundation under Grant 95237, and in part by the Malawi Government.

R. S. Bakolo, J. A. Delpont, and C. J. Fourie are with the Department of Electrical and Electronic Engineering, Stellenbosch University, Stellenbosch 7600, South Africa (e-mail: coenrad@sun.ac.za).

P. Febvre is with the Université Savoie Mont Blanc, IMEP-LAHC (CNRS UMR5130), Le Bourget du Lac 73376, France.

Digital Object Identifier 10.1109/TASC.2017.2669646

parallel ones, and bias lines. The concept of shielding is not new, but only the standard solid, continuous superconducting layer, shield is reported [6]–[8]. Inductance of striplines between multiple ground planes, where the shielding layer forms a top ground plane, is lower than for microstrip lines above a single ground plane. For solid shield layer layouts, inductance thus has to be adapted, resulting in more area consumed by inductors. A shielding mechanism that does not significantly decrease inductance of circuit elements would thus be beneficial to allow smaller layouts. Furthermore, solid shield layers increase the probability of flux trapping in the layer metal, so that a shield should consist of structures that minimize flux trapping probability. Hence, a shielding mechanism that offers a compromise while providing efficient shielding is a necessity. Three orthogonal coils were modelled to generate uniform magnetic fields onto an SFQ circuit, as shown in [9], to establish the effectiveness of different shielding approach configurations.

We present a method that can be used to analyse the effectiveness of on-chip magnetic shields for SFQ circuits. The method was validated through simulations that establish the best possible shield configuration. Efficient shielding reduces the vulnerability of SFQ circuits to external magnetic fields and hence a first step toward magnetic field tolerant SFQ design.

II. SHIELDING CONCEPT

The use of superconductors as shields is not new and it has been reported by numerous authors such as [10], [11]. However, most of the work does not apply to SFQ circuits. There is a huge reliance on Meissner effect to achieve perfect magnetic field exclusion. Niobium, the main superconductor in low-Tc SFQ circuits, is a Type II superconductor and therefore perfect diamagnetism is non-existent. An infinite ratio of the film thickness, d , to the London penetration depth, λ , is required for an ideal shield. However, the films used in SFQ circuits are very thin and therefore the ratio is always finite. For instance, the M3 layer in the *Hypres* process [12], has a thickness of $d = 600$ nm and penetration depth of $\lambda = 90$ nm. The ratio $\frac{d}{\lambda} = 6.67$. This reveals that shielding in SFQ is never perfect as $\approx 10\%$ of the magnetic field will still find its way through the shield [6], [13]–[15]. The aim of designing on-chip shields is to reduce mutual inductance that can exist between external magnetic field producing entity and circuit inductance, hence reducing coupling.

Grounding of on-chip shields is crucial. It was verified in [7], [8], that shields grounded at strategic positions improved the shielding efficiency. Shielding results in resultant surface flow

of currents and the ground contacts improve the flow. However, placement of ground contacts works better in less complex structures and the layout designer may only place them as the layout permits in larger, complex circuits. The concept of grounding and the resulting efficiency can be described using the effective inductance. This is because grounding reduces the effective inductance of the shielding layer, which results in improved flow of surface currents. In addition, the coupling between the shield and circuit reduces in a grounded shield thereby improving the shield's efficiency.

III. SHIELDING APPROACH

A. Simulation Tools

Outlined next are software tools that were used to analyse the shielding approach presented in this paper. SFQ cells were laid out in Layout System for Individuals (LASI) [16], from which GDSII format binary files were generated. Three orthogonal coils, that produce uniform magnetic fields on-chip, were modelled in *InductEx* [17], [18] to produce roughly uniform magnetic fields of any required orientation. All circuit inductances and couplings with the coils, calculated again in *InductEx*, were used in circuit simulations. Schematic capture and parameterisation were done in gEDA [19], while Josephson Simulator (JSIM) [20] was used for transient simulations of the SFQ circuits. To obtain operating margins in the circuits, *analyse*, which is part of our in-house tool-set was used.

B. A Grid Patterned Shield

The shielding analysis presented here was done with layouts made with the *Hypres'* 4.5 kA/cm^2 process design rules [12]. The process' top-most layer, M3, which has technology fixed thickness $d = 600 \text{ nm}$ and penetration depth $\lambda = 90 \text{ nm}$ was used to implement on-chip shields.

The use of orthogonal coils, shown in Fig. 1, envisages that magnetic fields, as vectors, can take any orientation. Therefore, the circuit needs to be protected against that. The magnetic field orientation, dictated by positive or negative coil currents, remains constant and fixed for each simulation described here. Although the field vector was not swept over all directions, which would be time intensive for small sweep steps, it does contain equal components in each axial direction and thus allows us to determine shielding efficiency in the presence of all axial components. A grid patterned shield, which resembles a Faraday's cage, was simulated with Delay Flip-Flop (DFF) and Direct Current to SFQ (DC-SFQ) converter cells. The grid shield offers similar shielding protection to the normal solid shield, with an added advantage that it affects circuit inductance less [21]. In our earlier simulations, only a grounded grid shield (in Fig. 2) of $2.5 \mu\text{m}$ grid bar width and spacing of $5 \mu\text{m}$ was considered [21] and hereby referred to as the standard grid shield. In order to obtain the best possible grid configuration, we considered seven grid spacing configurations. All coupling coefficients between coils and circuit inductance were recorded and used in margin analysis simulations while the coil currents were gradually increased, at each stage. For each SFQ cell, there is a

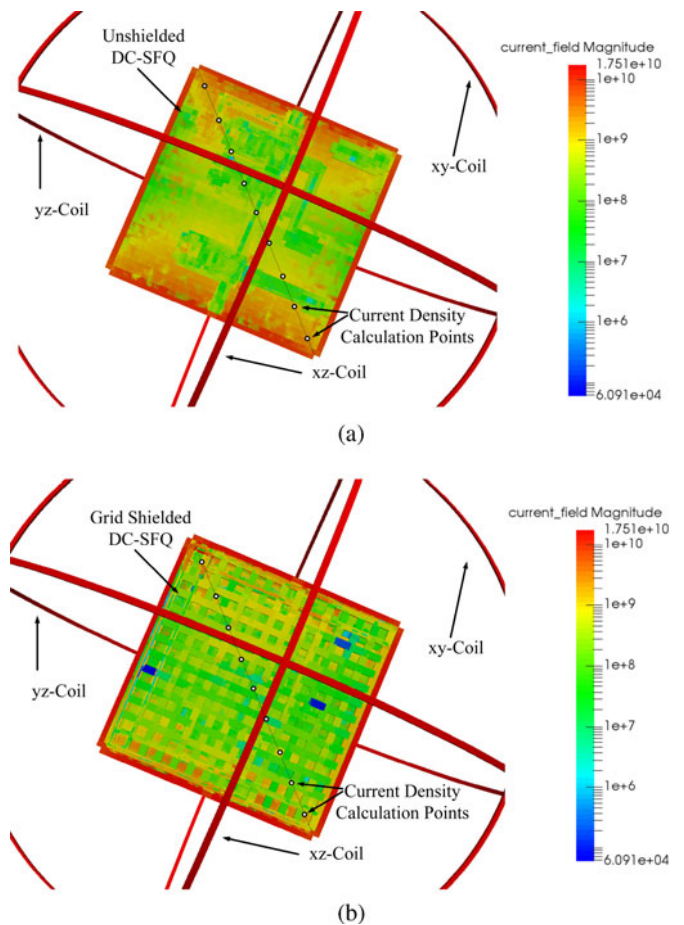


Fig. 1. Current density plots of a DC-SFQ surrounded by 3 orthogonal coils, (a) unshielded and (b) under a standard grid shield. The current density calculations were done all around the cell, however only the top part is shown here. The current density is highest in the coils (red) and affects the cell due to coupling. The current density plots were generated with *Magix*, a visualization tool for *InductEx*, which produces files viewable with *ParaView* [22]. In the simulations, a 1 volt sinusoidal voltage at 10 GHz is applied to each coil, which results in a maximum current density of $\approx 1.75 \times 10^{10} \text{ A/m}^2$. Radii of coils are $R_{xy} = 125 \mu\text{m}$, $R_{yz} = 130 \mu\text{m}$ and $R_{xz} = 135 \mu\text{m}$. To further validate the effectiveness of the grid shield, current densities were calculated at the indicated points.

point at which the shielding no longer offers protection, called an operating field margin (OFM) or failure point. The OFM of an SFQ cell is hereby defined as the point where the circuit parameter and bias margins diminish to 0%. The varied currents represent different magnetic field density magnitudes the circuit could be exposed to.

In addition, the SFQ cells were simulated with field coil currents that vary both negatively and positively, thereby effectively reversing the magnetic field orientation in each coil and fix it for each simulation. This approach produced two OFMs, one for the positive field and the other for the negative field. These OFMs cannot be necessarily the same as the circuit inductance orientation and position can influence the results. This is a thorough approach and remains indicative at best, but a good shield has to provide protection from all possible field vectors. At any given time, all coils have the same amount of current and therefore

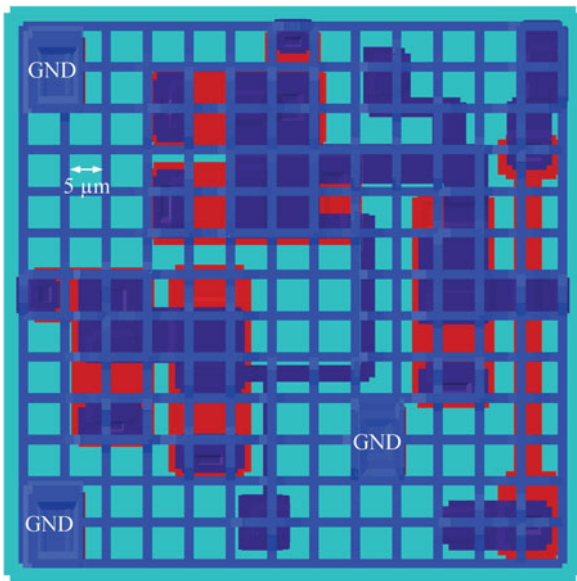


Fig. 2. A DFF covered in a grid patterned shield - each grid bar is $2.5 \mu\text{m}$ wide and are spaced $5 \mu\text{m}$ from each other. The layout was done in the *Hyppres'* 4.5 kA/cm^2 , in which the layer, M3, with fixed thickness $d = 600 \text{ nm}$ was used to make the grid shield. The ground contacts, marked GND, were placed to provide the best possible shielding and lowest drop on circuit inductance. To avoid major drops in circuit inductance grounding must be laid out in a line that crosses circuit inductance at 90° or any angle as the layout permits, but not in parallel and close to major circuit inductance.

the vector sum of the resulting fields has an inclination, $\theta \approx 55^\circ$ and azimuth, $\varphi \approx 45^\circ$ in a spherical coordinate system.

Fig. 1 shows current density plots of a DC-SFQ cell surrounded by 3 orthogonal coils, for both the unshielded and standard grid shielded cases. The red colour shows the highest current density, while blue is lowest. With the aid of the color field magnitude key in Fig. 1, it can be observed from the color variations that the grid shield reduces the density on the DC-SFQ cell that occurs due to coupling with the 3 coils. By calculating current densities at the nine points shown in Fig. 1, on both the unshielded and shielded DC-SFQ cells, we can show that the grid shield reduces the current density by 40%. The grid shield was only implemented in the top-most layer of the *Hyppres'* 4.5 kA/cm^2 fabrication process for our simulations described here. This approach leaves the ground plane unmodified to avoid other design challenges that might arise because of additional changes of circuit inductances.

IV. ANALYSIS OF THE GRID PATTERNED SHIELD

In this analysis, simulations for both the DFF and DC-SFQ were conducted to determine the effect of grid bar spacing on shielding effectiveness. Grid bar spaces of 0, 2.5, 4, 5, 6, 7.5, 9 and $10 \mu\text{m}$ were used in the analysis. The bar width of each grid was fixed to $2.5 \mu\text{m}$ for all grid spaces. An example of grid shield is shown in Fig. 2. At this point, each grid shield configuration was simulated and OFMs recorded for each cell. The results are summarised in Fig. 3, and it can be observed that the shielding effectiveness reduces with increase in spacing between grid bars. So the smaller the spaces, the better the shield. As the spaces become wider, coupling between circuit

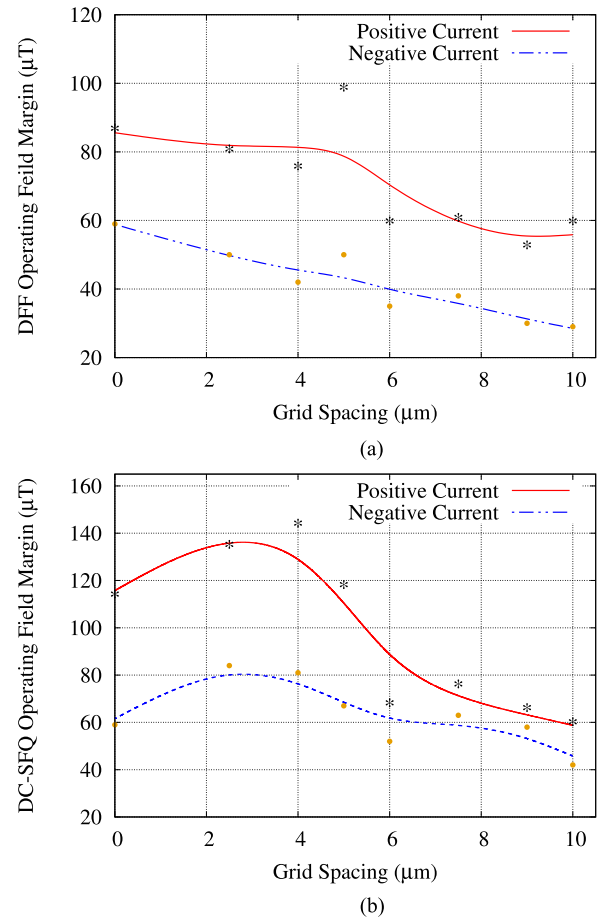


Fig. 3. The variation of shielding effectiveness with changing grid spacing - A grid shield with smaller spaces offers better shielding, but the downside is that the circuit inductance are affected to the point that re-optimisation of the entire circuit might be necessary and vice-versa. Positive and negative currents refer to fields from coils.

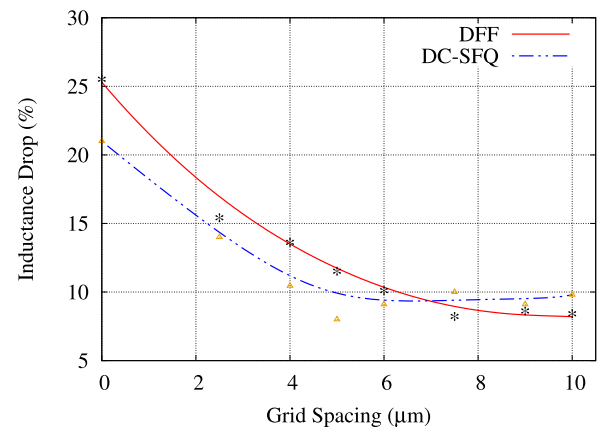


Fig. 4. Percentage inductance drop against variations in grid spacing in both the DFF and DC-SFQ.

and external coils increases, which leads to poorer shielding. However, it can also be noted from Fig. 4, that the overall influence on circuit inductance reduces when increasing the spacing of the grid bars from 0 to $7.5 \mu\text{m}$. Any further increase in spacing beyond $7.5 \mu\text{m}$ produces no further drop in circuit inductance. It can be concluded that even the widest possible

grid bar spacing can cause a drop in circuit inductance of about 10% (see Fig. 4). It is at the discretion of the circuit designer on how much magnetic shielding can be applied without sacrificing parameter and bias margins due to the drop in circuit parameter values.

