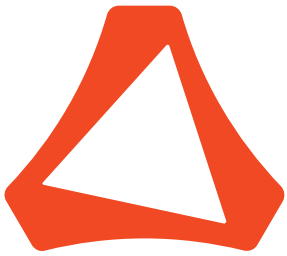

ALTAIR

ONLY FORWARD

Altair EDA 2024.1.0

Automation Tutorials

Contents

EDA Automation Tutorial

1

This chapter covers the following:

The purpose of this tutorial is to illustrate how FlowTracer can be used in the context of a complex chip design. First we guide you through a demonstration, then we analyze the components in detail.

Our assumptions are:

- The chip we are designing is called "gigalite"
- We are using CVS for revision control
- We are emphasizing the browser interface as opposed to the programmable Command Line Interface (CLI) or the dynamic Graphical User Interface (GUI)
- We expect multiple designers to be involved in the design. Each designer has a private workspace. All information exchange among the designers goes through the CVS vault.
- The design flow is based upon a set of (fictitious) tools with name begins with "cdt" (Chip Design Tool). For example, to perform the synthesis of a block called "mmu" we will invoke the command:

  ```
  % cdt_synth mmu
  ```

- The structure of the chip is described in a file called `$VOVDIR/eda/EDADEMO/chipStruct.tcl`

EDA Demo Part 1: Run the Demo

Setup

You should have already setup your account using `vovsetupuser`.

1. Run `vovcheck` to ensure that there are no problems with your installation:

   ```
   % vovcheck
   ```

2. Set up the environment BASE so that you have `cvs` in your path:

   a) Switch to the environment BASE.

   ```
   % ves BASE
   ```

   b) Check to ensure that CVS is in the path.

   ```
   % cvs
   Usage: cvs [cvs-options] command [command-options-and-arguments]
   ...
   ```

   c) If CVS is not available, get the package as follows. You will need to have sudo or root access on your Linus box.

   ```
   % sudo yum install cvs
   ```

   Assuming your Linux distribution is RedHat release:

   ```
   % cat /etc/redhat-release
   ```

   CentOS Linux release 7.7.1908 (Core).

   If you Linux distribution is another version, please consult your manual on how to access the tool.

3. If CVS is not available, access the package as shown below. You need to have sudo or root access on your Linux box to complete this step.

   ```
   % sudo yum install cvs
   ```

   Assuming your Linux distribution is RedHat release (CentOS Linux release 7.7.1908 (Core)).

   ```
   % cat /etc/redhat-release
   ```

   If your Linux distro is something else, then please consult your manual as how to get the tool.

Start the Project

1. Create a directory to run the demo.

   ```
   % mkdir ~/EDADEMO
   ```

```
% cd ~/EDADEMO
```

2. Create a directory for the project administration and another one to do hold the design data.

```
% mkdir ftadmin
% mkdir work
```

3. Create and start the project called `edademo`. We want to do this in the environment BASE to make sure that the vovserver has access to cvs:

```
% ves BASE
% vovproject create -dir ftadmin -type edademo edademo
% vovproject enable edademo
```

Customize the Project

Because the browser interface for this demo is dependent on a CGI script called `edademo.cgi`, we want to make sure that all the HTML pages generated by the vovserver contain a link back to the CGI pages.

We can do this by controlling the value of the EXTLINKS property.

1. Complete the command below to control the value of the EXTLINKS property.

```
% vovprop set -text 1 EXTLINKS "EDADEMO /cgi/edademo.cgi"
```

2. The tasker list file needs to be customized as well. Open it at `ftadmin/edademo.swd/taskers.tcl`.
3. The default list specifies many taskers on the local host. While this is probably ok for the purposes of the demo, you are encouraged to replace the default list with a list of real machines, as in the following example:

```
# Example of taskers.tcl file.
vtk_tasker_set_default -resources "@STD@ @RAMTOTAL@"

vtk_tasker_define hyppo -CPUS 4
vtk_tasker_define rat   -CPUS 1
vtk_tasker_define  cat   -CPUS 1
```

Check Out the Data

1. Make sure you are in the context of the project `edademo` using `vovproject enable` and switch to the environment BASE+EDADEMO using `ves`.

```
% ves BASE+EDADEMO
```

2. Create a workspace for integration and check-out the data from the CVS repository:

```
% export CVSROOT=$VOVDIR/eda/EDADEMO/cvsVault
% cd ~/EDADEMO/work
% mkdir integration
% cd integration
% cvs co gigalite
% export CVSROOT=$VOVDIR/eda/EDADEMO/cvsVault
```

3. Build the flow for integration. The flow we are using is `$EDADEMO/flows/BlockFlow.tcl`:

```
% vovbuild -f $EDADEMO/flows/BlockFlow.tcl
```

Here are the above steps, using a designer called "Mary":

1. If necessary, setup a correct project context and a proper environment:

```
% vovproject enable edademo
% ves BASE+EDADEMO
```

2. Create a workspace for the user and check-out the data from the CVS repository:

```
% cd ~/EDADEMO/work
% mkdir mary
% cd mary
% cvs co gigalite
```

3. Here we assume that this designer does not need to work on the entire chip, but rather on just a few units.

```
% mkdir local
% cp $VOVDIR/eda/EDADEMO/chipStruct.tcl local/chipStruct.tcl
% vi  local/chipStruct.tcl

# Example of a local/chipStruct.tcl file for an individual designer
set listOfUnits {
    adder  rtl
    alu    rtl
    cpu    rtl
    mmu    rtl
}
```

4. Build the flow for this user:

```
% vovbuild -f $EDADEMO/flows/BlockFlow.tcl
```

5. Repeat the steps for other users as necessary.

Start the Browser Interface

1. Use `vovbrowser` to get the URL of the project

```
% vovbrowser
http://your_computer_name:6407
```

Here you can see a few examples of what to expect when using the browser interface. The first picture shows the summary for all workspaces in the project, in this case 'integration' and 'mary'. The second image shows the report for the workspace for user 'mary'.

