SuperMix: A Flexible Software Library for High-Frequency Circuit Simulation, Including SIS Mixers and Superconducting Elements

John Ward, Frank Rice, Goutam Chattopadhyay, and Jonas Zmuidzinas

California Institute of Technology, 320-47, Pasadena, CA 91125, USA.

Abstract

"SuperMix" is a software library written to aid in the calculation and optimization of the signal and noise performance of high-frequency circuits, especially those including superconductors and superconducting tunnel junctions. Using this library, C++ programs can be written to simulate circuits of arbitrary size, complexity, and topology. The library includes an optimizer which can minimize an arbitrary error function by varying chosen circuit parameters. Using SuperMix, it is now possible to directly optimize an SIS receiver design to minimize the average noise temperature across the desired RF frequency band, including a full harmonic balance analysis at every frequency point together with a full analysis of the IF amplifier circuit and its noise contribution.

I. INTRODUCTION

A. Motivation: designing and simulating SIS receivers

Although there are many excellent software packages available to aid in the design of microwave circuits, none provide the specialized elements needed for the design of complete superconducting tunnel junction (SIS) receivers. For instance, thin film microstrip lines, which are widely used in SIS mixers for impedance matching circuits, have characteristics which depend on the surface impedance of the normal or superconducting metal films. Surface impedance calculations usually involve nontrivial numerical computations, such as numerical integration or the solution of integral equations [1], [2], [3], and are not available in commercial microwave software packages. Furthermore, the calculation of the signal and noise properties of SIS mixers requires the use of Tucker's theory [4] combined with a nonlinear harmonic–balance calculation [5], [6] of the local oscillator waveform. Again, the required calculations are numerically intensive and are not available in commercial packages.

It is clear that a complete simulation of an SIS mixer is a substantial computational task. Because of this, SIS mixer design is usually performed using simplifying approximations. For instance, the RF circuit is usually designed and optimized by treating the problem as an impedance matching exercise. Once a circuit is designed in this way, the embedding admittances it presents to the SIS junction can be calculated, and these can be used in a Tucker theory calculation to determine the conversion loss and noise temperature of the mixer. The results of such a calculation could then be imported into a standard microwave program to explore the effect of the HEMT IF amplifier and its matching circuit. This was the approach followed by Padin et al. [7] for the design of an integrated HEMT IF amplifier for an SIS mixer.

However, to our knowledge, no one has ever designed an SIS receiver by directly calculating the noise temperature across some desired RF band, including the harmonic effects and the IF amplifier contribution, and optimizing this total receiver noise temperature as a function of the circuit parameters. Even though this sounds like the most obvious and straightforward approach to the design, the software to do this has simply been unavailable until now. Furthermore, with the exception of a few special cases involving two junction circuits [8], [9], parallel arrays [10], and distributed junctions [11], most of the Tucker–theory programs that have been written for SIS design assume that only a single SIS junction will be used. However, now that junction fabrication processes are reliable and reproducible, there is a clear trend toward more complex mixer designs incorporating multiple junctions.

B. Alternative approaches

There are several possible solutions to our problem. Some commercial circuit simulators allow user–defined elements to be written using an interpreted "scripting" language. However, we felt that this approach would not provide the necessary level of computational performance needed for simulating and optimizing superconducting mixers. Indeed, the Tucker theory SIS mixer programs we are aware of are written in standard compiled programming languages such as FORTRAN. Alternatively, we could have attempted to work with a software vendor to have our routines for SIS mixer simulation incorporated directly into a commercial microwave package. However, our judgement was that few vendors would be interested given the very limited market. Furthermore, given that we are continually developing new materials and mixer designs, we felt that it would be extremely important to be able to rapidly add new elements and models to the code. Finally, we wanted to have software that would run on a large variety of platforms, and not be plagued by licensing restrictions. For these reasons we decided to develop our own stand–alone code.

C. SuperMix: a C++ class library

We have developed a C++ library, which we call "SuperMix", to provide a flexible tool with which to design and optimize receiver circuits, and microwave circuits in general.

SuperMix calculates circuits in the familiar wave representation, in which the behavior of a linear circuit is specified by a scattering matrix, a noise wave correlation matrix, and an outgoing source wave amplitude vector. The source wave vector represents the effects of generators inside the circuit, such as a local oscillator. Easy-to-use functions can convert the wave representation to other quantities, such as impedance or admittance matrices, gain, or noise temperature.