The OFMs in Fig. 3 can be compared to unshielded cases for both cells. The OFMs for the unshielded DFF were $46 \mu\text{T}$ and $38 \mu\text{T}$ [21] for positive and negative currents, respectively, while for the unshielded DC-SFQ the OFMs were $63 \mu\text{T}$ and $38 \mu\text{T}$ for positive and negative currents, respectively.

A grid shield of $2.5 \mu\text{m}$ wide bars and $5 \mu\text{m}$ spacing offers the best compromise at OFMs of $98 \mu\text{T}$ for positive current and $50 \mu\text{T}$ for negative current (refer to Fig. 3(a)) in the DFF. Whereas for the DC-SFQ under the same grid shield dimensions produced OFMs of $117 \mu\text{T}$ and $67 \mu\text{T}$ for positive and negative current respectively (see Fig. 3(b)). In comparison, the OFMs for solid shield covered cells were $86 \mu\text{T}$ and $50 \mu\text{T}$ [21] in the DFF and $113 \mu\text{T}$ and $59 \mu\text{T}$ in the DC-SFQ.

The percentage inductance drop for the grid shield with $2.5 \mu\text{m}$ grid bar width and $5 \mu\text{m}$ spacing are 8% and 11% for the DC-SFQ and DFF respectively, as shown in Fig. 4. In comparison, the drop is quite high when a top solid shield is used with 21% in the DC-SFQ and 25% in the DFF. Such high variations can negatively impact operating margins if redesigning and optimization are not done. However, the parameter variations for the grid shield with $2.5 \mu\text{m}$ grid bar width and $5 \mu\text{m}$ spacing showed to have minimal effect on operating margins of the cells. Another advantage of the grid shield is that the bar width has been chosen narrow enough so that trapped magnetic flux, from which originate some noise and digital malfunctions, cannot occur, unlike for the case of a solid shield. Another advantage of the grid shield is that the bar width has been chosen narrow enough so that magnetic flux trapping, from which originate some noise and digital malfunctions, is unlikely to occur in the bars or the grid holes (see [23] Section 5.4, pp. 43-45), unlike for the case of a solid shield.

V. MAGNETIC FIELD TOLERANT DESIGN

Apart from shielding, SFQ circuits can be made more immune to magnetic fields likely to remain inside shielded cryo-cooled environments. We investigated the effect of altering selected circuit parameters, such as Josephson junctions' critical currents and inductance values. Schematics, optimised parameter values and bias margins for the DFF and DC-SFQ are shown in Figs. 5 and 6, respectively.

Most SFQ cells have shown to recover operation, with broader OFMs, by adjusting bias currents to compensate for the induced currents from external magnetic fields. This approach is mostly not practical due to the nature of biasing systems for SFQ circuits. It is not easy to adjust the bias current for a single cell in a large circuit, without affecting the rest. A better approach is to alter selected circuit parameters. Margin analysis results of a circuit close to failure Red show what parameters can be Red altered in order to recover functionality. In Table I, the *Initial OFMs* are the ones shown in Fig. 3 for both the grid (at $5 \mu\text{m}$ spacing) and the solid (at $0 \mu\text{m}$ spacing) shields.

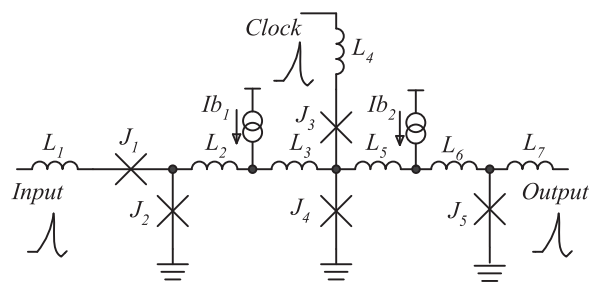


Fig. 5. A 5-Josephson junction DFF with parameters: $L_1 = 1.86 \text{ pH}$, $L_2 = 1.59 \text{ pH}$, $L_3 = 7.73 \text{ pH}$, $L_4 = 1.5 \text{ pH}$, $L_5 = 2.13 \text{ pH}$, $L_6 = 1.3 \text{ pH}$, $L_7 = 1.91 \text{ pH}$, $J_1 = J_4 = 200 \mu\text{A}$, $J_2 = J_5 = 250 \mu\text{A}$, $J_3 = 150 \mu\text{A}$, $I_{b1} = 230 \mu\text{A}$ and $I_{b2} = 135 \mu\text{A}$. Optimised bias margins: $-53\% \sim 42\%$.

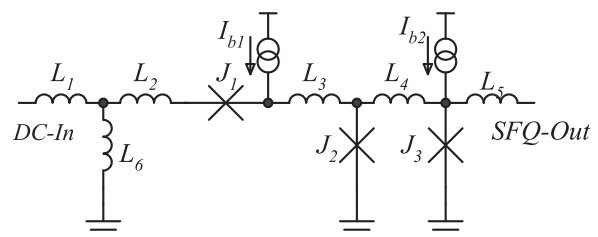


Fig. 6. A DC-SFQ with parameters: $L_1 = 0.56 \text{ pH}$, $L_2 = 0.52 \text{ pH}$, $L_3 = 1.0 \text{ pH}$, $L_4 = 4.78 \text{ pH}$, $L_5 = 2.2 \text{ pH}$, $L_6 = 4.1 \text{ pH}$, $J_1 = 225 \mu\text{A}$, $J_2 = 225 \mu\text{A}$, $J_3 = 250 \mu\text{A}$, $I_{b1} = 275 \mu\text{A}$ and $I_{b2} = 175 \mu\text{A}$. Optimised bias margins: $-56\% \sim 34\%$.

TABLE I
NEW OPERATING FIELD MARGINS (OFMs) FOR THE DFF AND DC-SFQ AFTER SELECTED PARAMETERS WERE ALTERED

Cell	Initial OFMs	Altered Parameters	New OFMs
DFF - Grid	98 & $50 \mu\text{T}$	$L_5 = 2.6 \text{ pH}$	126 & $42 \mu\text{T}$
DFF - Solid	86 & $50 \mu\text{T}$	$L_5 = 2.6 \text{ pH}$	101 & $32 \mu\text{T}$
DC-SFQ - Grid	117 & $67 \mu\text{T}$	$L_6 = 3.5 \text{ pH}$ $J_1 = 200 \mu\text{A}$ $J_3 = 275 \mu\text{A}$	134 & $67 \mu\text{T}$
DC-SFQ - Solid	113 & $59 \mu\text{T}$	$L_6 = 3.5 \text{ pH}$ $J_1 = 200 \mu\text{A}$ $J_3 = 275 \mu\text{A}$	123 & $49 \mu\text{T}$

OFMs presented for positive & negative currents, respectively.

The *New OFMs* were calculated after selected parameters were altered in both cells. In the DFF, only the value of L_5 was changed, while in DC-SFQ, the values of L_6 , J_1 and J_3 were changed (refer to Figs. 5 and 6 and Table I). Simulations for both grid and solid shielded DFF and DC-SFQ cells showed improvement in OFMs, especially for positive currents, as summarised in Table I. With this approach, coupled with good shielding, grid or solid, SFQ circuits can be made more tolerant to magnetic fields of specific orientation.

VI. CONCLUSION

A method that can be used to analyse and validate magnetic shielding for SFQ circuits has been presented. Further, we have shown the best possible configuration for the grid-patterned shield for SFQ circuits. The problem with ordinary solid shield is

that it greatly reduces the values of circuit inductance and further optimisation is always required to recover circuit operation. We have shown that a grid-patterned shield with $2.5\ \mu\text{m}$ grid bar width and spacing of $5\ \mu\text{m}$ offers good shielding and affects the circuit inductance less. It has also been shown that SFQ circuits fail at different levels of external fields due to varying orientations of external magnetic fields. As such, shielding has to be made with the most critical orientation in mind. We have further shown that selected circuit parameters can be altered to make SFQ circuits more tolerant to external magnetic fields.

ACKNOWLEDGEMENTS

The authors would like to thank Ruben van Staden for the assistance rendered in validating the grid shields with *Magix*.

REFERENCES

- [1] 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, no. 1, pp. 3–28, Mar. 1991.
- [2] R. Collot, P. Febvre, J. Kunert, and H. Meyer, "Operation of low-Tc circuits in a magnetic environment," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, Jun. 2013, Art. no. 1700404.
- [3] M. Suzuki, M. Maezawa, and F. Hirayama, "Effects of magnetic fields induced by bias currents on operation of RSFQ circuits," *Physica C, Supercond.*, vol. 412–414, pp. 1576–1579, 2004.
- [4] H. Terai, Y. Kameda, S. Yorozu, A. Fujimaki, and Z. Wang, "The effects of dc bias current in large-scale SFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 13, no. 2, pp. 502–506, Jun. 2003.
- [5] E. Tolkacheva, H. Engseth, I. Kataeva, and A. Kidiyarova-Shevchenko, "Influence of the bias supply lines on the performance of RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 15, no. 2, pp. 276–279, Jun. 2005.
- [6] Y. Mizugaki, R. Kashiwa, M. Moriya, K. Usami, and T. Kobayashi, "Grounding positions of superconducting layer for effective magnetic isolation in josephson integrated circuits," *J. Appl. Phys.*, vol. 101, no. 11, 2007, Art. no. 114509.
- [7] Y. Mizugaki and R. Kashiwa, "Magnetic shielding effect of grounded superconducting niobium layers," *J. Phys., Conf. Series*, vol. 97, no. 1, 2008, Art. no. 012056.
- [8] Y. Mizugaki, R. Kashiwa, A. Kawai, M. Moriya, K. Usami, and T. Kobayashi, "Magnetic isolation enhanced by a superconducting loop in josephson integrated circuits," *Jpn. J. Appl. Phys.*, vol. 48, no. 7R, 2009, Art. no. 073001.
- [9] R. S. Bakolo and C. J. Fourie, "Modelling of the influence of magnetic fields on the operation of digital superconductive circuits," in *Proc. 2015 15th Int. Supercond. Electron. Conf.*, 2015, pp. 1–3.
- [10] K. H. Carpenter, "Magnetostatic simulations for design of superconducting magnetic shields," *IEEE Trans. Appl. Supercond.*, vol. 6, no. 3, pp. 142–146, Sep. 1996.
- [11] L. Gozzelino, R. Gerbaldo, G. Ghigo, F. Laviano, and M. Truccato, "Comparison of the shielding properties of superconducting and superconducting/ferromagnetic bi- and multi-layer systems," *J. Supercond. Novel Magn.*, pp. 1–8, Aug. 2016.
- [12] Hypres, "Niobium integrated circuit fabrication: Design rules rev 24," Hypres, Tech. Rep., Jan. 11, 2008. [Online]. Available: <http://www.hypres.com>
- [13] Y. Mizugaki *et al.*, "Magnetic isolation on a superconducting ground plane," *Jpn. J. Appl. Phys.*, vol. 38, no. 10R, 1999, Art. no. 5869.
- [14] Y. Mizugaki, H. Hakii, M. Moriya, K. Usami, and T. Kobayashi, "Mutual inductance coupled through superconducting thin film in niobium josephson integrated circuits," *Jpn. J. Appl. Phys.*, vol. 44, no. 6L, 2005, Art. no. L763.
- [15] K. Suzuki, S. Yorozu, Y. Kameda, and K. Tanabe, "Investigation of magnetic flux state in nb SFQ circuits by scanning squid microscope," *Physica C, Supercond. Appl.*, vol. 445, pp. 1034–1036, 2006.
- [16] "Lasi," Tech. Rep. [Online]. Available: <http://www.lasihomesite.com/>
- [17] "Inductex: Inductex v5.03," Stellenbosch Univ., Stellenbosch, South Africa, Tech. Rep., 2016. [Online]. Available: <http://www0.sun.ac.za/ix/>
- [18] C. J. Fourie, O. Wetzstein, J. Kunert, H. Toepfer, and H.-G. Meyer, "Experimentally verified inductance extraction and parameter study for superconductive integrated circuit wires crossing ground plane holes," *Supercond. Sci. Technol.*, vol. 26, no. 1, 2013, Art. no. 015016.
- [19] gEDA, "GNU's general public licence—Electronic design automation," Tech. Rep. [Online]. Available: <http://www.geda-project.org/>
- [20] J. E. Fang and T. V. Duzer, "A Josephson integrated circuit SIMulator (JSIM) for superconductive electronics application," in *Proc. Extended Abstracts 2nd Int. Supercond. Electron. Conf.*, 1989, pp. 407–410.
- [21] R. S. Bakolo, R. Staden, P. Febvre, and C. J. Fourie, "Modelling magnetic fields and shielding efficiency in superconductive integrated circuits," *J. Supercond. Novel Magn.*, pp. 1–5, Sep. 2016.
- [22] ParaView. [Online]. Available: <http://www.paraview.org/>
- [23] M. Schmelz, "Development of a high sensitive receiver system for transient electromagnetics," Univ. Twente, Enschede, The Netherlands, 2014.

Appendix C

Journal Paper - A Static Timing Analysis tool for RSFQ and ERSFQ Superconducting Digital Circuit applications

- J. A. Delpont and C. J. Fourie, "A Static Timing Analysis Tool for RSFQ and ERSFQ Superconducting Digital Circuit Applications," *IEEE Transactions on Applied Superconductivity*, 2018.[24]

SuperSTA is introduced in this paper as a static timing analysis tool for superconducting circuits. Examples are executed to demonstrate the capabilities and drawbacks fo the tool. All contributions are my own and copyright for this paper is held by IEEE Transactions on Applied Superconductivity.

A Static Timing Analysis Tool for RSFQ and ERSFQ Superconducting Digital Circuit Applications

Johannes A. Delpoort  and Coenrad J. Fourie , *Member, IEEE*

Abstract—Static timing analysis in the rapid-single-flux-quantum and energy-efficient RSFQ superconducting digital circuit domain is yet to be achieved in a generic sense. A static timing analysis tool is proposed here for preplacement designs as well as postplaced and routed designs. Preplaced static timing analysis attempts to find a general gauge for the performance of a JSIM/SPICE netlist in the absence of information on wiring delays. The postplaced and routed static timing analysis provides a more accurate analysis of a design in the cadence design exchange format. This design file should include all the wire lengths as well as a complete clocking scheme. Results of this postplaced and routed static timing analysis are then used to determine the maximum clock speed that the design can be run at to prevent timing violations. Results for both methods of analysis are presented and then incorporated in an hardware description language representation of the design to show that there are no timing violations at the presented clock speed. We show the implementation of a static timing analysis method developed as a precursor to the IARPA SuperTools project, and how it applies to circuits with H-tree and HL-tree clocking schemes.

Index Terms—SFQ clock frequency, superconducting integrated circuits, superconducting logic circuits, timing analysis, wiring delay.

I. INTRODUCTION

STATIC timing analysis (STA) is a technique that is used to provide an estimation of the expected timing (and power) of a digital circuit without the requirement for simulation. The number of timing paths (input to any output) increases as an exponential function with respect to the number of logic gates in the circuit. Therefore, it is impractical to perform a full-chip simulation at the electrical level. Timing information about a circuit is a crucial part of the standard cell-based design flow, and for the analysis of rapid-single-flux quantum (RSFQ)-based superconducting digital circuits, we split this task in two parts, namely pre-placed STA and post-place-and-route STA. To ensure that the results of the analysis remains accurate, we rely on the accurately characterized timing information of a standard cell library.

