Contents

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. 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. These effects range from 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 the we scale device sizes and introduce dependence on other factor, such as 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. As 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 is a continuous form of non-linearity introduced by the inertia of cooper pairs in a nanowire. In thin films, kinetic inductance is highly dependent on the film thickness and temperature [3]. A nanowire's iductance is almost entirely due to kinetic inductance, and therefore its reliance on the bias current is of importance [4]. 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 pulse – this is a well-studied effect in non-linear transmission lines [5]. 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. Non-linearities 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

timestep smaller until the magnitude of change is small. In boolean states, there is no sense of continuity, and in the limit of smaller timesteps, 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 non-linearities exhibited by nanowires.

A more complicated - but sometimes necessary - picture includes coupling to a thermal equation. A nanowires critical current, i_c, and critical temperature, T_c, are functions of the current state of the superconducto. These two parameters are related by the critical surface and has 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 [6, 7].

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 [8, 9]. This picture tends to enforce simulation constraints as the discretization and network size increase. This picture accounts for time delays introduced for a signal entering and leaving a nanowire, resonances that might occur in the nanowire, as well as distributed thermal and electrical effects that don't make sense in the single element lumped picture. For meanders longer than the wavelength of frequencies carried, modeling them as distributed devices is essential [8]. 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 incidence) 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 $1k\Omega$).

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 50Ω impedance shunted with a 50Ω resistor, the current is split and equally

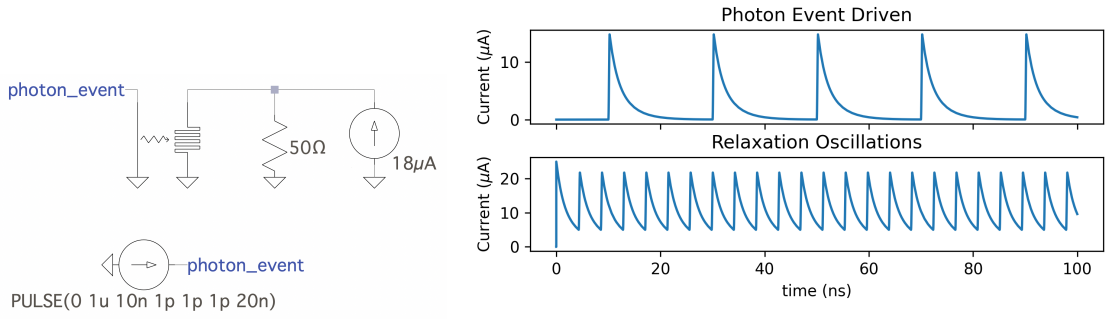


Figure 1: Caption

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 $\sim 7k\Omega$ 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 [10].

In this topology, the nanowire can produce high frequency relaxation oscillations when biased above the critical current [11]. 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

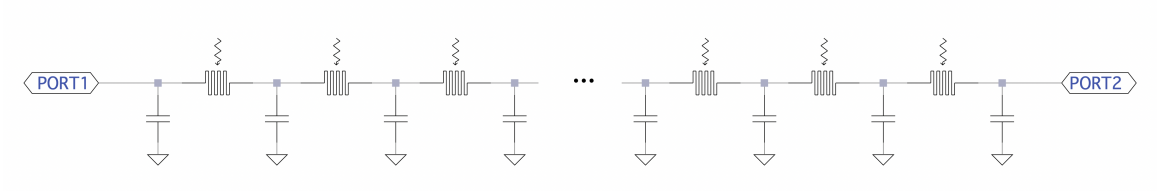


Figure 2: Caption

meanders [12]. 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 non-linearities 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 this image, this can be thought of as a long chain of discrete lumped nanowire elements in parallel with linear capacitor elements as shown in figure 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

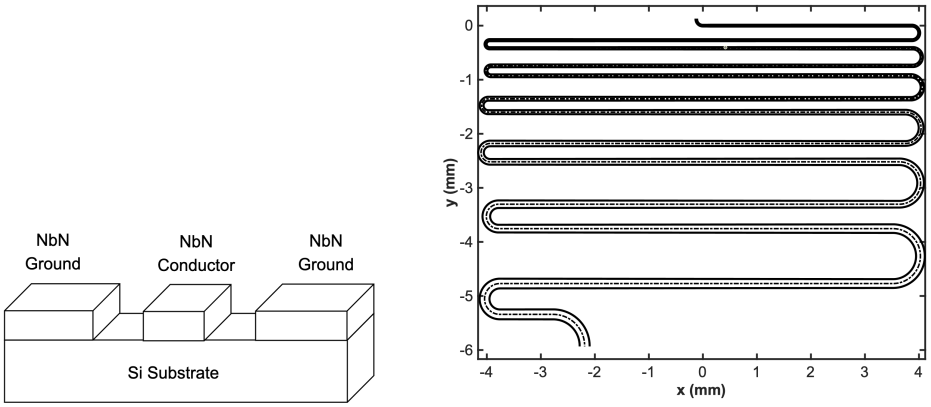
Usually, the nanowire's impedance is not similar enough to that of the input and output circuitry. This mismatch causes the signal to reflect back into the wire instead of propagating into the next stage, causing interference and distortions. Impedance

matching is done by designing tapers: extensions of the same line that increase in width slowly as shown in figure 3(b). 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 [13]. 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 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 use readout pulses in more interesting ways and help preserve logic pulses [12, 14, 15]. One example application is using impedance matching tapers to perform photon number resolution - infer the number of incident photons on a nanowire from the readout pulse as shown in more detail in section 2.5.1 [16] . Another application is in devices like SNSPIs, where tapers help preserve the fast rising edge that is essential for spatially resolving photons [12].

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 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 don't 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 taper meanders are folded into strips of straight wires connected by curved edges with large radii. This curvature is chosen in a manner that minimizes the



(a) (b)

Figure 3: (a) Vertical slice a Co-Planar Waveguide (CPW) geometry for a Niobium Nitride (NbN) nanowire. (b) Top view of a Klopfenstein taper with a folded meander.

total amount of reflection and current crowding: an unwanted effect caused by non-homogenous distributions of current density through a conductor that changes the frequency response and impedance of the wire [17]. 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 [18]. A CPW geometry nanowire, as shown in figure 3(a), consists of

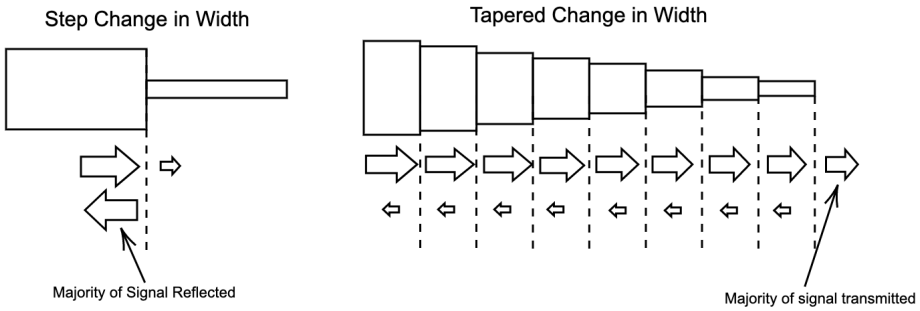


Figure 4: A tapered change in width causes smaller reflections at each boundary which add up to a lower total amount of reflection than a singular step change in width. Change in width is roughly proportional to change in impedance. The inclusion of multiple step changes in width increases the number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

multiple electrically and thermally coupled layers.

