SVATS ver.0.1

Francesco Campedelli <campedel@email.com>

September 26, 2001

"Safety Verification Analysis
Added To Spice"

SVATS ver.0.1 USER GUIDE (user guide ver.1)

Contents

1 INTRODUCTION

SVATS is a collection of Python scripts used for adding repetitive analysis capabilities to electronic circuit simulators like Berkeley Spice. It has been developed to assist the author in doing safety verification analysis for electronic circuits used in railway signalling applications. As such, SVATS can be useful first of all for Railway Safety Verification analysis of the so called "inherent fail-safe" hardware, which is basically made of analog circuits. This kind of circuits shall have a deterministic behaviour under random failure of any of its components. Random failures are those not caused by overstressing the component; each kind of electronic component can have a statistical failure rate, which value depends on technological, environmental and application parameters as determined, for

example, in MIL-HDBK-217F or in other similar standards. In this case SVATS can be used to perform an FMEA (Failure Mode and Effect Analysis) on a given circuit, to help the designer (or the validation engineer) assessing the safety of a circuit or of a part of it, according to Railway Safety Standards ORE A155.3 and prEN50129 (these standars define a catalogue of random failures that should be considered for each kind of discrete electronic component). SVATS can also evaluate components' stress levels, such as maximum voltage applied to capacitors or semiconductors and power dissipation level in resistors, under normal working conditions or under fault conditions. These parameters are used for reliability analysis (MTBF calculation). With minor variations, I think SVATS could be used for safety analysis also in other electronic fields where it's very important to assess the safety integrity level of the circuits, since a failure can be critical for human life (such as in avionics and biomedical), or when you deal with safety standards like IEC950/EN60950.

2 ASSUMPTIONS

In this manual it is assumed that you are already familiar with Spice 3F syntax and you know how to write a spice file for simulation.

3 SVATS REQUIREMENTS AND INSTALLATION

As of v.0.1, SVATS runs on a computer with a UNIX-like O.S. (because of the use of POSIX calls); actually it has been developed and tested under Linux. Before running SVATS be sure to have installed the following required packages:

- Python 1.52 or newer (see: http://www.python.org)
- Gadfly 1 (see: http://www.chordate.com/gadfly.html) - Adds SQL Relational Database capabilities to Python

Some Linux distributions have already prebuilt packages included.
In order to perform electronic circuit simulations with SVATS, you also need a Berkeley Spice 3f5 compatible simulator (i.e. that accept the same circuit sintax and has the same command line options). The original Berkeley Spice 3f5 is available in source or in binary format at ibiblio (formerly Metalab, see: http://www.ibiblio.org/pub/Linux/apps/circuits/). Other useful FREE versions are:

- OPUS Spice, a version of Spice modified by a group of people at the University of Ljubljana, which includes also XSpice digital capabilities for mixed mode simulations (see: http://fides.fe.uni-lj.si/spice/). The stress analysis option described later is working ONLY with Opus Spice v.2.03 or later, since it relies on predefined "min" and "max" functions.
- NG Spice (see: http://ieee.ing.uniroma1.it/ngspice)

Installing SVATS is a simple process: first of all you can type

```
tar -xvvzf svats-0.1.tar.gz
```

obtaining a subdirectory called `svats-0.1`; then you can either move the following executable files:

```
svats_db_create
svats_run
svats_write
```

to a directory included in your PATH variable (i.e. /usr/local/bin) or add the complete path comprising the subdirectory called svats-0.1 to your PATH variable. The subdirectory called "`svats-0.1/doc`" contains various documentation (this manual included), the one called "`svats-0.1/examples`" contains a simple circuit and the files needed to perform the analysis described later in this document.