Previous work has shown that single flux quantum (SFQ) cells are very different to semiconductor (CMOS) cells in terms of

delay and power consumption [1], [2]. For this work, we considered only RSFQ [3], [4] and energy-efficient RSFQ (ERSFQ) [5] logic families, for which the differences in STA analysis compared to CMOS are largely due to the requirement that all logic cells in these families need to be clocked. This introduces a challenge in terms of clock tree design that needs to be considered in the design of the STA tool. Difficulties related to modeling timing behavior inside SFQ logic cells have already been outlined earlier [6], [7], especially in terms of the strong dependence of timing values on circuit parameters such as bias current, and on dynamic conditions such as the state of a cell and the state of its neighboring cells. Here we investigate the timing paths through a design under the assumption that a cell library has been characterized at the required bias current, and that timing values are listed as worst-case over all states. In our work, which assumes a row-based large-scale placement and routing strategy [8], the timing parameters of a cell are not affected by the state of a neighboring cell, because all cells are connected through passive transmission lines. Small resistances in series with transmission lines prevent flux trapping, but conveniently also isolate cells from each other.

We apply the same methodology for design analysis to both pre- and post-placed analysis parts, and present the results in table form. The hybrid HL-tree clocking scheme [8] is also analyzed for which a system clock is presented. A conclusion is presented and future improvements are identified.

II. PRE-PLACED STA

For the pre-placed STA, we use a JSIM [9] or SPICE compatible electrical netlist of a circuit which appears as an intermediate step in the synthesis process. This netlist contains all the cells, either as direct implementations, or as subcircuits in the SPICE format. The netlist is analyzed, and inputs and outputs (IO) are identified by either supplying these by command, or by letting the tool identify each algorithmically. Once all IOs are identified, a breadth-first search (BFS) is run to connect all the input nodes to all possible outputs. This process identifies all the possible paths through the design which is then used to calculate delay. The complexity in the BFS increases exponentially for every pulse splitter found on the path as this causes a new branch which eventually becomes a path on its own. For small circuits this is trivial but as the design increases in size so does the evaluation time needed to identify all the paths. This however remains vastly superior to physical (electrical) simulation of equal size designs.

Manuscript received November 16, 2017; revised January 19, 2018; accepted March 5, 2018. Date of publication March 14, 2018; date of current version April 4, 2018. This work was supported by IARPA contract FA8750-15-C-0203-IARPA-BAA-14-03. (*Corresponding author: Johannes A. Delpoort.*)

The authors are with the Stellenbosch University, Stellenbosch 7599, South Africa (e-mail: joeydelp@gmail.com; coenrad@sun.ac.za).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TASC.2018.2815919

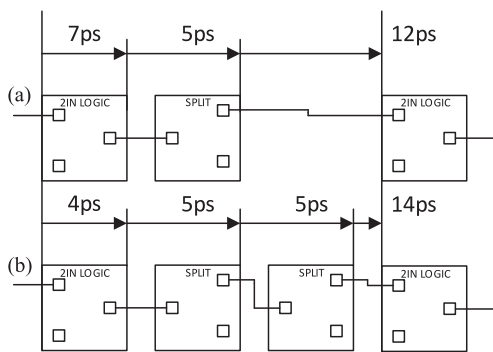


Fig. 1. Splitter consideration in system clock calculation.

BFS was chosen as the graph traversal algorithm due to its relative simplicity for implementation. No additional data structures are required as it only needs to keep track of when the path splits. Essentially start and ending points are chosen and the BFS algorithm follows the path from start to end, making a note of every time a splitter is reached. Once a path has been found start to end, the algorithm identifies where the last splitter was found (if any) and follows that path to the end. Once all the splitters have been exhausted along the path all possible paths from start to finish have been found for the specified points. Depth first search traverses a path and each time a splitter is reached it branches and then continues by visiting one node in one branch and then one in the other. Results are similar to BFS, but harder to implement.

Through the aid of the accurately characterized timing information in the standard cell library, each logic cell in every path can be identified and their delays added together. The greatest delay through the design from any input to any output we then identify as the critical delay, and the path along which it is calculated is identified as the critical path. By using this critical delay, we can then identify the preliminary global clock speed for the design. This is the clock speed at which a result would be produced at the output in each clock cycle.

Logic gates in RSFQ and ERSFQ logic need to be clocked, which effectively provides pipelining in every path. If a circuit is designed for wave-pipelining [10], which has been demonstrated for RSFQ [11], and paths are balanced with clocked delay cells, then we can designate the maximum possible system clock as that which is needed to successfully shuttle bits between successive logic gates. To identify the system clock we need to analyze once more all the paths, but this time identify the largest gate-to-gate delay considering all the splitters between gates. Here, clocked delay elements count as gates.

RSFQ/ERSFQ cells have a fan-out of one, so that signal connection to multiple inputs require a succession of one-to-two pulse splitters. Fig. 1 illustrates how the system clock can be influenced by splitter cells that do not need to be clocked. In Fig. 1(a) a case is depicted where the output signal from a logic gate only has to traverse one splitter before the next logic gate. Fig. 1(b) depicts a case where the logic cell has a much shorter delay, but where the output signal traverses two splitters before the next logic gate. If these branches are part of the same system, the system clock has to be $1/14\text{ps}$ to account for the largest gate-to-gate delay.

III. POST PLACED-AND-ROUTED STA

For post placed-and-routed STA we analyze a file in the Cadence DEF format [12]. The file is generated by the RSFQ Mapper application [8], developed as part of the same IARPA project that funded this research. RSFQ Mapper receives a high-level design (in Verilog HDL or BLIF format) and maps it to a RSFQ chip. The mapping process is composed of logic synthesis, placement (global and detailed), clock tree synthesis, and routing steps. The final output of the RSFQ Mapper is a placed-and-routed netlist in Design Exchange Format (DEF). To analyze the file, we first identify all the components and nets (wires) in the design. Thereafter, we connect all the components through their respective nets. Utilizing the same BFS as in the pre-placed case, we identify all the paths in the design. Identifying the global clock follows a similar process as before, but now includes the wire delays as well as via delays that are not present in the netlist representation. The longest of these delays is again the critical time, which can be used to calculate the global clock speed. The RSFQ Mapper toolchain can implement a hybrid clocking scheme, which becomes inherent in the post placed-and-routed description of a circuit and consequently also in the calculation of critical paths and delays. This is discussed in Section IV. Our method estimates a system clock from the critical path on the assumption that a concurrent or H-tree clocking scheme is used, and by following a similar process as in the pre-placed method. It is not valid for other clocking schemes.

Our method produces further information regarding the timing of the design in the form of the slack, total slack, mean path time, path variance as well as the standard deviation. In the case of the slack, the user would be required to specify a target time for the design which would then be compared to the critical path time. If the target time exceeds the critical time, positive slack is reported, and the design can be considered as good. If the target time is less than the critical time, negative slack is reported, which means that the design is slower than intended. In the case of negative slack, the value for total slack is also produced, which is a measure of the total paths that exceed the target time. This value is an indication of how good the design is or whether a complete redesign is necessary.

IV. CLOCK SCHEME CONSIDERATIONS

Clock scheme selection for RSFQ-based circuits is not trivial, and several topologies are possible (for a review, see [6]). In SFQ circuits, logic cells are synchronous and therefore require a clock to propagate pulses from the input through the cells. For the design of the RSFQ Mapper application, research was done in what the best way would be to clock large designs in RSFQ [13]. In the research presented here we only account for concurrent clocking, H-tree clocking and the hybrid HL-tree clocking found in the final version of the RSFQ Mapper application.

The H-tree clock can be determined if it is assumed that every gate would be a leaf to a balanced H-tree and would thus receive its clock at the same time. This makes the calculation of the clock purely the depth of the H-tree as a measure of the total delay accrued through all the clock splitters on its path. However, the method fails if any branch of the H-tree network

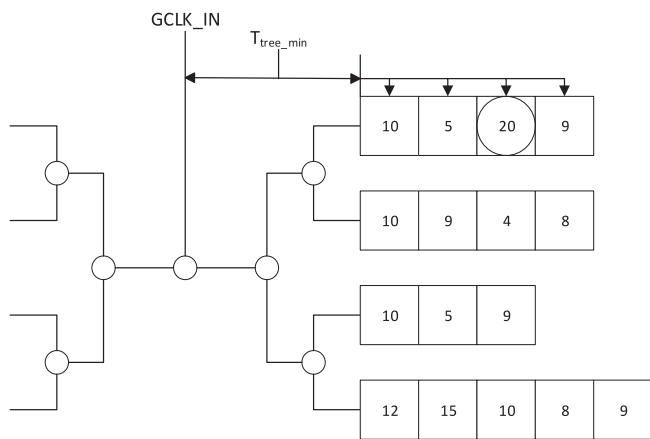


Fig. 2. HL-tree clocking scheme.

is unbalanced, so that a more general solution is required. We assume that minor imperfections and randomness in component values would not affect the timing in such a way that unbalance in the clock would appear.

The HL-tree clocking scheme is depicted in Fig. 2. It is evident that a pure H-tree network is simply an HL-tree network with exactly one leaf to each branch. The aim with the HL-tree network is to have a balanced H-tree up to a point, after which gates (that may be dependent on the output of its nearest neighbor, but not necessarily so) are stacked in a row and clocked concurrently. To calculate the HL-tree system clock we reconstruct the clock paths found in the DEF file and identify the minimum clock depth between an arbitrary clock input (GCLK_IN in Fig. 2) and the first logic gate. This is denoted as T_{tree_min} shown in Fig. 2. The system clock is then identified as the longest delay through concurrent splitters and a logic gate minus T_{tree_min} .

This method inherently supports an unbalanced clock tree.

V. RESULTS AND CONSIDERATIONS

We developed an STA tool to implement the methods discussed above. This tool, SuperSTA, is demonstrated on a 16-bit Kogge-Stone adder (KSA) that was synthesized with RSFQ Mapper. STA is analysed for both pre-placed and for post-placed-and-routed versions.

The STA output for the 16-bit Kogge-Stone adder netlist, before placing and routing, is shown in Fig. 3.

As seen in Fig. 3, the results from a netlist produces no clock speed as this would not be an accurate result due to the lack of wiring delays. This intermediate step application is useful to gauge whether the circuit would even come close to the target time before doing a complete place and route—it is also important to note that the critical time is calculated from the clock-to-out delay of every logic gate. It can therefore be seen as an early assessment of the design. The design is then placed and routed, after which it is analyzed with the STA tool again, now with full information on actual signal wire lengths. The results of this analysis are shown in Fig. 4.

It is clear from the critical path time how large an influence the wiring and via delays have on the maximum clock frequency

```
File specified to analyse: "KS16.netlist"

There are 770 lines
of which 463 are relevant

No input nodes specified. Taking a guess. Specify nodes
for better accuracy.
There are 33 input nodes

Critical path: 1 164 446 170 171 172 431 432 185 186 402
70 334 71 332 162 34

Total delay through the circuit: 1.182e-10s or rather
118.2ps

Slack: 1.818e-10s

Mean path time: 7.38934e-11s or rather 73.8934ps

Path time variance: 4.34687e-22s^2 or rather 434.6870ps^2

Path time standard deviation: 2.08492e-11s or rather
20.8492ps
```

Fig. 3. STA extraction results for a 16-bit Kogge-Stone adder read from a netlist.

```
File specified to analyse: "16bitsa_route.def"

Critical path time: 5.54009e-10s or rather 554.009ps

Global clock: 1.80502GHz

Critical path:
a0_Pad
Split_646_n1894
xor2a_51_n51
Split_500_n1748
Split_501_n1749
and2_55_n55
or2_56_n56
Split_506_n1754
Split_507_n1755
and2_74_n74
or2_75_n75
Split_522_n1770
Split_523_n1771
and2_132_n132
or2_133_n133
Split_561_n1809
and2_267_n267
or2_268_n268
xor2a_269_sum15
sum15_Pad

Slack: -2.54009e-10s

Total slack: -3.17481e-07s

Mean path time: 4.58558e-10s or rather 458.558ps

Path time variance: 1.49185e-21s or rather 1.49185e-09ps

Path time standard deviation: 3.86245e-11s or rather
38.6245ps

Longest NET: xor2a_104_n104 -> DFF_438_n1686 => 2790
micron

HL-Tree critical time: 100.7ps

HL-tree clock: 9.93049GHz
```

Fig. 4. STA extraction results for a placed-and-routed 16-bit Kogge-Stone adder.

of the circuit. The circuit design, which was not optimized for switching speed, reduced from an estimated 8.46 GHz from the pre-placed and unrouted netlist down to a mere 1.8 GHz system (global) clock. This shows the potential of the STA analysis presented here to yield information on clock speed reduction

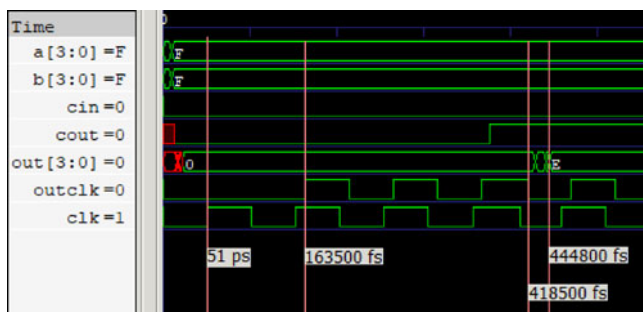


Fig. 5. Verilog simulation results of a 4-bit KSA at 19.6 GHz, with correct output.

after automated place and route procedures, and to assist with clock speed optimization.

Furthermore, due to the design utilizing the HL-tree clocking scheme, the results include HL-tree information. The measure of slack indicates that the design is slower than initially intended. From slightly positive in the netlist case, it reduces to negative in the placed-and-routed case, with a total slack for the latter that indicates that majority of the paths are slower than the target time. Such a result indicates that reconsideration or redesign is needed.

As another demonstration, consider a 4-bit Kogge-Stone adder subjected to the same process. Here we compare the results of the STA to that of an HDL simulation. The HDL simulation includes all the wire delays, as it is a gate level description of the post-placed and routed 4-bit KSA. SuperSTA identifies the resulting HL-tree clock as 19.72 GHz or rather a clock pulse every 50.7 ps. We then adjust the clock speed for the HDL design accordingly and compare results.

From Fig. 5 we can see that at 19.60 GHz the design still produces the correct results. It is important to note that we use event-based logic for HDL simulation, which implies that at every level change in the simulation indicates the occurrence of a pulse (event). It is also important to note that the critical path depth of the 4-bit KSA is 6 clocked gates, which means that the result to a sum of two parallel-input 4-bit words will only be shown after 6 clock pulses. The design was then over-clocked to attempt to find the maximum clock before the output produced critical timing violations. This maximum overclock was achieved at 19.72 GHz for the input words shown in Fig. 5, which is exactly the limit suggested by SuperSTA. It is important to note that every possible input vector was probed, and though some did not produce critical timing violations at the suggested limit, the purpose of the STA tool is to identify the maximum clock at which violations will occur for any input. A SPICE simulation for the 4-bit KSA is shown in Fig. 6 with the maximum clock also set at 19.6 GHz. The correct output is still produced at this frequency, but pulse repulsion occurs above this frequency. This produces incorrect output and thus further verifies the critical clock timing presented by SuperSTA.