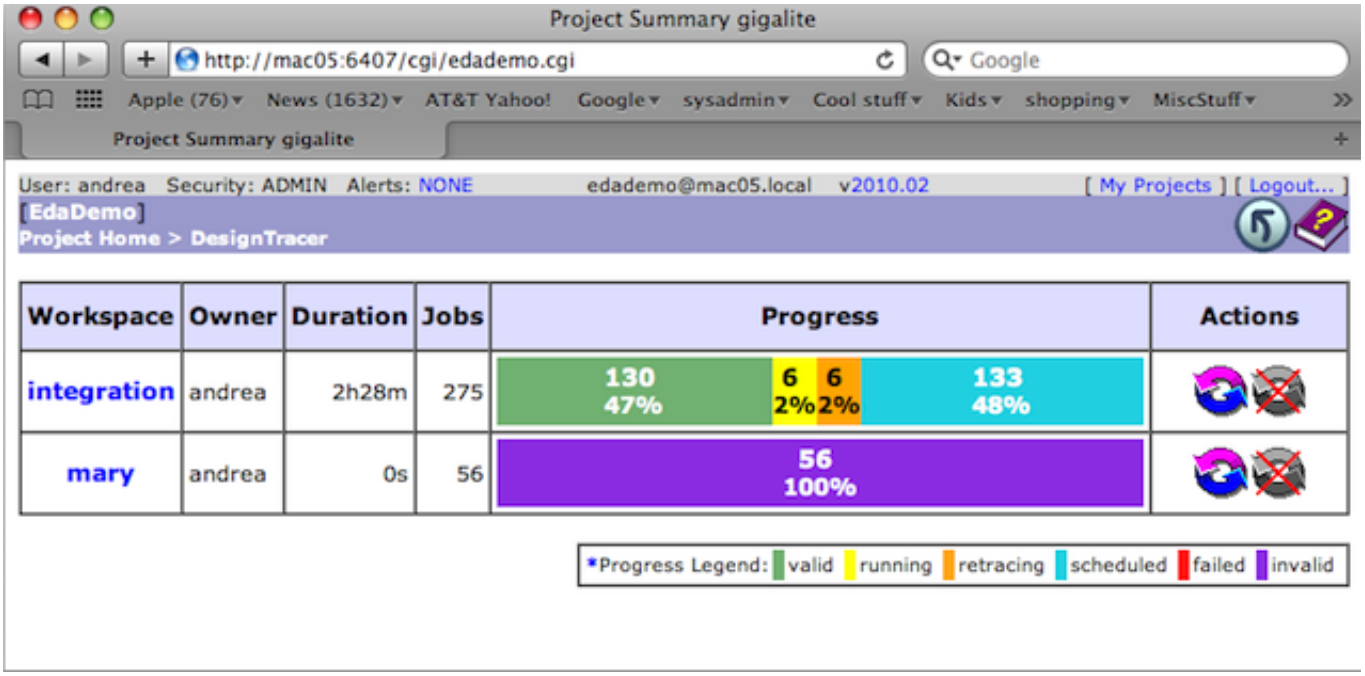

Figure 1: The summary page shows the status of the workspaces

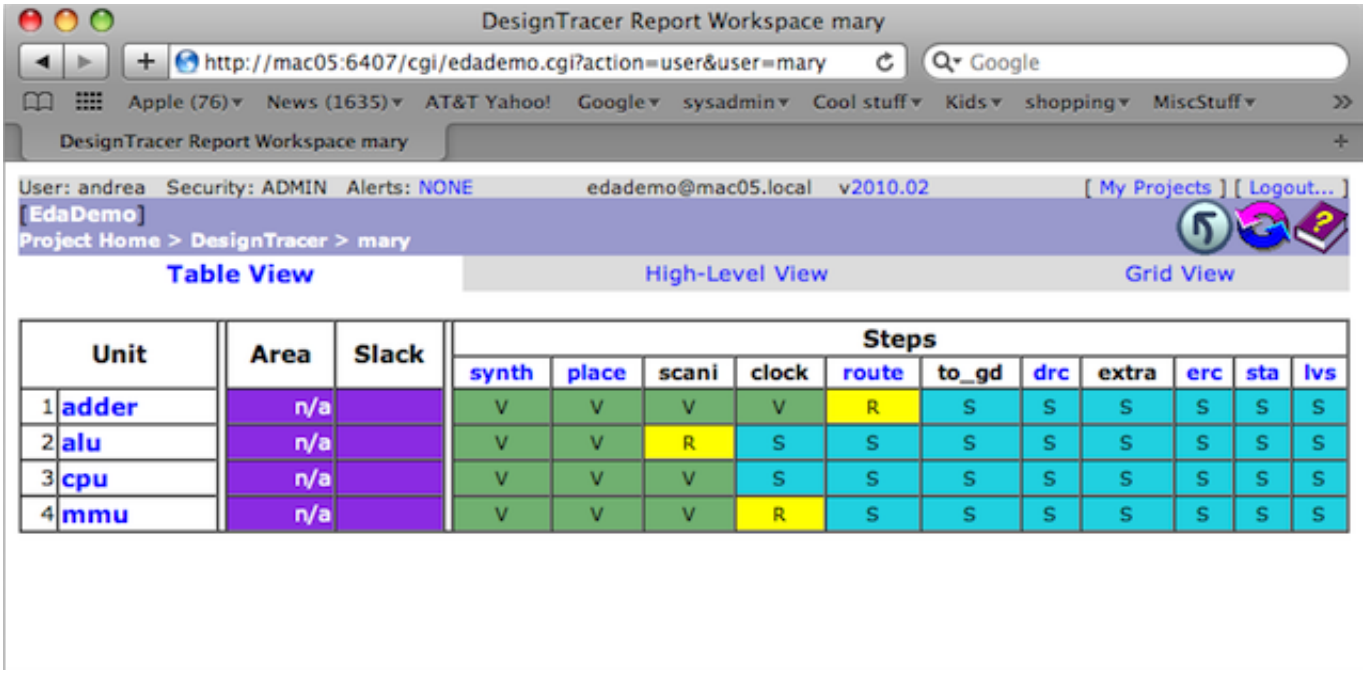

Figure 2: The user page in this EDA-Demo shows a matrix that represents the status of jobs in a specific workspace. There is a row for each block in the design and a column for each of the important jobs in the flow. The color of the cell shows the status of the job.

Here you can see what to expect if you use the VOV Console to run this demo. In particular, you can see the graph view for user 'integration' and the Hi-Level Flows.

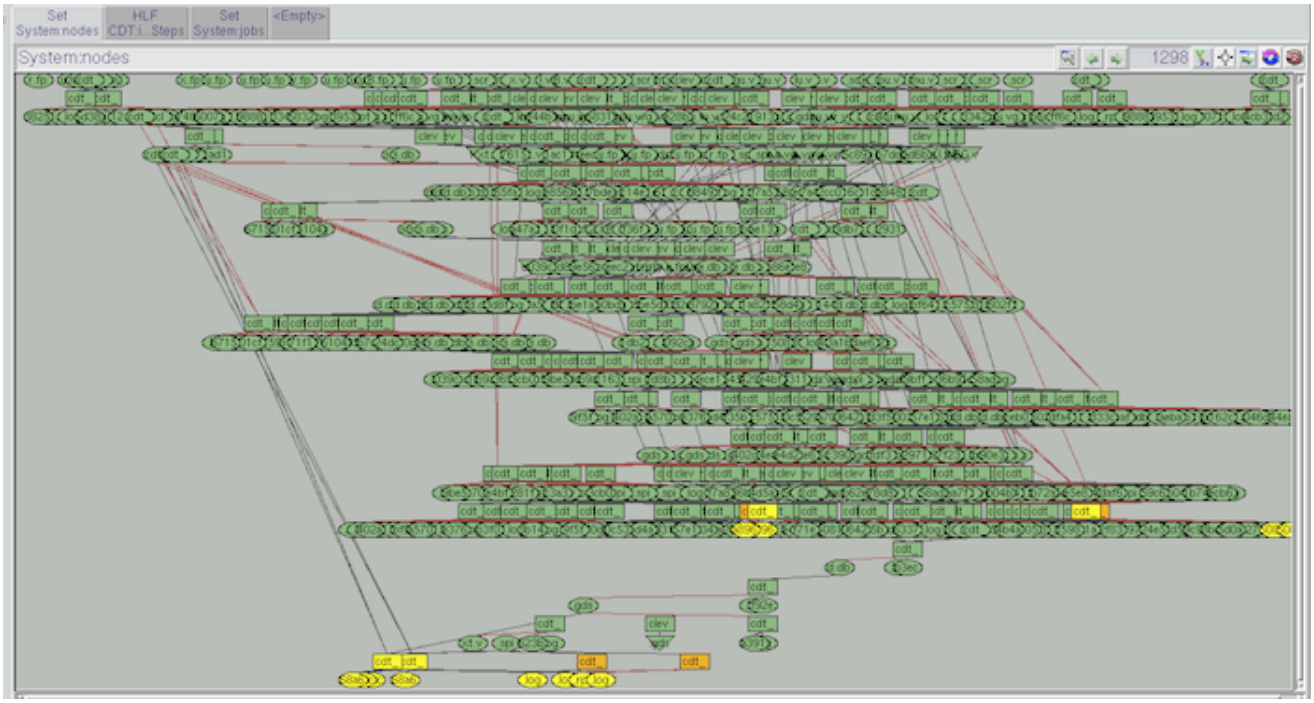

Figure 3: The console shows the complexity of this flow

The Hi-Level Flow representation uses big blocks to represent large sets of jobs. Each block can be clicked on to expose the status of the individual jobs in the set. A little histogram on the bottom left of each block shows the status distribution of the jobs in the set.

2. From this point on, you should be able to:

a) Navigate the browser interface

b) Run either parts of the flow, or the whole flow

c) Stop running jobs

d) Monitor the execution of jobs

e) Diagnose the failure of jobs

f) Edi input files, check them into CVS, and check out any version of a file.