4 FEATURES AND GUIDELINES

Basically, SVATS takes a file that contains an electronic circuit description in Berkeley Spice 3F4 syntax, previously tested by the designer, and generates a series of similar files: each file simulates a failure in one of the discrete electronic components (resistors, capacitors, transistors, etc.). The failure implemented can be chosen among those indicated in two european railway safety standards, ORE 155.A3 and prEN50129. For what I've seen, these should also include the failure modes used in avionics and naval electronics FMEAs.
The main random failures considered are:

- interruption of one of the pins of the component;
- short circuit of two or three pins of the component;
- increase or decrease of the basic value of the component (resistance for resistors, capacitance for capacitors, etc.);
- increase of the leakage current through the component;
- increase of the series resistance of the component;
- decrease of gain and breakdown voltage of semiconductors (not yet implemented in SVATS 0.1)

The components usually considered are the simplest discrete ones; integrated circuit are not considered because it is assumed that they can have every kind of failure you can think of.
The original Spice file should have been written to perform batch simulation, i.e. shall contain a ".control .... .endc" block with instructions for the analysis (tran, ac, dc, etc.).
The specified analysis is written in each of the generated files. It is possible to add in each file, into the control block, a line for saving the variables to be observed in a Spice raw file, and also to generate a PostScript File with the desidered variables plotted.
Beware that in Opus Spice v.2.03 the PostScript saving of plots is not fully functional (sometimes it works, sometimes Opus crashes).
One of the SVATS scripts takes the list of the files with the failures added and, for each of them, calls Spice in batch mode. Results (in PostScript files or in Spice raw data format) can then be analyzed.

This kind of analysis, in which you look at the effect of random failure of each single component, can be made part of an **FMEA** (Failure Mode and Effect Analysis) on the complete circuit/system.
SVATS can also perform a simple **Stress Analysis** of the components.
WARNING: the stress analysis works ONLY for transient analysis AND with a simulator (like Opus Spice v.2.03 or later) that has internally defined "min" and "max" functions (it should not be difficult to define such kind of functions also in Spice 3F5); if this analysis is specified, SVATS will write in the simulation files some instructions to save, in a "stress" file, the maximum voltage applied to capacitors and semiconductors and the maximum average power dissipated in resistors. The files can then be analyzed to look for overstressed components, either in the normal use or in failure condition; the latter is important to see if a single failure can eventually propagate to other components.
It's the author will to add automatic evaluation of the stress analysis in the future releases of SVATS; this is the main reason why SVATS uses a relational database (Gadfly under Python) for storing component informations. An analysis mode in which you can specify two or more failures at a time is also planned for future releases.

5 STEP by STEP GUIDE

In the following subpargraphs the steps needed to perform a FMEA-like analysis using SVATS are briefly described.

5.1 PREPARE YOUR CIRCUIT

First of all you need to create a file that describes your circuit and the kind of analysis (transient, AC, DC) you want to perform. The file MUST end with ".cir" extension.
The file must contain a ".control .... .endc" block with instructions for the analysis, so that it can be run in batch mode [i.e. typing the command "spice3 -b foo.cir"].
It is obviously strongly reccomended that the file has been simulated BEFORE using SVATS, so that convergence and other problems have been already debugged! Since SVATS could generate a lot of files and consequently a lot of simulations, it is important that you write down a reasonable maximum time allowed for each simulation run, in order to avoid Spice locks on a single file.
SVATS (v.0.1) can apply failures only to dipoles (two terminals components like resistors, capacitors, inductors, diodes) and tripoles (three terminals components like transistors).
Spice uses four terminals for MOS components, but in all the applications of **discrete** MOS components that I've seen, the body terminal is always connected to the Source terminal; to add failures to three terminals MOS for which you have a four terminals model and for which you connect the body with the source, just include them in a three terminals subcircuit (Xfoo..) and add an appropriate .SUBCKT description to your file.