Table I shows the solution times for problems of increasing size. It can clearly be seen that as the possible paths increase so does the time to solution. The first two entries are pre-placed netlist designs and are easier to analyze. Solutions

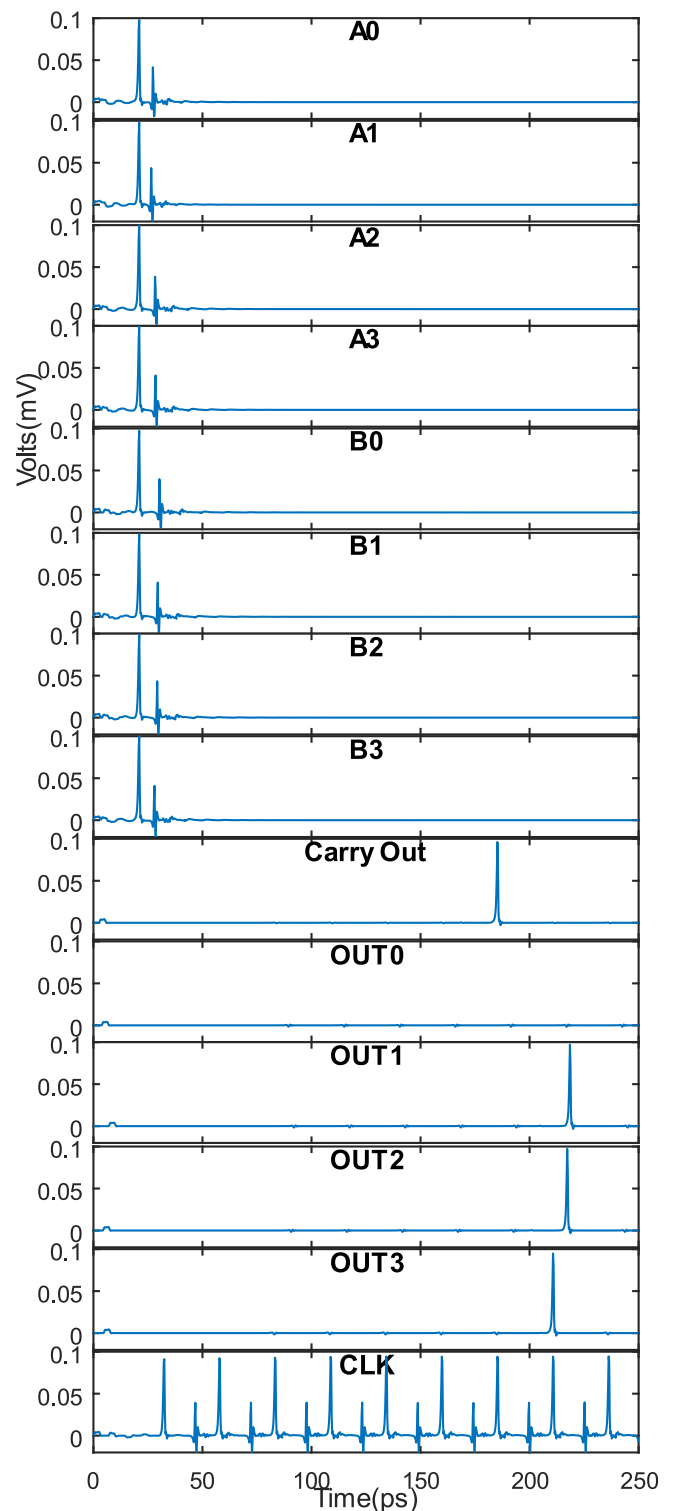


Fig. 6. SPICE simulation of the 4-bit KSA at 19.6 GHz, with correct output.

for post-placed designs are harder to analyze as they require reconstruction to netlist form and also include additional calculation time for transmission lines and vias. The time it takes to calculate the large problems is still very reasonable, so that a simulation could easily be rerun if changes are needed in the design. Furthermore, the execution of a SPICE version of the

TABLE 1
TIME-TO-SOLUTION AND DESIGN SIZE COMPARISON

Design	Possible Paths	Solution time (s)	
		SuperSTA	HDL
KS4 netlist	47	0.0367246	-
KS16 netlist	610	0.0659592	-
KSA4 routed	61	0.0553873	0.1819739
KSA8 routed	197	0.1258301	0.2733722
KSA16 routed	693	0.6152803	1.6130728
KSA32 routed	2589	4.2927526	6.8411896

4bit KSA runs a total of 45.22s in JSIM on a modern quad-core computer, so that it becomes impractical for larger circuits to run multiple JSIM simulations for detecting timing violations.

VI. CONCLUSION

A static timing analysis tool was proposed for the superconducting circuit design domain, which could speed up the design and redesign of large scale designs without the need for physical simulation. Though not generally applicable to all design methodologies in superconducting circuit design, the results produced in the test cases satisfy the general concept of STA and would suffice as a proof of concept tool.

We also demonstrated through HDL simulations that the STA tool, in finding the worst-case safe timing, does not overestimate the maximum clock speed.

Although not specifically discussed in the text, it is possible to include jitter caused by thermal noise or timing changes due to parameter variations as worst-case values added to the delay times of every gate. The STA tool would then find the associated worst-case delay value.

Importantly, it is now possible to verify quickly if a synthesized, placed-and-routed circuit fares better or worse than another instantiation in terms of maximum clock speed, which is an important requirement for clock speed optimization.

Future considerations are the inclusion of multiple clocking schemes as well as a more generally used place-and-route format.

ACKNOWLEDGMENT

The authors would like to thank M. Pedram, N. Katam, S. Shahsavani, and T.-R. Lin (University of Southern

California) for helpful discussions and for synthesized versions of the Kogge-Stone adders used here, as well as Lieze Schindler (Stellenbosch University) for her contribution of the HDL simulation data for the Kogge-Stone adder.

REFERENCES

- [1] S. Yorozu, Y. Kameda, H. Terai, A. Fujimaki, T. Yamada, and S. Tahara, "A single flux quantum standard logic cell library," *Physica C, Supercond.*, vols. 378–381, pp. 1471–1474, 2002.
- [2] H. Akaike *et al.*, "Design of single flux quantum cells for a 10-Nb-layer process," *Physica C, Supercond.*, vol. 469, pp. 1670–1673, 2009.
- [3] K. K. Likharev, O. A. Mukhanov, and V. K. Semenov, "Ultimate performance of RSFQ logic circuits," *IEEE Trans. Magn.*, vol. MAG-23, no. 2, pp. 759–762, Mar. 1987.
- [4] 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, no. 2, pp. 3–28, Mar. 1991.
- [5] D. E. Kirichenko, S. Sarwana, and A. F. Kirichenko, "Zero static power dissipation biasing of RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 776–779, Jun. 2011.
- [6] K. Gaj, E. Friedman, and M. J. Feldman, "Timing of multi-gigahertz rapid single flux quantum digital circuits," *J. VLSI Signal Process.*, vol. 9, pp. 247–276, 1997.
- [7] S. Intiso, I. Kataeva, E. Tolkacheva, H. Engseth, K. Platov, and A. Kidiyarova-Schevchenko, "Time-delay optimization of RSFQ cells," *IEEE Trans. Appl. Supercond.*, vol. 15, no. 2, pp. 328–331, Jun. 2005.
- [8] S. N. Shahsavani, T.-R. Lin, A. Shafaei, C. J. Fourie, and M. Pedram, "An integrated row-based cell placement an interconnect synthesis tool for large SFQ logic circuits," *IEEE Trans. Appl. Supercond.*, vol. 27, no. 4, Jun. 2017, Art. no. 1302008.
- [9] E. S. Fang and T. Van Duzer, "A Josephson integrated circuit simulator (JSIM) for superconductive electronic applications," in *Proc. Ext. Abs. Int. Supercond. Electron. Conf.*, Tokyo, 1989, pp. 407–410.
- [10] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-pipelining: A tutorial and research survey," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, no. 3, pp. 464–474, Sep. 1998.
- [11] T. Filippov, M. Dorjjevets, A. Sahu, A. Kirichenko, C. Ayala, and O. Mukhanov, "8-Bit asynchronous wave-pipelined RSFQ arithmetic logic unit," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 847–851, Jun. 2011.
- [12] Cadence, "LEF/DEF language reference - international symposium on physical design," 7 Nov 2009. [Online]. Available: www.ispd.cc/contests/14/web/doc/lefdefref.pdf, Accessed on: 25 May, 2017.
- [13] N. Katam, A. Shafaei, and M. Pedram, "Design of multiple fanout clock distribution network for rapid single flux quantum technology," in *Proc. 2017 22nd Asia South Pacific Des. Autom. Conf.*, 2017, pp. 384–389.

Authors' biographies not available at the time of publication.

Appendix D

Journal Paper - JoSIM – Superconductor SPICE Simulator

- J. A. Delpont, K. Jackman, P. le Roux, and C. J. Fourie, "JoSIM - Superconductor SPICE Simulator," IEEE Transactions on Applied Superconductivity, 2019. [25]

In this paper we introduce JoSIM simulator for superconducting circuits. The development decisions are discussed and comparisons are made to existing simulators. All contributions to this paper are my own.

JoSIM – Superconductor SPICE Simulator

Johannes A. Delpoort, *Student Member, IEEE*, Kyle Jackman, Paul le Roux *Student Member, IEEE*,
and Coenrad J. Fourie, *Senior Member, IEEE*

Abstract—We present JoSIM, a SPICE based circuit simulator that utilizes the modified nodal voltage analysis method and trapezoidal integration to solve systems of linear equations. The objective of JoSIM is to provide accurate simulation results with major improvements in terms of simulation speed and expandability. JoSIM is written in such a way that optimization engines can be directly written into the source as functions. JoSIM incorporates the ability to do phase-based simulation through a modified nodal phase analysis method. A full data visualization GUI, built using open-source graphical libraries, is included. We show the results of simulations with JoSIM and compare them to the results of JSIM, as well as comparisons between simulation times. We also show extremely large simulations which are not realizable in reasonable time using JSIM. The software tool is packaged and presented as part of a US government funded project.

Index Terms—Circuit analysis, Circuit simulation, Josephson junctions, SPICE, Superconducting integrated circuits

I. INTRODUCTION

SPICE simulation in superconductivity is a rather niche field due to the complexity of the Josephson junction (JJ) element and the non-linearities created by it. Most simulators rely on approximations such as the resistively and capacitive shunted junction (RCSJ) to model the tunnel current effect of the JJ [1]. Regardless of being approximations, the modelled effect is suitable for simulation purpose and near enough to practical results to be acceptable in most cases. The closest approximation to the Josephson junction that models the Josephson effect most accurately was done by Werthamer in 1966 [2]. This approximation though has not seen exact implementation in a general simulation engine, with the closest being the microscopic tunnel junction (MTJ) in the personal superconductor circuit analyser (PSCAN) [3].

The first documented case of an attempt at the Josephson effect in SPICE was by Jewett at University of California Berkeley in 1982 [4]. The JJ model was added to the existing SPICE 2G5 and allowed the user to choose one of 3 types of quasiparticle resistances (Rtype). This method was however rather slow due to the numerical method used by SPICE for accurate simulation of transistor type devices. The SPICE 2G5 with the implementation of the JJ was named JSPICE.

WRspice is a SPICE engine developed by Whiteley Research Incorporated in Sunnyvale, CA. Until October of 2017

The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120.

J. Delpoort (jdelport@sun.ac.za), K. Jackman (kjackman@sun.ac.za), P. le Roux (17500966@sun.ac.za) and C. Fourie (coenrad@sun.ac.za) are all with the Department of Electrical and Electronic Engineering, Stellenbosch University, Stellenbosch, WC, 7600 South Africa.

Manuscript received October 30, 2018.

it was a commercial SPICE engine and part of a toolset called XicTools, which included the layout package Xic. Development started as a project to rewrite JSPICE3 in C++ while maintaining full compatibility for older SPICE simulators.

Josephson simulator (JSIM) [5] developed by Fang and Van Duzer in 1989 is a SPICE simulator dedicated to simulation of JJs and does not support any semiconductor devices. The simulator is very light weight due to the need to only support a few components. JSIM makes use of nodal analysis to compute solutions of large matrices.

In this paper we present JoSIM, a Josephson junction SPICE simulation engine for transient analysis akin to JSIM and WRspice written in modern C++ with emphasis on extendibility while remaining focused on superconductive circuit elements.

We compare JoSIM to JSIM and WRspice in terms of accuracy of the JJ model and execution speed of various size simulations. We discuss the design philosophy involved and introduce the ability to perform phase-based simulations and discuss the potential of using phase to improve simulation speed and memory usage for large simulations.

II. MODIFIED NODAL ANALYSIS

A. Voltage

When attempting to solve a set of linear equations to find the voltage at every node as in the modified nodal analysis (MNA) [6], we need to first decide on an integration method that would approximate the current or voltage of non-linear components. The most basic of these methods is simply the backward Euler methods, which interpolates the value based on the next or previous value. This method is however a first order method which does not model the behaviour accurately enough and is prone to cumulative error. We therefore opt to use a second order method such as the trapezoidal integration method. Trapezoidal integration method can be defined as

$$\frac{dy}{dt}_n = \frac{2}{h_n} (y_n - y_{n-1}) - \frac{dy}{dt}_{n-1} \quad (1)$$

where n is the iteration count and h_n the current time step in the transient analysis. This integration method is suitably accurate in most cases but does however still produce spikes when a rapid change in y occurs.

JoSIM programmatically creates component matrices using generic MNA stamps for the component type. We demonstrate the creation of these generic MNA stamps through an inductor as example.

$$v = L \frac{di}{dt} \quad (2)$$

If we apply the equation in (1) to (2) we obtain an equation for the inductor voltage that is dependent on the previous time step current and voltage.

$$V_n - \frac{2L}{h_n} I_n = -\frac{2L}{h_n} I_{n-1} - V_{n-1} \quad (3)$$

Where (3) can be written in general matrix form as

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{2L}{h_n} \end{bmatrix} \begin{bmatrix} V^+ \\ V^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L}{h_n} I_{n-1} - V_{n-1} \end{bmatrix}$$

This general matrix form is used to solve the $Ax = b$ linear algebra problem in circuit simulation.

The MNA stamp used for the JJ coincides with the matrix found in JSIM, which relies on a second order guess of the phase for the next time step.

$$\begin{bmatrix} \frac{1}{R} + \frac{2C}{h_n} & -\frac{1}{R} - \frac{2C}{h_n} & 0 \\ -\frac{1}{R} - \frac{2C}{h_n} & \frac{1}{R} + \frac{2C}{h_n} & 0 \\ -\frac{h_n}{2} \frac{2e}{\hbar} & \frac{h_n}{2} \frac{2e}{\hbar} & 1 \end{bmatrix} \begin{bmatrix} V^+ \\ V^- \\ \phi \end{bmatrix} = \begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} V_{n-1} \end{bmatrix}$$

where the phase node ϕ is a virtual node not connected physically in the circuit and e and \hbar are the electron charge and Plancks constant respectively.

$$I_s = -I_c \sin \phi^0 + \frac{2C}{h_n} V_{n-1} + C \dot{V}_{n-1} \quad (4)$$

with

$$\phi_n^0 = \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} (V_{n-1} + v_n^0) \quad (5)$$

and

$$v_n^0 = V_{n-1} + h_n \dot{V}_{n-1} \quad (6)$$

B. Phase

PSCAN first introduced in 1991 utilized a similar method of computing the phase as standard variables, however this tool was practically unobtainable until a recent re-release by Shevchenko, written in Python, made it open-source. The tool utilizes its own unique input language, SFQHDL, which allows the self-verification of circuit results using the built-in optimization engine.

The JJ being is a phase-based element, therefore the direct calculation of the phase is more practical. What phase mode analysis presents is a direct relation between voltage and phase which can be substituted into any voltage dependent equation. The act of derivation implies extraction of information from a source and thus retaining data that is less informative than the original. This is evident through constant derivation of voltage to obtain phase an accumulation of error is observed.

It therefore becomes sensible to perform the entire transient analysis in phase since each component affects the phase of the entire circuit. We have already established a MNA system that handles the calculation of voltage and current in a circuit thus it would be sensible to simply adapt this method to calculate phase.

To start this process we substitute Josephson voltage-phase relation [7] in (7) into every component equation and reduce it to find a modified nodal phase analysis (MNPA) matrix.

$$v = \frac{\Phi_0}{2\pi} \frac{d\phi}{dt} \quad (7)$$

We once again demonstrate this using the inductor as example.

$$\frac{\Phi_0}{2\pi} \frac{d\phi}{dt} = L \frac{di}{dt} \quad (8)$$

$$\phi_n - \frac{2\pi L}{\Phi_0} I_n = 0 \quad (9)$$

which leads to the MNPA stamp

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -\frac{2\pi L}{\Phi_0} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Utilizing phase no longer depends on the previous time step values for inductors, which reduces overall complexity since superconducting circuits are largely inductive.