The C++ programming language [12] was chosen for this project to provide flexibility, ease of use, and a means to design the library with a modular approach. For this project, complex numbers, vectors, and matrices are used extensively. Although these data types are not included as part of standard C++, the language can be extended to include them in a natural and powerful way. For instance, the standard arithmetic

Basic Elements	Transmission Lines	Transistors	Miscellaneous Elements
Resistor	Superconducting Film	Transconductance	Mixer
Capacitor	Normal Metal Film	FET	SIS junction
Inductor	Layered Film	Fujitsu FHR02X	Transformer
Attenuator	Dielectric	Fujitsu FHX13X	180° Hybrid
Terminator	Microstrip	Kukje HEMT	90° Hybrid
	Transmission Line		Circulator
	Radial Stub		Voltage Source
	Time Delay		Current Sink
			Generator

TABLE I

CIRCUIT ELEMENTS CURRENTLY PROVIDED BY THE LIBRARY.

operations with complex numbers can be defined, as can the operation of multiplying a vector by a matrix.

As an object-oriented language, C++ provides a natural way to write modular programs. The SuperMix library was written from the ground up to be highly modular, flexible, and expandable. Circuits and circuit elements are implemented as C++ class objects. The modular approach used allows new elements to be added to the library by writing only a few lines of code (to calculate the scattering and noise matrices) without having to deal with details about how the overall library works. The modular design of the optimizer provides a straightforward means to add new optimization algorithms if needed. Almost every part of the library benefits from the flexibility provided by the C++ programming language.

Each circuit element in SuperMix inherits the properties common to all devices. This way, each element added to the library automatically includes a standard interface, for instance, a function to return scattering parameters of the circuit element. Lossy elements which generate noise can be assigned arbitrary temperatures. A list of circuit elements currently available in the library is given in table I.

II. THE WAVE REPRESENTATION

Internally, circuits are represented using a "wave" representation based on the scattering matrix. The incoming and outgoing wave amplitudes at some port i of a linear circuit are defined in terms of the voltage V_i across the port terminals and the current I_i flowing *into* the positive terminal by the expressions

$$a_i = \frac{1}{2\sqrt{Z_0}}(V_i + Z_0 I_i) \quad \text{and} \quad b_i = \frac{1}{2\sqrt{Z_0}}(V_i - Z_0 I_i)$$

where Z_0 is some standard normalizing impedance. These definitions are motivated by transmission line theory. Note that the normalization is chosen such that power can be calculated simply by squaring the wave amplitudes. For instance, the incoming

power at port i is $P_{incident} = |a_i|^2$ and the reflected power is $P_{reflected} = |b_i|^2$. Extensive discussions of the scattering matrix and its use in circuit calculations can be found in the references [13], [14]. [15].

SuperMix uses three quantities to represent a circuit: the scattering matrix S. the noise wave correlation matrix C_S. and the wave source vector $\mathbf{b_s}$. The definition of these quantities comes from the following expression. in which the outgoing wave amplitudes $\mathbf{b}$ are considered to be dependent upon the incoming wave amplitudes $\mathbf{a}$:

$$\mathbf{b} = S\mathbf{a} + \delta\mathbf{b} + \mathbf{b_s}$$

The source vector $\mathbf{b_s}$ represents the effect of *deterministic* voltage or current generators inside the circuit which cause outgoing waves with various amplitudes and phases to emanate from the circuit's ports even in the case that the incoming wave amplitudes $\mathbf{a}$ all vanish. In contrast. $\delta\mathbf{b}$ represents outgoing *noise* waves. which are the result of various noise current or noise voltage generators inside the circuit. These noise waves are characterized by the noise wave correlation matrix. whose elements are the expectation values

$$(C_S)_{ij} = \left\langle \delta b_i \, \delta b_j^* \right\rangle$$

Finally, the product $S\mathbf{a}$ represents the outgoing waves that are generated when the incoming deterministic waves $\mathbf{a}$ are scattered by the N-port circuit. which has a scattering matrix S.

III. Noise in Passive Elements

The noise correlation matrices of *passive* elements or passive circuits at a uniform temperature T are computed from their scattering matrices:

$$C_S = \frac{h\nu}{2k_B} \coth(h\nu/2k_B T)\,(I - SS^\dagger)$$

where S is the scattering matrix, I is the identity matrix. and C_S is the noise correlation matrix. These equations are derived and discussed in [16]. [17]. [13].

