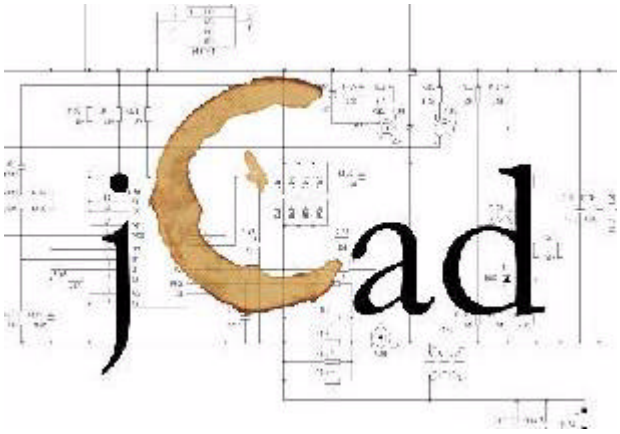

jCad Reference

updated for jCad 0.27

2002-01-06

Table of Contents

CHAPTER 1

Legal Stuff

1.1 jCad Background

jCad is modeled after the commercial “Vanguard” electrical CAD system. Vanguard was originally developed by CASE Technology Inc, then taken over by TERADYNE Inc. The product was withdrawn form the European and US markets by Teradyne at the beginning of the 1990’s, but was taken over by Sophia Systems, Japan, who still provide a Japanese only version of the system.

What I have done is a from scratch design of a very similar system - called jCad. jCad has a very similar user interface, and uses exactly the same file formats for all work files as Vanguard version 4.1 I have not used (or even seen) a single line of code from the original Vanguard product - evidence of this is that jCad is entirely written in Java, which did not exist at the time of this Vanguard version.

Some files in the distribution have some connection to Vanguard:
The main configuration file “config.g” is a copy from my licensed Vanguard copy, but it has been modified in several aspects.
The font files are copies from my licensed Vanguard copy, but they have been modified (scaled for more narrow appearance).
The component libraries shipped with the distribution were originally drawn using Vaguard - but are not the libraries shipped together with Vanguard.
All other parts of the jCad distribution are completely written/designed from scratch for jCad.

To my understanding this should be OK - no one can claim stolen property of any kind. If anyone does not agree in my view on this, please tell me so I can correct. (address, see below).

1.2 Disclaimer

jCad is a piece of software under construction. I provide no support, and take no responsibility for how it might affect your computer or files. Use at own risk.

1.3 License

I have not figured this out yet, but the intention is to keep everything open source software (GNU GPL?). Until further notice, use it for any purpose.

Mats Barkell

email: Mats.Barkell@era.ericsson.se

CHAPTER 2

Introduction

This document is yet very preliminary, it just contains the things necessary to configure jCad - setting up custom parts lists, using other locations of libraries and other files.

2.1 jCad Command Line

Optional command line parameters for jCad are:

JCADHOME=*jcadpath*
CURDIR=*cwdpath*

"*jcadpath*" specifies the path to the directory (without trailing "/" or "\") of the config.g file to be used. The config.g file defines all other file configurations. If the JCADHOME=*jcadpath* parameter is omitted, the current directory is assumed.

"*cwdpath*" specifies the current directory (without trailing "/" or "\") to be used by jCad. All relative paths specified in config.g will be relative to this directory. Use of this parameter enables jCad to be started by a desktop icon - or several different icons for different projects.

CHAPTER 3

Simulation

3.1 Spice

3.1.1 Spice config.g configuration

See the description of the spice configuration in config.g in .

3.1.2 Drawing requirements for Spice interface commands

A drawing that is intended to produce a .cir (for just saving or to run the simulator) must contain a component having MNAME equal to SPICE. An example such component is provided in the spicesao.lib schematic library file. This component should have properties named as an contigous series of numbers starting from “1”. The values of these properties produce command lines in the written .cir file. Since at least the kind of simulation is necessary input to spice, there must be at least one such property. All these property values are written directly to the .cir file. Example properties of the SPICE component might be:

property name	property value
1	.TRAN 1n 1m
2	.OPT ABSTOL=1E-9
3	.PLOT V(R21)

A drawing taht is intended to produce a any subcircuit (.sbc or .lib) file must contain a component having MNAME equal to SPC_SUBCCT. This component must have a property SPC_TERMINALS set to a value defining the signals that are the terminals to the subcircuit. It must also have a property SPC_PARAMS if the subcircuit has any parameters. If the drawing will be saved as a library (.lib) file, it must also have property SPC_LIBFILE set to an appropriate value. Also “number properties” according to above described for the SPICE component may be used, their contents is written before the subcircuit definition. The subcircuit name is taken from the drawing filename, by stripping off any leading path and the .drw extension. The generated .lib or .sbc file output thus contains (optional info in brackets):

[any “number property” values]
...
.subcircuit .drw-filname SPC_TERMINALS [params: SPC_PARAMS]
component lines
...
.ends .drw-filename

The SPICE and/or SPICE_SUBCCT components do not generate any connectivity information - they are removed from the component list before the circuit definition information is written.

