TimEx User's Manual

V2.05

Coenrad Fourie

15 May 2020

This work was supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

Table of Contents

Introduction and setup

Introduction

TimEx was developed under IARPA contracts FA8750-15-C-0203-IARPA-BAA-14-03 and SuperTools/ColdFlux (via the U.S. Army Research Office grant W911NF-17-1-0120), and is based on the research described in [1] and [2].

TimEx takes a JoSIM deck file as the first command line parameter and considers this as the Device-Under-Test (DUT). The DUT needs to be described as a subcircuit in the deck file, and input and output ports must be specified. *TimEx* then constructs a simulation test bench consisting of specified load cells at each input and output as well as specified source and sink cells. This is illustrated in Figure 1.

Through the variation of input sequences, all states and all input-to-output delays for the DUT are found. Critical Timing parameters and illegal inputs are then identified through iterative methods, and a Verilog model of the DUT is constructed that defines all states, output delay times and critical timing parameters. A Verilog test bench is also created to verify the operation of the DUT model.

TimEx also writes a .gv file (the DOT format) for viewing a Mealy Finite State Machine diagram of the DUT with *GraphViz*.

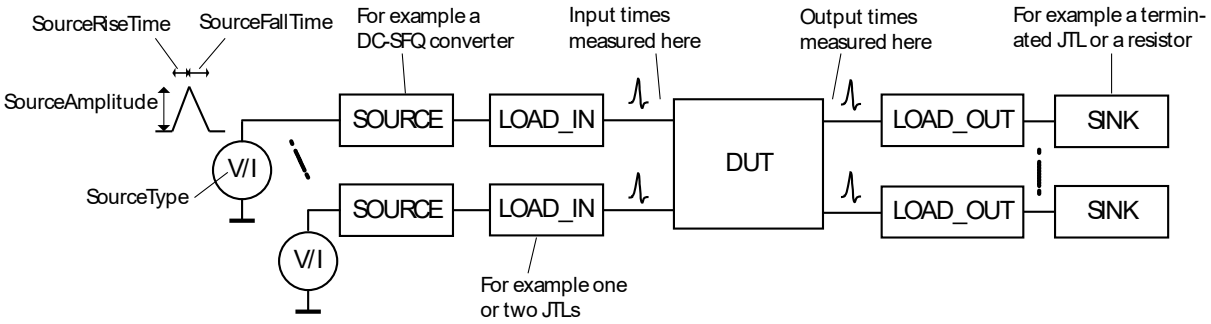


Figure 1: Schematic diagram to describe test bench setup around the Device-Under-Test in Timex.

Initial setup

TimEx is a console application, and requires no setup. The executable or binary files can be placed in any directory (accessible with path), while project files can be placed in a working directory.

TimEx relies on the light-weight superconducting circuit simulator *JoSIM* or *JSIM_n* to perform transient electrical simulations. Make sure that *JoSIM* or *JSIM_n* is in a folder on the path.

If *TimEx* is executed with the "-x" switch, you also need to have *iverilog*, *vvp* and *dot* (*GraphViz*) stored in a folder on the path.

Under MS Windows, the executables for *iverilog, vvp* and *jsim_n* are provided. You can install *dot* from the Graphviz website, www.graphviz.org, and *JoSIM* can be obtained from the GitHub at https://github.com/JoeyDelp/JoSIM.

For Linux, the binaries for *JSIM_n* and *TimEx* are provided. Install the other components (*iverilog*, *vvp* and *dot*) with:

$ sudo apt install iverilog

$ sudo apt install graphviz

This was tested under CentOS7 and Ubuntu 16.04 LTS (Xenial Xerus).

JoSIM can be built from source code, available at https://github.com/JoeyDelp/JoSIM.

License

TimEx v2 is free to distribute and/or modify under the terms of the MIT license.

Building the source code under Windows

Under Windows, open the file `TimEx.dproj` in the directory `src` with Embarcadero Delphi (this was tested with Delphi XE 5 and Rad Studio 10.1 Berlin).

In the Project Manager window (upper right in Figure 2), select the Release build configuration, and 32-bit Windows as the target platform.

The project can then be built from the Project menu. The binary files will be placed in `bin\Win32\Release`.

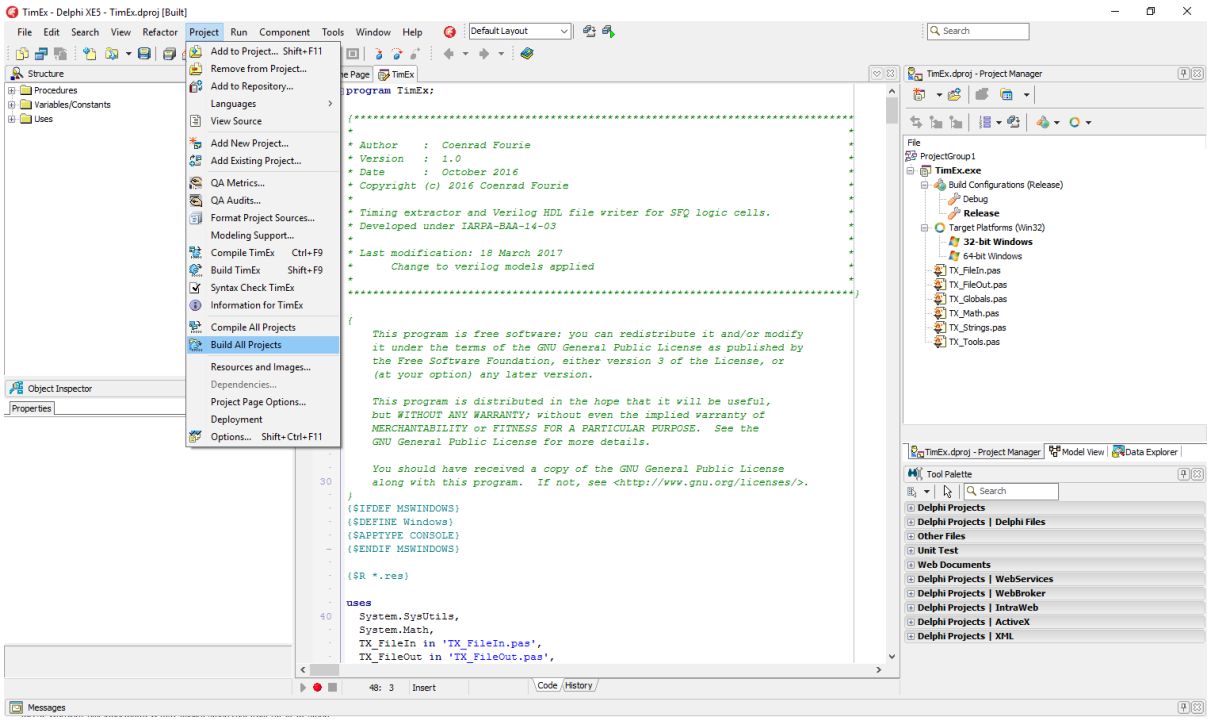


Figure 2: Building TimEx in Embarcadero Delphi XE5

The 64-bit executable for Windows is provided with the project files, under `bin\Win64\Release`.

