Static and Dynamic Checks

Abstract

This rapid adoption kit/workshop database describes the usage of the Spectre APS/XPS static and dynamic design checks available in MMSIM14.1.0. These checks may be used to identify typical design problems including high impedance nodes, DC leakage paths, extreme rise and fall times, excessive device currents, setup and hold timing errors, voltage domain issues or connectivity problems. While the static checks are basic topology checks, the dynamic checks are performed during a Spectre APS/XPS transient simulation.

Software

- MMSIM14.1.0

Prerequisites

- Basic working knowledge of running simulations with Spectre.

7,085,700

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

Introduction

Circuit checks enable you to analyze design problems, such as high impedance nodes, leakage paths between power supplies, timing errors, power issues, connectivity problems, or extreme rise and fall times. They can be separated into dynamic and static checks. Dynamic checks are performed during transient simulation. Static checks are topology checks, which do not require any simulation.

MMSIM contains dynamic and static checks for Spectre® APS and Spectre® XPS. This workshop introduces all the dynamic and static checks available in MMSIM.

Static checks are performed after the voltage propagation technique is applied on the topology of a circuit. It is to be noted that voltage propagation technique does not require a transient simulation. Therefore, a static check is faster than a dynamic check. Static check gives the worst case possible for a given scenario. For Spectre® XPS, static checks are applicable to FLASH and SRAM circuits only. Static checks use the `static_` keyword prefix, and write the results into a file with the extension `.static.xml`. The xml file can be viewed with any web browser.

Dynamic checks are performed during the transient simulation. Therefore, their result depends on the stimuli applied to the design. Dynamic checks use the `dyn_` keyword prefix, and write the result into a file with the extension `.dynamic.xml`. The xml file can be viewed with any web browser.

The default output format of all circuit checks is xml. Therefore, the output files have an extension .xml and are located in the raw directory. You can use the check_format parameter with the options statement to convert the xml files to a text format, as shown below:

```
opt options check_format=text (Spectre format)
.option check_format=text (SPICE format)
```

After the conversion, the converted files in text format have an extension .rpt and are located in the raw directory. The prefix of these files is same as the prefix of xml files.

All design checks can be applied either globally to the entire design, or locally to specific blocks of a design. The scoping options are available to define the scope of each design check. The design checks can be applied to a specific subcircuit instance (`inst`), or to all instances of a subckt definition (`subckt`). Exclusion of a specific subcircuit instance (`xinst`), or all instances of a subckt (`xsubckt`) is supported. All scoping options support wildcarding, and the hierarchy level of the scope can be defined by the `depth` option. For further details on the scoping, please check the Spectre User Manual or use command "`% spectre -h`".

The following static checks are supported in Spectre® XPS and APS:

1. Static high impedance node check (`static_highz`)
2. Static DC leakage path check (`static_dcpath`)
3. Static NMOS forward bias bulk check (`static_nmosb`)
4. Static PMOS forward bias bulk check (`static_pmosb`)
5. Static voltage domain check (`static_voltdomain`)
6. Static transmission gate check (`static_tgate`)
7. Static always conducting NMOS check (`static_nmosvgs`)
8. Static always conducting PMOS check (`static_pmosvgs`)
9. Static MOSFET voltage check (`static_mosv`)
10. Static resistor voltage check (`static_resv`)
11. Static capacitor voltage check (`static_capv`)

12. Static Diode check (`static_diodev`)
13. Static resistor check (`static_resistor`)
14. Static capacitor check (`static_capacitor`)
15. Static ERC check (`static_erc`)
16. Static RCdelay check (`static_rcdelay`)

The following table lists the dynamic checks that are supported in Spectre®, Spectre® APS, Spectre® XPS SPICE mode and Spectre® XPS:

		Spectre®	Spectre® APS	Spectre® XPS SPICE	Spectre® XPS
1	Dynamic High Impedance Node Check (`dyn_highz`)	Yes	Yes	Yes	Yes
2	Dynamic DC Leakage Current Path Check (`dyn_dcpath`)	Yes	Yes	Yes	Yes
3	Dynamic Floating Gate Induced Leakage Check (`dyn_floatdcpath`)	Yes	Yes	Yes	Yes
4	Dynamic MOSFET Voltage Check (`dyn_mosv`)	No	No	No	Yes
5	Dynamic Resistor Voltage Check (`dyn_resv`)	No	No	No	Yes
6	Dynamic Capacitor Voltage Check (`dyn_capv`)	No	No	No	Yes
7	Dynamic Diode Voltage Check (`dyn_diodev`)	No	No	No	Yes
8	Dynamic Excessive Element Current Check (`dyn_exi`)	Yes	Yes	Yes	Yes
9	Dynamic Excessive Rise and Fall Time Check (`dyn_exrf`)	Yes	Yes	Yes	Yes
10	Dynamic Glitch Check (`dyn_glitch`)	Yes	Yes	Yes	Yes
11	Dynamic Setup and Hold Check (`dyn_setuphold`)	Yes	Yes	Yes	Yes
12	Dynamic Noisy Node Check (`dyn_noisynode`)	Yes	Yes	Yes	Yes
13	Dynamic Node Capacitance Check (`dyn_nodecap`)	Yes	Yes	Yes	Yes
14	Dynamic Subckt Port Power Check (`dyn_subcktpwr`)	Yes	Yes	Yes	Yes
15	Dynamic Pulse Width Check (`dyn_pulsewidth`)	Yes	Yes	Yes	Yes
16	Dynamic Active Node check (`dyn_actnode`)	No	No	No	Yes
17	Dynamic Subckt Instance Activity Check (`dyn_activity`)	No	No	No	Yes

Use Model

The syntax used by the static and dynamic checks can be characterized by the following syntax.

Syntax

```
Name check_keyword <parameter=value>
```

Examples:

```
static_hz1 static_highz node=[*]
static_dc1 static_dcpath node=[vdd gnd]
sta_nmosb1 static_nmosb model=[nmos]
dyn_hz1 dyn_highz node=[*] duration=2n time_window=[1n 20n]
```

The key parameters are:

1. `node` … nodes to which a check is applied to.
2. `model` … device model definitions to which a check is applied to.
3. `time_window` …defines a time window in which a check has to be performed
4. `duration` …defines a time limit over which a violation is reported

The default value for each parameter is dependent on an individual design check. Run `spectre -h` followed by the checker keyword to get detailed information of each design check. For example:

```
% spectre -h static_dcpath

% spectre -h dyn_highz
```

The next two sections demonstrate the static and dynamic checks. Notice that for each check, its corresponding netlist <*netlistName*> is located in the sub-directory that is named by <*netlistName*>.

Workshop exercise for static check

In this section, different static checks are introduced and applied to simple circuits with their output shown.

Starting with MMSIM 14.1, the voltage propagation algorithm has been changed from logic based to voltage-based because voltage-based voltage propagation is better than logic based voltage propagation. The new voltage-based voltage propagation provides better results in analog and digital circuit designs. Due to this change, the parameters vlth and vhth that were related to logic-based voltage propagation have been retired. For voltage-based voltage propagation, two new parameters vnth and vpth have been introduced.

Starting with MMSIM 14.1, the static checks will be available in XPS and APS, only. Running static checks with APS gives more accurate results than XPS.

1. Static high impedance check

Action 1: Open the netlist `static_highz.sp`, review the circuit and the design check statement.

```
static_hz1 static_highz node=[*]
```

The above statement checks for any high impedance node in a design. A node is considered as a high impedance node when it does not have any conducting path to a DC power supply or ground.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_highz.sp
```

Action 3: Open the result file `static_highz.static.xml` with a webbrowser.

The result file reports that node `inp` is a high impedance node. Check the connectivity of this node to understand why it is floating. Fix the connectivity to resolve the problem.

Static HighZ Node Check Violations

static_highz: static_hz1

static_hz1 static_highz node=["*"]

Violation Count: 1

Static HighZ Node Check - Violation

Title	Node Name
static_hz1	inp

2. Static DC leakage path check

Action 1: Open the netlist `static_dcpath.sp`, review the circuit and the design check statement.

```
dc1 static_dcpath node=[vdd 0]
```

The above statement checks for any leakage path between `vdd` and `0`.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_dcpath.sp
```

Action 3: Open the result file `static_dcpath.static.xml` with a webbrowser.

The result file reports that there is a DC leakage path between `vdd` and `0` through the elements x3.mp1, and x3.mn2. Check the connectivity of this path. Fix the connectivity to resolve the problem.

Static DC Leakage Path Check Violations

static_dcpath: dc1

dc1 static_dcpath net=["vdd" "0"]

Violation Count: 1

Static DC Leakage Path Check - Violation

Title	From Net	To Net
dc1	vdd	0

Path Elements:

x3.mp1

x3.mn2

3. Static NMOS forward bias voltage check

Action 1: Open the netlist `static_nmosb.sp`, review the circuit and the design check statement.

```
nmosb1 static_nmosb model=nmos
```

The above statement checks for any NMOS device defined by model `nmos` with a forward biased bulk condition. A violation is generated when the bulk bias voltage meets the following conditions:

- when `mode' = `definite':


```
min( Vb ) >= min( Vd, Vs ) + abs( vt )
```

- when `mode' = `possible':

```
max( Vb ) >= min( Vd, Vs ) + abs( vt )
```

where `Vt` is the threshold for p-n junction being checked. If user does not specify, either mode or `Vt`, then the default values will be used, which are “definite” and 0.3V, respectively.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_nmosb.sp
```

Action 3: Open the result file `static_nmosb.static.xml` with a webbrowser.

The result file reports that NMOS device x1.mn1 is having a forward biased bulk. The reason is that the following condition is true:

```
min( Vb ) >= min( Vd, Vs ) + abs( vt )
```

3 >= 0 + 0.3 is true

Check the bulk connection of this NMOS device and fix it. You may create a similar test case to test the `static_pmosb` check that is for PMOS devices.

Static Forward Bias Bulk Check Violations

static_nmosb: nmosb1

nmosb1 static_nmosb model=["nmos"]

Violation Count: 1

Static Forward Bias Bulk Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
nmosb1	x1.mn1	nmos	inv	(0, 3)	(0, 3)	(0, 0)	(3, 3)

4. Static voltage domain check

Action 1: Open the netlist `static_voltdomain.sp`, review the circuit and the design check statements.

```
chk1 static_voltdomain model=[nmos_thin pmos_thin] vmin=0 vmax=1.8
```


This voltage domain check analyses if any device is connected OUTSIDE the voltage domain/range defined by `vmin` and `vmax` This check can be used to analyze if any low-voltage device is connected to high-voltage nodes.

In this netlist, the device models "`nmos_thin`" and "`pmos_thin`" are low-voltage (thin oxide) devices, which should only be connected to low-voltage nodes. The first statement checks for any instance of these low-voltage devices connecting to any node with voltage OUTSIDE the domain of 0V and 1.8V.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_voltdomain.sp
```

Action 3: Open the result file `static_voltdomain.static.xml` with a webbrowser.

The first check reports that devices x2.mn2 (model=`nmoslo`), and x2.mp1 (model=`pmoslo`) are connected to a high-voltage node `mid`. Check the connectivity of these reported devices and the voltage on node `mid`.

Static Voltage Domain Device Check Violations

static_voltdomain: chk1

chk1 static_voltdomain model=["nmos_thin" "pmos_thin"] vmin=0 vmax=1.8

Violation Count: 2

Static Voltage Domain Device Check - Violation

Title	Node Name	Instance Name
chk1	mid	x_thin.mn2
chk1	mid	x_thin.mp1

5. Static transmission gate check

Action 1: Open the netlist `static_tgate.sp`, review the circuit and the design check statement.

```
tgate1 static_tgate node=[*]
```

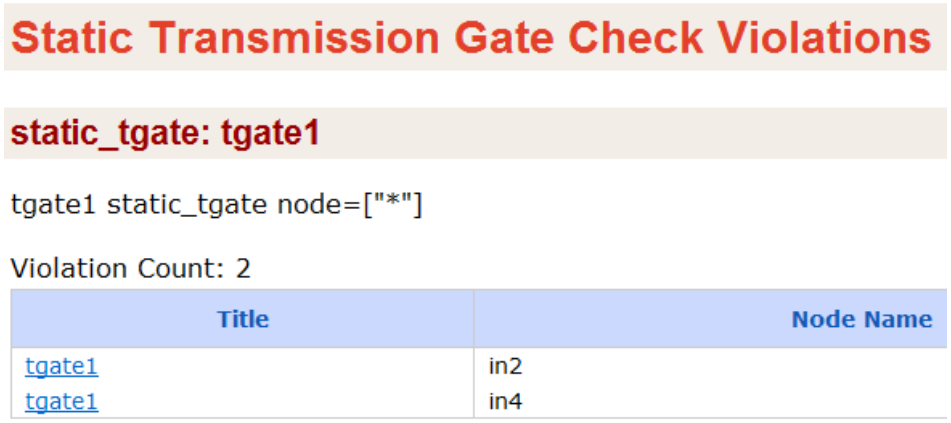
Typically a transmission gate consists of a combination of a NMOS and a PMOS device. Nevertheless, if one of the devices is missing then it may cause a leakage current in the driven stage. This feature checks for any such missing MOSFET in a transmission gate which may cause a leakage current between power supplies.

Action 2: Run XPS on the netlist


```
% spectre +spice +xps +cktpreset=sram static_tgate.sp
```

Action 3: Open the result file `static_tgate.static.xml` with a webbrowser.

The result file reports that nodes `in2` and `in4` are both driven by only one device. The stage driven by these nodes may create leakage paths between power supplies. Check the connectivity of the reported nodes, and analyze the transmission gate problem. Add the missing devices and run the check again.



