Malt 3.1.1 Methods and Meaning

Every Option Defined! Nothing Left Out!

Quentin Herr
Trent Josephsen

21 April 2025

Mathematical Overview

Malt is a parametric yield calculator and optimizer of digital circuits. Malt is named in honor of Malthus, who was the first to describe systems in which the demand for resources grows exponentially, while the resources themselves grow linearly. Multi-parameter optimization using limited compute resources is like that.

Algorithm for Calculating Yield. The yield algorithm calculates the multi-dimensional Gaussian integral across the operating region of the circuit. In the simplest case, yield is given by the Normalized Upper Incomplete Gamma Function, $Q(a,z)$, which is a generalization of the *complementary error function* $\operatorname{erfc}(z)$. In N dimensions, given a circuit with the operating region defined by the hypersphere of radius r,

$$\text{yieldc}_{\text{hypersphere}}(N,r) = Q\left(\frac{N}{2},\frac{r^2}{2}\right).$$

This reduces to the familiar complementary error function for N=1,

$$\text{yieldc}_{\text{hypersphere}}(1,r) = Q\left(\frac{1}{2},\frac{r^2}{2}\right) = \operatorname{erfc}\left(\frac{r}{\sqrt{2}}\right).$$

For an operating region of arbitrary shape, yield is given by the integrating the Gaussian probability function over the entire volume of the operating region. Numerical integration can be used to compute the yield of a real circuit by mapping out the organic shape of the operating region piece-by-piece. If the shape of the operating region is well-behaved (see the requirements below), the operating region can be mapped out and partitioned into simplexes using a binary search to find points on the boundary of the operating region. Specifically, the set n of points associated with each simplex gives the mean value of the complementary yield and the standard deviation. This set of points also gives the value of the solid angle of the simplex Ω. These terms provide a numerical estimate for the contribution of each simplex:

$$\text{yieldc}_{\text{simplex}}(n) = \Omega(n)\text{ave}\left(Q\left(\frac{N}{2},\frac{r_n^2}{2}\right)\right)$$

$$\text{var}\left(\text{yieldc}_{\text{simplex}}(n)\right) = \Omega(n)\text{var}\left(Q\left(\frac{N}{2},\frac{r_n^2}{2}\right)\right)$$

$$\text{yieldc}_{\text{region}} = \sum_n \text{yieldc}_{\text{simplex}}(n) \pm \left(\sum_n \text{var}\left(\text{yieldc}_{\text{simplex}}(n)\right)\right)^{\frac{1}{2}}$$

The contribution of each simplex can be used in an adaptive algorithm that determines where to add new points on the boundary.

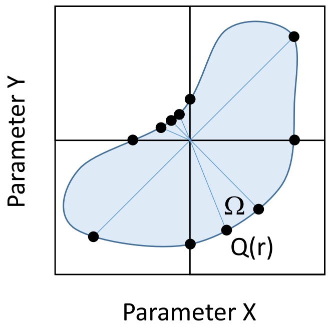


Figure 1. In the numerical integration, the average Q value of each simplex (defined by N points in N dimensions) is weighted by the solid angle Ω. The weighted variance of the Q values gives an error estimate.

Malt first calculates the margin of the "corner vector" in each quadrant, octant, or orthant, and then calculates yield on a per-orthant basis using the margin of the corner vector to predict the contribution. This is effective to the extent that the parameters are highly interdependent. (If one or more of your parameters is not highly interdependent, so much the better. Remove it from the group and calculate its contribution to yield based on individual parameter margins using the $\mathrm{yieldc}_{\mathrm{simplex}}$ expression above.) The number of orthants is large, equal to 2^N. However, the yield integral is dominated by the points closest to the origin (nominal parameter values), due to the rapid decay of the Gaussian. It means that the solution can be approximated efficiently by preferentially evaluating yield only in those orthants where the boundary of the operating region is closest to the origin.

The algorithm requires that the binary searches used to define the operating region are single-valued. Some different operating regions that illustrate this requirement are shown in Figure 2.

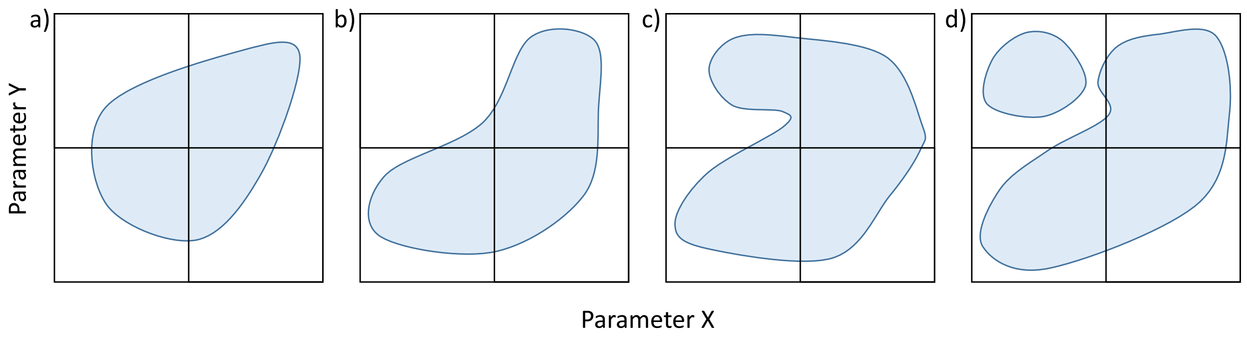


*Figure 2. Four operating scenarios are shown: a) convex; b) not convex, but single-valued; c) not single-valued, but simply-connected; d) not simply-connected. The optimization (**−o**) algorithm requires that the operating region be convex. The yield (**−y**) algorithm has the less restrictive requirement that the operating region boundary be single-valued for binary searches starting at the origin. Single-valuedness may therefore depend on the positioning of the origin within the operating region.*

Algorithm for Parametric Yield Optimization. Malt optimizes parameter values using the design-centering algorithm of Director and Hachtel found in *IEEE Circuits and Systems*, 1977. The algorithm maps out the operating region using binary search, builds a simplex to approximate the operating region, and inscribes the largest possible hypersphere (effectively a hyperellipsoid) in the simplex using linear programming. The center of the hypersphere corresponds to the optimal parameter values. This heuristic is a good approximation to yield optimization as it positions the parameter values as far as possible from the edges of the operating region in multi-parameter space. The complexity of the simplex is large. In eight dimensions, 100 iterations of the binary search might well produce 100,000 simplex elements. Higher dimensions are supported, but the computation time will increasingly be dominated by simplex operations instead of circuit simulation.

The convexity requirement, illustrated in Fig. 2, bears further discussion. Josephson circuits using SFQ or RQL data encoding are usually well-behaved in this respect, and the optimization is fast and efficient. The algorithm may fail for circuits with multiple internal modes of operation that pass, or circuits with underdamped junctions that have stochastic effects. The routine reports the validity of the convexity assumption after completing parameter optimization and then recalculates individual parameter margins. This makes the success or failure of the optimization apparent.

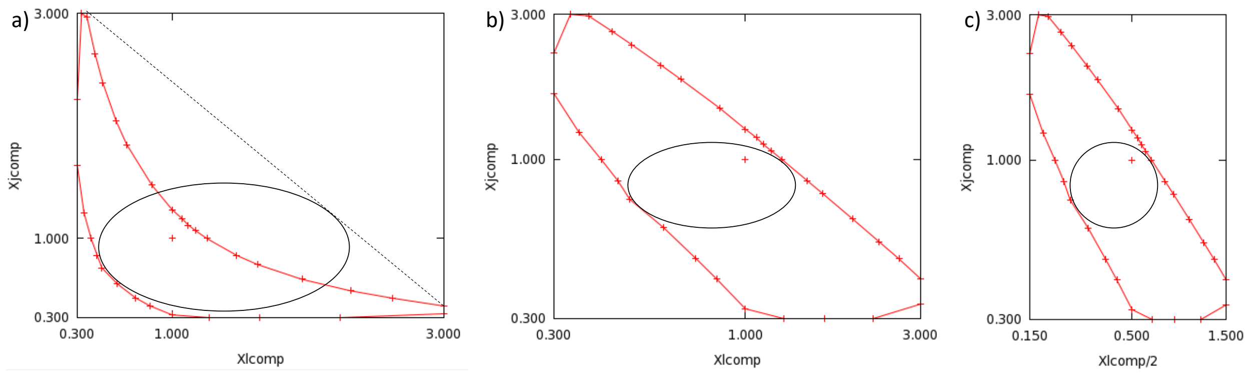


Figure 3. Malt-space transforms are illustrated. a) Many parameters have an inverse dependence, causing a strongly non-convex operating region. Points interior to the convex simplex will be ignored, leading to a poor approximation. b) Transforming parameters to log space solves gives the desired result. c) The optimization algorithm effectively inscribes an ellipse in the operating region, with axes proportional to the sigma of each parameter. This is implemented by inverse scaling of the parameter values in malt space.

