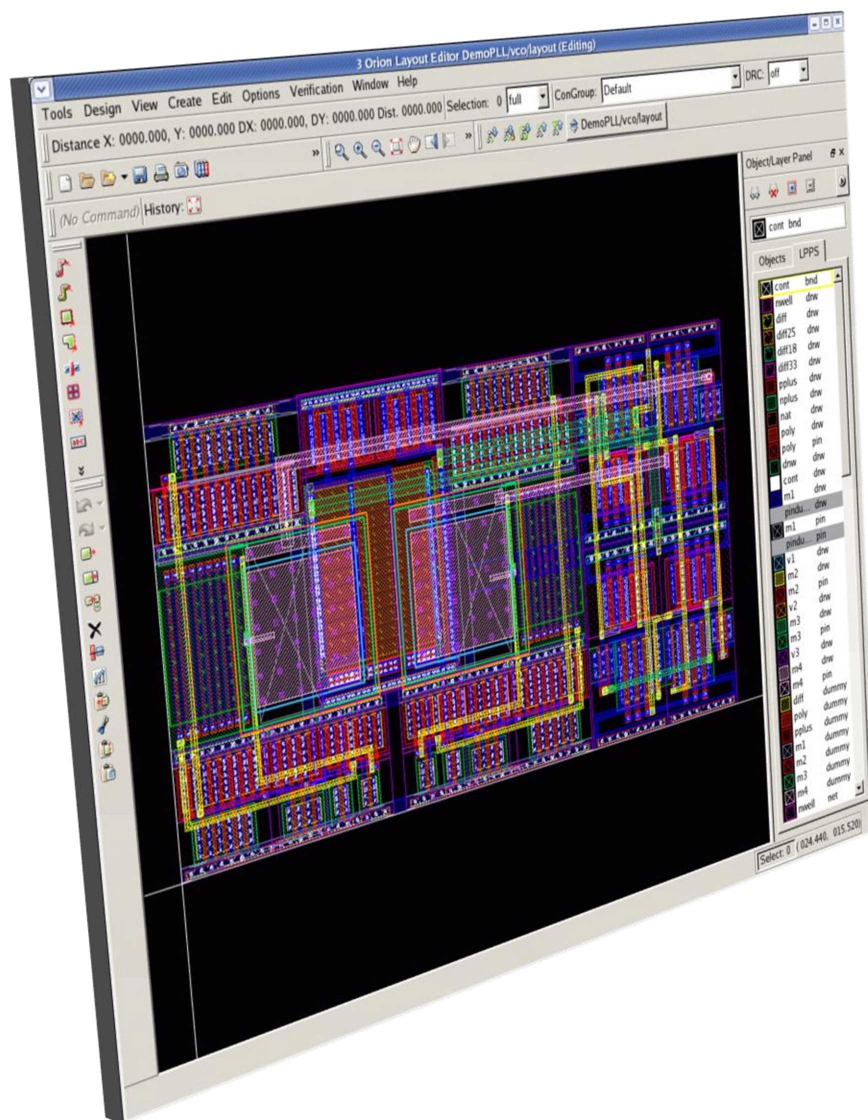
SYNOPSYS®
Predictable Success



Custom Designer: An Open Custom Design Platform

Scott Chase

November 6, 2008

Overview

- **An Introduction to Custom Designer**

A product-level overview of features and capabilities of Custom Designer from an R&D Perspective