Static Transmission Gate Check Violations

static_tgate: tgate1

tgate1 static_tgate node=["*"]

Violation Count: 2

Title	Node Name
tgate1	in2
tgate1	in4

6. Static always conducting NMOSFET check

Action 1: Open the netlist `static_nmosvgs.sp`, review the circuit and the design check statement.

```
mos1 static_nmosvgs model=nmos vt=0.5
```

The above statement checks for any NMOS device, with model `nmos`, that is always being in “on” state. The NMOS device is considered to be in “on” state if the following condition is satisfied:

```
min(Vg) > min (Vs/Vd) + abs(vt)
```

where `Vt` is the MOSFET threshold voltage set by the user.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_nmosvgs.sp
```

Action 3: Open the result file `static_nmosvgs.static.xml` with a webbrowser.

The result file reports that NMOS device x2.mn2 is fulfilling the always “on” condition. Check the connectivity of the device to understand this check. You may create a similar case to test the `static_pmosvgs` check for PMOS devices.

Static Always Conducting NMOSFET Check Violations

static_nmosvgs: mos1

mos1 static_nmosvgs model=["nmos"] vt=0.5

Violation Count: 1

Static Always Conducting NMOSFET Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos1	x2.mn2	nmos	inv	(0, 0)	(1.8, 1.8)	(0, 0)	(0, 0)

7. Static MOSFET voltage check

Action 1: Open the netlist `static_mosv.sp`, review the circuit and the design check statements.

```
mos1 static_mosv model=["nmos"] cond="v(g,s)>1.9"
```

The above statement checks for any MOSFET device with the model `nmos` that has a voltage difference between gate and source greater than 1.9V.

```
mos2 static_mosv model=[*] cond="v(d)>1.9"
```

Similarly, the above statement checks for any MOSFET device that has a voltage greater than 1.9V at its drain terminal.

```
mos3 static_mosv model=[*] cond="w<30e-6"
```

The above statement checks for any MOSFET device that has a width less than 30um.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_mosv.sp
```

Action 3: Open the result file `static_mosv.static.xml` with a webbrowser.

The first check reports that two devices x1.mn2 and x2.mn2 have `v(g,s)` greater than 1.9V. The second check reports that x1.mn2 and x1.mp1 have drain voltage greater than 1.9V. The third check reports that x1.m2 and x2.m2 have width less than 30um.

Static MOSFET Voltage Check Violations

static_mosv: mos1

mos1 static_mosv model=["nmos"] cond="v(g,s)>1.9"

Violation Count: 2

Static MOSFET Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos1	x1.mn2	nmos	invhi	(0, 3.3)	(0, 3.3)	(0, 0)	(0, 0)
mos1	x2.mn2	nmos	inv	(0, 1.8)	(0, 3.3)	(0, 0)	(0, 0)

static_mosv: mos2

mos2 static_mosv model=["*"] cond="v(d)>1.9"

Violation Count: 2

Static MOSFET Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos2	x1.mn2	nmos	invhi	(0, 3.3)	(0, 3.3)	(0, 0)	(0, 0)
mos2	x1.mp1	pmos	invhi	(0, 3.3)	(0, 3.3)	(3.3, 3.3)	(3.3, 3.3)

static_mosv: mos3

mos3 static_mosv model=["*"] cond="w<30e-6"

Violation Count: 2

Static MOSFET Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos3	x1.mn2	nmos	invhi	(0, 3.3)	(0, 3.3)	(0, 0)	(0, 0)
mos3	x2.mn2	nmos	inv	(0, 1.8)	(0, 3.3)	(0, 0)	(0, 0)

8. Static resistor/capacitor voltage check

Action 1: Open the netlist `static_resv_capv.sp`, review the circuit and the design check statement.

```
resv1 static_resv cond="v(1,2)>0"
```

The above statement checks for any resistor that has a voltage across its terminals greater than 0V.

```
capv1 static_capv cond="v(1,2)>0"
```

Similarly, the above statement checks for any capacitor that has a voltage across its terminals greater than 0V. Here, “1” stands for positive terminal and “2” stands for negative terminal of an element.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_resv_capv.sp
```

Action 3: Open the result file `static_resv_capv.static.xml` with a webbrowser.

The result file reports that the resistor r1 has a voltage greater than 0V.

Static Resistor Voltage Check Violations

static_resv: resv1

resv1 static_resv cond="v(1,2)>0"

Violation Count: 1

Static Resistor Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	V1	V2
resv1	r1	resistor	-	(0, 3.3)	(0, 3.3)

Similarly, it reports that the capacitor c1 has a voltage greater than 0V.

Static Capacitor Voltage Check Violations

static_capv: capv1

capv1 static_capv cond="v(1,2)>0"

Violation Count: 1

Static Capacitor Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	V1	V2
capv1	c1	capacitor	-	(0, 3.3)	(0, 3.3)

9. Static diode voltage check

Action 1: Open the netlist `static_diodev.sp`, review the circuit and the design check statement.

```
dv1 static_diodev model=["diode1"] cond="v(a,c)>=0.5"
```

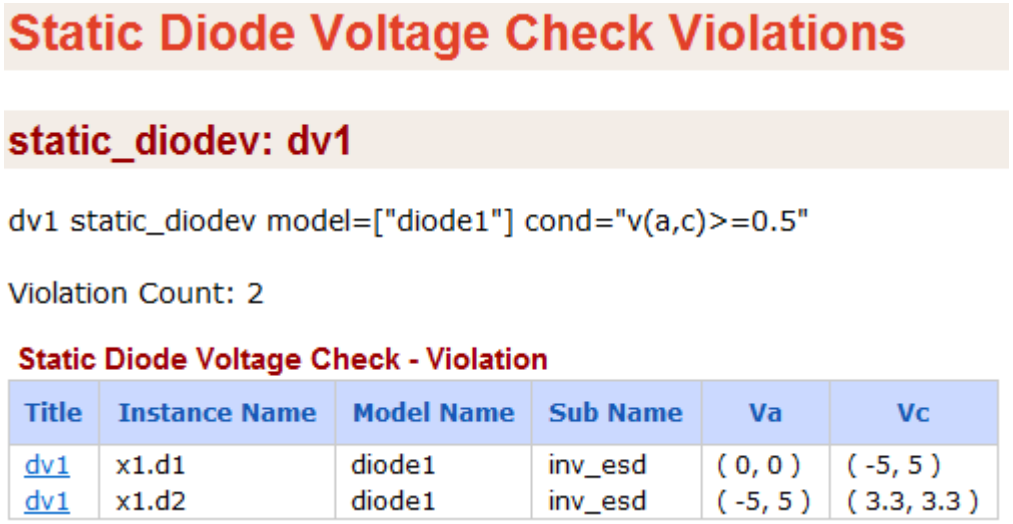
The above statement checks for any diode with model name "`diode1`" that has a voltage across its terminals greater than 0V.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram static_diodev.sp
```

Action 3: Open the result file `static_diodev.static.xml` with a webbrowser.

The result file reports that the diodes `x1.d1` and `x1.d2` has a voltage greater than 0.5V.



Static Diode Voltage Check Violations

static_diodev: dv1

dv1 static_diodev model=["diode1"] cond="v(a,c)>=0.5"

Violation Count: 2

Static Diode Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Va	Vc
dv1	x1.d1	diode1	inv_esd	(0, 0)	(-5, 5)
dv1	x1.d2	diode1	inv_esd	(-5, 5)	(3.3, 3.3)

10. Static Capacitor and Resistor Check

Action 1: Open the netlist `static_res_cap.sp`, review the circuit and the design check statements.

```
chk1 static_resistor type=distr

chk2 static_resistor type=print rmin=1k rmax=1M

chk3 static_capacitor type=distr

chk4 static_capacitor type=print cmin=1p cmax=1u
```

The `static_capacitor` check reports capacitors outside or within the range defined by `cmin` and `cmax`. For example, if `type` is set to `range`, then check will report each capacitor outside the range of `cmin` and `cmax` (capacitor < `cmin` OR capacitor > `cmax`). If `type` is set to `print`, then check will print a report each capacitor within the range of `cmin` and `cmax` (`cmin` <= capacitors <= `cmax`). If `type` is set to `distr`, then check will print a distribution (or statistical) report.

Similarly, the `static_resistor` check follows the same behavior for resistors but with `rmin` and `rmax`.

Action 2: Run the netlist with spectre

```
% spectre +spice +xps +cktpreset=sram static_res_cap.sp
```

Action 3: Open the result file `static_res_cap.static.xml` with a webbrowser.

The first statement with `type=distr` will print a report in the xml file as shown below.

Static Resistor Check Violations

static_resistor: chk1

chk1 static_resistor type=distr

Violation Count: 12

Static Resistor Check - Resistor Value Distribution

Title	Range	Count
chk1	(-Inf, 0)	0
chk1	[0, 1 m)	1
chk1	[1 m, 10 m)	1
chk1	[10 m, 100 m)	1
chk1	[100 m, 1)	1
chk1	[1 , 10)	1
chk1	[10 , 100)	1
chk1	[100 , 1 K)	1
chk1	[1 K, 10 K)	1
chk1	[10 K, 100 K)	1
chk1	[100 K, 1 M)	1
chk1	[1 M, Inf)	2

The second statement with `type=print` will print a report in the xml file as shown below.

static_resistor: chk2

chk2 static_resistor type=print rmin=1000 rmax=1e+06

Violation Count: 3

Static Resistor Check - Resistor Value Report

Title	Device name	Resistance
chk2	r3	5.000000e+03
chk2	r4	5.000000e+04
chk2	r5	5.000000e+05

Similarly, the third and fourth statements will print a report for capacitors in the xml file as shown below.

Static Capacitor Check Violations

static_capacitor: chk3

chk3 static_capacitor type=distr

Violation Count: 9

Static Capacitor Check - Capacitor Value Distribution

Title	Range	Count
chk3	(-Inf, 0)	0
chk3	[0, 10 a)	1
chk3	[10 a, 100 a)	1
chk3	[100 a, 1 f)	1
chk3	[1 f, 10 f)	1
chk3	[10 f, 100 f)	1
chk3	[100 f, 1 p)	1
chk3	[1 p, 10 p)	1
chk3	[10 p, Inf)	5

Notice that the device names can be sorted by clicking the cell highlighted in red box.

static_capacitor: chk4

chk4 static_capacitor type=print cmin=1e-15 cmax=1e-12

Violation Count: 3

Static Capacitor Check - Capacitor Value Report

Title	Device name	Capacitance(F)
chk4	c1	5.000000e-13
chk4	x1.c3	5.000000e-15
chk4	x1.c4	5.000000e-14

11. Static ERC Check: Floatgate

Action 1: Open the netlist `static_erc_floatgate.scs`, review the circuit and the design check statement.

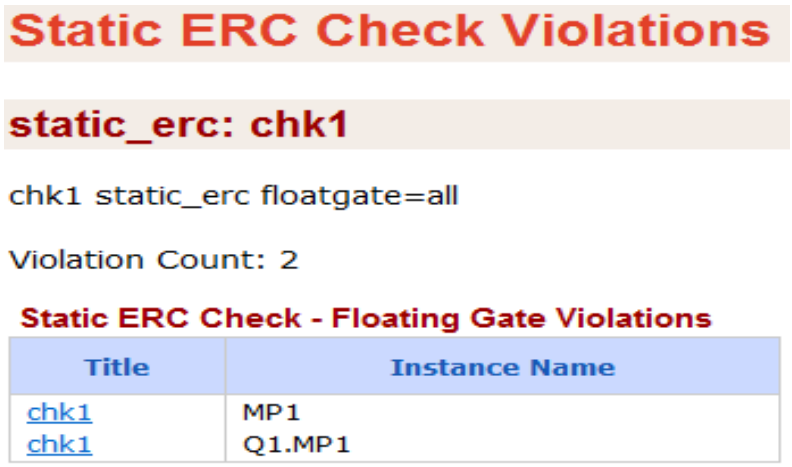
```
chk1 static_erc floatgate=all
```

The above statement checks all MOSFETs and reports if their gates are floating. A gate is considered floating if a voltage cannot propagate to that gate.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_floatgate.scs
```

Action 3: Open the result file `static_erc_floatgate.static.xml` with a webbrowser.



Static ERC Check Violations

static_erc: chk1

chk1 static_erc floatgate=all

Violation Count: 2

Static ERC Check - Floating Gate Violations

Title	Instance Name
chk1	MP1
chk1	Q1.MP1

In this example, the check reports three ERC floating gate violations. Please check the circuit to determine the reasons for these violations.

It is to be noted that if `floatgate=no_top` then MP1 will not be reported because MP1 is a top level transistor. Please read the other checks, with these different `floatgate` parameters, given in the netlist and their corresponding reports in the `static_erc_floatgate.static.xml`.

12. Static ERC Check: Floatbulk

Action 1: Open the netlist `static_erc_floatbulk.scs`, review the circuit and the design check statement.

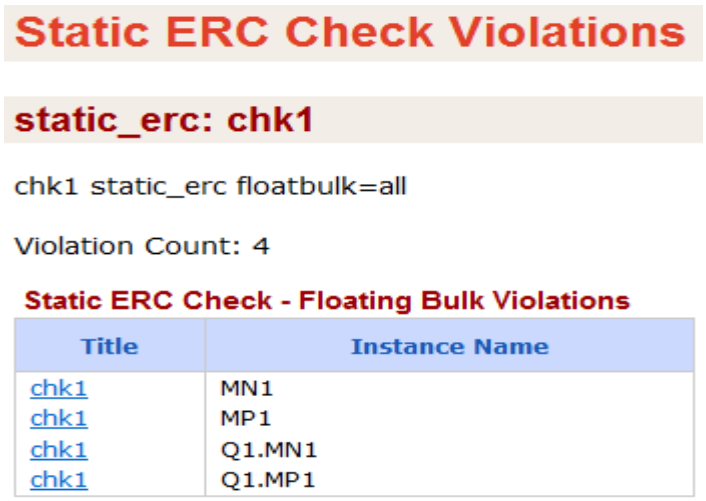
```
chk1 static_erc floatbulk=all
```

The above statement checks all MOSFETs and reports if their bulks are floating. A bulk is considered floating if a voltage cannot propagate to that bulk.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_floatbulk.scs
```

Action 3: Open the result file `static_erc_floatbulk.static.xml` with a webbrowser.



Static ERC Check Violations

static_erc: chk1

chk1 static_erc floatbulk=all

Violation Count: 4

Static ERC Check - Floating Bulk Violations

Title	Instance Name
chk1	MN1
chk1	MP1
chk1	Q1.MN1
chk1	Q1.MP1

In this example, the check reports four ERC float bulk violations. Please check the circuit to determine the reason for these violations.

