Efficient simulation of Large-Scale Superconducting Nanowire Circuits

by

Tareq "Torque" El Dandachi

B.S. in Electrical Engineering and Computer Science and in Engineering
as Recommended by the Department of Mechanical Engineering

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2023

Author .....
Department of Electrical Engineering and Computer Science
January 27, 2023

Certified by .....
Karl K. Berggren
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Efficient simulation of Large-Scale Superconducting Nanowire Circuits

by

Tareq "Torque" El Dandachi

Submitted to the Department of Electrical Engineering and Computer Science
on January 27, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

As the size of superconducting nanowire devices increases and the influence of second-order effects, such as thermal or electrostatic coupling, becomes more significant, the complexity of models required to accurately and efficiently simulate the device's behavior becomes more challenging. Traditional circuit simulators used for superconducting devices tend to focus on frequency-domain simulation and are not optimized for simulating superconducting nanowire geometries in the time-domain. This thesis presents an integrated simulator environment designed with the goal of simulating superconducting nanowires. The work presented in this thesis introduces:

1. an integrated environment for SPICE software that extends its modeling capabilities optimized for superconducting nanowire devices and accompanying experiments;
2. a simple procedure to measure the stability of circuit models used to present an improved nanowire SPICE model; and
3. an efficient Julia-based simulator optimized for superconducting nanowire devices and nonlinear microwave circuits.

Thesis Supervisor: Karl K. Berggren
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to extend my deepest gratitude to my friends and colleagues, who have been a constant source of support and inspiration throughout my research. Their unwavering encouragement and advice has helped fuel my passion for research and their belief in me was instrumental to my ability to complete this thesis. I'd especially like to thank:

My advisor, Professor Karl Berggren, who introduced me to superconducting nanowires, and whose advice and expertise guided me throughout. His guidance was instrumental in shaping my research and I am deeply grateful for the opportunity to be part of his lab. The Quantum Nanostructures and Nanofabrication (QNN) group will forever hold a special place in my heart.

Marco Colangelo, who taught me everything I know about the measurement and fabrication of superconducting devices. He has and always will be my role model, not only in his expertise and knowledge but also in his dedication and spirit. His invaluable advice, feedback, and support was instrumental in shaping my work and who I am. I am deeply grateful to have had him as a mentor and colleague.

Professor Stefan Krastanov, an exceptional educator and mentor, who introduced me to the field of simulation, the Julia programming language, and sparked my love for optimization. His support and advice was instrumental in my pursuit of research and I will forever appreciate having done research under your supervision.

Reed Foster, my partner in crime, who I had the pleasure of meeting during my undergraduate studies at MIT, taking multiple classes together, conducting undergraduate research and starting our masters studies together at QNN. The conversations

we had in the shared undergraduate office and all our discussions on RF electronics, FPGAs and Julia will always hold a special place in my heart. I wish Reed all the best in his Ph.D. studies and hope our paths will cross again in the future.

Emma Batson and Adina Bechhofer, whom I'm lucky to call both friends and colleagues, for making the lab such a fun and enjoyable experience. Their indulgence of my love for optimization and theory made the research process so much more fulfilling.

John Simonaitis, Stewart Koppell, Owen Medeiros, Dip Joti Paul, Maurice Krielaart, Dorothy Fleischer and Alessandro Buzzi for their advice, daily discussions, and camaraderie.

Savoldy and Colin Greybosh, who listened to me obsess, complain and fall in love with my work every single day.

Ayo Lindblad for all our lunches and chats that fueled my love for math.

And finally, to my dear friend, Ashika Verma, for always believing in me and pushing me to be a better person.

Contents

List of Figures

Chapter 1

Introduction

The non-linear behavior of superconducting nanowires is a crucial aspect of many of their applications, including superconducting nanowire single-photon detectors (SNSPDs) and neuromorphic computing [1, 2]. However, this non-linearity also makes it challenging to simulate nanowires, particularly when considering their microwave properties. One common method of simulating these nanowire electronics relies on existing circuit simulation environments [3]. These environments were optimized for classical electronics and lack optimizations that account for the microwave and superconducting characteristics of the models.

While plenty of good superconducting simulators exist for the frequency domain modeling of devices, they tend to neglect effects that nanowire-based device designer care about [4, 5]. These effects include simulating pulses, thermal modeling of hotspot generation, and thermal coupling in stacks. Simulating the dominant effects in our superconducting electronics in the time domain is a crucial step for device design.

As we scale device sizes and introduce dependence on thermal effects and electrostatic coupling, the complexity of the models makes it increasingly difficult to accurately and efficiently simulate the device's time behavior properly. While the need for more accurate nanowire simulations and the complexity of our models continue to grow, the tools used to simulate these devices must also evolve to meet these challenges. We aim to address that by introducing wrappers around existing SPICE software, a method for quickly assessing simulation stability and present the building blocks for a new nanowire electronics simulator built in Julia.

1.1 Non-linearity in superconducting nanowires

Superconducting nanowires are highly non-linear and present three main forms of non-linearity: (1) kinetic inductance, (2) normal-superconducting state transitions and (3) coupling to other non-linear dynamics.

Kinetic inductance in superconducting nanowires is a continuous form of non-linearity introduced by the cooper pair dependence on the bias current in a nanowire [6]. In thin films, kinetic inductance is highly dependent on the film thickness and temperature [7]. A nanowire's inductance is almost entirely due to kinetic inductance, and therefore its reliance on the bias current is of importance [8]. Designing more complicated electronics and SNSPDs requires us to simulate the effects of current behavior other than DC (such as pulses) on the kinetic inductance. This effect is important as the non-linearity of the nanowire can change the shapes of pulses – this is a well-studied effect in non-linear transmission lines [9]. Accounting for the non-linear microwave properties causes even simple designs – such as a superconducting transmission line operating only in the superconducting regime – to behave in a

difficult-to-anticipate non-linear fashion.

The second form of non-linearity pertains to the superconducting state. By assuming the device is experiencing a constant temperature, there is a constant threshold critical current i_c where if the current exceeds i_c locally along a nanowire, it switches into the resistive state. This switching behavior is a non-linearity over a Boolean state that is dependent on the current flowing through each portion of the nanowire. Nonlinearities over a Boolean state are particularly hard to simulate as they involve sudden large magnitude changes. Typical non-linear solvers are optimized for continuous non-linear systems where the solver enters a loop making the time step smaller until the magnitude of change is small [10, 11]. In Boolean states, there is no sense of continuity, and in the limit of smaller time steps, the change in response magnitude will be just as large.

1.2 Nanowire Elements

From an electronics standpoint, a nanowire's lumped model is a non-linear inductor when superconducting. When resistive, an additional resistor is in series with that inductor. These two building blocks (a continuously non-linear inductor and a discrete non-linear resistance) are the basis for modeling the behavior of superconducting nanowires in the electronics picture. This model covers the two main types of nonlinearities exhibited by nanowires.

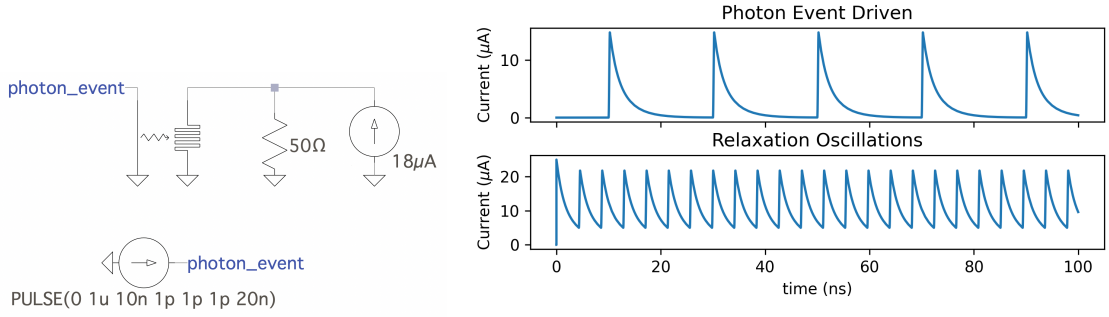
A more complicated - but sometimes necessary - picture includes coupling to a thermal equation. A nanowire's critical current i_c and critical temperature T_c are functions of the current state of the superconductor. These two parameters are related by the critical surface; leading to implications such as $T_c(i = 0) \neq T_c(i = 0.75i_c)$. The

superconducting-to-normal state transition begins a coupled chain reaction between a thermal system and an electrical system, making modeling nanowires harder. When a portion of the nanowire switches into the resistive state, a normal region starts to form in the wire that dissipates thermal energy. This energy heats up the surrounding portions of the nanowire, decreasing their critical current. At the same time, the normal region has a higher impedance than the nanowire diverting current around it, allowing portions of the nanowire to see a higher density of the current, making it more likely to switch in the plane of the hotspot. The hotspot also dissipates heat to the stack and fridge. These are well-studied phenomena for nanowires and tend to be modeled through experimentally fitted parameters [12, 3].

Another picture that tends to be neglected is the distributed picture of the nanowire. In reality, the nanowire has a spatial dimension to it and is a microwave device [13, 14]. This picture tends to enforce simulation constraints as the discretization and network size increase. This picture accounts for time delays introduced to a signal entering and leaving a nanowire, resonances that might occur inside the nanowire, as well as distributed thermal and electrical effects that cannot be replicated in a lumped single-element picture. For nanowire meanders longer than the wavelength of frequencies carried, modeling the nanowire as a distributed device is essential [13]. Not doing so neglects distributed behavior such as resonance and pulse reshaping, discussed in section 1.2.3.

1.2.1 SNSPDs

One geometry a nanowire circuit can be designed to be in is the superconducting nanowire single-photon detector (SNSPD) circuit. By having a nanowire meander biased near its critical current, a small energy perturbation (such as a photon in-



(a) (b)

Figure 1-1: (a) Typical SNSPD readout circuit with a 50 Ω shunt resistor. (b) Two operation regimes for a nanowire with $i_c = 21\,\mu A$ are shown. The top plot is 5 separate photon detection events using a bias of 18 µA. The bottom plot showcases relaxation oscillation done by biasing the nanowire with 25 µA.

cidence) can cause a transition into the normal state. As a result a single photon injecting a small amount of energy into the nanowire has an amplified output from a previously unimpeded bias current that now is flowing across a large resistor (usually on the order of 1 kΩ).

For simulating a standard SNSPD topology, it is enough to model the device using a lumped model and a shunt resistor in parallel for readout. Assuming an SNSPD with 50 Ω impedance shunted with a 50 Ω resistor, the current is split and equally diverted into the shunt and nanowire. If biased at the right threshold, a photon count would correspond to a tiny spike in the current flowing through the nanowire – "a switching event". The nanowire produces a voltage pulse as a $\sim 7\,k\Omega$ resistive region starts to form (this number is dependent on multiple design parameters). Due to the impedance mismatch, current is diverted into the shunt resistor, which allows the hotspot to cool off and resets the SNSPD [15].

In this topology, the nanowire can produce high frequency relaxation oscillations

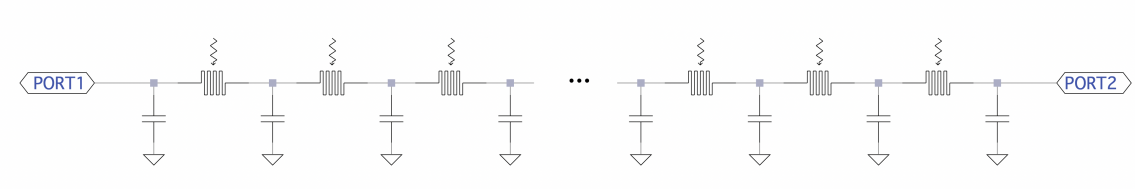


Figure 1-2: A lumped model approximation of an SNSPI meander. The model uses the same topology used to approximate distributed effects in linear transmission lines, swapping out the linear inductor with a non-linear nanowire.

when biased above the critical current [16]. These oscillations are caused by the current periodically being diverted into and out of the resistive shunt. These relaxation oscillations consist of periodically repeating SNSPD spikes, similar to a switching event.

1.2.2 SNSPIs

Superconducting nanowire single-photon imagers (SNSPIs) are used in a similar fashion to SNSPDs but take advantage of the distributed picture for longer nanowire meanders [17]. SNSPIs tend to be designed to have a slower propagation velocity and longer meanders. These effects cause a switching effect in the wire to take time to propagate to the two ends of the nanowire. By performing differential readout, we can spatially resolve the photon's incidence location as a function of the delay between the 2 device ports.

In this image, the SNSPI can be thought of as a non-linear transmission line that has the additional state nonlinearities described in section 1.1. By discretizing the transmission line into multiple lumped elements, the local non-linear contributions can be modeled by a non-linear inductor and resistor and a linear capacitor. In the

distributed picture image, a nanowire can be thought of as a long chain of discrete lumped nanowire elements in parallel with linear capacitor elements as shown in figure 1-2. This topology captures the distributed picture of pulses propagating in nanowire meanders and allows us to simulate SNSPIs.

1.2.3 Impedance Matching Tapers

Usually, the nanowire's impedance is not similar enough to that of the input and output circuitry. This mismatch causes the signal to reflect back into the wire instead of propagating into the next stage, causing interference and distortions. Impedance matching is done by designing tapers: extensions of the same line that increase in width slowly as shown in figure 1-3. This slow increase ensures that there is a minimal step change in impedance allowing for fewer overall reflections. A Klopfenstein taper is the optimal taper geometry when minimizing the total amount of reflections and the taper's length [18]. The slow change in width still causes internal reflections along the length of the wire, however, their overall magnitude is smaller than one step change as illustrated in figure 1-4.

Impedance matching tapers are often used on nanowire devices when we care about preserving the signal shape and magnitude. Thus, tapers allow us to reduce jitter in SNSPDs and preserve logic pulses in nanowire-based electronics [19, 20]. Impedance matching tapers can also be used to resolve the number of photons incident on an SNSPD [21]. Tapers are also used in SNSPIs, where the tapers preserve the fast-rising edge that is essential for spatially resolving photons [17].

One side effect of using a Klopfenstein taper is the change in propagation velocity caused by the change of electrical characteristics in the wire. Using $L(x)$ and $C(x)$ as

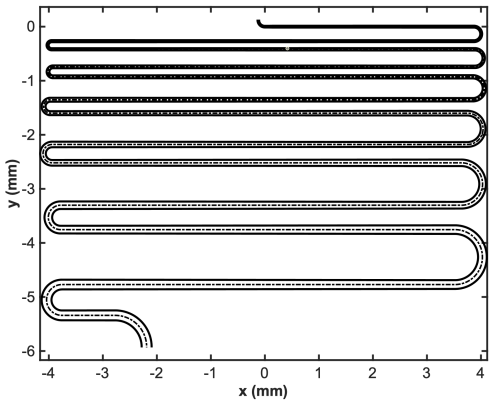


Figure 1-3: Top view of a Klopfenstein taper with a folded meander.

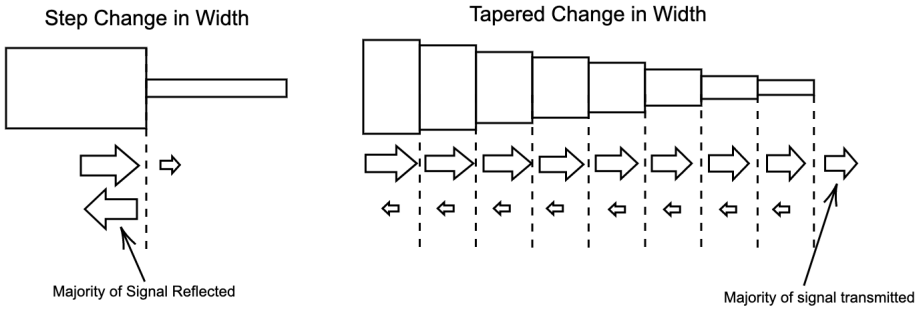


Figure 1-4: A tapered change in width causes smaller reflections at each boundary which add up to a lower total amount of reflection than a singular step change in width. Change in width is roughly proportional to change in impedance. The inclusion of multiple step changes in width increases the number of individual reflections occurring. The number of interferences a simulator would need to keep track of grows exponentially, increasing the complexity of the simulation.

the inductance and capacitance along the length of the wire, the propagation velocity is dependent on $L \cdot C$ and varies along x (since L and C do not scale inversely). Coupling this with the fact that the thinner wire will have a higher current density, this becomes a huge source of distortion.

Long tapers (and nanowires) are wound up in a boustrophedonic pattern of parallel straight wires connected by curved edges with large radii as shown in figure 1-3. This curvature is chosen in a manner that minimizes the total amount of reflection and current crowding: an unwanted effect caused by non-homogeneous distributions of current density through a conductor that changes the frequency response and impedance of the wire [22]. Impedance matching and the folding of the wire are both sources of distortions to signals travelling down the meander. As a result accurate simulation requires the ability to account for both the taper reshaping and the coupling of the folds.

1.2.4 hTron

A nanowire's accessible state space is limited by its current thermal state. Since the nanowire is also a thermal system that can generate heat, modeling its thermal behavior is important for accurate device characterization to account for effects such as latching [23].

While this coupling can be unwanted, devices, such as the heater-tron (hTron), take advantage of that coupling [24]. The hTron involves two superconducting stacks separated by an insulating layer. One of the stacks contains a resistor that generates heat while the other stack contains a nanowire above the resistor. When current flows through the resistor it generates heat that gets transmitted through the stack to the

nanowire. Through this thermal coupling, the nanowire can be thermally biased.

Heat transport through the stack can be modeled as interactions between electrons and phonons [24]. Each layer in the stack has electron and phonon systems that can interact with other layers at the boundaries. The electron systems can be heated by Joule dissipation. This model can account for heat generation, conduction and dissipation.

1.3 Problems Simulating Nanowires

A full-model that simulates all the dynamics of superconducting nanowires is hard to achieve due to the complexity, long simulation time and convergence issues that arise. Previous parts of this section demonstrated multiple regimes nanowires can be used in, from thermal coupling to a distributed microwave picture, each of which involves solving stiff non-linear differential equations in the time domain. Getting a model to simulate the electro-thermal coupling of a nanowire with respect to the stack, noise and photons as well as the distributed effect accounting for nonlinearities results in stiff non-linear equations. As a result of this complexity, work is usually done on the individual parts with experimental fits for each picture but no model incorporating how these different pictures interact.

Berggren et al. implemented a nanowire model in LTspice based on the phenomenological hotspot velocity model developed by Kerman et al. [3, 12]. This model is a lumped-element nanowire model that accounts for the hotspot dynamics using experimentally fitted parameters. It is implemented in LTspice and contains both non-linearities exhibited by a nanowire – discussed in more depth in section 3 [3]. This model only accounts for the small-signal solution and cannot be used in

noise, AC or DC analysis. The model also suffers from instability around the state non-linearity to be discussed in section 3.4.1. The instability and simulation modes are further discussed in section 3.4.1.

Since the speed of the taper is not constant and the inductance is non-linear, we expect input pulses to be reshaped non-trivially. Pulses traveling in a taper experience reshaping due to the linear-taper aspect, the change in impedance reflects certain components from the pulse while leaving other frequencies untouched. When on its own, this reshaping can be completely captured by the scattering parameters and linear transmission lines. However, non-linear transmission lines of constant width are also known to cause pulse reshaping, implying that scattering parameters on their own cannot fully describe a nanowire geometry [9]. Modeling tapers in LTspice tends to use a sequence of transmission lines (namely the lossy transmission line model with $R = 0$, discussed in 2.5.1) in series that have decreasing impedance. This model is sufficient in simulating the linear part of the reshaping under the assumption that the current flowing in the taper is much lower than the critical current, or that in the DC picture, this effect reaches equilibrium and causes a final shift in the perceived critical current of the device. In reality, this is not accurate, as pulses being carried down a biased line also experience non-linear reshaping, which cannot be accounted for by the linear transmission line segments.

Non-linear simulation for superconducting electronics has been widely studied in the frequency domain using techniques like Harmonic Balance [25]. These methods can account for the continuous non-linearity presented by kinetic inductance, but not the state transition non-linearity. These methods can generate scattering parameters that are dependent on frequency and magnitude to account for the non-linearity. JosephsonCircuits.jl for instance is designed to simulate Josephson Traveling Wave Parametric Amplifiers (JTWPA) topologies in the frequency domain using Harmonic

Balance [4]. WRspice and Xyce both have Harmonic Balance backends that are very efficient [5, 26]. However, for topologies that utilize the binary-state non-linearity, time-domain simulation is needed to characterize the device behavior. Given the nature of WRspice and Xyce, this constraint implies that the entire circuit must be simulated in the time-domain. This problem is addressed in section 4.3.

The non-linearity of the electrical model gets even harder to simulate when coupled to a thermal equation. As a result, the thermal coupling is usually linearized around the regime we care about. For example, for nanowires that have no need to thermally interact with other elements, the hotspot growth is simulated via the phenomenological hotspot velocity model [12, 27]. For geometries that rely on thermal coupling such as the hTron, the critical current of the device is presumed to be a function of the electron temperature [27]. These models are sufficient for some applications but lack the ability to simulate the thermal behavior of the device, neglecting the thermal coupling that might occur between two nanowires.