5.2 DEFINE THE ANALYSIS DEFAULTS

According to the kind of analysis you have to perform, you should write a file which contains the analysis defaults values and modes. It is recommended that you open one of the default.def files included, modify the parameters you like and save the file with a different name (foo.def).
The file default.def will generate all the possible failure modes supported by SVATS; the file railway.def contains only the failure modes relevant to railway safety analysis.
For he sake of clearness, this is the content of default.def:

```
# file:  default.def
# SVATS DEFAULT VALUES FILE
# date:  29/12/00
# analyst:  f.campedelli

#[GENERAL]#################################################

rshort:  1e-3
ropen:  10e9
val_max:  100
val_min:  0.1
val_inc_step:  4
val_dec_step:  4
val_type:  log
sr_max:  1e6
sr_min:  1
sr_step:  2
sr_type:  log
lr_min:  1
lr_max:  1e6
lr_step:  2
lr_type:  log


#[RESISTORS]###############################################

r_modes:  [ro, rs, rdec, rinc]


#[CAPACITORS]##############################################

c_modes:  [co, cs, cisr, cdec, cinc, cilc]


#[INDUCTANCES]#############################################

l_modes:  [lo, ls, lisr, ldec, linc]
```

```
#[DIODES]#################################################

d_modes:  [do, ds, dilc, disr]
d_bv_min:  2
d_bv_step:  3
d_bv_type:  lin


#[TRANSISTORS]############################################
#Warning!  q_modes list shall be on a single line!
q_modes:  [qbo, qco, qeo, qbc_s, qbc_s_eo, qbe_s, qbe_s_co, qec_s, qec_s_bo,
qbc_ilc, qbc_ilc_eo, qbe_ilc, qbe_ilc_co, qec_ilc, qec_ilc_bo]
q_g_min:  10
q_g_step:  3
q_g_type:  lin


#[BETWEEN NODES] (not yet impl.)##########################
#nodes_modes:  [ns, nilc]
```

Basically, for each kind of component you have to define:

- which failures you wish to apply: lists whose name contain "_modes"
- the minimum and maximum values for the parameters of the failure

The meaning of each parameter is as follow:

param	**meaning**
rshort	short circuit value
ropen	open resistor value
val_max	max relative value for increase in Value (10 => Rmax=10*Rnom)
val_min	min relative value for decrease in Value (0.1 => Rmin=0.1*Rnom)
val_inc_step	number of steps from nominal Value to maximum Value
val_dec_step	number of steps from nominal Value to minimum Value
val_type	stepping type for Value variations (linear or logarithmic)
sr_max	maximum value of Serial Resistance (SR)
sr_min	minimum value of Serial Resistance (SR)
sr_step	number of steps from SR min to SR max
sr_type	SR stepping type (linear or logarithmic)
lr_min	minimum value of Leakage (parallel) Resistance (LR)
lr_max	maximum value of Leakage (parallel) Resistance (LR)
lr_step	number of steps from LR min to LR max
lr_type	LR stepping type (linear or logarithmic)

What follows (transistors' gain and diodes' breakdown voltage variations) is not yet implemented (in SVATS rel.0.1):

param	meaning
d_bv_min	minimum relative value for decrease in Breakdown Voltage (Diodes)
d_bv_step	number of steps from BV nom to BV min
d_bv_type	BV stepping type (linear or logarithmic)
q_g_min	minimum relative value for decrease in Gain (Transistors)
q_g_step	number of steps from Gain nom to Gain min
q_g_type	Gain stepping type (linear or logarithmic)

Failure modes acronyms:

	Resistors
rinc	Increase in resistance
rdec	Decrease in resistance
ro	Open
rs	Short

	Capacitors
cisr	Increase of serial resistance
cdec	Decrease in capacitance
cinc	Increase in capacitance
co	Open
cs	Short
cilc	Increase of leakage current

	Inductors
ldec	Decrease in Inductance
lo	Open
ls	Short
lisr	Increase of serial resistance

	Diodes
ddzv	decrease in breakdown/zener voltage (not impl.)
do	Open
dilc	Increase of leakage current
disr	Increase of serial resistance
dizv	increase in breakdown/zener voltage (not impl.)
ds	Short

	Transistors (BJT, MOS, MesFet, JFet)
qbc_ilc	Increase of leak.current between B & C
qbc_ilc_eo	increase of leak.current between B & C, E open
qbe_ilc	increase of leak.current between B & E
qbe_ilc_co	increase of leak.current between B & E, C open
qec_ilc	increase of leak.current between E & C
qec_ilc_bo	increase of leak.current between E & C, B open
qbo	B open
qco	C open
qeo	E open
qbc_s	short between B & C
qbc_s_eo	short between B & C, E open
qbe_s	short between B & E
qbe_s_co	short between B & E, C open
qec_s	short between C & E
qec_s_bo	short between C & E, B open
qb_isr	increase of series resistance of B
qe_isr	increase of series resistance of E
qc_isr	increase of series resistance of C
qg_dec	decrease in gain (not impl.)

