

S-Edit v13

Examples Guide

Tanner EDA Division
Tanner Research, Inc.
825 South Myrtle Avenue
Monrovia, CA 91016-3424
Tel: (626) 471-9700

TABLE OF CONTENTS

1.	TRAFFIC LIGHT CONTROLLER — LIGHTS	4
2.	BUSES AND ARRAYS.....	5
2.1.	SIMPLE BUSES	5
2.2.	SPLITTING BUSES	7
2.3.	PORTBUNDLES	9
2.4.	1-DIMENSIONAL ARRAYS.....	11
2.5.	2-DIMENSIONAL ARRAYS.....	12
3.	SPICE EXPORT	14
3.1.	SPICE PRIMITIVES	14
3.2.	PASSING PARAMETERS DOWN HIERACHY	18
3.3.	SUBCIRCUITS	20
3.4.	SPICE EXPORT CONTROL PROPERTY	23
4.	GLOBAL NETS, GLOBAL SYMBOLS, AND NETCAPS	25
4.1.	SIMPLE GLOBAL NETS	25
4.2.	SEPARATE POWER SUPPLIES	27
4.3.	RENAMING SEPARATE POWER SUPPLIES.....	29
4.4.	GLOBALS4	31
5.	MULTIPLE VIEWS.....	33

5.1.	MOSFET WITH 4 AND 3-TERMINAL SYMBOLS	33
5.2.	NMOS WITH IEEE AND IEC SYMBOLS	35
5.3.	ADDER WITH 2 DIFFERENT SYMBOLS	36
6.	TCL/TK SCRIPTS	37
6.1.	MODIFYING TEXT	37
6.2.	RESIZING TEXT WITH TK DIALOG TO ENTER PARAMETERS.....	39

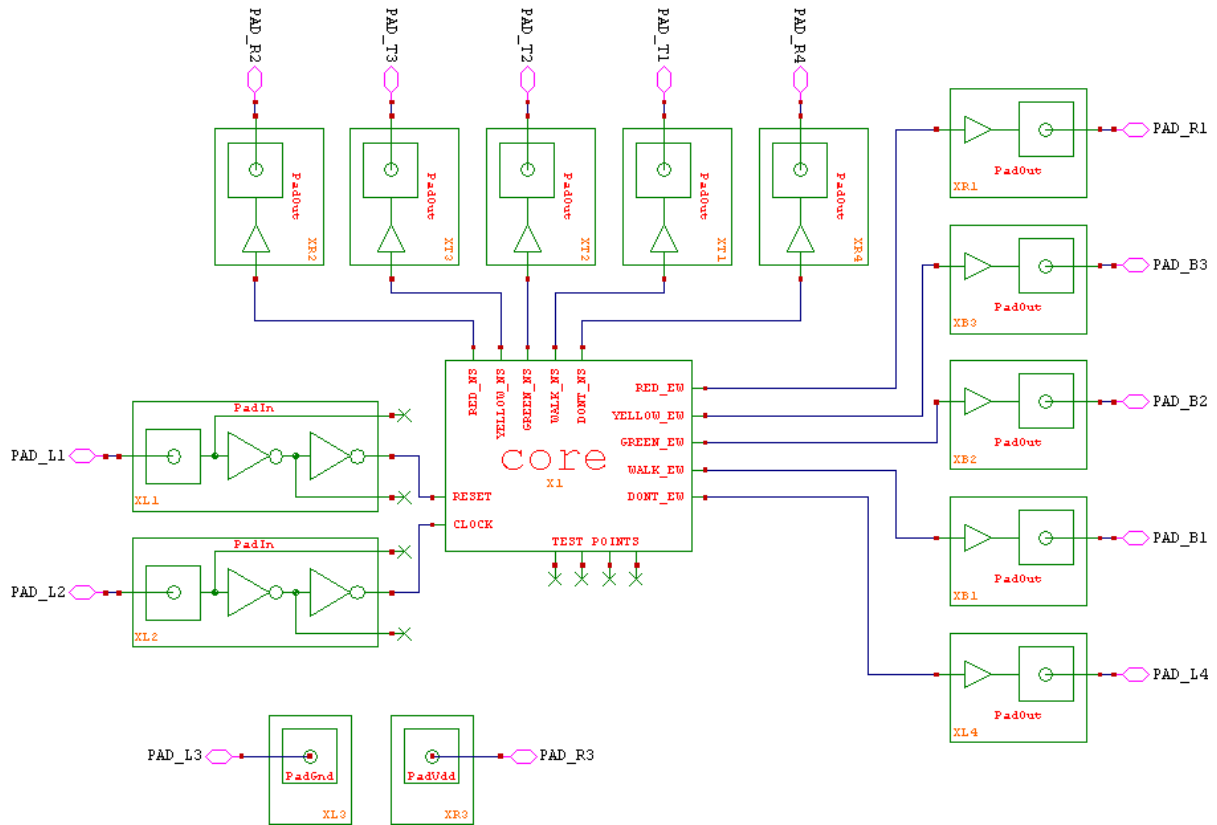
1. Traffic Light Controller — Lights

Design \L-Edit and LVS\SPR\Lights\Lights.tanner

Cell Lights

This example shows the organization of a project into libraries. Here Lights is the main design, IO_Pads, LogicGates, and Misc are libraries of Lights, and Devices is a library used by IO_Pads, LogicGates, and Misc.

Schematic



Schematic of cell Lights

2. Buses and Arrays

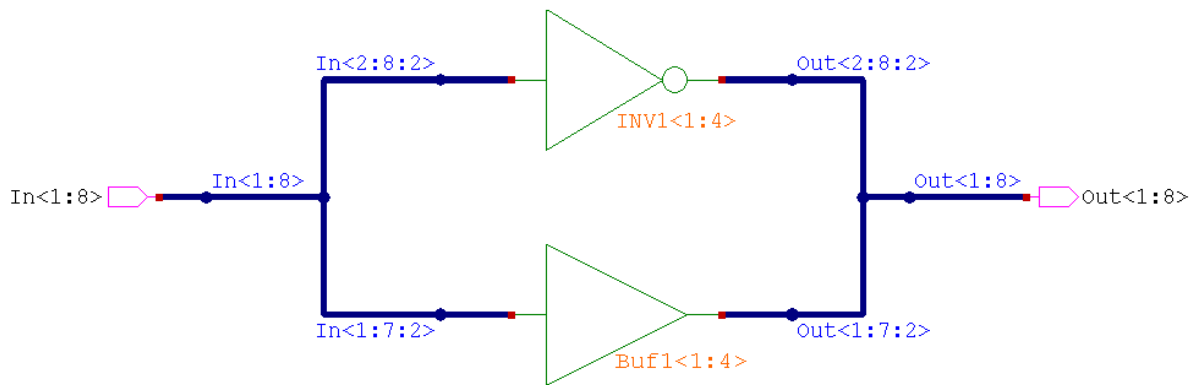
2.1. Simple Buses

Design \S-Edit\Examples\BusesAndArrays\BusesAndArrays.tanner

Cell Example1

This example illustrates the basic syntax and usage of buses and arrays. An 8-bit wide bus, `In<1:8>`, is split into two buses, one containing the even numbered bits and the other containing the odd numbered bits. The third value in the bus specification, indicating a step value of 2, is used perform this split. The even numbered bits connect to a 4x array of inverters, and the odd numbered bits connect to a 4x array of buffers. The inverter and the buffer each have a single input and output connection, so the 4x arrays of each of these provides a 4-bit wide input and output connection to match the dimension of the buses that connect. When connecting buses to instances or arrays of instances, it is important to make sure that the dimensions match. Invoking **Tools > Design Checks** will issue warnings for mismatched bus and instance dimensions. The output of the inverters and the output of the buffers are then combined to form an 8-bit wide output bus, `Out<1:8>`.

Schematic



SPICE Netlist

```
XINV1<1> In<2> Out<2> Gnd Vdd INV
XINV1<2> In<4> Out<4> Gnd Vdd INV
XINV1<3> In<6> Out<6> Gnd Vdd INV
XINV1<4> In<8> Out<8> Gnd Vdd INV
XBuf1<1> In<1> Out<1> Gnd Vdd Buf1
XBuf1<2> In<3> Out<3> Gnd Vdd Buf1
XBuf1<3> In<5> Out<5> Gnd Vdd Buf1
XBuf1<4> In<7> Out<7> Gnd Vdd Buf1
.end
```

Error Indications

If there is a dimension mismatch between nets an error indication will be output in the command window upon performing a Design Check. For example, if the Out<1:8> port is changed to Out<1:9> the following error will be output:

```
# EXT Error: In schematic of cell "Example1" illegal connection between nets "Out<1:9>" and "Out<1:8>" found (dimension mismatch).  
# CHK Warning: Cell: Example1, Net: "N_1" in schematic: is not connected to any port.  
# SED Error: Design check complete. Design: BusesAndArrays and 0 libraries, 5 cells and 10 views have been checked. 1 error and 1 warning were found.
```