While most of these effects are hard to simulate on their own, the simulation of multiple nanowires is essential for scaling devices. As a result, it is important to develop a more stable scalable nanowire model, a dedicated efficient simulator, and a more standardized simulation environment that optimizes various nanowire topologies.

This thesis presents an integrated simulator environment designed with the goal of simulating superconducting nanowires. Superconducting device models have been designed to work in existing simulators but tend to favor the frequency domain and can only account for the electrical non-linearities exhibited by superconductors [4, 5]. Fast parallel circuit simulators with time and frequency domain capabilities exist, such as Xyce from Sandia National Laboratories, are not optimized for simulating

superconducting nanowire geometries [26]. The work presented in this thesis will be divided into 3 sections tackling:

1. an integrated environment for SPICE software that extends its modeling capabilities optimized for superconducting nanowire devices and accompanying experiments;

2. a simple procedure to measure the stability of circuit models used to present an improved nanowire SPICE model; and

3. finally, preliminary work done to develop an efficient Julia-based simulator optimized for superconducting nanowire devices and nonlinear microwave circuits.

Chapter 2

spice-daemon – a Python wrapper for SPICE solvers

This chapter introduces qnn-spice and spice-daemon, python libraries that improve traditional SPICE environments (such as LTspice) tailored for the design and simulation of superconducting nanowire devices. qnn-spice is a toolkit that helps synchronize and version track all the models produced by MIT's Quantum Nanostructures and Nanofabrication group using the traditional git workflow. spice-daemon is a python package that adds new functionality such as hyperparametrized models, noise and post-processing to LTspice.

2.1 SPICE

One popular way of simulating electronics is using SPICE (Simulation Program with Integrated Circuit Emphasis). SPICE solvers are an industry standard method of

simulation that combines DC analysis (also known as operating point analysis), AC analysis (linear small-signal frequency domain analysis), and transient analysis (time-domain analysis for non-linear differential algebraic equations) among other analysis methods [28].

Since then, the original Berkley SPICE inspired multiple other SPICE solvers including LTspice, a popular free circuit simulator [29]. SPICE models for superconducting electronics exist including models for the nanowire, hTrons and Josephson Junctions implemented as SPICE netlists. For nanowires, we particularly care about transient analysis where the solver is based on a piece-wise finite method using dynamic timestepping fitted to a low-order polynomial [10].

2.1.1 Interfacing with LTspice

Interfacing with SPICE software involves generating a netlist — a code snippet that defines how the different circuit elements are connected to each other. Netlists have a '.net' (and sometimes a '.cir') extension and can be used across different SPICE implementations. Netlists are encoded as ASCII files and as such editing them is straightforward.

Some commercial versions of SPICE software, including LTspice, add Schematic Capture capability. Schematic Capture allows for a native GUI encoding of a circuit to be converted into a netlist (in LTspice, that is a schematic file with the extension '.asc').

LTspice generates multiple types of files after each successful run. The most common type is a compressed binary '.raw' file that is generated after AC and transient simulations. An optional '.op.raw' file is also generated that saves the DC solution

and can be imported to skip DC simulation. LTspice also generated a '.plt' file that encodes the layout of and variables plotted in the LTspice plotting window. The python package PyLTSpice has built-in methods to read and write RAW files [30]. RAW files are the only form of input that can be used by the LTspice waveform viewer.

A '.log' file is always generated, regardless of the type of simulation and/or its success. For a smoother experience using LTspice on Mac with spice-daemon, it is recommended you uncheck "automatically delete .raw and .log files" under the operation sub-menu in the LTspice preferences. This setting is by default checked on Mac (but not on Windows) and deletes files that spice-daemon uses to track LTspice simulations (discussed in section 2.4).

SPICE directives refer to text that is passed from the Schematic Capture circuit directly to the netlist. For example, the .tran directive sets up a transient simulation with a specified stop time (and other optional parameters). If an undefined directive is specified, for example, misspelling .tarn instead of .tran, it is passed to the netlist normally, and upon running a simulation, a warning is raised with no effect on the simulation output.

2.1.2 Models

Regular SPICE models are composed of two main types of files, symbol ('.asy') and library ('.lib') files. A symbol file defines the visual metaphor used by the schematic capture part of LTspice to visualize the element and its ports. The library file contains the subcircuit definitions for models. A subcircuit defines how ports connect to each other using other components or subcircuits. A library file can contain the subcircuits

and they can each be referenced individually by a separate symbol file.

LTspice allows these files to exist in two locations by default. The Model Library folder and the circuit directory. In other words, whenever the LTspice schematic needs to reference a symbol or library file, unless a path explicitly references a full path, LTspice checks the Model Library folders and the parent folder for the schematic. This makes it hard to continuously develop models in a repository while still being able to use the most up to date version. This issue is addressed in section 2.3.

2.2 Installing spice-daemon and qnn-spice

Installing qnn-spice and spice-daemon can be done by cloning the repositories and symbolically linking the update bash script and spice.py files, from each repository respectively, to your local /bin directory. For instance using the ln -s command to link $PATH/qnn-spice/update.sh to spice-update and $PATH/spice-daemon/spice.py to sd, you can invoke the two scripts from using the two commands spice-update and sd from the terminal at any time. spice-daemon can also be installed as a Python package with pip to allow for more flexible control through python scripts – this is ideal for more optimized workflows, large parameter sweeps, niche post-processing types or experiment-simulation hybrid setups.

2.3 qnn-spice

In a collaborative setup where SPICE models might be edited (either continuously or with infrequent small fixes), having the ability to track the version of the models

is important. One solution is to include a version string that the editor updates between revisions. Doing so however, does not handle merge conflicts natively and does not track file differences – "diffs". From these requirements, the widely used version tracking software git can be used to track diffs and users will always have to re-download the latest version of the model.

One way to manage models in LTspice is to download them individually and place them in the library folder that contains all the base models. However, this can be tedious as it requires repeating the process for each model and your models can't be version tracked easily. An alternative approach is to store all the models in the same directory as the circuits and manually download each model as needed. This has the advantage of forcing the models to be version tracked in your repository, but it can result in a cluttered directory and multiple copies of each model on the system. Both methods don't guarantee that you are using the latest version of a model.

This is where qnn-spice comes in, MIT's Quantum Nanostructures and Nanofabrication group (QNN) has multiple repositories, each with multiple spice models and different access rights. By having a single repository track every repository containing SPICE models, a single repository could track all the changes across every model produced by the QNN group and still respect each user's access rights. This single repository method takes advantage of git submodules, which track the head of each sub-repository. A helper update script pulls every submodule and creates symbolic links into LTspice's library folder to each model. The model library and symbol files to be included are specified in a markdown file at the base of each repository. The main respository tracks the remotes (location and branch) of each repository using git's built-in submodule branch tracker.

The use of symbolic links means LTspice is agnostic to where the model was edited

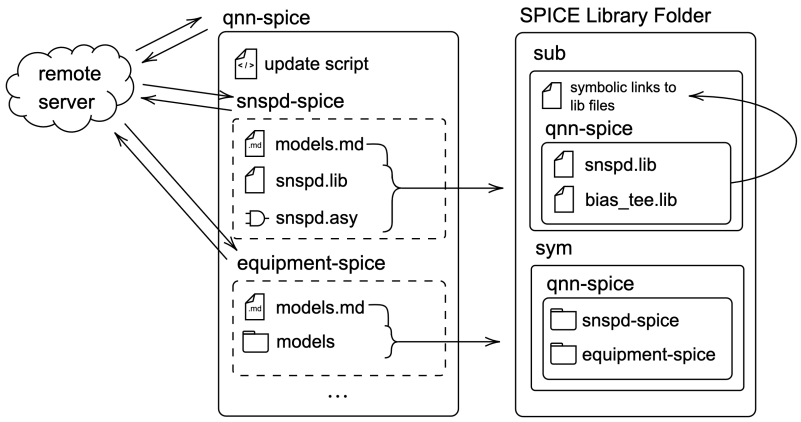


Figure 2-1: A notional diagram of the qnn-spice repository and linking to the LTspice Library folder. The qnn-spice repository tracks the heads (not contents) of multiple git submodules, other repositories version tracked using a remote server. The qnn-spice update script creates symbolic links from the local repository to the Library folder. The folder structure of the submodules (and inner-folders as defined in each submodule's 'models.md' markdown file) is preserved when moving symbol files to the sym directory. The folder structure is not necessarily preserved in the library file, it is then flattened by adding another layer of symbolic links in the sub folder's root to each '.lib' file.

from (locally from the home directory, locally from the model directory or from the remote branch). When the update script pulls the main and sub repositories, the previous symbolic links are deleted and new ones are made. The sub-repository structure is copied into two qnn-spice folders are created in the sub/ and sym/ subfolders of LTspice's library folder. Another set of symbolic links is created in the sub/ directory for each file in sub/qnn-spice. This makes sure that symbol instances find the referenced subcircuit when they don't reference the parent qnn-spice folder.

The markdown file consists of a list that maps the path of each file to include in the repositories to a destination path in the two qnn-spice subfolders based on their extension. The destination path allows users to change the grouping of elements agnostic to how the repositories were laid out, i.e. you can have two repositories have their elements grouped together and you can split one repo into multiple folders.

LTspice updates the underlying library subcircuits for models upon every simulation, and as such you don't need to restart LTspice for this to take effect. However, it is necessary to restart LTspice to reflect changes in the symbol files. This is also applicable to spice-daemon dynamic models in section 2.4 and 2.5.

qnn-spice is also capable of merging libs. This means that all the subcircuits can live on the remote at all times and running a simulation involves redownloading all the subcircuits and caching them. This can be done using the .inc (include) directive that takes in the URL for the remote lib. The include directive allows for a built-in way of accessing local and remote files - it can be used to import remote library files. Every simulation run forces them to pull (and cache) the latest subcircuits for a model without having to manage this outside of LTspice. This can be used in qnn-spice by invoking the library compiler python script that compiles a complete library of all subcircuits that are part of qnn-spice. Unfortunately, this doesn't work on the Mac's

version of LTspice 17.0 (but does on Windows and Linux). This feature is useful for continually changing library files in a rapid development environment and for syncing models onto new devices without any overhead.

2.4 spice-daemon assisted LTspice simulations

The main input for spice-daemon is a YAML – a human-readable data- serialization language – file that defines simulation parameters, spice-daemon models, and toolkits. A YAML file can also be version tracked, allowing all parameterizations to be known by the host python script.

spice-daemon creates a Simulation object for each daemon instance. Each simulation object has a list of files, watch_files, that are used to indicate the versioning of a simulation. spice-daemon defines a WatchDog object whose job is to make sure spice-daemon modules and toolkits are up to date if any of these watch_files are edited. The WatchDog module periodically checks for edits on each of the watch_files and starts a new thread to regenerate some (or all) of the spice-daemon-produced files. For instance, if someone edits an attribute for a component in the YAML specification file, the component library file needs to be regenerated to reflect the change in the attribute (but not the symbol file and, depending on the module, the PWL file to be discussed in section 2.5.2).

Every Simulation object defines a couple of important File objects that are always present regardless of the user's setup for LTspice. Files are an extension of the Python Standard Library's Path object that can additionally:

1. track edit timelines,

2. detect LTspice-native file encodings,

3. generate dictionaries from YAML files, and

4. read/write to files.

spice-daemon initializes circuits by placing a block that imports a textfile (trancmd.txt) generated in the spice-daemon data directory. This textfile overwrites SPICE operational variables (such as reltol), defines new spice-daemon instantiated parameters and defines the similation time and step size. This block involves adding text to a schematic and involves heavy encoding checking when importing the schematic. A failure to encode the data correctly could result in corrupting the schematic.

spice-daemon's WatchDog can be called from the terminal or from a Python script. The terminal bash script suffices for basic usage of spice-daemon intended for non-experimental environments. When you call spiced from the terminal, spice-daemon launches the WatchDog that checks for periodic changes in files, runs the module and toolkit initialization and begins the post-processing logic accordingly. It also performs a check on the spice schematic (read using PyLTSpice) to see if it has been initialized by spice-daemon, if not, it writes a new instantiation block. spice-daemon needs to access simulation parameters - such as simulation time, steps, etc. - before LTspice starts solving the circuit. This is through spice-daemon's parameter acquisition and injection features.

spice-daemon's main feature is hyperparametrization. It allows for external control over LTspice variables and circuit topology from a python environment. The topology based features are discussed in section 2.5. spice-daemon allows for running large sweeps of data efficiently and allows for producing large datasets on parameter sweeps using Python's numpy and pandas packages. This is mostly intended to be

done using the python spice-daemon interface where you can automate both topology, macroscopic parameters and individual element parameters. This is an important feature for models that change the underlying differential equation coupling such as in nanowires.

While spice-daemon was designed for and tested on LTspice, it should work on any SPICE based simulator and many other non-SPICE simulators. spice-daemon has been used in tandem with Xyce and JosephsonCircuits.jl. For Xyce, and any SPICE-like simulator that generates files by default, modifying the watch_files allows you to operate spice-daemon normally. For packages such as JosephsonCircuits.jl, you can trigger spice-daemon to run using the python interface provided. This allows spice-daemon to be an abstract wrapper that can help precompute and layout large networks for any arbitrary circuit simulator.

In summary, LTspice generates multiple files during every run (discussed in section 2.1.1). spice-daemon is launched in parallel with your circuit and tracks the edit history of the log file, the YAML specification file, and the circuit schematic using a WatchDog object. Based on the edit history, spice-daemon can infer what files have to be regenerated.

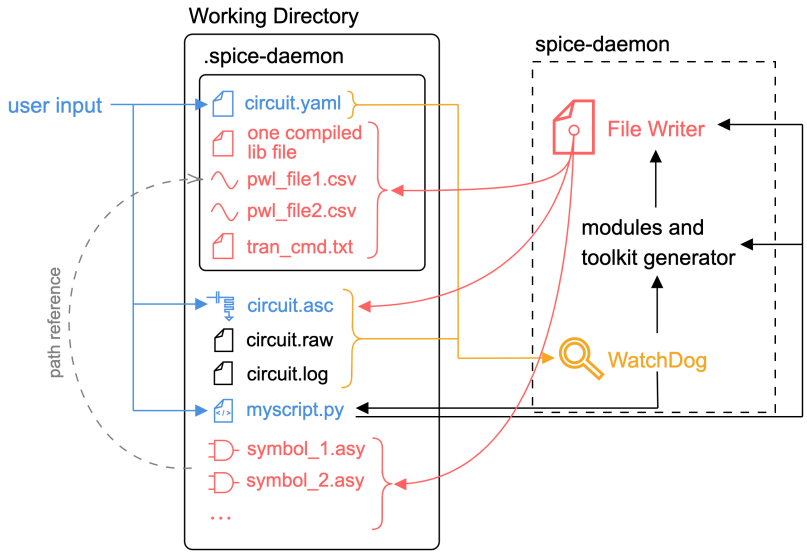


Figure 2-2: Notional diagram showcasing the interaction between different parts of the spice-daemon library and a typical SPICE workflow. Blue files are edited by the user, this includes: the SPICE circuit, a YAML file that defines all the spice-daemon-related parameters, and any scripts that use the spice-daemon python API. Pink files are generated by spice-daemon, this includes: library files, tran_cmd.txt, PWL files, symbol files, and post-processing plot exports. The WatchDog module watches for changes in the orange files, including LTspice generated files and user-input enabled files, and triggers module and toolkit regeneration.

2.5 Dynamic Models

LTspice components are able to be parameterized using a constant global parameter space that can be used when math expressions are being evaluated (such as the output voltage of a behavioral source or the inductance of an inductor). spice-daemon adds the ability to hyperparameterize components beyond expressions by granting the ability to create a PWL file and modify the netlist (circuit topology) of the model between runs.

2.5.1 Lumped Element Transmission Lines and Tapers

One type of dynamic model that is incorporated into spice-daemon is lumped element transmission lines. Instead of using LTspice's built-in transmission line models (either the Lossless Transmission Lines (T elements) or the Lossy Transmission Lines (O elements)), spice-daemon allows you to specify a variable discretization length lumped-element version.

The Lossless Transmission Line model has a bunch of limitations: it models only one propagation mode, does not support non-linear response functions, and does not model the DC behavior correctly. The Lossy Transmission Line also suffers from multiple caveats: it does not support frequency dependence for loss and it also does not support non-linear response functions [31]. This has been recognized by members of the LTspice modeling community and a separate frequency based modeling method was developed for PSpice and LTspice based on the telegrapher's method [32]. Unfortunately, the LTspice Laplace method in transient analysis is highly unstable and this causes simulating more than 1 transmission line impractical. For repeating geometries where accuracy is important, simulating the transmission lines as a repeating

sequence of lumped elements will guarantee the best convergence and stability.

Another drawback of built-in transmission lines is that SPICE programs will have more trouble converging on a correct solution. The inclusion of a transmission line introduces new breakpoints at the beginning of the simulation since no timestep during the simulation should be more than the delay of a transmission line. Including multiple transmission lines of different delays makes the situation worse and the number of breakpoints added becomes impractical to simulate (it grows with the greatest common multiple of all transmission line delays) [10]. As a result, you shouldn't use transmission line concatenation to model tapers as the timestepping algorithm will take impractically long to simulate.

For well-defined convergence with non-convolution based models, we need to be able to simulate repeating lumped element models from within LTspice. However, this would involve laying down thousands of repeating chunks of elements manually. One use of dynamic models is generating a model that encodes variable length logic. In this method of programming a lumped element transmission line, the circuit topology can be parametrized by a single parameter in the configuration file (in this case number of nodes). This type of automation is not possible using LTspice's built-in parameterization as the SPICE parameter resolver is queried after topology checks.

This method of simulating a transmission line not only solves the issues introduced by the T and O models, but also gives us the ability to simulate more complicated transmission lines. For instance, inductors on a transmission line can be non-linear, making simulating a superconducting transmission line more accurate. Other possibilities that aren't possible in the LTspice environment include adding custom elements instead of a repeating sequence of inductors and capacitors allowing us to model JTWPAs of variable length easily.

Another extension to this one-to-many mapping for the transmission line can be extended to model lumped-element tapers. The transmission line models in LTspice work for lines with constant parameters (impedance, propagation velocity, loss, etc.). With a lumped element model that is fully controlled by spice-daemon, changing the impedance of one port can map the inductance and capacitance of each finite element to a pair of values based on the taper geometry chosen. This adds another layer of abstraction where we can define an impedance-matched transmission line with a Klopfenstein geometry between two impedances Z_{in}, Z_{out}. If we change Z_{in}, the spice-daemon instance calculates new tapering parameters smoothly perturbing the impedance of the line from Z_{in} to Z_{out} and updates the library file for the taper element. When LTspice runs a new simulation, it pulls the latest lib file with the new impedance-matched taper. This non-uniform version of the transmission line is included as a separate taper model in spice-daemon that has additional logic pertaining to impedance-matching geometries.

Figure 2-3 showcases the use of a spice-daemon generated taper element to perform Photon Number Resolution on a nanowire strip. Figure 2-3(a) showcases 4 nanowire elements, mimicking a single nanowire element recieving up to 4 separate photons. The nanowire elements are connected to an impedance matching klopfesntein taper of 1000 elements generated by spice-daemon. A noise source as described in section 2.5.2 is connected in parallel to the nanowires. Figure 2-3(b) shows the simulation output for photon number resolution using tapers with and without noise.