While this coupling can be unwanted, devices, such as the heater-tron (hTron), take advantage of that coupling [19]. 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 modelled as interactions between electrons and phonons [19]. Each stacks' eletron system can be heated through Joule dissipation. The electron system is coupled to the phonon systems which have interact on the layer's boundaries.

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 simualte the electro-thermal coupling of a nanowire with respect to the stack, noise and photons as well as the distributed effect accounting for non-linearities 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 Kermal et al [7, 6]. 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 [7]. 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 [5]. 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 [20]. 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 JTWPA topologies in the frequency domain using Harmonic Balance [21]. WRspice and Xyce both have Harmonic Balance backends that are very efficient [22, 23]. 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 implies that the entire circuit must be simulated in the time-domain. This is addressed in section 4.2.

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 [6, 24]. 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 [24]. These models are sufficient but don't encompass a full thermal image 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 proposes having an integrated simulator environment designed with the goal of simulating superconducting nanowires. Existing simulators exist for simulating superconducting electronics, but tend to favor the frequency domain and can only account for the electrical non-linearities exhibited by superconductors [21, 22]. 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 [23]. The work presented in this thesis will be divided into 3 sections tackling:

1. an integrated environment for LTspice to design specific models and tools for

superconducting devices and accompanying experiments.

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

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

2 spice-daemon — a Python wrapper for SPICE solvers

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 [25].

Since then, the original Berkley SPICE inspired multiple other SPICE solvers including LTspice, a popular free circuit simulator [26]. 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 dy-

namic timestepping fitted to a low-order polynomial [27].

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 [28]. 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).

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 Installating spice-daemon and qnn-spice

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 YAML file – a human-readable data- serialization language – 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 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 YAML file 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 repos group elements together and you can split a repo to 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 directive that takes in the url for the remote lib. 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 environments 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 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 who's 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 runs 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.

2.5 Dynamic Models

LTspice components are parametrizable 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 [29]. 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 [30]. 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) [27]. As a result, you shouldn't use transmission line concatenation to model tapers as the timestepping algorithm will take impractically long to simulate.

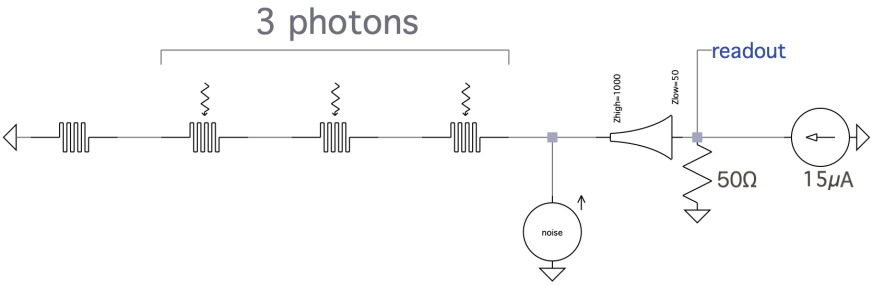
For well-defined convergence with non-convolution based models, we need to be able to simulate repeating lumped element models from within LTspice. However, this would involve laying down thousands of repeating chunks of elements manually. One use of dynamic models is generating a model that encodes 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

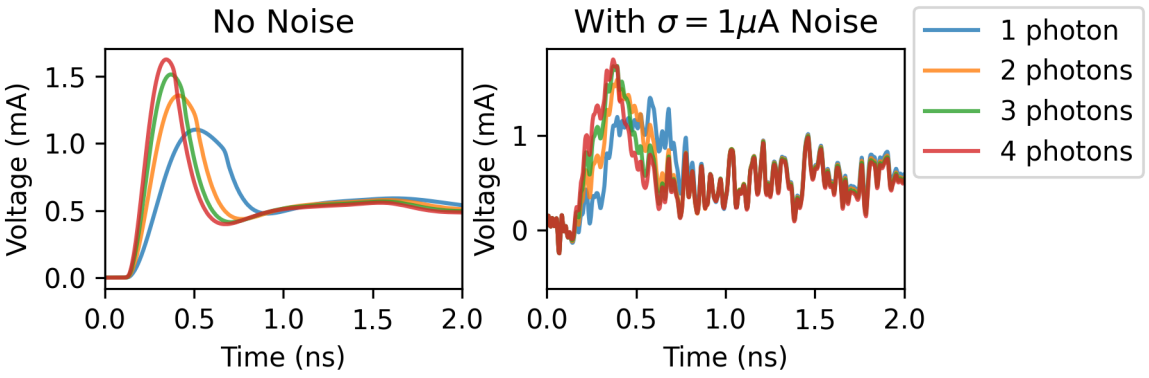
by the T and O models, but also gives us the ability to simulate more complicated transmission lines. For instance, inductors on a transmission line can be non-linear, making simulating a superconducting transmission line more accurate. Other possibilities that aren't possible in the LTspice environment 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 5 showcases the use of a spice-daemon generated taper element to perform Photon Number Resolution on a nanowire strip. Figure 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



(a)



(b)

Figure 5: Caption

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 5(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 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 [31]. 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 severly 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 [32]. 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 5.

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 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.

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 $= 1000$,

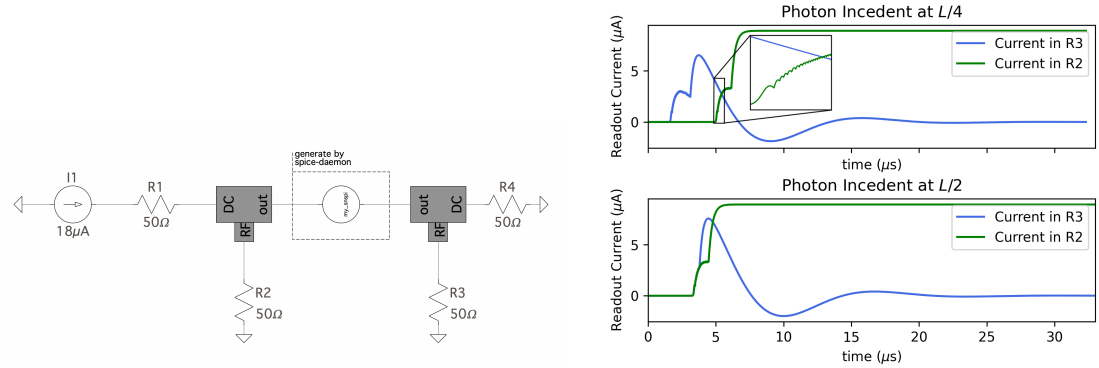


Figure 6: Caption

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 integrtor 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 18 and the dependency graph collapses to only one node in figure 19).