The Callen–Welton formula [18] used for the noise spectral density can be considered to be the sum of a Planck term and a zero–point quantum fluctuation term.

$$\frac{h\nu}{2k_B} \coth(h\nu/2k_B T) = \frac{h\nu/k_B}{\exp(h\nu/k_B T) - 1} + \frac{h\nu}{2k_B}$$

and reduces to T in the Rayleigh–Jeans limit. $h\nu \ll k_B T$. Because the zero–point term is included, the use of the Callen-Welton spectral density for passive elements ensures that the correct value of the IF output noise is calculated for a mixer whose RF input is terminated with a load [19]. [20]. Thus, SuperMix automatically includes both quantum noise as well as the thermal noise produced by lossy elements (e.g. warm optics). Note that commercial microwave programs usually use the Rayleigh–Jeans approximation for thermal noise, which is inappropriate for cryogenic receiver

systems at millimeter wavelengths which have components at temperatures below $h\nu/k_B$.

The passive noise computations are carried out by a single standard function. Thus, when adding new passive elements to the library, it is usually sufficient to program only the scattering matrix calculation, and then use the standard library function to calculate the noise correlation matrix.

IV. CONNECTING CIRCUITS

The SuperMix library provides an object of type `circuit` to create composite circuits. A user can assemble a circuit from the basic elements (resistors, capacitors, etc.) by calling the `connect` "member" function belonging to the `circuit` object. The `connect` function allows the user to connect ports between circuits or basic elements. For instance, the output port of an isolator might be connected to the input port of an amplifier. The circuits being connected can have arbitrary numbers of ports; hence the concept implemented by the `connect` function is much more general than a simple cascade of two–ports. Furthermore, the `connect` function allows both *interconnections*, in which the two subcircuits are distinct, as well as *intraconnections*, in which two different ports of the same circuit are being connected. Series and parallel "tee" components are available to allow series or parallel connections.

Since all elements including the composite circuits share the same "software interface" (in C++ parlance, they are derived from a single generic device class), composite circuits themselves can be used in other composite circuits. Thus, a hierarchical design is possible: large, complicated circuits can be built up from smaller, more manageable pieces. This capability is a direct benefit of the modular design described in the Introduction.

A composite circuit calculates itself by making a single connection between the ports of its elements or subcircuits at a time. The S and C_S matrices for the two subcircuits are combined using equations developed by Wedge and Rutledge [13] using signal flow graphs. When a composite circuit is asked to calculate itself, it splits itself into two subcircuits and asks each for its S and C_S matrices and source vector $\mathbf{b_s}$. Each of these subcircuits splits itself in two. The process continues until all subcircuits are broken down to the original basic elements. This process creates a binary tree. The desired circuit is the base, and the tips of the branches are the components of the circuit. The intermediate subcircuits form the branches. This algorithm is fast because the tree only needs to be built a single time yet can be used for an unlimited number of calculations, such as for different frequency points or in the inner loop of the optimizer.

V. TRANSMISSION LINES

The SuperMix C−− library includes a number of classes which can be used to calculate the properties of normal and superconducting transmission lines such as the characteristic impedance and propagation constant. The transmission lines can be used as elements in circuits such as SIS mixers. At present, microstrip is the only physical type of transmission line available in SuperMix, although adding new types

is straightforward. In particular, the user only needs to calculate the characteristic impedance and the propagation constant; the conversion of these quantities into the scattering and noise correlation matrices is handled by library routines.

The characteristics of thin–film transmission lines (such as microstrip) often depend heavily on the surface impedance of the metal conducting films used. At present, SuperMix can calculate the surface impedance for normal metals and superconductors [1] in the case of a *local* conductivity $\sigma(\omega)$, defined by Ohm's law $\vec{J}(\vec{r},\omega) = \sigma(\omega)\,\vec{E}(\vec{r},\omega)$. Addition of the nonlocal *anomalous* skin effect in normal metals is planned for a future release. Multilayered metal films can also be created. The simplifying assumption is that the conductivity of any given layer is not influenced by the properties of the surrounding layers. This assumption may not be valid in some circumstances – for instance, in cases where the superconducting proximity effect is important.