Parameter name	**Value type**	**def val**	**Permissible values**
rshort	float	1e-3	
ropen	float	10e9	
val_max	float	100	
val_min	float	0.1	
val_inc_step	int	4	
val_dec_step	int	4	
val_type	char	log	log, lin
sr_max	float	1e6	
sr_min	float	1	
sr_step	int	4	
sr_type	char	log	log, lin
lr_min	float	1	
lr_max	float	1e6	
lr_step	int	4	
lr_type	char	log	log, lin
d_bv_min	float	2	
d_bv_step	int	4	
d_bv_type	char	log	log, lin
q_g_min	float	10	
q_g_step	int	4	
q_g_type	char	log	log, lin

It is important to note that the file MUST contain a line for ALL the parameters in the default.def file, otherwise the scripts will fail; if you don't need a parameter, just put zero after the colon, or nothing between square brackets in lists. *Don't change the names of the variables!*

Also, a white space MUST be inserted between the colon and the value:

val_min: 2	=>	this is OK
r_modes: []	=>	this is OK
lr_min:3	=>	this is NOT OK!
l_modes:	=>	this is NOT OK!
l_modes: [0]	=>	this is NOT OK!

5.3 DEFINE THE PROJECT FILE

The next step is to define what components are to be considered for the failure analysis, what is the file which contains the defaults to be applied, where the Spice compatible simulator is, what electrical variables you want to save for successive analysis and if you want them saved in a Spice raw file and/or in a PostScript file, if you want also a stress analysis and finally the project directory that will contain the SVATS generated spice files and the result of the multiple Spice run. All of these informations must be saved in a file.
So, first of all you have to create a project directory that will contain all the files generated by Svats to complete the desidered analysis; let's call it "`fmea2`"; then you can copy to it the file called `svats_proj.txt` and eventually rename it as you like.
Then open the file and modify the lines as follows:

- from your netlist (.cir file), add the ref of the components for which the failure analysis is to be applied; put each ref in the square brackets of the list of the appropriate component type (i.e. r_list for resistors, q_list for each kind of transistors, including BJT, JFET and also three terminal subcircuits, and so on ...). Each ref shuld be separated from the following by a comma AND a whitespace.
- Add for "default_dir:" the complete path of the directory that contains the default file (dafault.def, foo.def or whatever else); due to author's lazyness, remember to add a slash at the end ..
- Add for "default_file:" the name of the default file (dafault.def, foo.def or whatever else).
- Add for "input_dir:" the complete path of the directory that contains the spice file to which you want to apply the Failure Mode Analysis; as ususal remember to add a slash at the end.
- Add for "input_file:" the name of the spice file.
- Add for "project_dir:" the complete path of the above mentioned and previously created directory (i.e. "`fmea2`").
- Add for "spice_path:" the complete path, in this case also including the name, of the electronic circuit simulator (Spice3 or the like).

- Add for "out_type:" one of "ps / raw / both" to indicate how you want the interested electrical variables saved (as a PostScript file and/or a Spice raw file).
- Add for "out_vars:" the interested electrical variables, separated from each other by a white space.
- Add for "stress_en:" one of "y / n" to indicate if you also want a stress analysis performed (Remeber: only with a simulator that supports "min" and "max" functions!).
- Add for "timeout:" a numerical value that will be taken as a timeout value in s for each performed simulation [example: for 10min-> timeout: 600].