It is to be noted that if `floatbulk=no_top` then MN1 and MP1 will not be reported because both are in a top level netlist. Please check the second check, with `floatgate=no_top`, given in the netlist and its corresponding reports in the `static_erc_floatbulk.static.xml`.

13. Static ERC Check: hotwell

Action 1: Open the netlist `static_erc_hotwell.scs`, review the circuit and the design check statement.

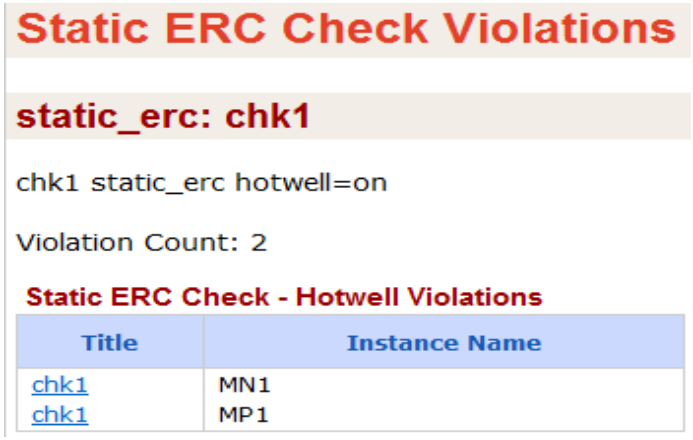
```
chk1 static_erc hotwell=all
```

The above statement checks all MOSFETs and reports if their bulks are not connected to either VDD or GND.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_hotwell.scs
```

Action 3: Open the result file `static_erc_hotwell.static.xml` with a webbrowser.



Static ERC Check Violations

static_erc: chk1

chk1 static_erc hotwell=on

Violation Count: 2

Static ERC Check - Hotwell Violations

Title	Instance Name
chk1	MN1
chk1	MP1

In this example, the check reports two ERC hotwell violations. Please check the circuit to determine the reason for these violations.

14. Static ERC Check: dangle

Action 1: Open the netlist `static_erc_dangle.scs`, review the circuit and the design check statement.

```
chk1 static_erc dangle=all
```

The above statement checks all nodes of all MOSFETs and reports dangling nodes. A node is considered dangling if a voltage cannot propagate to that node.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_dangle.scs
```

Action 3: Open the result file `static_erc_dangle.static.xml` with a webbrowser.

Static ERC Check Violations

static_erc: chk1

chk1 static_erc dangle=all

Violation Count: 6

Static ERC Check - Dangling Node Violations

Title	Node Name
chk1	dan1
chk1	dan2
chk1	dan3
chk1	Q1.dan1
chk1	Q1.dan2
chk1	Q1.dan3

In this example, the check reports six ERC dangling-node violations. Please check the circuit to determine the reason for these violations. Notice that the voltage cannot propagate to these six nodes, causing them to dangle.

It is to be noted that if `dangle=no_top` then nodes dan1, dan2 and dan3 will not be reported because they are in a top level netlist. Please check the second check, with `dangle=no_top`, given in the netlist and its corresponding reports in the `static_erc_dangle.static.xml`.

15. Static ERC Check: gate2power

Action 1: Open the netlist `static_erc_gate2power.scs`, review the circuit and the design check statement.

```
chk1 static_erc gate2power=on vlth=0.1 vhth=1.0
```

The above statement checks all gates of all MOSFETs and reports PMOS connected to GND or NMOS connected to VDD. A voltage source is considered as VDD if its voltage is above `vhth`. A voltage source is considered GND if its voltage is below `vlth`.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_gate2power.scs
```

Action 3: Open the result file `static_erc_gate2power.static.xml` with a webbrowser.

Static ERC Check Violations

static_erc: chk1

chk1 static_erc gate2power=on vlth=0.1 vhth=1

Violation Count: 2

Static ERC Check - Gate2Power Violations

Title	Gate2Power Inst Name	Gate Terminal	Gate Voltage(V)
chk1	MN2	in2	1.100000e+00
chk1	MP1	in1	0.000000e+00

In this example, the check reports that the NMOS, *MN2*, is connected to gate voltage of 1.1 and the PMOS, *MP1*, is connected to gate voltage of 0. Please check the circuit to determine the reason for these violations.

16. Static ERC Check: rmax

Action 1: Open the netlist `static_erc_rmax.scs`, review the circuit and the design check statement.

```
chk1 static_erc floatgate=all rmax=1k

chk2 static_erc floatgate=all
```

The above statements check all nodes of all MOSFETs and report floating gates. A node is considered floating if a voltage cannot propagate to that node. The voltage cannot propagate through those resistors whose resistance is more than rmax. The default value for rmax is 100M ohms.

Action 2: Run the netlist with spectre

```
% spectre +xps +cktpreset=sram static_erc_rmax.scs
```

Action 3: Open the result file `static_erc_rmax.static.xml` with a webbrowser.

Static ERC Check Violations

static_erc: chk1

chk1 static_erc floatgate=all rmax=1000

Violation Count: 2

Static ERC Check - Floating Gate Violations

Title	Instance Name
chk1	Q1.MN1
chk1	Q1.MP1

The first check reports two ERC floating-gate violations. Please check the circuit to determine the reason for these violations. Notice that the voltage cannot propagate to the gates of MOSFETs Q1.MN1 and Q1.MP1 because voltage cannot propagate through resistor Q1.R1, causing them to float. The voltage cannot propagate through resistor Q1.R1 because its value is larger than rmax=1k for this first check

The second check will report no violations because rmax is 100M ohms by default and voltage can propagate through all resistors in this circuit. Please check the second check given in the netlist and its corresponding reports in the `static_erc_rmax.static.xml`.

17. Static RC delay Check

Action 1: Open the netlist `static_rcdelay.scs`, review the circuit and the design check statement.

```
chk2 static_rcdelay node=[*] minnrise=2 minnfall=2 maxnrise=2 maxnfall=2
```

The above statement reports nodes with extreme rise and fall times. This check **estimates** their rise or fall time in terms of charging or discharging and report the times too large or too small as violations.

The `minrise=2`, reports the two lowest rise-times. The `minfall=2`, reports the two lowest fall-times. The `maxnrise=2`, reports the two highest rise-times. The `maxnfall=2`, reports the two highest fall-times. There will be a table for each parameter. At least one parameter is required to run this check.

Action 2: Run the netlist with spectre

```
% spectre +spice +xps +cktpreset=sram static_rcdelay.sp
```


Action 3: Open the result file `static_rcdelay.static.xml` with a webbrowser.

There will be four tables as shown below.

Static RC Delay Check - Max Rise-time Delays

Title	Node Name	Delay(s)	Receiver	Driver	Node Cap(F)
chk2	E	5.210000e-10	x_d2.mn2	x_r1.mp1	2.029857e-13
chk2	C	2.470000e-10	x_r1.mp2	x_t1.mp1	3.891608e-13

The above report shows the top rise-times in descending order. The node E has the maximum rise time is 0.521ns. The path to this node from VDD is through x_r1.mp1 and x_r1.mp2. The node C is has the second maximum rise-time of 0.247ns. The path to this node from VDD is through x_t1.mp1. In both cases, the Driver is a first MOSFET connected to the VDD.

Static RC Delay Check - Min Rise-time Delays

Title	Node Name	Delay(s)	Receiver	Driver	Node Cap(F)
chk2	D	1.340000e-10	x_d2.mp1	x_d1.mp2	2.099902e-13
chk2	F	2.140000e-10	x_t2.mp1	x_d2.mp2	3.344123e-13

The above report shows the bottom rise-times in ascending order. The node D has the minimum rise time is 0.134ns. The path to this node from VDD is through x_d1.mp2. The node F is has the second minimum rise-time of 0.214ns. The path to this node from VDD is through x_d2.mp1. In both cases, the Driver is a first MOSFET connected to the VDD.

Static RC Delay Check - Max Fall-time Delays

Title	Node Name	Delay(s)	Receiver	Driver	Node Cap(F)
chk2	F	8.640000e-10	x_t2.mp1	x_d2.mn2	3.344123e-13
chk2	D	5.430000e-10	x_d2.mp1	x_d1.mn2	2.099902e-13

The above report shows the top fall-times in descending order. The node F has the maximum fall-time is 0.864ns. The path from this node to GND is through x_d2.mn1 and x_d2.mn2. The node D has the second maximum fall-time of 0.543ns. The path from this node to GND is through x_d1.mn1 and x_d1.mn2. In both cases, the Driver is a first MOSFET connected to the GND.

Static RC Delay Check - Min Fall-time Delays

Title	Node Name	Delay(s)	Receiver	Driver	Node Cap(F)
chk2	E	2.620000e-10	x_d2.mn2	x_r1.mn1	2.029857e-13
chk2	C	5.030000e-10	x_r1.mp2	x_t1.mn2	3.891608e-13

The above report shows the bottom fall-times in ascending order. The node E has the minimum fall-time is 0.262ns. The path from this node to GND is through x_r1.mn1. The node C has the

second minimum fall-time of 0.503ns. The path from this node to GND is through x_t1.mn2. In both cases, the Driver is a first MOSFET connected to the GND.

In all reports, the Receiver is the random MOSFET with gate connection to the reported Node. The Node Cap is the capacitance of the reported Node.

Workshop exercise for dynamic check

In this section, different dynamic checks are introduced and applied to simple circuits with their output shown.

1. Dynamic high impedance node check

Highz node definition:

A highz node is a node that does not have a path to VDD or GND. In other words, if a node has a path to VDD then it's not a highz node. If a node has a path to GND then it's not a highz node. If a node does NOT have a path to VDD and GND then it's a highz node. For more details please see "Appendix: Floating node detection".

Action 1: Open the netlist `dyn_highz.scs`, review the circuit and the design check statement.

```
hz1 dyn_highz node=[*] duration=2n time_window=[0n 20n] mos_ith=100n
mos_gds=10u
```

The above statement checks for any high impedance node in the design with duration larger than 2ns, within the time window from 0ns to 20ns.

Action 2: Run the netlist with spectre