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 .

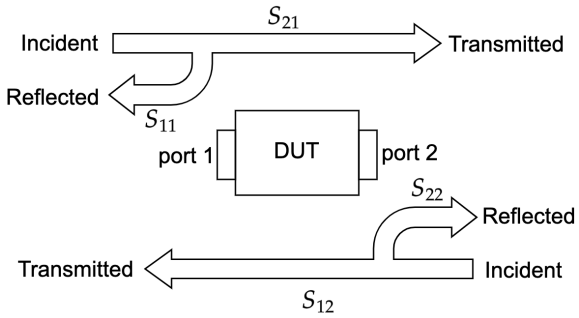


Figure 7: Illustration of scattering parameters showcasing the transmitted (S_{21} and S_{12}) and reflected (S_{11} and S_{22}) signals of a DUT.

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 generatlize 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 behavior of the device [33]. An accompanying spice-daemon module (spice-arbnport) can generate a model with this topology automatically given the scattering parameters measured for a device.

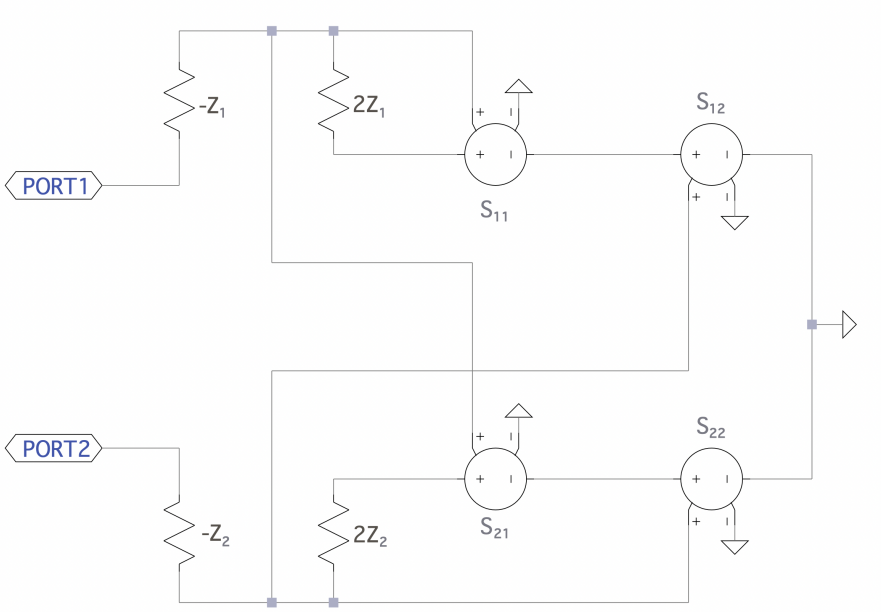


Figure 8: Subcircuit for a 2-port model generated by S-parameters. **TODO: I can't decide whether this should work for non-passive devices. I think it should?**

2.6.1 2-port model

We can model devices with arbitrary frequency-dependent scattering parameters using the topology presented in figure 8 [33]. 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 8 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).

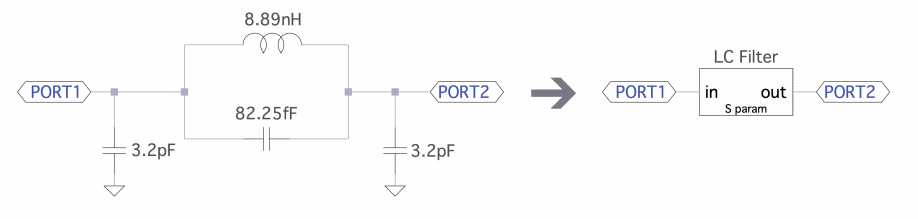
One limitation corresponds to the number of data points, as it increases, the SPICE gets hung up on generating logic lines. LTspice doesn't document the existence or process of creating logic lines, however, reverse engineering the binary indicates

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.

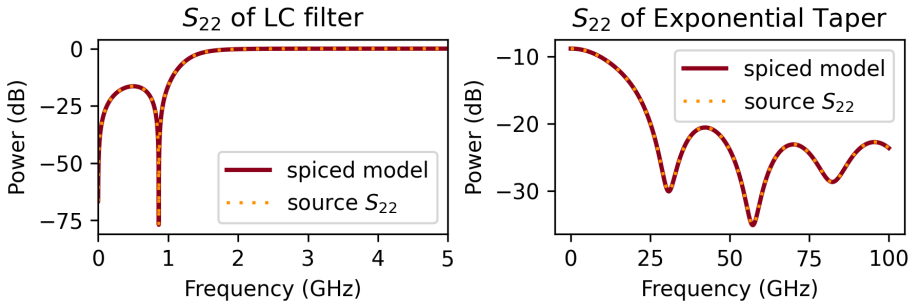
The complexity of simulation in the time and frequency domain for LTspice is minimal – the freq interpolation backend is similar to that used by the table directive and PWL source mode. If the logic line creation time is avoided, compilation time is also small, making this method much more practical 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 9(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 9(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 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 10 the agreement in transient analysis by simulating the spice-daemon model vs. the actual LC filter's netlsit in LTspice. We test the response of the LC filter and the spice-daemon generated model against various sine waves



(a)



(b)

Figure 9: (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.