Malt Space. Converting parameters to log space can help achieve the convexity requirement, as shown in Figure 3. In fact, most parameters are Gaussian distributed in log space, not linear space. This is generally true of any parameter for which it is unphysical to have both positive and negative values, such as inductors, capacitors, resistors, and junction critical currents. Parameters that are better in normal space include bias currents such as the bias tap offset and input pattern offset. What about the ac clock amplitude? That is better in linear space. However, if you specify the ac clock in dB, you should use log space. In fact, these two paradigms are mathematically equivalent! In linear space, the center between high and low values is the mean. In log space, it is the geometric mean.

Malt uses a variation on log space scaled to the unit of the parameter. The transform to log space is

$$f(x) = \text{nom} \cdot \left(\log\left(\frac{x}{\text{nom}}\right) + 1.0\right),$$

where nom is the nominal parameter value. The function maps the nominal value to itself. The function is plotted in Figure 4.

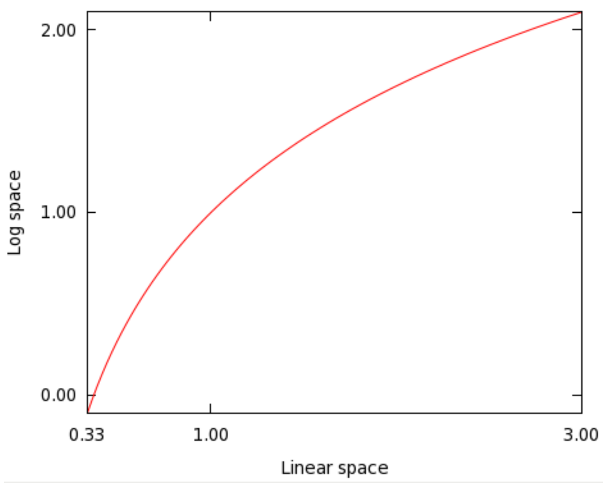


Figure 4. Figure on the transform in units of nominal. This transform is applied before the scaling with sigma mentioned above.

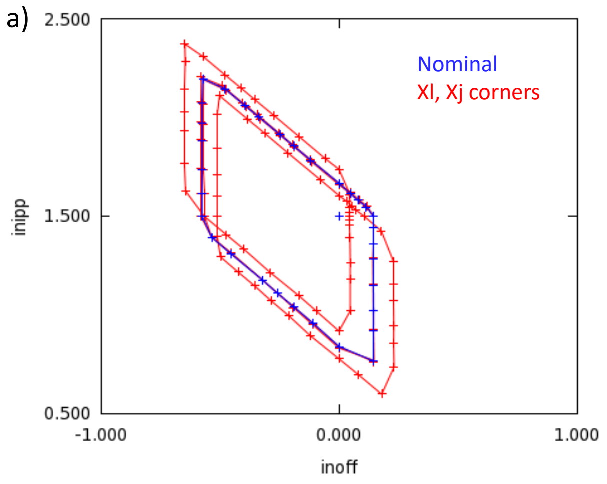
Linear vs. log-space is specified on a per-parameter basis. Either way, high and low margins are quoted both as physical parameter values, and as units of sigma. Margins in units of sigma can be derived from the physical parameter values for both linear and log space as follows:

$$\mathrm{marg}_{\mathrm{lin}}(x) = \frac{x - \mathrm{nom}}{\sigma}$$

$$\mathrm{marg}_{\mathrm{log}}(x) = \frac{f(x) - \mathrm{nom}}{\sigma}$$

where x is the high or low physical parameter value and σ the sigma value. As defined above, nom is the nominal physical parameter value and $f(x)$ is the transform to malt space.

Corner parameters. Malt allows specifying some parameters to be used for corners. Corner parameters are not included in the binary search, but are instead alternately set to high and low values. If there are N corner parameters, there are 2^N combinations of high and low, which are the parameter corners. Margins are calculated at each parameter corner (in parallel, up to the configured process limit). Only the worst-case margin is reported. This means that the operating region of the circuit is the logical intersection of the operating regions at all corners, as illustrated in Figure 5. Parameters that have uniform distributions should be specified as corner parameters. This is true for global parameters, where parts that are out-of-tolerance are discarded. Treating globals as corner parameters has an additional advantage for optimization, as their inclusion in this way does not increase the dimensionality of the operating-region simplex.



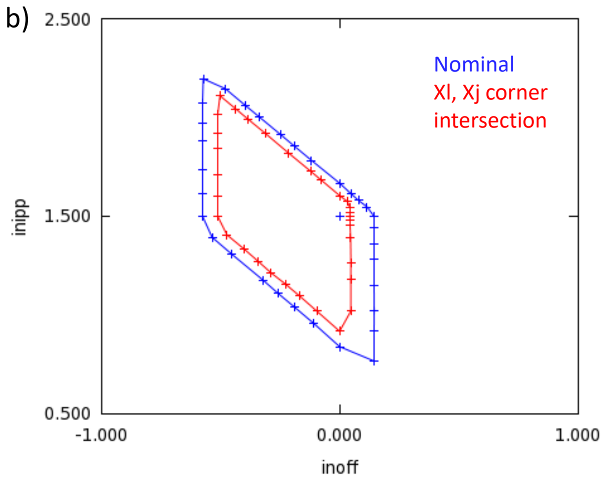


Figure 5. The operating region of two parameters Xl *and* Xj *is shown a) with all other parameters at nominal, and with* Xl *and* Xj *set to their parameter corners, each in turn. When corner parameters are specified as in b), Malt reports the intersection of the corner-based operating regions.*

Noise and Bit-Error Rate. Margins and yields are calculated in noise-free circuit simulation. The resulting margins correspond to a bit-error-rate of 0.5, whereas large-scale applications may require a bit-error rate per gate of 10^{-24} or even lower. An effective way to prorate the margins and yields for bit-error rate is to

1) measure the bit-error rate to determine the factor by which margins are narrowed between a BER of 0.5 and the desired BER.
2) Scale the sigmas for all parameters up by this factor.

This method is shown in Fig. 6.

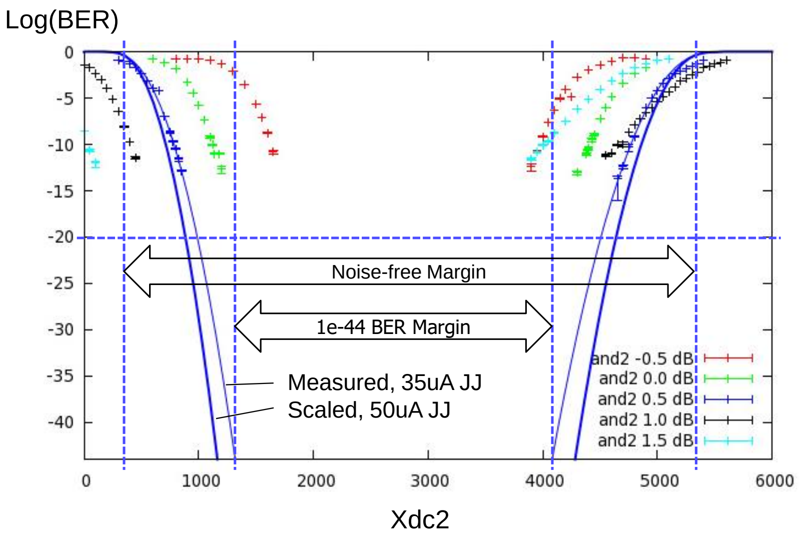


Figure 6. Measured BER and error-function curve fits are shown for an AND2 gate. The effective margin at a particular BER is always narrower than that given in noise-free simulation, which corresponds to a BER of 0.5. The factor by which margins narrow, F, may be largely independent of the particular knob plotted on the x axis. Yield at a particular BER can be estimated by increasing all device sigmas by a factor of $1/F$.