2.5.2 Generating Noise

The ability to add various types of noise when designing superconducting nanowire devices is necessary in order to achieve realistic operation margins and understand

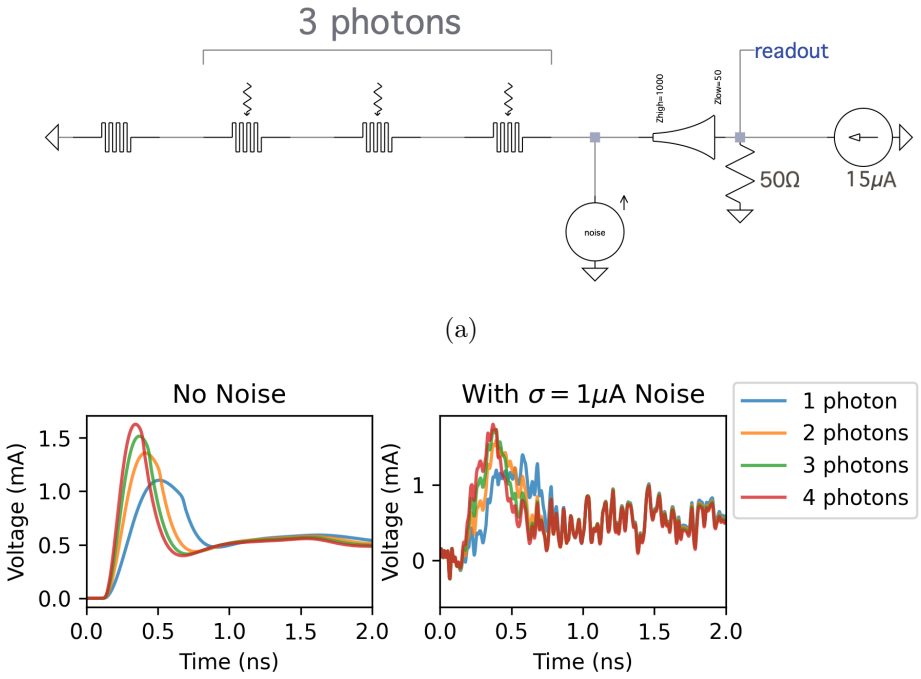


Figure 2-3: A spice-daemon assisted SPICE simulation of photon number resolution on an SNSPD. (a) Illustration of 4 nanowires elements in series that are simulating one effective nanowire with 16 possibilities of a photon incidence. These elements have no delay between them, making them a simpler model that doesn't account for microwave properties of a nanowire. The nanowire elements are connected to a 1 kΩ to 50 Ω taper biased by 15 µA. A gaussian noisy current source is added after the taper to roughly simulate noise a nanowire might experience. The taper and noise source are dynamic models generated using spice-daemon. (b) Measurement at the circuit readout showcases our ability to resolve the number of photons incident on the wire using a taper. When the system experiences noise, the margin between photon counts more than 2 gets slimmer.

the sources of noise in the device. There are multiple types of noise distributions that can couple to nanowires from various sources: gaussian distributions as Johnson noise, Poisson noise as random photon events and $1/f$ noise due to device-specific microscopic degrees of freedom [33]. The simulation, fabrication and testing process of a device when accounting for noise on each level can inform each step in the process, making noise simulation essential. The noise model used in a simulation informs the geometry to be used in fab and through testing you can identify pitfalls of the noise model used in simulation.

Since we care about the non-linear transition between the superconducting and normal state, the addition of noise severely limits the operational margin and maximum performance of our device. In the example of using an SNSPD, optimal operation of the device requires it to be biased at $i_c - \varepsilon$, where ε is a factor that correlates the magnitude of noise and the rate of dark counts (false switching events). For instance, assume the current source is noisy and generates gaussian noise centered around 0 with a standard deviation (magnitude) of σ. If we bias using $\varepsilon = 2\sigma$ that means 2.28% of all detected counts are dark counts caused by this noise distribution. Alongside the noise constraint, the signal being measured must always be greater than ε to cause a detection event.

The generation of uncorrelated noise is important for device characterization and noise source investigation. While there are methods for generating uncorrelated noise for a simulator, LTspice uses one global random number generator causing distributions to be inherently correlated.

In transient analysis, we can use a behavioral voltage source with LTspice native math commands to generate noise such as random and white. However, LTspice noise commands generate noise that is correlated amongst instances and is not gaussian in

nature. One workaround that was adopted by the LTspice community involves the Central Limit Theorem [34]. By using 4 voltage sources each producing a shifted seed for gaussian noise (guaranteeing that the seed does not overlap with the simulation time), the noise sources are uncorrelated. By adding the outputs of the behavioral voltage sources, the noise distribution approaches that of a gaussian distribution via the Central Limit Theorem. Note that the number 4 was picked due to a trade-off between the complexity of generating and simulating that noise and how gaussian it is – the more sources there are, the more gaussian the noise distribution is.

Another workaround is to use PWL files. PWL files allow you to input piece-wise linear functions into LTspice sources that are not necessarily behavioral. The PWL operation mode maps (time, value) pairs to a continuous output value based on the simulation time. However, if a simulation has N timesteps, N noise points need to be generated to prevent extrapolation. However, running the simulation with a different number of points can change the number of timesteps taken and therefore this requires verifying that the numbers are in agreement. The process of re-creating this noise file, ensuring there are enough data points as timesteps and that the noise data follows a certain distribution is tedious.

To solve this issue, spice-daemon can handle the creation of noise sources and their accompanying PWL files, abstracting them behind a symbol file. The user defines a noise source type (voltage or current), the noise distribution it should follow (Poisson, Gaussian, $1/f$, etc.) and distribution parameters (mean, standard deviation, etc.). The spice-daemon instance then generates a symbol file for a noise source that references a sub-circuit for each noise instance. Each sub-circuit references a separate PWL file that encodes a list of (time, value) pairs generated in Python using NumPy. As a result, we know that the noise inputs to LTspice actually follow a specific distribution, and we can verify that the correct noise distribution is being simulated inside

LTspice.

This method allows us to easily have multiple non-correlated noise sources each with an arbitrary noise distribution. Each source has a separate symbol and the noise distributions are recalculated after each simulation under the invocation of the WatchDog class. A spice-daemon-generated current noise source is included in the photon number resolution example showcased in figure 2-3.

2.5.3 Creating a new dynamic model – an SNSPI

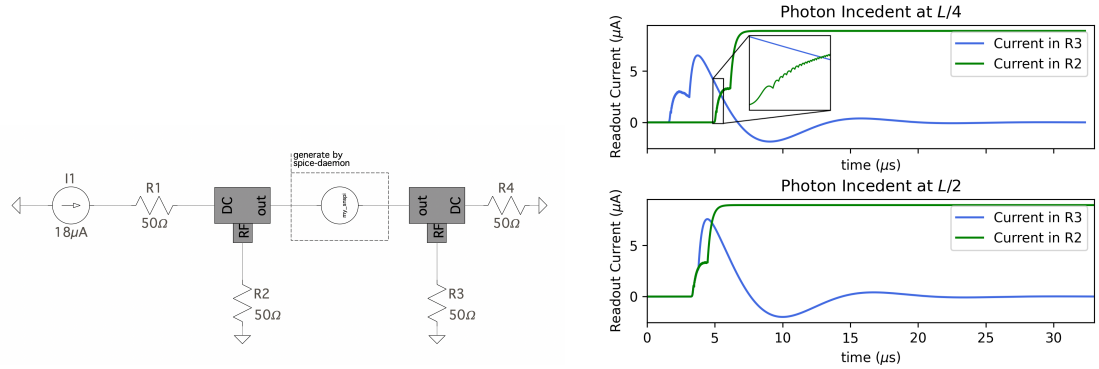
This section will walk through making an SNSPI dynamic model to highlight the benefits of hyperparametrization and serve as a tutorial for extending the spice-daemon dynamic model library. An SNSPI as discussed in section 1.2.2 can be thought of as a non-linear transmission line that exhibits the same switching properties of a nanowire locally. As a result, an SNSPI can be modeled in the transmission line lumped element picture (advantages of lumped over built-in O- and T-models discussed in 2.5.1) using nanowires instead of inductors as shown in figure 1-2. In the superconducting regime, they act as a non-linear inductor and can locally become resistive upon a photon event.

The spice-daemon module template has 3 base functions defined: (1) update_PWL_file, (2) lib_code and (3) generate_asy_content. Function (1) is used to generate and write PWL file data that is synced to the simulation timesteps. Function (2) generates the SPICE library code that controls the electrical behavior of your model. Function (3) generates the symbol file that is used to draw the model's metaphor. This section will go through modifying the second function and leaves (1) and (3) unchanged from the template.

To develop a module, first define the hyperparameters involved to design your module. The five degrees of freedom the distributed SNSPI model will simulate will be: (1) the critical current i_c, (2) inductance per unit L, (3)capacitance per unit C, (4) the number of discretization num_units, (5) the incident photon locations and times (a one-to-many map from incidence location to incidence times). This information is input through the YAML file.

We can then append a new SPICE netlist line for each nanowire and capacitor to generate the meander, making sure the first and last elements connect to the PINS defined in the class. Since we are using the nanowire model, we need to include the lib file containing the nanowire definition (synced using qnn-spice) using the .lib directive. This is handled automatically (added for every instance) if you are using the Element.* class to add the components (with duplicate declarations removed at compilation). Note the Berggren et al. nanowire model is a 4 terminal device, where 2 ports are the photon inlet/outlet and the other 2 are the electrical contacts for the nanowire [3].

Due to hyperparametrization, we have information about the circuit that we can use to simplify the simulation. For instance, we can replace the entire nanowire model with one inductor for each "chunk" that won't switch. Since spice-daemon knows the photon locations, we could simplify the netlist. For an SNSPI with num_units $= 1000$, we could make one in every 50 nanowire elements an actual nanowire, while the rest are only the inductor portion of the model. This means that only about 5% of the inductors have the additional hotspot integrator and hotspot resistor. A result of section 3 is our model is more stable and converges faster (we have 3800 less highly non-linear behavioral sources and switches from figure 3-6 and the dependency graph collapses to only one node in figure 3-7).



(a) (b)

Figure 2-4: Example of a spice-daemon generated model for an SNSPI. (a) Setup for measuring an SNSPI using differential readout on the RF ports of 2 bias-tees. The circular element is the default symbol for spice-daemon dynamic model, representing the SNSPI model. (b) Simulation output for a photon incident at $x = L/4$ and $x = L/2$ for an SNSPI of length L and 200 discrete elements. In both plots, we see that the SNSPI latches, and the SNSPI carries a voltage across it. When the pulses form at $L/2$, they take the same amount of time to reach either end of the wire. When the photon is incident at $L/4$, the generated pulse closer $x = 0$ reaches the RF readout port first.

2.6 Arbitrary modeling using scattering parameters

Modeling experimental equipment, such as a coax cable or bias-tee, involves making a lumped approximation of the device and using them in your circuit. This type of modeling is sufficient for capturing the qualitative behavior of a device, but often is not identical to the response of the equipment in the lab. One solution to this is simulation using the direct scattering parameters - which is done in programs like Cadence PSpice, but not LTspice [11].

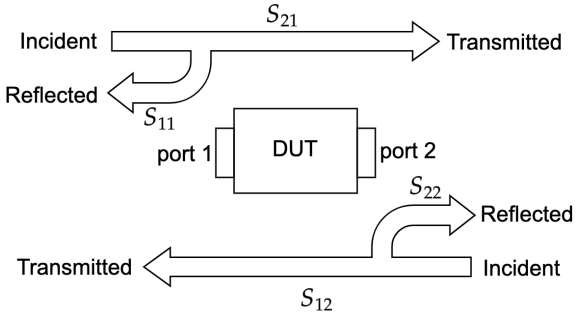


Figure 2-5: Illustration of scattering parameters showcasing the transmitted (S_{21} and S_{12}) and reflected (S_{11} and S_{22}) signals of a DUT.

The scattering parameters (S-parameters) S_{xy} of a 2-port device describes the change in magnitude and phase a signal seen at x relative to the signal input into port y. This notion can be generalize to multiple ports by taking successive measurements and termination the third port using a matched load. The S-parameters are frequency dependent and can be described as an $n \times n$ matrix for a n-port device. For passive devices, There exists a 1-to-1 mapping from S-parameters to Z- and Y-parameters (impedance parameters and admittance parameters respectively).

If we restrict the type of devices we are working with to passive devices, then we can perform this mapping using voltage-dependent sources in LTspice. These sources will use the built-in frequency dependence of b-sources to define the frequency

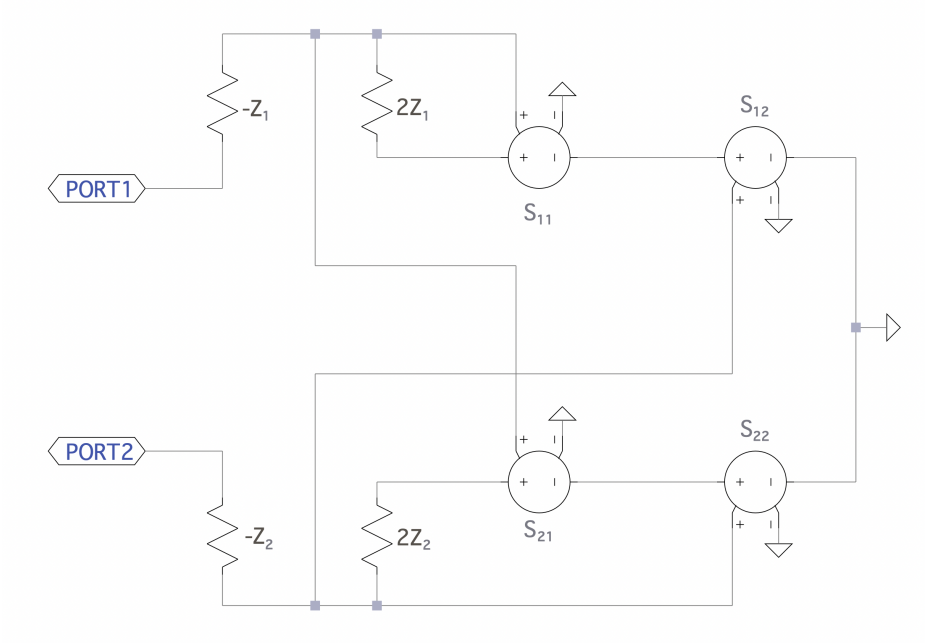


Figure 2-6: Subcircuit for a 2-port model generated by S-parameters.

behavior of the device [35]. An accompanying spice-daemon module (spice-arbnport) can generate a model with this topology automatically given the scattering parameters measured for a device.

2.6.1 2-port model

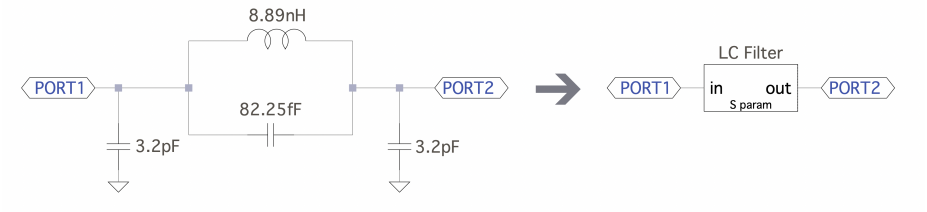
We can model devices with arbitrary frequency-dependent scattering parameters using the topology presented in figure 2-6 [35]. Port 1 and 2 see an impedance of Z_1 and Z_2 respectively. Taking $Z_1 = Z_2 = 50\Omega$ and working with one frequency, each dependent voltage source applies 1 dimension of the S parameters onto the signal. While figure 2-6 uses E-sources, the actual source uses B-sources (a note on why later in this section). For instance, if the device is completely transmissive ($|S_{12}| = |S_{21}| = 1$ and $S_{11} = S_{22} = 0$) then the sources S_{21} and S_{12} are shorts and an input V at port 1

corresponds to an output of magnitude $|S_{11}| \cdot \left| \frac{V}{2Z_1 - Z_1} \cdot (-Z_1) \right| = |V|$, i.e. all the power was reflected back. The outputs due to reflection and transmission for a given port are independent and as a result (the voltage for each S_{xy} parameter is added up at port y and the input is at port x), you have 8 degrees of freedom in this network (2 per E-source).

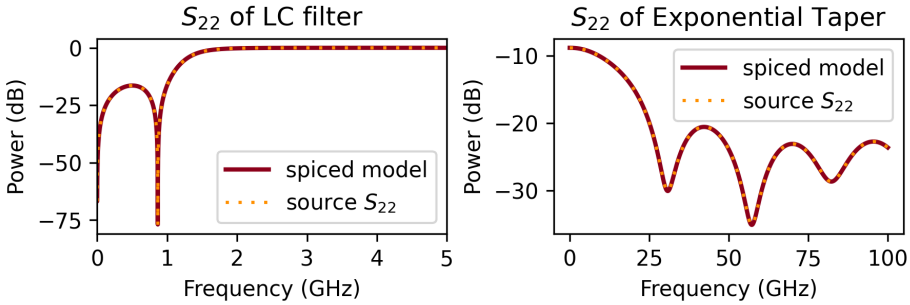
Like other dynamic models, spice-daemon generates the netlist for S-parameter-based devices and adds the S-parameter CSV source to Simulation.watch_list. By specifying the model name and directory for the CSV in the YAML file, spice-daemon will handle model generation. Each S-parameter results in generating one b-source that defines a PWL using a list of 3-tuples. The tuples are encoded as (frequency, magnitude [in dB], phase [in degrees]) by default (this can be changed at the expense of slower compilation).

One limitation corresponds to the number of data points, as it increases, the SPICE gets hung up on generating logic lines. LTspice doesn't document the existence or process of creating logic lines, however, reverse engineering the binary indicates that logic lines are related to the process of generating an internal SPICE netlist. This netlist is generated before any topology checks (and subsequent DC, tran, AC analysis). The logic line assembler runs on every line separately, and as a result, overloading all the frequency PWL information on one line speeds up the logic line generation. For N 3-tuples, this takes the processing time down from $O(N) \rightarrow O(1)$ for each source. In practice, for 1.2 million 3-tuples, the logic line creation time goes down from above 30 seconds to below 0.1 seconds.

The complexity of simulation in the time and frequency domain for LTspice is minimal – the freq interpolation backend is similar to that used by the table directive and PWL source mode. If the logic line creation time is avoided, compilation time is



(a)



(b)

Figure 2-7: (a) Example of an LC subcircuit with 4 elements being encoded into a 1-element S-parameter based 2-port device using spice-daemon's dynamic models. (b) S_{22} of Analytical vs Scattering Parameter Based Model for an LC filter (left) and an exponential taper (right). The solid lines are the S-parameters computed from an AC analysis in LTspice using the spice-daemon generated 2-port model. The dotted line is the source parameters used by spice-daemon to create the model generated using scikit-rf.

also small, making this method much more practical for simulating large networks. Using S-parameters, you can simulate an arbitrary number of ports using a constant-sized network - at the expense of interpolation time and errors.

Figure 2-7(b) compares the source S-parameters used to generate the dynamic model to the resultant S-parameters. We plot the reflection magnitude computed by the scikit-rf package for an LC filter (figure 2-7(a)) and an exponential taper ($70.8\Omega \rightarrow 33.1\Omega$). We then make a spice-daemon model using these S-parameters (dotted) and plot the calculated S-parameters from the AC analysis. We see a strong

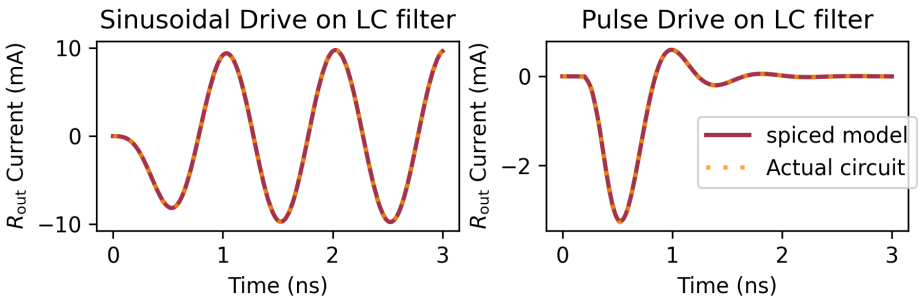


Figure 2-8: Time-domain simulation on the LC filter from figure 2-7(a) comparing the actual circuit to a spice-daemon generated dynamic model from only the S-parameters. The S-parameter based model is in agreement with the actual filter's solution for both a sinusoid and a pulse input.

agreement, indicating that the model functions correctly in simulating the original device. The model is much faster for a linear taper in both AC analysis and transient analysis.

We also show in figure 2-8 the agreement in transient analysis by simulating the spice-daemon model vs. the actual LC filter's netlist in LTspice. We test the response of the LC filter and the spice-daemon generated model against various sine waves frequency (1GHz shown) and various width pulses (0.1ns shown). We see a strong agreement in the time domain (the error is bounded below 0.5% for the above figure).

The ability to simulate S-parameters directly in a simulation environment is important for many common topologies used for SNSPD readout. When connecting to a setup, you have to model every element with a separate model making sure the values in the experiment and simulation agree. With the ability to use S-parameters directly, you can measure the response of the entire setup (2 measurements for a 2-port DUT regardless of the number of elements). This provides a model that is an accurate reflection of the experiment being run (even if the experimental setup has a mistake). This verification can confirm results that occur even on incorrect setups

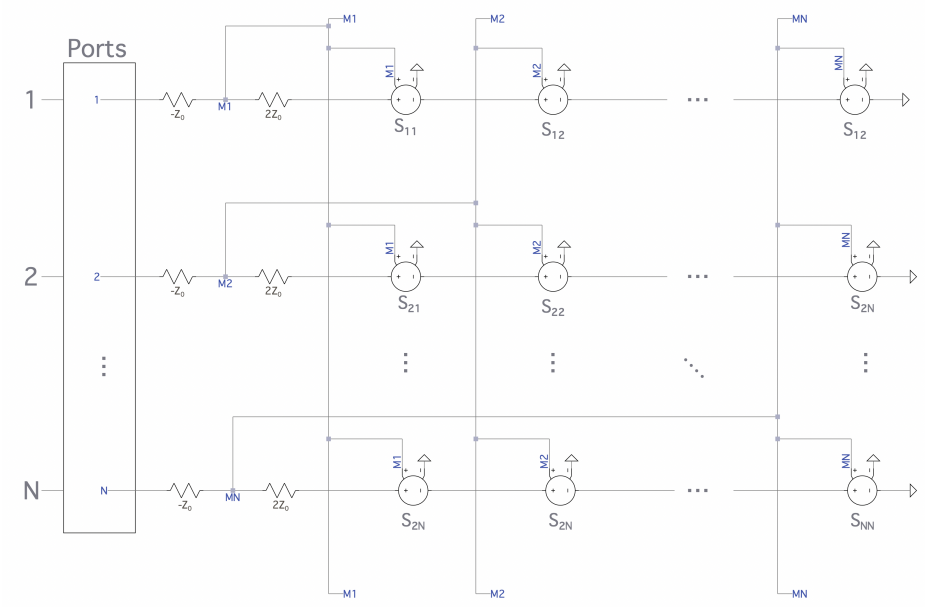


Figure 2-9: Subcircuit for an n-port model to be generated from S-parameters.

since we know the exact setup response and can simulate the device under test with that exact response.