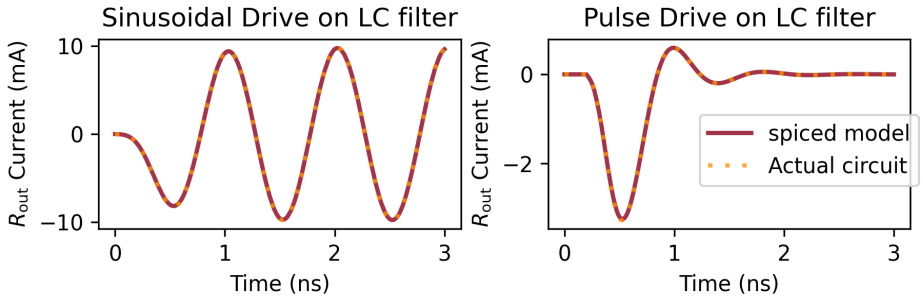


Figure 10: Caption

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 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 11. This scales as n^2 with the 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 $\frac{n(n-1)}{2}$

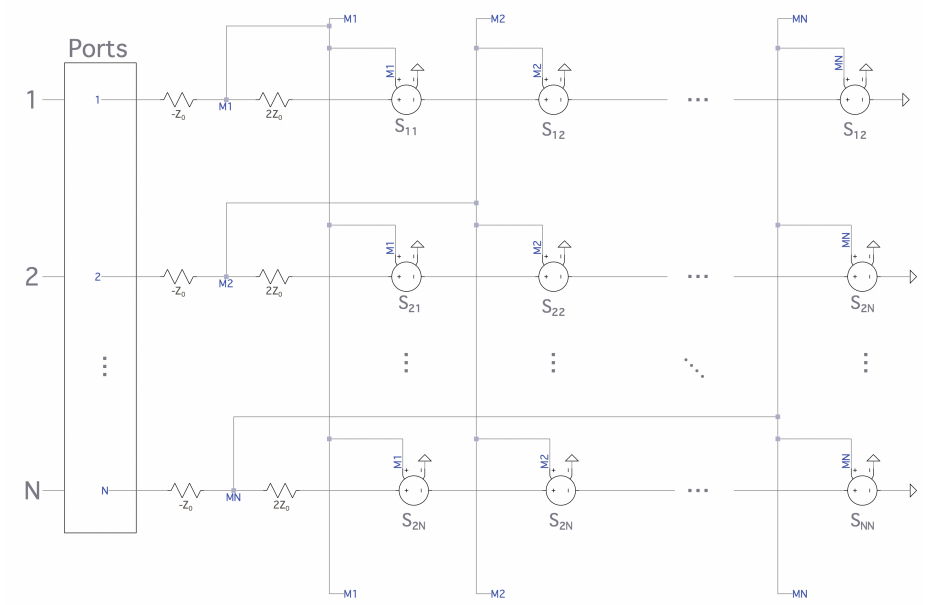


Figure 11: Subcircuit for an n-port model generated by S-parameters.

sweeps are needed to fully characterize a device [34].

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 11) is composed of order independent behavioural sources that can be merged to one 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 dependences 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.

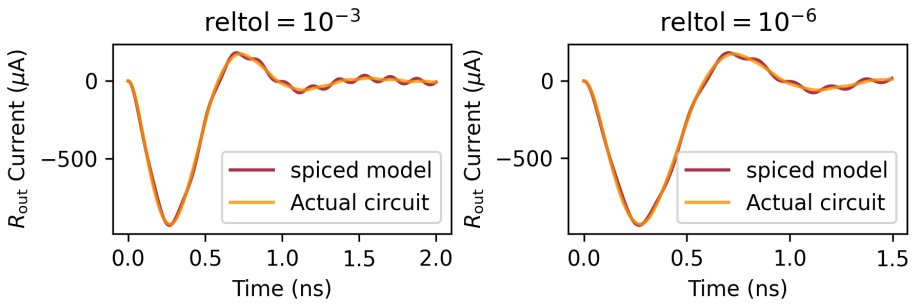


Figure 12: Caption

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 [35]. The spice-daemon source can be modified to realize a corresponding minimal order state-space circuit from a fitted version of the transfer function [36]. 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 12 showcases incorrect oscilations 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/frequecy 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 correpsonds 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 extraplated 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 spicedaemon 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. spicedaemon 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 default 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

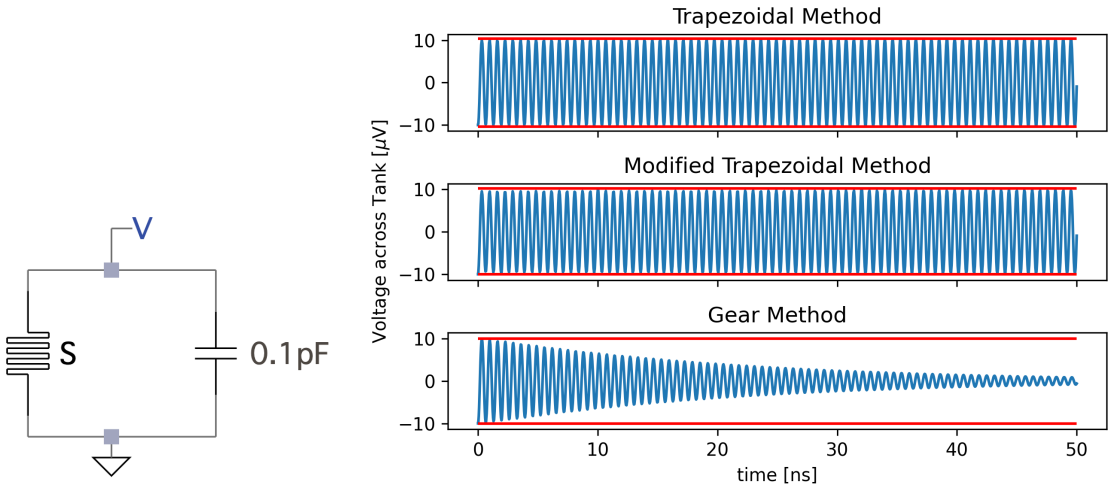
can invoke what the next parameter to search is. For instance, you can implement gradient descent in LTspice using post-processing toolkits since they have the ability to hook to modules. Given an arbitrary nanowire circuit, you might want to find the maximum bias that doesn't switch the wire (accounting 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.

3 Model Stability

3.1 Integration in SPICE

SPICE implementations in general rely on second-order integration, namely trapezoidal and Gear integration, each with their own benefits. LTspice allows the use 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 [26, 27].

Trapezoidal integration is more accurate and faster than the other options but introduces a ringing numerical artifact that occurs on adjacent timesteps on stiff systems. Ringing is dampened by the gear method which means this numerical artifact is mostly eliminated, however, the gear method dampens most ringing. We typically care about oscillatory behavior in nanowires that can be filtered by the gear method 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



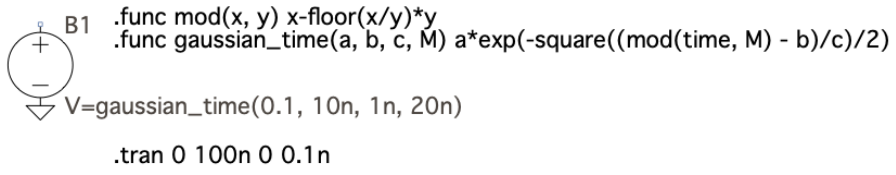
(a) (b)

Figure 13: No caption yet :(

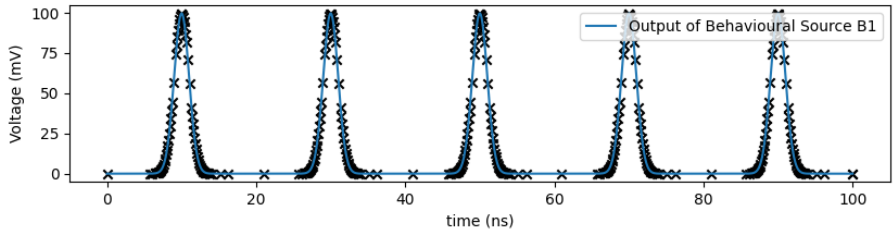
numerical dampening [26].