Note that the simplex-based yield estimate method described here can efficiently compute yield for multiple sigma values, so long as they are scaled by a common factor. This only involves reevaluation of the analytic Q function, and does not require additional circuit simulations. Yield as a function of sigma may be prove to be a good fit to the Q function, where both radius and dimension are fitting parameters.

Quick Start Guide

Prerequisites. Malt depends on WRspice, gnuplot, and the Linux command line. You can type in your own netlist with your favorite text editor or generate it using the native WRspice schematic editor, but the model-based custom netlister that we have implemented in the Cadence schematic editor environment is highly recommended. All files invoked below are listed in Appendix 2 and may be included with your Malt distribution.

Netlist. Consider the following netlist:

```
netlist.cir
```

which is self-contained except for the definition of certain parameter nominal values,

```
netlist.nominals
```

and the included model file,

```
my_models.lib
```

which is intended only to serve as a self-contained example for present purposes.

The netlist and model file will normally be created by the netlister, but the individual parameters and nominals must be defined by the user by editing the text files.

WRspice. From the `wrspice` command line, type:

```
>  source netlist.nominals
>  source netlist.cir
>  run
>  let
>  plot v(phi.Xb0.XI1) v(phi.Xb1.XI12)
```

where the `let` command is very useful in figuring out the names of the circuit vectors that you may want to plot.

All these commands and more are included in a WRspice script. From the WRspice command line, run the script by typing its name:

```
>  netlist.run
```

Assuming you are satisfied with your inspection of the results, use the following command to exit WRspice:

```
>  quit
```

Malt. Invoke the `malt` executable with the `-h` option to display a help message.

```
$  malt -h
```

```
Malt 3.1.1
  Parametric yield optimization utility for use with WRSpice
USAGE
  malt [-h] {-d|-m|-t|-2|-y|-o} [-k] CONFIG

COMMANDS
  -h    Print help message and exit
  -d    Define correct circuit operation
  -m    Calculate individual parameter margins
  -t    Trace nodes at marginal parameter values
  -2    Calculate operating region in 2 dimensions
  -y    Calculate circuit yield using corner analysis
  -o    Optimize yield using inscribed hyperspheres

OPTIONS
  -k    Keep (don't delete) additional temporary files

CONFIG
  Path (relative to the current directory) of the most specific applicable
  configuration (.toml) file for this analysis. When Malt runs, it searches in
  the current directory and all ancestors for a file named Malt.toml, which is
  taken as the base configuration. Then all .toml files of which CONFIG is a
  path prefix are processed from least to most specific.

  Generated files such as parameter, pass/fail, and envelope files are searched
  for in similar fashion starting from the _malt directory, if one is present.
```

Malt will make use of the netlist file `netlist.cir`, where the `.cir` file extension is what Malt expects for netlist files (this extension can be modified in the configuration file). The "liberated" parameters, and the node to be checked, are defined in the similarly-named file, `netlist.toml`, where the `.toml` file extension is what malt requires for configuration files. See Appendix 1 for more details on the TOML format and supported configuration constructs.

To define correct circuit operation, from a shell run:

```
$  malt -d netlist
```

Malt will

1) invoke WRspice to simulate `netlist.cir` at nominal parameter values
2) plot the node waveforms
3) plot the envelopes around the nodes that define correct circuit operation
4) exit to the WRspice command line so you can manipulate the plots.

To compute parametric margins, yield, and optimized parameter values, after `malt -d` run:

```
$  malt -m netlist
$  malt -y netlist
$  malt -o netlist
```

respectively. Of course, you could also recompute yield after the optimization by updating the nominal parameter values in the config.

Malt Reference

Project Structure, Input and Output Files

A Malt project consists of a directory containing a file named `Malt.toml` (capitalization is significant) and all the directories and files under it. Malt calls this the **project tree**. When you run `malt`, it will search for the `Malt.toml` file first in the current working directory and then in all its ancestors starting at the parent. If the current directory is not part of a project tree, Malt generates a new `Malt.toml` file in the current directory. Reading the automatically generated file is a useful way to learn about Malt's configuration options.

Malt is not a circuit simulator; it relies on an external command (WRspice) to perform simulations. Both Malt and the simulation may generate additional files when run. To keep project directories clean, Malt creates its generated files and runs the simulator in a directory named `_malt`, located in the root of the project tree (next to `Malt.toml`). Internally, Malt calls the `_malt` directory and its contents the **working tree**.

The working tree contains files automatically generated by Malt and the simulation, while the project tree contains mostly files created and edited by the user. You may create these kinds of input files in the project tree which Malt will automatically detect and use:

- Malt configuration files (with a `.toml` extension)
- WRspice netlists (with a `.cir` extension[1])
- Optional pass/fail control files (with a `.passf` extension[1])

How Malt discovers and parses configuration (`.toml`) files

When you invoke Malt on the command line, one or more `.toml` files may be read depending on the `CONFIG` argument. A file will be processed by Malt if the full path to the file (minus the significant extension) is a prefix of the CONFIG named on the command line (minus an optional `.cir` or `.toml` suffix, which is ignored). The files are processed from least to most **specific**; that is, starting at the project root and working down to the level named on the command line.

Example. Consider the following (not necessarily complete) project tree:

```
project/
├── netlist.toml
├── netlist.cir
├── netlist/
│   ├── one.toml
│   └── one/
│       ├── two.toml
│       └── two/
│           └── three.toml
└── Malt.toml
```

The directory named `project` is the project root, since it contains `Malt.toml`. Now imagine running the following command (when inside the project root):

```
$  malt -d netlist/one/two
```

The configuration named on the command line is `netlist/one/two`. The configuration files that apply and will be parsed by Malt are:

1. `Malt.toml` (always parsed)
2. `netlist.toml`
3. `netlist/one.toml`
4. `netlist/one/two.toml`

The only required configuration file is `Malt.toml`, since its location defines the project tree. All the other `.toml` files are optional. The contents of the more specific files (lower on this list) override settings defined in less specific files.

How Malt discovers circuit netlists and pass/fail control files

[1] The expected file extensions for netlists and control files can be changed in `Malt.toml`.

Malt searches for netlists in much the same way as configuration files, except for the file extension. By default, Malt expects files containing circuits to have a `.cir` extension; however, this can be changed in the `[extensions]` section of `Malt.toml`.

If there is no applicable `.cir` file, Malt will fail with an error message. If there are multiple applicable `.cir` files in the project tree, Malt uses only the most specific one. In the above example, `netlist.cir` will be used, since there is no more specific file with that extension.

Pass/fail (.passf) files (and other control files discussed in the next section) are searched for in the same way as .cir files. However, all such files are optional. A .passf file is only required when not using the envelope method to define correct circuit operation.

Other control files and generated files

Malt uses several other kinds of control files that are generated internally. These include `.param`, `.envelope`, and `.env_call` files. These files are created inside the working tree (that is, under the `_malt` directory) and should normally not need to be edited. The path of a file created in the working tree generally comes from the `CONFIG` argument given on the command line; for instance,

```
$  malt -d netlist/one/two
```

will generate files inside `_malt/netlist/one/two`.

Malt also saves its entire configuration from each run into a `.toml` file in the working tree. The name of the `.toml` file comes from the command line switch and its path comes from the `CONFIG` argument given on the command line. In the same example above, the calculated configuration will be saved in `_malt/netlist/one/two/d.toml`. You may use this file to check if Malt is using the configuration you expect.