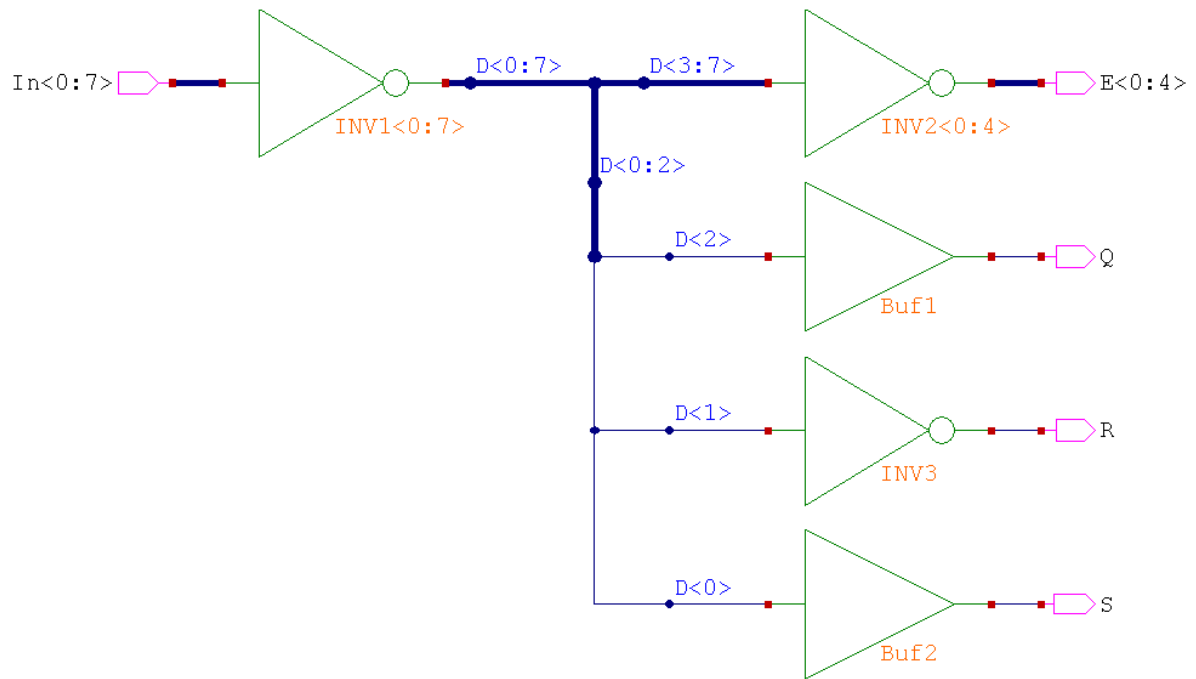
2.2. Splitting Buses

Design \S-Edit\Examples\BusesAndArrays\BusesAndArrays.tanner

Cell Example2

This example illustrates labeling requirements when splitting buses. An 8-bit wide bus, $In<0:7>$, is input to an 8x array of inverters, and an 8-bit wide bus, $D<0:7>$, is output. The 8-bit bus $D<0:7>$ is then split into a 5-bit wide bus, $D<3:7>$, and a 3-bit wide bus, $D<0:2>$. Note that whenever there is a T-junction of buses, all branches of the “T” must be explicitly labeled in order to unambiguously identify the dimension and components of each branch. Individual bits $D<2>$, $D<1>$, and $D<0>$ are then ripped from the bus and connected to a buffer, inverter, and another buffer, and output as nets Q , R , and S .

Schematic



SPICE Netlist

```
XINV1<0> In<0> D<0> Gnd Vdd INV
XINV1<1> In<1> D<1> Gnd Vdd INV
XINV1<2> In<2> D<2> Gnd Vdd INV
XINV1<3> In<3> D<3> Gnd Vdd INV
XINV1<4> In<4> D<4> Gnd Vdd INV
XINV1<5> In<5> D<5> Gnd Vdd INV
XINV1<6> In<6> D<6> Gnd Vdd INV
XINV1<7> In<7> D<7> Gnd Vdd INV
```

```
XINV2<0> D<3> E<0> Gnd Vdd INV
XINV2<1> D<4> E<1> Gnd Vdd INV
XINV2<2> D<5> E<2> Gnd Vdd INV
XINV2<3> D<6> E<3> Gnd Vdd INV
XINV2<4> D<7> E<4> Gnd Vdd INV
```

```
XBuf1 D<2> Q Gnd Vdd Buf1
XINV3 D<1> R Gnd Vdd INV
XBuf2 D<0> S Gnd Vdd Buf1
.end
```

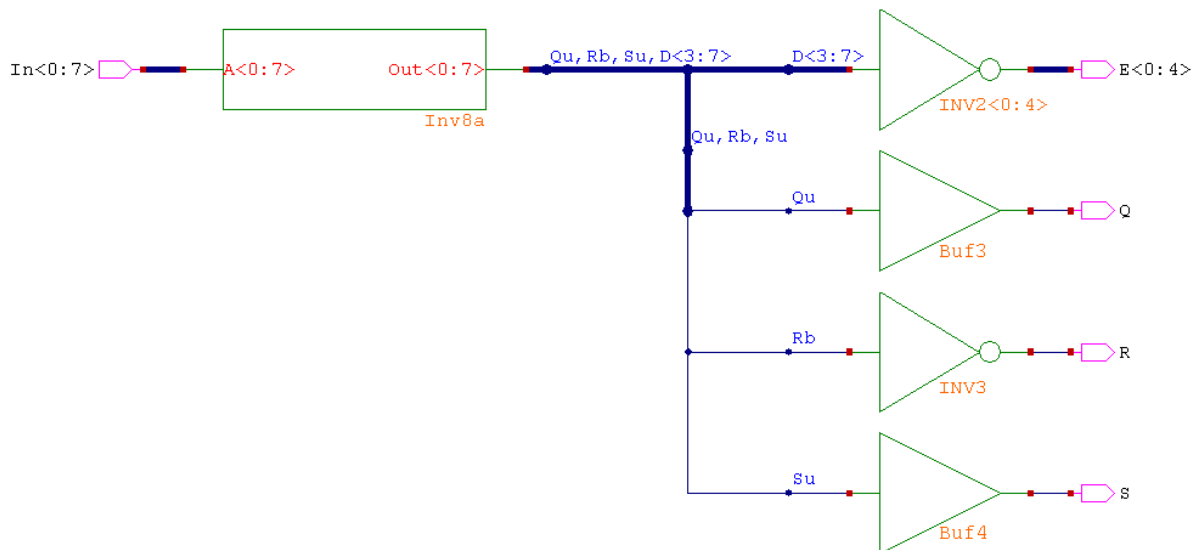

2.3. Port Bundles

Design \S-Edit\Examples\BusesAndArrays\BusesAndArrays.tanner

Cell Example3

This example illustrates the use of port bundles in a symbol. Example3 is similar to Example 2, however here the 8-bit input bus, In<0:7>, is connected to a single instance, Inv8a, rather than to an array. The symbol of Inv8_1 contains an 8-bit port bundle, A<0:7>, to which the input bus is connected, thereby matching dimensions of the bus with the instance connection. The port bundle can be a single bus, A<0:7>, as is the case in this example, or it could be a collection of buses and nets, such as A<0:4:2>, B<0:3>, C. The output of the instance is an 8-bit port bundle, Out<0:7> which connects to an 8-bit wide bus, Qu, Rb, Su, D<3:7>. The 8-bit bus Qu, Rb, Su, D<3:7> is then split into a 5-bit wide bus, D<3:7>, and a 3-bit wide bus, Qu, Rb, Su. Individual bits Qu, Rb, and Su are then ripped from the bus and connected to a buffer, inverter, and another buffer, and output as nets Q, R, and S.

Schematic



SPICE Netlist

```
***** Subcircuits *****
.subckt Inv8 A<0> A<1> A<2> A<3> A<4> A<5> A<6> A<7> Out<0> Out<1> Out<2>
Out<3> Out<4> Out<5> Out<6> Out<7> Gnd Vdd
XBuf1 A<0> Out<0> Gnd Vdd Buf1
XINV1 A<1> Out<1> Gnd Vdd INV
XBuf2 A<2> Out<2> Gnd Vdd Buf1
XINV2 A<3> Out<3> Gnd Vdd INV
XBuf3 A<4> Out<4> Gnd Vdd Buf1
XINV3 A<5> Out<5> Gnd Vdd INV
XBuf4 A<6> Out<6> Gnd Vdd Buf1
XINV4 A<7> Out<7> Gnd Vdd INV
.ends
```

```
***** Simulation Settings - Parameters and SPICE Options *****
```

```

XInv8a In<0> In<1> In<2> In<3> In<4> In<5> In<6> In<7> Qu Rb Su D<3> D<4>
D<5> D<6> D<7> Gnd Vdd Inv8
XINV2<0> D<3> E<0> Gnd Vdd INV
XINV2<1> D<4> E<1> Gnd Vdd INV
XINV2<2> D<5> E<2> Gnd Vdd INV
XINV2<3> D<6> E<3> Gnd Vdd INV
XINV2<4> D<7> E<4> Gnd Vdd INV
XBuf3 Qu Q Gnd Vdd Buf1
XINV3 Rb R Gnd Vdd INV
XBuf4 Su S Gnd Vdd Buf1
.end

```

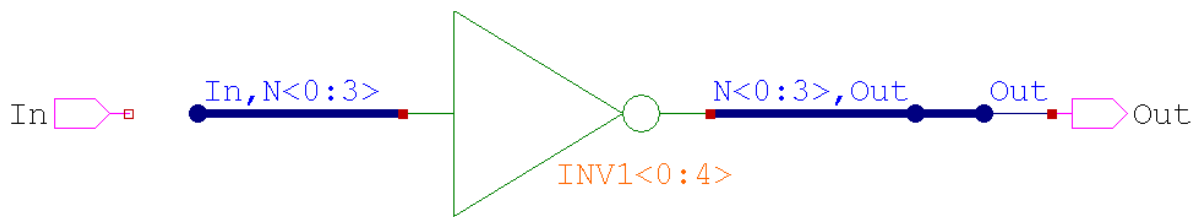
2.4. 1-Dimensional Arrays

Design \S-Edit\Examples\BusesAndArrays\BusesAndArrays.tanner

Cell Example4

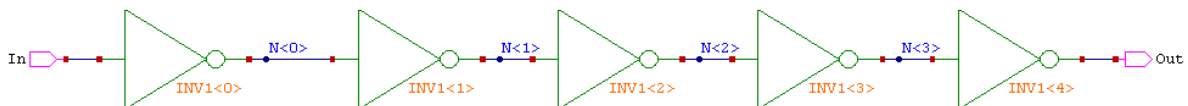
This example illustrates how to connect the input and output of an array to form a connection in series. The input to the 5x array of inverters is In, N<0>, N<1>, N<2>, N<3>, and the output is N<0>, N<1>, N<2>, N<3>, Out. Notice the offset by one in the position of N<0:3> in the naming of the input and output buses. This causes the output of one inverter to be connected to the input of the next inverter. The connection is formed by naming the output and input labels with the same name. There does not need to be a wire actually making a connection. In addition, as can be seen for the input, no physical wire connection is made between the In port and the bus. For the output, a wire connection is made and the net is labeled Out to match that of the Out port. Either method will produce the same result.