5.4 RUN SVATS

Running Svats involves calling three different scripts from the command line in a definite sequence:

1. first you invoke the script "`svats_db_create`", passing as the only argument the name (preceeded eventually by the full path) of your project file. This script only opens the project file, reads the name of the input spice file and then reads the file itself; then opens the defaults file and generates a database of the circuit, including for each component the failure modes and the failure parameters defined in the defaults file. Next releases of Svats will have the capability to modify the defaults values for each single component, to fine tune the analysis. The database files will be saved in a subdirectory called "`circ_db`" under the project_dir. This script will also create three more subdirectories: "`files`", which will contain the input spice, failure modified files; "`stress`", which will contain the stress files; "`results`", which will contain the result (ps or raw files) of the analysis. If you need to change the failure parameters for one or more of the components in your netlist, at the moment (SVATS 0.1) you need to start a python shell, import the gadfly module, open the circuit database and execute the appropriate python/SQL queries (you know what you're doing, right? ;-)). A simple user interface will be added to future releases, to avoid even this burden to the poor FMEA or analog engineer ...
2. second you invoke the script "`svats_write`", again passing as the only argument the name (preceeded eventually by the full path) of your project file. This script reads the input spice file and, based on the informations written in the database, for each component which failures are to be analyzed, creates one or more new modified spice files. At the end it generates also a file which contains the list of the generated files; write down the name of this file, since it is the argument to be passed to the third and last script; if you forget it, its name is built by adding to the input spice file name (with the .cir suffix stripped) the fixed string "`SVATS-list.txt`" (i.e, if your input spice file is "`foo.cir`", the list file will be "`foo.SVATS-list.txt`"). This file will be saved in the "`files`" subdirectory.

3. third and last you invoke the script "`svats_run`", passing as the only argument the name (preceeded eventually by the full path) of the above mentioned list file ("`foo.SVATS-list.txt`"). While it runs, it will print on the terminal the name of the file that is currently being simulated. Each simulation is done by forking a Spice batch run in background and periodically checking that the process is still alive; if the simulation will not end before the stated timeout, it will be aborted, since we assume that Spice could have found some problems. `svats_run` will write in a log file if each simulated file has succeded the run or not. If you want to halt the simulations, just press "CTRL - C" and answer yes.

Why is this so (apparently) complex, instead of having a single and simple frontend? The base idea (other than following the UNIX philosophy ...) is to have the possibility to launch the last step, which can be very time, memory and disk space consuming, when you don't worry about having an unusable PC, for example by night. You can also define FMEA analysis for more circuits/subcircuits, all to be performed by a series of "`cron`" commands ... It would also be nice to find a way to distribute all the simulations among some network connected computers. And finally, a single frontend can always be written ...

5.5 LOOKING AT THE RESULTS

Once `svats_run` has completed the simulations, in the subdirectort "`result`" you should find the files with the results of your simulations, in the format that you have specified (spice raw file, which you can open with Spice, Nutmeg, gWave, etc., or Postcript which you can open with GV or the like). In particular, the files will be named after the name of the original Spice file, complemented with the ref of the component being analyzed, the acronym of the failure and the eventual step number.
For example, suppose you have written in the project file that you want to apply failures to the component C23, and in the def file that the failures to be considered are CDEC (decrement of capacitance) in 5 steps, CO (open capacitor) and CILR (increase in leakage current) in 3 step; suppose also that your spice file is, as usual, "`foo.cir`" and that you have chosen to write the result in a raw file. If you have crossed your fingers and all the simulations ran without errors (i.e. without exceeding the specified timeout), you will find the following files:

```
foo.C23_OPEN.raw
foo.C23_DEC_step1.raw
foo.C23_DEC_step2.raw
foo.C23_DEC_step3.raw
foo.C23_DEC_step4.raw
foo.C23_DEC_step5.raw
foo.C23_ILR_step1.raw
foo.C23_ILR_step2.raw
foo.C23_ILR_step3.raw
```