- **Extending Custom Designer**

Customization of Custom Designer and Integration of 3rd-Party Tools

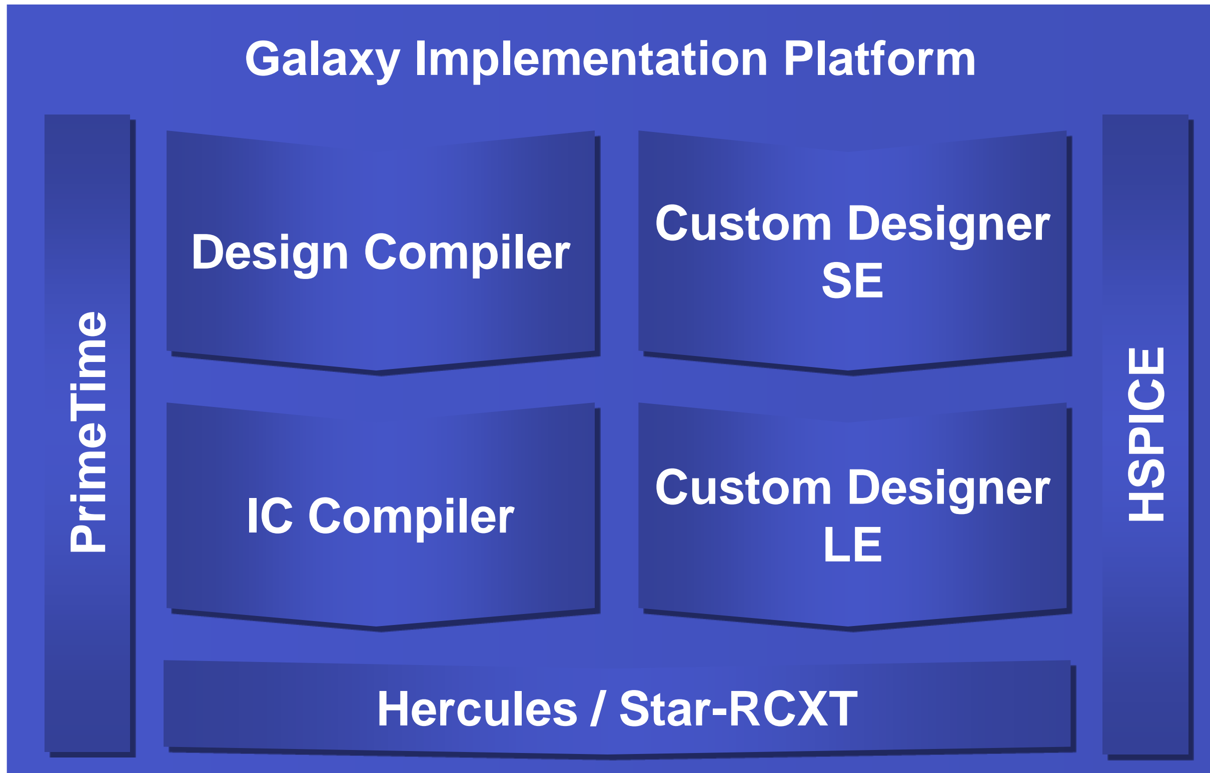
- **A Fully-Coded Real World Example**

Integrating the CiraNova Helix Analog Placer into Custom Designer

Synopsys
Predictable Success

An Introduction to Custom Designer

Custom Designer
An Open OA-based Custom Design Platform

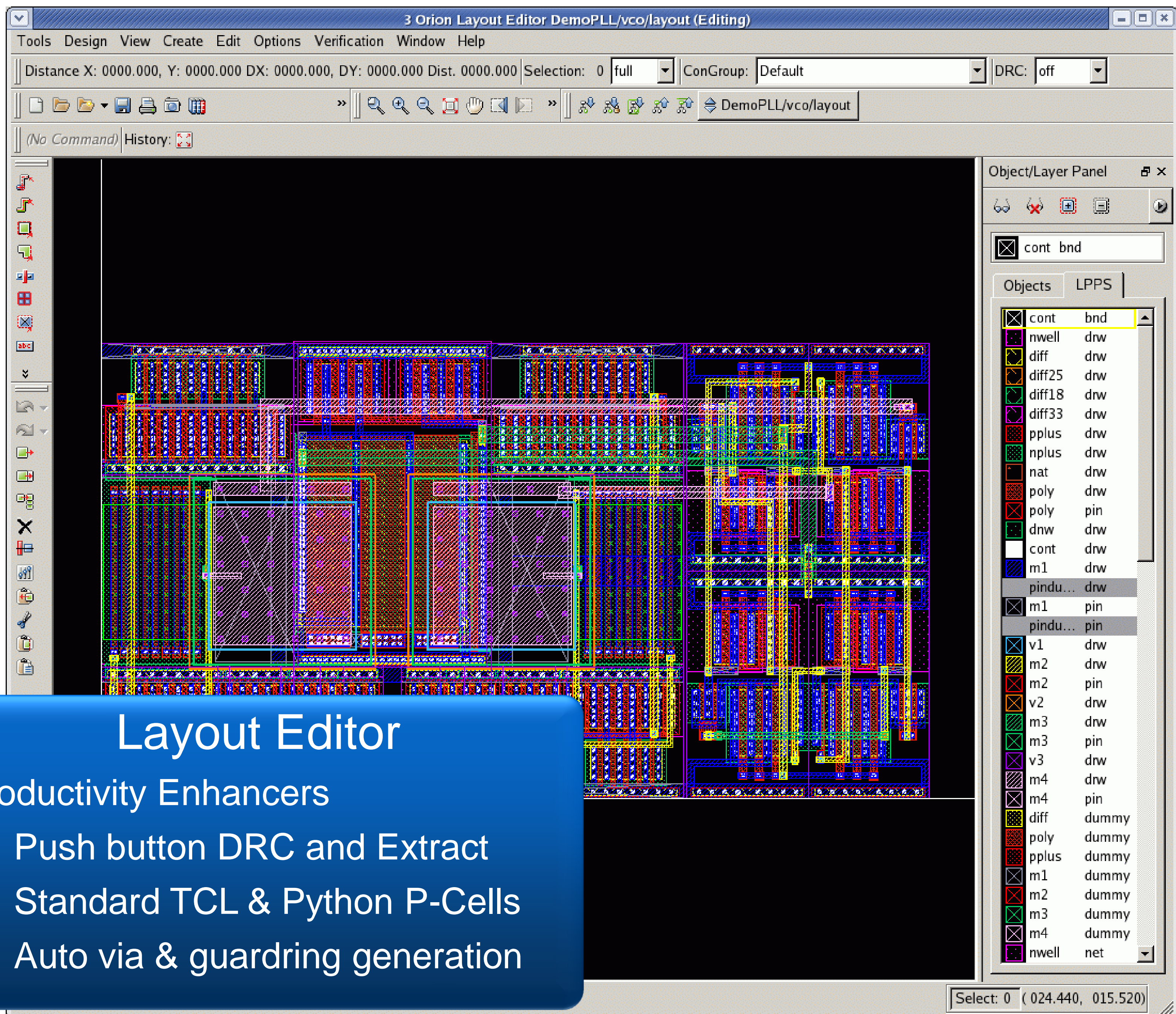


- Architected for productivity
- Complete custom design solution
- Open environment

- Production Release announced Sept 22, 2008 at BSNUG

©Synopsys 2008

SYNOPSYS® Predictable Success



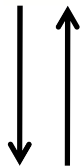
Layout Editor

Productivity Enhancers

- Push button DRC and Extract
- Standard TCL & Python P-Cells
- Auto via & guardring generation