This can also imply more true-to-experimental setup models, even if the measurement is not repeated. For instance, characterizing a coax line of a certain length gives an accurate reflection of the coax line in simulation. Another example is using characterizing filters and running a sweep on them to decide which filter is the best for a certain application.

2.6.2 n-port model

It is also of interest to simulate n-port devices, for example, a bias-tee. The model can be scaled to work for n-ports as shown in figure 2-9. This scales as n^2 with the

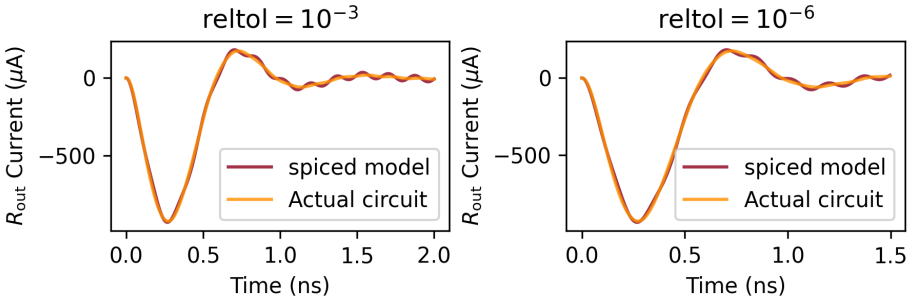


Figure 2-10: Oscillating instability that can be experienced in a transient simulation by a S-parameter-based dynamic model generated by LTspice.

number of ports n. The actual measurement for parameters S_{xy} can be performed by terminating all ports other than x and y with a matched termination. At least $\frac{n(n-1)}{2}$ sweeps are needed to fully characterize a device [36].

As the number of ports increases, the number of sources only increases linearly unlike using a circuit model. The expression at node M1 (in figure 2-9) is composed of ordering-independent behavioural sources that can be merged to one source (the order of the individual dependent voltage sources doesn't matter and therefore the sources can all be merged into one dependent source). The addition of one port results in adding one more behavioural voltage source at node M(N+1). This unfortunately is also coupled with a higher inaccuracy in transient simulations of varying timesteps. Since these inaccuracies are hard to predict, one option to extend S-parameter based multi-port devices is to unhook port dependencies that don't matter. For instance, in a 3-port device where port 0 either reflects the entire signal or transmits it to port 1, we don't need to include the source on branch M2 that is a function of port 0.

2.6.3 Inaccuracy of S-parameter Modeling in Transient Analysis

Although the generated models accurately replicate the behavior of the devices in AC simulation, they do not consistently perform well in transient analysis. Simulating S-parameters in a time-domain simulation raises time-complexity and accuracy issues due to frequency- and time-domain mixing [37]. The spice-daemon source can be modified to realize a corresponding minimal order state-space circuit from a fitted version of the transfer function [38]. This method generates linear sub-networks of lumped and non-linear devices that describes a reduced-order macromodel of an n-port device that are more stable.

Figure 2-10 showcases incorrect oscillations that are caused by using the spice-daemon model in transient analysis. Note that the transmittance (S_{12}) for both models is identical, however, this is a result of time/frequency mixing. These oscillations are present in the actual model but are more attenuated (a factor of 5). This effect is seen when the input sinusoid has a frequency of 5.88GHz, which corresponds to a sharp pole of the filter's transmittance.

2.7 Post-processing using spice-daemon Toolkits

While LTspice transient simulations are sufficient for characterizing the time behavior of a superconducting circuit, an optimized simulation environment should have the ability to post-process the data in a meaningful way, producing plots that are familiar to measurements taken in a lab setting. SPICE transient simulations are optimized to provide node voltages as a function of time. However, there may be cases where

we are interested in other bases such as the bias current, temperature, frequency bins, or other variables where time is not the independent variable. A widely used measurement for SNSPDs is a PCR (Photon Count Rate) curve, where the y-axis is the count rate (how many hotspots form on the SNSPD) and the x-axis is the bias current. The ability to replicate that in a simulation environment is an essential step for design verification and iteration.

Creating a PCR curve requires creating a plot where the x-axis is the bias current used for multiple periods of simulation time. The y-axis would contain the sum of spikes that the SNSPD has over each period of time. This sort of rate measurement cannot be performed in LTspice without a complicated secondary circuit that is bothersome. Using a separate circuit within the simulation is also detrimental to the simulation performance as it can introduce a non-linear coupling between the counting circuit and the nanowire circuit. This coupling can cause the SPICE solver to take longer to run the simulation and cause the nanowire model to misbehave. The effect of coupling nonlinear circuits is further discussed in Section 3.

By hooking a post-processing function to the WatchDog class, spice-daemon can call this function at the end of each simulation. Using PyLTSpice's LTSpiceRawRead function, the contents of LTspice's RAW output file can be dumped into a Python object. This object separates out each individual trace (voltage at a node, current through a component, time, etc.) from each simulation step (a single run produced by the .step SPICE directive) as separate waveforms. The post-processor can then perform any kind of analysis needed from one or multiple simulation runs. This data is accessible in spice-daemon as NumPy arrays and can be extended to new types of post-processing in the same way Dynamic Models can. spice-daemon can then update the plot automatically after each simulation run is completed.

The inclusion of post-processing rounds out spice-daemon allowing pre- and post-computation to all happen in an environment optimized for the specific device. While packages like PyLTSpice can be used to manually run a script, spice-daemon provides the ability to generate the plots after each simulation invocation automatically. It also gives a standard way of building post-processing blocks that are directly related and parametrized by the circuit schematic. Post-processing objects can be customized to be device-specific. For instance, by instantiating a PCR object, spice-daemon automatically infers that the nanowire circuit is of interest and allows for more compact toolkit specifications. Since spice-daemon has access to the global variables used in the SPICE simulation, it can also use them in its computation – a feature missing from most circuit simulation toolkits.

Toolkits require uniform spacing between datapoints, but SPICE simulators don't have fixed timesteps, causing the need for converting data between toolkit invocations. When the RAW file output is parsed in spice-daemon, the PostProcessing class (parent to all toolkits) uses extrapolation on these waveforms. The function PostProcessing.trace(trace_name, step) returns an extrapolated numpy array of the waveform that has uniform spacing corresponding to the minimum step size. This can get really large for a smooth signal if all the datapoints are concentrated around one transition (rising edge of an SNSPD) and can be changed by adding the optional parameter dt that defines the minimum timestep to extrapolate from.

2.7.1 IV curves

IV (current-voltage) curves are an important tool for characterizing the operational margin for a device and ensuring the device was fabricated according to the design. By plotting the current passing through the device as a function of the applied voltage, IV

curves provide valuable information about the device's behavior: the critical current, normal resistance, and the hysteresis margin. Verifying these properties is a common indicator that the fabrication process was successful. This form of process verification and quality analysis requires an accurate comparison to provide a useful metric. Being able to simulate the curve for a sample given the desired geometry and comparing it to the actual fabricated sample provides useful insight into what could have gone wrong during fabrication.

Given the standardization of IV curves, a good simulation environment for superconducting nanowire based devices should be able to generate IV plots readily. While a pseudo-DC simulation for the IV curve can be performed using a slow ramp of bias current, it is bandwidth limited, not an accurate representation of the device physics, and takes much longer to perform than doing parallel measurements for the IV curve of a device. Since spice-daemon has the ability to control models and post-analyze signals, we are able to perform an IV curve measurement through the spice-daemon toolkits (post-processing) framework. By running multiple simulations of a PULSE voltage bias source across the nanowire we can accurately get the DC operating point solution for the nanowire and reconstruct the measured current and voltage across the nanowire from each simulation into an IV curve. A PULSE source is used instead of a DC source for two main reasons: (1) simulating hysteresis and (2) the nanowire model is not DC-compatible (discussed in section 3).

2.7.2 Power Spectral Density

The Power Spectral Density (PSD) of a signal is a useful tool for designing and studying superconducting single-photon detectors. It is a measure of the power present in a signal as a function of frequency and can provide valuable information about the

frequency content of the signal. This can be particularly important in nanowires, as the PSD can reveal the level of noise present in the system and help identify any potential sources of noise that may impact the effective critical current. The spice-daemon toolkit can be used to generate PSD plots in real-time and can optionally incorporate it into the LTspice user interface.

LTspice cannot generate a PSD plot natively and it is unlikely that LTspice can generate such a plot without a post-processing framework. Note that LTspice allows users to plot the Fast Fourier transform (FFT) of a signal but not the PSD. spice-daemon handles that by taking the FFT in Python and plotting it either in Python's plotting package matplotlib or by injecting a new node into the RAW output. In addition, it allows for the use of windowing functions.

2.7.3 Output Methods

Post-processing modules need to somehow display the signal back to the user. spice-daemon handles this in one of two ways. The preferred method of displaying a plot uses matplotlib to spawn a separate window that contains all the post-processing plots. The other optional way of doing it is by relying on PyLTSpice's LTSpiceRawWrite to write new waveforms as specified by the post-processing module. This method is less preferred as it allows for less customization for plots, it only supports a square plotting window, shares the x-axis, and doesn't allow for additional markers. As such, unless explicitly specified, the matplotlib backend is used by default. The matplotlib plotting backend also has multiple choices of backends and allows for integration with jupyter notebooks.

spice-daemon provides two options for visualizing post-processing toolkits. The de-

fault option uses matplotlib to generate a separate window with all the post-processing plots. The other option uses PyLTSpice's LTSpiceRawWrite to write the waveforms to the RAW file directly. This option allows for the plots generated to be laid out in the default LTspice waveform viewer. This method is less flexible, supporting only square windows, forces sharing the x-axis, and doesn't support using markers. It also forces extrapolation at export such that the time axis and exported axis are time-synced. The matplotlib backend on the other hand allows for more customization and can be integrated with-in jupyter notebooks and various other matplotlib backends.

Post-processing toolkits also allow for saving data in a csv and/or Python objects. Toolkits can be used for large-parameter sweeps where the post-processing toolkits can invoke what the next parameter to search is. For instance, you can implement gradient descent in LTspice using post-processing toolkits since they have the ability to hook to modules. Given an arbitrary nanowire circuit, you might want to find the maximum bias that doesn't switch the wire (accounting for noise). You can use the post-processing toolkit to invoke a sequence of LTspice simulations to find the optimal bias value through gradient descent.

2.8 Outlook and applications

The two toolkits demonstrated in this chapter, qnn-spice and spice-daemon, showcase the additional benefits to having a separate program from the typical SPICE simulator.

The qnn-spice toolkit allows for the efficient syncing of multiple repositories, serving as the base for all git repositories and SPICE models produced around the nanowire model. It serves as a method to efficiently version track and update all

the SPICE models produced by the Quantum Nanostructures and Nanofabrication (QNN) group in a way that is easily accessible to SPICE modeling software. Furthermore, it allows for easily onboarding new users and developers to all the QNN group's SPICE model infrastructure.

The spice-daemon toolkit provides additional computational features to SPICE solvers that are not part of typical SPICE modeling software. The toolkit allows precomputing workflows that can change the circuit topology, compile complicated circuit models, precompute model parameters, as well as, model noise efficiently. This layer of abstraction adds modeling power that is applicable beyond simulating nanowires only, especially for distributed modeling such as modeling travelling-wave parametric amplifiers and microwave control circuitry. This modeling extensions allow us to further improve our understanding of superconducting nanowires as opting to include separate effects can be made easier, such as studying the sources of noise in SNSPDs or the effect of a distributed line.

The post-processing ability of spice-daemon makes it easier to perform more complicated signal analysis as part of the SPICE workflow. This adds the ability to perform typical nanowire-related measurements, such as photon count rate curves and IV-curve measurements, as well as more complicated logic, such as filtering or generating the power spectral density of a plot.

The spice-daemon toolkit is also present as a python package that can be used as part of the standard python programming workflow, allowing for tight integration between SPICE and Jupyter notebooks. This toolkit can also be extended to model lab equipment and have the same code that generates and parameterizes the SPICE models, also control the equipment in the lab.

Chapter 3

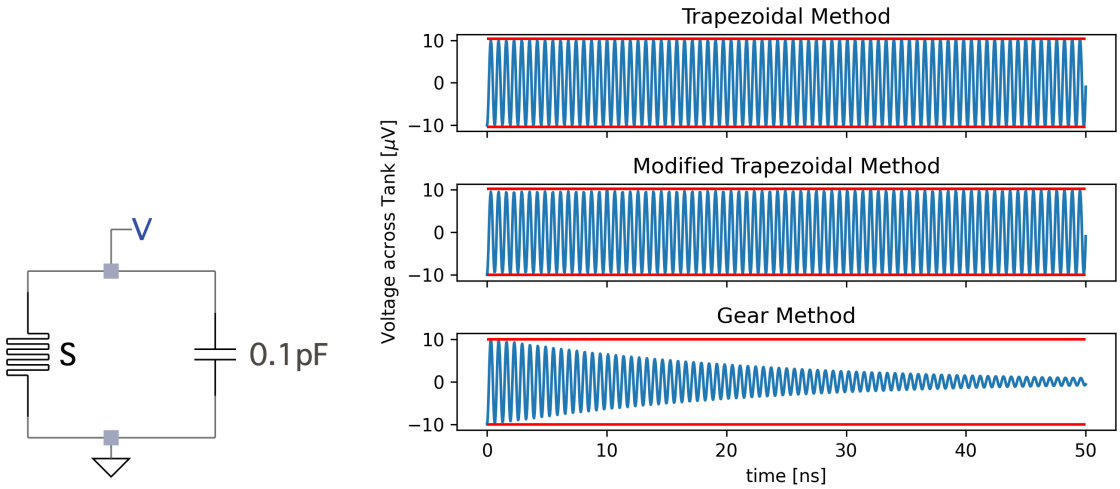
Model Stability

When we start to care about modelling multiple nanowires and including more complicated dynamics (such as thermal devices, hTrons, or multiport nanowires, nTrons), the stability of the underlying base model is crucial. The nanowire model is the basis for all these models and is unreliable leading to switching behavior due to its non-linearities constantly. The SPICE environment wasn't built to handle harsh non-linearities such as the state non-linearity and therefore requires a careful analysis of how submodels interact together to guarantee the correctness and stability of the model. We introduce a simple enumerative method that can be used to debug non-linear (and linear) circuits and compare their stability. We then use this method to improve the stability of the nanowire model by changing the hotspot integrator.

3.1 Integration in SPICE

SPICE implementations in general rely on second-order integration, namely trapezoidal and Gear integration. LTspice allows the user to pick between 4 options: trapezoidal, Gear, (1st Order) Backward Euler and a proprietary modified trapezoidal method. In general, Backwards is the most stable and least accurate, followed by Gear integration [29, 10].

Trapezoidal integration is generally faster and more accurate than Gear but introduces a ringing numerical artifact that occurs at adjacent timesteps on stiff systems. Ringing is dampened by the Gear method causing this numerical artifact to be mostly eliminated. The Gear method achieves that by dampening most ringing as shown in figure 3-1. The setup in figure 3-1 uses an nanowire-capacitor oscillator working in the linear regime starting with an initial amount of energy. We typically care about oscillatory behavior in nanowires that can be filtered by the Gear method (especially in the timeframe shown) and as a result, we choose to use the trapezoidal method. LTspice's default integration method is a proprietary modified version of trapezoidal integration that cancels out the trapezoidal ringing introduced by the regular implementation without numerical dampening [29].



(a) (b)

Figure 3-1: Comparison of the 3 different second-order integration methods LTspice is equipped to use. (a) An LC resonator that uses a non-linear inductor (nanowire) in parallel with a capacitor. The initial condition for node V is set to 100μV. (b) The voltage as a function of time computed using the 3 different integration methods. Gear method causes significant signal decay when there shouldn't be decay at timescales we typically care about in nanowires. The modified trapezoidal method has less consistent magnitudes of the output sine wave.

3.2 Stability in Transient Simulations

Stability of a finite element method is intimately related to the consistency and convergence of the method through the Dahlquist Equivalence Theorem [39]. One result of this for non-linear systems, such as the nanowire model, is their solution should be smooth as you decrease the timestep. As in, there must exist a timestep Δt for which all timesteps $< \Delta t$ the method gives a result bounded around that of using the timestep Δt. Transient simulations, which are a type of continuation method parametrized with time, often use straightforward convergence correction. In this method, the timestep for continuous signals is decreased until convergence is achieved, which is guaranteed to occur for continuous signals [10].

One way of visualizing solving a continuous system using a finite element method is by represent timestep corrections as projections. For instance, solving for the final state $u(T)$ for a circuit C at a time T transforms $u(0) \rightarrow u(T)$ smoothly when continuous. However, when a finite method is used with coarse discretizations, the method steps around this continuous evolution. Overshoots due to the coarseness could exist outside the state-space and the solution trajectory but are corrected for as illustrated in figure 3-2. These corrections (decreasing the timestep and adding gmin capacitances) can be thought of as projections back into the subspace of possible solutions. The subspace of possible solutions is a lightcone around the current state where the size of the lightcone is constrained by the circuit topology and simulation parameters such as reltol. The lightcone of a state is the set of states you can reach from that state in one timstep. The lightcone of a target state is the set of states that can reach it in one timestep.

In a non-linear system, these projections can lead to integrating errors. Consider the state non-linearity for instance, if a projection happens to enter that regime at

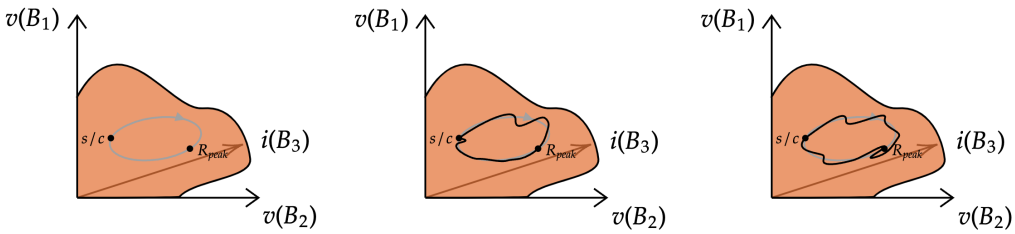


Figure 3-2: A notional diagram of the evolution of a circuit simulation for a nanowire. The orange space represents all the states (combinations of node voltages and currents) that the circuit can be in. The experimentally-observed continuous dynamics of the nanowire follow the gray line's evolution between two points, the superconducting (s/c) state and the maximum resistance state (R_{peak}). The system in the first plot encodes multiple simple configurations of superconducting nanowires, such as SNSPDs and relaxation oscillations, with the three axes being the behavioral sources in the nanowire model. A nanowire in the superconducting state starts at the (s/c) node. When a photon is incident, it begins to evolve around the path crossing through R_{peac} and returning to the (s/c) node. The second diagram shows a finite approximation of this path that a solver might take, meanwhile producing the correct number of state transitions. The third diagram shows how a projection caused by a finite method could cause the wire to trigger twice. The finite method overshoots R_{peak}, corrects to before it, and crosses it again (causing two SNSPD spikes).

any point in the simulation accidentally, it would cause a misfire. Now consider the coupling of this to the continuous non-linearity which amplifies values. This amplification can make it easier to reach the state non-linearity lightcone and accidentally trigger the hotspot integrator.

3.2.1 Stability of the Nanowire Model

Boolean state non-linearities are integral to optimizing and training Neural Networks, and as a result are a well studied concept. A common way of solving this issue is smoothening out the change by modeling the boolean state as a continuous state tran-

sition with a smooth interpolating function, such as a sigmoid function. In nanowires, we particularly care about smoothness of state transition over time while the transition dependence is on current (which is also a function of time). Macroscopically, this state transition involves a chain reaction and can be modelled as a smooth ramp up (the hotspot thermal growth is a smooth change). This is how the hotspot integrator in the existing nanowire model handles the non-linear transition into the resistive state.