Building the source code under Linux

Install freepascal and lazarus under CentOS7 Linux

Under CentOS7 Linux, the *freepascal* compiler and *lazarus* IDE need to be installed. From a web browser, access https://sourceforge.net/projects/lazarus/files/. Download the latest RPM files, which should be in the directory *Lazarus Linux x86_64 RPM* for a 64-bit Intel system. Download the *fpc*, *fpc-src* and *lazarus* packages.

Install these with:

```
$ sudo yum install gtk2-devel
$ sudo rpm –Uvh *.rpm
```

Install freepascal and lazarus under Ubuntu Linux

Under Linux Ubuntu, *freepascal* and *lazarus* can be installed from the terminal directly with:

```
$ sudo apt-get install fp-compiler
$ sudo apt-get install lazarus
```

Build TimEx

Open the *TimEx* project file with *lazarus*, as shown in Figure 3.

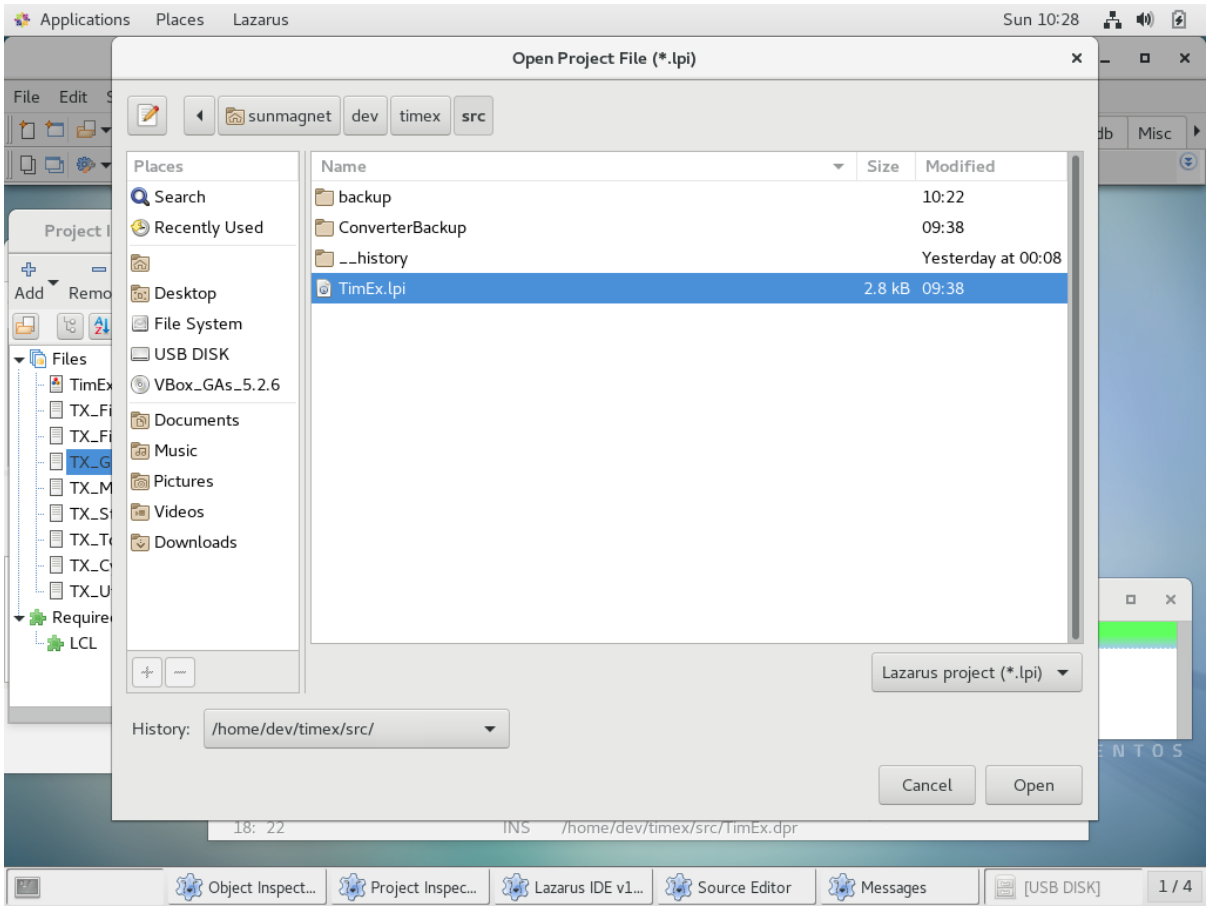


Figure 3: Opening the TimEx project file in lazarus under CentOS7 Linux.

Select build, as shown in Figure 4 to build the source code.