Schematic



Schematic_Flat

The above array is equivalent to the following schematic:



SPICE Netlist

```
XINV1<0> In N<0> Gnd Vdd INV
XINV1<1> N<0> N<1> Gnd Vdd INV
XINV1<2> N<1> N<2> Gnd Vdd INV
XINV1<3> N<2> N<3> Gnd Vdd INV
XINV1<4> N<3> Out Gnd Vdd INV
.end
```

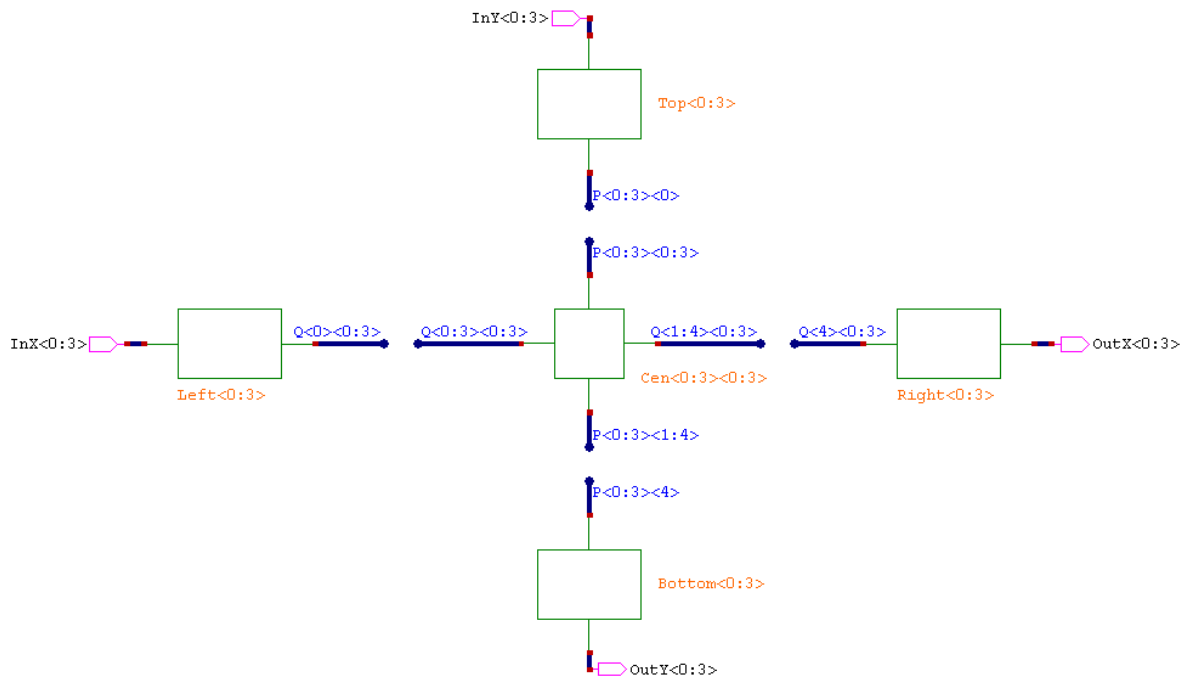
2.5. 2-Dimensional Arrays

Design \S-Edit\Examples\BusesAndArrays\BusesAndArrays.tanner

Cell Example5

This example illustrates the usage and syntax of two dimensional arrays. Arrays Left, Top, Bottom, and Right are 1-D arrays which are connected to around the perimeter of a 2-D array Cen using connection by name, similar to Example 4. Internal nets are color coded to aid in showing the structure and connections of the array.

Schematic



SPICE Netlist

```
XLeft<0> InX<0> Q<0><0> Left
XLeft<1> InX<1> Q<0><1> Left
XLeft<2> InX<2> Q<0><2> Left
XLeft<3> InX<3> Q<0><3> Left
```

```
XTop<0> InY<0> P<0><0> Top
XTop<1> InY<1> P<1><0> Top
XTop<2> InY<2> P<2><0> Top
XTop<3> InY<3> P<3><0> Top
```

```
XCen<0><0> Q<0><0> P<0><0> Q<1><0> P<0><1> A
XCen<0><1> Q<0><1> P<0><1> Q<1><1> P<0><2> A
XCen<0><2> Q<0><2> P<0><2> Q<1><2> P<0><3> A
XCen<0><3> Q<0><3> P<0><3> Q<1><3> P<0><4> A
```

```
XCen<1><0> Q<1><0> P<1><0> Q<2><0> P<1><1> A
XCen<1><1> Q<1><1> P<1><1> Q<2><1> P<1><2> A
```

```

XCen<1><2> Q<1><2> P<1><2> Q<2><2> P<1><3> A
XCen<1><3> Q<1><3> P<1><3> Q<2><3> P<1><4> A

XCen<2><0> Q<2><0> P<2><0> Q<3><0> P<2><1> A
XCen<2><1> Q<2><1> P<2><1> Q<3><1> P<2><2> A
XCen<2><2> Q<2><2> P<2><2> Q<3><2> P<2><3> A
XCen<2><3> Q<2><3> P<2><3> Q<3><3> P<2><4> A

XCen<3><0> Q<3><0> P<3><0> Q<4><0> P<3><1> A
XCen<3><1> Q<3><1> P<3><1> Q<4><1> P<3><2> A
XCen<3><2> Q<3><2> P<3><2> Q<4><2> P<3><3> A
XCen<3><3> Q<3><3> P<3><3> Q<4><3> P<3><4> A

XRight<0> Q<4><0> OutX<0> Right
XRight<1> Q<4><1> OutX<1> Right
XRight<2> Q<4><2> OutX<2> Right
XRight<3> Q<4><3> OutX<3> Right

XBottom<0> P<0><4> OutY<0> Bottom
XBottom<1> P<1><4> OutY<1> Bottom
XBottom<2> P<2><4> OutY<2> Bottom
XBottom<3> P<3><4> OutY<3> Bottom

.end

```

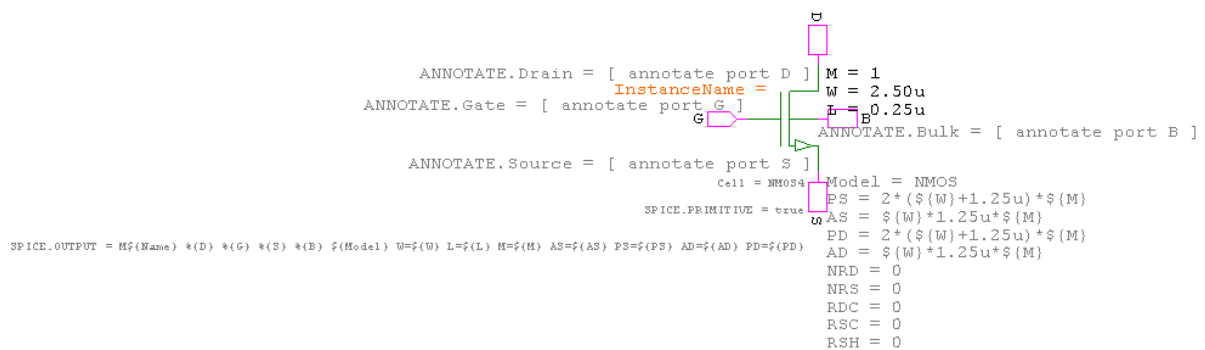
3. SPICE Export

3.1. SPICE primitives

Design \S-Edit\Examples\SpiceOutput\SpiceOutput.tanner

Cells NMOS4, PMOS4, INV

This example illustrates the use of the SPICE.OUTPUT property to output SPICE for a primitive device. A primitive device is the lowest level device, for which there is no schematic, and the output to SPICE is determined by the SPICE.OUTPUT property on the symbol. The symbol of cell NMOS4, an NMOS transistor is shown below:



Symbol of NMOS4 transistor, view NMOS4

The symbol has several properties:

```
M = 1
W = 2.5u
L = 0.25u
Model = NMOS
PS = 2*($ {W}+1.25u)*$ {M}
AS = $ {W}*1.25u*$ {M}
PD = 2*($ {W}+1.25u)*$ {M}
AD = $ {W}*1.25u*$ {M}
NRD = 0
NRS = 0
RDC = 0
RSC = 0
RSH = 0
```

A SPICE.OUTPUT property on the symbol specifies the SPICE call written for each instance of the symbol, and a SPICE.PRIMITIVE = True property on the symbol indicates that the device is a primitive. The SPICE.OUTPUT and SPICE.PRIMITIVE properties are as follows:

```
SPICE.OUTPUT = M$ {Name} *(D) *(G) *(S) *(B) $ {Model} W=$ {W} L=$ {L}
M=$ {M} AS=$ {AS} PS=$ {PS} AD=$ {AD} PD=$ {PD}
SPICE.PRIMITIVE = True
```

The SPICE Export Control Property to enter when exporting SPICE is the name of the property that contains the subproperties OUTPUT and PRIMITIVE. In this case, the word **SPICE** is the Export Control property.

Inspecting the SPICE.OUTPUT statement in detail, each element of the property is written out as follows:

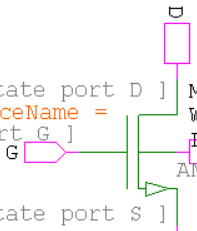
M	Write “M” literally
$\$ \{ \text{Name} \}$	Write the name of the instance
$\% \{ D \}, \% \{ G \}, \% \{ S \}, \% \{ B \}$	Write the net names connected to pins D, G, S, and B
$\$ \{ \text{Model} \}$	Write the value of the property “Model” on this instance
W=	Write “W=” literally
$\$ \{ W \}$	Write the value of the property “W” on this instance
L=	Write “L=” literally
$\$ \{ L \}$	Write the value of the property “L” on this instance
M=	Write “M=” literally
$\$ \{ M \}$	Write the value of the property “M” on this instance
AS=	Write “AS=” literally
$\$ \{ AS \}$	Write the value of the property “AS” on this instance
PS=	Write “PS=” literally
$\$ \{ PS \}$	Write the value of the property “PS” on this instance
AD=	Write “AD=” literally
$\$ \{ AD \}$	Write the value of the property “AD” on this instance
PD=	Write “PD=” literally
$\$ \{ PD \}$	Write the value of the property “PD” on this instance

The symbol of cell PMOS4 has similar properties and a similar SPICE.OUTPUT property as cell NMOS4. Cell INV makes use of cells NMOS4 and PMOS4. The SPICE output for the schematic of INV is as follows:

```
MP1 Out A Vdd Vdd PMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p
PS=7.5u AD=3.125p PD=7.5u
MN1 Out A Gnd Gnd NMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p
PS=7.5u AD=3.125p PD=7.5u
.end
```

We can see the substitutions of the instance name, net names, and property values in each SPICE call line, according to the table above.