3.1.3 General notes on conversion to spice format

All conversion from drawing (.drw file) to spice (.cir, .sbc or .lib) format uses the WDP (.wdp) format as an intermediate step. This means that properties having “%” as the first letter of their name are interpreted and then removed. At present time, the only such property is %PIN, which can be used to add pins not visible in the drawing (see separate description of WDP special properties elsewhere). The most common use of %PIN properties is to add power supply pins.

3.1.4 Component requirements for Spice interface commands

All components that are supposed to generate connectivity information in spice (.cir, .sbc or .lib) files, must have a SPC_TEMPLATE property. The value of this property defines the component line in the spice file completely. See below for the format of this template value.

A component may also have a SPC_INCL property stating a filename as its value. The file referred to by this property normally is a .sbc or .lib file containing any necessary .model or .subcircuit definitions.

Spice pin names for the component could use the normal pin names, or they can be overridden by using pin properties SPC_PIN. The use of this property to define pin names is recommended, since it makes the simulation models independent of any physical package constraints. The value of the SPC_PIN property also allows a comma-separated list of more than one name - useful for example MOSFETs having just three pins, having three pins in the drawing while the spice model requires four.

See also below on handling of special properties starting with “%”.

3.1.5 Definition of SPC_TEMPLATE format

The value of the SPC_TEMPLATE property for each component is a text string that generates the corresponding spice circuit description output. This text string is however translated using the following rules before being written to the circuit description:

The characters @, &, ?, ~, #, % and ^ are special, and are expanded as follows:

A “*@name* ”, where *name* is a property name used by the component, is replaced by the value of the same property. If the property is not used within the component, an error will occur.

A “&*name* ” behaves the same as “*@name*”, except that it does not produce an error. If the property is not used within the component, the “&*name*” is just removed from the template before being further expanded.

A “%*name* “, where *name* is a pin name used by the component, is replaced by the name of the signal connected to that pin. If the pin is not used by the component, an error will occur.

A “^” defines a mandatory beginning of the circuit description line - all characters up to the “^” character in the template must appear also in the output line. If they do not do so naturally, characters are inserted as necessary. Characters up to and including the “^” character are first removed, and the rest of the template is evaluated/expanded. Finally, it is checked whether the removed characters, excluding the “^”, appear in the resulting axpanded template - if not, characters are added a necessary.

The “?”, “#” and “~” define various conditional structures - description to be added.

CHAPTER 4

File Formats

4.1 config.g File

The config.g is the common configuration file controlling the behaviour of jCad. It is a ASCII text file read once at startup - any changes made from within jCad are thus "per session" only. It is split in a number of subsections which must appear in the below described order. Text lines between the her described sections is normally ignored.

4.1.1 Header

This section contains the two rows:

CONFIG.G
411

4.1.2 File Configuration

This section has the format

;FILE CONFIGURATION
pathlist (Schematic Library Search Paths)
pathlist(Schematic Drawings Serach Paths)
empty line *//Skipped*
pathlist(Layout Library Search Paths)
pathlist(Layout Drawing Search Paths)
empty line *//Skipped*
pathlist() *//Skipped*
pathlist() *//Skipped*
empty line *//Skipped*
Fonts Directory
Help File *//Read but not used*
Common Macro File *//Read but not used*
empty lines (1-20 lines) *//Skipped*

Each "*pathlist*" is a comma separated list of directory names. The list ends after the first entry not ended by a comma and may contain line breaks. Directory names may be relative (to the current working directory) or absolute and must end with the file separator ("/" or "\").

4.1.3 Numeric Parameters

This section has the format (none of it currently used):

;NUMERIC PARAMETERS
maximum 200 text lines *//Skipped*
;

4.1.4 Colour Parameters

This section has the format:

;COLOR PARAMETERS

max 20 text lines *//Skipped*
Schematic Color Map
colourdef()
;
Layout Color Map
colourdef()
;
max 1000 text lines *//Skipped*
Oscilloscope Color Map
max 100 text lines *//Skipped*
;
colournames()
;
devicecolours()
;

Each "*colourdef*" contains zero to 256 lines of text of the format "n m", where n is a layer number and m is the colour number to assign to that layer. All layers not listed here get are assigned colour number zero.

"*colournames*" is a repetion of lines of the form:

name
m

where "name" is a colour name and "m" is the corresponding colour number. This format is repeated for each of the 16 colours used by jCad.

"devicecolours" is a repetition of lines of the form:

m r g b

where m is the colour number and r, g, b is the red, green and blue values of the corresponding colour. This format is repeated for each of the 16 colours used by jCad.

4.1.5 On/Off Parameters

This section has the format (none of it currently used):

;ON/OFF PARAMETERS
maximum 1000 text lines *//Skipped*
;

4.1.6 Special Strings

This section has the format:

;SPECIAL STRINGS
28 strings used by jCad //Really?
;

4.1.7 String Constants

This section has the format:

;STRING CONSTANTS
stringdef()
999

where "*stringdef*" is a repetition of lines of the form "n text", where n is a number (<999) and text is an arbitrary text string. These strings are used in dialogs, prompts, messages and also as keywords.