Finally, Malt saves its own output stream to a file ending in `.out`. You may want to check this file if you run Malt in a script, or in case you forget to save the output for some reason. Note, though, that Malt will immediately overwrite this file the next time it runs with the same command line.

Files Recognized and/or Generated by Malt

Configuration (`.toml`)

This file, or cascade of files (see the section on project structure), defines the circuit parameters to be analyzed, the nodes to be monitored, and options for the various analysis types.

Project-wide and default settings may be defined in the special file `Malt.toml` in the project root.

All configuration files are written in TOML syntax. See Appendix 1 for details.

Circuit (`.cir`)

This file is the only file that must be provided by the user. It contains the WRspice circuit netlist, marked up with the parameter variables as needed.

Pass/fail (`.passf`)

This is an optional file that sets pass/fail criteria for correct circuit operation when not using the `-d` option. It is written in WRspice control code syntax.

Parameters (`.param`)

This is a file that specifies additional parameters with fixed values to the circuit netlist. It is usually not necessary. The usual way of specifying parameters from a Malt user's point of view is in an appropriately specific `.toml` file. It is written in WRspice control code syntax.

Envelope (`.envelope`)

This file is generated by the nominal simulation when running Malt with the `-d` (define) switch, and used by the other operating modes. It contains vectors that represent the allowed high and low bounds of each node in correct circuit operation, based on the nominal values and the envelope settings specified in `.toml` files. The `.envelope` file uses WRspice rawfile syntax.

Iterate (`*.iterate.[2syo]`)

To interrupt analysis iterations cleanly during runtime, remove the associated iterate file. This applies to the analysis types indicated on the line above.

Example Malt Commands and Related Files

This table may not be exhaustive.

Examples

Files read	Malt command	Config hierarchy	Files written
(empty directory)	`malt -d jtl`	(generates)	`Malt.toml`

`Malt.toml`
`jtl.toml`
`jtl.cir`
	`malt -d jtl`	`Malt.toml`
		`jtl.toml`
			`_malt/jtl/the.envelope`
			`_malt/jtl/d.toml`
			`_malt/jtl/d.out`

`Malt.toml`
`jtl.toml`
`jtl.cir`
`_malt/jtl/the.envelope`
	`malt -m jtl`	`Malt.toml`
		`jtl.toml`
			`_malt/jtl/m.toml`
			`_malt/jtl/m.out`

`Malt.toml`
`jtl.toml`
`jtl/2.toml`
`jtl.cir`
`_malt/jtl/the.envelope`
	`malt -m jtl/2`	`Malt.toml`
		`jtl.toml`
		`jtl/2.toml`
			`_malt/jtl/2/m.toml`
			`_malt/jtl/2/m.out`

`Malt.toml`
`jtl.toml`
`jtl/2.cir`
`_malt/jtl/the.envelope`
	`malt -m jtl/2`	`Malt.toml`
		`jtl.toml`
			`_malt/jtl/2/m.toml`
			`_malt/jtl/2/m.out`

Defining Correct Circuit Operation (`-d`)

With the `-d` switch, Malt simulates the circuit at nominal parameter values and generates envelopes for the waveforms that define the pass/fail criteria for later invocations of Malt (`-m`, `-o`, `-y`).

The pass/fail criteria for correct circuit operation is defined by **envelopes** on the waveforms for the listed nodes. An envelope is calculated by tracing the waveform with an ellipse centered on the simulated value. The dimensions of the ellipse can be configured in the `[envelope]` section of the applicable configuration file(s). See Appendix 1.

Example. Run in the `EXAMP/fluxshut` directory:

```
$  malt -d netlist
```

Note that

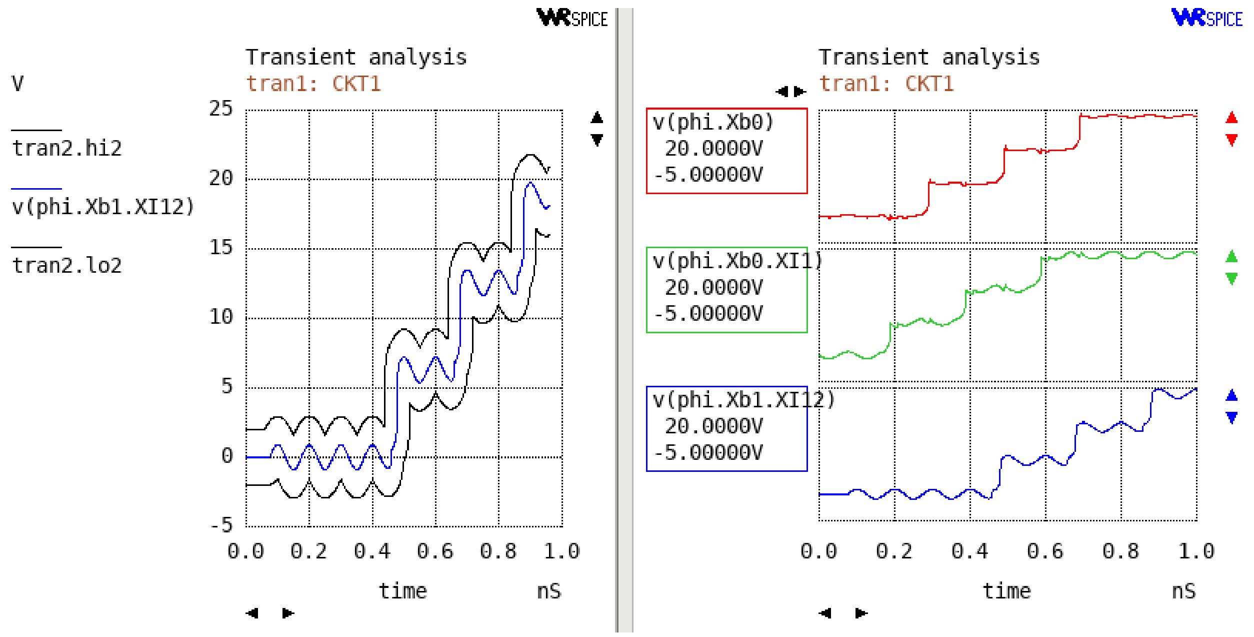


Figure 7. On the left, one node is plotted with its envelope (min and max values). The ellipse dimensions are chosen so that minor perturbations of the waveform are ignored, but a gross deviation indicating a bit error will be detected. On the right, all saved nodes are plotted at once.

Malt will automatically invoke WRspice to display the waveforms and envelopes for manual review. Type `quit` at the WRspice prompt when satisfied with the result.

The following supplementary output files will be created in the working tree (see “Other control files and generated files”):

```
the.envelope
d.plot
```

The first file contains the vector data, which will be used by later invocations of Malt that use the same configuration. The envelope file does not need to be manually edited. The second file is a WRspice script; invoke this from the WRspice command line to review the waveforms and envelopes.

Using a `.passf` file instead of envelopes to define pass/fail criteria

A hand-written file with a `.passf` extension can be used to define correct circuit operation in the event that the automatically-generated envelopes are inadequate. This file should be written in WRspice control-code syntax. It is intended to assess node values against some criteria, and set the flag `passf` to 1 or 0 accordingly, to indicate pass or fail.

Example. The files `netlist/ex1.passf` and `netlist/ex1.toml` illustrate dealing with the superconducting-phase waveform generated by the output amplifier. Run:

```
$   malt -d netlist/ex1
$   malt -m netlist/ex1
```

to compute margins including these criteria.

Note that only nodes listed in the `.toml` file(s) can be referenced by the `.passf` file, as no other nodes are saved.