```
% spectre dyn_highz.scs
```

Action 3: Open the result file `dyn_highz.dynamic.xml` with a webbrowser.

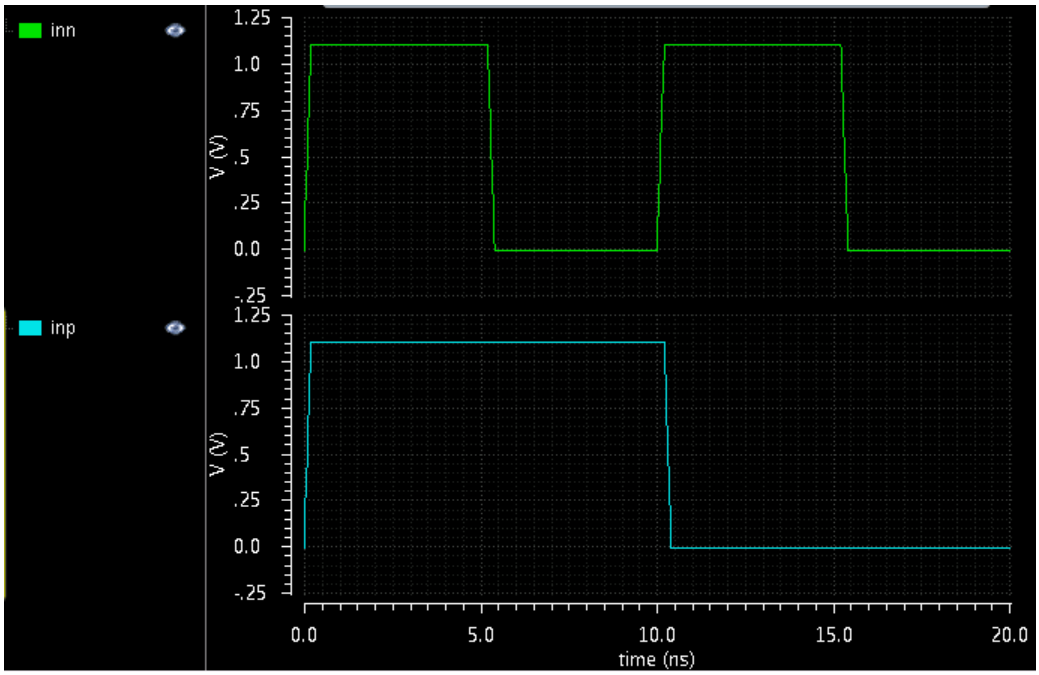


Figure 1: Two PWL sources

Check the connectivity of the devices in the netlist `dyn_highz.scs`. This simple circuit uses two PWL sources as described in Figure 1. Please find which node will have high impedance and at what time range. Please verify your finding with the check result.

The result file reports that the node `out` is a high impedance node. The violation happens for the duration of 4.8ns, staring at 5.3ns. Moreover, node `floatBulk` is also a high impedance node because it has not path to GND.

Dynamic HighZ Node Check Violations

dyn_highz: hz1

hz1 dyn_highz node=["*"] duration=2e-09 time_window=[0 2e-08] mos_ith=0.0001 mos_gds=0.0002

Violation Count: 2

Title	Node Name	Start(s)	Duration(s)
hz1	out	5.327138e-09	4.769384e-09
hz1	floatBulk	0.000000e+00	2.000000e-08

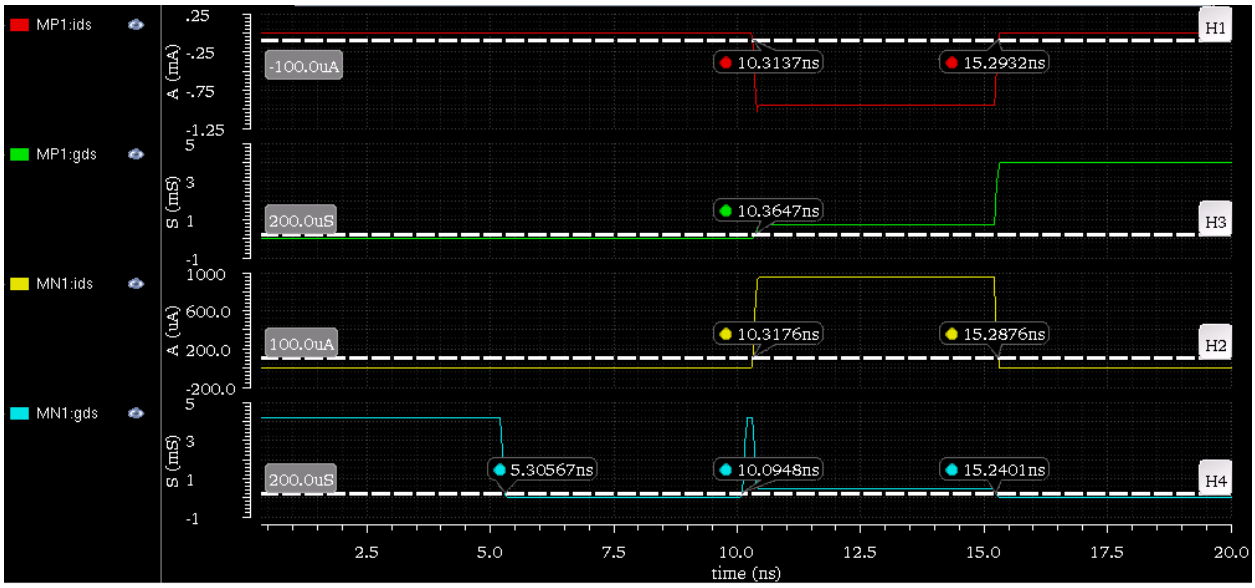


Figure 2: ids and gds values of MP1 and MN1

The conducting rule for NMOS is "MN1:ids > 100uA or MN1:gds > 200uS". Therefore, MN1 is in OFF state from 5.3s to 10s and 15.2s to 20s.

The conducting rule for PMOS is "abs(MP1:ids) > 100uA or MP1:gds > 200uS". Therefore, MP1 is in OFF state from 0s to 10.3s. Notice that the direction of current for PMOS is opposite of NMOS.

Approximate Time Window	State of MN1 MN1:ids > 100uA or MN1:gds > 200uS	State of MP1 abs(MP1:ids) > 100uA or MP1:gds > 200uS	Is node `out` floating?
0s to 5s	ON	OFF	No
5s to 10s	OFF	OFF	Yes
10s to 15s	ON	ON	No
15s to 20s	OFF	ON	No

Table 1: State of MN1 and MP1

The Table 1 shows that both MN1 and MP1 are in OFF state from 5s to 10s. Therefore, the node `out` is a highz (or floating) node from 5s to 10s.

If you are interested in knowing which node has path to VDD or GND, meaning non-highz node, then you can set parameter inverse to yes.

dyn_highz: hz2

hz2 dyn_highz node=["out"] duration=2e-09 time_window=[0 2e-08] mos_ith=0.0001 mos_gds=0.0002 inverse=yes

Violation Count: 2

Title	Node Name	Start(s)	Duration(s)
hz2	out	1.162918e-10	5.210846e-09
hz2	out	1.009652e-08	9.903477e-09

The Table 1 shows that either MN1 or MP1 are in ON state from 0s to 5s and 10s to 20s time windows. Therefore, the node `out` is a non-highz (or not-floating) nodes for duration of 5.2ns, starting from 1.2ns and for duration for 9.9ns, starting from 10.1ns.

The check will report all floating nodes by default, as shown in check `hz1`. Sometime, user is interested only in nodes connected to gates of MOSFETs. This can be achieved by filtering out such node using fanout parameter. By setting fanout=gate, only highz node connected to gates will be reported.

dyn_highz: hz3

hz3 dyn_highz node=["*"] duration=2e-09 time_window=[0 2e-08] mos_ith=0.0001 mos_gds=0.0002 fanout=gate

Violation Count: 1

Title	Node Name	Start(s)	Duration(s)
hz3	out	5.327138e-09	4.769384e-09

Notice that if user set mos_ith and mos_gds too low, then most of the MOSFETs will be in ON state and no floating node will be reported. On the contrary, if user set mos_ith and mos_gds too high, then most of the MOSFETs will be in OFF state and many floating node will be reported. If user does not set the mos_ith and mos_gds parameters then their default values will be used. Please see the default values in help by typing %spectre –h dyn_highz.

2. Dynamic DC leakage current path check

Action 1: Open the netlist `dyn_dcpath.scs`, review the circuit and the design check statement.

```
dcpath1 dyn_dcpath ith=200u duration=1n net=[vdd 0] time_window=[1n 20n]
```


The above statement checks for any DC paths between voltage sources, "vdd" and "0", with current larger than 200uA and for duration longer than 1ns, within the time window from 1ns to 20ns.

Action 2: Run the netlist with spectre

```
% spectre dyn_dcpath.scs
```

Action 3: Open the result file `dyn_dcpath.dynamic.xml` with a webbrowser.

This simple circuit uses the same sources as described in Figure 1. Please find any DC path violation within given time range. Please verify your finding with the check result.

The result file reports a DC path through PMOS "MP1" and NMOS "MN1". The violation happens for the duration of 4.9ns, starting at 10ns.

Dynamic DC Leakage Path Check Violations

dyn_dcpath: dcpath1

dcpath1 dyn_dcpath ith=0.0002 duration=1e-09 net=["vdd" "0"] time_window=[1e-09 2e-08]

Violation Count: 1

Title	From Net	To Net	Start(s)	Duration(s)
dcpath1	vdd	0	1.033611e-08	4.937547e-09

Path Elements:

MP1

MN1

3. Dynamic floating gate induced leakage check

Action 1: Open the netlist `dyn_floatdcpath.scs`, review the circuit and the design check statements.

```
dyn1 dyn_floatdcpath net=[vdd1 vdd2 0] leaki_times=[7.5n 12.5n]
mos_ith=100u mos_gds=200u floatgate=yes
```

The above statements will check for any leakage paths between the nets that are induced by floating gates at leaki_times. The check will do following three steps.

First, this check will internally do the leakage analysis at times, 7.5ns and 12.5ns, to determine the floating nodes. This leakage analysis is reported in the log by title "DC Analysis `tran_7.5e-09_leaki'" and "DC Analysis `tran_1.25e-08_leaki'".

Second, the floating node detection algorithm will use the DC solution to detect floating nodes. All MOSFETs with floating gates can be reported using parameter floatgate=yes (default is no). For more details about algorithm please see “Appendix: Floating node detection”.

Third, this check will report any leakage path from power supplies (`vdd1, vdd2`) to ground, induce by floating gates.

If you don’t want to run third step and only interested in floating gates, then use the following statement without net parameter.

```
dyn2 dyn_floatdcpath leaki_times=[7.5n 12.5n] mos_ith=100u mos_gds=200u
floatgate=yes
```

Action 2: Run the netlist with spectre

```
% spectre dyn_floatdcpath.scs
```

Action 3: Open the result file `dyn_floatdcpath.dynamic.xml` with a webbrowser.

This simple circuit uses the same sources as described in Figure 1. Please find any floating gate at 7.5ns and leakage path induced by such floating gate. Please verify your finding with the check result.

The result file reports a DC leakage path through *path elements* “`MP2`” and “`MN2`” that is induced by a *floating node* `out` at time 7.5ns.

Dynamic Floating Node Induced DC Leakage Path Check - MOSFET with Floating Gate

Title	Violation Time(s)	MOSFET	Float Gate Node
dyn1	7.500000e-09	MP2	out
dyn1	7.500000e-09	MN2	out

Dynamic Floating Node Induced DC Leakage Path Check Violations

dyn_floatdcpath: dyn1

dyn1 dyn_floatdcpath net=["vdd1" "vdd2" "0"] leaki_times=[7.5e-09 1.25e-08] mos_ith=0.0001 mos_gds=0.0002 floatgate=yes

Violation Count: 3

Dynamic Floating Node Induced DC Leakage Path Check - DC Path Violations

Title	From Net	To Net	Violation Time(s)
dyn1	vdd2	0	7.500000e-09

Path Elements:

MP2

MN2

Floatnodes Nodes:

out

Floatelems Elements:

MP2

MN2

4. Dynamic floating gate induced leakage check: Filtering Path per Floating MOSFET

Action 1: Open the netlist `dyn_floatdcpath_path_per_fm.scs`, review the circuit and the design check statements.

```
dynA1 dyn_floatdcpath net=[vdd1 0] leaki_times=[7.5n] mos_ith=100u
mos_gds=200u floatgate=yes detailed_path=yes
```

The above statements will check for any leakage paths between the nets that are induced by floating gates at leaki_times. The check will report all possible paths because value of detailed_path is set to yes.

Action 2: Run the netlist with spectre

```
% spectre dyn_floatdcpath_path_per_fm.scs
```

Action 3: Open the result file `dyn_floatdcpath_path_per_fm.dynamic.xml` with a webbrowser.

Please find any floating gate at 7.5ns and leakage path induced by such floating gate. Please verify your finding with the check result.

The result file reports two DC leakage paths through similar *path elements* "`MNA1`" that is induced by a *floating node* `float` at time 7.5ns.

Path Elements:	**Path Elements:**
MPA2	MPA1
MNA1	MNA1
MNA2	MNA2
Floatnodes Nodes:	**Floatnodes Nodes:**
float	float
Floatelems Elements:	**Floatelems Elements:**
MNA1	MNA1

To see one path per floating MOSFET, i.e MNA1, use the per_fm option for the parameter detailed_path, as shown in the following statement:

```
DynA2 dyn_floatdcpath net=[vdd1 0] leaki_times=[7.5n] mos_ith=100u
mos_gds=200u floatgate=yes detailed_path=per_fm
```

5. Dynamic floating gate induced leakage check: Filtering Path per Floating node

Action 4: Open the netlist `dyn_floatdcpath_path_per_fn.scs`, review the circuit and the design check statements.

```
dynB1 dyn_floatdcpath net=[vdd1 0] leaki_times=[7.5n] mos_ith=100u
mos_gds=200u floatgate=yes detailed_path=yes
```

The above statements will check for any leakage paths between the nets that are induced by floating gates at leaki_times. The check will report all possible paths because value of detailed_path is set to yes.

Action 5: Run the netlist with spectre

```
% spectre dyn_floatdcpath_path_per_fn.scs
```


Action 6: Open the result file `dyn_floatdcpath_path_per_fn.dynamic.xml` with a webbrowser.

Please find any floating gate at 7.5ns and leakage path induced by such floating gate. Please verify your finding with the check result.

The result file reports two DC leakage paths through similar *path elements* "`MNA1`" that is induced by a *floating node* `float` at time 7.5ns.

Path Elements:	**Path Elements:**
x_nor.MP1	x_nand.MP1
x_nor.MP2	x_nand.MN1
x_nor.MN1	x_nand.MN2
Floatnodes Nodes:	**Floatnodes Nodes:**
float	float
Floatelems Elements:	**Floatelems Elements:**
x_nor.MP2	x_nand.MP1
x_nor.MN1	x_nand.MN1

To see one path per floating Node, i.e float, use the per_fn option for the parameter detailed_path, as shown in the following statement:

```
dynB2 dyn_floatdcpath net=[vdd1 0] leaki_times=[7.5n] mos_ith=100u
mos_gds=200u floatgate=yes detailed_path=per_fn
```

Note that the per_fn option for detailed_path is only supported in APS and SPECTRE but not in XPS.

6. Dynamic MOSFET voltage check

Action 1: Open the netlist `dyn_mosv.sp`, review the circuit and the design check statement.

```
mos1 dyn_mosv model=[nmos] cond="v(g,s)>1.9" duration=2n
```

The above statement checks for any MOSFET with model `nmos` that satisfies the condition v(g,s)>1.9V for a duration longer than 2ns.


```
mos2 dyn_mosv model=[*] inst=[x2] cond="v(d)>1.8" duration=0.05n
```

The above statement checks for any MOSFET in instance x2 that has the drain voltage greater than 1.8V for the duration longer than 0.05ns.

```
mos3 dyn_mosv model=[*] cond="w<30e-6"
```

The above statement checks for any MOSFET that has width less than 30um.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram dyn_mosv.sp
```

Action 3: Open the result file `dyn_mosv.dynamic.xml` with a webbrowser.

Dynamic MOSFET Voltage Check Violations

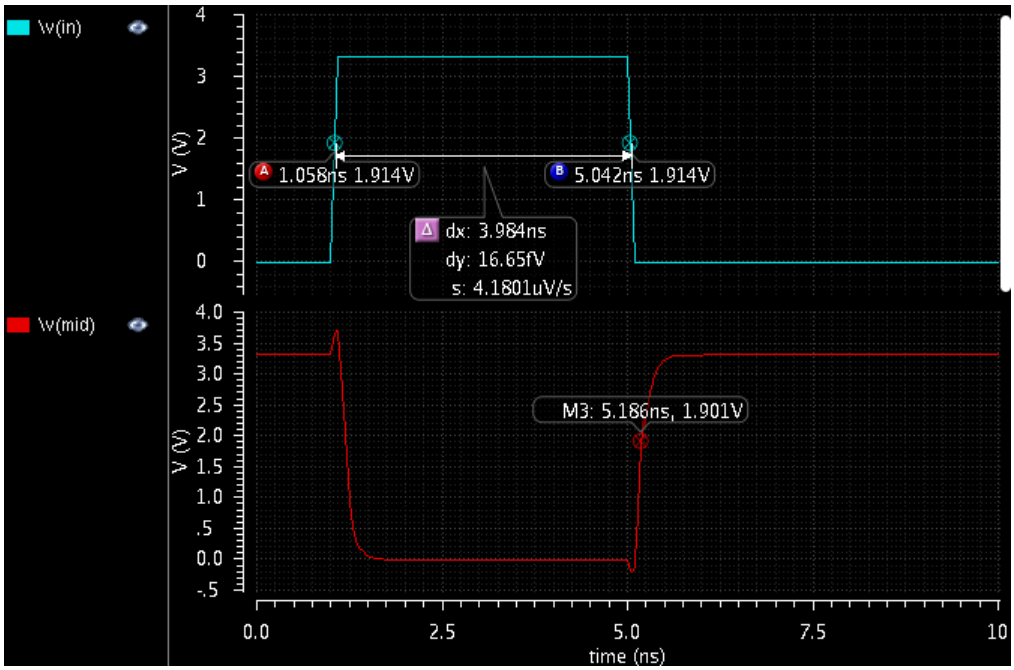
dyn_mosv: mos1

mos1 dyn_mosv model=["nmos"] cond="v(g,s)>1.9" duration=2e-09

Violation Count: 2

Title	Instance Name	Start(s)	Model Name	Drain Name	Gate Name	Source Name	Bulk Name	Duration(s)
mos1	x1.mn2	1.058000e-09	nmos	mid	in	gnd	gnd	3.984000e-09
mos1	x2.mn2	5.188000e-09	nmos	out	mid	gnd	gnd	4.812000e-09

In this example, the first check, `mos1`, reports two violations by *x1.mn2* and *x2.mn2*. The violation can be seen on the graph given below. The blue plot corresponds to v(g,s) of NMOS *x1.mn2*. The red plot corresponds to v(g,s) of NMOS *x2.mn2*.