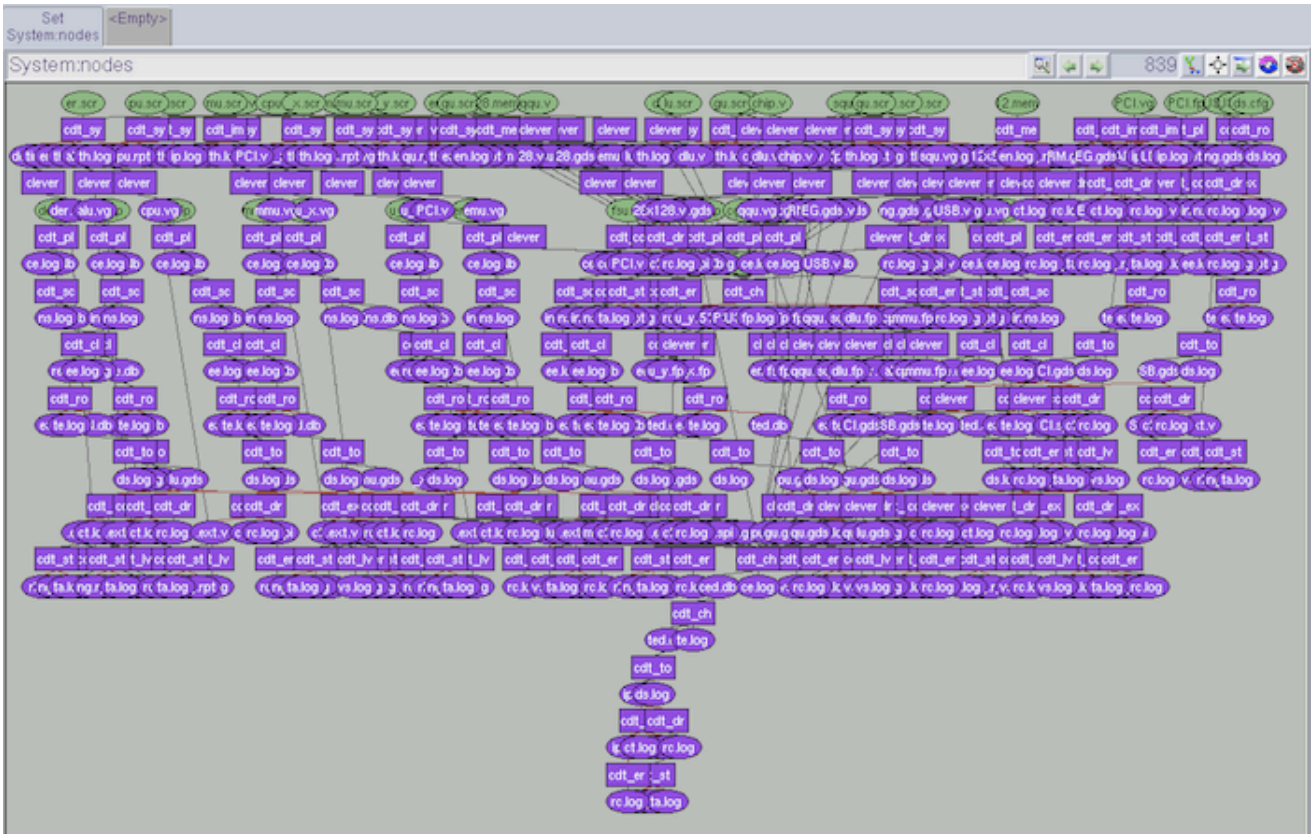

Figure 4:

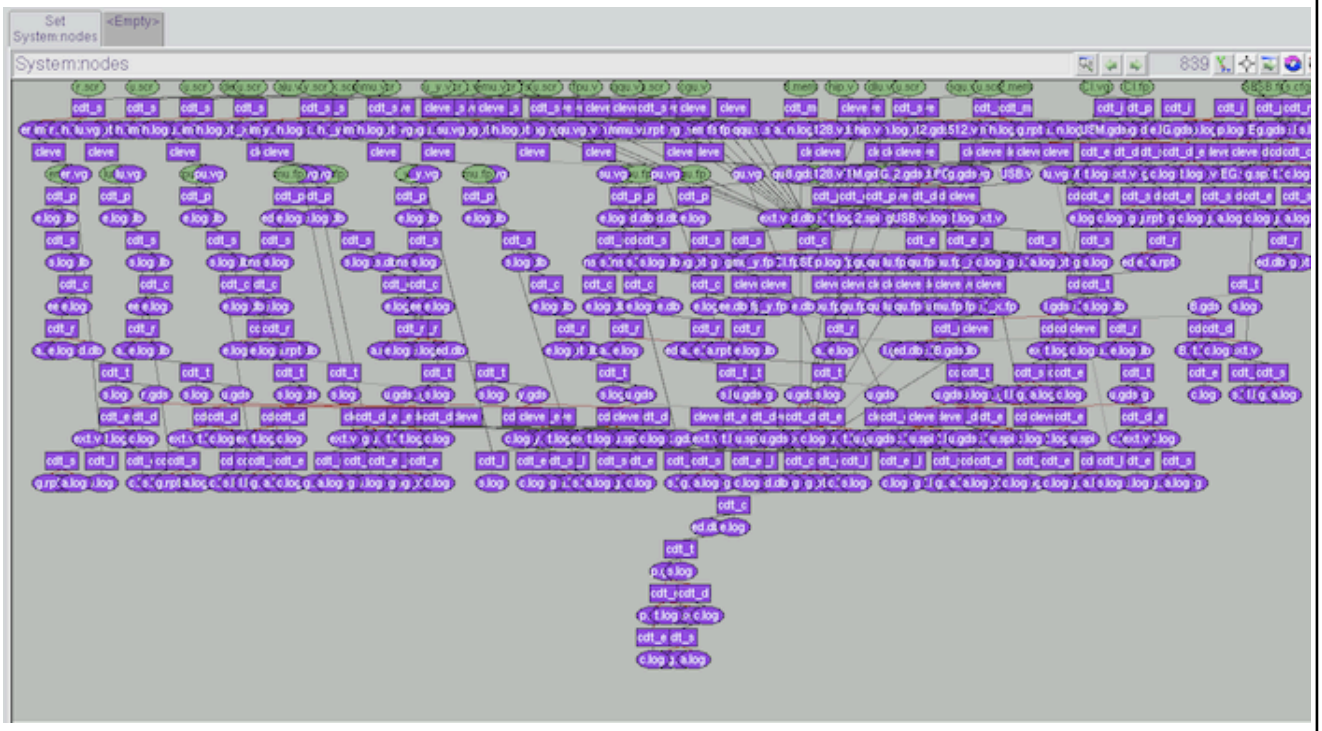

Figure 5:

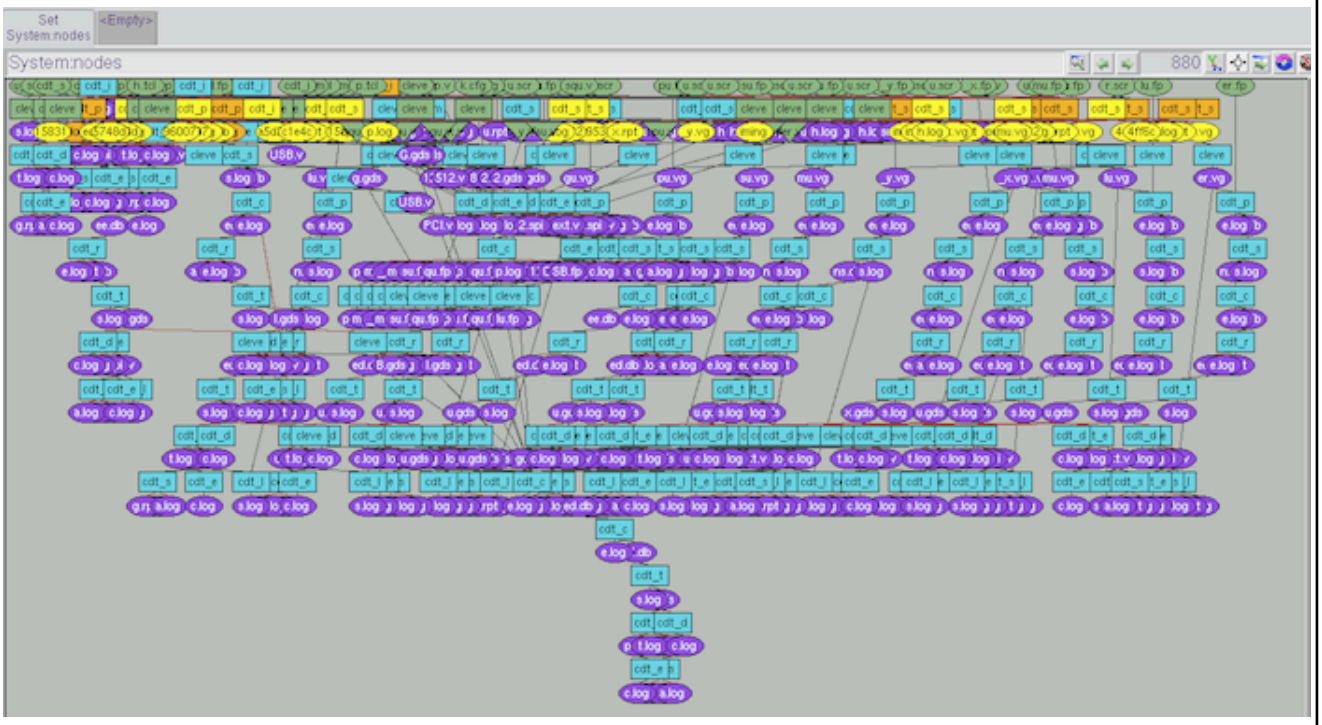

Figure 6:

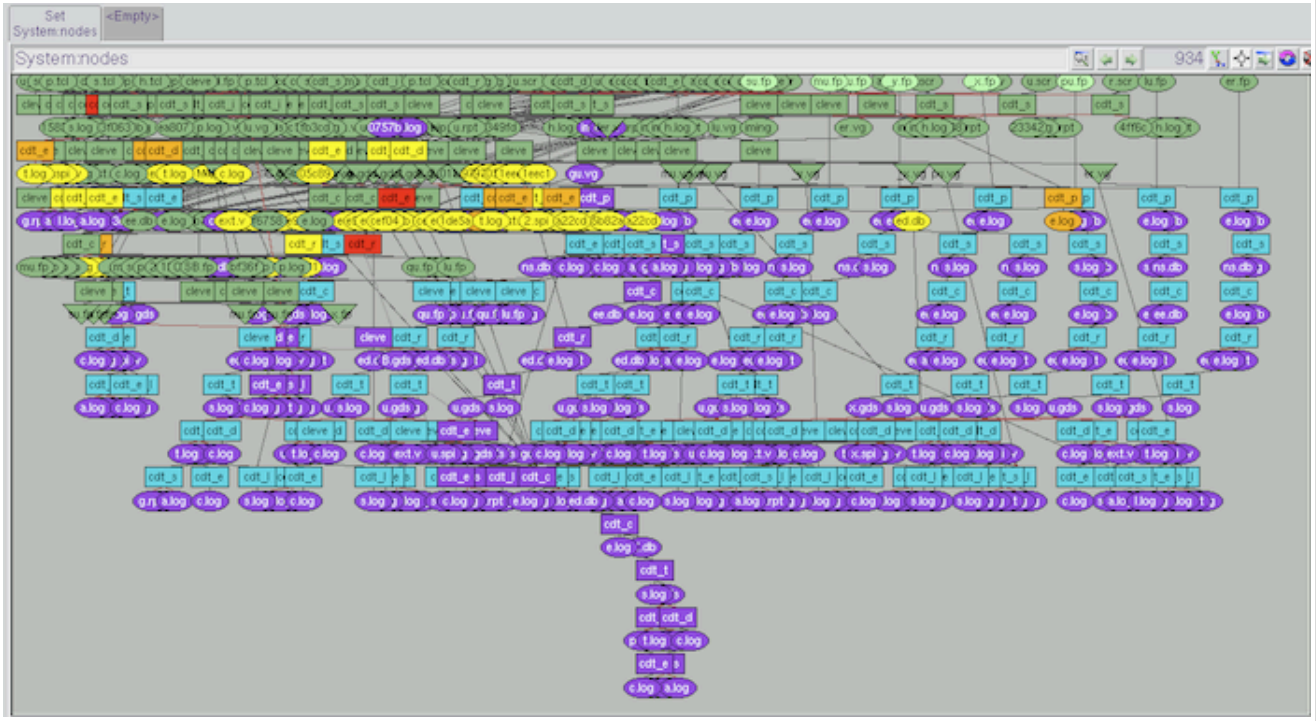

Figure 7:

Figure 8:

EDA Demo Part 2: Dissect the Demo

The Chip Structure File and cdt Script

Each design organization has its own way of describing the structure of a chip. For the demo, we have chosen to describe not so much the structure of the chip as much as the type of each block in the chip. We have RTL block, which need to be sythesized, we have memoy blocks, and both soft and hard IP cores.

`chipstruct.tcl`

```
set listOfUnits {
    adder  rtl
    alu    rtl
    cpu    rtl
    mmu    rtl
    mmu_x  rtl
    mmu_y  rtl
    emu    rtl
    fsu    rtl
    dlu    rtl

    mem12x128 memory
    mem32x512 memory
    PCI       softip
    USB       softip
    ring      pads
    chip      toplevel
}
```

In a similar way, each design organization has its own set of scripts to invoke the steps in the design flow.

The Capsules

We need a capsule for each tool. Basically, a capsule is just a script that figures out the inputs and outputs of an invocation of the tool. In the following example, we show the capsule for the tool `cdt_synth`. The file name of a capsule should be in the form of `vov_tool_name.tcl`. In this case, the capsule file for `cdt_synth` is named `vov_cdt_synth.tcl`.

The usage of this tool is:

```
% cdt_synth unit
```

It takes two files as inputs:

- rtl/*unit*.v
- rtl/*unit*.scr

and it generates four output files:

- results/*unit*.vg
- results/*unit*.timing

- results/*unit*.rpt
- results/*unit*.synth.log

So the capsule is as simple as the following:

`vov_cdt_synth.tcl`

```
set unit [shift]

VovInput  rtl/$unit.v
VovInput  rtl/$unit.scr

VovOutput results/$unit.vg
VovOutput results/$unit.timing
VovOutput results/$unit.rpt

VovOutput logs/$unit.synth.log
```

The Flow Description BlockFlow.tcl

This flow creates all jobs for each block (here called `unit`) described in the `chipStruct.tcl` file, taking into account the type of the block. The FDL procedure `S` is used to define sets of jobs, which are later used to generate CGI reports.

BlockFlow.tcl: The Detailed Flow

```
set PROJECT $env(PROJECT)
set USER    [file tail [pwd]]

source $env(EDADEMO)/chipStruct.tcl
if [file exists local/chipStruct.tcl] {
    VovMessage "Sourcing local/chipStruct.tcl"
    source local/chipStruct.tcl
}

S "CDT:$USER" {
    foreach { unit type } $listOfUnits {
        E EDADEMO
        lappend allUnits $unit
        set types($unit) $type
        S "CDT:$USER:unit:$unit" {
            indir -create $PROJECT/units/$unit {
                file mkdir netlists

                switch $type {
                    "rtl"  {
                        indir -create synthesis {
                            file mkdir results
                            J vw cdt synth $unit
                            J clevercopy results/$unit.vg ../netlists
                            J clevercopy rtl/$unit.v ../../../data/rtl
                        }
                    }
                }
                switch $type {
                    "rtl" - "softip" {
                        indir -create place {
                            J vw cdt place $unit
                            J vw cdt scanins $unit
```

```
                        }
                        indir -create route {
                            J vw cdt route $unit
                            J vw cdt clocktree $unit
                            J vw cdt to_gds $unit
                            J clevercopy ../gds/$unit.gds ../../../data/gds/$unit.gds
                        }
                        indir -create verify {
                            J vw cdt lvs $unit
                        }
                    }
                    "toplevel" {
                        omitted
                    }
                }

                omitted

                # All units.
                indir -create verify {
                    J vw cdt extract $unit
                    J vw cdt sta $unit
                    J vw cdt drc $unit
                    J vw cdt erc $unit
                }
            }
        }
    }
}
```