The JJ MNPA stamp remains mostly the same with simply the role of the voltage and phase swapped. This means that the phase is now connected, and the voltage becomes a virtual non-connected node.

$$\begin{bmatrix} 0 & 0 & \frac{1}{R} + \frac{2C}{h_n} \\ 0 & 0 & -\frac{1}{R} - \frac{2C}{h_n} \\ 1 & -1 & -\frac{h_n}{2} \frac{2e}{\hbar} \end{bmatrix} \begin{bmatrix} \phi^+ \\ \phi^- \\ V \end{bmatrix} = \begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} V_{n-1} \end{bmatrix}$$

The phase for the next time step remains the same second order guess which utilizes a guess voltage.

Direct calculation of the phase allows the addition of DC external magnetic field through mutual coupling with all the inductors in the circuit. This is a rather important feature in low temperature superconductivity due to the high susceptibility to external fields which was not possible using voltage-based methods [8].

III. FEATURES AND TEST METHODOLOGY

A. Features

JoSIM is written to accommodate standard SPICE syntax as well as the syntax utilized by JSIM. This allows the results of simulations to be quite easily compared with that of JSIM and WRspice. Apart from the capability to perform phase-based simulations, JoSIM allows alpha-numeric node numbers in the SPICE netlist.

The inclusion of an expression parsing algorithm based on Dijkstras shunting yard [9], which allows the creation of variables within the SPICE netlist which can be used to scale, compute or parameterize component values within the netlist.

JoSIM allows the output of result vectors in various ways including space- or comma-separated files. A key feature that is provided with JoSIM is the ability to plot the result vectors through either the cross-platform FLTK graphical library or the Python based matplotlib interface. The latter which provides the ability to scale, label and save the results as publication grade figures.

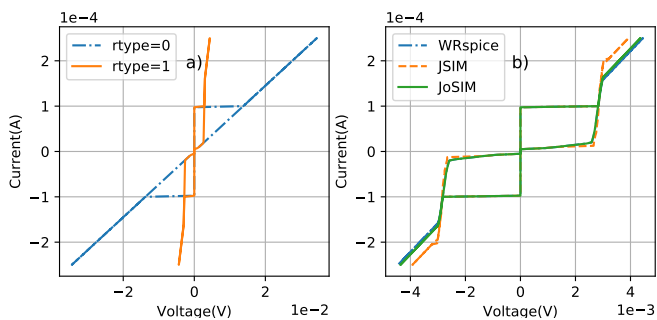


Fig. 1. a) JoSIM Rtype=0 and Rtype=1 I-V curves. b) JoSIM, JSIM and WRspice Rtype=1 I-V curves.

B. Test Methodology

We compare the quasiparticle resistance model used in JoSIM to the others. This is done by comparing the I-V curves which are generated by steadily ramping up the input current of a JJ and keeping it constant and averaging the voltage across the junction once stabilized. The current is incremented and the process is repeated until the input current reaches some large value whereafter the current is swept through zero to the negative peak and back to zero. Plotting these averaged voltage values against the current produces the I-V curve of the junction.

The only quasiparticle resistance models (Rtype) that are implemented in JoSIM at present is the unshunted (Rtype=0) and PWL resistance model (Rtype=1). This is similar to that found in JSIM, where WRspice has an exponentially derived resistance curve (Rtype=2), a fifth order polynomial expansion (Rtype=3) and a temperature variation model (Rtype=4) controlled by a specified current source. The models implemented by WRspice are scheduled for incorporation into JoSIM in the near future.

Additional tests are performed which benchmark the speed and simulation size capabilities of JoSIM compared to others. These benchmarks include a basic Josephson transmission line (JTL), a 4-bit Kogge-Stone Adder (4-bit KSA) and various large simulations created by stringing together JTLs up to a total of 10,000 JJs.

IV. RESULTS

A. I-V Curves

The first test involving the I-V curves of the JJ in JoSIM is presented in Figure 1 a. The comparison between the I-V curve of JoSIM, JSIM and WRspice can be seen in Figure 1 b. Since the JJ model used in JoSIM matches the one found in WRspice they are very close with the difference forming as a result of the way JoSIM approximates the voltage guess for the next time step.

The model used in JSIM differs to JoSIM and WRspice in the way the transitional current and conductance is calculated. JSIM approximates a transition conductance as the slope between the sub-gap and normal resistances for the gap voltage spread region (ΔV). Based on where the voltage is guessed to be in the next time step, the A matrix is adjusted using either sub-gap, transition or normal conductance. During the

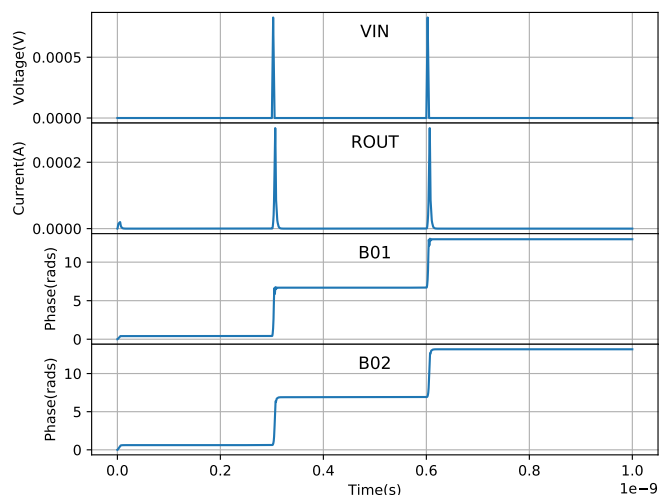


Fig. 2. Josephson transmission line (JTL) simulation using JoSIM.

transitional state, two regions are defined namely $V_{gap} + \Delta V$ and $V_{gap} + 2\Delta V$. When in the first region a constant current is added to the b matrix entry for the junction, where a varying current is added during the second region.

This model differs quite significantly from the one used in JoSIM and WRspice, where a single region ΔV is defined surrounding the gap voltage. The transition conductance is calculated using the critical current, ΔV and a ratio of critical to gap current (typically $\pi/4$). Similar to JSIM the A matrix is adjusted with the transitional conductance when entering the transition region, however the current with which the b matrix entry is adjusted differs and is continually added even when entering the normal resistive state.

This difference in RCSJ model implementation is what causes the normal region of WRspice and JoSIM to differ slightly when compared to JSIM. The WRspice/JoSIM JJ model provides a better match to measured I-V curves for processes such as MITLL.

B. JTL

The example tested using JoSIM is the basic JTL and is seen in Figure 2. We further compare these results to the same simulation performed using JSIM as well as WRspice and plot the percentage difference between the junction output phases. The results of this comparison can be observed in Figure 3. The difference percentage scale is expressed in logarithmic form for greater clarity.

C. Execution Speed

The execution speed and simulation size capabilities of JoSIM were tested and compared to that of JSIM and WRspice. The results of these simulations are shown in Table I.

D. Phase Simulations

When performing phase-based simulations the phase of each node in the circuit is calculated. This method was implemented

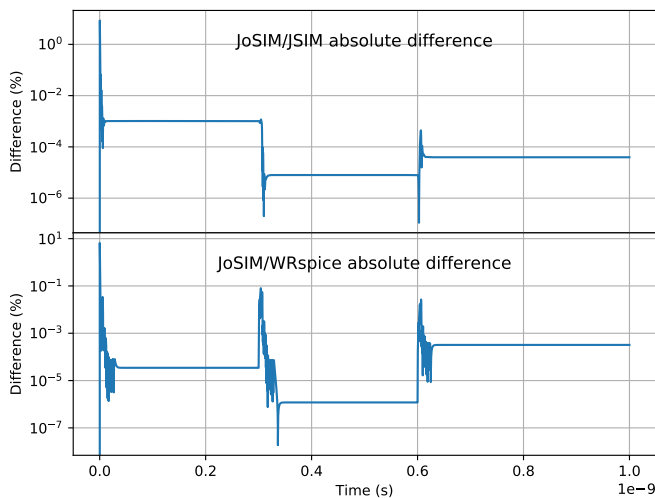


Fig. 3. JTL phase difference of JoSIM vs JSIM and WRspice.

TABLE I
COMPARISON OF SIMULATOR EXECUTION SPEEDS

Simulation	Execution Times(s)			
	JJ Count	JSIM	WRspice	JoSIM
Basic JTL	2	0.06	0.219	0.139
400 simulation I-V curve	400	60.5	56	54.87
4-bit KSA	2,095	73.9	DNF	23.49
General Partial Products	3,904	93	64	20.9
3,000 JTL string	6,006	276	159	91.88
4,000 JTL string	8,006	>3,600	232.7	130.87
5,000 JTL string	10,006	DNF	263.8	169.81

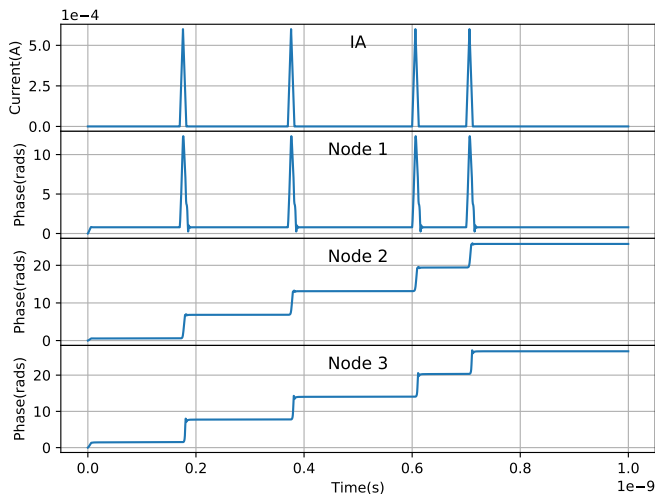


Fig. 4. Nodal phase analysis of a DC to SFQ connected to JTL with resistor termination.

in JoSIM in such a way that no alterations need be made to a circuit netlist to allow for phase-based analysis. We plot the results of the nodal phases of a DC to SFQ converter connected to a JTL and terminated using a resistor in Figure 4.

V. CONCLUSION

A superconducting circuit simulator JoSIM was demonstrated to have superior execution speed while maintaining

accuracy. Voltage- and phase-based analysis methods were discussed in detail with demonstrations of each. The simulator was compared to popular existing alternatives as well as percentage difference measured.

JoSIM has been proven to be a reliable new superconducting circuit simulator with clear advantages in simulation speed as well as compatibility with multiple formats. Advanced features such as the capability to do both phase and voltage analysis, parameterize circuit netlists through expressions as well as natively plotting outputs and saving them in a variety of formats.

Development on this simulator will continue with efforts to introduce parallel processing in certain regions to further improve performance on large circuits being investigated. Additional RCSJ models will be considered for future releases along with the elusive microscopic tunnel junction [10].

Additional improvements planned for future implementation include the addition of noise to simulations as well as the modulation of temperature locally on a per component base, as well as globally across the circuit. Development can be tracked through the JoSIM open-source online repository [11].

REFERENCES

- [1] B. D. Josephson, "Possible new effects in superconducting tunneling," *Phys. Lett.*, 1962.
- [2] N. R. Werthamer, "Nonlinear self-coupling of Josephson radiation in superconducting tunnel junctions," *Phys. Rev.*, 1966.
- [3] S. V. Polonsky, V. K. Semenov, and P. N. Shevchenko, "PSCAN: Personal superconductor circuit analyser," *Supercond. Sci. Technol.*, 1991.
- [4] R. Jewett, "Josephson junctions in SPICE 2G5," University of California Berkeley, 1982.
- [5] E. S. Fang and T. Van Duzer, "A Josephson integrated circuit simulator (JSIM) for superconductive electronics application," *Ext. Abstr. 2nd Int. Supercond. Electron. Conf.*, 1989.
- [6] U. V. Wali, R. N. Pal, and B. Chatterjee, "On the Modified Nodal Approach to Network Analysis," *Proc. IEEE*, 1985.
- [7] K. A. Delin and T. P. Orlando, "Foundations of Applied Superconductivity," Addison-Wesley Publishing Company, 1991, p. 406.
- [8] K. Jackman and C. J. Fourie, "Software tools for flux trapping and magnetic field analysis in superconducting circuits," *IEEE Trans. Appl. Supercond.*, Submitted for publication.
- [9] E. W. Dijkstra, "Algol-60 Translation," 1961
- [10] H. Kratz and W. Jutzi, "Microscopic simulation model of Josephson junctions for standard circuit analysis programs," *IEEE Trans. Magn.*, vol. 23, no. 2, pp. 731, Mar. 1987.
- [11] J. A. Delpoit, (2018, Oct 18) "JoSIM: Superconducting Circuit Simulator," [Online]. Available: <https://github.com/JoeyDelp/JoSIM>.

Appendix E

JoSIM - User Manual

JoSIM - Superconducting Circuit Simulator

Developers Manual
v2.1

Johannes A. Delpont

Stellenbosch University
South Africa
November 20, 2018

Copyright © 2017-2018 by Johannes Delpont

Permission is granted to anyone to make or distribute verbatim copies of this document as received, in any medium, provided that the copyright notice and the permission notice are preserved, and that the distributor grants the recipient permission for further redistribution as permitted by this notice

Linux is a registered trademark of Linus Torvalds.

Windows is a registered trademark of Microsoft Corporation.

macOS is a registered trademark of Apple, Inc.

All trademarks are the property of their respective owners.

Contents

1	Introduction and setup	1
1.1	Introduction	1
1.2	Initial setup	1
1.3	License	2
1.4	Building from source	2
1.4.1	UNIX	2
1.4.2	Windows	4
2	Technical discussion	5
2.1	Modified nodal analysis	5
2.2	Trapezoidal integration	5
2.3	LU decomposition	7
2.4	Data structures & speed considerations	7
2.5	FLTK	8
2.6	Matplotlib	8
2.7	Modified nodal phase analysis	8
3	Input files	10
3.1	JSIM Notation	10
3.2	WRSpice Notation	10
3.3	Subcircuit Convention	10
4	Command line arguments/switches	11
5	Output	12
5.1	Comma separated value	12
5.2	Space separated value	12
5.3	GUI plotting window	12
6	Examples	13
7	Error handling and exceptions	16
8	Planned improvements	16
	Appendices	18
A	Component Stamps	18
A.1	Resistor	18
A.2	Capacitor	18
A.3	Voltage source	19
A.4	Current source	19

A.5	Josephson junction	20
A.6	Transmission line	21
A.7	Mutual Inductance	22

1 Introduction and setup

1.1 Introduction

JoSIM was developed under IARPA contract SuperTools(via the U.S. Army Research Office grant W911NF-17-1-0120). JoSIM is a analogue circuit simulator with SPICE syntax input that has inherent support for the superconducting Josephson junction element.

JoSIM is meant to function as a replacement to the aging simulator JSIM[1]. JoSIM is written in modern C++ and is fully customizable and extendable to offer support for improved superconducting elements as well better approximations to the Josephson effect in superconducting materials.

A *.cir* or *.js* file containing a SPICE syntax circuit netlist is provided as input. The circuit netlist, given appropriate input excitations can then be simulated through transient analysis. Results of this simulation can be either plotted for quick reference or saved as either a space delimited (*.dat*) or a comma separated value (*.csv*) file.

Fig.1 shows an overview of what JoSIM aims to accomplish. This is much like any other SPICE deck simulator with the exception that it incorporates native handling of the Josephson junction.