Network Architecture for Graphical Applications using X + VNC

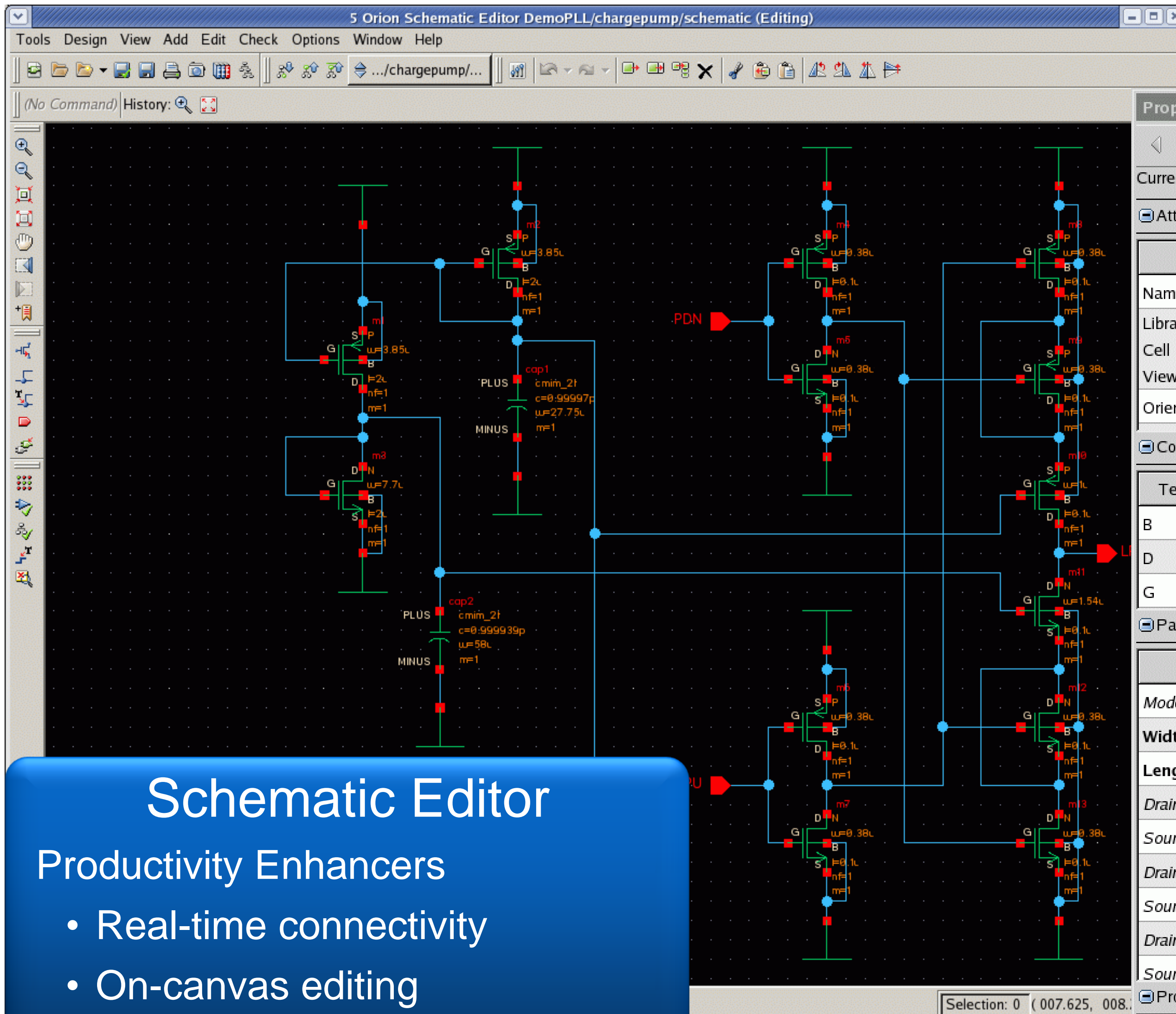
X Client running
Custom Designer



Socket connection (file descriptor or TCP/IP)

