Malt 3.1 Methods and Meaning

*Every Option Defined! Nothing Left Out!*

Quentin Herr  
Trent Josephsen

7 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, , which is a generalization of the *complementary error function* . In N dimensions, given a circuit with the operating region defined by the hypersphere of radius ,

.

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

.

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 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 . These terms provide a numerical estimate for the contribution of each simplex:

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 expression above.) The number of orthants is large, equal to . 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

,

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:

where is the high or low physical parameter value and the sigma value. As defined above, nom is the nominal physical parameter value and 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 corner parameters, there are 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 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, , 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 .

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 function, and does not require additional circuit simulations. Yield as a function of sigma may be prove to be a good fit to the 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 1 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\_spice.mod

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 3.2

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 that .cir file extension is what Malt requires for netlist files. 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 Input and Output File Names, later in this reference, for more details.

To define correct circuit operation, from a shell run:

* malt –d netlist

Malt will

1. plot the nodes at nominal parameter values
2. plot the envelopes around the nodes that define correct circuit operation
3. exit to the WRspice command line so you can manipulate the plots.

To compute parametric margins, yield, and optimized parameter values, 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]](#footnote-1))
* Optional pass/fail control files (with a .passf extension1)

### 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

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, in the \_malt directory) and should normally not need to be edited.

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 |

## Define Correct Circuit Operation (-d)

Run the simulation at nominal parameter values, plot the node waveforms and generate envelopes for the waveforms that define correct circuit operation.

Node names that will be saved and monitored. The dt and dx can be included on the same line or listed separately.

* node=<string> , dt=<float>, dx=<float>

The pass/fail criteria for correct circuit operation is defined with envelopes on the waveforms for the listed nodes. The envelope is defined by tracing the waveform with an ellipse, whose time axis is defined by dt and whose amplitude axis is defined by dx. A single set of dt and dx parameters can be applied to all nodes if listed separately, in advance of the nodes.

Time-wise tolerance of the pass/fail envelope.

* dt=<float>

Amplitude-wise tolerance of the pass/fail envelope.

* dx=<float>

Run the nominal circuit simulation yes/no.

* d\_simulate=<int>

Normally you need to run the simulation to in order to construct the envelopes. However, if you want to re-construct the envelopes using the same waveforms (but presumably different dt and dx), you can skip the simulation.

Save the generated envelopes, yes/no.

* d\_envelope=<int>

Normally, yes, but if you want to try out new envelopes without overwriting the old ones, then no.

The following supplementary output files will be created.

* <circuit\_name>.envelope
* <circuit\_name>.plot.d

The first file contains the vector data, for internal use only. The second file is a WRspice script; invoke this from the WRspice command line to review the waveforms and envelopes.

### Alternative to define (.passf)

A hand-written file with a .passf extension can be used to define correct circuit operation instead of, or in conjunction with, the envelope definition. The file is 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.

## Calculate Individual Parameter Margins (-m)

Calculate individual parameter margins using a binary search. Uses the pass/fail criteria defined above. In all binary-search-based routines here and below, normal parameters are margined, while corner parameters are set to the corner values.

Parameters.

* param=<string>, nominal=<float>, min=<float>, max=<float>, sigma=<float>, sig\_abs=<float>, include=<int>, logs=<int>, corner=<int>
  + String is the parameter variable name.
  + Nominal is the nominal parameter value.
  + Max and Min are the upper and lower bounds on the binary search.
  + Sigma is the 1-sigma parameter spread as a percentage of nominal. Sig\_abs is given as an absolute value. Just one of the two must be set to non-zero.
  + Include: Include the parameter in the analysis yes/no. If no, set its value to nominal and leave it alone.
  + Corner: The parameter is set to the corner values (specified by sigma), as opposed to being included in the binary search, yes/no.
  + Logs: The parameter is defined as having its distribution in log-space, yes/no.