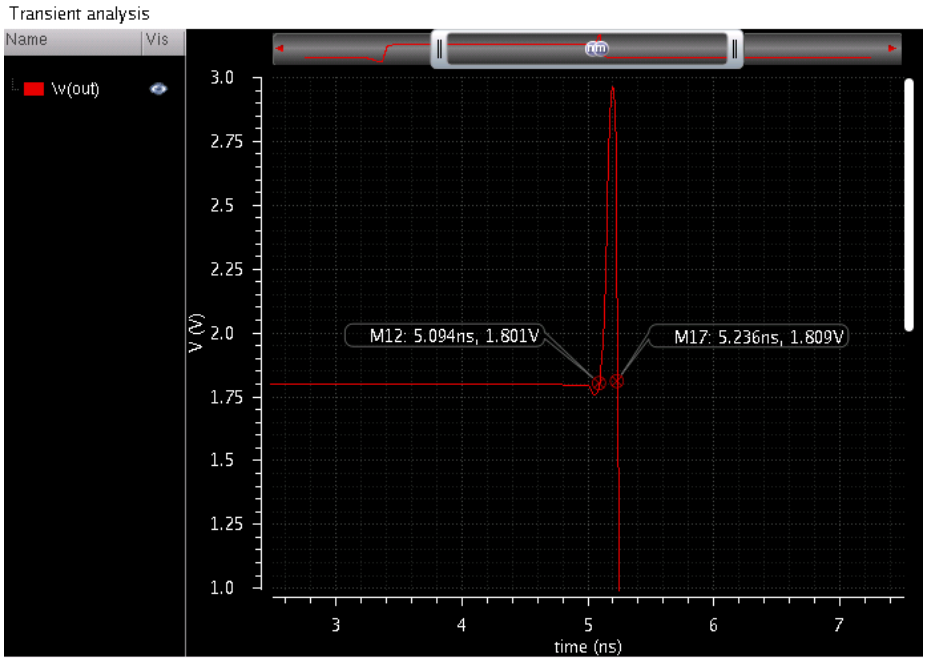
dyn_mosv: mos2

mos2 dyn_mosv model=["*"] inst=["x2"] cond="v(d)>1.8" duration=5e-11

Violation Count: 2

Title	Instance Name	Start(s)	Model Name	Drain Name	Gate Name	Source Name	Bulk Name	Duration(s)
mos2	x2.mp1	5.094000e-09	pmos	out	mid	vdd	vdd	1.420000e-10
mos2	x2.mn2	5.094000e-09	nmos	out	mid	gnd	gnd	1.420000e-10

The second check, `mos2`, reports two violations by *x2.mp1* and *x2.mn2* of an instance x2 that have a violation for duration longer than 0.05ns. The violation can be seen on the graph given below, which shows the voltage at the drain terminals (node `out`) of the devices.



dyn_mosv: mos3

mos3 dyn_mosv model=["*"] cond="w<30e-6"

Violation Count: 2

Title	Instance Name	Start(s)	Model Name	Drain Name	Gate Name	Source Name	Bulk Name	Duration(s)
mos3	x1.mn2	0.000000e+00	nmos	mid	in	gnd	gnd	1.000000e-08
mos3	x2.mn2	0.000000e+00	nmos	out	mid	gnd	gnd	1.000000e-08

The third check, `mos3`, reports two violations by *x1.mn2* and *x2.mn2* that have the width less than 30um. Please check the width of these two devices in the netlist.

This check is supported only by XPS. For Spectre and Spectre APS use assert statement.

7. Dynamic resistor/capacitor voltage check

Action 1: Open the netlist `dyn_resv_capv.sp`, review the circuit and the design check statements.

```
resv1 dyn_resv cond="v(1,2)>0.2" duration=0.5n
```

The above statement checks for any resistor that has a voltage across its terminals greater than 0.2V for duration longer than 0.5ns.

```
capv1 dyn_capv cond="v(1,2)>0.2" duration=0.5n
```

Similarly, the above statement checks for any capacitor that has a voltage across its terminals greater than 0.2V for duration longer than 0.5ns. Notice that “1” stands for positive terminal and “2” stands for negative terminal of a device.

Action 2: Run XPS on the netlist

```
% spectre +spice +xps +cktpreset=sram dyn_resv_capv.sp
```

Action 3: Open the result file `dyn_resv_capv.dynamic.xml` with a webbrowser.

Dynamic Resistor Voltage Check Violations

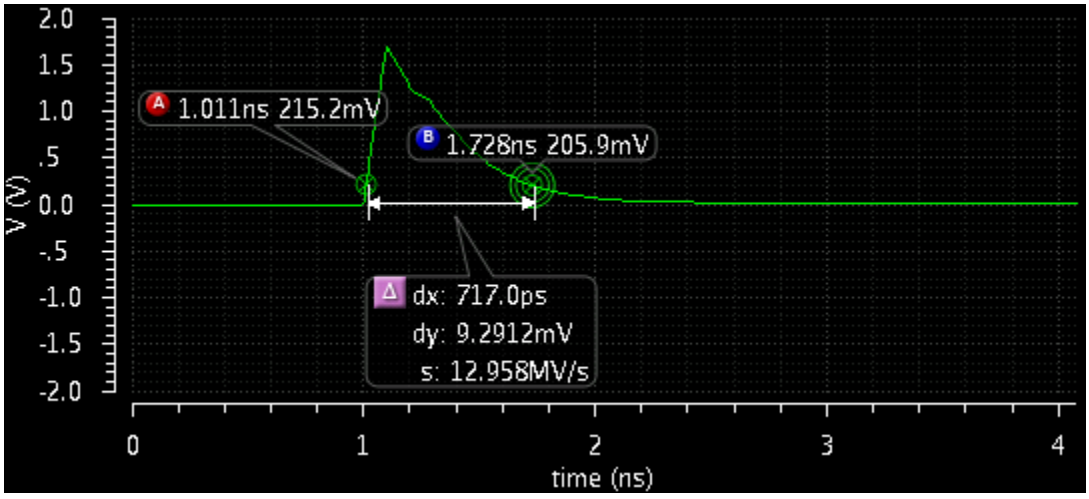
dyn_resv: resv1

resv1 dyn_resv cond="v(1,2)>0.2" duration=5e-10

Violation Count: 1

Title	Instance Name	Start(s)	Model Name	Term1 Name	Term2 Name	Duration(s)
resv1	r1	1.011000e-09	resistor	in	in1	7.170000e-10

The result file reports that the resistor r1 has voltage greater than 0.2V for the duration longer than 0.5ns. The voltage across the terminals of the resistor r1 is given in graph below. Notice that the violation starts at 1.011ns and stays for the duration of 0.717ns, which is longer than the duration given in the statement.



Similarly, the result file reports that the capacitor c1 has a voltage greater than 0.2V for duration longer than 0.5ns.

Dynamic Capacitor Voltage Check Violations

dyn_capv: capv2

capv2 dyn_capv cond="v(1,2)>0.2" duration=5e-10

Violation Count: 1

Title	Instance Name	Start(s)	Model Name	Term1 Name	Term2 Name	Duration(s)
capv2	c1	1.011000e-09	capacitor	in	in1	7.170000e-10

This check is supported only by XPS. For Spectre and Spectre APS use assert statement.

8. Dynamic diode voltage Check

Action 1: Open the netlist `dyn_diodev.sp`, review the circuit and the design check statements.

```
dv1 dyn_diodev model=diode1 cond="v(a,c)>0.5" duration=25E-09
```

The above statements will check for any diode with model name *diode1* that has a voltage across its terminals more than 0.5V for duration longer then 25ns.

Action 2: Run the netlist with XPS

```
% spectre +spice +xps +cktpreset=sram dyn_diodev.sp
```

Action 3: Open the result file `dyn_diodev.dynamic.xml` with a webbrowser.

The result file reports two diodes "`d1`" and "`d2`" that has violation. The diode "`d1`" has a violation starting at 51.7ns for the duration of 52.8ns. The diode "`d2`" has a violation starting at 4ns for the duration of 46.4ns.

Dynamic Diode Voltage Check Violations

dyn_diodev: dv1

dv1 dyn_diodev model=["diode1"] cond="v(a,c)>0.5" duration=2.5e-08

Violation Count: 2

Title	Instance Name	Start(s)	Model Name	Anode Name	Cathod Name	Duration(s)
dv1	d2	3.968000e-09	diode1	mid	vdd	4.638400e-08
dv1	d1	5.169600e-08	diode1	0	mid	5.288000e-08

Please analyze the netlist to identify this violation. This check is supported only by XPS. For Spectre and Spectre APS use assert statement.

9. Dynamic excessive element current check

Action 1: Open the netlist `dyn_exi.scs`, review the circuit and the design check statement.

```
exi1 dyn_exi dev=[*] ith=900u duration=2n time_window=[1n 20n]
```

The above statement checks for any device that carries a current greater than 900uA for duration longer than 2ns, in the time window 1ns to 20ns.

Action 2: Run the netlist with spectre

```
% spectre dyn_exi.scs
```

Action 3: Open the result file `dyn_exi.dynamic.xml` with a webbrowser.

In this example, the first check, `exi1`, reports two violations by `MN1` and `MP1`. The violation starts at 10ns and remains for the duration of 4.8ns. The maximum current during this violation is also given in the report.

Dynamic Excessive Element Current Check Violations

dyn_exi: exi1

exi1 dyn_exi dev=["*"] ith=0.0009 duration=2e-09 time_window=[1e-09 2e-08]

Violation Count: 2

Title	Instance Name	Start(s)	Duration(s)	Max Current(A)
exi1	MP1	1.039199e-08	4.815322e-09	-9.555268e-04
exi1	MN1	1.039199e-08	4.815319e-09	9.555272e-04

10. Dynamic excessive rise and fall time check

Action 1: Open the netlist `dyn_exrf.sp`, review the circuit and the design check statement.

```
exrf1 dyn_exrf node=[out out1] rise=500p fall=500p utime=1n vlth=0.3
vhth=2.7 time_window=[1n 9n]
```

The above statement checks for the nodes `out` and `out1` that has either rise/fall time longer than 500ps or undefined state longer than 1ns, in time window of 1ns to 9ns. The rise time is defined as the time taken by a waveform from `vlth` to `vhth`. Similarly, fall time is defined as the time taken by a waveform from `vhth` to `vlth`. Undefined state is defined as the time when the waveform is between `vlth` and `vhth`.

Action 2: Run the netlist with spectre


```
% spectre +spice dyn_exrf.sp
```

Action 3: Open the result file `dyn_exrf.dynamic.xml` with a webbrowser.

Dynamic Excessive Rise, Fall, Undefined State Time Check Violations

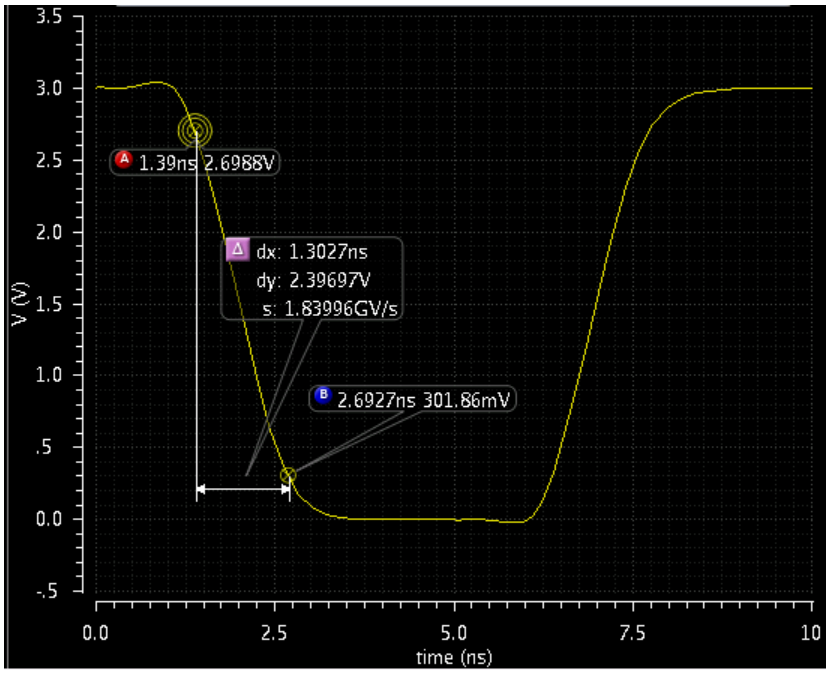
dyn_exrf: exrf1

exrf1 dyn_exrf node=["out" "out1"] rise=5e-10 fall=5e-10 utime=1e-09 vlth=0.3 vhth=2.7 time_window=[1e-09 9e-09]