X Server hosting DISPLAY

SYNOPSYS®
Predictable Success

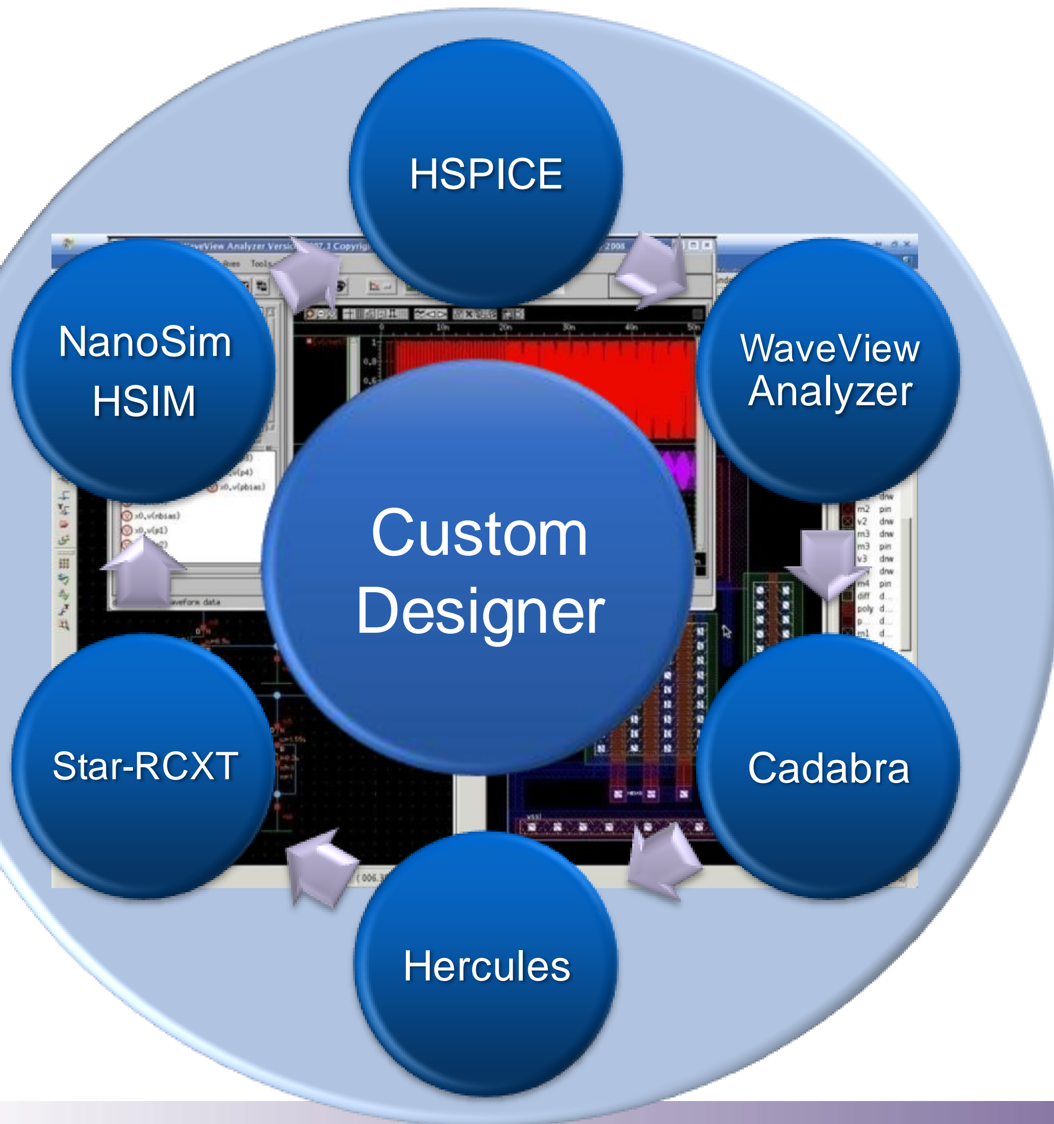


Schematic Editor

Productivity Enhancers

- Real-time connectivity
- On-canvas editing
- Smart connect

Complete & Comprehensive Solution



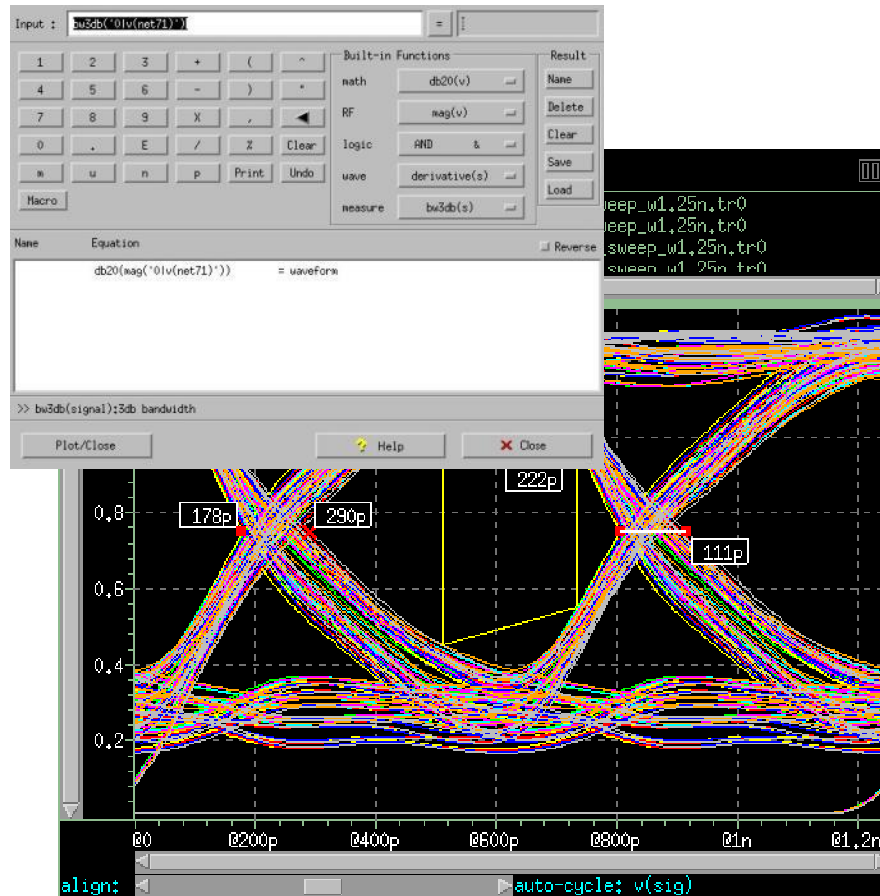
- Unified environment
- Familiar look and feel
- Full custom chip and block authoring flows

©Synopsys 2008

SYNOPSYS®
Predictable Success

Complete & Comprehensive Solution

WaveView Analyzer



- Highest capacity and performance
- Complex analysis toolbox
- Automated TCL verification scripting

SYNOPSYS®
Predictable Success

Custom Designer
An Open OA-based Custom Design Platform

•Primary Editors: LE, SE, Symbol editors

• Supporting Tools:

Library Manager, Hierarchy Editor, Display Resource Editor, Job Monitor, Programmable Netlister, etc.

• Dockable Assistants:

Property Editor, Hierarchy Navigator, Marker Browser, Device Label Editor, Probe Assistant, Transaction History, Object Layer Panel, Schematic Object Filter, etc.

SYNOPSYS®
Predictable Success

Extending Custom Designer:

Customization of Custom Designer and Integration of 3rd-Party Tools

Custom Designer

An Open OA-based Custom Design Platform

Major EDA Platforms for custom design have been closed in 4 key ways:

- Proprietary Database
- Proprietary Libraries
- Scripting languages the only way to extend
- Proprietary UIs

That is changing. With Custom Designer:

- Design Database is Open Access DM4
- Support for IPL's interoperable PDK, including PyCells
- Scripting interface is Tcl, including oaTcl
- GUI Toolkit is Qt 4.x

• Customize the platform through standard interfaces, or just dynamically link in your own stuff with Tcl's 'load' command.

Infrastructure for Customizing and Extending Custom Designer

- Key Bindings
  - Define hotkeys for your favorite Tcl command, action or user-defined Tcl proc
- Window Framework customization
- Design Editor Tools
- Command Management
- Events and Callbacks
- Opal User Interface Design
- Shared Object Load
- Design Editor Plug-in Interface

SYNOPSYS® Predictable Success

Window Framework Customization

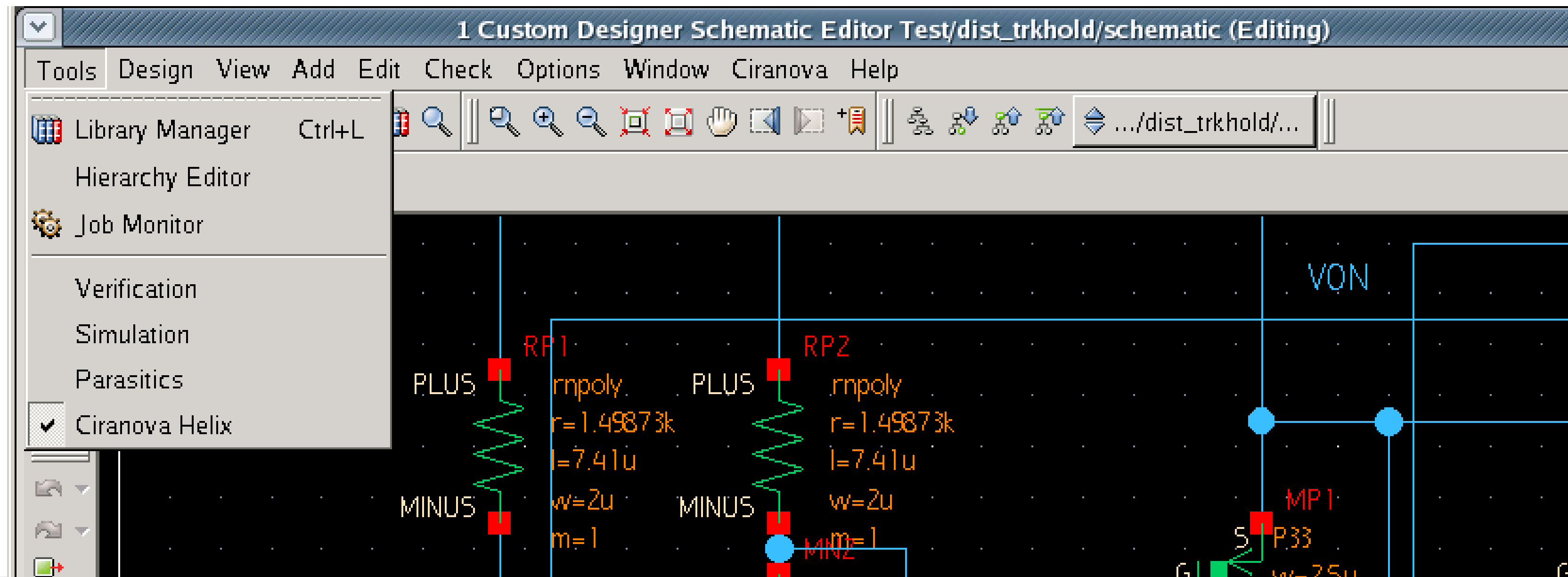
- Menus
  - add/remove from Menu Bar
  - add/remove actions from Menu
  - preShow proc allows dynamic content
- Toolbars
  - add/remove from windowType or window
  - add/remove actions from Toolbar
  - show/hide
  - change dock position
- Dockable Assistants
  - add/remove from windowType or window
  - show/hide
  - dock/undock/change dock
  - write your own Dockable Assistant!

Design Editor Tools

A *Tool* is an add-on component which augments the functionality provided by the editor and may be used to provide addition interfaces for advanced features such as Physical Verification or Simulation Interfaces.