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

Generates plots for each node on both sides of the operating boundary for each parameter. Useful for debugging the envelope generation, the parameters margins, or both. The total number of plots is potentially large, scaling as the product of the number of nodes and the number of included parameters.

The options used are those listed for parameter margins analysis.

The following supplementary output files will be created.

* <circuit\_name>.<param>.<max|min>.<pass|fail>
* <circuit\_name>.plot.t

The first file contains the vector data, for internal use only. The second file is a WRspice script. Invoke the script from the WRspice command line to review the waveforms and envelopes.

## **Plot the Operating Region Boundary in 2 Dimensions** (-2)

Plot points on the operating boundary in two dimensions using binary search.

Parameters (see above).

* param=<string>, nominal=<float>, min=<float>, max=<float>, sigma=<float>, sig\_abs=<float>, include=<int>, corner=<int>, logs=<string>

Defines the two parameters for the 2D margin run. You can have multiple lines

* param\_x=<string>, param\_y=<string>

Number of points to generate that define the operating region

* 2D\_iter=<int>

The following supplementary output files will be created.

* netlist2.2
* netlist2.2gnu

The first file contains the margin data, for internal use only. The second file is a gnuplot script. Invoke the script from the gnuplot command line to generate the plots. You can edit the script to suit your tastes.

## Calculate Yield Using Corner Analysis (-y)

Evaluate parametric yield using an adaptive algorithm following an anneal schedule. 1) Calculate 2^N to 3^N margins in N-space where N is the total number of included parameters. Corner parameters are set to the corner values, while normal parameters are margined with a binary search. 2) Continue further until the desired accuracy is achieved. Control parameters are defined below.

Parameters.

* param=<string>, nominal=<float>, min=<float>, max=<float>, sigma=<float>, sig\_abs=<float>, include=<int>, corner=<int>, logs=<string>

Step (1) defined above proceeds for 2^N iterations if depth=0, and 3^N iterations if depth=10. Range: 0-10. Default: 5.

* y\_search\_depth=<int>

Step (1) defined above initially partitions space into equal solid angles if width=9, 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. Range: 0-9. Default: 5.

* y\_search\_width=<int>

Step (1) becomes more adaptive in a certain number of steps. Inherently uninteresting. Range: 1-40. Default: 12.

* y\_search steps=<int>

Maximum amount of estimated memory that can be allocated in step (2). This may be an underestimate of true system memory usage. Default: 4194304.

* y\_max\_mem\_k=<int>

Estimated percent accuracy to in order to terminate step (2). Default: 10.

* y\_accurancy=<int>

Print every iteration, else print only running most significant iteration. Default: 0.

* y\_print\_every=<int>

## Optimize Yield Using Simplex Approximation (-o)

Center the parameter values within a simplex that approximates the operating region. As above, normal parameters are margined, while corner parameters are set to the corner values. Options for controlling the iterations are listed below.

Parameters.

* param=<string>, nominal=<float>, min=<float>, max=<float>, sigma=<float>, sig\_abs=<float>, static=<int>, include=<int>, corner=<int>, logs=<string>, nom\_min=<float>, nom\_max=<float>

Limits on the optimized nominal parameter values.

* Nom\_max and nom\_min can be set to limit the range of the optimized parameter values.

Maximum number of iterations. Hard stop on this number.

* o\_max\_iter=<int>

Minimum number of iterations. Analysis will terminate after the last this-many iterations fail to improve the margins by more than y\_accuracy.

* o\_min\_iter=<int>

Maximum number of simplex facets. Hard stop on this number.

* o\_max\_planes=<int>

## More Malt Features

Additional features, in no particular order, are as follows.

Slices of the **operating region in two dimensions** are defined in netlist.1.config. To produce the result, from the linux command line type:

> malt –2 netlist.1

and then:

> gnuplot netlist.1.2gnu

to view the results. This file is a gnuplot script that can be edited to suit. In particular, changing the pause statements to “ pause -1” will 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.