An alternate method that may be used instead of defining one SPICE.OUTPUT property to specify the SPICE call is to define the SPICE.PREFIX, SPICE.PINORDER, SPICE.MODEL, and SPICE.PARAMETERS properties. These four properties when used in conjunction with each other will also specify the SPICE call written for each instance of the symbol. An example of this is show in symbol NMOS4, view NMOS4_Expand.



```

ANNOTATE.Drain = [ annotate port D ] M = 1
                                W = 2.50u
ANNOTATE.Gate = [ annotate port G ] L = 0.25u
                                ANNOTATE.Bulk = [ annotate port B ]

ANNOTATE.Source = [ annotate port S ]
                                Cell = NMOS4
                                Model = NMOS
                                SPICE.PRIMITIVE = true
                                PS = 2*({W}+1.25u)*${M}
                                AS = ${W}*1.25u*${M}
                                SPICE.PREFIX = M
                                PD = 2*({W}+1.25u)*${M}
                                AD = ${W}*1.25u*${M}
                                SPICE.PINORDER = D G S B
                                NRD = 0
                                SPICE.MODEL = $Model
                                NRS = 0
                                RDC = 0
                                SPICE.PARAMETERS = W= L= M~ AS= PS= AD= PD= NRD~ NRS~ RDC~ RSC~ RSH~
                                RSC = 0
                                RSH = 0

```

Symbol of NMOS4 transistor, view NMOS4_Expand

The SPICE.PREFIX, SPICE.PINORDER, SPICE.MODEL, and SPICE.PARAMETERS properties are as follows:

```

SPICE.PREFIX = M
SPICE.PINORDER = D G S B
SPICE.MODEL = $Model
SPICE.PARAMETERS = W= L= M~ AS= PS= AD= PD= NRD~ NRS~ RDC~ RSC~ RSH~

```

Inspecting the SPICE.PREFIX statement, the property is written out as follows:

M Write “M” literally followed by the name of the instance

The output of SPICE.PREFIX will be followed by the SPICE.PINORDER statement, the SPICE.PINORDER property is written out as follows:

D G S B Write the net names connected to pins D, G, S, and B

The output of SPICE.PINORDER will be followed by the SPICE.MODEL statement, the SPICE.MODEL property is written out as follows:

\$Model Write the value of the property “Model” on this instance

The output of SPICE.MODEL will be followed by the SPICE.PARAMETERS statement, the .PARAMETERS property is written out as follows:

W= Write “W=” literally followed by the value of the property “W” on this instance

L= Write “L=” literally followed by the value of the property “L” on this instance

M~ Write “M=” literally followed by the value of the property “M” on this instance only if the value of M differs from its default value

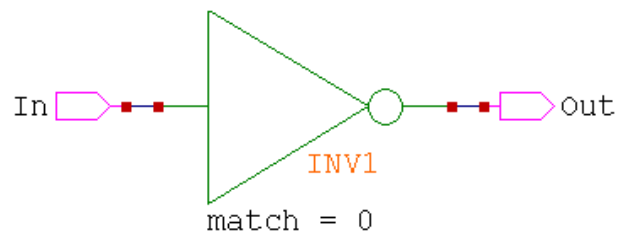
AS=	Write “AS=” literally followed by the value of the property “AS” on this instance
PS=	Write “PS=” literally followed by the value of the property “PS” on this instance
AD=	Write “AD=” literally followed by the value of the property “AD” on this instance
PD=	Write “PD=” literally followed by the value of the property “PD” on this instance
NRD~	Write “NRD=” literally followed by the value of the property “NRD” on this instance only if the value of NRD differs from its default value
NRS~	Write “NRS=” literally followed by the value of the property “NRS” on this instance only if the value of NRS differs from its default value
RDC~	Write “RDC=” literally followed by the value of the property “RDC” on this instance only if the value of RDC differs from its default value
RSC~	Write “RSC=” literally followed by the value of the property “RSC” on this instance only if the value of RSC differs from its default value
RSH~	Write “RSH=” literally followed by the value of the property “RSH” on this instance only if the value of RSH differs from its default value

3.2. Passing parameters down hierarchy

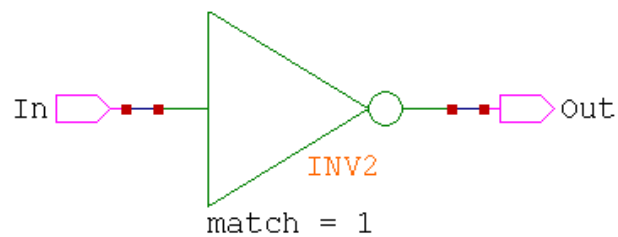
Design \S-Edit\Examples\SpiceOutput\SpiceOutput.tanner

Cell Inverters

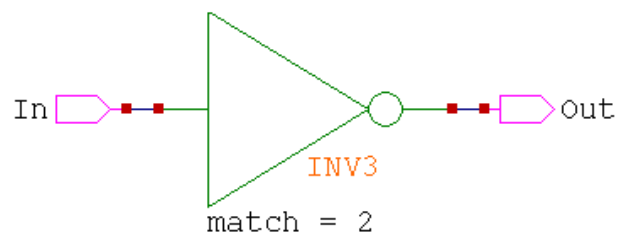
This example illustrates how parameters can be passed down the hierarchy and written to SPICE. Cell Inverters contains three instances of cell INV. The symbol for INV contains a property “match=0” The first instance has no local override of match, the second instance has a local override “match=1” and the third instance has a local override “match=2”



match = 0 is default value on symbol of INV



match = 1 is local override on instance INV2



match = 2 is local override on instance INV3

Schematic of cell Inverters

SPICE Netlist

```
.subckt INV A Out Gnd Vdd match=0
MP1 Out A Vdd Vdd PMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
MN1 Out A Gnd Gnd NMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
.ends

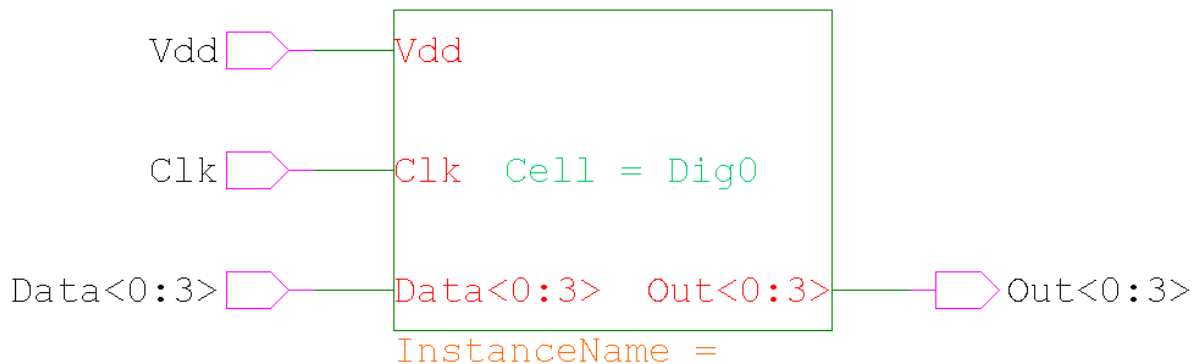
XINV1 In Out Gnd Vdd INV match=0
XINV2 In Out Gnd Vdd INV match=1
XINV3 In Out Gnd Vdd INV match=2
```

3.3. Subcircuits

Design \S-Edit\Examples\SpiceOutput\SpiceOutput.tanner

Cells Dig0, Dig1, Dig2, Subcircuits

This example illustrates how to use the SPICE.OUTPUT and SPICE.DEFINITION properties to control the SPICE output written for a subcircuit. A subcircuit is a symbol that is not a primitive. The symbol for cell Dig0 has no SPICE.OUTPUT or SPICE.DEFINITION properties. The symbol is shown below



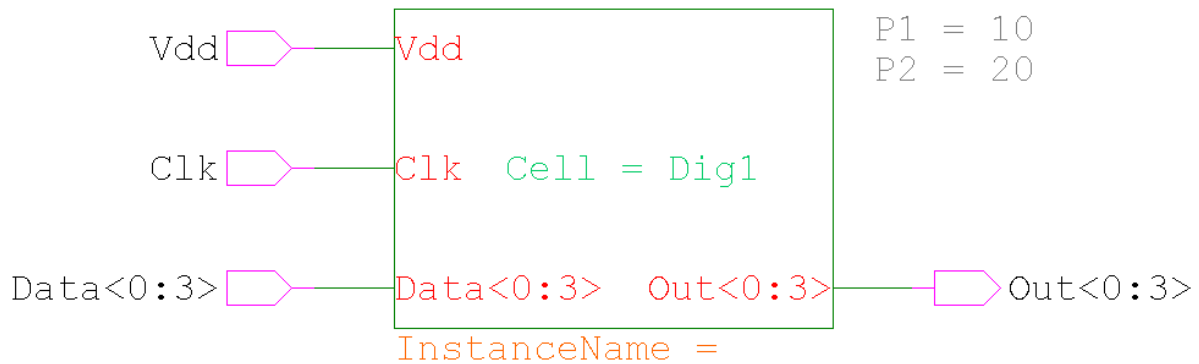
Dig0 Symbol with no SPICE.OUTPUT or SPICE.DEFINITION

When no SPICE.DEFINITION property is present, the subcircuit definition will contain all pins listed in alphabetical order, with global ports listed last, also in alphabetical order. When no SPICE.OUTPUT property is present (nor the SPICE.PREFIX, SPICE.PINORDER, SPICE.MODEL, or SPICE.PARAMETERS properties), the SPICE written corresponding to each symbol instance will contain all pins, followed by all interface parameters. An interface parameter is a parameter with sub-property IsInterface = True.

Exporting SPICE for cell “Subcircuits”, we see the definition and call for Dig0 appears as follows:

```
.subckt Dig0 Clk Data<0> Data<1> Data<2> Data<3> Out<0> Out<1>
Out<2> Out<3> Vdd
...
.ends
XDig0_1 Clock A<0> A<1> A<2> A<3> B<0> B<1> B<2> B<3> PWR Dig0
```

Now consider cell Dig1. Cell Dig1 demonstrates the use of the SPICE.DEFINITION property to pass parameters on the definition of a subcircuit.



```
SPICE.DEFINITION = .subckt $Cell %% $$ Level = 5
```

Symbol with SPICE.DEFINITION to pass parameters

The symbol for cell Dig1 has a SPICE.DEFINITION property as follows:

```
SPICE.DEFINITION = .subckt $Cell %% $$ Level = 5
```

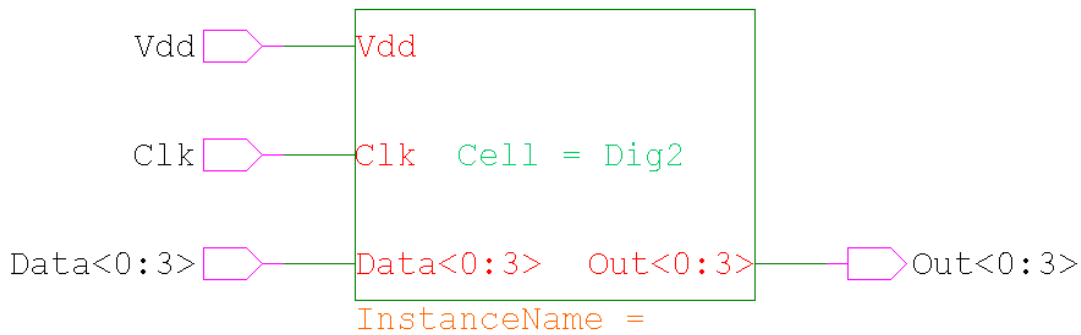
Inspecting the SPICE.DEFINITION statement in detail, each element of the property is written out on the SPICE definition interface as follows:

.subckt	Write “.subckt” literally
\$Cell	Write the name of the cell
%%	Write the names of the ports on the symbol in the default order.
\$\$	Write all interface properties, written as “property_name1 = property_value1 ...”
Level = 5	Write “Level = 5” literally

Exporting the SPICE for cell “Subcircuits”, we see the definition and call for Dig1 appears as follows:

```
.subckt Dig1 Clk Data<0> Data<1> Data<2> Data<3> Out<0> Out<1>
Out<2> Out<3> Vdd P1=10 P2=20 Level = 5
...
.ends
XDig1_1 Clock A<0> A<1> A<2> A<3> B<0> B<1> B<2> B<3> PWR Dig1 P1=10
P2=20
```

Now consider cell Dig2. Cell Dig2 demonstrates the use of the SPICE.OUTPUT and SPICE.DEFINITION properties to customize the pin order and to add special syntax to the definition and call for a subcircuit.



```
SPICE.DEFINITION = .subckt $Cell {%Data<3:0>} {%Vdd} {%Clk} {%Out<0:3>}
SPICE.OUTPUT = X${Name} {%Data<3:0>} {%Vdd} {%Clk} {%Out<0:3>} $MasterCell
```

Symbol with SPICE.OUTPUT and SPICE.DEFINITION to customize pin order and other syntax

The symbol for cell Dig2 has a SPICE.DEFINITION and SPICE.OUTPUT property as follows:

```
SPICE.DEFINITION = .subckt $Cell {%Data<3:0>} {%Vdd} {%Clk}
{%Out<0:3>}

SPICE.OUTPUT = X${Name} {%Data<3:0>} {%Vdd} {%Clk} {%Out<0:3>}
$MasterCell
```

Inspecting the SPICE.DEFINITION statement in detail, each element of the property is written out on the SPICE definition interface as follows:

.subckt	Write “.subckt” literally
\$Cell	Write the name of the cell
(Write “(“ literally
{%Data<3:0>}	Write Data ports, reversing the default order
)	Write “)” literally
{%Vdd}	Write Vdd port
{%Clk}	Write Clk port
(Write “(“ literally
{%Out<0:3>}	Write “Out” ports
)	Write “)” literally

Inspecting the SPICE.OUTPUT statement in detail, each element is similarly constructed. Exporting the SPICE for cell “Subcircuits”, we see the definition and call for Dig2 appears as follows:

```
.subckt Dig2 (Data<3> Data<2> Data<1> Data<0>) Vdd Clk (Out<0> Out<1>
Out<2> Out<3>)
...
.ends
XDig2_1 (A<3> A<2> A<1> A<0>) PWR Clock (B<0> B<1> B<2> B<3>) Dig2
```

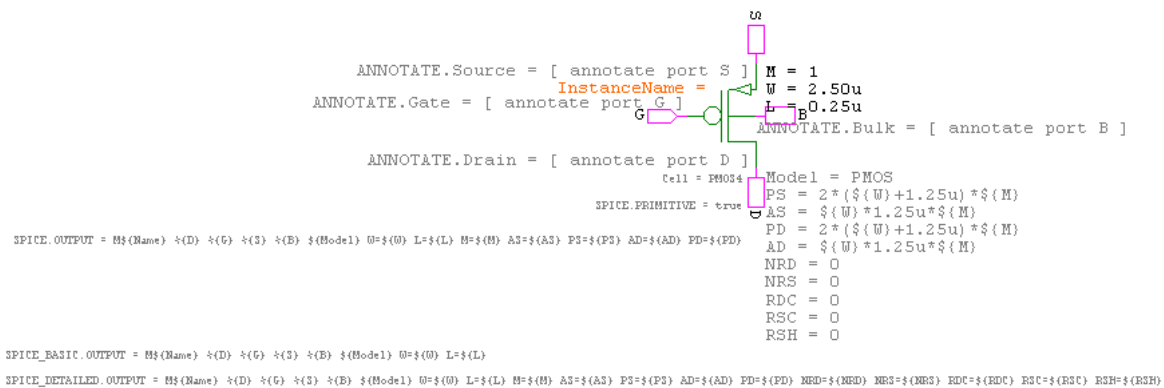
3.4. SPICE Export Control property

Design \S-Edit\Examples\SpiceOutput\SpiceOutput.tanner

Cells PMOS4

This example illustrates the use of the SPICE Export Control Property to control the SPICE output for a device. For a given device, one may want to define several SPICE output properties, for example a default property, a basic property, and a detailed property. Each property might have different parameters for different levels of simulation. The SPICE Export Control property determines which output property is used when writing out a SPICE netlist. When a list of property names is entered in the Export Control Property, SPICE will be written according to the first Export Control Property in the list that exists on the device being written.

Consider cell PMOS4, a PMOS transistor, whose symbol is shown below.



Symbol of NMOS4 transistor, view NMOS4_Expand

Three SPICE.OUTPUT properties are defined, as shown below. The SPICE Export Control Properties for these Output properties are SPICE_BASIC, SPICE, and SPICE_DETAILED.

```
SPICE_BASIC.OUTPUT = M$ {Name} % {D} % {G} % {S} % {B} $ {Model} W=$ {W} L=$ {L}
```

```
SPICE.OUTPUT = M$ {Name} % {D} % {G} % {S} % {B} $ {Model} W=$ {W} L=$ {L} M=$ {M}
AS=$ {AS} PS=$ {PS} AD=$ {AD} PD=$ {PD}
```

```
SPICE_DETAILED.OUTPUT = M$ {Name} % {D} % {G} % {S} % {B} $ {Model} W=$ {W}
L=$ {L} M=$ {M} AS=$ {AS} PS=$ {PS} AD=$ {AD} PD=$ {PD} NRD=$ {NRD} NRS=$ {NRS}
RDC=$ {RDC} RSC=$ {RSC} RSH=$ {RSH}
```

If we export SPICE from cell INV, and enter SPICE_DETAILED, SPICE, SPICE_BASIC for the SPICE Control Property, we get the following output:

```
MP1 Out A Vdd Vdd PMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u NRD=0 NRS=0 RDC=0 RSC=0 RSH=0
MN1 Out A Gnd Gnd NMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
```

The SPICE_DETAILED.OUTPUT property was used to export the PMOS4 instance because it was first in the SPICE Control Property list and it existed on the PMOS4 instance. No SPICE_DETAILED.OUTPUT property exists on the NMOS4 instance, so the next property in the list was used, which is the SPICE.OUTPUT property.

If we export SPICE from cell INV, and enter SPICE_BASIC, SPICE for the SPICE Control Property, we get the following output:

```
MP1 Out A Vdd Vdd PMOS W=2.5u L='0.25u-(10n*match)'
MN1 Out A Gnd Gnd NMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
```

The SPICE_BASIC.OUTPUT property was used to export the PMOS4 instance because it was first in the SPICE Control Property list and was present on the PMOS4 instance. No SPICE_BASIC.OUTPUT property exists on the NMOS4 instance, so the next property in the list was used, which is the SPICE.OUTPUT property.

If we export SPICE from Cell INV and enter only SPICE for the SPICE Control Property, we get the following output:

```
MP1 Out A Vdd Vdd PMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
MN1 Out A Gnd Gnd NMOS W=2.5u L='0.25u-(10n*match)' M=1 AS=3.125p PS=7.5u
AD=3.125p PD=7.5u
```

The SPICE.OUTPUT property was used for both the PMOS4 and NMOS4 instances.

4. Global Nets, Global Symbols, and Netcaps

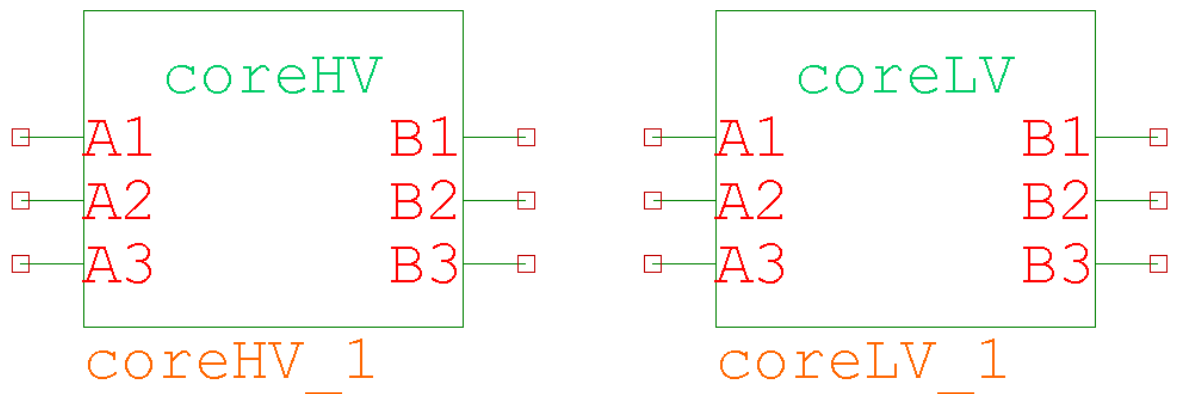
4.1. Simple Global Nets

Design \S-Edit\Examples\GlobalNets\Globals1\Globals1.tanner
Cell Toplevel

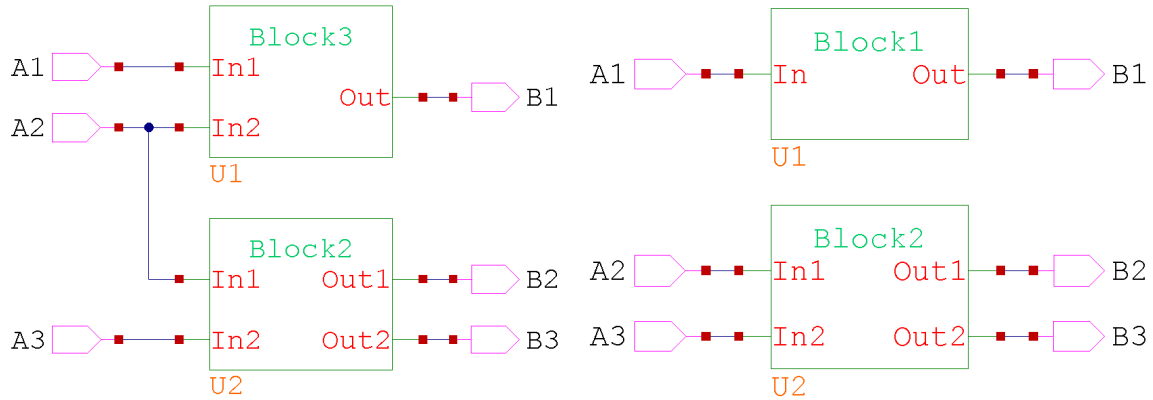
Global nets in S-Edit are connected through the design hierarchy, without explicitly placing ports for them at every level. In this example there are two cores, `coreHV` and `coreLV` instanced in cell `Toplevel`. Inside `coreHV`, we have instances of `Block2` and `Block3`, and inside `coreLV` we have instances of `Block1` and `Block2`. These can be seen in the `.subckt` definitions of `coreHV` and `coreLV` in the netlist below. Each schematic of `Block1`, `Block2`, and `Block3` has a global symbol for `Vdd` and `Gnd`.