The nanowire model is unstable and inconsistent over both Boolean and continuous non-linearities (discussed in section 1.1) in a disguised fashion. While this can be improved by tweaking the relative tolerance (reltol) of the simulation, improving the model's stability is essential to scaling devices and integration with other circuits. One big issue with the instability is the behavior of the model is not out of question, in that, even though the solution is incorrect, it looks plausible. For instance, you might see extra counts when you aren't supposed to see them when you include tapers (due to the modified time-stepping behavior for circuits when tapers are included, discussed in 2.5.1). Other instability effects include things like pre- and post-firing of the nanowire, which can be hard to discern from reflections occurring in the circuit that might actually cause the wire to re-fire.

This instability over the non-linearity is amplified due to the time-stepping. An approximate solution to a non-linear system has a response dependent on the magnitude at the previous timestep. As a result, near portions of the transfer functions that can't be approximated as linear, a projection onto an acceptible relative tolerance value at timestep n does not correspond to an acceptable relative tolerance at a timestep $n+1$ due to the magnitude dependence. This forward propagation of error explains false switching events at timesteps that are too close together caused by a time-coupled element (more on that in 3.3). We see this in effect in figure 3-3 with

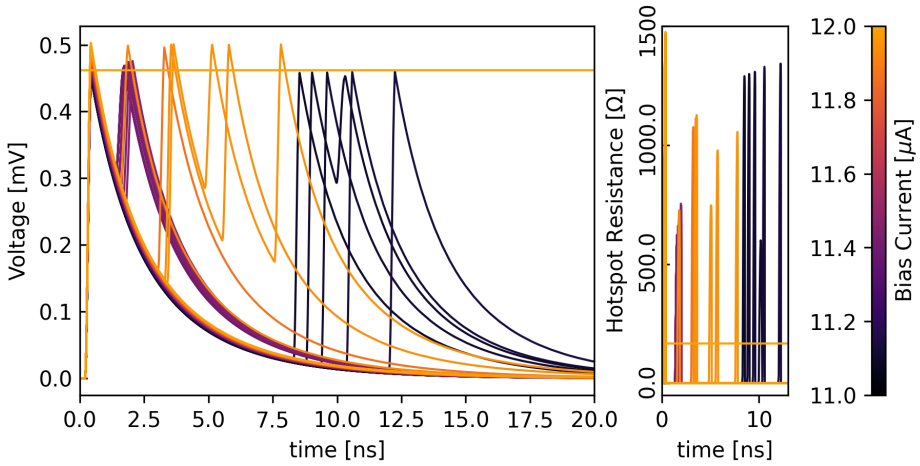


Figure 3-3: SNSPD readout circuit tested against 100 different bias values between 11μA and 12μA for a device with switching current 12μA. The simulation was carried out using the existing nanowire model and the default options for LTspice on Mac (reltol of 10^{-3}, voltol of 10^{-6}, trtol of 2). We expect to see only one pulse at 10ns for all biases (except 12μA). The instability of the model is related to the ratio of correct and incorrect simulations.

multiple false switching events.

3.2.2 Relative Tolerance for the Nanowire Model

Relative tolerance in SPICE is a simulation parameter that imposes a convergence criterion. It is typically imposed in the form [40]:

$$|v_{k+1} - v_k| \leq \text{reltol} \cdot \max(|v_{k+1}|, |v_k|) + \text{vntol}$$

This imposes a condition on the change in node voltage v between iterations k and $k+1$. vntol is equivalent to the voltage resolution and should be at least one order of magnitude smaller than any node voltage (including the thermal integrator,

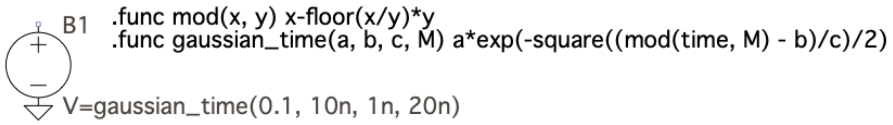
more on that in section 3.4.1. Even though the nanowire might be a 2-port element (ignoring the photon port), the relative tolerance is a check performed on the entire circuit. This implies that the relative tolerance needs to satisfy this constraint on all logic inside the nanowire.

3.2.3 Behavioral Sources

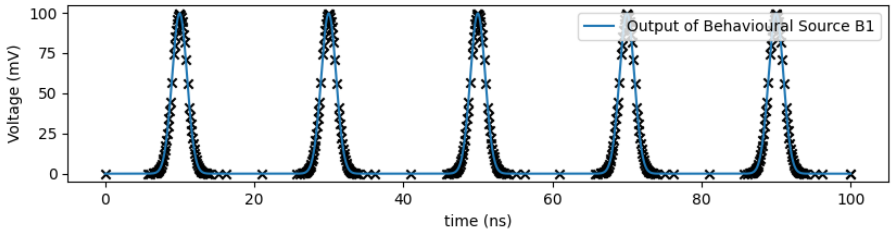
Dependent sources are a powerful model in SPICE software that allows their output to be dependent on other node voltages and currents. Along with the multiple types of dependent sources that are supported by SPICE (e-, f-, g- and h-sources), LTspice supports an additional type of dependent source called the behavioral source (b-source). Behavioral sources can mimic the behavior of any other source type and additionally can have multiple inputs (as opposed to the other sources' dependence on only one value). B-sources also allow for the use of arbitrary math functions that allow them to compute non-trivial expressions, such as time integrals, derivatives and modulos. B-sources can use the outputs of functions defined using the .func directive as a current, voltage, resistance and power outputs (resistance and power aren't well documented).

For example, one can construct a gaussian pulse source using dependent sources by defining the two functions mod and gaussian_time as shown in 3-4(a). Using a b-source expression V=gaussian_time(0.1, 10n, 1n, 20n) outputs a gaussian pulse with magnitude $0.1V$ with peaks spaced out by 10ns with a standard deviation of 1ns as shown in figure 3-4(b).

B-sources are a helpful tool when designing circuits for stability as they allow you to abstract complexity away from the circuit topology and remove reliance on reltol.



(a)



(b)

Figure 3-4: Example of a behavioral source outputting gaussian pulses.

They can also function as state variables and perform math on expressions that have varying output values – such as performing math on outputs that differ by more than 6 orders of magnitude.

3.3 Malicious Circuits

One method proposed to test for the stability of a nanowire model is by enumerating the number of false switching events. In the same fashion that the state nonlinearity can be used to amplify single events of small magnitude, the model can be used to nanowire count erroneous state crossings. If a simulation Σ_C of a nanowire circuit C accesses a state $\Sigma_C(u(t=0), T) = u(T)$ after a time T using a finite method, then running the same method with more steps should yield $u(T)$ if the method converges [39]. However, if the projections taken between simulation steps due to criterion based

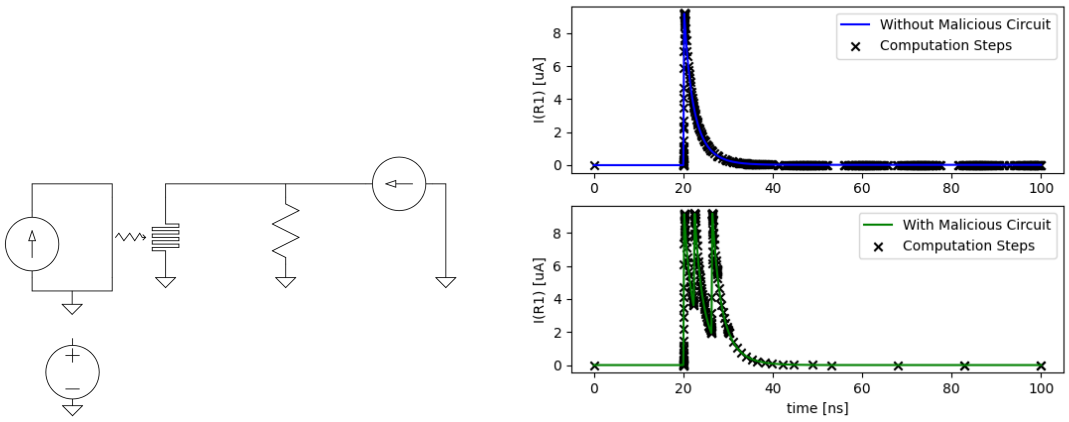
convergence introduce some deviation ε at time t' from the true circuit state, then the final solution will evolve from an ε lightcone away from $u_C(t')$. In a linear system, the ε lightcone might be hard to distinguish as $\Sigma_C(u(t=0), T) + \Sigma_C(\varepsilon, T - t') \approx \Sigma_C(u(t=0), T)$.

However, since the nanowire's state is non-linear, if the ε deviation happens near a switching event, then the projection has a probability of switching the nanowire. By enumerating simulations of a nanowire varying the timestepping, we can force the find the frequency of ε errors occuring from switching events.

Unfortunately, we can't perform this enumeration by varying timesteps in LTspice trivially. One workaround however, is simulating a circuit $C' = C + M$ that contains the original circuit C and a new malicious circuit M. M does not have to be coupled to C topologically (a particularly simple and interesting case is when M and C are topologically decoupled: sharing no nodes, currents or behavioral coupling). C and M however are always time coupled: if M exhibits a strong non-linearity and smaller timestepping to account for that, then the simulation of C also experiences finer timesteps. This time coupling allows for enumerating simulations of C without manually modifying the timesteps taken.

One example malicious circuit M could be a voltage source exhibiting a pulse with a magnitude at least as big as (voltol)/(reltol) as shown in figure 3-5(a). This forces the simulator to use finer steps near the pulse edges and as a result introduces some timestep variation to C as well. Another example malicious circuit could be a one-way coupled circuit, such as a behavioral source with an output coupled to a node in C and amplifies its magnitude. This is equivalent to having the simulation tolerances be different for one node in the circuit.

A special case for M is the transmission line. The inclusion of a transmission line



(a) (b)

Figure 3-5: (a) diagram of the circuit tested with a bias current of 11.1μA and a photon incident at 20ns on the existing nanowire model. The bottom circuit is a "malicious circuit," it is a pulse source that produces a 10ns square pulse during the evolution of the hotspot. (b) The top plot showcases a single hotspot forming when simulating the SNSPD circuit without the malicious circuit included. The bottom plot shows the evolution when the malicious circuit is included. The bottom plot shows multiple oscillations with peaks even though it should only show one. The malicious circuit projected errors on the hotspot evolution and caused the nanowire model to switch when it wasn't supposed to. The crosses indicate the individual timesteps the solver computed the waveform at.

in SPICE creates breakpoints that force the timestepping resolution for the entire simulation to be below half the total line delay [11, 10]. One possible enumeration would be varying the delay of an LTRA (O- or T-models). This works even when the transmission line is not connected to anything.

Figure 3-5(a) shows a typical SNSPD readout circuit biased with 11.1μA and a decoupled malicious voltage source. By running a simulation with and without that malicious subcircuit, we can see that the inclusion of a malicious circuit affects the output result in figure 3-5(b). Note that the resulting solution is not possible given

the typical geometry, even though it looks plausible, since the period of oscillations is not regular. Since we know the malicious voltage source in no way affects the main circuit, we know that our model is unstable over time steps taken by PULSE sources. Note that not all bias currents misbehave, but by sweeping a range of bias values, we are able to find a couple that don't work as shown in figure 3-3. About 6% of bias values output inconsistent results for a sweep with between 11μA and 12μA in steps of 10nA.

This method can be extended to detect the stability of linear systems by coupling a non-linear discriminator D to the circuit C and testing the stability by simulating $C+D+M$. The discriminator D needs amplify ε deviations, the most straight-forward way of doing that projects a node value from C onto a finite field. For instance D could be a behavioral source with an expression similar to $V = \mathsf{IF}(\mathsf{n001} > 10\text{u}, 1, 0)$ to couple to node n001 in circuit C. D should have no effect on the operation of C, any operation change in C is an indicator of instability of C.

3.4 Improving the Nanowire Model

Now that we have a method to measure the stability of a subcircuit, we can use that to measure and improve the nanowire model. The nanowire model's constituent subcircuits are tested separately using the Malicious Circuits method to assign a relative stability score. Using that method, we can highlight what subcircuits of the nanowire model are unstable and change the topology such that their governing dynamics are the same as the old subcircuit but don't impose the same instability over the timestepping algorithm.

3.4.1 Current nanowire model

Berggren et al.'s nanowire model comprises of four sub-circuits outlined in figure 3-6. The main subcircuit (subcircuit a), consists of a non-linear inductor in series with a b-source in parallel with a resistor. The non-linear inductor simulates the continuously non-linearity due to kinetic inductance, while the resistor and b-source simulate switching and hotspot dynamics. Subcircuit (b) stores the value of the boolean state tracking whether the nanowire is in the normal or superconducting state. Subcircuit (c) is the hotspot integrator, the subcircuit is the circuit analog for the hotspot integral solving for the node voltage v_3 which represents the hotspot resistance. Subcircuit (d) is an input port for photons.

Using dependency graphs, we can visualize how the model elements are correlated in figure 3-7. By using directed graphs to represent execution order for nodes and element values, we can visualize the dependece of parameters on each other. A system is said to have a dependency graph $G = (V, E)$ where V represents variables as nodes and E are the dependency edges. A set of variables $V_1, V_2 \subseteq V$ are said to be decoupled if all nodes in V_1 have node edges pointing to V_2 and vice versa. The dependency graph for a system highlights how projection errors integrate over time.

The current nanowire model also currently lacks a DC part (operational mode) of the model. LTspice DC solver struggles to efficiently represent a nanowire that is undergoing relaxation oscillations (the solver tends to find that it is switched) and misrepresents DC solutions for the superconducting state as the normal state. This implies that for nanowire simulations, the initial state of everything must be zero since transient analysis relies on operational point analysis beforehand. Namely, all bias sources, DC or not, should start at 0. So a DC bias would be a PULSE starting at 0, ramping to a value and let it rest there for a bit. This is a by-product of LTspice

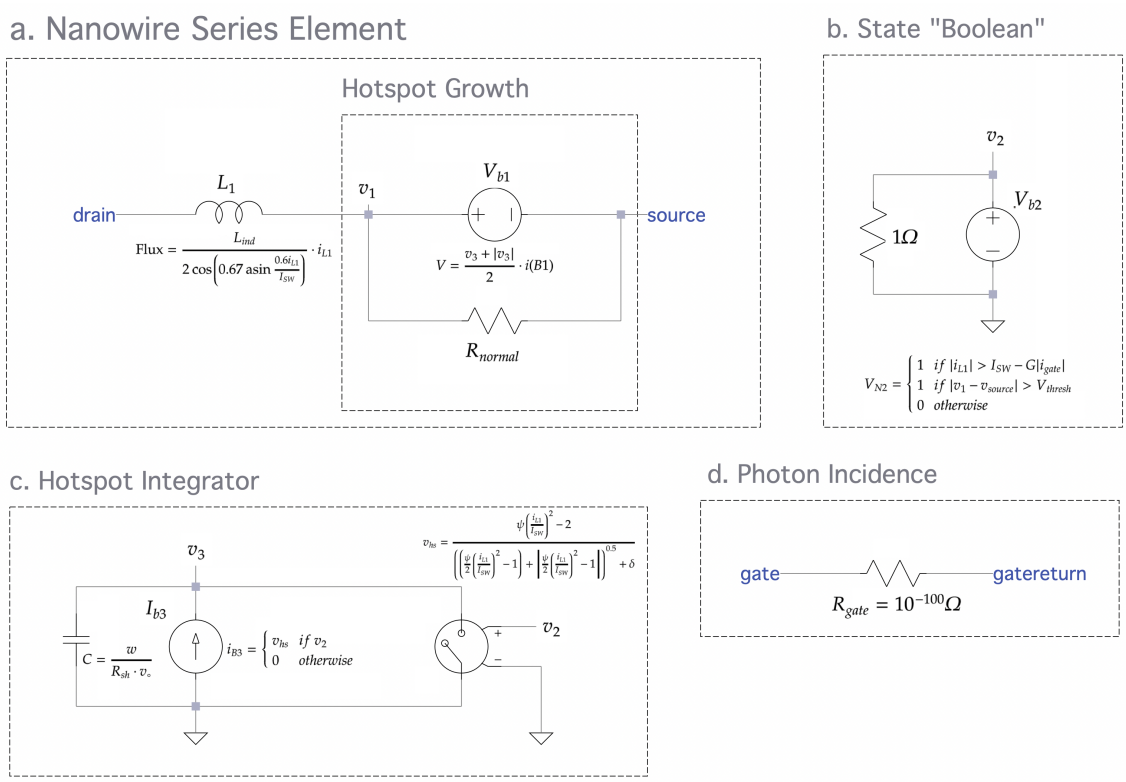


Figure 3-6: Diagram showcasing the four subcircuits used in the Berggren et al. SPICE dynamic nanowire model. Subcircuit a accounts for the kinetic inductance continuous non-linearity and the resistance in the normal state. Subcircuit b tracks a boolean state of superconducting vs. normal. Subcircuit c integrates the hotspot velocity as per the phenomenological hotspot model. Subcircuit d is the photon inlet.

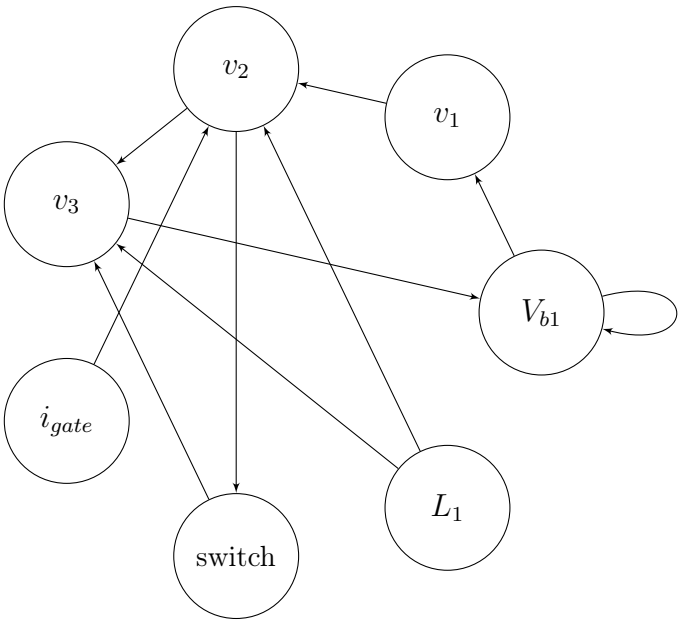


Figure 3-7: A dependency graph for some elements and nodes of the nanowire model as presented in figure 3-6. Each node in the graph represents a circuit node value that is calculated or an element parameter. Since the compiler is a black box, some degree and outdegree 1 nodes were removed. Note that the time dependence of parameters is also removed, i.e. an edge to a variable could represent dependence in the same timestep or on the previous timestep of the value. This is a heuristic of how simulation parameters are dependent on each other, i.e. how stiffness or errors in one graph node couple to other nodes.

not recognizing meta-stable systems. This is tackled by the new simulator introduced in section 4.

3.4.2 Stability of the original nanowire model

By varying the relative tolerance and testing different geometries, the value 10^{-6} seems to perform the best in terms of stability and accuracy. By testing the LC tank geometry shown in 3-1(a) using a nanowire and swapping it with a linear inductor, we can test several reltol values and see if we see the expected oscillation behavior. Intuitively, as the relative tolerance value used gets smaller, the circuit should be "more correct" than when allowing for a bigger tolerance. For the nanowire model, we don't see that behavior. Relative tolerances between 10^{-3} and 10^{-7} consistently worked in our tests, producing a sinusoidal oscillation. Values above 10^{-7} either switch into the resistive state and lose all the energy immideatly or oscillate in a decaying fashion. Meanwhile, for the SNSPD readout configuration, we see consistently better behavior using the malicious circuits method presented in 3.3 as we go from 10^{-3} to 10^{-6} across photon events and relaxation oscillations.

Even with a reltol value of 10^{-6}, the existing nanowire model is highly unstable and can be improved through stability analysis. Using Malicious Circuits, we test multiple operation regimes for the nanowire and study the stability of every sub-circuit separately. Namely, we care about the nanowire behavior in 4 different regimes: (1) as an inductor and resistor when normal, (2) as an inductor when superconducting, (3) the hotspot evolution when a photon is incident, and (4) relaxation oscillation regime. We test these different regimes using one circuit with one nanowire element across reltol values of 10^{-3} and 10^{-6} and compare it to the expected solution.

3.4.3 Different Integrator

The malicious circuits analysis on the old model indicated that the integrator circuit is the main source of instability. We replaced it with a behavioral source that integrates the same hotspot and observed an improvement in overall behavior. This model is mathematically identical to the previous one with 2 changes to the implementation: (1) use a built-in integrator instead of the circuit integrator (subcircuit c) and (2) replacing $\frac{x+|x|}{2}$ with a conditional on $x > 0$ (or bound x by the limit).