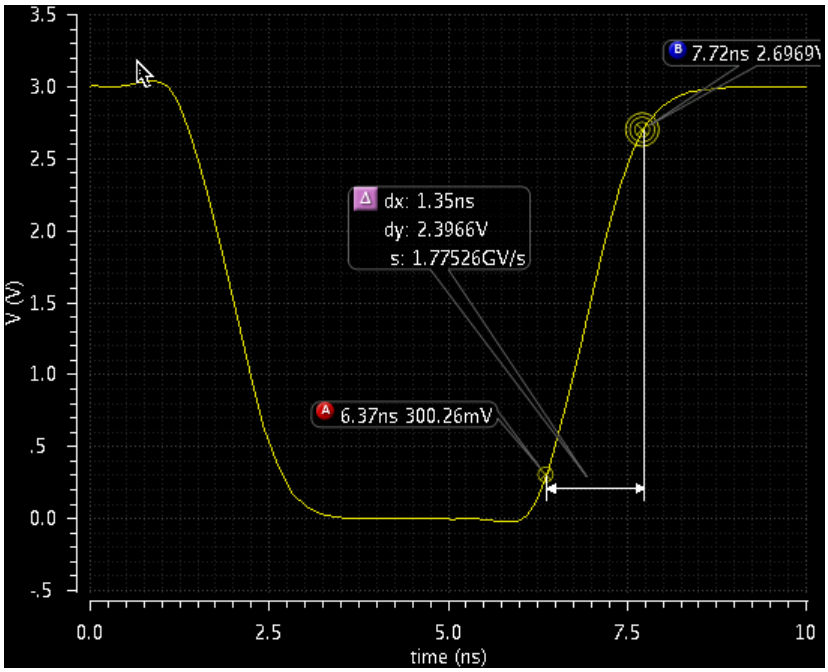
Violation Count: 3

Title	Node Name	Type	Start(s)	Duration(s)
exrf1	out	fall	1.389296e-09	1.305463e-09
exrf1	out	rise	6.369792e-09	1.353608e-09
exrf1	out1	utime	1.440439e-09	7.734277e-09

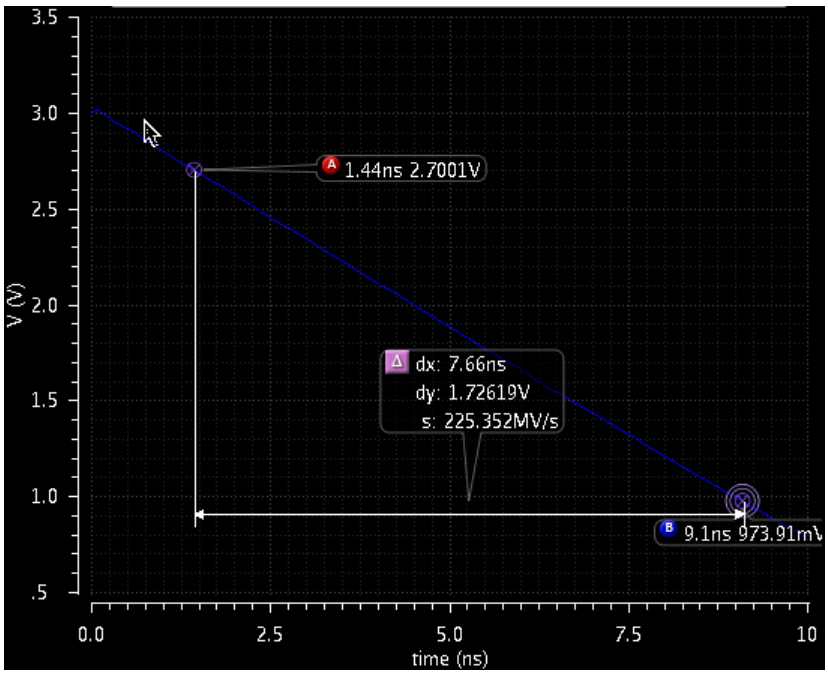
The above result-table reports that the node *out* has a fall time of 1.30ns, starting at 1.39ns. The fall time of *out* is shown in the plot described below. Notice that the fall time is longer than 500ps, therefore, it violates the condition given in the design statement.



Similarly, the result-table reports that the node *out* has a rise time of 1.35ns, starting at 6.37ns. The rise time of *out* is shown in the plot described below. Notice that the rise time is longer than 500ps that violates the condition given in the design statement.



The result-table reports that the node *out1* has an undefined time of 7.7ns, starting at 1.4ns. The undefined time of *out1* is shown in the plot described below. Notice that the undefined time is longer than 1ns that violates the condition given in the design statement within the time window of 1ns to 9ns.



11. Dynamic glitch check

Action 1: Open the netlist `dyn_glitch.sp`, review the circuit and the design check statement.


```
dyn_glitch1 dyn_glitch node=[IN OUT] duration=1n low=0 high=1.2
```

The above statement checks the nodes `IN` and `OUT` for a glitch for duration *shorter* than 1ns. A glitch occurs when:

■ A low signal goes above the mid level, and crosses the mid level again in a time less than the user-defined duration.

■ A high signal goes below mid level, and crosses the mid level again in a time less than the user-defined duration

The mid-level is equal to 0.5*(low+high). In this statement, the values for low and high are 0V and 1.2V, respectively, therefore, mid-level in equal to 0.6V.

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_glitch.sp
```

Action 3: Open the result file `dyn_glitch.dynamic.xml` with a webbrowser.

Dynamic Glitch Check Violations

dyn_glitch: dyn_glitch1

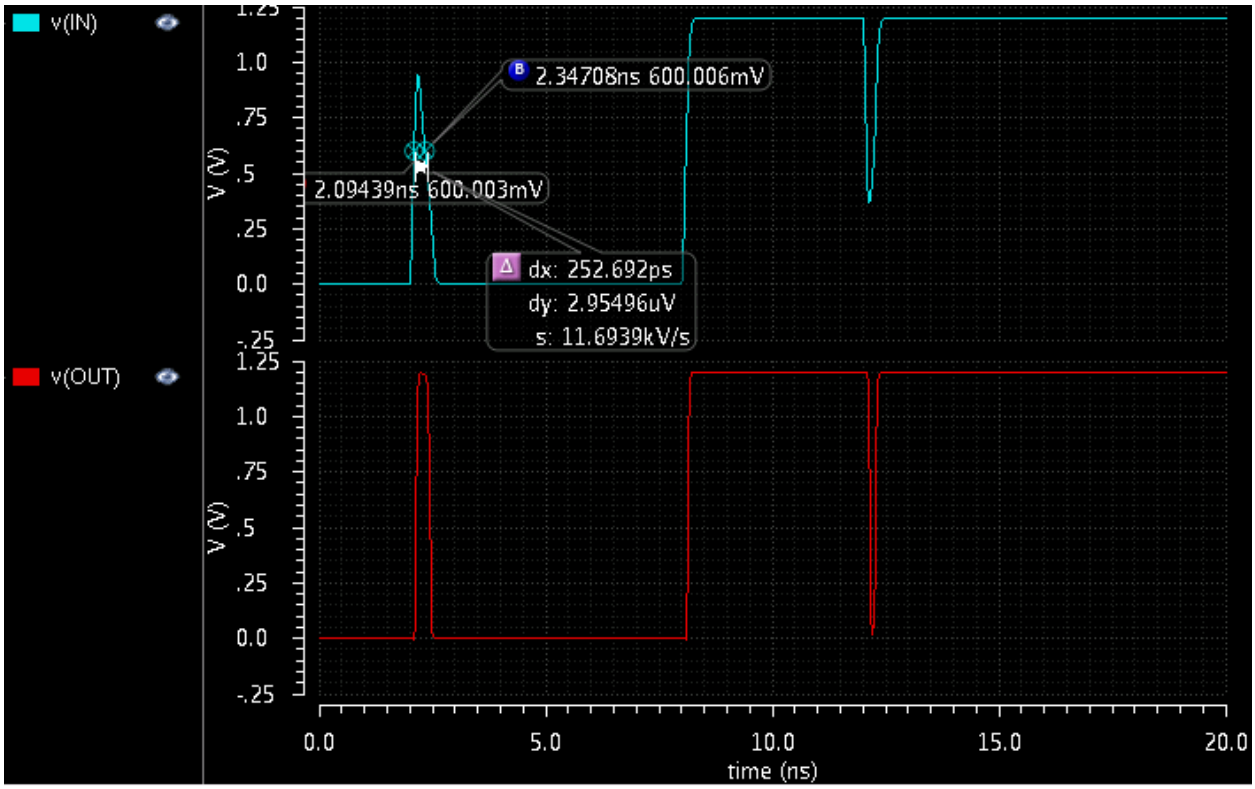
dyn_glitch1 dyn_glitch node=["IN" "OUT"] duration=1e-09 low=0 high=1.2

Violation Count: 4

Title	Node Name	Start(s)	Duration(s)	Peak Value(V)	Static Voltage(V)
dyn_glitch1	IN	2.094388e-09	2.526949e-10	9.414577e-01	0.000000e+00
dyn_glitch1	OUT	2.139668e-09	3.079011e-10	1.197572e+00	0.000000e+00
dyn_glitch1	IN	1.208683e-08	1.582034e-10	3.630733e-01	1.200000e+00
dyn_glitch1	OUT	1.215317e-08	1.312476e-10	1.544449e-02	1.200000e+00

In this example, the check reports two nodes with violations. For node *IN*, the crest is greater than “low (static) voltage (0V) + 0.6V” around 2.09ns for duration of 0.25ns. Also, for node *IN*, it reports the trough that is less than “high (static) voltage (1.2V) - 0.6V” around 1.2ns for duration 0.16ns. Notice that the duration for both crest and trough of node *IN* is shorter than the duration given in the check statement.

Please see the plot described below for more details.



12. Dynamic setup and hold check

Action 1: Open the netlist `dyn_setuphold.sp`, review the circuit and the design check statement.

This check reports any setup and hold time violations. This check will report any timing violation if signal net transition happens within the setup-time violation window, which is set by `setup_time` and/or hold-time violation window, which is set by `hold_time`.

```
my_dyn_sh1 dyn_setuphold node=[in] depth=0 edge=rise ref_node=out
ref_edge=rise setup_time=3n vhth=2.5 ref_vhth=2.5
```

This statement checks if the difference between the rising edges of the signal node, `node=[in]`, and the reference node, `ref_node=out`, is within the setup-time window. The rising edge of the reference node is defined as the time point when its voltage crosses `ref_vhth=2.5V`. Similarly, the rising edge of a signal node is defined as the time point when its voltage crosses `vhth=2.5V`.

```
my_dyn_sh2 dyn_setuphold node=[in] depth=0 edge=fall ref_node=out
ref_edge=fall setup_time=3n vlth=0.5 ref_vlth=0.5
```

This statement checks if the difference between the falling edges of a signal node, `node=[in]`, and the reference node, `ref_node=out`, is within the setup-time window. The falling edge of a

reference node is defined as the time point when its voltage crosses `ref_vlth=0.5V`. Similarly, the rising edge of a signal node is defined as the time point when its voltage crosses `vlth=0.5V`.

```
my_dyn_sh3 dyn_setuphold node=[out] depth=0 edge=rise ref_node=in
ref_edge=rise hold_time=3n vhth=2.5 ref_vhth=2.5
```

This statement checks if the difference between the rising edges of a signal node, `node=[out]`, and the reference node, `ref_node=in`, is within the hold-time window.

```
my_dyn_sh4 dyn_setuphold node=[out] depth=0 edge=fall ref_node=in
ref_edge=fall hold_time=3n vlth=0.5 ref_vlth=0.5
```

This statement checks if the difference between the falling edges of a signal node, `node=[out]`, and the reference node, `ref_node=in`, is within the hold-time window.

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_setuphold.sp
```

Action 3: Open the result file `dyn_setuphold.dynamic.xml` with a webbrowser.

The following two checks report setup time violations.

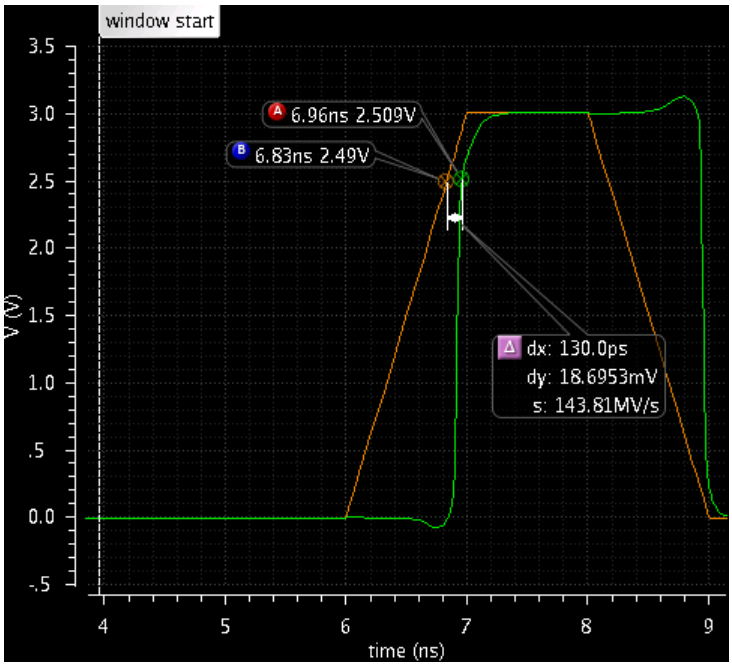
Dynamic Setup and Hold Check Violations

dyn_setuphold: my_dyn_sh1

my_dyn_sh1 dyn_setuphold node=["in"] depth=0 edge=rise ref_node="out" ref_edge=rise setup_time=3e-09 vhth=2.5 ref_vhth=2.5

Violation Count: 1

Title	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
my_dyn_sh1	in	setup	rise	rise	6.833333e-09	6.958704e-09	3.958704e-09	6.958704e-09



In this plot, the green colored line is reference net `out` and the orange colored line is signal net `in`. Marker A is for the reference net and marker B is for the signal net.