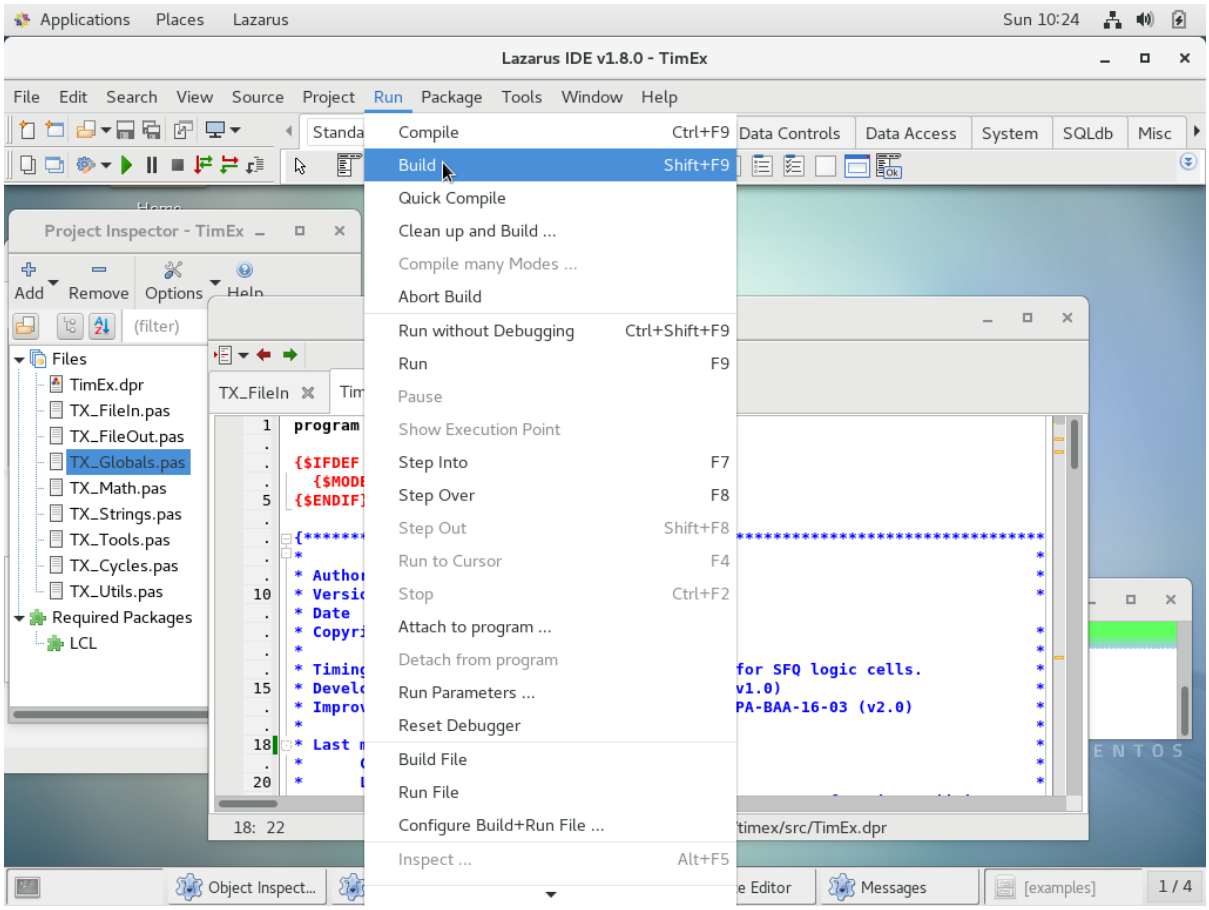


Figure 4: Building TimEx in lazarus under CentOS7 Linux.

Installation under Windows

In order to use *TimEx* fully, install the following components:

1. Download *Icarus Verilog* from http://bleyer.org/icarus/ and install.
2. Download *Graphviz* from http://www.graphviz.org/ and install:
3. Add the following directories to the system PATH variable:

   <TimEx_INSTALL_DIR>\bin\win

   <Icarus Verilog_INSTALL_DIR>\bin

   <Icarus Verilog_INSTALL_DIR>\gtkwave\bin

   <Graphviz_INSTALL_DIR>\bin

   where, <X_INSTALL_DIR> is the directory where X has been installed.

Technical discussion

Cycle identification and flux calculation

For the state of a cell to be investigated, the flux in every cycle (or loop / mesh) in the circuit must be evaluated as

$$\Phi_{\text{total}} = \sum_{n=1}^{N} I_n L_n \tag{1}$$

where I_n is the signed branch current and L_n is the branch inductance. The flux is expected to be -1, 0 or 1 times the magnetic flux quantum (Φ_0). An algorithm thus finds all the meshes in the circuit so that cycle flux can be calculated from the sum of branch currents multiplied by element inductances. For Josephson junctions, the total equivalent inductance L_{Jt} is used [2], where

$$L_{Jt} = L_{J0}\left(\frac{\sin^{-1}(i_J/I_C)}{i_J/I_C}\right) \tag{2}$$

and

$$L_{J0} = \frac{\Phi_0}{2\pi I_C}\,. \tag{3}$$

Resistive cycles do not store flux, so that any cycles that contain resistors are ignored. *TimEx* also disregards cycles that contain any input or output ports, as the assumption is that flux storage in the interconnect inductance between any load and input/output of the DUT represents unacceptable circuit behaviour.

Numerical errors and errors in the Josephson inductances lead to cycle flux results that are spread over the range of roughly $0.98\Phi_0$ to $1.02\Phi_0$. In the current version, *TimEx* then divides the cycle flux results by Φ_0 and rounds the values to the nearest integer (-1, 0 or 1), with the sign depending on the direction of flux through a cycle.

Pulse detection

Even though analyses of time-based electrical simulations of SFQ circuits mostly rely on the phase evolution over Josephson junctions, *TimEx* investigates the voltage pulses at the inputs and outputs of the DUT. This is necessitated by the Verilog descriptions, which require response to the arrival of signals at the DUT inputs (which are typically inductors) and for which signal arrival times at the outputs need to be specified.

A simple way to detect pulses is to look for the peak values in voltage vs time plots, but this is risky. Firstly, a cell might fail to switch correctly and just create a ripple in the output of which peaks might be mistaken for pulses. Secondly, measurement of the pulses between

the inductances of inputs/outputs to/from a load and the DUT often results in pulses that seem to oscillate, so that the peak value could shift by a few picoseconds depending on the inductances.

The voltage pulses transmitted between SFQ elements integrate to exactly one fluxon, so that a much more reliable way of detecting a pulse is to use a sliding numerical integrator (see Figure 5 [2].) When the area inside the sliding integrator is compared to a threshold (set as a fraction of Φ_0), the time at which the threshold is passed is a very stable way to characterise pulse arrival time.

In *TimEx*, the parameters SlidingIntegratorLength and PulseDetectThreshold set the length of the sliding integrator window (in time) and the integrated area threshold as a fraction of Φ_0 for pulse detection respectively.

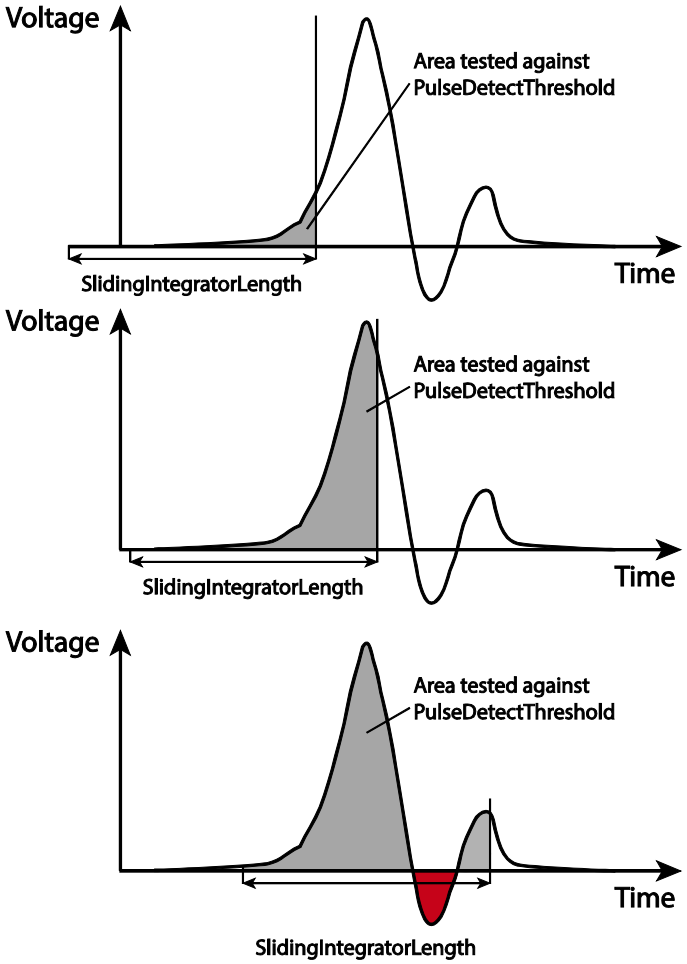


Figure 5: Pulse detection in *TimEx* with a sliding voltage-time integrator.

When pulses are transmitted over passive transmission line interconnects, the resistance in series with the PTL elements results in pulse areas lower than Φ_0. In order to accommodate

this, the minimum area for a successful pulse to be detected can be defined with the parameter PulseFluxonFraction. Such a reduced size output pulse can be seen in Figure 6(c) [2].