Transmission lines such as microstrips often use dielectric films or substrates, and the complex dielectric constant ϵ of the material must be specified. Dielectrics can be defined in SuperMix by specifying the real part of ϵ and the loss tangent, $\mathrm{Im}[\epsilon]/\mathrm{Re}[\epsilon]$. If necessary, a table of values specifying a frequency–dependent $\epsilon(\nu)$ can be read from a data file and automatically interpolated. Actually this can be done for any circuit parameter, and the interpolation can be done with respect to any other parameter (e.g. temperature), not just frequency. Metal films, layered films, and dielectrics form the building-blocks for transmission lines. After the dielectrics and conductors have been specified, transmission lines such as microstrip can be assembled from them.

VI. Mixers

Mixers are created using a combination of linear `circuits` and devices derived from a generic nonlinear device class named `junction`. The `junction` class simply provides a standard "software interface" for devices that can calculate both their large and small signal responses. At the moment, the only nonlinear devices available within SuperMix are superconducting tunnel (SIS) junctions. The SIS junction reads its current-voltage (IV) curve from a data file, which can be either experimentally measured or mathematically generated. However, other nonlinear devices (such as Schottky diodes) could be added by writing the code that simulates the device behavior, and interfacing this code to SuperMix using the `junction` class. This would allow a complicated multiple diode mixer circuit to be simulated, including a full harmonic balance solution of the LO waveform. In principle, one could even simulate a circuit containing a mixture of superconducting and semiconducting nonlinear devices.

The `mixer` device is assembled from any number of `junctions`, a radio-frequency (RF) linear circuit, and an intermediate-frequency (IF) linear circuit. These circuits can be arbitrarily complicated and can contain arbitrary numbers of ports. The local oscillator (LO) can be injected anywhere in the RF linear circuit. The `mixer` will then calculate its operating state by performing a harmonic balance; the number of harmonics used in the balance is arbitrary as well. Once the operating state is determined, the user can ask the `mixer` object to perform a small signal analysis [6]. Once the calculation is complete, which typically takes a few tens of milliseconds per frequency point, the user may examine standard quantities such as noise temperature

or conversion gain, or look into details such as the impedance, admittance, or reflection coefficient looking into each port at each sideband of each harmonic of the LO, as well as the transmission via mixing from one harmonic of the LO to another, including the IF. In other words, the entire multifrequency scattering and noise matrices are calculated and are available to the user.

VII. THE OPTIMIZER

The optimizer was designed to be a very general tool that can minimize general error functions by varying nearly any circuit parameter, real or complex. At present, SuperMix allows the standard quantities of interest such as gain, noise temperature, input or output match to be readily included in the definition of the error function. These quantities are usually calculated while sweeping the frequency over one or more bands. For instance. the design of a bandpass filter requires that the transmission across the passband be maximized while the transmission in stopbands be minimized. SuperMix supports this capability; furthermore, any parameter can be swept, not just frequency. Even multi-dimensional sweeps are allowed.

The modular design allows easy addition of new types of error function terms. In addition, the interface between the error function and the optimization algorithm is very simple and generic, and so it is easy to add new minimization algorithms. Although SuperMix allows for the possibility of multiple minimization algorithms, the only algorithm currently implemented is Powell's method. This routine is both robust and fast. In practice, it has been used to optimize 10 parameters of a 50 element linear amplifier across an octave bandwidth. On a 333 MHz Sun Ultra 10, this entire optimization and simulation runs in less than 10 seconds. One downside of Powell's method is that it can converge upon local minima, so the user must choose reasonable initial values for the parameters being optimized. This is a generic feature of aggressive optimization alogrithms that make use of gradient information. Other approaches. particularly the stochastic algorithms. are much slower but are better at finding global minima. and could easily be added to SuperMix.

VIII. USING THE SUPERMIX LIBRARY

A circuit is simulated by writing a C+- program that uses the SuperMix library. Devices in the library are represented as C++ class objects. which can be used simply by declaring a variable of the appropriate type. For example. declaring a variable of type **resistor** will automatically provide all functionality needed for the simulation of a resistor. including parameters for resistance and physical temperature, and functions to return the scattering matrix. noise correlation matrix, noise temperature, etc. of this element. All elements are derived from a single generic device class, guaranteeing a uniform standard interface for all devices and allowing the user to add elements to the library with a minimum of effort.