BlockFlow.tcl: High Level Flow Definition

```
######## Hi-Level Flow Section ########################

H "CDT:$USER:Steps" {
    foreach step $listOfImportantSteps {
        set B$step [B "$step" SETNAME "CDT:$USER:step:$step"]
    }

    C $Bsynth $Bplace
    C $Bplace $Broute
    C $Broute $Bsta
    C $Broute [list $Bdrc $Berc]
    C $Bdrc   $Blvs

    C autoplace
}

H "CDT:$USER:units" {
    set x   100
    set y   100
    foreach { unit type } $listOfUnits {
        set this [B $unit SETNAME "CDT:$USER:unit:$unit" -xy $x $y]
        if { [incr x 100] > 300 } {
            incr y  50
            set x 100
        }
    }
}
```

The CGI script edademo.cgi

This is the most complex piece of the demo. It is also the least necessary, given that we have already covered all pieces necessary to efficietly completing a FlowTracer based design using the CLI and the GUI.

Let's start from the main part of the code, which looks quite similar to the CGI scripts that we have already seen in the CGI tutorial. In bold, you can notice the parsing of QUERY_STRING, which is needed to determine which page to display, based on the values of the "user" and "action" variables.

edademo.cgi: Main code

```
#!/bin/csh -f
# The rest is -*- Tcl -*- \
exec vovsh -t $0 $*

## Definition of all procedures

################################## MAIN CODE

VOVHTML_START

set listOfUnits {}
set listOfSteps { synth place route sta drc lvs }
set listOfUsers {}
set results()   "Initialization of results array"

set opt(reload) 0
set opt(user)   ""
set opt(mode)   ""
set opt(action) "summary"

foreach option [split $env(QUERY_STRING) &] {
    if [regexp {(.*)=(.*)} $option all var value] {
        set var [string tolower $var]
        set opt($var) [url_decode $value]
    }
}

switch $opt(action) {
    "job"    {
        showJobReport  $opt(id)
    }
    "user"   {
        getResultsJobs   $opt(user)
        getResultsArea   $opt(user)
        getResultsTiming $opt(user)
        showUserReport $opt(user) $opt(mode) $opt(reload)
    }
    default {
        showProjectSummary  $opt(reload)
    }
}

VOVHTML_FINISH
```

getResultsArea

In the procedure `getResultsArea` we search the trace database for files that contain area information, which in our case are files with a suffix `area.rpt`. We create a temporary set of all such files (with `vtk_set_create`), and then we get all

elements in the set (with `vtk_set_get_elements`). We look inside each of the files (with source `$namex`) and store the area information in the results array, taking care of flagging whether the data up-to-date or not, by looking at the status (@STATUS@) of the report file. Finally, we cleanup by forgetting the temporary set.

edademo.cgi: collect data from report files:

```
proc getResultsArea { userToReport } {
    global results
    set setName "@@@:tmp:grarp[pid]"
    set setRule "isfile name~/$userToReport/.*area.rpt$"
    set setId   [vtk_set_create $setName $setRule]
    foreach fileInfo [vtk_set_get_elements $setId "@STATUS@ @NAME@"] {
        set status [shift fileInfo]
        set name   [shift fileInfo]
        set namex  [vtk_path_expand $name]
        if [file exists $namex] {
            set unit [file root [file root [file tail  $namex]]]
            catch {source $namex}
            set results($unit,area,status) $status
        }
    }
    vtk_set_forget $setId
}
```

getResultsJobs

In the procedure `getResultsJobs` we query the trace database for information about the status, duration, etc. about each of the CDT jobs in the flow. We store the information in the results array.

First we get a list of all the sets in the trace (with `vtk_trace_list_sets`). We are interested only in the sets with name beginning with "CDT:". We get the elements of each set (with `vtk_set_find` and `vtk_set_get_elements`).

edademo.cgi: collect data from trace

```
proc getResultsJobs { userToReport } {
    global results
    global listOfUnits
    global listOfUsers

    foreach setName [vtk_trace_list_sets] {
    # puts '$setName'
    if [regexp {^CDT:(.*):unit:([^:]+)$} $setName all user unit] {
        lappend_no_dup listOfUnits $unit
        lappend_no_dup listOfUsers $user

        if { $user != $userToReport } continue

        set setId [vtk_set_find $setName]
        set results($unit,setId) $setId
        foreach jobInfo [vtk_set_get_elements $setId "@ID@ @STATUS@ @COMMAND@"] {
            set id   [shift jobInfo]
            set st   [shift jobInfo]
            set tool [lindex $jobInfo 1]

            if  {  $tool  == "cdt" } {
                set step [lindex $jobInfo 2]
                set results($unit,$step,id) $id
                set results($unit,$step,st) $st
            }
        }
```

```
    }
    if [regexp {^CDT:(.*):step:([^:]+)$} $setName all user step] {
        if { $user != $userToReport } continue
            set setId [vtk_set_find $setName]
            set results(step,$step,setId) $setId
        }
    }

    set listOfUnits [lsort $listOfUnits]
    set listOfUsers [lsort $listOfUsers]
}
```

Page Layout

To give all pages a similar look and functionality, we wrote procedure EDADEMOPAGE, which takes the following arguments:

- *title*, which is the title of the page
- *reloadMs*, the value in milliseconds of the auto-reload period
- *menu_script*, a script to customize the menu on the left hand side of the page
- *body_script*, a script with the actual content of the page

The scripts are evaluated using the Tcl command uplevel.