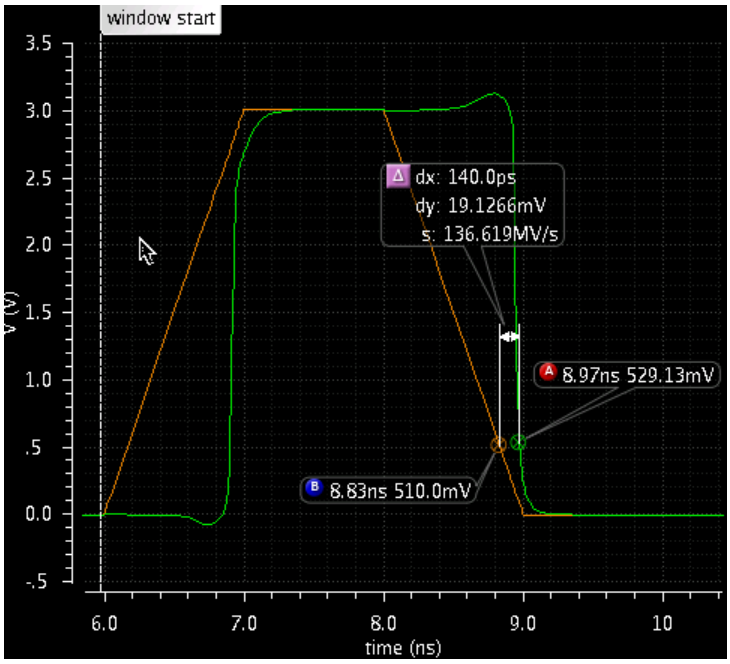
Notice the difference between the rising edges of the signal net and reference net is 130ps, which is less than the setup-time window of 3ns. In this check, the violation window starts from "*Reference Time*" minus `setup_time`, where "*Reference Time*" is 6.96ns and `setup_time` is 3ns. Therefore, violation window starts from 6.96ns as shown in above plot. Moreover, violation window ends at "*Reference Time*".

dyn_setuphold: my_dyn_sh2

my_dyn_sh2 dyn_setuphold node=["in"] depth=0 edge=fall ref_node="out" ref_edge=fall setup_time=3e-09 vlth=0.5 ref_vlth=0.5

Violation Count: 1

Title	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
my_dyn_sh2	in	setup	fall	fall	8.833333e-09	8.971853e-09	5.971853e-09	8.971853e-09



In this plot, the green colored line is reference net `out` and the orange colored line is signal net `in`. Marker A is for the reference net and marker B is for the signal net.

Notice the difference between the falling edges of the signal net and reference net is 140ps, which is less than the setup-time window of 3ns. In this check, the violation window starts from "*Reference Time*" minus `setup_time`, where "*Reference Time*" is 8.97ns and `setup_time` is 3ns. Therefore, violation window starts from 5.97ns as shown in above plot. Moreover, violation window ends at "*Reference Time*".

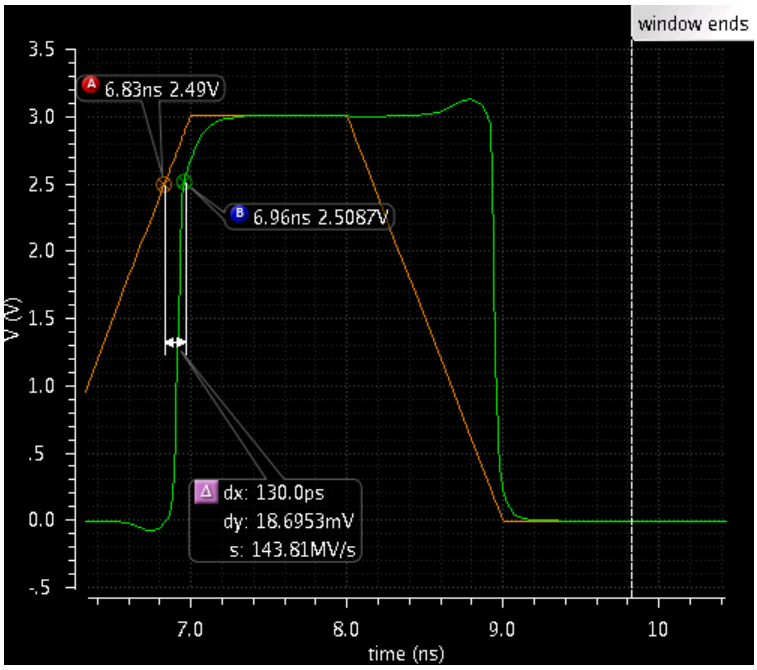
The following two checks report hold time violations.

dyn_setuphold: my_dyn_sh3

my_dyn_sh3 dyn_setuphold node=["out"] depth=0 edge=rise ref_node="in" ref_edge=rise hold_time=3e-09 vhth=2.5 ref_vhth=2.5

Violation Count: 1

Title	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
my_dyn_sh3	out	hold	rise	rise	6.958704e-09	6.833333e-09	6.833333e-09	9.833333e-09



In this plot, the orange colored line is reference net `in` and the green colored line is signal net `out`. Marker A is for the reference net and marker B is for the signal net

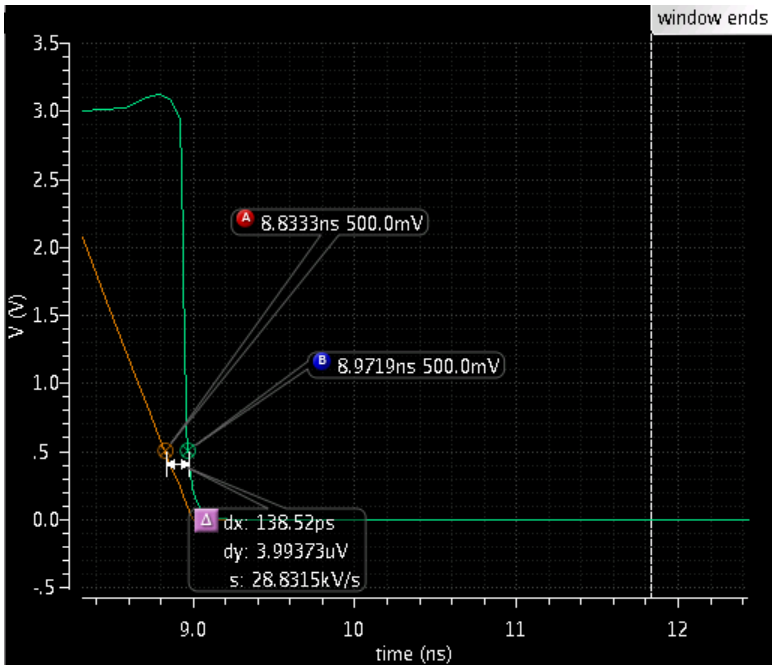
Notice the difference between the rising edges of the reference net and signal net is 130ps, which is less than hold-time window of 3ns. In this check, the violation window starts from "*Reference Time*", where "*Reference Time*" is 6.83ns. The violation window ends at "*Reference Time*" plus `hold_time` (3ns). Therefore, the violation window ends at 9.83ns as shown in plot above.

dyn_setuphold: my_dyn_sh4

my_dyn_sh4 dyn_setuphold node=["out"] depth=0 edge=fall ref_node="in" ref_edge=fall hold_time=3e-09 vlth=0.5 ref_vlth=0.5

Violation Count: 1

Title	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
my_dyn_sh4	out	hold	fall	fall	8.971853e-09	8.833333e-09	8.833333e-09	1.183333e-08



In this plot, the orange colored line is reference net `in` and the green colored line is signal net `out`. Marker A is for the reference net and marker B is for the signal net

Notice the difference between the falling edges of the signal net and reference net is 138ps, which is less than hold-time window of 3ns. In this check, the violation window starts from "*Reference Time*", where "*Reference Time*" is 8.83ns. The violation window ends at "*Reference Time*" plus `hold_time` (3ns). Therefore, the violation window ends at 11.83ns as shown in plot above.

13. Dynamic noisy node check

Action 1: Open the netlist `dyn_noisynode.sp`, review the circuit and the design check statement.

```
u1 dyn_noisynode node=[*] duration=10n e1=5e4 e2=2e16 skip=150e-12
```

The above statement reports any node that has a waveform with first derivative dv/dt greater than `e1` and second derivative d^2v/dt^2 greater than `e2`, for the duration longer than 10ns. Any stable phase less than 150ps, during the unstable duration of 10ns, will be ignored.

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_noisynode.sp
```

Action 3: Open the result file `dyn_noisynode.dynamic.xml` with a webbrowser.

In this example, the check reports two nodes with violations. Nodes 2 and 3 are noisy nodes from approximately 0s to 20ns.

Dynamic Noisy Node Check Violations

dyn_noisynode: u1

u1 dyn_noisynode node=["*"] duration=1e-08 e1=50000 e2=2e+16 skip=1.5e-10

Violation Count: 2

Title	Node Name	Start(s)	Duration(s)
u1	2	8.333333e-12	1.998745e-08
u1	3	8.333333e-12	1.998745e-08

14. Dynamic node capacitance check

Action 1: Open the netlist `dyn_nodecap.sp`, review the circuit and the design check statement.

```
n1 dyn_nodecap node=[*] time=[0 5n]
```

The above statement reports total capacitance on each node at time equal to 0 and 5ns. Device capacitances, voltage dependent capacitances, grounded and coupling caps are combined into one value. Nodes connecting to power supplies are not reported.

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_nodecap.sp
```

Action 3: Open the result file `dyn_nodecap.dynamic.xml` with a webbrowser.

In this example, this check reports two nodes with its capacitances at each time points given in parameter "`time`".

Dynamic Node Capacitance Check Violations

dyn_nodecap: n1

n1 dyn_nodecap node=["*"] time=[0 5e-09]

Violation Count: 4

Title	Node Name	Time(s)	Capacitance(F)
n1	mid	0.000000e+00	3.243946e-13
n1	out	0.000000e+00	1.038904e-13
n1	mid	5.000000e-09	2.548875e-13
n1	out	5.000000e-09	1.986409e-13

15. Dynamic subckt port power check

Action 1: Open the netlist `dyn_subcktpwr.sp`, review the circuit and the design check statements.

```
dyn1 dyn_subcktpwr inst=[*] port=[*] time_window=[0 10n] power=on
```

The above statement reports port currents, port powers through all the ports of all the subcircuit-instances. Moreover, it also report subckt powers consumed by all the subcircuit-instances.

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_subcktpwr.sp
```

Action 3: Open the result file `dyn_subcktpwr.dynamic.xml` with a webbrowser.

The port current is positive when current is going into a subcircuit. This check will report average, RMS and maximum values of the current entering all the ports, as shown below.

Dynamic Subckt Port Power Check - Port Current Report

Title	Port Name	Avg(A)	RMS(A)	Max(A)	Max Time(s)	From(s)	To(s)
dyn1	x1.in	-8.521560e-07	7.407120e-04	-5.622466e-03	5.050463e-09	0.000000e+00	1.000000e-08
dyn1	x1.out	4.849004e-08	5.236529e-04	2.528045e-03	1.167596e-09	0.000000e+00	1.000000e-08
dyn1	x1.vdd	8.643894e-05	6.313704e-04	6.318925e-03	5.100000e-09	0.000000e+00	1.000000e-08
dyn1	x1.0	-8.563527e-05	4.971732e-04	-4.127164e-03	1.100000e-09	0.000000e+00	1.000000e-08
dyn1	x2.in	-4.849004e-08	5.236529e-04	-2.528045e-03	1.167596e-09	0.000000e+00	1.000000e-08
dyn1	x2.out	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e-08
dyn1	x2.vdd	1.904729e-05	4.230470e-04	2.333418e-03	1.257721e-09	0.000000e+00	1.000000e-08
dyn1	x2.0	-1.899880e-05	2.871272e-04	-1.864157e-03	5.279976e-09	0.000000e+00	1.000000e-08

The power analysis can be done by using the parameter `power'. With `power=on` two additional sections are generated. The first section reports the average, RMS and maximum of power entering at all ports. The first section is given below.

Dynamic Subckt Port Power Check - Port Power Report

Title	Port Name	Avg(W)	RMS(W)	Max(W)	Max Time(s)	From(s)	To(s)
dyn1	x1.in	3.585677e-05	1.395275e-03	1.735319e-02	1.100000e-09	0.000000e+00	1.000000e-08
dyn1	x1.out	-1.913732e-05	1.031789e-03	8.748949e-03	1.110952e-09	0.000000e+00	1.000000e-08
dyn1	x1.vdd	2.852485e-04	2.083522e-03	2.085245e-02	5.100000e-09	0.000000e+00	1.000000e-08
dyn1	x1.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e-08
dyn1	x2.in	1.913732e-05	1.031789e-03	-8.748949e-03	1.110952e-09	0.000000e+00	1.000000e-08
dyn1	x2.out	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e-08
dyn1	x2.vdd	6.285605e-05	1.396055e-03	7.700280e-03	1.257721e-09	0.000000e+00	1.000000e-08
dyn1	x2.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e-08

The second section reports the average, RMS and maximum of power consumed by each instance of a subcircuit, which is given in the parameter `inst'. Note that for the second section it is mandatory to have the parameter `port' set to [*]. The second section is given below.

Dynamic Subckt Port Power Check - Subckt Power Report

Title	Instance Name	Avg(W)	RMS(W)	Max(W)	Max Time(s)	From(s)	To(s)
dyn1	x1	3.019680e-04	1.805895e-03	2.086922e-02	5.100000e-09	0.000000e+00	1.000000e-08
dyn1	x2	8.199337e-05	1.033609e-03	-6.409615e-03	5.104102e-09	0.000000e+00	1.000000e-08

Notice that the time window should not be overlapped. For example, for different time window use time_window=[0n 5n **5.1n** 10n]. The time_window such as [0n 5n **5n** 10n] is considered as bad practice.

The parameter `filter' can be used to filter out ports that are connected only to the gate of MOSFETs. More information about this parameter can be found in Spectre help by using

```
% spectre -h dyn_subcktpwr
```

16. Dynamic pulse width check

Action 1: Open the netlist `dyn_pulsewidth.sp`, review the circuit and the design check statements.