The LTspice built-in integrator is better handled by spice than the integrator's circuit equivalent model. It involves doing one operation instead of increasing the circuit matrix size. This approach not only involves creating fewer projections onto the circuit (leading to fewer errors) but using the built-in integrator allows LTspice to better keep track of integration time and shorting to ground. Note that the new integrator using the built-in sdt math command uses a reset condition. This condition is better handled as it doesn't involve decreasing the timestep in a projective fashion as with circuit non-linearities. The reset condition is substituting the switch that shorts v_3 to ground in figure 3-6. When the $v_2 = 0$ (wire is superconducting), the integrator resets to the initial condition 0.

The previous nanowire model also involved the switch toggling between the off and on state with resistances of $1\,\mathrm{m\Omega}$ and $10\,\mathrm{G\Omega}$. This switching behavior introduces a strong non-linearity that can be better handled by the reset method used by the sdt command. In SPICE, a resistance changing value by an order of 10^6 instantly is unstable, let alone a magnitude of 10^{15}. This magnitude change coupled with using a lower reltol leads to a large instability that should be avoided.

Replacing $\frac{1}{2}(x + |x|)$ with an IF statement (or limit) allows the circuit matrix

compiler to have an easier time optimizing the simulation. The evaluation of this statement results in x if $x > 0$, otherwise 0. While $\frac{1}{2}(x + |x|)$ preserves continuity, a carefully crafted IF statement can exhibit the same continuity. In this case, the function we are trying to replicate with an IF statement is not continuous, it would be harder to perform this optimization, however, since continuity is preserved, the speedup offered is not at the expense of stability. This implies that no advantage from a convergence standpoint is offered, however, the compiler has to perform multiple operations instead of evaluating one conditional. The limit math command is a built-in way of handling this as well, where the lower limit is 0 and the upper limit is the peak hotspot resistance. While the implementation of the limit command is unknown, its implementation (and accompanying optimization) should intuitively be similar to that of an IF statement. These two methods also allow for greater readability than the absolute value conditional.

Along with more accurate simulations of the behavior of the nanowire, we also observe the simulator using fewer time steps overall (fewer crosses in figure 3-8). This corresponds to faster convergence and also is an indicator of fewer projections that needed to be done, indicating less error over the binary state. Needing fewer points to solve a system suggests linearity. In this case, we can observe that the model can comfortably relax back into linearity when the pulse magnitude is constant and there is no switching behavior occurring. The higher density of datapoints in the improved model is reserved for the beginning of the state non-linearity transition (hotspot growth region) as opposed to being employed during the entire evolution in the old model.

Figure 3-9 compares the performance of the old and new nanowire models at a reltol= 10^{-6}. This example is for the same bias as in figure 3-8, where the current output seems correct. However, looking at the hotspot resistance by scoping the

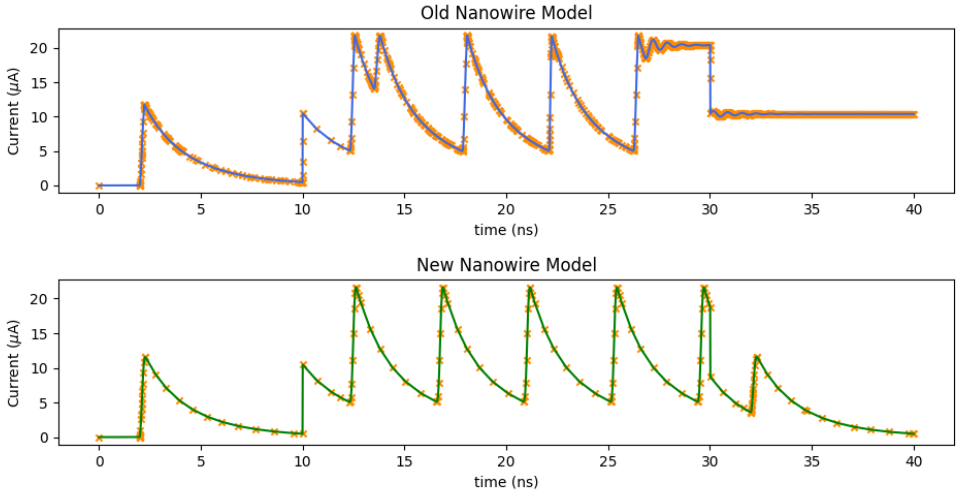


Figure 3-8: Comparison of the old and new nanowire model that replaces the circuit integrator with the internal LTspice resetting integrator. Solid lines represent the output waveform and the crosses are the values evaluated at each individual timestep. Photons are incident at 2 and 32ns on a wire biased by 15μA. The bias is increased to 25μA between 10 and 30ns to enter the wire into the relaxation oscillation regime.We see that in this example, the new model is more accurate, as well as, requiring fewer discrete timesteps to solve the equation at.

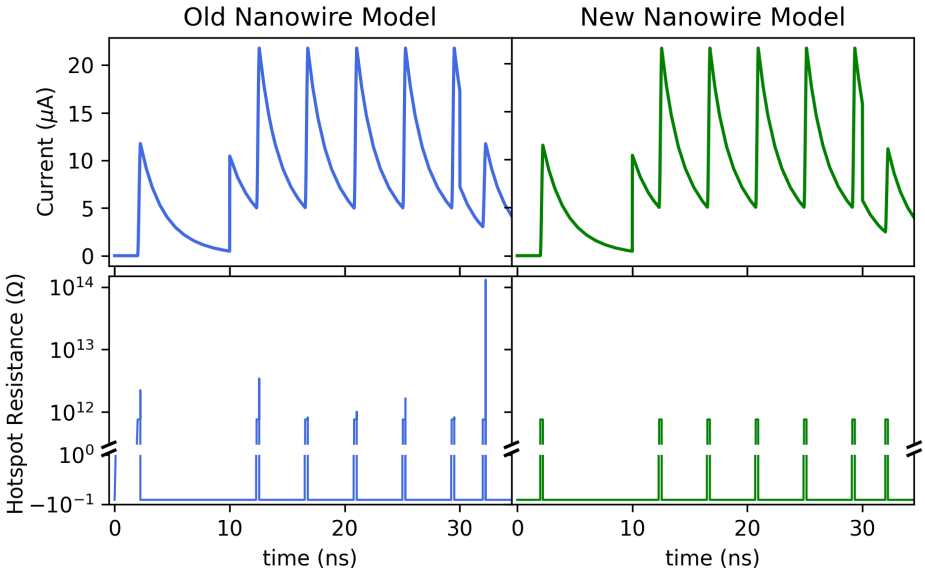


Figure 3-9: For the same setup in figure 3-8, we see that both models seem to perform well on readout when using a reltol value of 10^{-6}. However, the hotspot resistance is unstable in the old model peaking up to 2 orders of magnitude above the actual value.

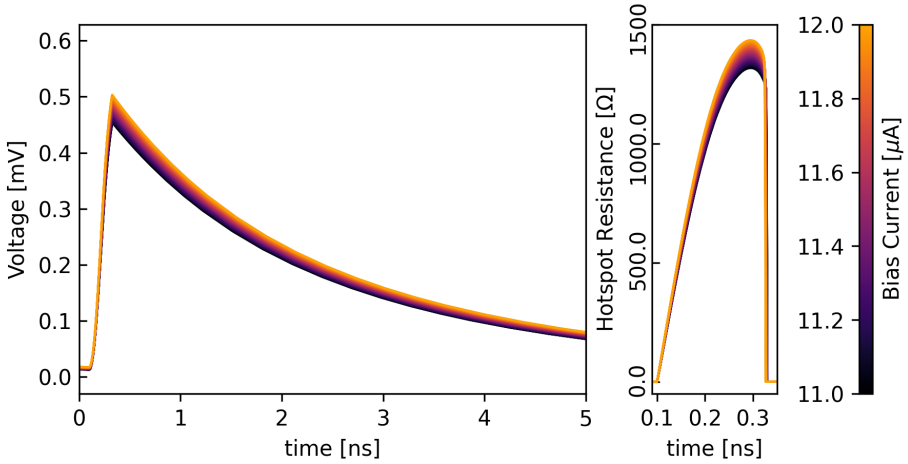


Figure 3-10: A sweep of bias currents on the new nanowire model with a photon detection event. 100 equally spaced bias values between 11μA and 12μA were tested. We observe a smooth gradient of the resultant hotspot resistance and voltage spike that is much more consistent than the old nanowire as shown in figure 3-3.

internal variables of the model, we see that the hotspot resistance has huge spikes, up to 2 orders of magnitude above the correct value. This is a source of instability in the model that can also be analyzed using Malicious Circuits, even though the hotspot resistance itself is a combined measurable ($R_{\text{hotspot}} = \frac{v_1 - v_{\text{source}}}{i}$).

We can see the same sweep performed on the original nanowire in figure 3-3 performed on the model with improved stability in figure 3-10. We can see a smooth gradient of responses form as the bias increases (as opposed to random activity seen in the old model). This is expected, as the device should experience one harsh state non-linearity. After that transition, the system to should be continuously non-linear in a fashion that can be simulated by transient simulations accurately. We also can see the hotspot peak resistance increase smoothly with the bias current in the new model, while witnessing the hotspot decaying faster. This showcases that our model is directing current to the shunt at a faster rate – allowing the wire to cool back into

the superconducting state earlier on.

3.4.4 1-Element Models

Collapsing a sub-circuit to one element can provide speed and convergence advantages, especially for nonlinear elements, at the risk of instability past a certain point. This method can be useful in applications where the nanowire is used in one configuration repeatedly. For example, in the SNSPI model generated in section 2.5.3, replacing the nanowire element with a behavioral one-element model that spikes the resistance upon a photon incidence without integrating the entire hotspot velocity. This can be done by precomputing the effect of the bias in a program like spice-daemon and reusing that computed value or by loading multidimensional PWL tables into LTspice.

A good resource for one-element models are behavioral sources (b-sources). B-sources can operate in the voltage (BV source) and current source (BI source) mode with their outputs set by an arbitrary maths expression which can be a function of global parameters such as node values. B-sources can also behave as behavioral resistors (BR) and power sources (BP), these two modes aren't well documented. BR sources are useful candidates for modelling hotspots, as they can have a 0 resistance that spikes to the hotspot peak resistance. The tolerances are more lenient for a BR than a BV source across a resistor. The nanowire model has a small $10^{-3}\omega$ resistance, which is inconsequential for most simulations. Care should be taken when using a BR source starting at zero resistance as it may cause convergence issues if nonlinear effects trigger smaller timesteps near the switching threshold.

Precomputing switching behavior outside LTspice can be done by modelling any complicated thermal model in outside software and imported into LTspice via spice-

daemon. This could involve stripping nanowires from the hotspot integrator and having voltage spikes output across it as a normal nanowire would have given the bias the resistor is experiencing. Since spice-daemon can access simulation data, the nanowire bias can be deduced either explicitly from the YAML file or by running an initial DC simulation to determine the nanowire bias.

Multidimensional tables can also be computed for nanowires and imported into LTspice. While LTspice does not support 2D tables or interpolation, it only supports 1D piece-wise-linear (PWL) interpolation through the built-in table(x, x1, y1, $\cdots$) command [41]. However this 1D interpolation function can be used as a well-convergent 2D interpolator by discritizing the search space using a finite number of squares (which is remappable to 1D), then implementing the 2D interpolation using built-in LTspice math commands. For example, a 2D map on x, y evaluating to $f(x, y)$ can be encoded by enumerating each square with the diagonal points (x, y) and $(x + \Delta x, y + \Delta y)$ and computing f at the corners of each square. This enumeration is linear, one general example mapping is $M(x, y) = \frac{1}{k_x}\lfloor k_x x \rfloor + G \cdot \frac{1}{k_y}\lfloor k_y y \rfloor$ where k_x, k_y are related to the resolution $(\Delta x, \Delta y)$ and G is related to $\max x$ and k_x. The mapping $M(x, y)$ evaluates to one unique number for each pair x, y on the corners of the squares. We can now interpolate any function f across that range for any two query points x', y' by evaluating a function f over the points $M(x, y), M(x + \Delta x, y), M(x, y + \Delta y), M(x + \Delta x, y + \Delta y)$. The weighted sum of these evaluations is a 2D linear interpolation. A different simpler method could be used (such as rounding) for a 2D table with no interpolation. The basic principle for encoding higher order maps into LTspice is to convert the space to a finite enumerated map, compute the target function over the corners of each square and encode that using the table math command.

These models require less operations and checks per iteration, allowing for faster

compute. These models are ideal for large scale simulation, such as simulating thousands of nanowires. Using a behavioral source with finitely many operations takes less than 6 operations per iteration and acts on one node. Even in a sparse setting, arbitrary subcircuits span multiple entries and force a larger number of operations a second - many resulting from individual reltol (and other tolerance) checks. Using a macro-model to encode the behavior of the nanowire decreases the number of possible stability issues. However, these macro-models encode a finite number of interactions, and as such, can act in a behavior that seems correct (since it is hard coded) when it is not. This motivates being extra careful with models designed for high performance and large-scale system modeling. Having an environment controlled check to validate the sensibility of a circuit output is a useful add-on. For example, having the pre-compute workflow in spice-daemon compute the behvioral model and the post-compute workflow check that when the nanowire switched, the current in the device was above the critical current, a photon was incident, etc.

3.5 Outlook and applications

This chapter demonstrated a simple stability analysis technique for SPICE solvers that involves counting malicious circuits – uncoupled circuits that affect each others output through timestep coupling. This method can be used to highlight unstable subcircuits or elements of a model to improve the model's stability. We use this to analyse the stability of the nanowire model to show that the hotspot integrator is highly unstable and is the main culprit for erroneous switching events seen when using the nanowire model. By replacing the hotspot integrator with a behavioral source reset-equipped integrator, we showcase a much more stable and accurate nanowire model. We also find that the most well-convergent relative tolerance for both nanowire

models is $\mathsf{reltol} = 10^{-6}$ and is essential when considering the state nonlinearity.

The increase in the stability of the nanowire model is essential when scaling up devices: as the number of nonlinear models increases, erroneous switching events become more prevalent and hard to distinguish from regular events. The more new more stable model also showcases a speed improvement when being simulated. This method can be extended to work on both nonlinear and linear circuits by coupling to a malicious circuit.

Chapter 4

Efficient Simulation

In some cases, we would like to optimize our simulations of nanowires more efficiently and in a more stable manner than in LTspice. In devices with hundreds of thousands of nanowires, LTspice model optimizations can only go so far. This raises the need for a nanowire-centric simulation environment. We designed a nanowire simulator in Julia, optimized for efficiently simulating 2-port nonlinear transmission lines. Since we care about time domain (TD) simulation for the types of nonlinearities we exploit, the model was designed to simulate the time behavior of nanowires in a manner similar to the phenomenological nanowire model [12]. We however also use nonlinear frequency domain (FD) methods to simulate FD macro-models on the fly that are reused in the TD simulation.

4.1 Transmission-Line Model

We choose to model nanowires as nonlinear transmission lines encoding both nonlinearities by default into the model. Using this method, we can get all the microwave characteristics of the device by default and optimize for this topology. Unlike in LTspice where we had to optimize the choice of nanowire model based on the microwave topology to increase performance, we design a simulator around the nanowire nonlinear transmission line and as such guarantee optimizations for the nanowire devices. On top of that, all devices on the same chip are superconducting, and as such, a nanowire simulation environment can accurately model devices that are typically modeled only in the linear regime while preserving the optimizations granted by the nonlinear solver.

The transmission-line model is modeled similarly to the SNSPI presented in figure 1-2. Modeling all nanowire transmission lines (and SNSPDs) in an architecture similar to the SNSPI is an accurate depiction of the true dynamics. Nanowire transmission lines exhibit the same nonlinearities but can have a different photon behavior which can be encoded by the user. For example, accounting for polarization, absorptive layers, etc. can be added on top of an efficient transmission-line model.

In the discrete sense that we care about for efficiently simulating for meanders with varying widths, we can realize that any 2-terminal nanowire can simulate the same classes of devices with some added differential equations. The basic nanowire model in the Julia simulator can model transmission lines, SNSPDs, SNSPIs and even tapers with low overhead for complex geometries. It can be extended to work on n-terminal devices by coupling node voltages.

We chose to employ a hybrid fixed-variable time step method over SPICE's adap-

tive time step. LTspice and most other SPICE solvers use a variable-time stepping method for its efficiency and ease-of-adapting in circuits given the nature of multiple time constants at play [10, 29]. WRspice uses a similar fixed time step simulator for simulating superconducting electronics as they care about simulating highly nonlinear superconducting elements [5]. A fixed time step method is better at generating correct solutions for the correct timestep size, however can be misleading and fail at converging to the right solution. We employ a solver that uses a Proportional-Integral controller (more in section 4.2.1) for variable timestep control, as well as, defining set-points for computing over harsh nonlinearities. One such harsh nonlinearity is the state nonlinearity, and therefore, we can choose to ask the solver to compute small timesteps near those nonlinearities through heuristic methods. We also use callbacks (more in section 4.2) to detect passing over nonlinearities that should have been addressed during timestepping as a second type of timestep size controller.

4.1.1 Equivalent Circuit

An arbitrary nanowire model can be simulated by repeating "chunks" of nonlinear inductors and capacitors as shown in figure 4-1(a). Using the trapezoidal method on the circuit is equivalent to converting each element to a current source and resistor in parallel as shown in figure 4-1(b) [42]. The inductor's non-linear inductance can be encoded into the resistor and current source by approximating the continuous nonlinearity as linear between two successive timesteps. We can then write the general matrix for any nanowire device as follows:

$$\begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & -R_{L,i-1} & R_{L,i} + R_{C,i} + R_{L,i-1} & R_{L,i} & \cdots \\ & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ v_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ i_{L,k} - i_{L,k-1} + i_{C,k-1} \\ \vdots \end{bmatrix}$$

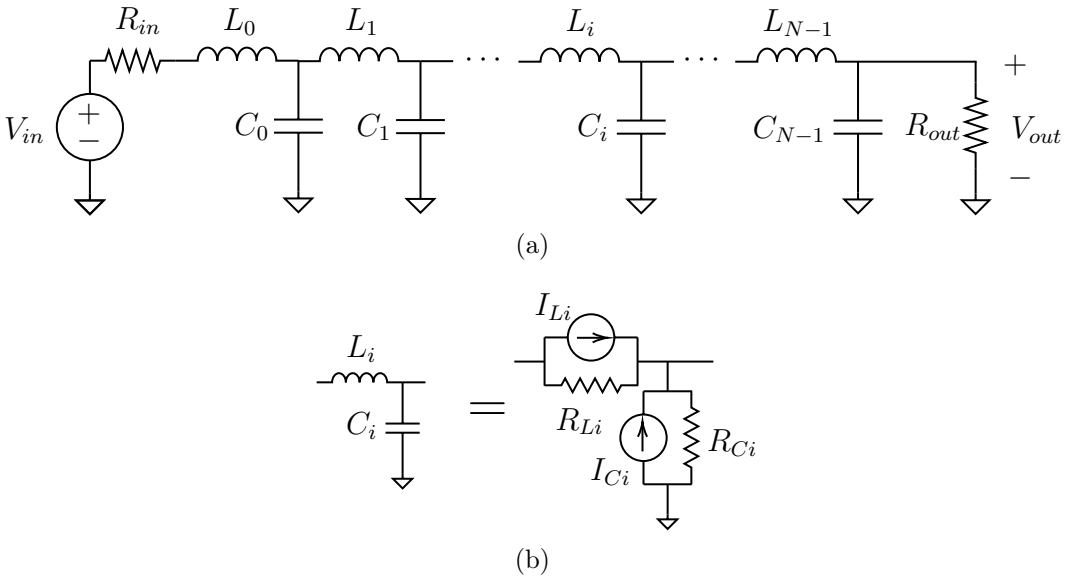


Figure 4-1: A transmission-line model with N discrete chunks. (a) a typical representation of a lossless transmission line that has N inductor and capacitor (LC) lumped elements in series with each other. On the left $V_{in}(t)$ is an input voltage that varies over time with some input resistance R_{in}. A readout voltage $V_{out}(t)$ is induced on the right across the output resistor. (b) An equivalent model used for approximating the transmission line using trapezoidal integration that can also account for DC characteristics and hotspot generation more efficiently than its LC counterpart.

The solution for this linear equation produces the node voltages v_j for the nanowire between each chunk $j+1$. This is solved for each timestep using the previous timestep as the initial condition. In figure 4-1(a), $v_0 = V_{in}$, v_1 is v_0 minus the voltage drop across the resistor R_{in} and so on until $v_{n+1} = V_{out}$. The currents $i_{L,k}$ and $i_{C,k}$ are the currents going through each inductor L_k and capacitor C_k at a given timestep.

The trapezoidal rule for the inductor above sets the resistor in the companion model shown in figure 4-1(b) to $R_{L,i}(t) = \frac{2L}{h}$ for a timestep size of h and unit inductance L. The current source's value would correspond to $i_{L,i}(t) = \frac{h}{2L} v_n(t - \Delta t) + i_{L,i}(t - \Delta t)$ [42]. This forms a complete basis for calculating any circuit parameter and evolving the circuit.