4.1.8 Error Messages

This section has the format:

;ERROR MESSAGES
errmsg(n)
errmsg(999)

where "*errmsg(n)*" is a repetition of lines of the form "n[l] text", where n is the errmsg argument, l the error severity level "I" (information), "W" (warning) or "F" (fatal) and text is the associated error text.

4.1.9 BOM Definitions

This section has the format:

;BOM DEFINITIONS
bomdef()
;

where "bomdef" is a repetition of descriptions of the format:

maindef()
coldef()

"*maindef*" is here a single line of the format:

name;*ext*;*mergesect*;*fd*;*ld*;*pd*;*pl*

where "*name*" is the identification name for this type of bom, "*ext*" its file extension, "*mergesect*" a YES/NO flag controlling whether sections of a component should be merged (YES) or listed separately (NO), "*fd*" is the field delimiter string, "*ld*" the line delimiter string, "*pd*" the page delimiter string and "*pl*" the page length (row count). The "*fd*", "*ld*" and "*pd*" delimiters allow use of C-style escapes for non-printing characters.

"*coldef*" is a repetition of lines defining the columns of the bom, each having format:

+*header*;*desc*;*length*;*merge*;*md*;*sort*

where "*header*" is the column name, "*desc*" is the property name or the keyword "QTY" to be shown in this column, "*length*" is the width of this column (-1 means no padding), "*merge*" is a YES/NO flag whether multiple values of "*desc*" are shown on the same row, "*md*" is the delimiter string between values (if merge=YES) and "*sort*" is a YES/NO flag indicating whether the bom should be sorted based on the values of this column or not.

The number of "*bomdef*":s allowed is limited only by the main window prompting text field size, choosing short "*name*":s makes a higher number practical.

4.1.10 Simulation Configuration

This section has the format:

;SIMULATION CONFIGURATION
keywords()
;

where allowed "keywords" are according to the below definitions of spicecmd and spicelib statements:

spicecmd=*text*

here "*text*" is the command line to be used. If it contains the strings "$INFILE" and/or "$OUTFILE", these strings are replaced by the drawing file name havings its normal .drw extension replaced by ".cir" and ".dat" respectively.

spicelibpath
pathlist(Spice Library Search Paths)

where "*pathlist*" is a comma separated list of directory names. The list ends after the first entry not ended by a comma and may contain line breaks. Directory names may be relative (to the current working directory) or absolute and must end with the file separator ("/" or "\").

4.1.11 Decoding Configuration

This section has the format:

This section has the format (none of it currently used):

;DECODING CONFIGURATION
maximum 1000 text lines *//Skipped*
;

4.1.12 Keyboard Configuration

This section has the format (none of it currently used):

;KEYBOARD CONFIGURATION
maximum 5000 text lines *//Skipped*
;

4.1.13 Keyboard Macros

This section has the format (none of it currently used):

;KEYBOARD MACROS
maximum 1000 text lines *//Skipped*
;

4.1.14 Menu Configuration

This section has the format:

;MENU CONFIGURATION
maximum 1000 text lines *//Skipped*
; end of preselection strings
menudef()
;

where "*menudef*" is a repetition of menu and submenu definition lines. Lines starting by "#" are immediately ignored. For other lines, leading whitespace is allowed (it is used in config.g only to indicate menu nesting level, removed by jCad before interpreted). All other lines have "*pline*", "*sline*", "*tline*", "*aline*" or "*mline*" formats. "*pline*" and "*sline*" are currently not used by jCad (intrended for context based preselctions). A "tline" is used to describe a submenu with the following format:

T:*flags*: :*text*
menudef()
;

A "*aline*" describes a menu item by the format:

A:*flags*:*cmd*:*text*

A "*mline*" describes a menu item by the format:

M:*flags*:*cmd*:*text*

"*flags*" defines in which jCad window types this menuitem or submenu is visible. It is a 14 to 16 characters field containing blanks or the any of the letters IOACHLMTESRVUWF, where each of the letters correspond to a jCad window type. "*cmd*" describes the action to take when this menuitem is selected. It contains (0-3) blanks for "*tline*":s, a macro number for "*aline*":s, and a text macro name (defined in shells.mac) for "*mline*":s. "*text*" is the menu text displayed, it must contain at least one ">" for "*tline*":s.

Macro numbers are described somwhere else in this document.

Text macros are currently not used by jCad.

The letters of the IOACHLMTESRVUWF string has the following translation to jCad windows:

I	Property List window	
O	Oscilloscope window	//not used
A	Simulation Data window	//not used
C	Component Editor window	
H	Help List window	//not used
L	Line Editor	
M	Property window	
T	Component List window	
E	Document Editor window	//not used
S	Message window	//not used
R	Reference window	//not used
V	Schematic Editor window	
U	Layout Editor window	
W	(not used)	
F	(not used)	