Some **corner parameters** are defined in netlist.2.config. To recompute margins for this case, type:

> malt –m netlist.2

and compare the results to that above. The same can be done for yield and optimized parameter values.

A word about **input file precedence** is in order. In the above example, Malt first loads the configuration file netlist.config, then loads the file netlist.2.config, overwriting those fields that are redefined. The final configuration is documented in the file netlist.2.config.m. Malt also looks for the file netist.2.cir, but not finding it, looks for (and finds) the file netlist.cir. All this is described in greater detail in the next section.

The **.param file** can be used to define additional parameters that do mathematical operations on the parameters defined in the netlist. This is demonstrated with the files netlist.3.param and netlist.3.config. Type:

> malt –m netlist.3

and again compare the results to that above. This example happens to collapse a four parameter corners down to two corners, 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.

The trace routine is used to **visualize circuit behavior at the operating boundary** corresponding to individual parameter margins. For each parameter margin, the waveforms for each node are plotted just inside and just outside the operating boundary. Type:

> malt –t netlist.4

to generate the plots associated with the reduced set of parameters specified in netlist.4.config. To view the plots, from the WRspice prompt, type:

> netlist.4.plot.t

which is the generated script written in WRspice syntax. Delete the multiple \*.pass and \*.fail files that were generated, to reduce cruft when you are done.

Arbitrary criteria for correct circuit operation can be specified in the **.passf file** in the event that the automatically-generated envelopes are inadequate. An example is given in the files netlist.5.passf and netlist.5.config, dealing with the superconducting-phase waveform generated by the output amplifier. This is illustrated in Fig. 7. Type:

> malt –d netlist.5

> malt –m netlist.5

to compute margins including these new criteria. Note that only nodes listed in the .config can be referenced by the .passf file, as no other nodes are saved.



**Fig. 7.** The phase (time-integral of voltage) waveform of the output is not amenable to envelope definition. a) Instead, the amplitude tolerance on the envelope is set to values that cannot fail. b) The .passf file in this example defines pass criteria in terms of minimum and maximum phase excursions between adjacent checkpoints, as illustrated.

# Configuration File Syntax

## Options & Outputs

### General Options

Accuracy of the binary search.

* binsearch\_accuracy=<float>

This number is given as a fraction of sigma for each parameter.

Name of the WRspice executable.

* spice\_call\_name=<string>

Normally, “wrspice”.

Send all the messages generated by spice to the terminal, yes/no.

* verbose=<int>

Not particularly pleasant, but useful for debug.

Echo output to the terminal, yes/no.

* print\_terminal=<int>

Output is printed to the output file in any case.

# Files Used in Examples

EXAMP/AA\_fluxshut\_top\_4malt\_clean/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\_spice.mod

\*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 termination

Xb1 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/AA\_fluxshut\_top\_4malt\_clean/my\_models\_spice.mod

\* 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/AA\_fluxshut\_top\_4malt\_clean/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/AA\_fluxshut\_top\_4malt\_clean/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/AA\_fluxshut\_top\_4malt\_clean/netlist.config

\* Global parameters

param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=0

param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=0

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1, nom\_min=1, nom\_max=1

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min=-1.0, max=1.0, sig\_abs=0.04,logs=0, corners=0, include=1

param=inipp, nominal=1.00, min=0.0, max=3.0, sig\_abs=0.04,logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=1

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=1

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=0

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=0

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=0

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=0

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\*\*\* Options for Corners Yield \*\*\*

\*\*\* ranges: 0-10, 0-9, 1-40

y\_search\_depth = 5

y\_search\_width = 5

y\_search\_steps = 12

y\_max\_mem\_k = 4194304

y\_accuracy = 10

y\_print\_every = 0

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.1.config

\* Global parameters

param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=0, include=1

param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=0, include=1

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1, nom\_min=1, nom\_max=1

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min=-1.0, max=1.0, sig\_abs=0.04,logs=0, corners=0, include=1

param=inipp, nominal=1.00, min=0.0, max=3.0, sig\_abs=0.04,logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=1