A. Simulation of a low-noise amplifier

The code in figure 6 is a complete program to optimize and simulate the single-stage low-noise HEMT amplifier drawn in figure 1. This program demonstrates the

key features of the linear circuit library, including creating circuit elements. connecting them, and running the optimizer. Results are plotted in figure 2.
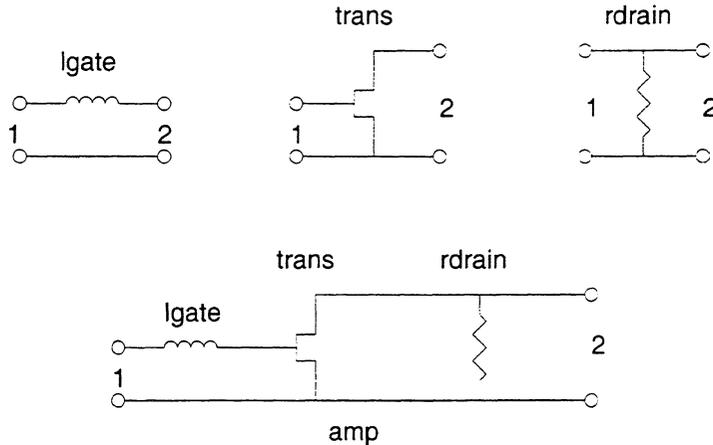

Fig. 1. Block diagram of single stage LNA.

In this example, there are only three subcircuits being connected together to produce the composite circuit amp: the gate inductor lgate, the Fujitsu FHX13X transistor trans, and the drain resistor rdrain. (The SuperMix library actually constructs a composite equivalent circuit to calculate the transistor according to the Pospieszalski model [21].) Each of the subcircuits is a two-port: the ports are labeled by integers as shown in figure 1. The C++ statement amp.connect(lgate, 2, trans, 1) connects port 2 of the inductor to port 1 (the gate) of the transistor. Similarly, amp.connect(trans, 2, rdrain, 1) connects the drain of the transistor to the resistor. The statement amp.add_port(lgate, 1) simply specifies that port 1 of the inductor will be used as port 1 (input) of the overall amplifier circuit. Finally. amp.add_port(rdrain, 2) specifies the output port of the amplifier.
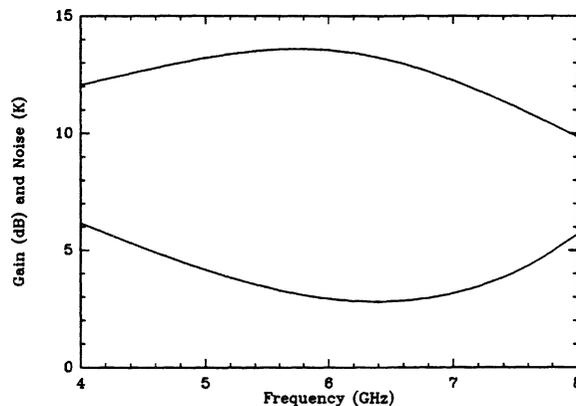

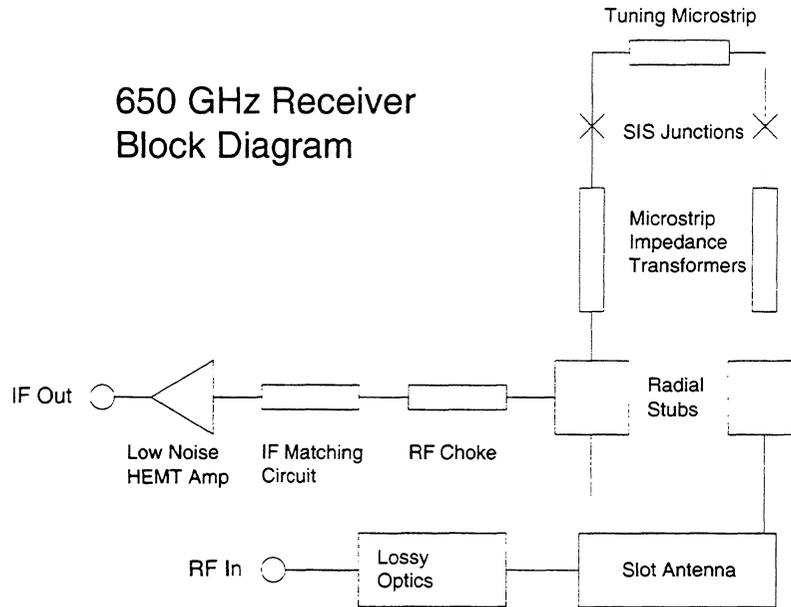Fig. 2. Gain (top curve) and noise temperature (bottom curve) of a single stage LNA.