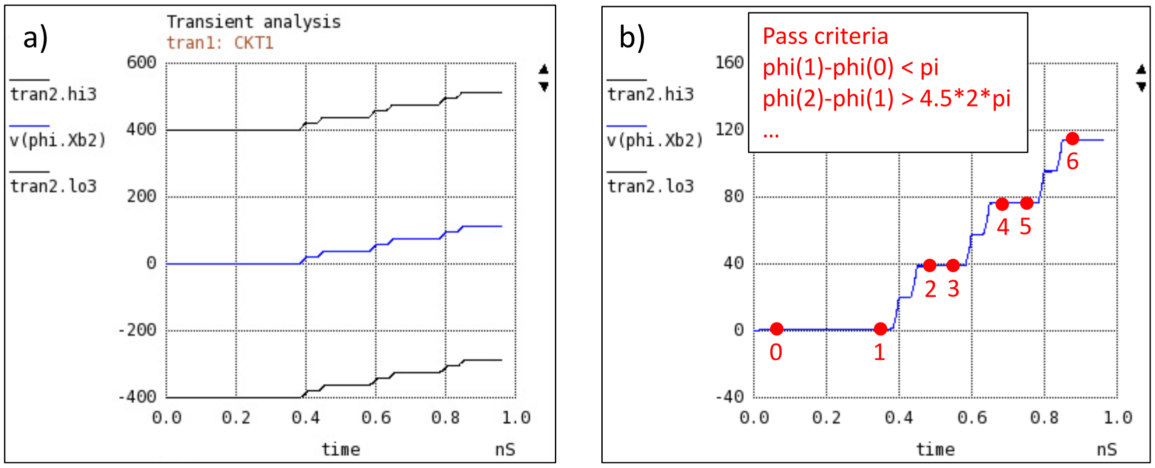


Figure 8. The phase (time-integral of voltage) waveform of the output (`phi.Xb2`) is not amenable to envelope definition, so an alternative approach is required. a) The amplitude tolerance on the envelope is set to values that cannot fail. b) The .passf file defines pass criteria in terms of minimum and maximum phase excursions between adjacent checkpoints.

Calculate Margins of Individual Parameters (`-m`)

Malt will calculate 1-dimensional parameter margins using binary search and the pass/fail criteria defined by the envelope and/or `.passf` file (see above).

All non-corner parameters that are marked as **included** in the `.toml` file will be margined.

Example. To take margins on all the included parameters in `netlist.toml`, run in the `EXAMP/fluxshut` directory:

```
$  malt -m netlist
```

Note that you should have already run `malt -d netlist`.

Corners in `-m`, `-y`, and `-o` Routines

Some circuit parameters may be marked in the `.toml` file as **corner parameters**. These are used to compute corners. The boundary of the operating region is always found across all corners. Figure 5 illustrates how corners affect the computation of the operating region. The corners are defined as every combination of minimum or maximum value for all corner parameters.

Example. To take margins across ±5% Xlcomp and ±5% Xjcomp corners, run:

```
$  malt -m netlist/ex2
```

Note that this command will still use the envelope generated by `malt -d netlist`. You do not have to run `malt -d netlist/ex2` to define a new envelope; the more general envelope is presumed to apply to more specific commands.

Trace Nodes at Marginal Parameter Values (`-t`)

This function is like `-m`, but also generates plots of each node on both sides of the operating boundary for each parameter. It is useful for debugging the envelope generation, the parameter margins, or both. The total number of plots is potentially large, because it scales as the product of the number of nodes and the number of included parameters.

Corner analysis does not work in `-t` mode.

The following output files will be created in the working tree (under `_malt/`), in addition to those generated by `-m` (see "Other control files and generated files"):

```
<param>_<max|min>.<pass|fail>
t.plot
```

The `.pass` and `.fail` files contain the vector data. The `t.plot` file is a WRspice script. Invoke the script from the WRspice command line to recreate the plots without re-running `malt -t`.

Example. Figure 9 is part of the output of the following command:

```
$  malt -t netlist
```

This demonstrates a failure at both the top and bottom margins of Xac. (Since not every node fails at the same time, some plots show failures only at one margin, or at neither.)

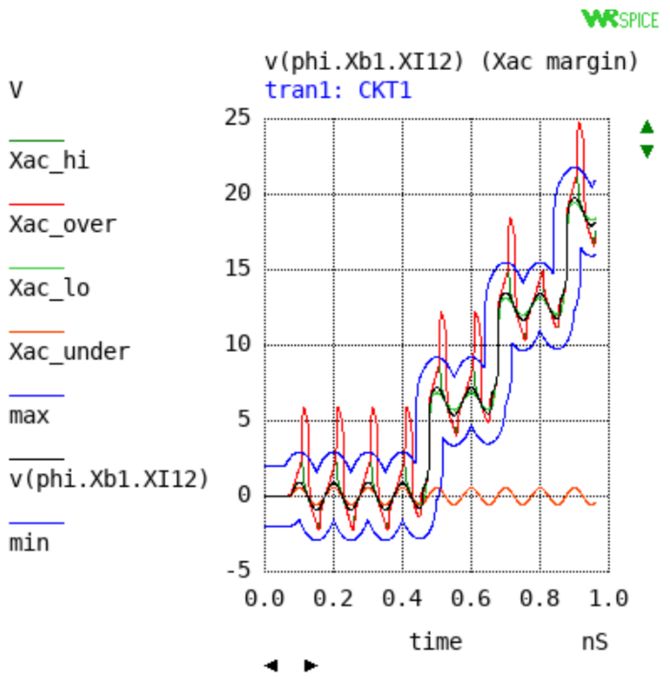


Figure 9. Example output of the trace routine showing passing/failing waveforms at extreme Xac. Note that some of the waveforms are hidden under the nominal (black) one.

Plot the Operating Region Boundary in 2 Dimensions (−2)

Malt will plot points on one or more two-dimensional slices of the operating boundary using binary search. Use the `[xy]` section of the configuration file to define what sweeps are of interest and how many points to generate for each.

The following supplementary output files will be created in the working tree (under `_malt/`):

```
2.txt
2.gpi
```

The first file contains the margin data, for internal use only. Malt will print the full path to the second file before exiting; it is a gnuplot script. Invoke this script from the gnuplot command line to re-generate the plots. You can edit the script to suit your tastes.

Example. Two-dimensional slices of the operating region are defined in `netlist/ex3.toml`. To produce the result, from a shell run:

```
$  malt -2 netlist/ex3
```

By default, each plot is displayed for 3 seconds. Malt will then print the full path of the Gnuplot script to the terminal, which you can use to re-invoke it.

```
$  gnuplot
>  load "_malt/netlist/ex3/2.gpi"
```

Consider editing the gnuplot script to make the plots more useful; `pause -1` may give better interactive results. Two-dimensional margins are a poor substitute for higher-dimensional analysis, but can be useful, particularly for judging the convexity of the operating region.

Calculate Yield Using Corner Analysis (`-y`)

The `-y` function evaluates parametric yield using an adaptive algorithm following an anneal schedule. This happens in two distinct steps.

Step 1. Malt will calculate between 2^N and 3^N points on the operating region boundary in N-space where N is the total number of included parameters. Corner parameters are set to the corner values and evaluated in parallel, while normal parameters are margined with a binary search. The number of points calculated and adaptivity of the algorithm can be fine-tuned by the settings in the `[yield]` section of the applicable configuration file(s).

Step 2. Continue adding points on the operating region boundary until the desired accuracy threshold is met. The accuracy target can be configured in in the `[yield]` section of the configuration file(s).

Optimize Yield Using Simplex Approximation (`-o`)

This function will attempt to center the included parameter values within a simplex that approximates the operating region. As above, corner parameters are evaluated at all of the corner values in parallel for the operating region search.

You can narrow the search range for parameters by setting the parameter's `nom_min` and `nom_max` in the configuration file. Other options for fine-tuning the optimization are listed in the `[optimize]` section of Appendix 1.

Additional Malt Features

The **.param file** can be used to define additional parameters that do mathematical operations on the parameters defined in the netlist. (The `.param` extension can be changed in the `.toml` configuration file.) This is demonstrated with the files `netlist/ex4.toml` and `netlist/ex4.param`. Run:

```
$  malt -m netlist/ex4
```

and compare the results to `netlist/ex2` as demonstrated above. This example happens to collapse four corners (two parameters high/low) down to two corners (one parameter high/low), and also happens to get the same result. In general, any parametric knob the mind can conceive could be achieved here. Use of a separate file is necessary because variables cannot be assigned using mathematical expressions within the `netlist.cir` file.