Tools may be defined in C++ or Tcl.

An initProc obtains the window associated with the context and adds its interface elements to the window.



SYNOPSYS® Predictable Success

Create your own Command

de::createCommand provides command registration which integrates a Tcl proc (or C++ function) into:

- Console Help
  - Command Category
  - --help for Syntax Statement
- Tab Completion of:
  - Command Name
  - Argument Names
  - Enumerated Types
  - Path names
- Pre-execution argument name/value parsing and validation
  - Control does not pass to the execute() method unless arguments are valid
- Shift-Tab Wild-card Completion

SYNOPSYS® Predictable Success

Events and Callbacks

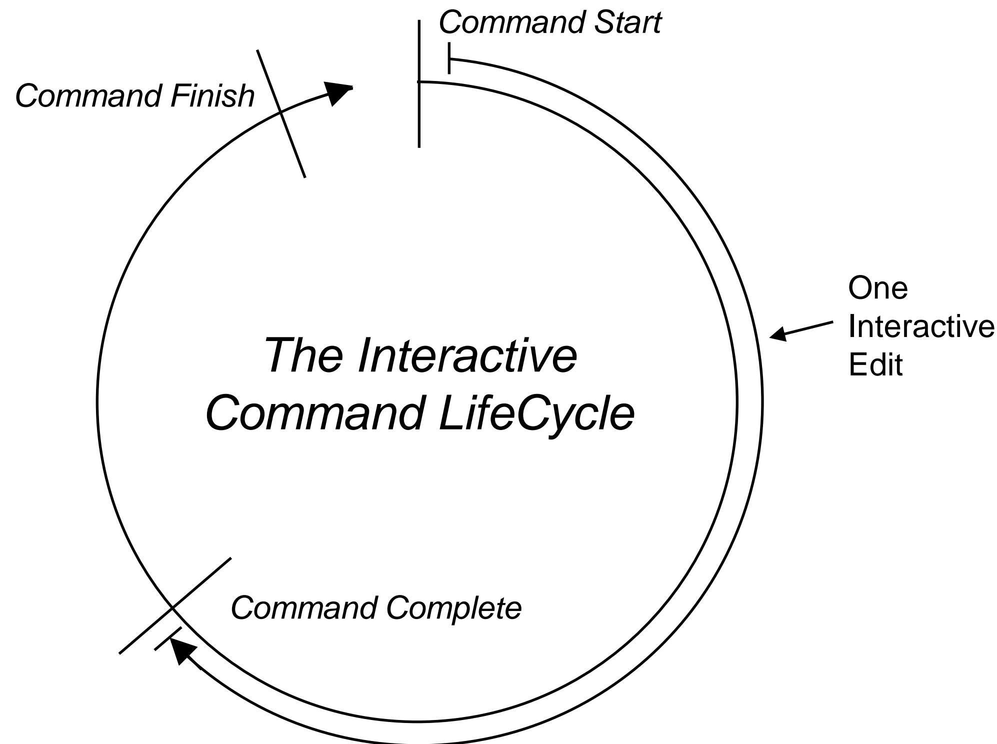
- Compile-time registration of events
- Run-time registration of C++ callbacks for those events
- Run-time registration of TCL callbacks for published events (db::createCallback)
- Events can be posted from C++ or Tcl (db::sendEvent; 2009.0x)
- Event Service – listens for registered events and triggering the appropriate callbacks when they are received.
- OA integration – listens to oaObserver events and generates the appropriate events
- Events are implemented for many window framework, design editor, database transaction, command management, data management, etc., events of significance.

This is a powerful mechanism for customizing Custom Designer behavior and integrating 3rd-party functionality (e.g., Version Control).

©Synopsys 2008

SYNOPSYS® Predictable Success

SYNOPSYS®
Predictable Success



Callbacks registered for postCommandStart, postCommandComplete and preCommandFinish events allow the behavior of any interactive command to be modified. Example: Autoabutment.

Custom User Interfaces with Qt and Opal

- Traditionally GUI access is restricted to
  - Proprietary technology
  - Limited scripting capabilities
  - Or a separate process
- Custom Designer also enables
  - UI development directly in Qt
  - Opal, powerful easy-to-use Tcl-based scripting
- What if you wanted to build this?

©Synopsys 2008

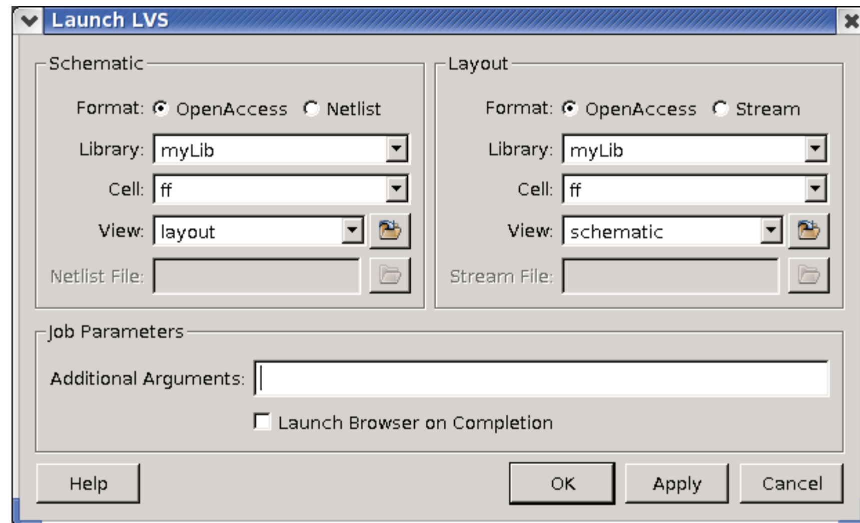
UI Development using the Qt toolkit

- Don’t try doing that in any vendors proprietary language
- It’s natively written in C++/Qt
- With an ability to load C++ shared objects
  - You can create GUIs like this
  - And load them into a platform like Custom Designer

SYNOPSYS®
Predictable Success

Opal Advantages

- Automatic, HIG-compliant layout
- Tcl API
- High-level widgets (i.e., *dmCellViewInput*)
- Preference integration
- Logging & replay
- Valid state
- “Required” inputs
  - OK/Apply disabled if at least one required input is invalid



SYNOPSYS®
Predictable Success

Scripting Languages: The only way to extend?

- Most platforms allow access via a scripting language
  - Tcl
  - SKILL
  - All low performance
  - Try writing a DRC check for real time use
  - Try writing an editor
  - They're simply too slow
- OpenAccess provides a C++ high performance API
  - Can be used to access the same design data
  - At the performance needed for complex tools
    - Routers
    - DRC engines for proprietary rules

SYNOPSYS® Predictable Success

Scripting Languages: The only way to extend?