Fig. 3. Block diagram of a 650 GHz quasi-optical twin-slot double-junction SIS receiver.

B. Simulation of a section of microstrip

The code in figure 7 is a complete program to simulate a segment of superconducting microstrip. It demonstrates the ease with which a circuit can be constructed and the high level of readability of a C++ circuit file.

C. Simulation of a 650 GHz receiver

A 650 GHz quasi-optical twin-slot double-junction SIS receiver (figure 3) was simulated using the SuperMix library. The frequency-dependent impedance of the slot antenna and a measured unpumped current-voltage (IV) curve were read from data files. The losses of optical components such as the LO injection beamsplitter, the cryostat vacuum window, etc. were simulated using attenuators whose temperatures were set to the physical temperatures of the optical components. In this way, the thermal noise produced by these components was included in the simulation. In addition, a complete 3 stage HEMT amplifier was included at the IF output as part of the simulation. The complete simulation can calculate about 35 frequency points per second on a 333 MHz Sun workstation, including a full simultaneous harmonic balance calculation for the two SIS junctions at every frequency point.

The simulated pumped IV curve and IF power curves (figure 4) agree reasonably well with measured data. The measured and simulated receiver passband and noise temperatures (figure 5) are also similar. In this example, we have made no attempt to adjust the parameters of the simulation to make it agree with the measured data. However, in the future these types of simulations could be used to extract information about the performance of the receiver (e.g. the noise contribution due to optics
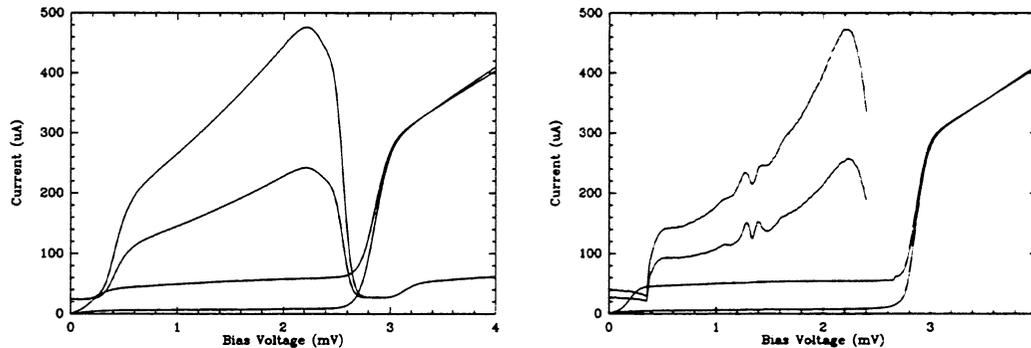
Fig. 4. Simulated pumped IV curve with 300 K and 77 K IF power curves at 638 GHz (left) compared to the measured values (right). The HEMT amplifier and IF matching circuit that were simulated are not the same as in the actual receiver; hence, the shapes of the IF power curves are not expected to be the same in detail.
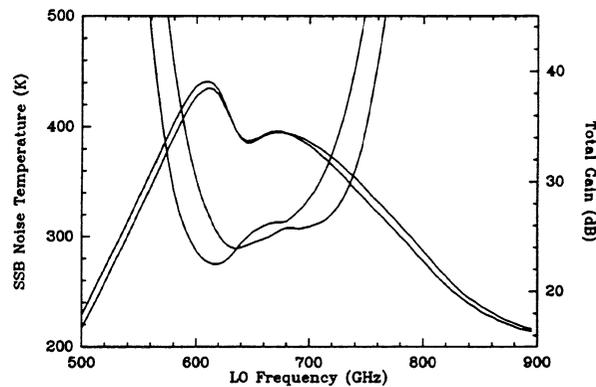

Fig. 5. Simulated passband and noise temperature of a 650 GHz receiver. The double curves are for lower and upper sidebands; the IF frequency is 6 GHz.

losses, the behavior of the tuning circuit, etc.) and to provide insight for possible optimizations.

IX. COMPATIBILITY AND AVAILABILITY

The software was written in standard C++ and should compile on any UNIX platform using an ANSI C++ compiler that supports the Standard Template Library (STL.) The package has been tested using the freely available GNU g++ compiler on the SPARC Solaris 2.6 and Intel Linux platforms.