Appendix 1 Configuration File Syntax

Creating a Default `Malt.toml`

When run in a directory that is not part of a Malt project tree (that is, neither the current directory nor any parent contains a `Malt.toml` file), Malt will generate a new `Malt.toml` with default settings and brief explanatory comments.

Converting Legacy `.config` Files

Older versions of Malt used a bespoke text file format for configuration. These files use a `.config` file extension. You can use the `maltcfg2toml` script, found in the `scripts/` directory of the Malt repository, to convert `.config` files to TOML format. Comments are not preserved. For instructions, run:

```
$  maltcfg2toml -h
```

Introduction to TOML

TOML (Tom's Obvious, Minimal Language) is a simple format for configuration files, similar to the INI format. This section is a brief introduction to the TOML language.

Key-Value Pairs

Each setting is written as a key (the name of the setting), an equals sign, and a value. For example:

```
print_terminal = true
```

Keys must be quoted if they contain spaces or punctuation.

Types of values

Numbers in TOML can be integers or floating-point values. Scientific notation is allowed, as is the use of _ as a separator for long strings of digits.

```
max_mem_k = 4_194_304
binsearch_accuracy = 0.01
```

Strings, including paths, go in quotes. Double-quoted strings support the usual backslash-escapes, but single-quoted strings do not.

```
command = "/home/trent/bin/wrspice"
```

Booleans are represented by the keywords true and false. Note that where Malt expects a value to be boolean, you can use the integers 1 or 0 instead.

```
verbose = true
```

Lists of values can be written in square brackets and separated by commas.

```
targets = ["gain", "power", "area"]
```

Tables

Groups of settings are organized in sections called **tables**, marked with square brackets, similar to INI file sections.

```
[envelope]
dt = 1.0e−10
dx = 1.0
```

All the key-value pairs following a section marker are considered a part of that table until the next marker.

Tables can be nested with dots in the table name. This helps organize related settings hierarchically.

```
[nodes.'v(phi.node0)']
dt = 1.1e−10
dx = 2.0
```

This creates a table called `v(phi.node0)` inside the table named `nodes`. Note that table names, like keys, must be quoted if they use punctuation characters: this example creates a table named `v(phi.node0)`, not two tables named `'v(phi` and `node0)'`.

Inline Tables

TOML also supports **inline tables**, which let you define a small group of related key-value pairs all on one line. This can be useful for compact settings that belong together:

```
[nodes]
'v(phi.node1)' = { dt = 1.1e−10, dx = 2.0 }
```

This is precisely equivalent to the previous example. This reference will use inline tables only for parameters and nodes.

General Options

These settings affect Malt's behavior in multiple modes. They must appear at the beginning of the file, before any `[section]` header.

Setting	Type	Default	Description
`binsearch_accuracy`	float	`0.1`	Accuracy of the binary search. This number is given as a fraction of sigma for each parameter.
`print_terminal`	boolean	`true`	Echo output to the terminal as well as to the output file. See "Other control files and generated files".

`[simulator]`

These options affect how Malt calls WRspice.

Setting	Type	Default	Description
`max_subprocesses`	integer	`0`	The maximum number of concurrent simulator jobs Malt will try to run when doing corner analysis. The default, 0, means there is no limit.
`Command`	string	`wrspice`	Name or path to the WRspice executable. Malt will use the shell to invoke WRspice, so the name will be looked up in `$PATH` unless it contains a `/`.
`verbose`	boolean	`false`	Send all the messages generated by WRspice to the terminal, in addition to Malt's own output. This can generate a lot of text output, mostly useful for debugging.

[define]

These settings affect Malt's behavior when run with the `-d` flag.

Setting	Type	Default	Description
`simulate`	boolean	`true`	Simulate the circuit to find the nominal values. If false, the envelope will be calculated and plotted based on the nominal values from a previous run. This permits tweaking the ellipse parameters (`dx` and `dt`) without resimulating.
`envelope`	boolean	`true`	Save the envelope files for future runs. If false, the envelope will still be calculated and plotted in WRspice, but not saved to a `.envelope` file.

[envelope]

This section contains the default ellipse parameters used by `malt -d` when calculating pass/fail criteria. These can be overridden on a per-node basis in the `[nodes]` section, if desired.

Setting	Type	Default	Description
`dx`	float	`1.0`	Amplitude tolerance of the pass/fail envelope (vertical axis of the ellipse).
`dt`	float	`1e-10`	Time tolerance of the pass/fail envelope (horizontal axis of the ellipse).

[extensions]

This section allows the user to control which file extensions Malt uses to recognize or generate certain files. All values are strings. Some extensions of generated files, e.g. `.gpi` or `.pname`, cannot be set here.

Setting	Default	Description
`circuit`	`.cir`	Input circuit description in WRspice syntax.
`plot`	`.plot`	Generated WRspice control file that creates plots.
`parameters`	`.param`	Input parameters file. WRspice control syntax.
`passfail`	`.passf`	Input pass-fail criteria file. WRspice control syntax.
`envelope`	`.envelope`	Generated envelope file (upper and lower bounds on each node). Rawfile syntax.
`env_call`	`.env_call`	Generated WRspice control file that sources the envelope file. For internal use only.

[nodes]

This section contains the nodes that are used in the pass/fail criteria, whether by envelopes or `.passf`. See Defining Correct Circuit Operation (`-d`).

The value of each node may be a nested table, a boolean, or the integers 1 or 0. In the examples in this manual, nodes and parameters are represented by inline tables in TOML (see Inline Tables). For example:

```
[nodes]
"v(phi.Xb0)" = {}
"v(phi.Xb0.XI1)" = 1
```

Please note the following:

1. Node names should be quoted with single or double quotes, to avoid problems with embedded punctuation.
2. If the settings specified in `[envelope]` work for all or most nodes, which is not unusual, the nested table may be empty, as there are no non-default settings to list. In this case, the value of the node can be set to an empty table `{}` or to `1` or `true`, with the same effect. However, setting the value to `0` or `false` will cause Malt to ignore it.

The keys that may be specified for each parameter are the same as in the `[envelope]` section.

[parameters]

This section contains circuit parameters that are set by Malt. The value of each parameter is a nested table. For example:

```
[parameters]
b0 = { nominal = 0.08, min = 0.03, max = 0.3, sig_pct = 4.0, include = true }
l0 = { nominal = 5.0, min = 1.5, max = 15.0, sig_pct = 4.0, include = true }
```

As a shortcut, parameters that should be fixed to the nominal value (and not margined or used for corners) may be set to that value directly, e.g. `b1 = 0.018`.

The keys that may be specified for each parameter are detailed in the following table.

Parameter Setting	**Type**	**Default**	**Description**
`nominal`	float		Required. The nominal value of this parameter, used in `-d` and when finding 1-dimensional margins (`-m`) on other parameters.
`min, max`	float		Required for included parameters. For non-corner parameters, the lower and upper boundaries, respectively, of the binary search. For corner parameters, defines the corner values.
`nom_min, nom_max`	float	`see →`	Used in `-o` (optimize) mode to further constrain the optimized nominal value of this parameter. When not provided, the value is constrained only by `min` and `max`.
`sigma, sig_pct`	float	`0.0`	The 1σ spread (standard deviation) of the parameter, either as an absolute value (`sigma`) or as a percentage of nominal (`sig_pct`). Used in yield calculation and optimization and to display margins in σ. Either `sigma` or `sig_pct` must be nonzero for included, non-corner parameters.
`logs`	boolean	`true`	Use log space for the distribution of this parameter. For physical quantities and measurements like inductance, current, etc. this is normally what you want. See Figure 3.
`include`	boolean	`true`	Include this parameter in the analysis. For non-corner parameters, if true, this will be one of the parameters whose margins are found. For corner parameters, this parameter will be used to

			compute corners. When false, the parameter will be set to its nominal value during all simulations.
`corners`	boolean	`false`	Use min/max of this parameter to define corners, instead of including it in the binary search. When `include` is false, this setting has no effect.