- Custom Designer lets you do much much more
  - Allows you to create Tcl scripts
  - And also load shared objects (C++ .so)
  - Enabling you to write truly high performance applications
    - Written natively in OpenAccess C++
    - Portable
      - Separate binaries in other tools
      - Loaded directly into our platform!

Design Editor

Layout, schematic and symbol view types are a few of a virtually unlimited set of design representations which also includes netlist and textual views such as Verilog and HSpice. The Design Editor infrastructure provides a common, generic, abstract, non-graphical interface for editors of any design representation.

With access to the abstract design_editor base class, anyone can write an editor for their view type that implements all necessary DE interfaces, e.g.:

- **<constructor>(file)**
- **open(read_only, depth)**
- **close()**
- **activate()**
- **deactivate()**
- **save()**
- **save_as(cellview)**
- **revert()**

And associate it with that view type. DE manages the edit context and provides registry and factory services.

©Synopsys 2008

SYNOPSYS®
Predictable Success

Summary

Custom Designer exposes all fundamental aspects of its application architecture to customization and extension by users, CAD teams and 3rd-party vendors.

Access is provided in both Tcl and C++.

This is totally new. No other EDA platform offers this level of openness, customizability and extensibility.

You can:

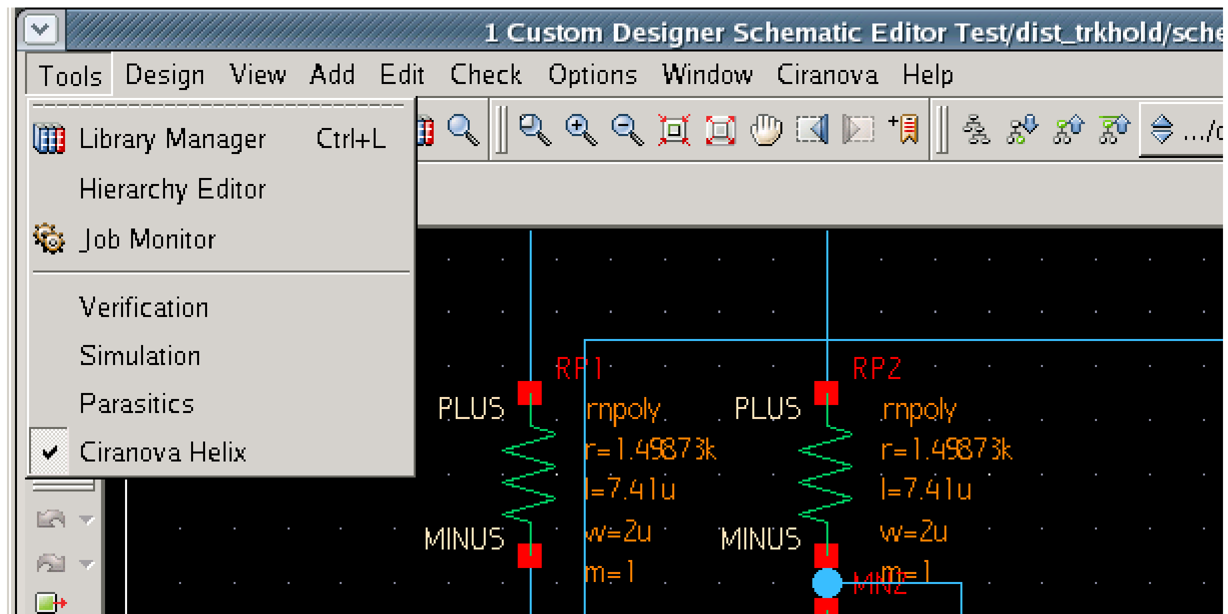
- Customize the look and feel of the tool to meet your users expectations
- Author your own procedural and interactive commands
- Mix in your own interfaces as Tool add-ons
- Customize the behavior of Custom Design through callbacks to platform Events
- Create your own custom UIs in Qt or with Opal
- Even add or replace an entire editor by associating your own Design Editor

A Fully-Coded Real World Example:

Integrating the CiraNova Helix Analog Placer into Custom Designer

Step 1: Register Helix as a DE Tool

```
# Create and register the tool
set t [de::createTool cnHelixTool -title "Ciranova Helix" \
        -initProc cn::helix::internal::init \
        -decorateProc cn::helix::internal::decorate \
        -deactivateProc cn::helix::internal::deactivate]
de::addTool $t -association seSchematic
gi::addActions cnHelixTool -to [gi::getMenus seSchToolsMenu]
```

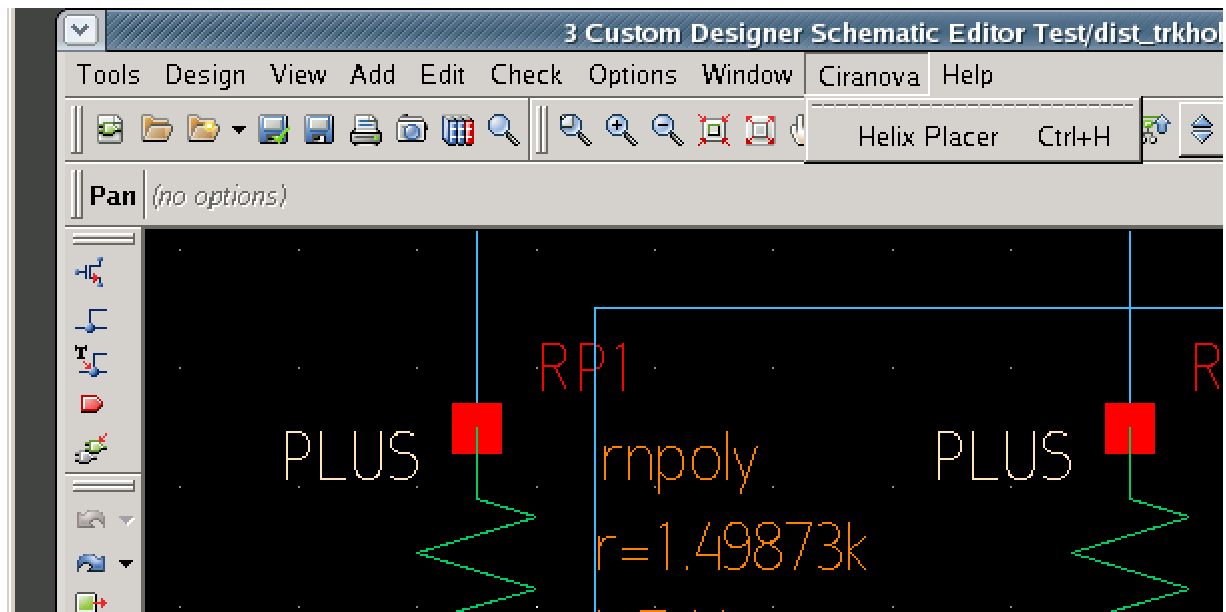


©Synopsys 2008

SYNOPSYS®
Predictable Success

Step 2: Define the Window Decorations

```
set m [gi::createMenu cnMenu -title "Ciranova"]
set a [gi::createAction cnHelix \
        -title "Helix Placer" \
        -command { cn::showHelixSetup -parent %w } \
        -prompt "Configure and launch the Helix placer"]
gi::addActions $a -to $m
gi::createBinding -event Ctrl-h -action cnHelix -activeTool cnHelixTool
```