In this design, `Vdd` and `Gnd` are global, and are connected through the entire design hierarchy.

Schematics



Schematic of cell `Toplevel`



Schematic of cell coreHV

Schematic of cell coreLV

SPICE Netlist

```
***** Subcircuits *****
.subckt Block1 In Out Gnd Vdd
.ends

.subckt Block2 In1 In2 Out1 Out2 Gnd Vdd
.ends

.subckt Block3 In1 In2 Out Gnd Vdd
.ends

.subckt coreHV A1 A2 A3 B1 B2 B3 Gnd Vdd
XU2 A2 A3 B2 B3 Gnd Vdd Block2
XU1 A1 A2 B1 Gnd Vdd Block3
.ends

.subckt coreLV A1 A2 A3 B1 B2 B3 Gnd Vdd
XU1 A1 B1 Gnd Vdd Block1
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

***** Simulation Settings - Parameters and SPICE Options *****

XcoreLV_1 N_10 N_8 N_11 N_9 N_12 N_7 Gnd Vdd coreLV
XcoreHV_1 N_3 N_5 N_2 N_4 N_1 N_6 Gnd Vdd coreHV

.end
```

4.2. Separate Power Supplies

Design \S-Edit\Examples\GlobalNets\Globals1\Globals2.tanner

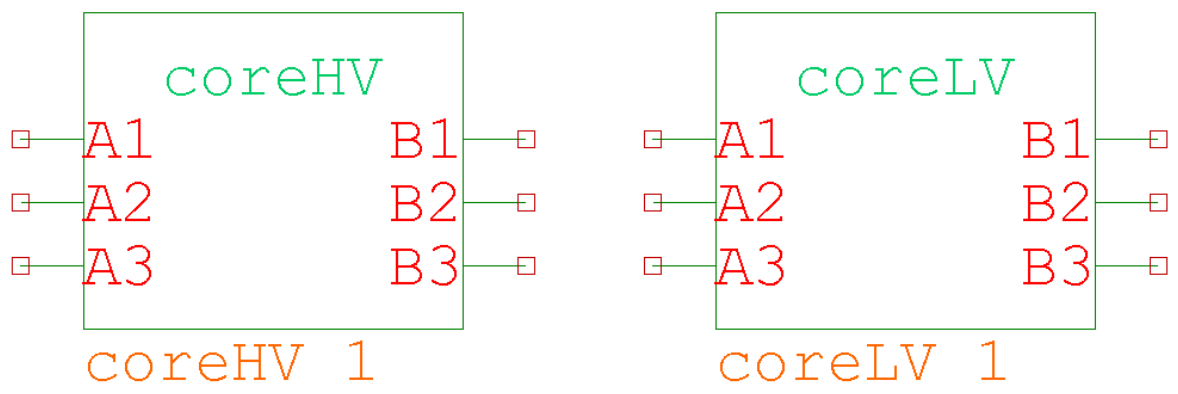
Cell Toplevel

This example illustrates how to isolate the global Vdd nets contained inside two cells. Consider the two core cells in the Globals1 design. We wish to isolate the global Vdd in coreHV from the global Vdd in coreLV.

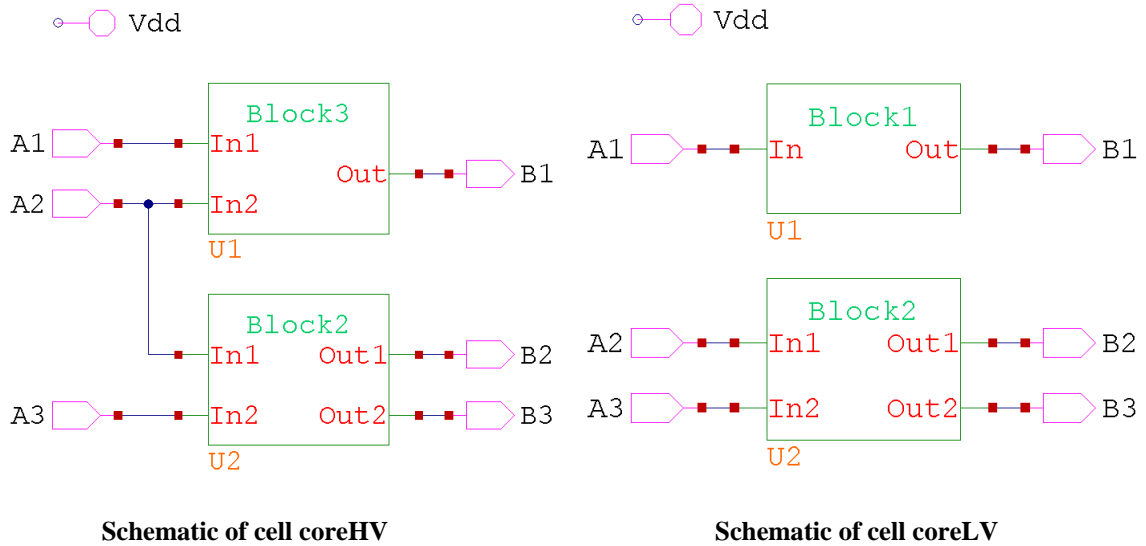
The design in Globals1 has been modified by adding netcaps for Vdd in coreHV and coreLV. The name of the netcap must match the name of the net being capped, including case sensitivity, in order for the net to be properly capped. Notice now that Vdd no longer appears in the parameter list for the definition of coreHV and coreLV, and is correspondingly absent in the calls to coreHV and coreLV in the main circuit. The Vdd inside coreHV is and the Vdd inside coreLV are therefore not connected to each other.

The Vdd nets in coreHV and coreLV can be reconnected by removing the netcaps, or alternatively by placing a .global Vdd command in the SPICE netlist. The .global command can be automatically put into the netlist in S-Edit, by creating a symbol with a property SPICE.OUTPUT = .global Vdd, and instantiating that symbol at the top level of the design. An example of this can be viewed by opening design example Globals5.

Schematics



Schematic of cell Toplevel



SPICE Netlist

```
***** Subcircuits *****
.subckt Block1 In Out Gnd Vdd
.ends

.subckt Block2 In1 In2 Out1 Out2 Gnd Vdd
.ends

.subckt Block3 In1 In2 Out Gnd Vdd
.ends

.subckt coreHV A1 A2 A3 B1 B2 B3 Gnd
XU1 A1 A2 B1 Gnd Vdd Block3
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

.subckt coreLV A1 A2 A3 B1 B2 B3 Gnd
XU1 A1 B1 Gnd Vdd Block1
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

***** Simulation Settings - Parameters and SPICE Options *****

XcoreLV_1 N_10 N_8 N_11 N_9 N_12 N_7 Gnd coreLV
XcoreHV_1 N_3 N_5 N_2 N_4 N_1 N_6 Gnd coreHV

.end
```

4.3. Renaming Separate Power Supplies

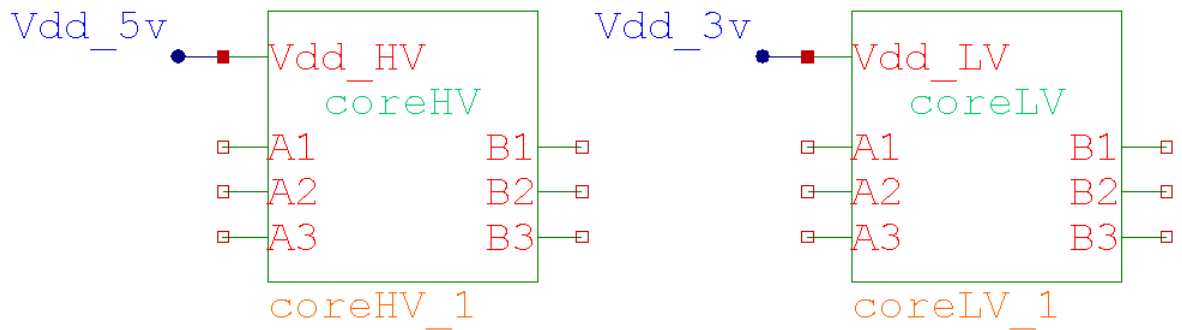
Design \S-Edit\Examples\GlobalNets\Globals1\Globals3.tanner

Cell Toplevel

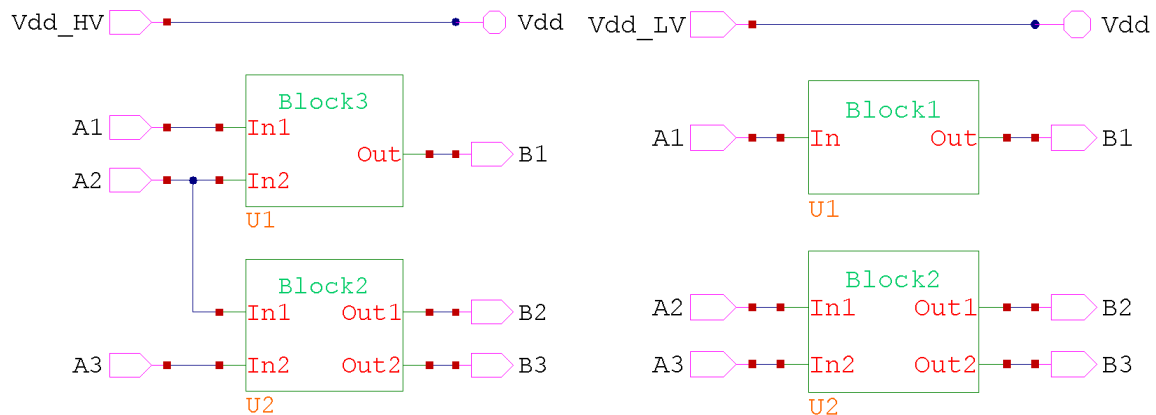
This example illustrates how to isolate the global Vdd nets in two cells from each other, and to connect to them with unique names. Consider the two core cells in the Globals2 design. In Globals2, we isolated the Vdd in coreHV from the Vdd in coreLV. We now wish to connect to coreHV with a net named Vdd_5v and to coreLV with a net named Vdd_3v.