[optimize]

This section contains settings that affect the `-o` function. In the `[parameters]` section, `nom_min` and `nom_max` also influence optimization by limiting the range of the optimized parameters. In the `[yield]` section, `accuracy` also has an effect, as described in the table below.

Setting	Type	Default	Description
`min_iter`	integer	`100`	Minimum number of iterations. Analysis will terminate after the last `min_iter` iterations fail to improve the margins by more than `yield.accuracy`.
`max_mem_k`	integer	`4194304`	Maximum amount of estimated memory that can be allocated during optimization. This may be an underestimate of true system memory usage.

[yield]

This section contains settings that affect Malt's behavior in -y mode. See Calculate Yield Using Corner Analysis (`-y`), earlier in this reference.

Setting	Type	Default	Description
`search_depth`	integer	`5`	Range: 0-10. This setting determines how many points will be generated in step 1 of yield analysis. When 0, 2^N points will be calculated; when 10, 3^N points.
`search_width`	integer	`5`	Range: 0-9. This setting affects how adaptive step 1 is. The routine initially partitions space into equal solid angles when `width` = 9, or into equal products of angle and function value if `width` = 0. A larger width value is more exploratory across space, and a smaller value is more adaptive based on existing information.
`search_steps`	integer	`12`	Range: 1-40. Step 1 becomes more adaptive in a certain number of steps. Inherently uninteresting.
`max_mem_k`	integer	`4194304`	Maximum amount of estimated memory that can be allocated in step 2. This may be an underestimate of true system memory usage.
`accuracy`	integer	`10`	Estimated percent accuracy in order to terminate step 2. This is also used in `-o` mode.
`print_every`	boolean	`false`	Print every iteration. When false (the default), print only the running most significant iteration.

[xy]

This section defines two-dimensional slices of the operating region for the `-2` function.

Setting	**Type**	**Default**	**Description**
`iterations`	integer	`32`	Number of points to plot for each two-dimensional search.
`sweeps`	list of tables		Each table in the list has keys `x` and `y`, which are the names of parameters. **Example:** `sweeps = [ { x = "Xl", y = "Xj" } ]` Sweeps may be defined in nested table form as above, or in an `[[xy.sweeps]]` table, per the TOML standard.

Appendix 2 Files Used in Examples

EXAMP/fluxshut/netlist.cir

```
* HNL Generated netlist of AA_fluxshut_top
* GLOBAL Parameters
.param amplitude = 0.700 frequency = 10G m_in = 0.1p
* GLOBAL Scaling Parameters
*.param Xac=1.0
*.param Xlcomp=1.0
*.param Xjcomp=1.0
.param Xl=1.0
.param Xj=1.0
*.param Xpdc=1.0
.param Xa=1.0
.param Xr=1.0
* Timing corner definitions
.param timingX = 1.0
.global 0
* Include Statements
.include my_models.lib
*Subcircuits
.subckt aa_fluxshut a i0 i1 q
XL0 a i0 rql_inductor_scale l=7.4e-12
Loff i0 i1 'm_in'
XL2 i1 net04 rql_inductor_scale l=1.47e-11
XL1 net04 q rql_inductor_scale l=7.4e-12
Xb0 i0 net023 rql_rsj_scale jjmod=rql25 ic=0.07 icrn=1
Xb1 net04 net08 rql_rsj_scale jjmod=rql25 ic=0.07 icrn=1
Lg0 net023 0 1e-12
Lg1 net08 0 1e-12
.ends aa_fluxshut
.subckt g_terminate i ic1=0.050
XR0 i 0 rql_resistor_scale r='0.7/ic1'
.ends g_terminate
*Input stage
L1 na 0 'm_in'
XL0 net017 na rql_inductor_scale l='1e-12*l0'
Xb0 net015 net017 rql_rsj_scale jjmod=rql25 ic='b0' icrn=1
*Flux shuttle stages
XI1 net015 i1a i1b net06 aa_fluxshut
XI2 net06 i2a i2b net07 aa_fluxshut
XI3 net07 i3a i3b net08 aa_fluxshut
XI4 net08 i4a i4b net05 aa_fluxshut
XI5 net05 i5a i5b net09 aa_fluxshut
XI6 net09 i6a i6b net010 aa_fluxshut
XI7 net010 i7a i7b net03 aa_fluxshut
XI8 net03 i8a i8b net029 aa_fluxshut
XI9 net029 i9a i9b net018 aa_fluxshut
XI10 net018 i10a i10b net025 aa_fluxshut
XI11 net025 i11a i11b net020 aa_fluxshut
XI12 net020 i12a i12b net02 aa_fluxshut
*flux shuttle terminationXb1 net02 net04 rql_rsj_scale jjmod=rql25 ic='b1' icrn=1
Lg1 net04 0 1e-12
XI13 net02 g_terminate ic1=0.07
*output amp
XL2 net031 net030 rql_inductor_scale l=1e-12
XL3 net032 0 rql_inductor_scale l=1e-12
XL4 net029 net026 rql_inductor_scale l='1e-12*l45'
XL5 net024 net025 rql_inductor_scale l='1e-12*l45'
XL6 net030 oo rql_inductor_scale l=4e-10
XR0 net030 oo rql_resistor_scale r=50
Xb2 net030 net032 rql_rsj_scale jjmod=rql25 ic='b2' \
icrn='1.0*b2*0.07/(b2*0.07+b34*0.040)'
Xb3 net026 net031 phib3 rql_junction_scale area='b34' jjmod=rql25
Xb4 net024 net031 phib4 rql_junction_scale area='b34' jjmod=rql25
*Input source
```

```
Ia 0 na 'phi0/m_in*inoff' +\
pulse(0 'phi0/m_in*inipp' 170ps 20ps 20ps 80ps 1) +\
pulse(0 'phi0/m_in*inipp' 370ps 20ps 20ps 80ps 1) +\
pulse(0 'phi0/m_in*inipp' 570ps 20ps 20ps 80ps 1) +\
0
*Four phase clock sources
I1 i1a i1b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 00ps)
I2 i2a i2b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 25ps)
I3 i3a i3b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 50ps)
I4 i4a i4b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 75ps)
I5 i5a i5b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 00ps)
I6 i6a i6b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 25ps)
I7 i7a i7b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 50ps)
I8 i8a i8b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 75ps)
I9 i9a i9b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 00ps)
I10 i10a i10b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 25ps)
I11 i11a i11b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 50ps)
I12 i12a i12b sin(0 'phi0/m_in*Xac*amplitude' 10GHz 75ps)
*Output source
Iout 0 oo pwl(0 0 20ps 'Xpdc*80uA')
Rout 0 oo 50
*Analysis definition
.tran 1ps 1000ps 0ps uic
.end
```

EXAMP/fluxshut/my_models.lib

```
* MODEL Declarations
*unshunted JJ with scalable parameters
.subckt rql_junction_scale PLUS MINUS PHI area=0.25 jjmod=rql25
b1 PLUS MINUS phi jjmod area='area*Xa*Xj*Xjcomp/timingX'
.ends rql_junction_scale
* rsj with scalable parameters
.subckt rql_rsj_scale PLUS MINUS jjmod=rql25 ic=0.25 icrn=0.7
Xb0 PLUS MINUS phi rql_junction_scale area='ic' jjmod='jjmod'
Xr0 PLUS MINUS rql_resistor_scale r='icrn/ic'
.ends rql_rsj_scale
* inductor models with scalable parameters
.subckt rql_inductor_scale PLUS MINUS l=1p
L0 PLUS MINUS 'l*Xl*Xlcomp/timingX'
.ends rql_inductor_scale
* resistor models with scalable parameters
.subckt rql_resistor_scale PLUS MINUS r=1
R0 PLUS MINUS 'r*Xr*timingX'
.ends rql_resistor_scale
*JJ models
.model rql25 jj(rtype=1,cct=1,icon=10m,vg=2.6m,delv=0.1m, icrit=1m,r0=40,rn=1.800,cap=0.70p)
* End MODEL Declarations
```