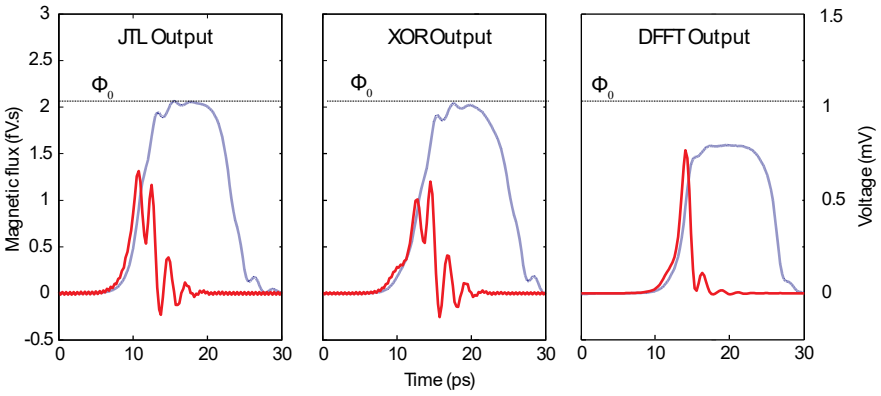


Figure 6: SFQ output pulses of (a) a JTL into a JTL load, (b) an RFSQ XOR gate into a JTL load and (c) an RSFQ DFF cell with integrated PTL driver into a matched transmission line (solid lines). Calculation results of a sliding window integrator with a window length of 12.5 ps are also shown (dashed lines).

Input files

Definition file

TimEx is configured and controlled through a definition file that can have any name.

The definition file defines parameters and optional load, sink or source netlists in demarcated blocks.

Table 1 shows the parameters available in the definition file, and what these control. The parameters are defined inside a control block starting with

```
$Parameters
```

and ending with

```
$End
```

Parameters are defined as:

```
Parametername = value
```

Table 1: Definition file parameters.

Parameter name	Default value	Function
CTDependencyThreshold	0.1E-12	Minimum binary search threshold for timing dependency between inputs. If a smaller critical timing dependency exists, it will be ignored.
InputChainDelay	7E-12	If the "-x" switch is used, a *JoSIM* simulation deck file is constructed with InputChainDelay subtracted from the time at which all inputs are applied. This accounts for the delay from the source element through the input load to the input of the DUT.
IOFullFluxon	true	Set true if pulses are full fluxons (all interconnects made with superconducting lines with no resistance).
MaxDelayChange	1E-12	The maximum time that the delay between an input and it's corresponding output may shift from nominal when critical timing is approached. Beyond this, an error condition is flagged.
MinSameInputSeparation	5E-12	The minimum time between separate pulses applied by the test bench to the same input when IOFullFluxon is false.

Parameter name	Default value	Function
NoiseTemperature	-1	The temperature at which noise spectral amplitude is calculated for resistors for noise simulations. At zero and below, noise analysis is ignored.
NumberSimsTolerance	0	The number of simulations with randomly distributed values and optional noise current sources from which the worst-case, mean and standard deviation of delay timing parameters are calculated. For values below 2, random variables and noise are ignored.
PulseDetectThreshold	0.5	The integration threshold (as a fraction of Φ_0) that indicates a detected pulse in a voltage-time curve.
PulseFluxonFraction	0.9	Minimum integration area (as a fraction of Φ_0) required to validate an SFQ pulse.
SimTimeStep	0.25E-12	Time step for *JoSIM* transient simulations.
SlidingIntegratorLength	10E-12	The time length of the sliding window over which $\int v \cdot dt$ is calculated to search for pulses.
SourceAmplitude	-	Amplitude of triangular input pulse at source element.
SourceFallTime	-	Fall time of triangular input pulse at source element.
SourceRiseTime	-	Rise time of triangular input pulse at source element.
SourceType	current	Source element type (current source or voltage source). Options are "current" and "voltage".
TimeFirstStable	10E-12	Time at which startup state is investigated in *JoSIM* simulation.
VerilogStableTime	10E-12	Time to first input in startup state for the Verilog test bench simulation model.
VerilogWaitTime	10E-12	Time between inputs for the Verilog test bench simulation model.
WaitForStateChange	50E-12	Time between inputs when state response to inputs are evaluated in *JoSIM*.

The definition file can contain optional definitions for the default source, load and sink netlist decks. These are loaded first, and can be replaced with optional netlist decks specified in the command line when *TimEx* is called.

The default input load (through which a pulse from a source is fed to an input in the DUT) must be a valid subcircuit and specified in a block starting with

```
$DefaultLoadIn
```

and ending with

```
$End
```

The default output load, which is connected to every output of the DUT, must be a valid subcircuit and specified in a block starting with

```
$DefaultLoadOut
```

The block is also terminated with the `$End` line.

The sink and source definition blocks are defined similarly, except for the block identifiers which are `$DefaultSource` and `$DefaultSink` respectively.

All netlist cards (lines) need to be less than 256 characters in length (the rest is truncated), and *TimEx* does not yet support line carry-over with the "+" identifier.

The definition file can also contain a control block, starting with

```
$Control
```

and ending with

```
$End
```

The sweep control is used to extract timing parameters that are functions of parameter variation. As an example, the bias of a DUT can be swept with the control

`Sweep` *variablename startvalue incrementvalue stopvalue*

This requires a variable named *variablename* to be defined in the DUT netlist file.

Examples are:

```
SWEEP bias 0.5 0.1 2.0
```

Although it is possible to specify multiple variables and multiple sweeps, the current version of *TimEx* only executes the first sweep in a netlist. This limit stems from the difficulty in setting up equations for timing parameters based on multiple parameters without incurring very long simulation times.

DUT netlist file

The DUT netlist file must be in the form of a subcircuit (with no components outside of the `.SUBCKT` to `.ENDS` block.

Very importantly, the netlist **must contain** a port definition card (anywhere in the file) of either the format:

```
*$Ports    in_anyname in_anyothername out_yetanothername
```

where

`*` identifies the card as a comment to *JoSIM* or *jsim*.

`$Ports` identifies the card as the port definition command to *TimEx*.

`in_` is the input identifier. Any text directly following it represents the input name.

`out_` is an output identifier. The output name follows directly;

or the format:

```
*$Ports   pin0 pin1 pin2 ...
```

where

`*` identifies the card as a comment to *JoSIM*.

`$Ports` identifies the card as the port definition command to *TimEx*.

Inputs are defined as any pin with the single character name `a` through `p` or the three-character name `clk`.

A single output is defined as `q`, or one of a multiple of outputs is defined as `q`*n* where *n* is a number starting at zero.

The first format is backwards compatible with earlier *TimEx* standards, while the second adheres to the SFQ logic pin name standard defined in the SuperTools ColdFlux cell library specification document.

All input and output names must be unique, but can have any order. The pin limit is set by the card length of 255 characters.

If a variable is swept, that variable must be declared as a parameter in the netlist:

`.PARAM` *variablename*=*nominalvalue*

The nominal value of a variable is the value that it is assigned when the nominal circuit is analysed to find all the states and state transitions. The variable name is not case sensitive.