The library, which is currently composed of over 28,000 lines of source code, is already nearly fully functional, and a user's manual is being prepared. Our intention is to provide this software, including the source code, free of charge to any interested users. We hope that other users will find SuperMix useful and will contribute to its

```
#include "supermix.h"

int main(void) {

  // Set the global temperature.
  device::T = 4.2 * Kelvin;

  // Declare the components we will use for the circuit.
  fhx13x trans;       // Fujitsu HEMT transistor
  inductor lgate;     // Gate tuning inductor
  resistor rdrain;    // Drain bias line

  // The inductor is series, resistor parallel.
  lgate.series();
  rdrain.parallel();

  // Construct the circuit.
  circuit amp;
  amp.add_port(lgate, 1);              // Put the gate inductor at the input.
  amp.connect(lgate, 2, trans, 1);     // Connect the inductor and transistor.
  amp.connect(trans, 2, rdrain, 1);    // Connect the transistor and resistor.
  amp.add_port(rdrain, 2);             // Make the drain resistor the output.

  // Define the optimization band.
  sweeper band1;
  band1.sweep(device::f, 4.*GHz, 8.*GHz, 0.5*GHz);

  // Declare an error function.
  err_func ef;

  // Define which parameters are optimized.
  lgate.L = ef.vary(1.*nHenry, 3.0*nHenry, 10.*nHenry);
  rdrain.R = ef.vary(30.*Ohm, 50.*Ohm, 100.*Ohm);

  // Define the error functions.
  gain_dB gain(12, 1, 2, err_term_mode::MATCH);  // Optimize for flat gain.
  ef.add_term(1.0, band1, gain, amp);
  input_tn tn(0.0, 1, 2, err_term_mode::MATCH);  // Optimize for low noise.
  ef.add_term(2.0, band1, tn, amp);

  // Use the Powell algorithm for minimizing.
  powell pow1(ef);            // Minimize the error function we created above.
  pow1.POW_TOL = 0.00001;     // Define the tolerance for the convergence test.
  pow1.minimize();            // Run the optimizer.
  ef.get_parms().show();      // Print the optimized parameters.

  cout << "\n   Freq    S21(dB)   Tn(K)" << endl;

  // Print out gain and noise temperature in our optimization band.
  for(double freq = 4.0*GHz; freq <=8.0*GHz; freq += 0.2*GHz)
  {
    device::f = freq;
    sdata sd = amp.get_data();
    cout << freq/GHz              // Print the frequency in GHz.
         << "  "
         << sd.SdB(2,1)           // Print the gain in dB.
         << "  "
         << sd.tn(2,1)/Kelvin     // Print the noise temperature in Kelvin.
         << endl;
  }
}
```

Fig. 6. Simulation and optimization of a 1 stage LNA.

```
#include "supermix.h"

main()
{
  device::T = 4.2*Kelvin ;      // Default temperature
  device::Z0 = 50.*Ohm ;        // Normalization impedance

  // Define metal film
  super_film nb1;

  // Standard parameters for Niobium
  nb1.Vgap = 2.9*mVolt ;
  nb1.Tc = 9.2*Kelvin ;

  double cm = 1.e4 * Micron;
  nb1.rho_normal = 5.*Micro*Ohm*cm ;

  // Set thickness
  nb1.Thick = 2000.*Angstrom ;

  // Define dielectrics
  const_diel sio ;
  const_diel vacuum ;

  vacuum.eps = 1.0 ;
  vacuum.tand = 0.0 ;

  sio.eps = 5.6 ;    // dielectric constant
  sio.tand = 0. ;    // loss tangent

  // Now make a microstrip line
  microstrip ms1;

  ms1.top_strip(nb1) ;              // use Nb film for top strip
  ms1.ground_plane(nb1) ;           // same for ground plane
  ms1.superstrate(vacuum) ;         // nothing above
  ms1.substrate(sio) ;              // SiO insulator
  ms1.sub_thick = 2000.*Angstrom ;  // SiO thickness

  ms1.length = 100.0*Micron ;    // microstrip length
  ms1.width = 5.0*Micron ;       // microstrip width

  // Print out S21 for this microstrip from 100 to 1000 GHz, step by 10 GHz.
  for(double freq = 100. ; freq <= 1000.; freq += 10)
  {
    device::f = freq*GHz ;
    complex S21 = ms1.S(2,1) ;      // The S(2,1) function returns S21.
    cout << freq << "   "
      << zabs(S21) << "   "         // Print |S21|
      << arg(S21)/Degree << "   "   // and arg(S21)
      << endl ;
  }
}
```

Fig. 7. Simulation of superconducting microstrip in SuperMix.

development by adding new circuit elements. optimization algorithms, etc. However, we cannot yet make SuperMix available for public download until certain copyright issues have been resolved. After these issues are resolved and the documentation is completed, the package will be available at http://www.submm.caltech.edu/supermix.html.