We can modify this model to account for the nonlinearity by only editing the inductor. The inductance can be overloaded by changing L to $L(i)$ to account for the continuous nonlinearity in inductance. We can also change the inductor's companion resistor value $R_{L,i}$ to account for the hotspot formed when a nanowire switches over the boolean state nonlinearity. Note that this formulation has implications on any power calculations (for example when thermally coupling layers) as the current is across both a resistor for the hotspot and one for the inductor.

4.1.2 Kernel

Our large matrix is sparse and bidiagonal, allowing us to compress the node computations required to more efficiently simulate the nanowire. Each node on the nanowire is dependent only on the states of the two adjacent chunks in the previous timestep. This allows us to utilize in-place loop vectorization on a view of a large state vector with static kernels.

The state vector for the nanowire system is defined as $u = [\ \vec{v}\ \vec{i}\ \vec{s}\ ]^T$, where $\vec{v}$ and $\vec{i}$ are the voltages and currents of length $n+1$ and $2n$ for n chunks respectively. Vector $\vec{s}$ tracks other system variables such as thermal state (temperature), hotspot velocity, normal-superconducting state, resistance, etc. We choose to implement the simulator such that $\vec{s} = \emptyset$ by default. The hotspot parameters are tracked using a separate precomputed $\vec{R}$ that evolves the hotspot as per the typical solution, in cases where the hotspot dynamics are well predictable (i.e. no strong thermal coupling).

We can perform separate repeated computations along 3 different dimensions of u, namely, we can compute $\vec{i}(t+\Delta)$ based on $u(t)$, then compute $v(t+\Delta)$ from that, followed by updating any state variables in $\vec{s}$. This repetition can be optimized and can be exploited using the concept of views in Julia. A view is a data structure that acts as a matrix or vector where the underlying values are references to an original matrix or vector. In this example, we can perform operations on the $n+1$ dimensional vector $\vec{v}$ efficiently without extracting or copying it from the original state variable $\vec{u}$.

We can employ LoopVectorization.jl to vectorize for loops or broadcast operations to improve runtime performance [43]. Loop vectorization works well in cases where the operations are linear and local. For example, we can compute $d\vec{u}$ from $\vec{u}$ using loop vectorization by formulating it as du[i]= CL[i] $*$ (du[i] + R[i] u[i]). Threads are no longer ordered and as such, the operations have to be disjoint (user cannot rely on a specific order of instructions) hence enforcing linearity and locality on vector operations.

A kernel in image processing is a constant mask used to convolve a signal. A kernel K acting on $\vec{u}$ is a local operation that computes v_i as a function of $u_{i-k}, u_{i-k+1}, \cdots, u_{i+k}$. In our case, we produce a size-3 kernel, i.e. $v_i = K(u_{i-1}, u_i, u_{i+1})$. Notice that using a size-3 kernel, a vector $\vec{u}$ is taken to a vector $\vec{v}$ with 2 less dimensions, as such it

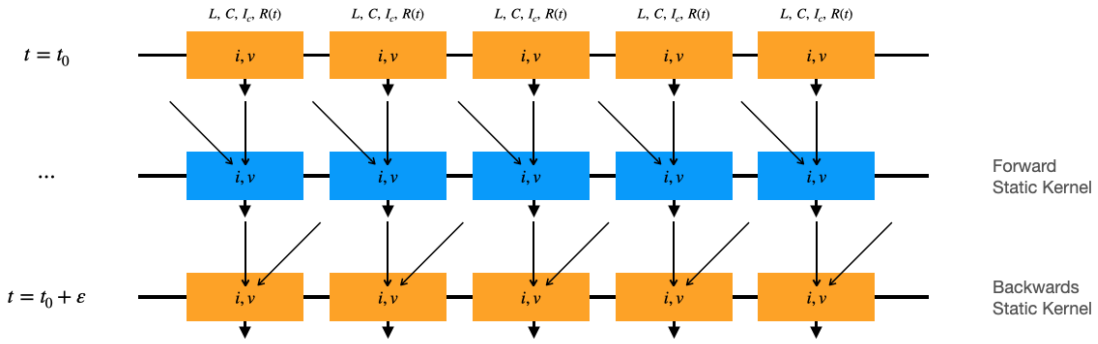


Figure 4-2: An illustration of the two kernel's used to compute the next state vector in a simulation. Each row represents a local chunk of a 2-port device with some stored state. Yellow rows correspond to a vector computed at a new timestamp, while blue rows correspond to intermediary timesteps used to generate the next timestep. All these operations happen in-place. Each row corresponds to the application of a new kernel, where the application of a forward then backwards kernel corresponds to one new timestep.

is typical to specify boundary conditions such that K is defined on the first and last entries of an input vector. Our kernel works as it corresponds to taking a local derivative, similar to the Telegrapher's equations. This kernel can be deconvolved to act as 2 separate kernels a "forward" and "backward" kernel. This deconvolution allows for faster processing and resource allocation when performing operations in-place. An illustration of the kernel we use to compute the next state vector is showcased in figure 4-2.

We can use non-allocating kernel operations using StaticKernels.jl to define efficient quick compiling small kernels [44]. The defined static kernel speeds up the computation by eliminating runtime checks, non-allocating operations, introducing boundary conditions efficiently and by the automatic vectorization of kernel loops for inlined kernel functions. Given the kernel we defined for a nanowire device, we can see that the locality of a bidiagonal matrix is an ideal candidate for using static

kernels. We use static kernels on views of $d\vec{u}$ to generate new $d\vec{v}$ and $d\vec{i}$ values for a given timestep. The kernels we define are local differences with k_f and k_b being the forward and backwards differences of the vectors, namely $k_f(w) = w[1] - w[0]$ and $k_b(w) = w[0] - w[-1]$. Kernels project a vector to one dimension lower, as a result, we extend the vectors $d\vec{v}$ and $d\vec{i}$ by V_{in} and $\frac{v_n + V_{out}}{R_{out}}$. We then apply the static kernel k_f onto $[\vec{i}, V_{in}]^T$ to get $d\vec{v}'$ and k_b onto $[\vec{v}, \frac{v_n + V_{out}}{R_{out}}]^T$ to get $d\vec{i}'$. We can then compute the change in state vector $d\vec{u}$ using the loop vectorized expression: du[i]= CL[i] $*$ (du[i] + R[i] u[i]).

4.1.3 Modeling the Hotspot

The simulator models the hotspot using two methods, a pre-computed hotspot evolution model, and the phenomenological hotspot model used in the SPICE model [3, 12].

The pre-computed model consists of a generalized fit for the solution of the hotspot evolution obtained by simulating the hotspot resistance and scaling its time and amplitude as a function of the bias current i and the critical current i_c. This approach allows for much faster simulation as you have fewer equations coupled, as well as, allowing for better parallelization. In this case, the hotspot resistance is a linear state evolution as opposed to a coupled nonlinear process. We use this by caching the hotspot resistance as a vector encoding resistance over discrete timestamps $\vec{R}$, where $R_1 = 0$ and $R_j = \max(\vec{R})$ for some time j. By scaling j and $\max R$ using a matrix $M_R(i, i_c)$, we can efficiently include the hotspot in the kernel by adding $M_R(i, i_c)R_k$. $R_k = R_0$ when the wire is non-switched, and the moment the wire switches, the state k begins to increment along the dimension of $\vec{R}$. This formulation can allow us to not only use it in efficient kernels, but also to cache the value of $\vec{R}$ into a GPUs shared

or texture memory to be efficiently reused across the simulation.

4.2 ODE Problems and Callbacks

We use DifferentialEquations.jl to repeatedly execute our time-domain kernel evolving our system in small time steps of variable width Δt. The DifferentialEquations.jl backend handles timestepping, but we also force some fixed evaluation times regardless of the variable timestep method used by DifferentialEquations.jl [45]. The DifferentialEquations.jl solvers benchmark as some of the fastest implementations of some methods as well as being optimized for high-precision and high-performance compute applications [46, 47]. The package also supports built-in interpolation, stochastic differential equations, sparsity and stiffness detection as well as arbitrary precision, allowing us to simulate arbitrary dynamics. This is important for coupling different types of differential equations to our model to simulate any arbitrary system on top of a 2-terminal nanowire.

The implemented solver also makes use of discrete and continuous callbacks, periodic checks on conditions that can be used to modify the behavior of the underlying differential equations. For example, a photon-incidence event can trigger a discrete callback, making a nanowire element switch into the resistive state at some position. Additional evaluation times are usually given with a discrete callback to ensure that the problem is evaluated during the evolution of the callback - in this case, the photon-incidence time would be an additional evaluation time. A continuous callback on the other hand involves a time-continuous function where the zero crossings modify the differential equation behavior. For example, a continuous callback over the value $\Gamma_c(t) = i(x,t) - i_c(x,t)$ can track whether a nanowire should be in the

superconducting or resistive state. If the nanowire exceeds the switching current for a time $t' < \Delta t$, a discrete callback on the nanowire simulation will likely remain in the superconducting state. However, since the function $\Gamma_c(t)$ is continuous, we know that a change in sign of $\Gamma_c(t)$ corresponds to a zero-crossing and as such, the wire is much more likely to accurately switch.

In contrast with the SPICE nanowire model, the switching behaviour on the state non-linearity is much better defined. We demonstrated in section 3 that the instability of the nanowire model is strongly influenced by the harsh state non-linearity. The use of a continuous callback for this nonlinearity makes it such that the process of entering the resistive state is a linear process. This in effect decreases the coupling between the timestepping algorithm and the state nonlinearity, allowing for more stable simulations. In the picture of projections shown in figure 3-2, we filter out cases where we evolve past the state nonlinearity, project to a state earlier in the state space and then evolve past the nonlinearity again. This leads to many fewer erroneous switching events.

We can further optimize kernel applications on larger simulations by moving parts of the solver to the GPU. This can be done using CUDA.jl and is handled by the DiffEqGPU.jl backend [48, 49]. We have two ways of compiling GPU kernels: (1) writing a custom GPU kernel optimized for GPU operations and (2) rewriting our Julia function so that the CUDA.jl package can convert it to an efficient GPU kernel. The first option is tedious but can provide a lot of optimization for specific problems. Since our kernel is small and acts locally, this would involve developing a kernel similar to that of prefix-sum [50, 51]. The second method offers similar speed-ups with typical optimizations such as using views, writing non-allocating code, relying on broadcast operations, etc. [51]. We chose to implement the second method, as it is more easily extendable if a new type of local coupling is introduced to the nanowire - for example,

including thermal effects would make a rewritten kernel obsolete.

4.2.1 Solvers

The DifferentialEquations.jl package comes with many built-in solvers, all optimized for different applications [47]. When using above formulations of nanowire nonlinearities using callbacks, the nanowire simulation is less stiff, allowing us to use non-stiff methods such as Tsit5. Tsit5 is a modified coefficient tableau for Runge–Kutta 4(5) method developed by Tsitouras that outperforms other known pairs for medium precision problems [52]. From the multiple methods evaluated, Tsit5 should be used for problems where pulses sent are relatively large and where pulse reshaping is not important due to Tsit5's lower relative accuracy. It is the fastest of the methods listed in this section.

When optimizing for higher accuracy, a method like Vern7 for Float64 precision (between 10^{-8} and 10^{-12}). Vern9 can be used for bigger tolerances using Julia's BigFloat type, however, this is not very useful for currently used models of nanowires that don't require precision below 10^{-12}. Vern7 [Vern9] is Verner's "Most Efficient" 7(6) [9(8)] Runge-Kutta method equipped with a lazy 7th [9th] order interpolant [53]. These two methods can be used in cases where the simulation is non-stiff but effects like pulse-reshaping and the continuous nonlinearity exhibited by the inductors matter. These methods are more robust than the equivalent higher order Adams-Bashforth methods to discontinuities (as well as are more efficient than the extrapolation required by Adams-Bashforth methods), which are important for state nonlinearities [47].

When the problem being simulated is stiff due to coupling with another differential

equation, we can use Rodas5P (5th order A-stable stiffly stable Rosenbrock method with a stiff-aware 4th order interpolant) for medium tolerances [47]. Rosenbrock23 can also be used for lower accuracy.

DifferentialEquations.jl supports composite integration algorithms, with the ability to switch solvers based on the stiffness. By specifiying a non-stiff and stiff tolerance, we can make a solver switch between using Tsit5 (or Vern7) when the problem is non-stiff and Rodas5 when the solver detects stiffness. The AutoSwitch algorithm can decide which method to use from a tuple for the next step based on a specified tolerance. The composite algorithms AutoVern7 and AutoTsit5 are pre-defined and weighted allowing for optimal automatic switching between the a nonstiff and a stiff solver based on the accuracy required.

Automatic time stepping in DifferentialEquations.jl is decoupled from the choice of algorithm. We use the default Proportional-Integral (PI) controller supplied with DifferentialEquations.jl. This modifies the timestep using a PI controller on the amount of error over time experienced by the solver [54]. This is similar to SPICE implementations, in that the scaled error at a certain timestep is used to either accept or reject a timestep, then used to calculate the next timestep size if it was rejected.

We might care about the evolution of the entire state vector $\vec{u}$ and as a result, efficiently tracking the state vector is important. While all our operations are non-allocating, we can pre-define timestamps to export the state vector's values at. This allows us to pre-allocate while computing the evolution in-place unlike in SPICE. The intermediate steps use the interpolation feature provided in the solvers, allowing us to compute values at well-converging timestamps without computing for the exact value. This allows us to trade-off the overhead of storing intermediate timesteps while evolving the state vector.

4.3 Harmonic Balance

An efficient method used for simulating nonlinear electronics in the frequency domain is harmonic balance and is the preferred frequency domain simulation implementation in WRspice and Xyce [5, 25, 26]. JosephsonCircuits.jl is a Julia implementation that was designed to simulate Josephson Traveling Wave Parametric Amplifiers (JTWPAs) in the frequency domain [4]. It can symbolically produced the nonlinear scattering parameters of long chains of nonlinear devices that are time decoupled, i.e. if the nonlinearity is a continuous function of current through a device. This can be an efficient way to produce the frequency response of a nanowire for certain types of inputs (namely, repeating inputs such as pulse trains) when they are in the superconducting state.

We use the JosephsonCircuits.jl solver to produce the nonlinear scattering parameters $S_{xy}(\tilde{i})$ for a device where $x, y \in \{1, 2\}$ and $\tilde{i}$ is the ratio of the current flowing through the nanowire and the switching current. However, we can't use this method to study the state nonlinearity, and as such, we have to restrict its usage for the continuous state. The output of the simulation is used to assist the time-domain simulation of long chains by producing a 1-element response oracle, similar to the S-parameter models generated by spice-daemon in section 2.6.

Given a chain of N uniform nanowire elements in series, we can compute the S-parameter response for a chain of length N/k and require $O(k)$ computations to find the response of the device. When the device switches to the resistive state, instead of simulating the entire N chunks in the time domain, we can now simulate $k-1$ chunks in the time domain and $N/k + O(k)$ elements in the time domain. This allows us to mix frequency and time domain models efficiently for nanowire devices that exhibit state nonlinearities.

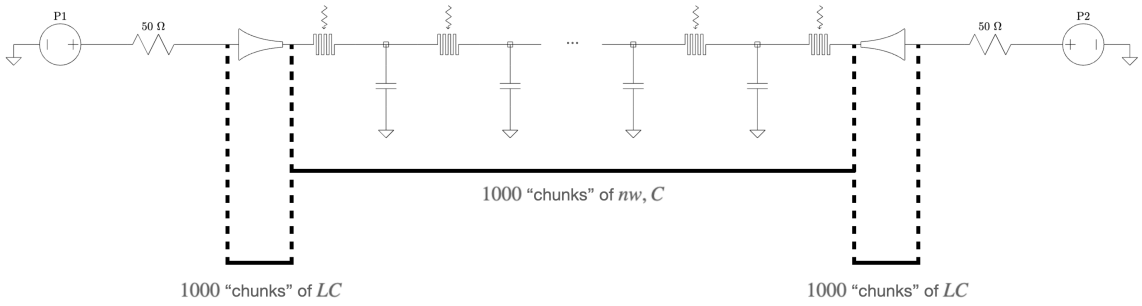


Figure 4-3: A typical device layout showcasing multiple device symmetry. The tapers are identical in shape and are always superconducting, therefore an efficient oracle can be made for one and reused for the other. The nanowire meander itself is uniform, an oracle for lumps of repeated elements can be used when the wire is superconducting, while the near critical current elements can be simulated normally.

Similar to the disclaimer mentioned in section 2.6, care should be taken when mixing frequency and time domain models as frequency-time domain mixing can cause simulation issues [37]. This should be generally avoided as it is now when designing devices that require non-pulse train inputs (even when using windowing functions), when used for pulse-reshaping analysis and when we particularly care about the switching nonlinearity. This method also removes the ability to study nonlinearity transitions using continuous callbacks, as such it could be detrimental for the performance (and accuracy) of smaller network devices.

4.3.1 Device Symmetries

One particular geometry we care about is simulating a nanowire sandwiched between two impedance-matching tapers. The tapers usually consist of the same order of number of elements as the nanowire between them. The tapers tend to be in the superconducting state the majority of the time. Given a promise of them being

superconducting the entire time, we can use the harmonic balance method to generate the taper response. This generates one oracle to simulate the behaviour of two tapers that were originally on the order of the wire, cutting simulation time by approximately a third.

4.4 Coupling Differential Equations

Given the use of DifferentialEquations.jl, we can have arbitrary systems with arbitrary dynamics coupled to our nanowire. This can be done by coupling to the state vector $\vec{u}$ via nodes and/or by increasing the dimensionality of $\vec{u}$. For example, we could model thermal transport between devices on a separate layer, electrically couple devices and differential equations that govern quantum aspects of a device for qubit applications.

Other than providing the ease of decoupling the actual nanowire solver from arbitrarily complicated dynamics, this method also allows the DifferentialEquations.jl to better utilize the automatic solver switching algorithm. For a differential equation that is not always active, for example, no heat being generated for the thermal example or no fast pulses travelling in the electrical coupling example, the dynamics are non-stiff, and the solver can simulate the nanowire using a faster method such as Tsit5. However, when the dynamics evolve to be more complicated, such as the hotspot heating a nanowire above it and nonlinearly changing the pulse shape as a function of time and current through the nanowire, the solver can switch to a slower stiff solver such as Rosenbrock23 for the duration of the stiffness.

We demonstrated two examples that showcase how coupling differential equations allows us to simulate more complex nanowire-based systems, (1) a nanowire-based Time-to-Digital Converter and (2) SNSPI Microstrip Self-Coupling

Time-to-Digital Converter

We simulate the heater-tron (hTron) to build a Time-to-Digital Converter (TDC). We write out the explicit differential equation for the hotspot using the phenomenological hotspot model to evolve the evolution of the resistance as a function of the system state $\vec{u}$. Using the power law on the hotspot, we can deduce the amount of thermal energy being dissipated by the hotspot. We then model the heat dissipation through the stack and thermal loss to the substrate using a differential equation governing the electron-phonon interactions between the multiple layers of the stack. Using this picture, we can extract how much each layer heats up, and see how that local heating up can affect a separate nanowire, allowing it to switch. We construct a TDC by laying out N parallel lines biased by a current perpendicular to one 2-port nanowire we call the "clock line". The clock line has a pulse train at a frequency f and a duty cycle $<< \frac{1}{f}$ travelling down it from port 1. An input pulse at port 2 will intersect the pulse train at a position, forming a hotspot and locally heating up the wire. The thermal coupling to the second layer switches the top device, causing a readout pulse on one of the N lines.

SNSPI Microstrip Self-Coupling

Modeling the electrostatic coupling between lines on a microstrip architecture can be done to optimize the device shape for optimal pulse readout. When designing a microstrip SNSPI, the conducting/superconducting meander is coupled to the ground plane vertically. Unlike in a Co-Planar Waveguide (CPW) geometry, the meander is not surrounded by ground on the same plane, and as a result the waveguide experiences much more capacitive coupling. Building a microstrip-based SNSPI with a densely packed meander causes pulses to reshape and arrive earlier at the readout

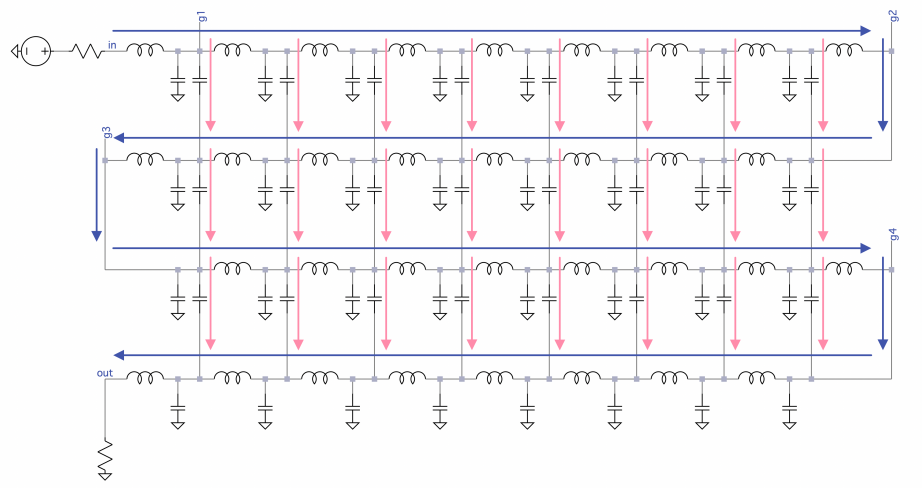


Figure 4-4: Circuit topology for a self-coupled microstrip SNSPI. As the coupling between the lines increases, an input signal supposed to travel down the blue path is more likely to couple across and travel along an alternate pink path. The ratio of the coupling to ground C_g and coupling between adjacent lines C_c controls how much the signal is transmitted along the blue versus pink paths.

ports as illustrated in figure 4-4. This makes it harder to use the SNSPI in its intended fashion of timing pulses. As a result, simulating the electrostatic coupling between multiple lines as a function of the width is useful. We can find the optimal ratio C_c/C_g, where C_c is the coupling between different microstrip lines and C_g is the coupling to ground, for a uniformly spaced line optimizing for the readout pulse distinguish-ability while trying to maximize C_c for a given line. In other words, we can find the maximally densely packed SNSPI line that still has the least amount of coupling between the lines.