3.1.1 Dependent 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 are powerful (they can mimic the behavior of any other source) and can have multiple inputs as opposed to the other sources' dependence on one value only. B-sources also allow for the use of arbitrary maths 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



(a)



(b)

Figure 14: Example of a behavioral source outputting gaussian pulses.

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 14(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 14(b).

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 [37]. One result of this for non-linear systems, such as the nanowire model, is their solution should

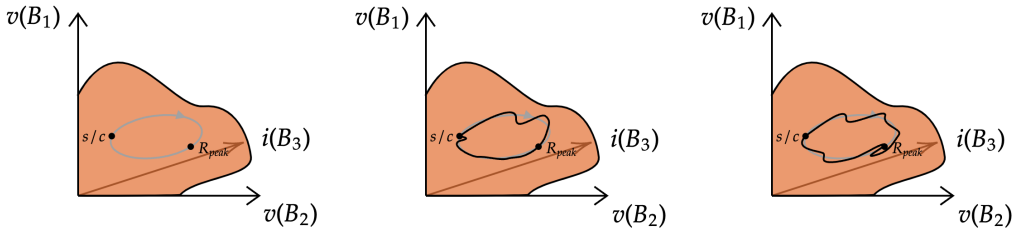


Figure 15: Caption

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 [27].

One way of visualizing solving a continuous system using a finite element method is via 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 solution trajectory but are corrected for. 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

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 transition 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 continuous nonlinearity exhibited in nanowires is important for all sorts of superconducting topologies including Josephson traveling-wave parametric amplifier. As the network size increases

The nanowire model is unstable and inconsistent over both types of non-linearities 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

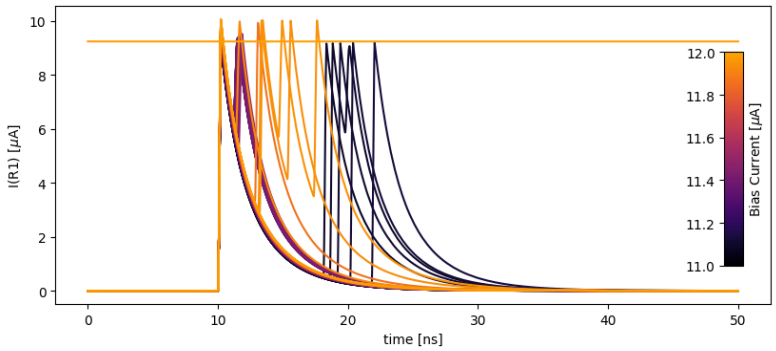


Figure 16: SNSPD readout circuit tested against 100 different bias values between 11μA and 12μA for a device with switching current 12μA. The simulation was carried out using the existing nanowire model and the default options for LTspice on Mac (reltol of 10^{-3}, voltol of 10^{-6}, trtol of 2). We 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.

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

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 [38]:

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

This imposes a condition on the node voltage v_i change 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.

By varying the relative tolerance and testing different geometries, the value 10^{-6} seems to perform the best in terms of stability and accuracy.

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 [37]. 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.

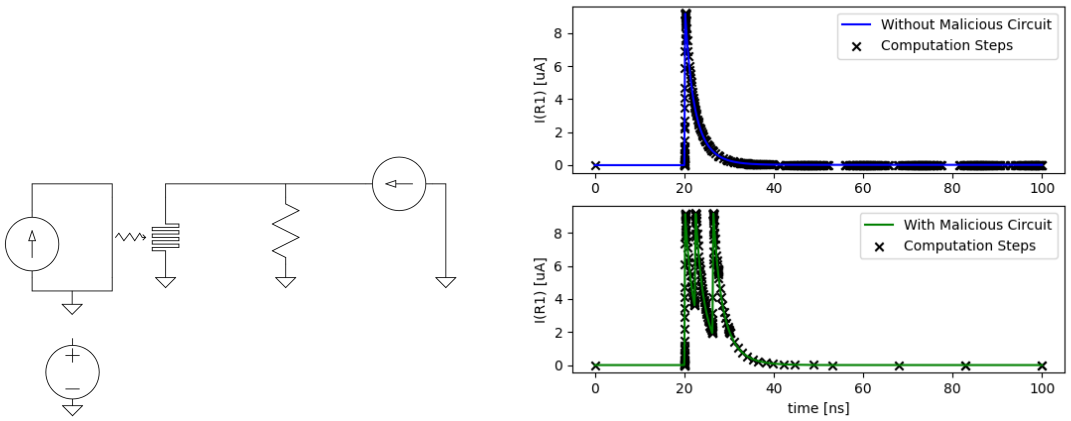
Unfortunately, we can't perform this enumeration by varying timesteps in LTspice

trivially. One workaround however, is simulating a circuit $C' = C + M$ that contains the original circuit C and a new malicious circuit M. M doesn't have to be coupled to C topologically (M and C share no nodes, currents or behavioural coupling). However, C and M are now time coupled, in that if M exhibits a strong non-linearity 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). This forces the simulator to use finer steps near the ppulse 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 behavioural source whos output is 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 in SPICE creates breakpoints that force the timestepping resolution for the entire simulation to be below half the total line delay [39, 27]. 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 17(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 get see how the inclusion of a malicious circuit can affect the output result in figure 17(b). Note that the resulting solution is not possible given the typical geometry, even though it looks plausible, since the period of oscillations



(a) (b)

Figure 17: (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.