In this example, the design in Globals2 has been modified by adding “In” ports Vdd_HV and Vdd_LV to cores coreHV and coreLV respectively, both on the schematic and symbol views. On the schematic views, the new ports are connected to the netcaps, thus continuing the propagation of the Vdd net up the hierarchy, but with a different name. In the calls in the main circuit, you can see nets Vdd_5v connecting to coreHV and Vdd_3v connecting to coreLV.

Schematics



Schematic of cell Toplevel



Schematic of cell coreHV

Schematic of cell coreLV

SPICE Netlist

```
***** Subcircuits *****
.subckt Block1 In Out Gnd Vdd
.ends

.subckt Block2 In1 In2 Out1 Out2 Gnd Vdd
.ends

.subckt Block3 In1 In2 Out Gnd Vdd
.ends

.subckt coreHV A1 A2 A3 B1 B2 B3 Vdd Gnd
XU1 A1 A2 B1 Gnd Vdd Block3
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

.subckt coreLV A1 A2 A3 B1 B2 B3 Vdd Gnd
XU1 A1 B1 Gnd Vdd Block1
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

***** Simulation Settings - Parameters and SPICE Options *****

XcoreLV_1 N_10 N_8 N_11 N_9 N_12 N_7 Vdd_3v Gnd coreLV
XcoreHV_1 N_3 N_5 N_2 N_4 N_1 N_6 Vdd_5v Gnd coreHV

.end
```

4.4. Globals4

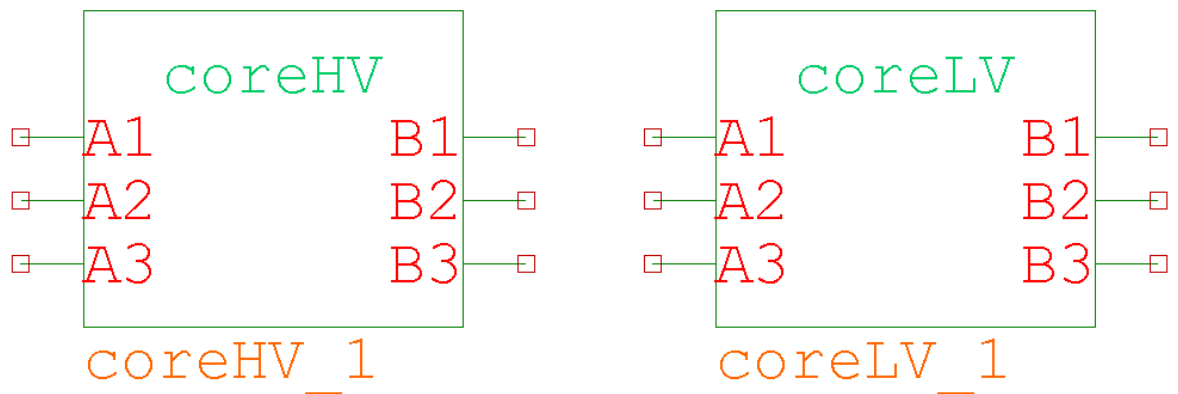
Design \S-Edit\Examples\GlobalNets\Globals1\Globals4.tanner

Cell Toplevel

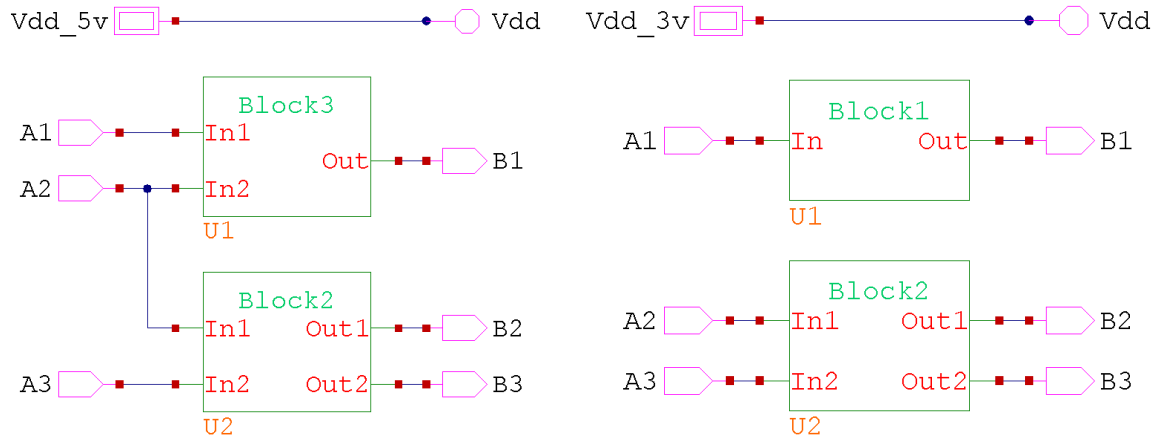
This example illustrates another way to isolate the global Vdd nets in two cells from each other, and to connect to them with unique names. Consider the two core cells in the Globals2 design. In Globals2, we isolated the Vdd in coreHV from the Vdd in coreLV. We now wish to connect to coreHV with a net named Vdd_5v and to coreLV with a net named Vdd_3v.

In this example, the design in Globals2 has been modified by adding “Global” ports Vdd_5v and Vdd_3v to the schematic views of cores coreHV and coreLV respectively. The new ports are connected to the netcaps, thus continuing the propagation of the Vdd net up the hierarchy, but with a different name. The name of the Global port takes precedence over the name of the netcap. In the calls in the main circuit, you can see nets Vdd_5v connecting to coreHV and Vdd_3v connecting to coreLV.

Schematics



Schematic of cell Toplevel



Schematic of cell coreHV

Schematic of cell coreLV

SPICE Netlist

```
***** Subcircuits *****
.subckt Block1 In Out Gnd Vdd
.ends

.subckt Block2 In1 In2 Out1 Out2 Gnd Vdd
.ends

.subckt Block3 In1 In2 Out Gnd Vdd
.ends

.subckt coreHV A1 A2 A3 B1 B2 B3 Gnd Vdd
XU1 A1 A2 B1 Gnd Vdd Block3
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

.subckt coreLV A1 A2 A3 B1 B2 B3 Gnd Vdd
XU1 A1 B1 Gnd Vdd Block1
XU2 A2 A3 B2 B3 Gnd Vdd Block2
.ends

***** Simulation Settings - Parameters and SPICE Options *****

XcoreLV_1 N_10 N_8 N_11 N_9 N_12 N_7 Gnd Vdd_3v coreLV
XcoreHV_1 N_3 N_5 N_2 N_4 N_1 N_6 Gnd Vdd_5v coreHV

.end
```


5. Multiple Views

5.1. MOSFET with 4- and 3-terminal symbols

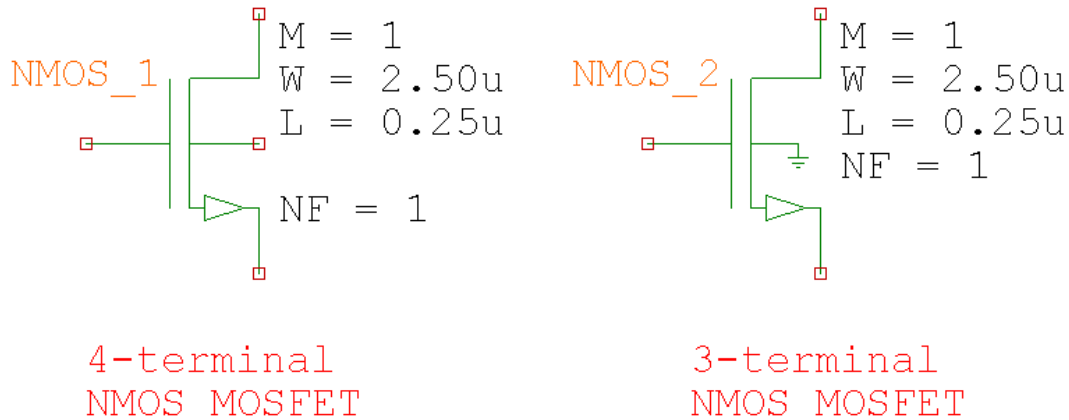
Design \S-Edit\Examples\MultipleViews\MultipleViews.tanner

Cell Toplevel and Devices\NMOS

This example illustrates the use of multiple views in a cell. The cell is an NMOS MOSFET, and there is a 4-terminal symbol and a 3-terminal symbol whose fourth terminal is automatically connected to ground. Cell NMOS consists of two interface views and two symbol views, as follows:

```
4-terminal NMOS MOSFET interface view:  NMOS4
4-terminal NMOS MOSFET symbol view:     NMOS4
3-terminal NMOS MOSFET interface view:  NMOS3
3-terminal NMOS MOSFET symbol view:     NMOS3
```

There is no schematic view in this case, as the cell is a SPICE primitive cell.



4- and 3-terminal symbols of an NMOS MOSFET

The fourth terminal of the 3-terminal MOSFET is connected to ground by writing 0 in the SPICE.OUTPUT property. Compare the SPICE properties of each symbol.

4-terminal SPICE.OUTPUT property:

```
SPICE.PREFIX = M
SPICE.PINORDER = D G S B
SPICE.MODEL = $Model
SPICE.PARAMETERS = W= L= M~ AS= PS= AD= PD= NRD~ NRS~ RDC~ RSC~ RSH~
GEO~ TABLES~
SPICE.OUTPUT is omitted
```

3-terminal SPICE.OUTPUT property:

```
SPICE.PREFIX = M
SPICE.PINORDER = D G S
SPICE.MODEL = $Model
SPICE.PARAMETERS = W= L= M~ AS= PS= AD= PD= NRD~ NRS~ RDC~ RSC~ RSH~
GEO~ TABLES~
SPICE.OUTPUT = ${SPICE.PREFIX}$Name %% 0 $Model $$
```

Cell PMOS is a 4- and 3-terminal PMOS MOSFET, analogous to NMOS.

5.2. NMOS with IEEE and IEC symbols

Design \S-Edit\Examples\MultipleViews\MultipleViews.tanner

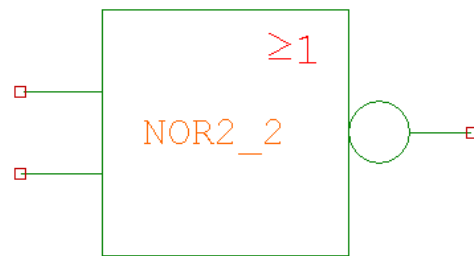
Cell **NOR2**

This example illustrates the use of multiple symbol views in a cell. The cell is a NOR gate, and there is an IEEE and IEC symbol. Cell NOR2 consists of one interface view, two symbol views, and one schematic view, as follows:

Interface view:	Main
IEEE symbol view:	IEEE
IEC symbol view:	IEC
Schematic view:	Main



IEEE symbol



IEC symbol

IEEE and IEC symbols for a NOR cell

Both symbols reference the same interface and the same schematic.