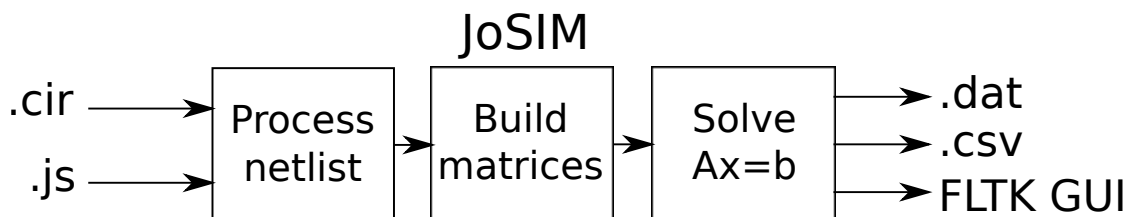


Figure 1: A macro overview of JoSIM

1.2 Initial setup

JoSIM can be found at [JoSIM.git](#) from where it can be cloned and compiled for either UNIX or Windows. Within this repository there will be a **CMakeLists.txt** which is a recipe used to compile JoSIM using CMake.

To compile the source a working C++ compiler with support for C++14 is required. Additionally SuiteSparse linear algebra libraries are required but are provided in the repository. Git version control software is recommended but is not required to compile JoSIM. Furthermore, the user has the option to enable additional features for plotting. These features are disabled by default, but would require either FLTK or Matplotlib (through Python) and can be enabled/disabled during compilation.

A single executable binary is generated using the CMake recipe and can be placed anywhere on the system as well as freely redistributed.

1.3 License

JoSIM is governed by the MIT license which is a very permissive license that allows anyone to redistribute it as well as commercialize it without repercussions. The MIT license allows use of this software within proprietary software as long as all copies of the licensed software includes a copy of the MIT license as well as the copyright notice.

1.4 Building from source

1.4.1 UNIX

These instructions were executed on a vanilla install of CentOS 7 Minimal to reduce oversight in the compilation instructions created by previous package installs. For other distributions please use the package manager relevant to the distribution of choice.

1.4.1.1 CentOS 7 Minimal Environment Set-up

A working internet connection is required, this might be enabled by default in other distributions however from Minimal the user would be required to run:

```
$ sudo ifup <network interface>
```

Where the network interface can be identified using the *ip addr show* command.

Once the internet connection is up and running, install git version control software using the command:

```
$ sudo yum install git
```

Thereafter, navigate to a directory where compilation will take place and execute:

```
$ git clone https://github.com/JoeyDelp/JoSIM.git
```

For the minimal install the user would also need to install standard development tools by executing:

```
$ sudo yum groupinstall "Development Tools"
```

This will install the latest version of *gcc* and *make* tools available to CentOS 7. Additionally, CMake 3 will be required to compile this tool properly and would require the activation of the *epel-release* repository.

```
$ sudo yum install epel-release  
$ sudo yum install cmake3
```

The basic version without graphical plotting, requires only a single external library SuiteSparse. Install it using the command:

```
$ sudo yum install suitesparse suitesparse-devel
```

Navigate to the newly cloned/extracted JoSIM directory then run the following commands:

```
$ mkdir build
$ cd build
$ cmake3 ..
$ make
```

This will generate a JoSIM executable in the *build* directory.

1.4.1.2 CentOS 7 Graphical Environment Set-up

JoSIM offers the ability to do visual plotting of the results through either the FLTK graphical library or the Python based Matplotlib library. The former will require the installation of the FLTK libraries using the command:

```
$ sudo yum install mesa-libGL-devel fltk fltk-fluid fltk-
    devel
```

FLTK plotting can be enabled using the cmake flag during the build step in the previous section:

```
$ cmake3 -DUSING_FLTK=1 ..
```

The latter Matplotlib library requires that the Python pip, devel, numpy and matplotlib be installed using:

```
$ sudo yum install python-pip python-devel tkinter tk tk-
    devel
$ sudo pip install numpy matplotlib
```

Matplotlib plotting can be enabled using the cmake flag during the build step in the previous section:

```
$ cmake3 -DUSING_MATPLOTLIB=1 ..
```

1.4.1.3 Apple macOS

Apple macOS is very similar to most Unix systems and therefore follows much the same procedure. The user would clone the repository and install CMake as well as the necessary libraries as indicated in previous sections. These libraries can be installed using either Homebrew, Macports or compiled from source using the standard macOS compilers. Much effort is made in the CMake file for JoSIM to attempt to identify the location of the required libraries. If the libraries cannot be found then alterations to the CMake file would need to be made to suit the users unique circumstances.

1.4.2 Windows

A Microsoft Visual Studio solution is provided and can be found in the *src* folder. This is by far the easiest way to compile the software under a Microsoft Windows environment. Simply open the solution and click build (F6) to build either Debug or Release targets for the software.

There are several configurations that allow for each of the graphical engines as well as the one without. FLTK static libraries for Windows are distributed with the source of JoSIM and will allow for the seamless compilation for this configuration.

Python on the other hand will require additional setup by the user. The easiest method of installing Python and the only one tested for this software is Anaconda Python. Either of the two versions 3.6 or 2.7 will work as the software has been tested to work on both. Version 3.6 is recommended as it boasts improvements to the UI elements of Matplotlib, which allow additional functionality.

To execute the Python version the following variables will need to be set/created in the Windows environment:

- PYTHONHOME = Anaconda directory
- QT_QPA_PLATFORM_PLUGIN_PATH = %PYTHONHOME%/Library/plugins/platforms

The executables once compiled can be found under *bin/win* followed by the chosen configuration.

2 Technical discussion

2.1 Modified nodal analysis

There are many ways to set-up a set of linear equations to solve the voltage or currents in a circuit. One of the more well known ways is to use nodal analysis which creates an equation for each node defined in the circuit netlist. This method is the basis on which the original Berkeley SPICE[2] was built. This method however only calculates the voltages of every node which makes it difficult to handle components that are voltage dependent such as inductors and junctions.

This drawback lead to the creation of the modified nodal analysis which is an extension to the prior with the ability of calculating some of the branch currents in the circuit.[3] We therefore make use of the MNA to build the set of linear equations within JoSIM due to the large use of inductors as well as Josephson junctions in superconductivity.

Another useful feature of MNA is the way that every component can be represented as a sub-matrix we call a stamp. The summation of all the stamps provide us with the A, x as well as b matrices that are required to solve the linear equations. These stamps will be discussed further in the following subsection.

2.2 Trapezoidal integration

Much like the nodal analysis mentioned before, there are multiple methods of solving differential equations with minimal error in a digital system. The most basic method is the forward Euler method which is a first-order approximation method where the error is proportional to the step size.[4]. Solving differential equations however produce a local truncation error which is the difference between the numerical solution and the exact solution after one step. It can be shown that the local truncation error for the forward Euler method is proportional to the square of the time step taken which makes this method less accurate compared to higher order methods.

We focus rather on using the trapezoidal rule for solving the linear equations as this method becomes increasingly more accurate as the time steps become smaller. The trapezoidal rule is a second-order method for solving differential equations. We can express the trapezoidal method as:

$$\left(\frac{dx}{dt}\right)_n = \frac{2}{h_n} (x_n - x_{n-1}) - \left(\frac{dx}{dt}\right)_{n-1} \quad (1)$$

In this case the n is the current time step and $n - 1$ refers to the previous time step. By using this method to solve differential equations we are able to create generic stamps for each component that JoSIM can handle.

To demonstrate this method and how a stamp is formed we will show an example of an

inductor. The inductor in Fig.2 has a general equation to determine the voltage across

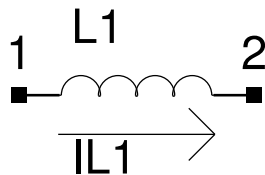


Figure 2: A basic inductor with current flowing through it

it as:

$$v_{L1}(t) = L1 \frac{di}{dt} \quad (2)$$

When we apply the trapezoidal rule we find (2) can be rewritten as:

$$\begin{aligned} v_n &= L1 \left[\frac{2}{h_n} (I_n - I_{n-1}) - \left(\frac{di}{dt} \right)_{n-1} \right] \quad (3) \\ &= \frac{2L1}{h_n} (I_n - I_{n-1}) - L1 \left(\frac{di}{dt} \right)_{n-1} \\ &= \frac{2L1}{h_n} (I_n - I_{n-1}) - v_{n-1} \end{aligned}$$

$$\therefore I_n = \frac{h_n}{2L1} (V_n + V_{n-1}) + I_{n-1} \quad (4)$$

$$\therefore \frac{h_n}{2L1} V_n - I_n = -\frac{h_n}{2L1} V_{n-1} + I_{n-1} \quad (5)$$

Where (5) is the current step voltage and current as a function of the previous step values. We further expand the (5) by stating that the voltage is the potential across the two nodes: $v = v_1 - v_2$.

$$\frac{h_n}{2L1} (V_1)_n - \frac{h_n}{2L1} (V_2)_n - I_n = -\left(\frac{h_n}{2L1} V(V_1)_{n-1} - \frac{h_n}{2L1} (V_2)_{n-1} \right) - I_{n-1} \quad (6)$$

Which we can then write in matrix form as:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & \frac{-2L1}{h_n} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_{L1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L1}{h_n} (I_{L1})_{n-1} - ((V_1)_{n-1} - (V_2)_{n-1}) \end{bmatrix} \quad (7)$$

(7) is a generic stamp that we can place into the A matrix which describes the inductor L1.

We can do the same for a resistor, capacitor, current source, voltage source, Josephson junction and a transmission line. The stamps for each of these components can be found in Appendix

2.3 LU decomposition

When the A matrix has been set up as detailed in the previous section all that is left to do is to solve the $Ax = b$ problem using some form of iterative method. We choose KLU from the SuiteSparse[5] library to accomplish this task.

This requires the A matrix to be in compressed row storage (CRS) format which is a data structure of 3 vectors. The first of these vectors contains all the non-zero elements in the A matrix. The second contains first a 0 followed the total number of non-zero elements after each row such that the final entry in the vector is the total number of non-zero elements. The third vector contains the column index of each non-zero element. As an example, the following sparse matrix of 5×5

$$\begin{bmatrix} 1 & 0 & 0 & 4 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

would yield a CSR format data structure of

$$\begin{aligned} nnz &= [1, 4, 3, 2, 1, 4, 5, 1] \\ rowptr &= [0, 2, 4, 5, 7, 8] \\ colind &= [0, 3, 1, 2, 2, 1, 3, 4] \end{aligned}$$

which has a total of 21 elements compared to the 25 required for the original A matrix. This difference of course becomes much larger the larger the A matrix becomes as well as the sparser the matrix becomes. Which is almost always the case for electrical type simulations.

Once in this format we can proceed with the KLU factorization. Due to the MNA forcing only the RHS to change upon every time step we can easily do the LU decomposition for the A matrix only once at the beginning of the time loop. Where after the system is simply solved using with a new RHS upon each iteration.

2.4 Data structures & speed considerations

JoSIM relies heavily on the use of the C++ unordered map data structure which creates a hash table for quick lookup. This immensely simplifies code legibility as well as component identification within later stages of the process.

Initially a standard map was used, however this very negatively impacted the speed of execution of the program and thus alternatives were sought. Ideally the use of unordered maps are not perfect if speed was the only consideration, however they do provide a good balance between implementation, speed and debugging.

There is still large room for improvement with regards to parallel processing of specific stages within the execution, however the largest part of the process (the time loop) is very loop dependent and therefore cannot be parallel processed. This also limits the speed improvement to relatively large circuits as the overhead required for thread delegation slows down the execution on smaller problems.

2.5 FLTK

JSIM lacked the ability to provide direct feedback on the results of the simulation. Unless the user had some script to plot the data file that it produced or some third-party application that could plot the results no clear feedback was received.

With JoSIM we introduce the implementation of the cross-platform C++ GUI library FLTK[6] which allows the results of the simulation to be instantly plotted in a data graph window. Though the full extent of the FLTK library has not been explored. The current implementation acts as a proof-of-concept design.

2.6 Matplotlib

With the complexity of FLTK yet to be explored an additional option for plotting the results has been provided. This method utilizes a C++ header only that interfaces with Python to display the results in a Matplotlib window. This method of plotting instantly allows the user to resize, pan and save the results. File formats include SVG, PNG, EPS and many more.

Though easy to implement and interpret, the method requires a lot of additional setup for each individual system and becomes very hard to debug if errors occur. This method is also limited to only the functionality provided by Matplotlib and alternatives will be looked at in the future.

2.7 Modified nodal phase analysis

First introduced in version 2.0 of JoSIM, the ability to perform a simulation that calculates the nodal phase instead of voltage is presented. This new analysis method is named the modified nodal phase analysis (MNPA) and utilizes the voltage-phase [7] relationship seen in (8).

$$v = \frac{\Phi_0}{2\pi} \frac{d\phi}{dt} \quad (8)$$

If this relationship is applied to all the component models found in JoSIM we obtain the MNPA stamps, which allow us to solve the phase directly. As example we demonstrate

this on the inductor equation (2).

$$\frac{\Phi_0}{2\pi} \frac{d\phi}{dt} = L1 \frac{di}{dt} \quad (9)$$

$$\frac{\Phi_0}{2\pi} \phi_n = L1 I_n \quad (10)$$

$$\frac{\Phi_0}{2\pi L1} \phi_n^+ - \frac{\Phi_0}{2\pi L1} \phi_n^- = I_n \quad (11)$$

Which is functionally equivalent to the resistor in voltage analysis. The computation required to solve the phase of an inductor is therefore far less complex than that of solving the voltage. With superconducting circuits being largely inductive, the use of phase reduces the overall complexity of the solution which in turn provides faster simulations and reduced memory usage.

JoSIM has been adapted to allow phase analysis on any design that works with voltage analysis without requiring alterations to the netlist file. Since the voltage is simply the scaled time derivative of the phase the voltage can be calculated as a post process if the user requests it.

Through the implementation of the MNPA method in JoSIM, additional improvements were introduced such as the objectification of every component at the matrix creation level that increases efficiency and debugging. This change allows for simplification of plotting functions since a direct link to the rows of the result matrix for each component reduces the indexing time required to identify the correct row.

Phase mode simulation can be enabled using the command line switch *-a* followed by a 1. If not provided the default for this command is 0, which indicates a voltage mode simulation will be performed.

3 Input files

Input files follow a SPICE syntax similar to JSIM. This allows for a direct use of the files that are already executable using JSIM as well as tools that write output files for use with JSIM.

Each SPICE file (*.cir* or *.js*) can be broken into subsections such as comments, subcircuits, main circuit and control sections. Each individual line is categorized by the first character. This is key to the efficiency of the entire tool as that single first character defines what the program does with that line as the file is read in line by line.

An example of an input file is given in the example section of this document.

3.1 JSIM Notation

Initially JoSIM only supported JSIM notation and follows a lot of the syntax listed in the JSIM manual found here. Do note though that not all the syntax in this file is supported, therefore only use it as a template.

A comprehensive syntax guideline will be compiled in the future as the tool becomes more stable and all the required features are supported.

3.2 WRSpice Notation

In addition to JSIM, WRSpice syntax support has been partially added in recent releases. This allows the simulation of *.cir* files generated using WRCAD XIC. The *.param* control card that allows parameterization of component values as well as variable declarations within netlists has also been added.

Once again not all WRSpice syntax is supported and it is requested that the user work cautiously until a comprehensive syntax guide is available.

3.3 Subcircuit Convention

A key difference between WRSpice and JSIM is the way that subcircuit declarations are handled. JSIM includes the name of the subcircuit between the label and the first node number. Where WRSpice puts this name at the end of the line, after the node numbers. Due to JoSIM handling nodes as alphanumeric instead of simply numeric, it becomes incredibly hard to discern between node and subcircuit name. We therefore include a command argument for convenience that specifies the subcircuit convention. The default would be 0, meaning JSIM and can be set to 1 for WRSpice. This is done using the *-c 1* command argument. If the user is using JSIM syntax this argument can be completely omitted.