A variable is invoked in the netlist by placing it inside braces. Expressions are evaluated.

For example:

```
IB1 0 3 pwl(0 0 5p {Bias*350e-6})
```

If `Bias` has a nominal value of 1.0, the bias current is nominally 350 μA. With the same variable, a bias voltage of 2.5 mV can be applied as:

```
VB1 4 0 pwl(0 0 5p {@Bias*2.5e-3})
```

Functions

Netlist files may contain functions in expressions. These functions are evaluated by *TimEx* before a netlist is sent to *JoSIM*. Currently only the following functions are supported:

GAUSS(*mean*, *stddev*)

This function calculates a normally distributed random variable (real) with mean *mean* and standard deviation *stddev*.

Command line parameters / switches

TimEx uses command line parameters and switches.

The first command line parameter must be the DUT subcircuit netlist file in *JoSIM* format.

Other parameters or switches can be supplied in any order.

`-a` – selects *jsim_n* as the simulation engine (default is *JoSIM*).

`-c` – generates a self-contained Verilog module.

`-d filename` – specifies the name of the *JoSIM* netlist file for the DUT subcircuit.

`-e filename` – name of the optional *JoSIM* netlist file for the source subcircuit.

`-L filename` – name of the optional *JoSIM* netlist file for the input load subcircuit.

`-l filename` – name of the optional *JoSIM* netlist file for the output load subcircuit.

`-o filename` – name of the optional text file to which the state map is written. The default name is `statemap.txt`.

`-s filename` – name of the optional *JoSIM* netlist file for the sink subcircuit.

`-v` – switch on verbose mode.

`-x` – Instructs *TimEx* to execute *JoSIM* or *jsim_n* on the simulation test bench for the DUT and write the output to a .dat file, execute *iverilog* and *vvp* on the Verilog test bench for the extracted HDL model of the DUT, and execute *dot* on the graph description (.gv file) for the Mealy FSM diagram of the DUT.

Output files

Self-contained Verilog model

TimEx creates a Verilog .v file from the characteristics extracted for the DUT, and places it in the working directory. With the `-c` command, this Verilog file contains all the necessary timing checks as part of the finite state machine definition.

The input and output names match those specified in the DUT netlist with the `*$Ports` command.

A real variable section is defined that contains all the timing values for easy lookup. Naming convention is standardised to allow automated readout by a suitably modified Static Timing Analysis tool.

The real variables are defined as:

`type_statenr_terminal1_terminal2`

where

`type` = {`delay | ct`} (in-to-out delay time | critical timing)

`statenr` = the number of the state in sequence of extraction, with 0 always the startup state

`terminal1` = the name of an input

`terminal2` = the name of an output (for delay timing) or another input (for critical timing.)

Timing values are in picoseconds, and represent either the delay from the arrival of an input pulse to the appearance of a corresponding output pulse (delay) or the critical timing value (minimum allowable separation) between an input and another input.

If a timing error or an illegal state is detected, all the outputs are changed to undefined.

An example of an RSFQ D Flip-Flop is shown in Figure 7.

```
timescale 1ps/100fs
module mitll_dff (set, reset, out);
input
  set, re;
output
  out;
reg
  out;
real
  delay_state1_reset_out = 5.0,
  ct_state0_reset_set = 0.2,
  ct_state1_reset_set = 1.0;
reg
   errorsignal_set,
   errorsignal_reset;
integer
   outfile,
   cell_state; // internal state of the cell
initial
   begin
     errorsignal_set = 0;
     errorsignal_reset = 0;
     cell_state = 0; // Startup state
     out = 0; // All outputs start at 0
   end
always @(posedge set or negedge set) // execute at pos and neg edges of input
   begin
     if ($time>4) // arbitrary steady-state time)
       begin
         if (errorsignal_set == 1'b1)  // A critical timing is active for this input
           begin
             outfile = $fopen("errors.txt", "a");
             $fdisplay(outfile, "Violation of CT in module %m; %0d ps.\n", $stime);
             $fclose(outfile);
             out <= 1'bX;  // Set all outputs to unknown
           end
         if (errorsignal_set == 0)
           begin
             case (cell_state)
               0: begin
                    cell_state = 1;  // Blocking statement -- immediately
                  end
             endcase
           end
       end
   end
always @(posedge reset or negedge reset) // execute at pos and neg edges of input
   begin
     if ($time>4) // arbitrary steady-state time)
       begin
         if (errorsignal_clk == 1'b1)  // A critical timing is active for this input
           begin
             outfile = $fopen("errors.txt", "a");
             $fdisplay(outfile, "Violation of CT in module %m; %0d ps.\n", $stime);
             $fclose(outfile);
             out <= 1'bX;  // Set all outputs to unknown
           end
         if (errorsignal_clk == 0)
           begin
             case (cell_state)
               0: begin
                    errorsignal_set = 1;  // Critical timing on this input; assign now
                    errorsignal_set <= #(ct_state0_reset_set) 0;  // Clear later
                  end
               1: begin
                    out <= #(delay_state1_reset_out) !out;
                    cell_state = 0;  // Blocking statement -- immediately
                    errorsignal_set = 1;  // Critical timing on this input; assign now
                    errorsignal_set <= #(ct_state1_reset_set) 0;  // Clear later
                  end
             endcase
           end
       end
   end
endmodule
```

Figure 7: Functional Verilog model of an RSFQ DFF extracted with TimEx.

Verilog module with SDF timing parameters

The default output from *TimEx* is a Verilog module that relies on SDF timing parameters for the generic support of commercial Verilog simulators. The Verilog module for the RSFQ D Flip-Flop discussed above is shown in Figure 8.

```
`ifndef begin_time
`define begin_time 8
`endif
`timescale 1ps/100fs
`celldefine
module mitll_dff #(parameter begin_time = `begin_time) (set, reset, out);
input
  set, reset;
output
  out;
reg
  internal_out;
assign out = internal_out;
integer state;
wire
  internal_state_0, internal_state_1;
assign internal_state_0 = state === 0;
assign internal_state_1 = state === 1;
specify
  specparam delay_state1_reset_out = (4.6:5.1:5.5);  // Mean = 5.093  StdDev =0.286
  specparam ct_state0_reset_set = 0.3;
  specparam ct_state1_reset_set = 1.1;
  if (internal_state_1) (reset => out) = delay_state1_reset_out;
  $hold( posedge reset &&& internal_state_0, set, ct_state0_reset_set);
  $hold( negedge reset &&& internal_state_0, set, ct_state0_reset_set);
  $hold( posedge reset &&& internal_state_1, set, ct_state1_reset_set);
  $hold( negedge reset &&& internal_state_1, set, ct_state1_reset_set);
endspecify
initial begin
   state = 1'bX;
   internal_out = 0; // All outputs start at 0
   #begin_time state = 0;
   end
always @(posedge set or negedge set)
case (state)
   0: begin
      state = 1;
   end
   1: begin
   end
endcase
always @(posedge reset or negedge reset)
case (state)
   0: begin
   end
   1: begin
      internal_out = !internal_out;
      state = 0;
   end
endcase
endmodule
```

Figure 8: Verilog model of an RSFQ DFF extracted with TimEx, with timing described in SDF file.