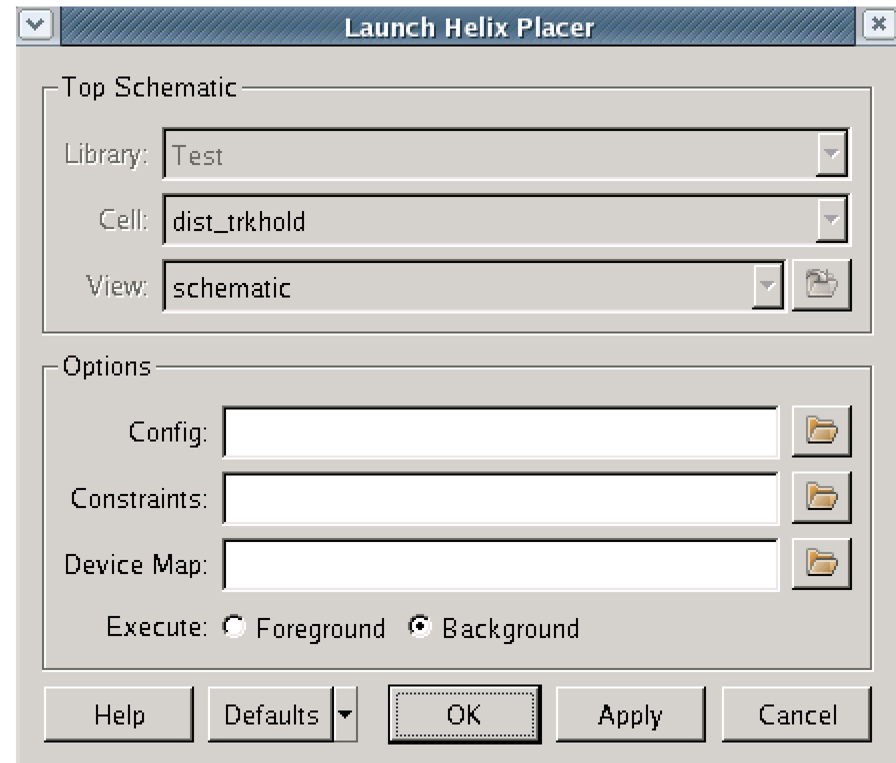


©Synopsys 2008

SYNOPSYS® Predictable Success

Step 3: Create the UI using Opal

```
set d [gi::createDialog cnLaunchHelix -parent $window  -title "Launch Helix Placer"  \
        -prefScope [db::getScopes $cv]  -execProc ::cn::showHelixSetup::okApply]

set g [gi::createGroup -parent $d -label "Top Schematic"
dm::createCellViewInput cellView -parent $g  \
        -value $cv -enabled false  -requireExisting false

set g [gi::createGroup -parent $d -label Options]

gi::createFileInput config -parent $g -label Config \
        -fileMasks {"Helix Config (*.hxcfg)"} \
        -prompt "Select a Helix Config Header" \
        -required true  -prefName cnConfigFile

gi::createFileInput constraints -parent $g -label Constraints \
        -fileMasks {"Helix Constraints (*.hxcs)"} \
        -prompt "Select a Helix Constraints File" \
        -required true  -prefName cnConstraintsFile

gi::createFileInput deviceMap -parent $g -label "Device Map" \
        -fileMasks {"Helix Device Map (*.map)"} \
        -prompt "Select a Helix Device Map File" \
        -required true -prefName cnDeviceMap

gi::createMutexInput execute -parent $g -label Execute \
        -enum {Foreground Background} -value Background  -prefName cnExecMethod
```



SYNOPSYS® Predictable Success

Step 4: A Command to Launch the UI

In Step 2, we created the cnHelix action, and added it to the cnMenu. With our Opal code in hand, we can now create that command:

```
de::createCommand cn::showHelixSetup \
        -description "Show the Helix Setup dialog"  -category "Ciranova" \
        -arguments [list  [de::createArgument "-parent" -description "Parent editor window" \
            -types {giWindow int}  -optional true  -default -1]]

namespace eval ::cn::showHelixSetup {
    proc execute {args} {
            array set argv $args
            set window [getParent $argv(-parent)]
            gi::setActiveWindow $window
            set cv [db::getAttr container -of [db::getAttr editFile -of \
                        [de::getContexts -window $window]]]

            # CODE TO CREATE THE OPAL UI GOES HERE
    }
}
```


SYNOPSYS® Predictable Success

Step 5: Instrument OK and Apply

Next we need an execProc for the Opal UI, which is invoked when OK or Apply is pressed, and launches our execute command:

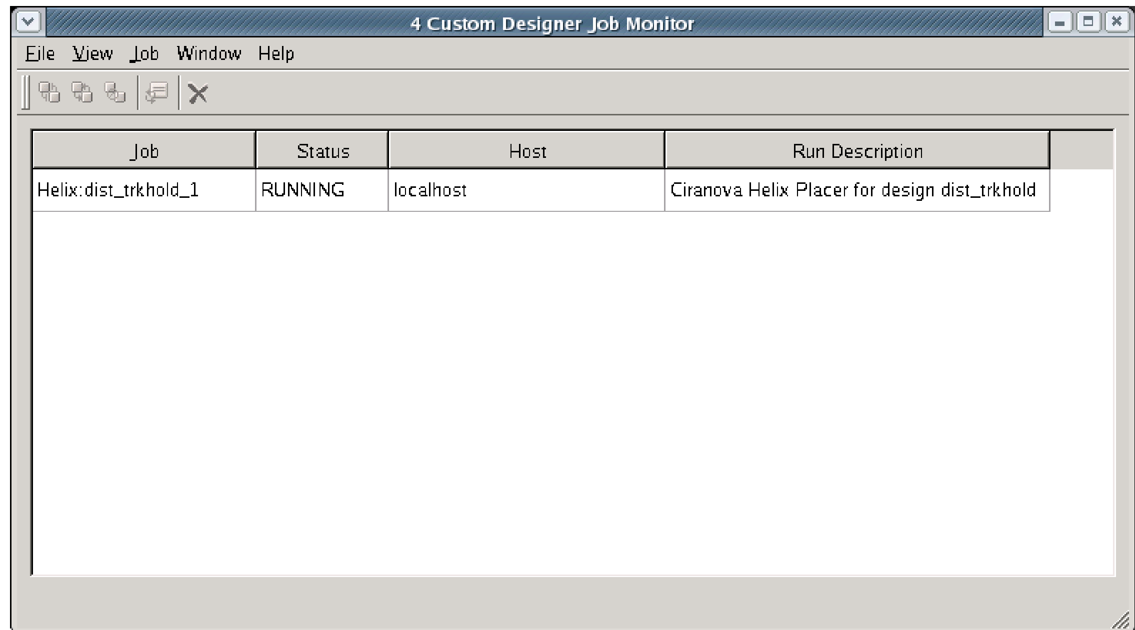
```
proc okApply {d} {
        if {[gi::findChild execute.value -in $d] == "Foreground"} {
                set batch false
        } else {
                set batch true
        }
        cn::executeHelix \
                -topDesign [gi::findChild cellView.value -in $d] \
                -constraintsFile \
                        [gi::findChild constraints.value -in $d] \
                -deviceMapFile [gi::findChild deviceMap.value -in $d] \
                -configFile [gi::findChild config.value -in $d] \
                -batch $batch
}
```