4 Command line arguments/switches

A full range of the available command switches and their arguments is provided to the user upon specification of the `-h` switch.

```

JoSIM: Josephson Junction Superconductive SPICE Circuit Simulator
Copyright (C) 2018 by Johannes Delpoort (jdelpoort@sun.ac.za)
v2 compiled on Oct 10 2018 at 13:28:42
Plotting engine: NONE
Parallelization is DISABLED

JoSIM help interface
=====
-a(nalysis) | Specifies the analysis type.
              | 0 for Voltage analysis (Default)
              | 1 for Phase analysis
-c(onvention) | Sets the subcircuit convention to left(0) or right(1).
              | Default is left. WRSpice (normal SPICE) use right
              | Eg. X01 SUBCKT 1 2 3 vs. X01 1 2 3 SUBCKT
-g(raph) | Plot the requested results using a plotting library
          | If this is enabled with verbose mode then all traces are plotted
-h(elp) | Displays this help menu
-o(utput) | Specify output file for simulation results (.csv)
          | Default will be output.csv if no file is specified
-m(atlab) | [Legacy] JSIM_N output file format (.dat)
          | Default will be output.dat if no file is specified
-v(erbose) | Runs JoSIM in verbose mode
--v(ersion) | Displays the JoSIM version info only

Example command: josim -g -o ./output.csv test.cir

```

Figure 3: JoSIM help command menu

Each help menu item is pretty self explanatory and should not require further documentation. These command switches can be placed in any order and can be concatenated so long as the final command is the input file.

5 Output

5.1 Comma seperated value

The output to a comma separated value format file was chosen due to the large support therefore and ease of import into third-party tools such as Microsoft Excel. This simplifies the movement, copying and removal of columns from the output file a lot simpler as well.

5.2 Space seperated value

This is the legacy format that JSIM uses to output it's results into. We opt to support this file format due to the large user base of JSIM and the amount of plotting/viewing tools written around this output format. This output format can also be easily read in and manipulated by third party tools and we shall therefore keep this option with the same command switch as JSIM.

5.3 GUI plotting window

The output of results directly into a plotting window that is user viewable improves the prototyping speed immensely as the user can immediately see whether the output is as desired or not. Though primitive at present with the lack of any axis or tooltips to show that the values of each graph are as well as the lack of the ability to resize the graphs within the window, we will continue to expand on this functionality to eventually have a user-friendly output window.

6 Examples

Below is an example file that Chains together a DCSFQ, 3 x JTLs and a SINK cell.

```

1 .SUBCKT JTL 4 5
2 B01 3 7 jj1 area=2.16
3 B02 6 8 jj1 area=2.16
4 IB01 0 1 pwl(0 0 5p 280u)
5 L01 4 3 2.031p
6 L02 3 2 2.425p
7 L03 2 6 2.425p
8 L04 6 5 2.031p
9 LP01 0 7 0.086p
10 LP02 0 8 0.086p
11 LPR01 2 1 0.278p
12 LRB01 7 9 1p
13 LRB02 8 10 1p
14 RB01 9 3 5.23
15 RB02 10 6 5.23
16 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
17 .ends JTL
18 .SUBCKT DCSFQ 2 17
19 B01 5 3 jj1 area=1.32
20 B02 5 6 jj1 area=1
21 B03 9 10 jj1 area=1.5
22 B04 13 14 jj1 area=1.96
23 B05 15 16 jj1 area=1.96
24 IB01 0 8 pwl(0 0 5p 162.5u)
25 IB02 0 12 pwl(0 0 5p 260u)
26 L01 2 1 0.848p
27 L02 0 1 7.712p
28 L03 1 3 1.778p
29 L04 5 7 0.543p
30 L05 7 9 3.149p
31 L06 9 11 1.323p
32 L07 11 13 1.095p
33 L08 13 15 2.951p
34 L09 15 17 1.63p
35 LP01 0 6 0.398p
36 LP02 0 10 0.211p
37 LP03 0 14 0.276p
38 LP04 0 16 0.224p
39 LPR01 7 8 0.915p
40 LPR02 11 12 0.307p
41 LRB01 4 5 1p
42 LRB02 18 6 1p
43 LRB03 19 10 1p
44 LRB04 20 14 1p
45 LRB05 21 16 1p
46 RB01 3 4 8.56
47 RB02 18 5 11.30
48 RB03 19 9 7.53
49 RB04 20 13 5.77
50 RB05 21 15 5.77
51 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
52 .ends DCSFQ
53 .SUBCKT SINK 2
54 B01 1 4 jj1 area=2.16
55 IB01 0 5 pwl(0 0 5p 280u)
56 L01 2 1 0.517p
57 L02 1 3 5.307p
58 LP01 0 4 0.086p
59 LPR01 1 5 0.265p
60 LRB01 4 6 1p
61 RB01 6 1 5.23
62 ROUT 0 3 4.02
63 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
64 .ends SINK
65 IA 0 1 pwl(0 0 170p 0 176p 600u 182p 0 370p 0 376p 600u 382p 0 600p 0 606p 600u 612p 0 700p 0
706p 600u 712p 0)
66 X01 DCSFQ 1 2
67 X02 JTL 2 3
68 X03 JTL 3 4
69 X04 JTL 4 5
70 X05 SINK 5
71 .tran 0.25p 1000p 0 0.25p
72 .print nodev 1 0
73 .print nodev 3 0
74 .print nodev 5 0

```

This file can be found under *JoSIM/test/dcsfq-jtl-sink.js* and we will now demonstrate

this example using JoSIM.

```
[$ ./JoSIM_mac_Debug -g -o ../test/dcsfq_jtl_sink.js

JoSIM: Josephson Junction Superconductive SPICE Circuit Simulator
Copyright (C) 2018 by Johannes Delpoort (jdelport@sun.ac.za)
v1.2 compiled on Jul 12 2018 at 15:33:10
Plotting engine: MATPLOTLIB
Parallelization is DISABLED

Path specified for input file: ../test/dcsfq_jtl_sink.js
Input file specified as: dcsfq_jtl_sink.js

Circuit characteristics:
Subcircuits:                3

SINK component count:      9
SINK JJ count:             1
DCSFQ component count:    32
DCSFQ JJ count:           5
JTL component count:      14
JTL JJ count:             2

Main circuit component count: 84
Main circuit JJ count:     12

Simulating:
100%[=====]
```

Figure 4: JoSIM command line window output from execution of the dcsfq_jtl_sink.js

Fig 4 shows the default output from the JoSIM executable produces some statistics about the file provided as input and the reuse of subcircuits within the file. It also gives the user a count of the amount of Josephson junctions used within the circuit as this is usually taken as a measure of how large the circuit is.

The results of the *-g* command switch can be seen in Fig 5 and 6. Which we can clearly see from is the results expected given the type of circuit. The 3 graphs that are plotted are the node voltages at the input to the DCSFQ, the middle JTL and the input of the SINK.

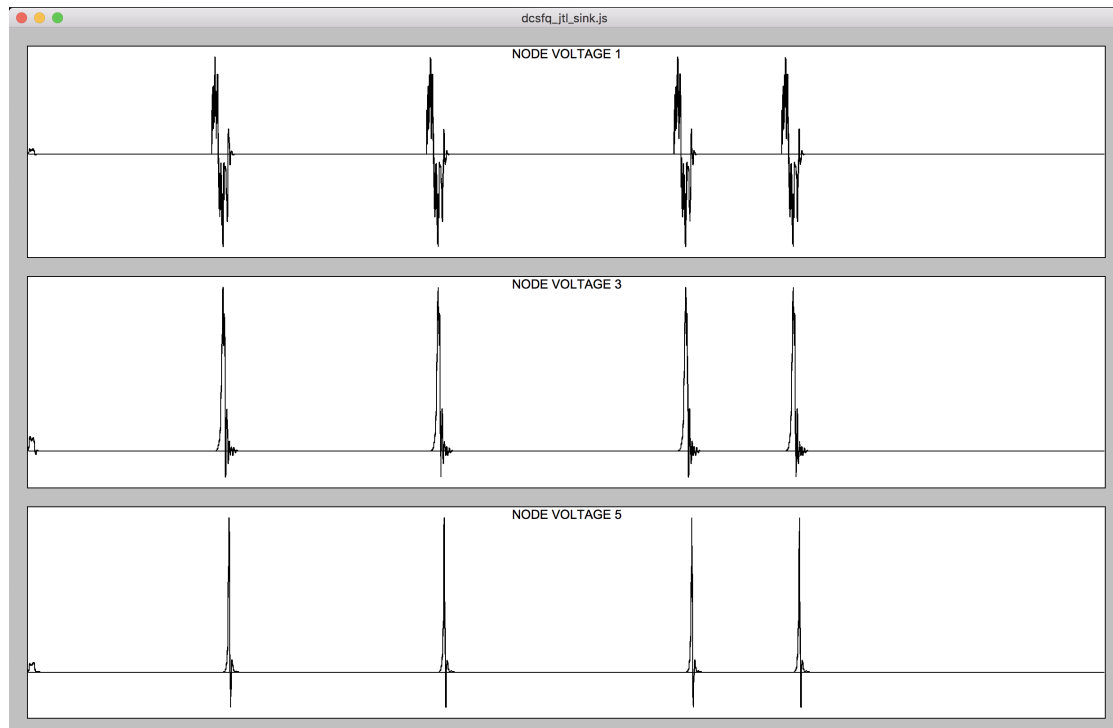


Figure 5: JoSIM FLTK results window from execution of the dcsfq_jtl_sink.js file

There are a few larger examples included in the test folder which can be executed to test the execution speed of JoSIM.

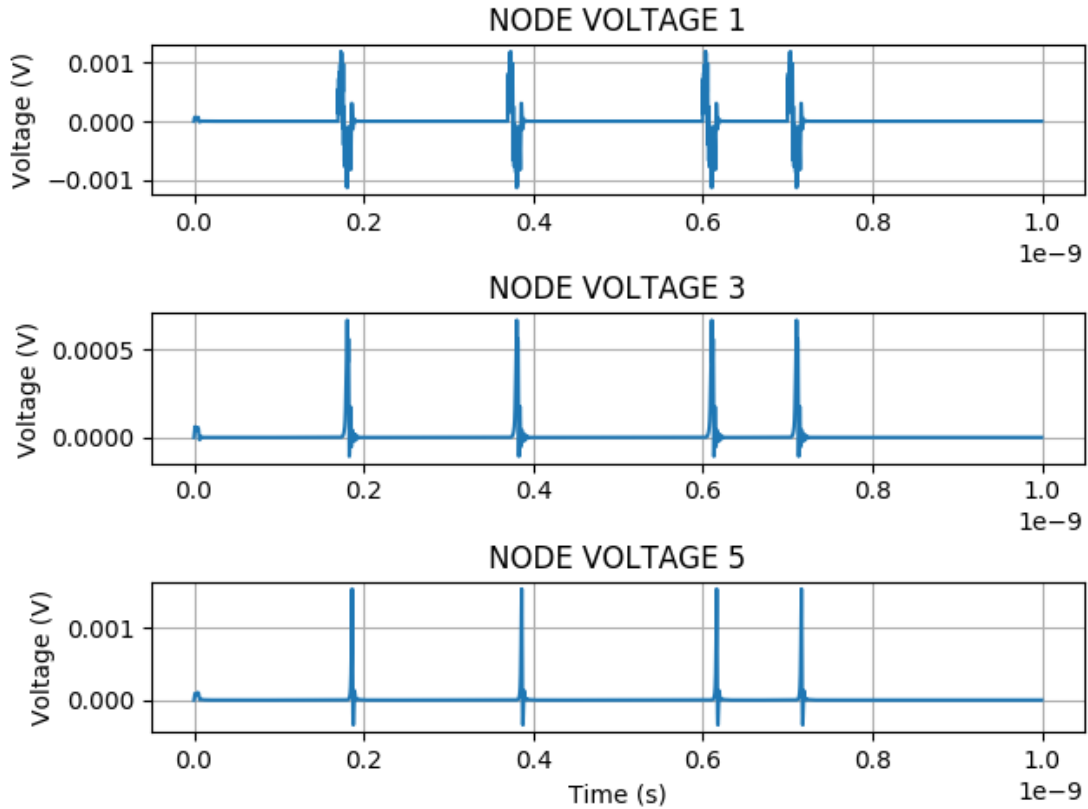


Figure 6: JoSIwwM Matplotlib results window from execution of the `dcsfq_jtl_sink.js` file

7 Error handling and exceptions

An attempt is made to handle as many of the exceptions thrown as possible and to provide the user with accurate and meaningful error messages in the case of an error. If any issues are not caught or bugs are picked up during the usage of JoSIM, please do not hesitate to report them on the git page under issues. The software has had very little exposure until now. It is a version 1 tool and there will be bugs, please use with caution and please report any issues.

8 Planned improvements

There are several planned improvements to JoSIM which include per device temperature dependence as well as global temperature to more accurately model the effects of superconductive materials.

Engine improvements will include new Josephson junction models that extend beyond the default RCSJ model, as well as the ability to simulate the Werthamer approximated

model for the Josephson junction. Further engine improvements will also include parallel processing of certain stages within the execution as well as the ability to switch to different linear solvers.

Optimizations will be investigated to further limit the memory impact JoSIM has on the host system. Among the areas of investigation will be the effect of identifying the vectors the user wishes to save before starting the simulation. In doing so only these identified vectors need to be saved for the entire simulation and only two or 3 time steps worth of data would need to be kept. This should improve memory impact significantly on large simulations. The only foreseen complication is when transmission lines are present but this can also be circumvented.