5.3. Adder with 3 different symbols

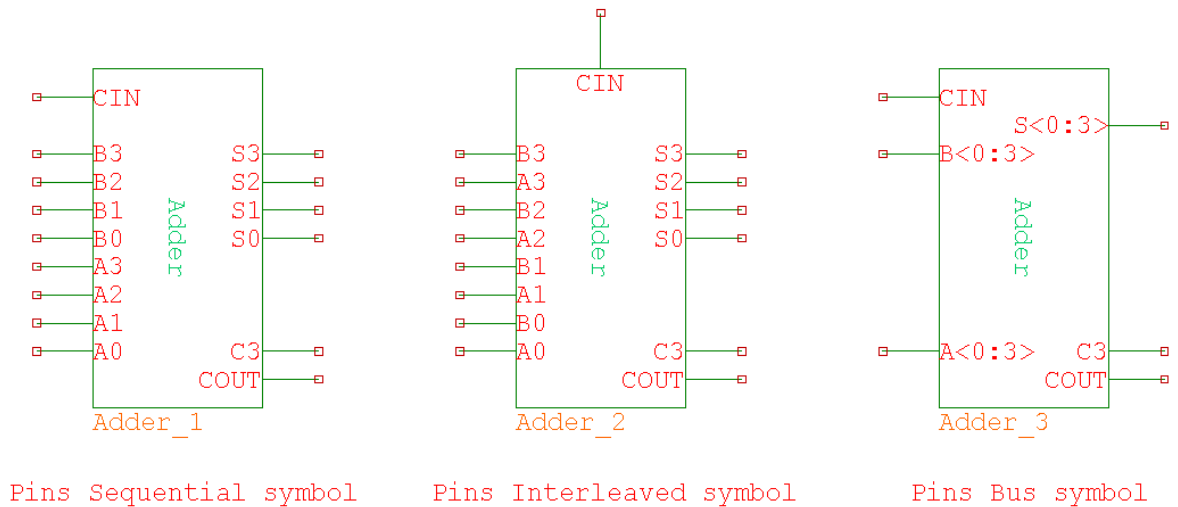
Design \S-Edit\Examples\MultipleViews\MultipleViews.tanner

Cell Adder

This example illustrates the use of multiple symbol views in a cell. The cell is an Adder, and there are three symbol views, one interface view, and one schematic view, as follows:

Interface view:	Main
Schematic view:	Main
Symbol view sequentially ordered:	Pins_Sequential
Symbol view interleaved:	Pins_Interleaved
Symbol view bus:	Pins_Bus

When drawing a schematic, it is sometimes convenient to have the pins of a symbol arranged in one particular order for making connections, and at other times one wants the pins arranged in a different order. This can be accomplished by having multiple symbol views, each of which has a different arrangement of pins. Here one symbol has input pins ordered on the left side as A0, A1, A2, A3, B0, B1, B2, B3, a second symbol has pins ordered as A0, B0, A1, B1, A2, B2, A3, B3, and a third symbol has pins grouped in busses. This is purely for drawing convenience, and does not affect the order of pins as written to SPICE.



Pins_Sequential, Pins_Interleaved, and Pins_Bus symbols for an Adder cell

6. TCL/TK Scripts

6.1. Modifying text

Script \S-Edit\Examples\Scripts\fixup_vdd_labels.tcl

This sample script illustrates how to cycle over all schematic views in the design, and change all ports and netlabels called “vdd” to “VDD”.

After loading the script, there are two functions:

<code>fix_vdd_names:</code>	Renames ports and net labels in the current cell from “vdd” to “VDD”
<code>fixall:</code>	Iterates through all schematic views in the database, calling “fix_vdd_names”

To run this script, first drag and drop it into the command window, then enter `fixall` to run the script. A full list of S-Edit TCL commands is available by typing `help` in the Command window. Help on any specific command, as well as a list of subcommands and options, can be obtained by entering the command name followed by `-help`.

```
# This sample script illustrates how to cycle over all schematic views in
# the design,
# and change all ports and netlabels called 'vdd' to 'VDD'
#
# After loading the script, there are two functions:
# fix_vdd_names: renames ports and netlabels in the current cell from
# 'vdd' to 'VDD'
# fixall: iterates through all schematic views in the database, calling
# 'fix_vdd_names'

# find all ports called 'vdd' and change them to 'VDD' (in the current
# view)
proc fix_vdd_names { args } {
    find port vdd
    property set -system Name -value VDD
    find netlabel vdd
    property set -system Name -value VDD
}
```

```

proc ForEachSchematicView { UserProcName } {
    mode renderoff
    set count 0
    set cells [ database cells ]
    foreach cell $cells {
        # traverse all interface views
        set interfaces [ database interfaces -cell $cell ]
        foreach interface $interfaces {
            # traverse all schematic views
            set views [ database views -cell $cell -type schematic
                        -interface $interface ]
            foreach view $views {
                # puts "Opening schematic($view) of cell $cell"
                cell open -cell $cell -view $view -type schematic
                    -interface $interface
                eval "$UserProcName $cell $view $interface"
                incr count
            }
        }
    }
    mode renderon
    return $count
}

proc fixall {} {
    set n [ForEachSchematicView fix_vdd_names]
    puts "$n views processed"
}

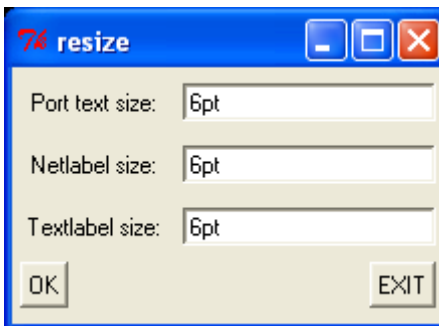
```

6.2. Resizing text with TK dialog to enter parameters

Script \S-Edit\Examples\Scripts\resizeText.tcl

This sample script illustrates how to cycle over all views in a design (schematic and symbol views) and modify the size of Ports, Netlabels, and Textlabels. It also shows how to use TK to write a dialog to enter parameters into a script. Note the use of the `toplevel` command to declare a window to write into. This is required for all TK scripts as the default `toplevel` window is the S-Edit application window, which the user is not permitted to modify.

After loading the script by dragging-and-dropping it into the command window, the following dialog appears. Enter appropriate text sizes, then press OK to resize Ports, Netlabels, and Textlabels in all views.

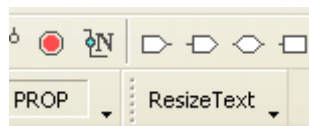


TK Dialog to enter Port, Netlabel, and Textlabel size.

A button to run the script can be created on a toolbar by removing the comment symbol (#) from the `workspace` command. Also comment out the `ResizeText` command so the script does not execute immediately when it is loaded, but only when the button is pressed.

```
workspace userbutton set ResizeText
# ResizeText
```

Modifications to script to add a button to a toolbar.



User button on toolbar to run the ResizeText script.

If this is a commonly used script, it can be placed in the **scripts\startup** folder to automatically get loaded when S-Edit starts up. The location of the startup folder is:

C:\Documents and Settings\<username>\Application Data\Tanner EDA\scripts\startup

Scripts can also be loaded automatically whenever a design is opened, or when S-Edit is shutdown. The location of the folders for these scripts is:

C:\Documents and Settings\<username>\Application Data\Tanner EDA\scripts\open.design

C:\Documents and Settings\<username>\Application Data\Tanner EDA\scripts\shutdown

The ResizeText script is shown below:

```
# This sample script illustrates how to cycle over all schematic views in
the design,
# and size all ports, netlabels, and labels to a user specified size. The
script shows
# how to use TK to create a simple dialog to enter parameters for the
script.

# resize all ports, netlabels, and labels (in the current view)
proc resize_text { new_portsize new_netlabelsize new_textlabelsize } {
    find none
    if { [string length $new_portsize] } {
        find port
        property set -system FontSize -value $new_portsize
    }
    if { [string length $new_netlabelsize] } {
        find netlabel
        property set -system FontSize -value $new_netlabelsize
    }
    if { [string length $new_textlabelsize] } {
        find textlabel
        property set -system FontSize -value $new_textlabelsize
    }
    find none
}

proc resizetext { new_portsize new_netlabelsize new_textlabelsize } {
    mode renderoff
    foreach cv [ database cellviews ] {
        set type [lindex $cv 4]
        if { $type == "schematic" || $type == "symbol" } {
            set design [lindex $cv 0]
            set cell [lindex $cv 1]
            set view [lindex $cv 2]
            set interface [lindex $cv 3]

            cell open -design $design -cell $cell -view $view
                    -interface $interface -type $type
            resize_text $new_portsize $new_netlabelsize
                        $new_textlabelsize
        }
    }
    mode renderon
}
```



```

proc ResizeText {} {
    global resize_portsize
    global resize_netlabelsize
    global resize_textlabelsize

    set resize_portsize 6pt
    set resize_netlabelsize 6pt
    set resize_textlabelsize 6pt

    toplevel .resize

    set portprompt [label .resize.portprompt -text "Port text size:" ]
    set portsize [entry .resize.portsize -textvariable resize_portsize ]

    set netlabelprompt [label .resize.netlabelprompt -text
        "Netlabel size:" ]
    set netlabelsize [entry .resize.netlabelsize -textvariable
        resize_netlabelsize ]

    set labelprompt [label .resize.labelprompt -text "Textlabel size:" ]
    set labelsize [entry .resize.labelsize -textvariable
        resize_textlabelsize ]

    set cmdframe [ frame .resize.buttons ]
    set ok [ button .resize.buttons.ok -text OK -command { \
        resizetext $resize_portsize $resize_netlabelsize
        $resize_textlabelsize ; \
        destroy .resize \
        } ]
    set quit [ button .resize.buttons.quit -text EXIT -command {destroy
        .resize} ]
    pack $ok -side left
    pack $quit -side right

    grid $portprompt $portsize -row 0 -sticky ew -padx 4 -pady {8 4}
    grid $netlabelprompt $netlabelsize -row 1 -sticky ew -padx 4 -pady
        {8 4}
    grid $labelprompt $labelsize -row 2 -sticky ew -padx 4 -pady {8 4}
    grid $cmdframe -column 0 -columnspan 2 -row 3 -sticky ew -padx 4 -
        pady {4 8}
    grid columnconfigure .resize 1 -weight 1
    focus $portsize
}

#workspace userbutton set ResizeText
ResizeText

```