REFERENCES

[1] D. C. Mattis and J. Bardeen. "Theory of the anomalous skin effect in normal and superconducting metals." *Phys. Rev.*, vol. 111, pp. 412–417. 1958.

[2] R. Pöpel. "Surface impedance and reflectivity of superconductors," *J. Appl. Phys.*, vol. 66, pp. 5950–5957. 1989.

[3] M. Bin. *Low-Noise Terahertz Niobium SIS Mixers*. PhD thesis. California Institute of Technology, 1997.

[4] J. R. Tucker and M. J. Feldman. "Quantum detection at millimeter wavelengths," *Rev. Mod. Phys.*, vol. 57, no. 4, pp. 1055–1113. 1985.

[5] S. Withington and E. L. Kollberg. "Spectral-domain analysis of harmonic effects in superconducting quasiparticle mixers." *MTT*, vol. 37, pp. 231–238. January 1989.

[6] F. Rice. "Fast harmonic balance of SIS mixers with multiple junctions and superconducting circuits," *Proceedings. Tenth International Symposium on Space Terahertz Technology*. 1999.

[7] S. Padin, D. P. Woody. J. A. Stern, H. G. LeDuc, R. Blundell, C. Y. E. Tong, and M. W. Pospieszalski, "An integrated SIS mixer and HEMT IF amplifier." *MTT*, vol. 44, pp. 987–990. June 1996.

[8] J. Zmuidzinas, H. G. Leduc. J. A. Stern, and S. R. Cypher. "Two-junction tuning circuits for submillimeter SIS mixers." *IEEE Transactions on Microwave Theory Techniques*, vol. 42, pp. 698–706, April 1994.

[9] T. Noguchi, S. C. Shi, and J. Inatani. "Parallel connected twin SIS junctions for millimeter and submillimeter-wave mixers – analysis and experimental verification," *IEICE Trans. Elect.*, vol. E78C, pp. 481–489. May 1995.

[10] S. C. Shi and T. Noguchi. "Low-noise superconducting receivers for millimeter and submillimeter wavelengths." *IEICE Trans. Elect.*, vol. E81C, pp. 1584–1594. October 1998.

[11] C. Y. E. Tong, L. Chen, and R. Blundell. "Theory of distributed mixing and amplification in a superconducting quasi-particle nonlinear transmission line," *MTT*, vol. 45, pp. 1086–1092, July 1997.

[12] B. Stroustrup. *The C++ Programming Language*. Reading: Addison-Wesley, 1997.

[13] S. Wedge. *Computer Aided Design of Low Noise Microwave Circuits*. PhD thesis, California Institute of Technology. 1991.

[14] S. W. Wedge and D. B. Rutledge. "Wave techniques for noise modeling and measurement," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-40, pp. 2004–2012, 1992.

[15] J. A. Dobrowolski and W. Ostrowski. *Computer-Aided Analysis, Modeling, and Design of Microwave Networks*. Boston: Artech House, 1996.

[16] H. Bosma. "On the theory of linear noisy systems," *Phillips Res. Repts. Suppl.*, vol. 10, 1967.

[17] S. W. Wedge and D. B. Rutledge. "Noise waves and passive linear multiports," *IEEE Microwave Guided Wave Lett.*, vol. MGL-1, pp. 117–119. 1991.

[18] H. B. Callen and T. A. Welton. "Irreversibility and generalized noise," *Phys. Rev.*, vol. 83, pp. 34–40, 1951.

[19] M. J. Wengler and D. P. Woody. "Quantum noise in heterodyne detection," *IEEE J. Quantum Electronics*, vol. QE-23, pp. 613–622, 1987.

[20] I. A. Devyatov, L. S. Kuzmin, K. K. Likharev, V. V. Migulin, and A. B. Zorin. "Quantum-statistical theory of microwave detection using superconducting tunnel junctions." *J. Appl. Phys.*, vol. 60, pp. 1808–1828, 1986.

[21] M. W. Pospieszalski. "Modeling of noise parameters of MESFETs and MODFETs and their frequency and temperature dependence." *MTT*, vol. 37, pp. 1340–1350, September 1989.