Appendices

A Component Stamps

A.1 Resistor

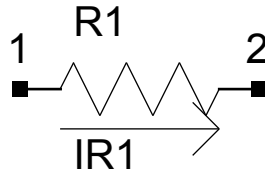


Figure 7: A basic resistor with current flowing through it

$$\begin{aligned}
 v_{R1}(t) &= i_{R1}(t)R1 \\
 (V_1 - V_2)_n &= I_n R1 \\
 \frac{1}{R1}(V_1)_n - \frac{1}{R1}(V_2)_n - I_n &= 0
 \end{aligned} \tag{12}$$

We however do not need to calculate the current through the Resistor as it is not always needed, we therefore omit it in the matrix and calculate it when the user requests it.

$$\begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

A.2 Capacitor

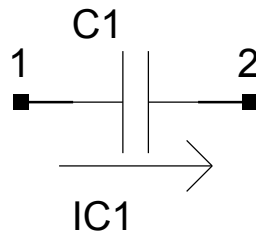


Figure 8: A basic capacitor with current flowing through it

$$\begin{aligned}
 i_n(t) &= C1 \frac{dv}{dt} \\
 i_n(t) &= \frac{2C1}{h_n} \left[(V_n - V_{n-1}) - \left(\frac{di}{dt} \right)_{n-1} \right] \\
 \frac{2C1}{h_n}(V_1)_n - \frac{2C1}{h_n}(V_2)_n - I(n) &= \frac{2C1}{h_n} (V_1 - V_2)_{n-1} + I_{n-1}
 \end{aligned} \tag{13}$$

Once again we omit the current for the capacitor as it will only complicate the calculation. We therefore only calculate the voltage output.

$$\begin{bmatrix} \frac{C}{h_n} & -\frac{C}{h_n} \\ -\frac{C}{h_n} & \frac{C}{h_n} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \frac{C}{h_n} \\ -\frac{C}{h_n} \end{bmatrix}$$

A.3 Voltage source

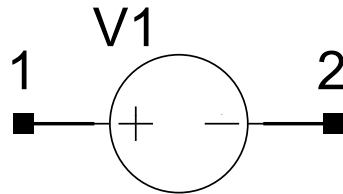


Figure 9: A basic voltage source

A voltage source simply adds the voltage as a new row and column to the matrix, similar to a branch current.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V_1 \end{bmatrix}$$

A.4 Current source

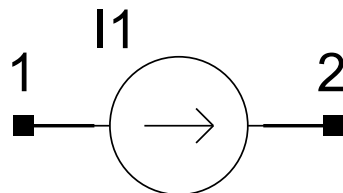


Figure 10: A basic capacitor with current flowing through it

A current source does not affect the A matrix in any way and simply adds or subtracts the current supplied at the respective nodes.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} -I_1 \\ I_1 \end{bmatrix}$$

A.5 Josephson junction

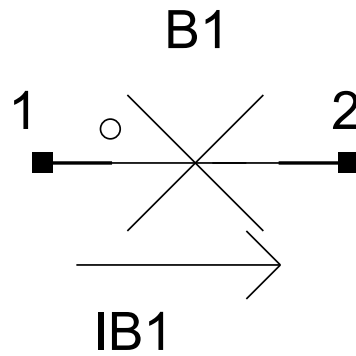


Figure 11: A basic Josephson junction

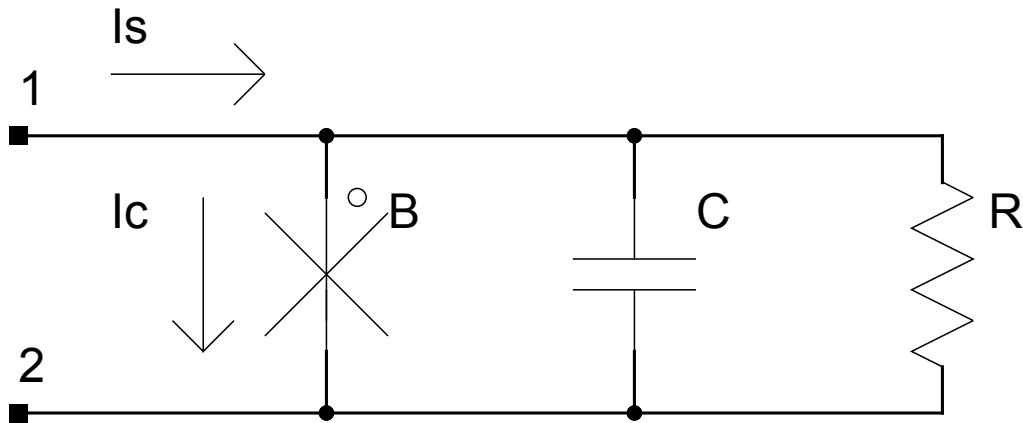


Figure 12: A basic Josephson junction in Resistively, Capacitively Shunted Junction form

$$I_s = -I_c \sin \phi_n^0 + \frac{2C}{h_n} v_{n-1} + C \left(\frac{dv}{dt} \right)_{n-1} \quad (14)$$

In this case I_c , C , ϕ_n^0 are the junction critical current, capacitance and initial phase guess.

$$\phi_n^0 = \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\hbar} (v_{n-1} + v_n^0) \quad (15)$$

$$\begin{aligned}
v_n^0 &= v_{n-1} + h_n \left(\frac{dv}{dt} \right)_{n-1} \\
&= v_{n-1} + h_n \left[\frac{2}{h_n} (v_{n-1} - v_{n-2}) - \left(\frac{dv}{dt} \right)_{n-2} \right] \\
v_n^0 &= 3v_{n-1} - 2v_{n-2} - h_n \left(\frac{dv}{dt} \right)_{n-2}
\end{aligned} \tag{16}$$

When we apply 16 to 15 we can approximate an equation for the initial phase guess. To approximate this we would need to specify values for 3 time steps before the first time step. By assuming these values are always 0 we can then implement this as a stamp.

$$\begin{bmatrix} \frac{2C}{h_n} + \frac{1}{R} & -\frac{2C}{h_n} - \frac{1}{R} & 0 \\ -\frac{2C}{h_n} - \frac{1}{R} & \frac{2C}{h_n} + \frac{1}{R} & 0 \\ -\frac{h_n}{2} \frac{2\epsilon}{h} & \frac{h_n}{2} \frac{2\epsilon}{h} & 1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \phi \end{bmatrix} = \begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2} \frac{2\epsilon}{h} v_{n-1} \end{bmatrix}$$

A.6 Transmission line

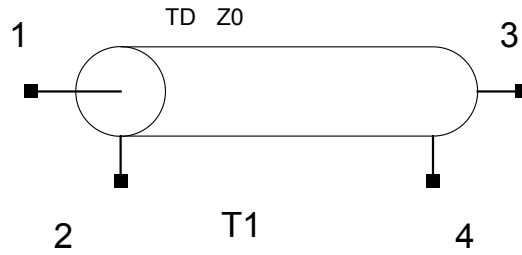


Figure 13: A basic lossless transmission line

The implementation of a lossless transmission line requires the everything on the right-hand side to be delayed by the TD specified by the user. The Z_0 is the line impedance and is a function of the transmission line length.

The voltage on between node 1 and 2 is delayed by TD before appearing at node 3 and 4, where after a reflection of the result at node 3 and 4 is observed at node 1 and 2. This effect continues every TD until the voltage is completely diminished by the line impedance.

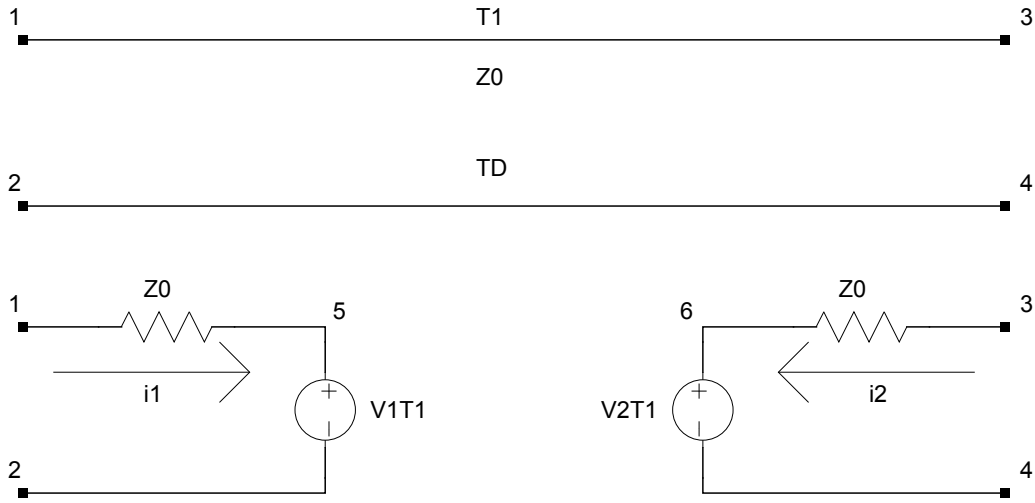


Figure 14: A basic lossless transmission line broken into components

The dependent voltage sources on either side of the transmission line can be described by

$$v_1(t) - Z_0 i_1(t) = v_2(t - TD) + Z_0 \cdot i_2(t - TD) \quad (17)$$

$$v_2(t) - Z_0 i_2(t) = v_1(t - TD) + Z_0 \cdot i_1(t - TD) \quad (18)$$

which leads to

$$(V_1^+) - (V_1^-) - Z_0(I_1) = (V_2)_{n-k} + Z_0(I_2)_{n-k} \quad (19)$$

$$(V_2^+) - (V_2^-) - Z_0(I_2) = (V_1)_{n-k} + Z_0(I_1)_{n-k} \quad (20)$$

with

$$k = \frac{TD}{h} \quad (21)$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -Z_0 & 0 \\ 0 & 0 & 1 & -1 & 0 & -Z_0 \end{bmatrix} \begin{bmatrix} V_1^+ \\ V_1^- \\ V_2^+ \\ V_2^- \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ (V_2)_{n-k} + Z_0(I_2)_{n-k} \\ (V_1)_{n-k} + Z_0(I_1)_{n-k} \end{bmatrix}$$

A.7 Mutual Inductance

Basic inductance equations for mutual inductance

$$V_{L_1} = L_1 \frac{di_1}{dt} + M \frac{di_2}{dt} \quad (22)$$

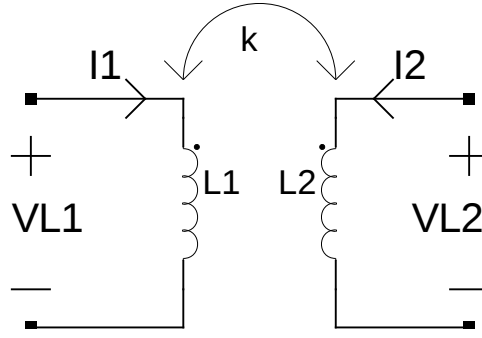


Figure 15: Standard mutual inductance

$$V_{L_2} = M \frac{di_1}{dt} + L_2 \frac{di_2}{dt} \quad (23)$$

M can be defined as the mutual inductance multiplied by a coupling factor k

$$M = k\sqrt{L_1 L_2} \quad (24)$$

Using the standard trapezoidal integration method on (22) and (23)

$$\left(\frac{dx}{dt}\right)_n = \frac{2}{h_n} (x_n - x_{n-1}) - \left(\frac{dx}{dt}\right)_{n-1} \quad (25)$$

we find

$$V_{L_1(n)} = \frac{2L_1}{h_n} (I_{L_1(n)} - I_{L_1(n-1)}) + \frac{2M}{h_n} (I_{L_2(n)} - I_{L_2(n-1)}) - V_{L_1(n-1)} \quad (26)$$

$$V_{L_2(n)} = \frac{2M}{h_n} (I_{L_1(n)} - I_{L_1(n-1)}) + \frac{2L_2}{h_n} (I_{L_2(n)} - I_{L_2(n-1)}) - V_{L_2(n-1)} \quad (27)$$

If we are to split the equations in (26) and (27) to a LHS and RHS. Where the LHS is what we want to calculate and the RHS is what we have from previous time steps.

$$V_{L_1(n)} - \frac{2L_1}{h_n} I_{L_1(n)} - \frac{2M}{h_n} I_{L_2(n)} = -\frac{2L_1}{h_n} I_{L_1(n-1)} - \frac{2M}{h_n} I_{L_2(n-1)} - V_{L_1(n-1)} \quad (28)$$

$$V_{L_2(n)} - \frac{2M}{h_n} I_{L_1(n)} - \frac{2L_2}{h_n} I_{L_2(n)} = -\frac{2M}{h_n} I_{L_1(n-1)} - \frac{2L_2}{h_n} I_{L_2(n-1)} - V_{L_2(n-1)} \quad (29)$$

Furthermore, we see that the voltage across the inductors are a voltage drop between two nodes.

$$V_{L_1(n)}^+ - V_{L_1(n)}^- - \frac{2L_1}{h_n} I_{L_1(n)} - \frac{2M}{h_n} I_{L_2(n)} = -\frac{2L_1}{h_n} I_{L_1(n-1)} - \frac{2M}{h_n} I_{L_2(n-1)} - V_{L_1(n-1)}^+ - V_{L_1(n-1)}^- \quad (30)$$

$$V_{L_2(n)}^+ - V_{L_2(n)}^- - \frac{2M}{h_n} I_{L_1(n)} - \frac{2L_2}{h_n} I_{L_2(n)} = -\frac{2M}{h_n} I_{L_1(n-1)} - \frac{2L_2}{h_n} I_{L_2(n-1)} - V_{L_2(n-1)}^+ - V_{L_2(n-1)}^- \quad (31)$$

We can now write these equations in standard matrix form as

$$\begin{bmatrix} 1 & -1 & -\frac{2L_1}{h_n} & -\frac{2M}{h_n} \end{bmatrix} \begin{bmatrix} V_{L_1(n)}^+ \\ V_{L_1(n)}^- \\ I_{L_1(n)} \\ I_{L_2(n)} \end{bmatrix} = \begin{bmatrix} -\frac{2L_1}{h_n} I_{L_1(n-1)} - \frac{2M}{h_n} I_{L_2(n-1)} - V_{L_1(n-1)}^+ - V_{L_1(n-1)}^- \end{bmatrix} \quad (32)$$

$$\begin{bmatrix} 1 & -1 & -\frac{2M}{h_n} & -\frac{2L_2}{h_n} \end{bmatrix} \begin{bmatrix} V_{L_2(n)}^+ \\ V_{L_2(n)}^- \\ I_{L_1(n)} \\ I_{L_2(n)} \end{bmatrix} = \begin{bmatrix} -\frac{2M}{h_n} I_{L_2(n-1)} - \frac{2L_2}{h_n} I_{L_2(n-1)} - V_{L_2(n-1)}^+ - V_{L_2(n-1)}^- \end{bmatrix} \quad (33)$$

Leading us to the MNA stamps

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & -1 & -\frac{2L_1}{h_n} & -\frac{2M}{h_n} \end{bmatrix} \begin{bmatrix} V_{L_1(n)}^+ \\ V_{L_1(n)}^- \\ I_{L_1(n)} \\ I_{L_2(n)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L_1}{h_n} I_{L_1(n-1)} - \frac{2M}{h_n} I_{L_2(n-1)} - V_{L_1(n-1)}^+ - V_{L_1(n-1)}^- \end{bmatrix} \quad (34)$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 1 & -1 & -\frac{2M}{h_n} & -\frac{2L_2}{h_n} \end{bmatrix} \begin{bmatrix} V_{L_2(n)}^+ \\ V_{L_2(n)}^- \\ I_{L_1(n)} \\ I_{L_2(n)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2M}{h_n} I_{L_2(n-1)} - \frac{2L_2}{h_n} I_{L_2(n-1)} - V_{L_2(n-1)}^+ - V_{L_2(n-1)}^- \end{bmatrix} \quad (35)$$

Which when collated gives us

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & -1 \\ 1 & -1 & -\frac{2L_1}{h_n} & -\frac{2M}{h_n} \\ 1 & -1 & -\frac{2M}{h_n} & -\frac{2L_2}{h_n} \end{bmatrix} \begin{bmatrix} V_{L_2(n)}^+ \\ V_{L_2(n)}^- \\ I_{L_1(n)} \\ I_{L_2(n)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L_1}{h_n} I_{L_1(n-1)} - \frac{2M}{h_n} I_{L_2(n-1)} - V_{L_1(n-1)}^+ - V_{L_1(n-1)}^- \\ -\frac{2M}{h_n} I_{L_2(n-1)} - \frac{2L_2}{h_n} I_{L_2(n-1)} - V_{L_2(n-1)}^+ - V_{L_2(n-1)}^- \end{bmatrix} \quad (36)$$

References

- [1] E. S. Fang and T. Van Duzer. *A Josephson integrated circuit simulator (JSIM) for superconductive electronics application*. Extended Abstracts of 1989 International Superconductivity Electronics Conference, 407-410, 1989.
- [2] Laurence W. Nagel and D.O. Pederson. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. EECS Department, University of California, Berkeley, 1973.
- [3] Ho, Ruehli, and Brennan. *The Modified Nodal Approach to Network Analysis*. Proc. 1974 Int. Symposium on Circuits and Systems, San Francisco. pp. 505509.
- [4] Atkinson, Kendall A. *An Introduction to Numerical Analysis (2nd ed.)* New York: John Wiley & Sons, 1989
- [5] Timothy A. Davis *Direct Methods for Sparse Linear Systems* SIAM, Philadelphia, Sept. 2006.
- [6] F. Costantini, D. Gibson, M. Melcher, A. Schlosser, B. Spitzak and M. Sweet. *FLTK 1.4.0 Programming Manual*. 2018. [Online]. Available: <http://www.fltk.org/>. [Accessed: 29- Apr- 2018].
- [7] T.P. Orlando and K.A. Delin. *Foundations of Applied Superconductivity*. Addison-Wesley Publishing Company. 1991. pp. 406.