```
chk1 dyn_pulsewidth node=[out1] pwmin_low=25n pwmax_low=30n
pwmin_high=25n pwmax_high=30n vlth=0.2 vhth=1.0
```

The above command will check the node `out1` and reports if either of the following conditions satisfies:

- The pulse-width in logic-low state is outside the range of parameters pwmin_low (25n) and pwmax_low (30n)
- The pulse-width in logic-high state is outside the range of parameters pwmin_high (25n) and pwmax_high (30n).

Action 2: Run the netlist with spectre

```
% spectre +spice dyn_pulsewidth.sp
```

Action 3: Open the result file `dyn_pulsewidth.dynamic.xml` with a webbrowser.

Dynamic Pulse Width Check Violations

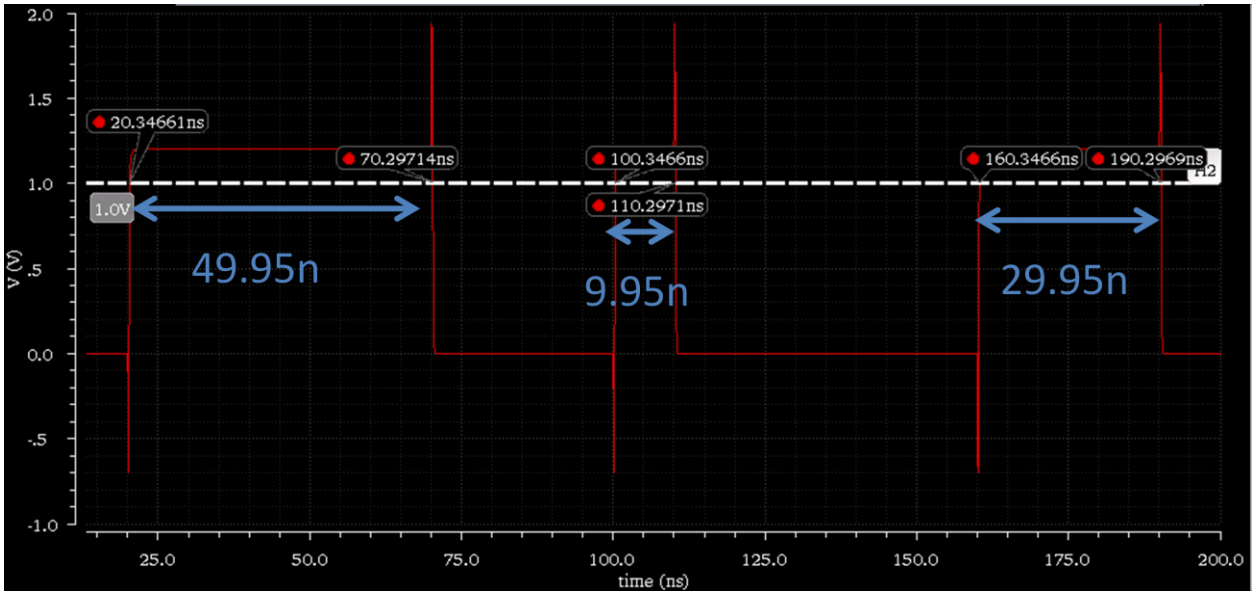
dyn_pulsewidth: chk1

chk1 dyn_pulsewidth node=["out1"] pwmin_low=2.5e-08 pwmax_low=3e-08 pwmin_high=2.5e-08 pwmax_high=3e-08 vlth=0.2 vhth=1

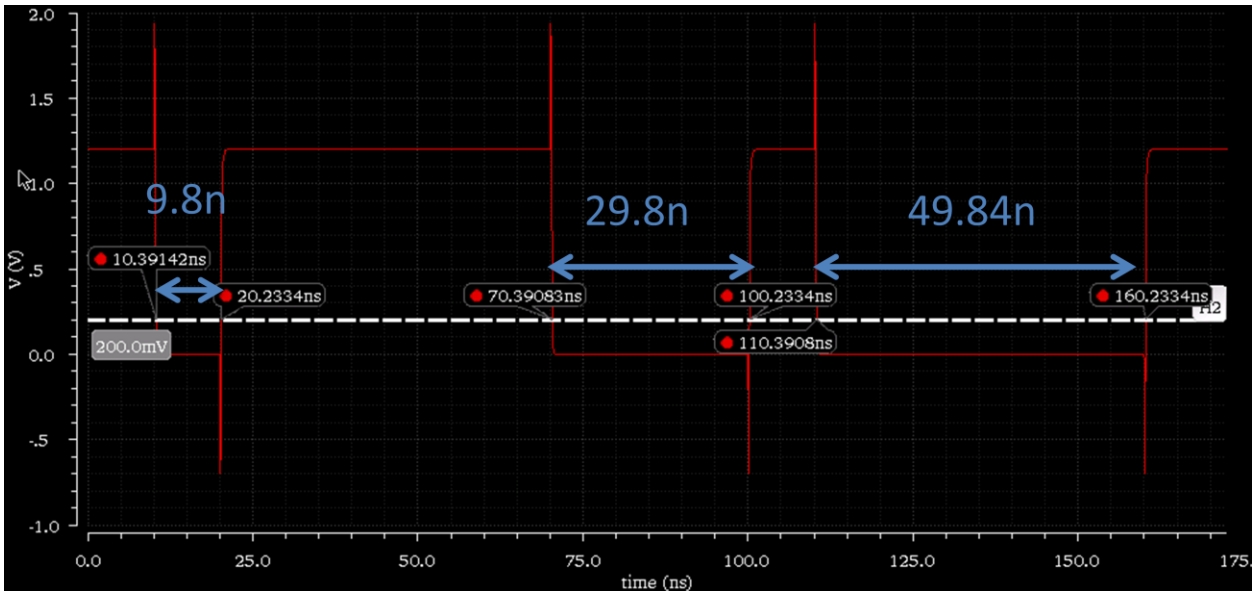
Violation Count: 4

Title	Node Name	Type	Time(s)	Pulse Width(s)
chk1	out1	high	2.034661e-08	4.995054e-08
chk1	out1	high	1.003466e-07	9.950533e-09
chk1	out1	low	1.039142e-08	9.841979e-09
chk1	out1	low	1.103908e-07	4.984257e-08

The result file reports that the pulsewidth in *high-state* has two violations. The source of two violations is shown in the following figure.



Similarly, the result file reports that the pulsewidth in *low-state* has two violations. The source of two violations is shown in the following figure.



17. Dynamic subckt instance activity check

Action 1: Open the netlist `dyn_activity.sp`, review the circuit and the design check statement.

```
chk1 dyn_activity inst=[x1 x2] time_window=[0n 20n] min_activity=0
```

The above command will check the activity percentage of instances `X1` and `X2`. The activity percentage is a ratio of number of events happening in an instance to number of events happening in whole circuit. The events start recording between the specified time window of 0ns and 20ns.

Action 2: Run the netlist with spectre

```
% spectre +spice +xps +cktpreset=sram dyn_activity.sp
```

Action 3: Open the result file `dyn_activity.dynamic.xml` with a webbrowser.

Dynamic Subckt Activity Check Violations

dyn_activity: chk1

chk1 dyn_activity inst=["x1" "x2"] time_window=[0 2e-08] min_activity=0

Violation Count: 10

Title	Instance Name	Activity	Subckt	From(s)	To(s)
chk1	x1	11.358%	logic_gates	0.000000e+00	2.000000e-08
chk1	x1.x_d1	3.580%	nand	0.000000e+00	2.000000e-08
chk1	x1.x_d2	3.126%	nand	0.000000e+00	2.000000e-08
chk1	x1.x_r1	3.669%	nor	0.000000e+00	2.000000e-08
chk1	x1.x_t2	0.983%	inv	0.000000e+00	2.000000e-08
chk1	x2	88.642%	logic_gates	0.000000e+00	2.000000e-08
chk1	x2.x_d1	31.813%	nand	0.000000e+00	2.000000e-08
chk1	x2.x_d2	20.177%	nand	0.000000e+00	2.000000e-08
chk1	x2.x_r1	33.956%	nor	0.000000e+00	2.000000e-08
chk1	x2.x_t2	2.696%	inv	0.000000e+00	2.000000e-08

The result file reports that there are 10 instances with activity above 0%. The most active instance, `X2`, has an activity percentage of 88.6%. Notice that since there are only two instances in a circuit, the sum of `X1` and `X2` is 100%.

Notice that the time window should not be overlapped. For example, for different time window use time_window=[0n 5n **5.1n** 10n]. The time_window such as [0n 5n **5n** 10n] is considered as bad practice.

This check is only supported in XPS.

18. Dynamic active node check

Action 1: Open the netlist `dyn_actnode.sp`, review the circuit and the design check statements.

```
chk2 dyn_actnode node=[*] dv=3.2 type=act time_window=[0 300n]
```

The above command will check all nodes and reports only active nodes between time window of 0ns and 300ns. A node is considered active if its peak-to-peak voltage between a given time window is larger than `3.2V`.

Action 2: Run the netlist with spectre

```
% spectre +spice +xps +cktpreset=sram dyn_actnode.sp
```

Action 3: Open the result file `dyn_actnode.dynamic.xml` with a webbrowser.

Dynamic Active Node Check Violations

dyn_actnode: chk1

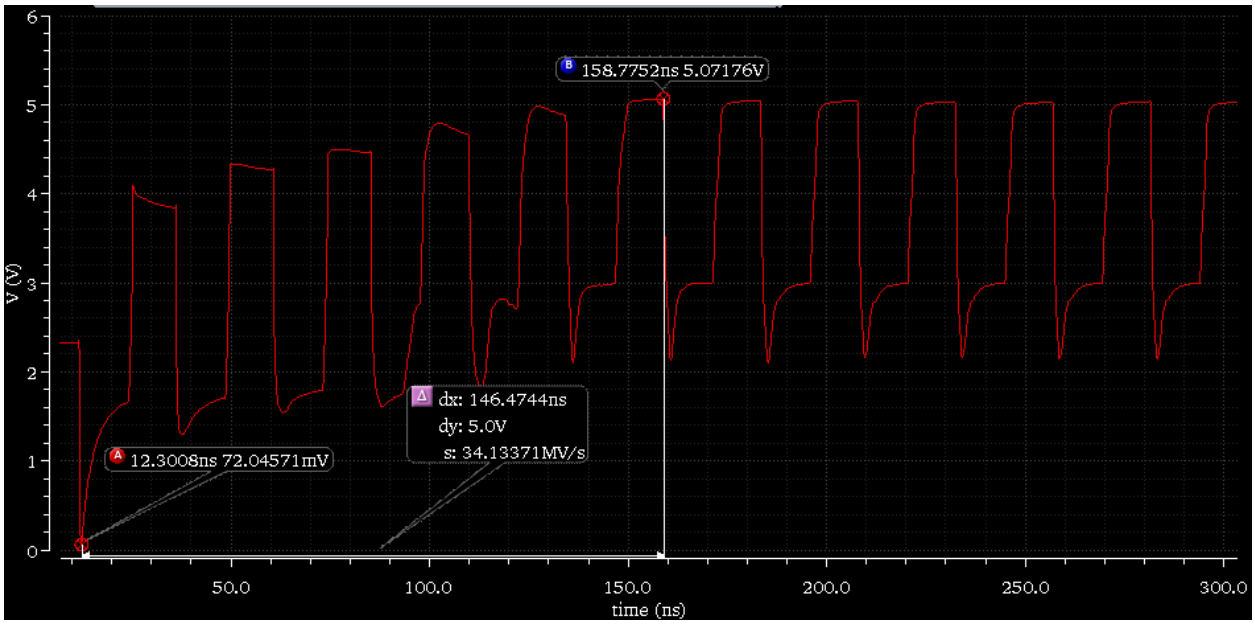
chk1 dyn_actnode node=["*"] dv=3.2 type=act time_window=[0 3e-07]

Violation Count: 14

Dynamic Active Node Check - Active Nodes

Title	Node Name	Vpp(V)	From(s)	To(s)
chk1	X5.NA	5.000248e+00	0.000000e+00	3.000000e-07
chk1	X5.N2N1485	4.649535e+00	0.000000e+00	3.000000e-07
chk1	X5.ND	4.526830e+00	0.000000e+00	3.000000e-07
chk1	X5.N2N1484	4.045392e+00	0.000000e+00	3.000000e-07
chk1	X5.NE	4.002830e+00	0.000000e+00	3.000000e-07

The result file reports that there are 14 active nodes. The most active node, `X5.NA`, has a peak-to-peak voltage of 5V. The plot of this active node is shown below. Notice that the peak-to-peak voltage is 5V between 0ns and 300ns.



Notice that the time window should not be overlapped. For example, for different time window use time_window=[0n 5n **5.1n** 10n]. The time_window such as [0n 5n **5n** 10n] is considered as bad practice. This check is only supported in XPS.

Appendix: Floating node detection

The floating (or highz) node detection algorithm is based on state of individual devices, which could be either ON or OFF. Such state is defined by a Boolean expression called conducting rule, which could be either TRUE or FALSE.

Consider the following conducting rules:

1. A MOSFET is in ON state if “ids > MOS_ITH or gds > MOS_GDS” is TRUE, where ids and gds are the current and trans-conductance, respectively, of a MOSFET. The MOS_ITH and MOS_GDS are the parameters given by a user. Notice that this word “state” is NOT referring to triode, saturation or cut-off states of a MOSFET.
2. A BJT is in ON state if “ic > BJT_ITH or vbe > BJT_VBE” is TRUE, where ic and vbe are the collector-current and voltage difference between base and emitter, respectively, of a BJT. The BJT_ITH and BJT_VBE are the parameters given by a user.
3. For more conducting rules see help by typing %spectre –h dyn_highz

A single node can have many routes to VDD or GND. The algorithm will check all possible routes for each node to VDD or GND. If any route contains devices in ON state, then that node has a path to VDD or GND. However, if a node does not a have any path to VDD or GND, then it’s considered as floating.

User can set MOS_ITH, MOS_GDS, BJT_ITH and BJT_VBE parameters in the dyn_highz and dyn_floatdcpath checks.