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 16. 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 behavioural 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

3.4.1 Current nanowire model

Berggren et al.'s nanowire model comprises of four sub-circuits outlined in figure 18. 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

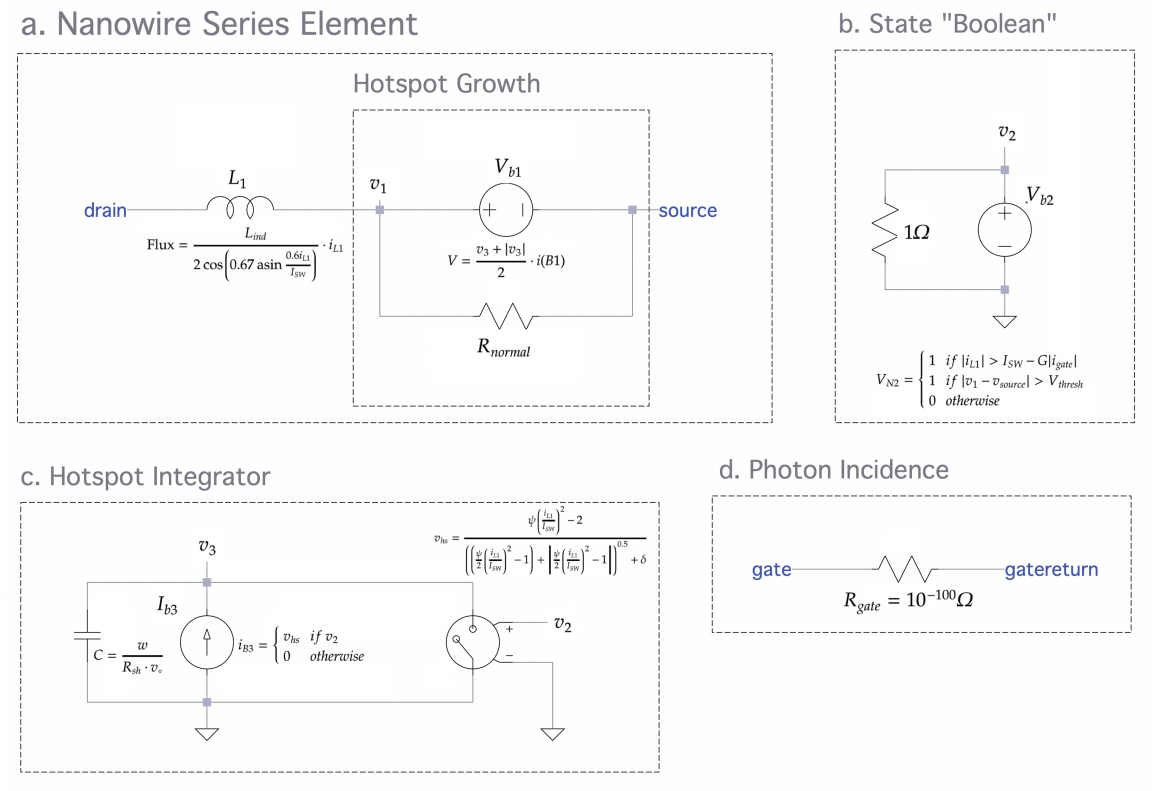


Figure 18: Diagram showcasing the four subcircuits used in the Berggren et al. SPICE dynamic nanowire model. Subcircuit a accounts for the kinetic inductance continuous non-linearity and the resistance in the normal state. Subcircuit b tracks a boolean state of superconducting vs. normal. Subcircuit c integrates the hotspot velocity as per the phenomenological hotspot model. Subcircuit d is the photon inlet.

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 19. 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

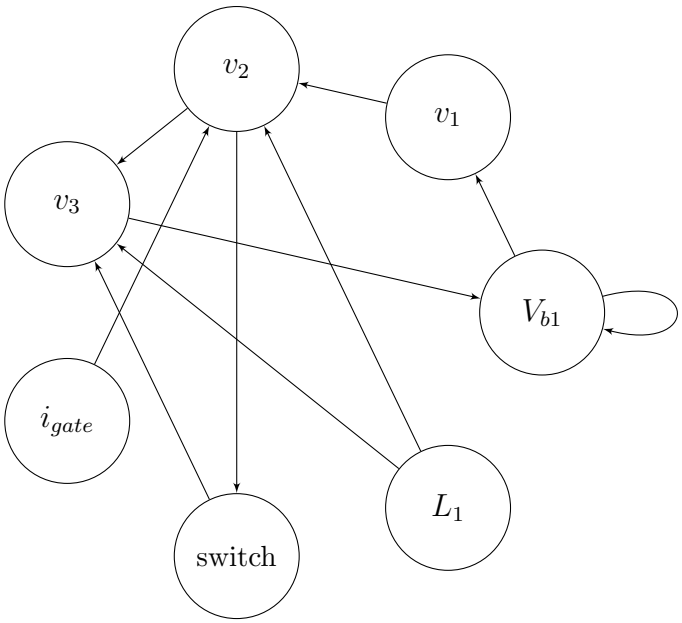


Figure 19: A dependency graph for some elements and nodes of the nanowire model as presented in figure 18. Each node in the graph represents a circuit node value that is calculated or an element parameter. Since the compiler is a black box, some degree and outdegree 1 nodes were removed. Note that the 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.

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

3.4.2 Stability of the original nanowire model

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 and we replaced it with a behavioral source that integrates the same hotspot. 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 $x > 0$.

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 not only involves creating fewer projections onto the circuit (leading to less 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 handles 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 18. 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 1mΩ and 10GΩ. This switching behaviour 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

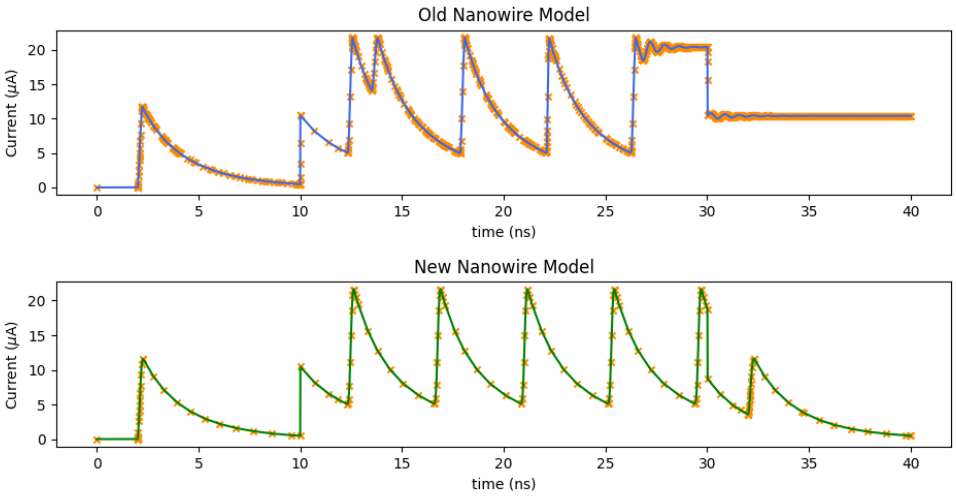


Figure 20: Caption

a lower reltol leads to a large instability that should be avoided.

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 20). 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, however, when switching, more data points are used up.