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=1

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=0

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=0

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=0

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=0

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\*\*\* 2D Margins \*\*\*

2D\_iter = 32

param\_x = Xlcomp, param\_y = Xjcomp

param\_x = inoff, param\_y = inipp

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.2.config

\* Global parameters

param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1, nom\_min=1, nom\_max=1

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min=-1.0, max=1.0, sig\_abs=0.04,logs=0, corners=0, include=1

param=inipp, nominal=1.00, min=0.0, max=3.0, sig\_abs=0.04,logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=1

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=1

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=0

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=0

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=0

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=0

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\*\*\* Options for Corners Yield \*\*\*

\*\*\* ranges: 0-10, 0-9, 1-40

y\_search\_depth = 5

y\_search\_width = 5

y\_search\_steps = 12

y\_max\_mem\_k = 4194304

y\_accuracy = 10

y\_print\_every = 0

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.3.config

\* Global parameters

\*these are defined in the .param file now:

\*param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

\*param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xljcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min= -1.0, max=1.0, sig\_abs=0.04, logs=0, corners=0, include=1

param=inipp, nominal=1.00, min= 0.0, max=3.0, sig\_abs=0.04, logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=1

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=1

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=0

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=0

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=0

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=0

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.3.param

\* define some parameters

.control

Xlcomp=Xljcomp

Xjcomp=Xljcomp

.endc

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.4.config

\* Global parameters

param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=1

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=0

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=0

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min= -1.0, max=1.0, sig\_abs=0.04, logs=0, corners=0, include=1

param=inipp, nominal=1.00, min= 0.0, max=3.0, sig\_abs=0.04, logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=0

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=0

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=0

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=0

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=0

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=0

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.5.config

\* Global parameters

param=Xlcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=0

param=Xjcomp, nominal=1.0, min=0.30, max=3.0, sigma=4.0, logs=1, corners=1, include=0

param=Xac, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

param=Xpdc, nominal=1.0, min=0.30, max=1.7, sigma=4.0, logs=0, corners=0, include=1

\* Individual Variables

\*Input waveform offset and peak-to-peak amplitude in units of Phi0

param=inoff, nominal=0.00, min= -1.0, max=1.0, sig\_abs=0.04, logs=0, corners=0, include=1

param=inipp, nominal=1.00, min= 0.0, max=3.0, sig\_abs=0.04, logs=0, corners=0, include=1

\*Input parameters

param=b0, nominal=0.080, min=0.030, max=0.30, sigma=4.0, logs=1, corners=0, include=1

param=l0, nominal=5.00, min=1.50, max=15.0, sigma=4.0, logs=1, corners=0, include=1

\*Output parameter

param=b1, nominal=0.018, min=0.006, max=0.060, sigma=4.0, logs=1, corners=0, include=1

param=b2, nominal=0.070, min=0.020, max=0.200, sigma=4.0, logs=1, corners=0, include=1

param=b34, nominal=0.040, min=0.012, max=0.120, sigma=4.0, logs=1, corners=0, include=1

param=l45, nominal=30.0, min=10, max=100, sigma=4.0, logs=1, corners=0, include=1

\*\*\* General Option \*\*\*

binsearch\_accuracy=0.1

\* Nodes for the Passfail criterion

dx=2

dt=40e-12

node=v(phi.Xb0)

node=v(phi.Xb0.XI1)

node=v(phi.Xb1.XI12)

node=v(phi.Xb2), dx=400

EXAMP/AA\_fluxshut\_top\_4malt\_clean/netlist.5.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

1. The expected file extensions for netlists and control files can be changed in Malt.toml. [↑](#footnote-ref-1)