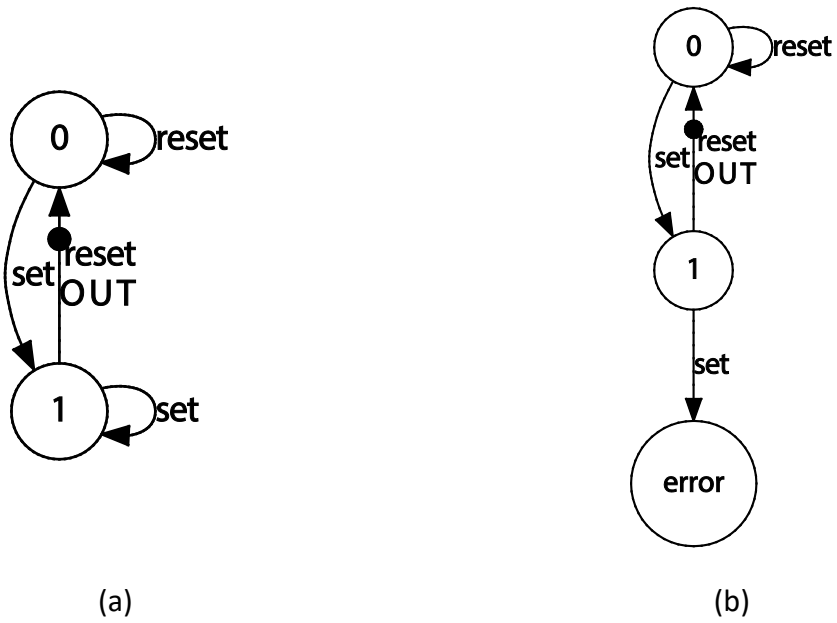
The SDF file holds the timing values, and is shown in Figure 9.

```
(DELAYFILE
  (SDFVERSION "4.0")
  (DESIGN "tb_mitll_dff")
  (DATE "2018/08/11 09:20:21")
  (VENDOR "ColdFlux")
  (PROGRAM "TimEx")
  (VERSION "2.03.00")
  (DIVIDER .)
  (PROCESS "typical")
  (TEMPERATURE 4:1:5)
  (TIMESCALE 100fs)
  (CELL
    (CELLTYPE "mitll_dff")
    (INSTANCE *)
    (DELAY
      (ABSOLUTE
        (COND internal_state_1
          (IOPATH reset out (46:51:55))
        )
      )
    )
    (TIMINGCHECK
      (HOLD set (COND internal_state_0 (posedge reset)) (3))
      (HOLD set (COND internal_state_0 (negedge reset)) (3))
      (HOLD set (COND internal_state_1 (posedge reset)) (11))
      (HOLD set (COND internal_state_1 (negedge reset)) (11))
    )
  )
)
```

Figure 9: Standard Delay Format file of an RSFQ DFF extracted with TimEx.

Mealy finite state machine representation

TimEx generates Mealy finite state machine diagrams for the cells which are extracted. This can be used to visualise all states and state transitions. A solid circle on a line between state transitions represents an output pulse. Examples are shown in Figure 10.



(a) (b)

Figure 10: (a) Mealy diagram of an RSFQ DFF with an input buffer junction at SET, and (b) Mealy diagram of an RSFQ DFF without an input buffer junction at SET and an error state resulting from a SET input in State 1.

Examples

Extraction of an RSFQ OR gate

A set of examples are included with the *TimEx* distribution package. At the start, this includes a JTL, two DFF circuits (one with an input buffer junction at the SET input, and the other without), an OR gate and an AND gate.

The examples can be executed with the `examples/runexamples.bat` script under Windows.

Linux examples are packaged in a compressed archive file: `examples/linux_examples.tar.gz`. The batch script is `runexamples.sh`.

Consider the example of an RSFQ OR gate, of which the schematic is shown in Figure 11.

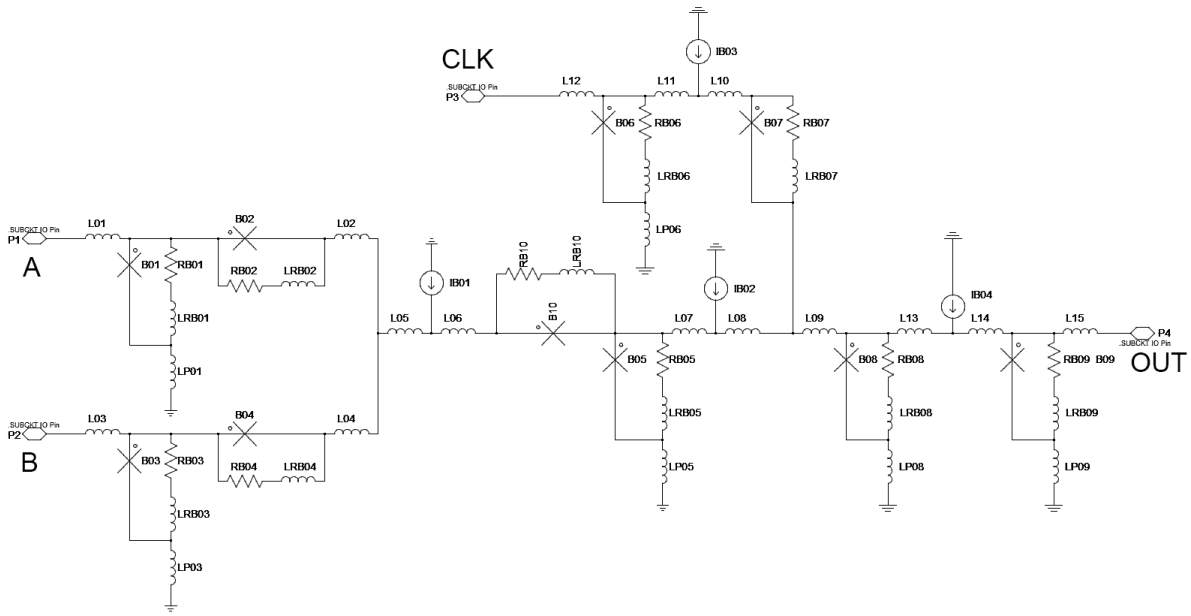


Figure 11: RSFQ OR gate schematic.