Figure 21 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 20, where the current output seems correct. However, looking at the hotspot resistance by scoping the internal variables of the model, we see that the hotspot resistance has huge spikes, up

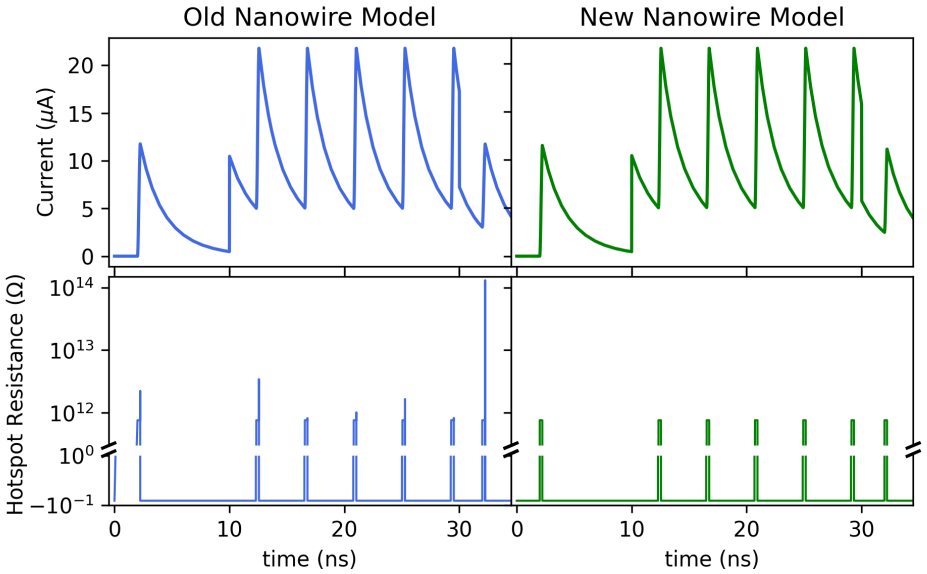


Figure 21: Caption

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 ($R_{\text{hotspot}} = \frac{v_1 - v_{\text{source}}}{i}$).

3.4.4 1 Element Models

3.4.5 0 Resistance Models

4 Efficient Simulation

4.1 Tline Model

4.1.1 Equivalent Circuit

4.1.2 Kernel

4.1.3 GPU

4.2 Harmonic Balance

4.2.1 TD Assist

4.3 Device Symmetries

4.4 Coupling Diff. Eq. (or Thermal Model?)

Maybe sections should be separate. Phonon-Electron transport.

Complex electric coupling

4.4.1 TDC

4.4.2 SNSPI coupling

4.5 Precomputation

4.6 ML Optimization

4.6.1 Symbolic Solver

4.6.2 Tapers

Note on resistive groups? -¿ Sonnet?

4.6.3 Differentiable Simulator

4.6.4 Inverse design

4.6.5 Monte Carlo Simulation

References

[1] G. N. Gol'tsman et al. "Picosecond superconducting single-photon optical detector". In: *Applied Physics Letters* 79.6 (Aug. 2001). Publisher: American Institute of Physics, pp. 705–707. ISSN: 0003-6951. DOI: 10.1063/1.1388868. URL: https://aip.scitation.org/doi/10.1063/1.1388868 (visited on 12/31/2022).

[2] Emily Toomey, Ken Segall, and Karl K. Berggren. "Design of a Power Efficient Artificial Neuron Using Superconducting Nanowires". In: *Frontiers in Neuroscience* 13 (2019). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2019.00933 (visited on 12/31/2022).

[3] Di Zhu. "Microwave engineering in superconducting nanowires for single-photon detection". eng. Accepted: 2020-03-09T18:59:00Z. Thesis. Massachusetts Institute of Technology, 2019. URL: https://dspace.mit.edu/handle/1721.1/124123 (visited on 12/14/2022).

[4] Andrew J. Kerman et al. "Kinetic-inductance-limited reset time of superconducting nanowire photon counters". In: *Applied Physics Letters* 88.11 (Mar. 2006). Publisher: American Institute of Physics, p. 111116. ISSN: 0003-6951. DOI: 10.1063/1.2183810. URL: https://aip.scitation.org/doi/abs/10.1063/1.2183810 (visited on 12/31/2022).

[5] E. Afshari and A. Hajimiri. "Nonlinear transmission lines for pulse shaping in silicon". In: *IEEE Journal of Solid-State Circuits* 40.3 (2005), pp. 744–752. DOI: 10.1109/JSSC.2005.843639.

[6] Andrew J. Kerman et al. “Electrothermal feedback in superconducting nanowire single-photon detectors”. In: *Phys. Rev. B* 79 (10 Mar. 2009), p. 100509. DOI: 10.1103/PhysRevB.79.100509. URL: https://link.aps.org/doi/10.1103/PhysRevB.79.100509.

[7] Karl K Berggren et al. “A superconducting nanowire can be modeled by using SPICE”. en. In: *Superconductor Science and Technology* 31.5 (May 2018), p. 055010. ISSN: 0953-2048, 1361-6668. DOI: 10.1088/1361-6668/aab149. URL: https://iopscience.iop.org/article/10.1088/1361-6668/aab149 (visited on 07/29/2022).

[8] Qing-Yuan Zhao et al. “A distributed electrical model for superconducting nanowire single photon detectors”. In: *Applied Physics Letters* 113.8 (Aug. 2018). Publisher: American Institute of Physics, p. 082601. ISSN: 0003-6951. DOI: 10.1063/1.5040150. URL: https://aip.scitation.org/doi/10.1063/1.5040150 (visited on 12/30/2022).

[9] Daniel F. Santavicca et al. “Microwave dynamics of high aspect ratio superconducting nanowires studied using self-resonance”. In: *Journal of Applied Physics* 119.23 (June 2016). Publisher: American Institute of Physics, p. 234302. ISSN: 0021-8979. DOI: 10.1063/1.4954068. URL: https://aip.scitation.org/doi/abs/10.1063/1.4954068 (visited on 12/30/2022).

[10] Emily Toomey. “Microwave response of nonlinear oscillations in resistively shunted superconducting nanowires”. eng. Accepted: 2018-03-02T21:39:25Z. Thesis. Massachusetts Institute of Technology, 2017. URL: https://dspace.mit.edu/handle/1721.1/113924 (visited on 12/31/2022).

[11] Emily Toomey et al. “Frequency Pulling and Mixing of Relaxation Oscillations in Superconducting Nanowires”. In: *Phys. Rev. Appl.* 9 (6 June 2018), p. 064021. DOI: 10.1103/PhysRevApplied.9.064021. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.9.064021.

[12] Qing-Yuan Zhao et al. “Single-photon imager based on a superconducting nanowire delay line”. In: *Nature Photonics* 11.4 (Apr. 2017), pp. 247–251. ISSN: 1749-4893. DOI: 10.1038/nphoton.2017.35. URL: https://doi.org/10.1038/nphoton.2017.35.