EXAMP/fluxshut/netlist.run

```
*netlist.run
.control
source netlist.nominals
source netlist.cir
run
plot \
v(phi.Xb0) \
v(phi.Xb0.XI1) \
v(phi.Xb1.XI1) \
v(phi.Xb0.XI12) \
v(phi.Xb1.XI12)
set noaskquit
.endc
```

EXAMP/fluxshut/netlist.nominals

```
*netlist
```

```
.control
*Parameter values
Xlcomp=1.0
Xjcomp=1.0
Xac =1.0
Xpdc =1.0
inoff =0.0
inipp =1.0
b0 =0.080
l0 =5.0
b1 =0.018
b2 =0.070
b34 =0.040
l45 =30.0
.endc
```

EXAMP/fluxshut/Malt.toml

```
### Malt Configuration File ###

# General Options
# (These options must precede any [section] in this file)
binsearch_accuracy = 0.1  # fraction of sigma
print_terminal = true

# Simulator options
[simulator]
max_subprocesses = 0
command = 'wrspice'
verbose = false

# File extensions used by Malt
[extensions]
circuit = '.cir'
parameters = '.param'
passfail = '.passf'
envelope = '.envelope'
env_call = '.env_call'
plot = '.plot'

# Options for defining correct circuit operation
[define]
simulate = true
envelope = true
```

EXAMP/fluxshut/netlist.toml

```
# Configuration for netlist.cir
[nodes]
"v(phi.Xb0)" = {}
"v(phi.Xb0.XI1)" = {}
"v(phi.Xb1.XI12)" = {}

[parameters]
Xlcomp = 1.0
Xjcomp = 1.0
Xac = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, sig_pct = 4.0 }
Xpdc = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, nom_min = 1.0, nom_max = 1.0, sig_pct
= 4.0 }
inoff = { nominal = 0.0, min = -1.0, max = 1.0, logs = false, sigma = 0.04 }
inipp = { nominal = 1.0, min = 0.0, max = 3.0, logs = false, sigma = 0.04 }
b0 = { nominal = 0.08, min = 0.03, max = 0.3, sig_pct = 4.0 }
l0 = { nominal = 5.0, min = 1.5, max = 15.0, sig_pct = 4.0 }
b1 = 0.018
b2 = 0.07
b34 = 0.04
l45 = 30.0
```

```
[envelope]
dx = 2.0
dt = 4e-11
```

EXAMP/fluxshut/netlist/ex1.toml

```
[nodes]
"v(phi.Xb0)" = {}
"v(phi.Xb0.XI1)" = {}
"v(phi.Xb1.XI12)" = {}
"v(phi.Xb2)" = { dx = 400.0 }

[parameters]
Xac = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, sig_pct = 4.0, include = true }
Xpdc = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, sig_pct = 4.0, include = true }
inoff = { nominal = 0.0, min = -1.0, max = 1.0, logs = false, sigma = 0.04, include = true }
inipp = { nominal = 1.0, min = 0.0, max = 3.0, logs = false, sigma = 0.04, include = true }
b0 = { nominal = 0.08, min = 0.03, max = 0.3, sig_pct = 4.0, include = true }
l0 = { nominal = 5.0, min = 1.5, max = 15.0, sig_pct = 4.0, include = true }
b1 = { nominal = 0.018, min = 0.006, max = 0.06, sig_pct = 4.0, include = true }
b2 = { nominal = 0.07, min = 0.02, max = 0.2, sig_pct = 4.0, include = true }
b34 = { nominal = 0.04, min = 0.012, max = 0.12, sig_pct = 4.0, include = true }
l45 = { nominal = 30.0, min = 10.0, max = 100.0, sig_pct = 4.0, include = true }

[envelope]
dx = 2.0
dt = 4e-11
```

EXAMP/fluxshut/netlist/ex1.passf

```
* first line is assumed to be a comment
.control
*zero indexed array. indexed with itime below
*units are ps as defined by the .tran line in the .cir file
compose myt values 90 344 490 544 690 744 890
* minimum phase advance when it is supposed to be on is 4.5*2*pi=28.3 rad
itime=2
dowhile itime < 7
jtime=itime-1
if (v(phi.Xb2)[$&myt[$&itime]]-v(phi.Xb2)[$&myt[$&jtime]]) < 28.3
echo Generated not enough phase in range $&myt[$&jtime] to $&myt[$&itime]
failed=1
end
itime=itime+2
end
* maximum phase advance when it is supposed to be off is 3 rad
itime=1
dowhile itime < 6
jtime=itime-1
if (v(phi.Xb2)[$&myt[$&itime]]-v(phi.Xb2)[$&myt[$&jtime]]) > 3
echo Generated too much phase in range $&myt[$&jtime] to $&myt[$&itime]
failed=1
end
itime=itime+2
end
.endc
```

EXAMP/fluxshut/netlist/ex2.toml

```
[parameters]
Xlcomp = { nominal = 1.0, min = 0.95, max = 1.05, corners = true, include = true }
Xjcomp = { nominal = 1.0, min = 0.95, max = 1.05, corners = true, include = true }
Xac = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, sig_pct = 4.0, include = true }
Xpdc = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, nom_min = 1.0, nom_max = 1.0, sig_pct
= 4.0, include = true }
inoff = { nominal = 0.0, min = -1.0, max = 1.0, logs = false, sigma = 0.04, include = true }
inipp = { nominal = 1.0, min = 0.0, max = 3.0, logs = false, sigma = 0.04, include = true }
b0 = { nominal = 0.08, min = 0.03, max = 0.3, sig_pct = 4.0, include = true }
```

```
l0 = { nominal = 5.0, min = 1.5, max = 15.0, sig_pct = 4.0, include = true }
b1 = { nominal = 0.018, min = 0.006, max = 0.06, sig_pct = 4.0, include = false }
b2 = { nominal = 0.07, min = 0.02, max = 0.2, sig_pct = 4.0, include = false }
b34 = { nominal = 0.04, min = 0.012, max = 0.12, sig_pct = 4.0, include = false }
l45 = { nominal = 30.0, min = 10.0, max = 100.0, sig_pct = 4.0, include = false }
```

EXAMP/fluxshut/netlist/ex3.toml

```
[parameters]
Xlcomp = { nominal = 1.0, min = 0.3, max = 3.0, sig_pct = 4.0}
Xjcomp = { nominal = 1.0, min = 0.3, max = 3.0, sig_pct = 4.0}

[xy]
iterations = 32
[[xy.sweeps]]
x = "Xlcomp"
y = "Xjcomp"

[[xy.sweeps]]
x = "inoff"
y = "inipp"
```

EXAMP/fluxshut/netlist/ex4.toml

```
[parameters]
Xljcomp = { nominal = 1.0, min = 0.95, max = 1.05, corners = true, include = true }
Xac = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, sig_pct = 4.0, include = true }
Xpdc = { nominal = 1.0, min = 0.3, max = 1.7, logs = false, nom_min = 1.0, nom_max = 1.0, sig_pct
= 4.0, include = true }
inoff = { nominal = 0.0, min = -1.0, max = 1.0, logs = false, sigma = 0.04, include = true }
inipp = { nominal = 1.0, min = 0.0, max = 3.0, logs = false, sigma = 0.04, include = true }
b0 = { nominal = 0.08, min = 0.03, max = 0.3, sig_pct = 4.0, include = true }
l0 = { nominal = 5.0, min = 1.5, max = 15.0, sig_pct = 4.0, include = true }
b1 = { nominal = 0.018, min = 0.006, max = 0.06, sig_pct = 4.0, include = false }
b2 = { nominal = 0.07, min = 0.02, max = 0.2, sig_pct = 4.0, include = false }
b34 = { nominal = 0.04, min = 0.012, max = 0.12, sig_pct = 4.0, include = false }
l45 = { nominal = 30.0, min = 10.0, max = 100.0, sig_pct = 4.0, include = false }
```

EXAMP/fluxshut/netlist/ex4.param

```
* define some parameters
.control
Xlcomp=Xljcomp
Xjcomp=Xljcomp
.endc
```