When *TimEx* is executed on the netlist file, the result is:

```
C:\usr\local\bin\examples>TimEx .\mitll_or\mitll_or.js -d
.\definitions\definitions_niobias.txt -x
TimEx v2.02.00 (10 Mar 2018). Copyright 2016-2018 Coenrad Fourie, Stellenbosch
University

This program comes with ABSOLUTELY NOE WARRANTY.

Definition file read.
Deck for Device-Under-Test read.

Finding all cycles.
Cycles:
[lp06,-b06,l11,l10,b07,l09,b08,-lp08]
```

```
[lp06,-b06,l11,l10,b07,l09,l13,l14,b09,-lp09]
[lp06,-b06,l11,l10,b07,-l08,-l07,b05,lp05]
[lp06,-b06,l11,l10,b07,-l08,-l07,-b10,-l06,-l05,-l02,-b02,b01,lp01]
[lp06,-b06,l11,l10,b07,-l08,-l07,-b10,-l06,-l05,-l04,-b04,b03,lp03]
[-lp01,-b01,b02,l02,-l04,-b04,b03,lp03]
[-lp01,-b01,b02,l02,l05,l06,b10,l07,l08,l09,b08,-lp08]
[-lp01,-b01,b02,l02,l05,l06,b10,l07,l08,l09,l13,l14,b09,-lp09]
[-lp01,-b01,b02,l02,l05,l06,b10,b05,lp05]
[-lp03,-b03,b04,l04,l05,l06,b10,l07,l08,l09,b08,-lp08]
[-lp03,-b03,b04,l04,l05,l06,b10,l07,l08,l09,l13,l14,b09,-lp09]
[-lp03,-b03,b04,l04,l05,l06,b10,b05,lp05]
[-lp05,-b05,l07,l08,l09,b08,-lp08]
[-lp05,-b05,l07,l08,l09,l13,l14,b09,-lp09]
[lp08,-b08,l13,l14,b09,-lp09]

.\mitll_or\mitll_or.js: Finding all states.
State 1: Input "clk" -> Output "out" after 7.25E-12 s.
States found: 2
.....State 0: No critical timing found a->a. (5 iterations.)
..........State 0: No critical timing found a->b. (10 iterations.)
.......xxxState 0: Critical timing found a->clk: 1.25E-12 s.
..........State 0: No critical timing found b->a. (10 iterations.)
.....State 0: No critical timing found b->b. (5 iterations.)
.......xxxState 0: Critical timing found b->clk: 1.25E-12 s.
..........State 0: No critical timing found clk->a. (10 iterations.)
..........State 0: No critical timing found clk->b. (10 iterations.)
.....State 0: No critical timing found clk->clk. (5 iterations.)
.....State 1: No critical timing found a->a. (5 iterations.)
..........State 1: No critical timing found a->b. (10 iterations.)
....xx.x.xState 1: Critical timing found a->clk: 4.219E-12 s.
..........State 1: No critical timing found b->a. (10 iterations.)
.....State 1: No critical timing found b->b. (5 iterations.)
....xx.x.xState 1: Critical timing found b->clk: 4.219E-12 s.
..........State 1: No critical timing found clk->a. (10 iterations.)
..........State 1: No critical timing found clk->b. (10 iterations.)
.....State 1: No critical timing found clk->clk. (5 iterations.)

Writing Verilog and testbench files.

Executing testbench simulations.
```

The cycles are listed. Note that a component such as L08 (first cycle) is connected with its positive node to ground in the netlist file, hence the negative sign in the cycle list.

TimEx identifies 2 states, and only one way to generate an output pulse – when an input is applied to "clk" in State 1. The clk->out delay is 7.25 ps. Some critical timing relationships are also found. The Mealy state diagram is shown in Figure 12. Note that an input to "clk" in State 1 results in an output pulse to "out".

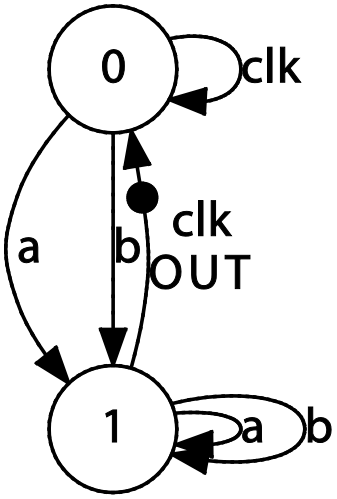


Figure 12: Mealy diagram extracted for the RSFQ OR gate.

The test bench generated by *TimEx* is simulated in both *JoSIM* and *iverilog/vvp*, and the results are shown in Figure 13. Note that *TimEx* Verilog models represent an input or output pulse as a low-high or high-low transition, which is easier to view for longer simulations. The pulse positions match, as expected, but note that the *JoSIM* output pulse "out" integrates to the detection threshold somewhere between its two peaks. Simple "largest peak" detection would lock on to the second peak, causing a timing miscalculation.

The error signals (used to test for critical timing violations) are also shown.

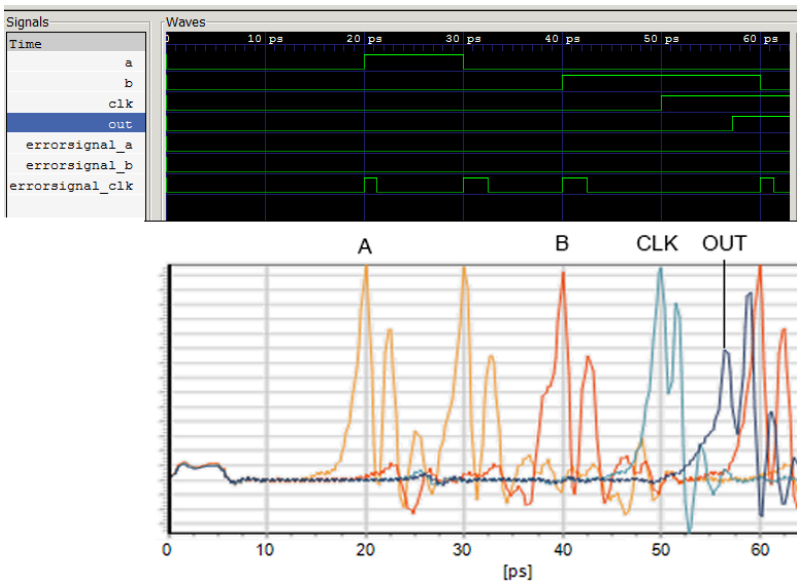


Figure 13: Verilog (top) and *jsim* (bottom) simulations of the RSFQ OR gate test bench.

If the "clk" input is moved to 1 ps after the first input on "b", a critical timing violation results in an undefined output at "out", as shown in Figure 14.

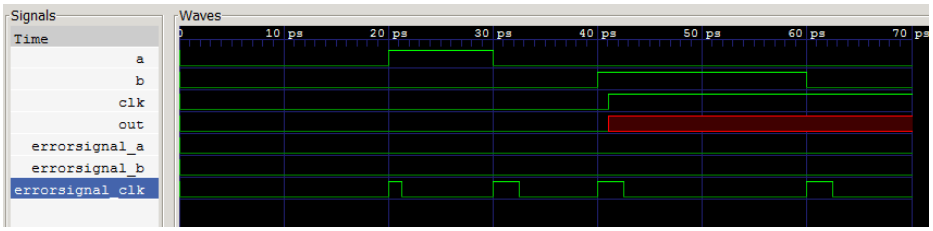


Figure 14: Verilog simulation of RSFQ OR gate with critical timing violation.

Bias sweep of RSFQ JTL

The state-dependent timing parameters of RSFQ circuits are influenced by other parameters such as circuit bias level, remanent magnetic field and output loads. In order to incorporate such influences in the Verilog HDL model of an RSFQ cell, TimEx supports swept variables.