[13] R. W. Klopfenstein. “A Transmission Line Taper of Improved Design”. In: *Proceedings of the IRE* 44.1 (1956), pp. 31–35. DOI: 10.1109/JRPROC.1956.274847.

[14] Di Zhu et al. “Superconducting nanowire single-photon detector with integrated impedance-matching taper”. In: *Applied Physics Letters* 114.4 (Jan. 2019). Publisher: American Institute of Physics, p. 042601. ISSN: 0003-6951. DOI: 10.1063/1.5080721. URL: https://aip.scitation.org/doi/full/10.1063/1.5080721 (visited on 08/12/2022).

[15] Labao Zhang et al. “Maximizing switching current of superconductor nanowires via improved impedance matching”. In: *Applied Physics Letters* 110.7 (Feb. 2017). Publisher: American Institute of Physics, p. 072602. ISSN: 0003-6951. DOI: 10.1063/1.4976705. URL: https://aip.scitation.org/doi/full/10.1063/1.4976705 (visited on 12/30/2022).

[16] Di Zhu et al. “Photon-Number Resolution using Superconducting Tapered Nanowire Detector”. In: *2020 Conference on Lasers and Electro-Optics (CLEO)*. 2020, pp. 1–2.

[17] Mohsen K. Akhlaghi et al. “Reduced dark counts in optimized geometries for superconducting nanowire single photon detectors”. In: *Opt. Express* 20.21 (Oct. 2012), pp. 23610–23616. DOI: 10.1364/OE.20.023610. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-20-21-23610.

[18] Joel K. W. Yang et al. “Modeling the Electrical and Thermal Response of Superconducting Nanowire Single-Photon Detectors”. In: *IEEE Transactions on Applied Superconductivity* 17.2 (June 2007). Conference Name: IEEE Transactions on Applied Superconductivity, pp. 581–585. ISSN: 1558-2515. DOI: 10.1109/TASC.2007.898660.

[19] Reza Baghdadi et al. “Multilayered Heater Nanocryotron: A Superconducting-Nanowire-Based Thermal Switch”. In: *Phys. Rev. Appl.* 14 (5 Nov. 2020), p. 054011. DOI: 10.1103/PhysRevApplied.14.054011. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.14.054011.

[20] Stephen Maas. 2003.

[21] K. P. O’Brien and K. Peng. *JosephsonCircuits.jl*. 2022. URL: https://github.com/kpobrien/JosephsonCircuits.jl (visited on 12/29/2022).

[22] S.R. Whiteley. “Josephson junctions in SPICE3”. In: *IEEE Transactions on Magnetics* 27.2 (1991), pp. 2902–2905. DOI: 10.1109/20.133816.

[23] Eric Keiter et al. “Xyce Parallel Electronic Simulator Reference Guide, Version 7.6.” In: (Oct. 2022). DOI: 10.2172/1895028. URL: https://www.osti.gov/biblio/1895028.

[24] Matteo Castellani. “Design of Superconducting Nanowire-Based Neurons and Synapses for Power-Efficient Spiking Neural Networks”. it. laurea. Politecnico

di Torino, Oct. 2020. URL: https://webthesis.biblio.polito.it/15926/ (visited on 12/30/2022).

[25] L. W. Nagel and D. O. Pederson. *Simulation Program with Integrated Circuit Emphasis (SPICE)*. Ontario, Canada, Apr. 1973. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/ERL-382.pdf (visited on 12/30/2022).

[26] Michael Engelhardt. *SPICE Differentiation*. URL: https://www.analog.com/en/technical-articles/spice-differentiation.html (visited on 12/30/2022).

[27] Kenneth Kundert. *The Designer's Guide to Spice and Spectre*. en. The Designer's Guide Book Series. Boston: Kluwer Academic Publishers, 2003. ISBN: 978-0-7923-9571-3. DOI: 10.1007/b101824. URL: http://link.springer.com/10.1007/b101824 (visited on 12/30/2022).

[28] Nuno Brum. *PyLTSpice*. original-date: 2018-04-23T20:02:26Z. Dec. 2022. URL: https://github.com/nunobrum/PyLTSpice (visited on 01/01/2023).

[29] *O-device (Lossy Transmission Line) and T-device (Lossless Transmission Line) modelling issues - LTwiki-Wiki for LTspice*. URL: https://ltwiki.org/index.php?title=O-device_(Lossy_Transmission_Line)_and_T-device_(Lossless_Transmission_Line)_modelling_issues (visited on 01/01/2023).

[30] Roy McCammon. *SPICE Simulation of Transmission Lines by the Telegrapher's Method*. June 2010. URL: https://i.cmpnet.com/rfdesignline/2010/06/C0580Pt1edited.pdf (visited on 12/30/2022).

[31] E. Paladino et al. "$1/f$ noise: Implications for solid-state quantum information". In: *Rev. Mod. Phys.* 86 (2 Apr. 2014), pp. 361–418. DOI: 10.1103/RevModPhys.86.361. URL: https://link.aps.org/doi/10.1103/RevModPhys.86.361.

[32] *How to Make Independent "Gaussian" White Noise Sources*. Oct. 2014. URL: https://groups.io/g/LTspice/topic/how_to_make_independent/50203924?p= (visited on 12/30/2022).

[33] *MicroSim Application Notes*. en. Tech. rep. MicroSim Corporation, June 1997. URL: http://www.it.uom.gr/project/digital/appnts.pdf (visited on 01/01/2023).

[34] *Measuring a Multiport Device with a 2-Port Network Analyzer — scikit-rf Documentation*. URL: https://scikit-rf.readthedocs.io/en/latest/examples/metrology/Measuring%20a%20Mutiport%20Device%20with%20a%202-Port%20Network%20Analyzer.html (visited on 01/02/2023).

[35] A. Dounavis, R. Achar, and M. Nakhla. "A general class of passive macromodels for lossy multiconductor transmission lines". In: *IEEE Transactions on Microwave Theory and Techniques* 49.10 (2001), pp. 1686–1696. DOI: 10.1109/22.954772.

[36] R. Neumayer et al. "Synthesis of SPICE-compatible broadband electrical models from n-port scattering parameter data". In: *2002 IEEE International Symposium on Electromagnetic Compatibility*. Vol. 1. 2002, 469–474 vol.1. DOI: 10.1109/ISEMC.2002.1032524.

[37] Germund Dahlquist. "Convergence and Stability in the Numerical Integration of Ordinary Differential Equations". In: *Mathematica Scandinavica* 4.1 (1956),

pp. 33–53. ISSN: 00255521, 19031807. URL: http://www.jstor.org/stable/24490010 (visited on 12/30/2022).

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

[39] *Star-Hspice Manual*. July 1998. URL: https://class.ece.uw.edu/cadta/hspice/ (visited on 12/30/2022).