SYNOPSYS® Predictable Success

Step 6: Launching the Placer

cn::executeHelix is defined using de::createCommand. Depending on the batch mode, it may launch an external job:

```
xt::createJob "Helix:$design" -type batch -data $dir
        -cmdLine "sh \"$dir/run.sh\" >\"$dir/log\" 2>&1" \
         -runDesc "Ciranova Helix Placer for design $design" \
         -exitProc ::cn::executeHelix::onComplete \
         -files $dir/log
```



SYNOPSYS® Predictable Success

Step 6: Launching the Placer, continued

Or, it can execute the Placer within the Custom Designer process by invoking code dynamically linked into the process using the Tcl ‘load’ command:

```
# Load dialog and internal execution, defined in C++
namespace eval ::cn::helix::internal {
    set f [lindex [db::resolve helix/helix.so] end]
    if [expr ![file readable $f]] {
        error "Unable to locate helix.so"
    }
    load $f
}
```

Which exports a Tcl command that can be invoked by the cn::executeHelix command which was executed by the okApply proc:

```
// C++ load logic:

extern "C" {
    int Helix_Init(Tcl_Interp *interp);
    }

    int
    Helix_Init(Tcl_Interp *interp) {
        Tcl_CreateCommand(interp, "::cn::executeHelix::execInternal",
                helix::exec_internal, 0, 0);
        return TCL_OK;
    }
```


SYNOPSYS®
Predictable Success

Placement Results

Tools Design View Create Edit Options Verification Window Help

Distance X: 0000.000, Y: 0000.000 DX: 0000.000, DY: 0000.000 Dist. 0000.000 | Selection: 0 full | ConGroup: Default | DRC: off

dbShowFindReplace .../dist_trkhold/...

(No Command) History:

Object/Layer Panel

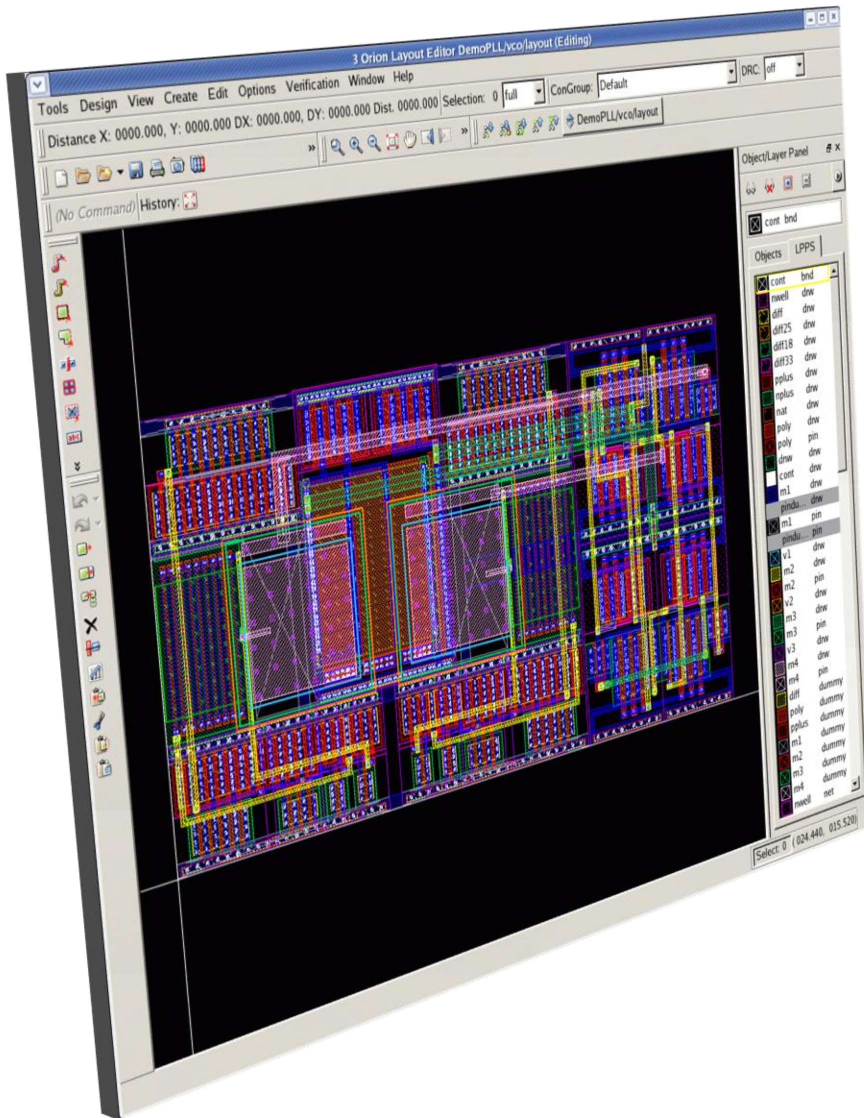
NWELL drawing

Objects LPPS

NWELL drawing
DIFF drawing
THGATE drawing
POLY drawing
POLY pin
PPLUS drawing
NPLUS drawing
SAB drawing
CONT drawing
METAL1 drawing
METAL1 pin
VIA12 drawing
METAL2 drawing
METAL2 pin
VIA23 drawing
METAL3 drawing
METAL3 pin
VIA34 drawing
METAL4 drawing
METAL4 pin
VIA45 drawing
METAL5 drawing
METAL5 pin
VIA56 drawing
METAL6 drawing
METAL6 pin

(325.235, 301.395)

SYNOPSYS® Predictable Success



Thank you!

©Synopsys 2008

SYNOPSYS®
Predictable Success