Although *jsim* does not support SPICE parameter declarations through the .PARAM card, the TimEx preprocessor evaluates .PARAM cards and expressions in the netlist to create compatibility between *JoSIM* and *jsim*. In order to allow a sweep of the bias, it must be declared as a parameter with a nominal value.

The use of a swept variable is demonstrated here for a JTL. The netlist file is:

```
* Variables
.PARAM bias=2.5
.PARAM Vc = 1.0
*$Ports  in_in  out_out
.SUBCKT jtl 2  5
B1  1 6 jjmitll100 area=2.5
B2  4 8 jjmitll100 area=2.5
IB1 0 3 pwl(0 0 5p {bias*1e-3/7.14})
L1  2 1 2p
L2  1 3 2p
L3  3 4 2p
L4  4 5 2p
LB1 7 6 1p
LB2 9 8 1p
Lp1 6 0 0.2p
Lp2 8 0 0.2p
RB1 1 7 {2.74*Vc}
RB2 4 9 {2.74*Vc}
.model jjmitll100 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
.ends jtl
```

A control block in the definitions file controls the sweep:

```
$Control
Sweep bias  1.5  0.25  4.5  // Sweep "bias" from 2.5 to 4.5 in steps of 0.25
$End
```

The resulting Verilog model is:

```
`timescale 1ps/100fs
module jtl_vc100 (in, out);
input
  in;
output
  out;
reg
  out;
parameter
  bias = 2.5;
real
  delay_state0_in_out = 4.0,  // Nominal value at bias=2.5
  ct_state0_in_in = 3.3;  // Nominal value at bias=2.5
reg
   errorsignal_in;
integer
   outfile,
   cell_state; // internal state of the cell
initial
   begin
      errorsignal_in = 0;
      cell_state = 0; // Startup state
      out = 0; // All outputs start at 0
    if (bias < 1.5)
      begin
        out <= 1'bX;
      end
    if ((bias >= 1.5) && (bias < 1.75))
      begin
        delay_state0_in_out = 6.0 + (-0.5/0.3)*(bias-1.5);
        ct_state0_in_in = 6.6 + (-3.4/0.3)*(bias-1.5);
      end
    if ((bias >= 1.75) && (bias < 2))
      begin
        delay_state0_in_out = 5.5 + (-0.8/0.3)*(bias-1.8);
        ct_state0_in_in = 3.3 + (0.0/0.3)*(bias-1.8);  // Testbench limit
      end
...
    if (bias > 4.25)
      begin
        out <= 1'bX;
      end
   end
...
```

The extracted timing delay of JTL as a function of bias voltage and a parameter that sets the junction damping coefficient is shown in Figure 15.

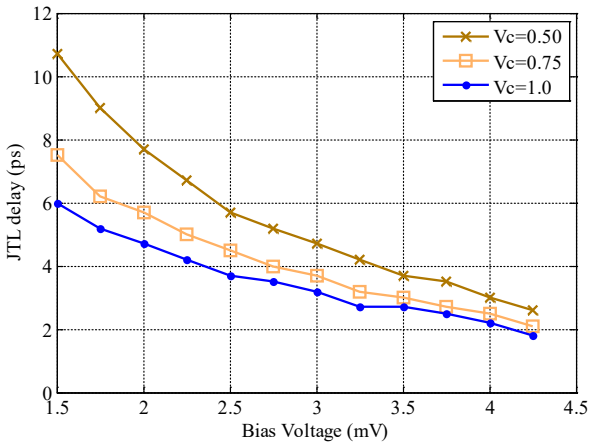


Figure 15: Extracted delay of a JTL with nominal I_C = 250 µA in the MIT Lincoln Laboratory 10 kA/cm2 process as a function of applied bias voltage and characteristic voltage V_C.

Process tolerance and noise

The effects of fabrication process tolerances and noise can be included during timing extraction. For process tolerances, both global (wafer-to-wafer and chip-to-chip) variations and local (component-to-component) tolerances can be modelled with random functions. In this version of *TimEx*, only Gaussian random variables are supported.

Process tolerances are added as shown below:

```
.PARAM bias=1.0
.PARAM globalb={gauss(1,0.03)}
.PARAM globall={gauss(1,0.05)}
.PARAM globali={gauss(1,0.06)}
.PARAM globalr={gauss(1,0.07)}
*$Ports        in_set in_reset out_out
.SUBCKT  dff   19     25       23
B1    1    2    jjmitll100 area={2.5*globalb*gauss(1,0.05)}
B2    10   16   jjmitll100 area={2*globalb*gauss(1,0.05)}
…
IB1   0    20   pwl(0 0 5p {260uA*bias*globali*gauss(1,0.1)})
…
L1    19   1    {2p*globall*gauss(1,0.05)}
RB1   1    12   {2.74*globalr*gauss(1,0.1)}
…
```

Here, the global variations are defined with .PARAM, while local variations are assigned when element values are defined.

Noise is added when a non-zero temperature is specified in the definition file.

```
NoiseTemperature = 4.2
```

When a noise temperature is specified, *TimEx* adds noise current sources in parallel with all resistors in the DUT.

TimEx uses the random variations and thermal noise to calculate delay times, but not critical times. The latter requires too many simulations at present.

The number of simulations from which delay times are calculated, is set by

`NumberSimsTolerance =` ***integervalue***

After completion of all simulations, the worst-case timing values for every delay are written to the Verilog model. Comments at every delay time list the calculated mean and standard deviations (assuming Gaussian distribution) for post-processing. An example result is:

```
real
  delay_state1_reset_out = 6.8,  // Mean = 5.198  StdDev = 0.680
  ct_state0_reset_set = 0.2,
  ct_state1_reset_set = 1.0;
```

Limitations and restrictions

Due to the limitations of other software such as *jsim*, and syntax restrictions in *JoSIM*, some restrictions apply to input files for TimEx:

The underscore ('_') character cannot be used in element names, as this breaks the *jsim* print command when the element is in a subcircuit.

The Gauss random function does not currently allow expressions, only real numbers. For example, `gauss(1,0.1)` is valid, but `gauss(somevariable_standard, 0.05*some_other_var)` is not allowed.

The *WRSpice* junction syntax (with a third node for phase measurement) is not supported, as it does not conform to the two-node model used by *JoSIM* and *jsim*.

Errors and exception handling

The current version of *TimEx* contains basic exception handling that is reasonably robust for file input errors. Halt codes are printed upon exit when obvious errors are detected or exceptions caught. Despite this, some user inputs may cause unhandled exceptions.

To limit unexpected exceptions, always use a definition file that has a default source, sink and load defined (*TimEx* is distributed with a default definition file for RSFQ cells matched to 250 μA JTL loads.) This way, if the source, sink and/or load netlists are not specified in the command line, there are default descriptions in memory.

References

[1] L. C. Müller and C. J. Fourie, "Automated state machine and timing characteristic extraction for RSFQ circuits," *IEEE Trans. Appl. Supercond.,* vol. 24, p. 1300110, 2014.

[2] C. J. Fourie, "Extraction of SFQ circuit Verilog models through flux loop analysis," *IEEE Trans. Appl. Supercond.,* vol. 28, p. 1300811, 2018.