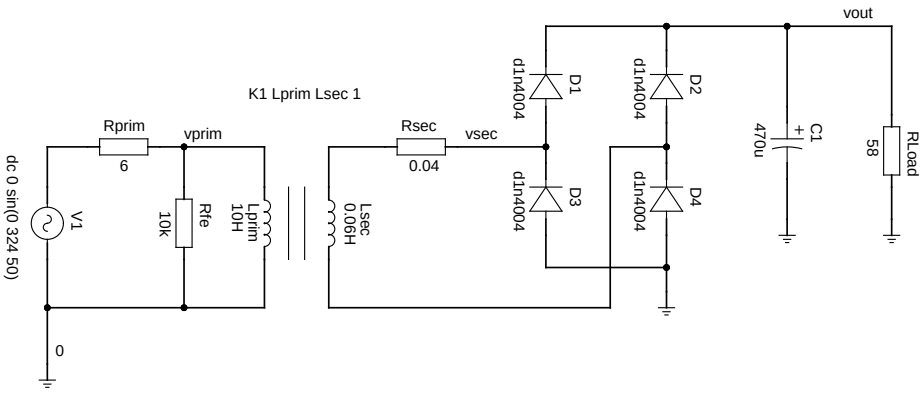
If you have also checked to perform stress analysis, in the subdirectory "`stress`" you will find:

```
foo.C23_OPEN.str
foo.C23_DEC_step1.str
foo.C23_DEC_step2.str
```

`foo.C23_DEC_step3.str`
`foo.C23_DEC_step4.str`
`foo.C23_DEC_step5.str`
`foo.C23_ILR_step1.str`
`foo.C23_ILR_step2.str`
`foo.C23_ILR_step3.str`
plus the file `foo.str` which contains the list of the stress variables.
In the start directory you will also find a file called `svats_run.log`, which shows what simulations succeded (it is simply an echo of the messages output to the terminal while `svats_run` is running).

6 EXAMPLE OF A SESSION

We take as an example the files included in the subdirectory "`examples`" of the .tar.gz distribution file. This is a very simple circuit (a simple full wave rectifier used as a power supply), as described in the following figure.



BTW, this figure has been drawn with the gschem program, part of the beautiful, already usable and GPL'd gEDA suite (available at http://www.geda.seul.org). gEDA also support spice netlisting ... if you're searching for a good and free Linux schematic entry software, you know where to look! In the "`examples`" directory I've also included the .sch file for gschem.
In the same dir you'll find:

- `example1.cir`: this is the spice input file to start from
- `example1.def`: this is the failure definition file
- `example1.pro` : this is the project file

and that's all you need to start the analysis.
So, when you have looked at these files, possibly understood them and eventually changed the path to your circuit simulator (you can eventually decrease the timeout value, since I suppose you own a "faster than my 133MHz PentiumI" machine ...), you simply start with cd'ing to the `examples` subdirectory and type at the prompt:

```
svats_db_create ./example.pro
```

this will create the circuit database in the subdir `../examples/circ_db`. Then:

```
svats_write ./example.pro
```

that will fill the `../examples/files` subdir with the appropriate files for Spice and finally:

```
svats_run ./files/example1.SVATS-list.txt
```

If everything is going well, you should see on the screen something like:

```
now running:  spice3 -b ./files/example1.C1_ISR_step3.cir

.

.

OK
```

This message will repeat for each of the files in the `../examples/files`; at the end the last two messages will be:

```
Analysis completed:  see Log File for details

Total elapsed time:  123.119220018
```

After that, if you aren't already under X, switch to a X session, invoke GV or whatever else PostScript viewer you have and take a look at the .ps files in the `../examples/results` subdirectory. If you didn't modify the .def and .pro files and if all the simulations ran without problems, you should find 20 .ps and 20 .raw files. You can also run nutmeg or Spice3 or any other program that can read Spice raw files and load the .raw files in the same directory. And that's all, folks (at least for the moment).

7 DISCLAIMER

This is alpha stage software and it is provided "as is", without any warranty. No warranties are provided that the result you obtain are acccurate, nor that the software won't damage your data. USE AT YOUR OWN RISK! Furthermore, you can't use by themselves the results that you can obtain running this software to prove anything, expecially the safety of a system or of a circuit; it is assumed that you know what you're doing and that you understand that the software is only a tool to help you in the analysis of your circuits.

References

[1] H.Kumamoto, E.J.Henley - "Probabilistic Risk Assesment and Management for Engineers and Scientists" - second edition - IEEE Press, 1996

[2] C.E.Hymowitz, L.G.Meares, B.Halal - "New Techniques for Fault Diagnosis and Isolation of Switched Mode Power Supplies" - Intusoft corp.