This topology requires coupling every $2k$ nanowire elements to another $2k$ elements in an alternating fashion for an SNSPI meander simulated with N nanowire elements and N/k folds. This forms a square grid with k^2 connections, forming a stiff problem that loses the advantages introduced by the nanowire optimized simulator for 2-

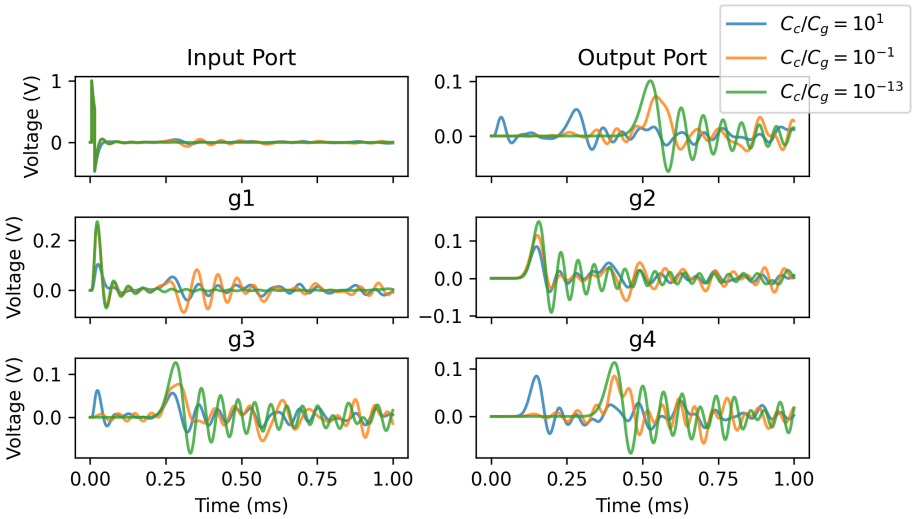


Figure 4-5: Simulation results for $C_c/C_g \in \{10^1, 10^{-1}, 10^{-13}\}$ showcasing the input and output ports, as well as 4 inflection points in the meander.

terminal devices. This can be overcome by introducing a separate differential equation that is coupled to $\vec{u}$. This coupling works using the equation of a capacitor between each node n and node $n + k$ and $n - k$. This separate coupling can also be made sparse, such that it is not computed for nodes with voltages below a certain threshold. As such, the network being simulated is a 2-terminal device except for when a voltage pulse is being dissipated along the meander. This preserves the optimizations granted by the 2-port model while being equivalent to the $O(N^2)$ coupled model.

4.5 Automated Optimization

One of the benefits introduced by a fast solver is the ability to run multiple iterations varying a single parameter to optimize your design. One particular optimization example is using gradient descent to improve a design parameter. We can use packages like Flux.jl and Optim.jl to run standard machine learning methods on dif-

ferential equations [55, 56]. We demonstrate this advantage by generating an optimal impedance-matching taper that serves as a high-pass filter using the simulator [57].

We used Optim.jl's Nelder–Mead method to design an optimized taper in conjunction with the harmonic balance backend. A linear taper of length l can be fully defined using two functions $L(x)$ and $C(x)$ for $x \in [0, l]$. By computing the scattering parameters for the linear taper, we can perturb $L(x), C(x)$ to find a more optimal geometry. The Nelder-Mead method is gradient free and as such, involves repeatedly evaluating multiple small perturbations. As such using it on the entirety of the discretization of the taper is too much. We define perturbing $L(x)$ and $C(x)$ as perturbing two vectors $\vec{v}_L$ and $\vec{v}_C$ of a dimension smaller than the discretization size. For example, we can simulate a taper using $10,000$ nanowire elements and set the dimensions of the perturbation vectors to 100 dimensions. We can then interpolate $100,000$ $L(x)$ values from 100 parameters.

Given the condition that impedance-matching tapers should be smooth to decrease reflections, the abstraction of $\vec{v}_L$ and $\vec{v}_C$ preserves smoothness for tiny perturbations. Starting from a reflection optimized taper, such as a Klopfenstein taper, we can perturb it to optimize for a specific application [18].

We define a composite loss function $f_{S_{11}} + f_{Z,\text{input}} + f_{\text{illegal}}$ that perturbs the inductances and capacitances of a symbolically generated taper. The first component is an optimization over the taper's scattering parameters. Given the $S_{11}(\omega)$ parameters of the taper (a complete basis for a superconducting 2 port device), we construct two sigmoid shaped filters around a design frequency to separate ω into two bands ($\omega_\downarrow$ and $\omega_\uparrow$), we used the design frequency of 2MHz. Ideally, we want one band to be purely passive, while the other is purely reflective [57]. Given a stop band suppression of k_S dB, we have the following loss function:

$$f_{S_{11}} = \sum_{\omega \in \omega_{\downarrow}} \left(S_{11}(\omega) \cdot \frac{1}{1 + \exp\left(k_1(\omega - k_S)\right)} \right) + \sum_{\omega \in \omega_{\uparrow}} \left(S_{11}(\omega) \cdot \frac{1}{1 + \exp\left(-k_1 \omega\right)} \right)$$

$f_{Z,\text{input}}$ ensures the input and output ports are impedance matched, i.e.

$$f_{Z,\text{input}} = k_2 \left(\left| Z_{in} - \sqrt{L(0)/C(0)} \right| + \left| Z_{out} - \sqrt{L(l)/C(l)} \right| \right)$$

and, f_{illegal} bounds our inductance and capacitance to be strictly above 0, i.e.

$$f_{\text{illegal}} = \begin{cases} k_3 & \text{if } L(x), C(x) < 0 \ \ \forall\, x \in [0, L] \\ 0 & \text{otherwise} \end{cases}$$

with three scaling parameters k_1, k_2 and k_3, (we used $-20, 10, 100$ respectively). Using Nelder-Mead on the fitted taper parameters, initial conditions of a Klopfenstein taper with some noise, simulated using the harmonic balance backend, we generated an optimized taper compared in figure 4-6 to a Klopfenstein taper.

This optimization uses Julia's symbolic capabilities, allowing it to be much more efficient. By computing a symbolic version of the harmonic balance problem, modifications to circuit parameters can be recomputed efficiently. This fast recomputation is useful for optimization methods such as Nelder-Mead since the bulk of the computation is done in the first iteration and a less substantial computation is used for the optimization parts.

A further optimization to this would rely on building a differentiable simulator.

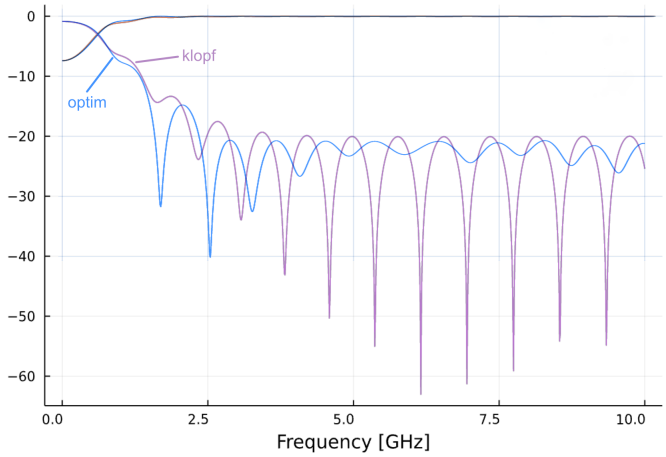


Figure 4-6: An optimized taper's scattering parameters compared to that of a Klopfenstein taper. The taper was optimized using the Julia simulator and Optim.jl's Nelder-Mead method.

In its current form, JosephsonCircuits.jl computes the harmonic balance solution using FFTW.jl and KLU.jl, both of which are wrappers for non-Julia native programs (the C implementations of FFTW and KLU in SuiteSparse) [58, 59, 60, 61]. This reliance on non-Julia natives takes away the automatic differentiation features introduced by Julia. Julia's auto-differentiation allows packages like Flux.jl to take derivatives of functions, and as such, find gradients along a parameter space [55]. These gradients are useful for optimization and allow for faster convergence. Nelder-Mead is a gradient-free method.

4.6 Outlook and applications

The Julia-based nanowire simulator demonstrated in this chapter makes use of the efficiency and speed of the Julia tool chain as well as nanowire and transmission line specific optimizations. We demonstrate a simulation method for nanowires that is

more stable, faster and more scalable than SPICE-based simulation methods. The demonstrated simulator also supports coupling arbitrary differential equations to nanowire models, allowing us to simulate arbitrary configurations of nanowires optimally. Through examples, we showcase that the simulator can be used to model arbitrary configurations of nanowires. The design of the simulator can be used beyond only simulating nanowires, but also to model any type of nonlinear transmission lines and circuits in a more stable and efficient fashion.

Bibliography

[1] G. N. Gol'tsman, O. Okunev, G. Chulkova, A. Lipatov, A. Semenov, K. Smirnov, B. Voronov, A. Dzardanov, C. Williams, and Roman Sobolewski. Picosecond superconducting single-photon optical detector. *Applied Physics Letters*, 79(6):705–707, August 2001. Publisher: American Institute of Physics.

[2] Emily Toomey, Ken Segall, and Karl K. Berggren. Design of a Power Efficient Artificial Neuron Using Superconducting Nanowires. *Frontiers in Neuroscience*, 13, 2019.

[3] Karl K Berggren, Qing-Yuan Zhao, Nathnael Abebe, Minjie Chen, Prasana Ravindran, Adam McCaughan, and Joseph C Bardin. A superconducting nanowire can be modeled by using SPICE. *Superconductor Science and Technology*, 31(5):055010, May 2018.

[4] Kevin O'Brien. JosephsonCircuits.jl, January 2023. original-date: 2022-06-15T16:07:09Z.

[5] S.R. Whiteley. Josephson junctions in spice3. *IEEE Transactions on Magnetics*, 27(2):2902–2905, 1991.

[6] Andrew J. Kerman, Eric A. Dauler, Joel K. W. Yang, Kristine M. Rosfjord, Vikas Anant, Karl K. Berggren, Gregory N. Gol'tsman, and Boris M. Voronov. Constriction-limited detection efficiency of superconducting nanowire single-photon detectors. *Applied Physics Letters*, 90(10):101110, March 2007. Publisher: American Institute of Physics.

[7] Di Zhu. *Microwave engineering in superconducting nanowires for single-photon detection*. Thesis, Massachusetts Institute of Technology, 2019. Accepted: 2020-03-09T18:59:00Z.

[8] Andrew J. Kerman, Eric A. Dauler, William E. Keicher, Joel K. W. Yang, Karl K. Berggren, G. Gol'tsman, and B. Voronov. Kinetic-inductance-limited

reset time of superconducting nanowire photon counters. *Applied Physics Letters*, 88(11):111116, March 2006. Publisher: American Institute of Physics.

[9] E. Afshari and A. Hajimiri. Nonlinear transmission lines for pulse shaping in silicon. *IEEE Journal of Solid-State Circuits*, 40(3):744–752, 2005.

[10] Kenneth Kundert. *The Designer's Guide to Spice and Spectre*. The Designer's Guide Book Series. Kluwer Academic Publishers, Boston, 2003.

[11] Star-Hspice Manual, July 1998.

[12] Andrew J. Kerman, Joel K. W. Yang, Richard J. Molnar, Eric A. Dauler, and Karl K. Berggren. Electrothermal feedback in superconducting nanowire single-photon detectors. *Phys. Rev. B*, 79:100509, Mar 2009.

[13] Qing-Yuan Zhao, Daniel F. Santavicca, Di Zhu, Brian Noble, and Karl K. Berggren. A distributed electrical model for superconducting nanowire single photon detectors. *Applied Physics Letters*, 113(8):082601, August 2018. Publisher: American Institute of Physics.

[14] Daniel F. Santavicca, Jesse K. Adams, Lierd E. Grant, Adam N. McCaughan, and Karl K. Berggren. Microwave dynamics of high aspect ratio superconducting nanowires studied using self-resonance. *Journal of Applied Physics*, 119(23):234302, June 2016. Publisher: American Institute of Physics.

[15] Emily Toomey. *Microwave response of nonlinear oscillations in resistively shunted superconducting nanowires*. Thesis, Massachusetts Institute of Technology, 2017. Accepted: 2018-03-02T21:39:25Z.

[16] Emily Toomey, Qing-Yuan Zhao, Adam N. McCaughan, and Karl K. Berggren. Frequency pulling and mixing of relaxation oscillations in superconducting nanowires. *Phys. Rev. Appl.*, 9:064021, Jun 2018.

[17] Qing-Yuan Zhao, Di Zhu, Niccolò Calandri, Andrew E. Dane, Adam N. McCaughan, Francesco Bellei, Hao-Zhu Wang, Daniel F. Santavicca, and Karl K. Berggren. Single-photon imager based on a superconducting nanowire delay line. *Nature Photonics*, 11(4):247–251, April 2017.

[18] R. W. Klopfenstein. A transmission line taper of improved design. *Proceedings of the IRE*, 44(1):31–35, 1956.

[19] Di Zhu, Marco Colangelo, Boris A. Korzh, Qing-Yuan Zhao, Simone Frasca, Andrew E. Dane, Angel E. Velasco, Andrew D. Beyer, Jason P. Allmaras, Edward Ramirez, William J. Strickland, Daniel F. Santavicca, Matthew D. Shaw,

and Karl K. Berggren. Superconducting nanowire single-photon detector with integrated impedance-matching taper. *Applied Physics Letters*, 114(4):042601, January 2019. Publisher: American Institute of Physics.

[20] Labao Zhang, Xiachao Yan, Xiaoqing Jia, Jian Chen, Lin Kang, and Peiheng Wu. Maximizing switching current of superconductor nanowires via improved impedance matching. *Applied Physics Letters*, 110(7):072602, February 2017. Publisher: American Institute of Physics.

[21] Di Zhu, Marco Colangelo, Changchen Chen, Boris A. Korzh, Franco N. C. Wong, Matthew D. Shaw, and Karl K. Berggren. Photon-number resolution using superconducting tapered nanowire detector. In *2020 Conference on Lasers and Electro-Optics (CLEO)*, pages 1–2, 2020.

[22] Mohsen K. Akhlaghi, Haig Atikian, Amin Eftekharian, Marko Loncar, and A. Hamed Majedi. Reduced dark counts in optimized geometries for superconducting nanowire single photon detectors. *Opt. Express*, 20(21):23610–23616, Oct 2012.

[23] Joel K. W. Yang, Andrew J. Kerman, Eric A. Dauler, Vikas Anant, Kristine M. Rosfjord, and Karl K. Berggren. Modeling the Electrical and Thermal Response of Superconducting Nanowire Single-Photon Detectors. *IEEE Transactions on Applied Superconductivity*, 17(2):581–585, June 2007. Conference Name: IEEE Transactions on Applied Superconductivity.

[24] Reza Baghdadi, Jason P. Allmaras, Brenden A. Butters, Andrew E. Dane, Saleem Iqbal, Adam N. McCaughan, Emily A. Toomey, Qing-Yuan Zhao, Alexander G. Kozorezov, and Karl K. Berggren. Multilayered heater nanocryotron: A superconducting-nanowire-based thermal switch. *Phys. Rev. Appl.*, 14:054011, Nov 2020.

[25] Stephen Maas. *Nonlinear Microwave and RF Circuits, Second Edition*. 2003.

[26] Eric Keiter, Thomas Russo, Richard Schiek, Heidi Thornquist, Ting Mei, Jason Verley, Karthik Aadithya, and Joshua Schickling. Xyce parallel electronic simulator reference guide, version 7.6. 10 2022.

[27] Matteo Castellani. *Design of Superconducting Nanowire-Based Neurons and Synapses for Power-Efficient Spiking Neural Networks*. laurea, Politecnico di Torino, October 2020.

[28] L. W. Nagel and D. O. Pederson. Simulation Program with Integrated Circuit Emphasis (SPICE), April 1973.

[29] Michael Engelhardt. SPICE Differentiation.

[30] Nuno Brum. PyLTSpice, December 2022. original-date: 2018-04-23T20:02:26Z.

[31] O-device (Lossy Transmission Line) and T-device (Lossless Transmission Line) modelling issues - LTwiki-Wiki for LTspice.

[32] Roy McCammon. SPICE Simulation of Transmission Lines by the Telegrapher's Method, June 2010.

[33] E. Paladino, Y. M. Galperin, G. Falci, and B. L. Altshuler. $1/f$ noise: Implications for solid-state quantum information. *Rev. Mod. Phys.*, 86:361–418, Apr 2014.

[34] How to Make Independent "Gaussian" White Noise Sources, October 2014.

[35] MicroSim Application Notes. Technical report, MicroSim Corporation, June 1997.

[36] Measuring a Multiport Device with a 2-Port Network Analyzer — scikit-rf Documentation.

[37] A. Dounavis, R. Achar, and M. Nakhla. A general class of passive macromodels for lossy multiconductor transmission lines. *IEEE Transactions on Microwave Theory and Techniques*, 49(10):1686–1696, 2001.

[38] R. Neumayer, F. Haslinger, A. Stelzer, and R. Weigel. Synthesis of spice-compatible broadband electrical models from n-port scattering parameter data. In *2002 IEEE International Symposium on Electromagnetic Compatibility*, volume 1, pages 469–474 vol.1, 2002.

[39] Germund Dahlquist. Convergence and stability in the numerical integration of ordinary differential equations. *Mathematica Scandinavica*, 4(1):33–53, 1956.

[40] K.S. Kundert and I.H. Clifford. Achieving accurate results with a circuit simulator. In *IEE Colloquium on SPICE: Surviving Problems in Circuit Evaluation*, pages 4/1–4/5, 1993.

[41] B sources (complete reference) - LTwiki-Wiki for LTspice.

[42] Charles Thompson. A STUDY OF NUMERICAL INTEGRATION TECHNIQUES FOR USE IN THE COMPANION CIRCUIT METHOD OF TRANSIENT CIRCUIT ANALYSIS. *Department of Electrical and Computer Engineering Technical Reports*, April 1992.

[43] LoopVectorization, January 2023. original-date: 2019-01-14T05:55:52Z.

[44] Stephan Hilb. StaticKernels, January 2023. original-date: 2020-04-15T13:53:37Z.

[45] Christopher Rackauckas and Qing Nie. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5(1):15, May 2017. Number: 1 Publisher: Ubiquity Press.

[46] DifferentialEquations.jl: Scientific Machine Learning (SciML) Enabled Simulation and Estimation · DifferentialEquations.jl.

[47] ODE Solvers · DifferentialEquations.jl.

[48] Home · CUDA.jl.

[49] DiffEqGPU: Massively Data-Parallel GPU Solving of ODEs · DiffEqGPU.jl.

[50] Chapter 39. Parallel Prefix Sum (Scan) with CUDA.

[51] Tim Besard. juliacon21-gpu_workshop/CUDA.ipynb at main · maleadt/juliacon21-gpu_workshop.

[52] Ch Tsitouras. Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011.

[53] J. H. Verner. Numerically optimal Runge–Kutta pairs with interpolants. *Numerical Algorithms*, 53(2):383–396, March 2010.

[54] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II*, volume 14 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[55] Flux: The Julia Machine Learning Library.

[56] Patrick K Mogensen and Asbjørn N Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, April 2018.

[57] R. E. Collin. The optimum tapered transmission line matching section. *Proceedings of the IRE*, 44(4):539–548, 1956.

[58] FFTW.jl, January 2023. original-date: 2017-05-23T19:34:29Z.

[59] KLU, January 2023. original-date: 2021-11-25T20:04:05Z.

[60] M. Frigo and S.G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, February 2005.

[61] Timothy A. Davis and Ekanathan Palamadai Natarajan. Algorithm 907: Klu, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3), sep 2010.