`edademo.cgi:` The page layout

```
proc EDADEMOPAGE { title reloadMs menu_script body_script } {
    global env
    HTML {
        HEAD { omitted }
        BODY {
            TABLE CELLSPACING=0 CELLPADDING=6 BORDER=0 WIDTH="100%"  {
                TR BGCOLOR="white" {
                    omitted title code
                }
                TR {
                    TD BGCOLOR="$bgColor" VALIGN=TOP {
                        HREF "/cgi/edademo.cgi" "Summary"; BR
                        if { $menu_script != {} } {
                            uplevel $menu_script
                        }
                    }

                    TD VALIGN=TOP COLSPAN=2 BGCOLOR="#BBCCBB" {
                        uplevel $body_script
                    }
                }
            }
        }
    }
}
```

Following is a collection of procedure used to render the data in HTML. Note the use of `vtk_time_pp` to print durations in a human readable form and of the global `array vov_color()` to colorize the cells in the tables according to the status of the node they represent.

edademo.cgi: Data rendering

```
proc showStepInfo { unit step displayMode } {
    global results
```

```
    global vov_color

    set st [getResult $unit,$step,st EMPTY]
    if { $st == "EMPTY" } {
        TD {}
        TD {}
        TD {}
        return
    }
    set du [getResult $unit,$step,du n/a]
    set id [getResult $unit,$step,id 0 ]
    set ag [getResult $unit,$step,ag 0 ]

    set url  "/cgi/edademo.cgi?action=job&id=$id"
    if { $du >= 0 } {
        set tim  [vtk_time_pp $du]
    } else {
        set tim  [vtk_time_pp $ag]
    }

    if { $displayMode == "simple" } {
        TD ALIGN="right" COLSPAN="2"  { SMALL { HREF $url $tim   }  }
        TD BGCOLOR=$vov_color($st) { OUT "  " }
    } else {
        TD ALIGN="right"  { SMALL { HREF $url $tim   }  }
        TD ALIGN="center" { SMALL { HREF $url [string tolower $st]   }  }
        TD BGCOLOR=$vov_color($st) { OUT "  " }
    }
}

proc showResults  { user { displayMode "simple"} } {
    global results
    global vov_color
    global listOfUnits
    global listOfSteps

    TABLE BORDER="0" CELLSPACING="2"  CELLPADDING="5" {
        TR BGCOLOR="white" {
            TH COLSPAN=2 { OUT "Unit" }
            TH { OUT "Area"  }
            TH { OUT "Slack" }
            foreach step $listOfSteps {
                set setId [getResult step,$step,setId 0]
                TH COLSPAN=3 {
                    if { $setId == 0 } {
                        OUT "$step"
                    } else {
                        HREF "/set?id=$setId&action=showgrid" $step
                        if { $displayMode != "simple" } {
                            BR
                            SMALL {
                                omitted: some links
                            }
                        }
                    }
                }
            }
        }

        set count 0
        foreach unit $listOfUnits {
            set setName "CDT:$user:unit:$unit"
            set setId   [vtk_set_find $setName]
```

```
            if { $setId != 0 } {
                set area    [getResult $unit,area        "n/a"]
                set areaSt  [getResult $unit,area,status "INVALID"]
                set slack   [getResult $unit,slack ""]
                set slackSt [getResult $unit,slack,status "INVALID"]
                TR {
                    TD ALIGN="RIGHT" {  OUT [incr count] }
                    TH ALIGN="LEFT" { HREF "/set?id=$setId" $unit }
                    TD ALIGN=RIGHT BGCOLOR=$vov_color($areaSt) {
                        COLOR $vov_color($areaSt,fg) $area
                    }
                    TH ALIGN=RIGHT BGCOLOR=$vov_color($slackSt) {
                        if { $slack != "" } {
                            if { $slack %lt; 0 } {
                                COLOR "#FFFF88" "($slack)"
                            } else {
                                COLOR $vov_color($slackSt,fg) $slack
                            }
                        }
                    }
                    foreach step $listOfSteps {
                        showStepInfo $unit $step $displayMode
                    }
                }
            }
        }
    }
}

proc showUserSummary { user setId }  {
    global vov_color

    if { $setId == 0 } {
        set status    EMPTY
        set nodes     0
        set places    0
    } else {
        set setInfo [vtk_set_statistics $setId]
        set saveSetInfo $setInfo
        set status     [shift setInfo]
        set nodes      [shift setInfo]
        set places     [shift setInfo]
        set jobs       [shift setInfo]
        set duration   [shift setInfo]
        set unknown    [shift setInfo]
        set placeStats [shift setInfo]
        set jobStats   [shift setInfo]

        set statusList { INVALID RUNNING VALID FAILED }
        foreach s $statusList { set js($s) 0 }
        foreach { s n } $jobStats {
            set js($s) $n
        }

        TR {
            TH { print user name   }
            TD ALIGN="RIGHT" { OUT [vtk_time_pp $duration] }
            TD ALIGN="RIGHT" { OUT $jobs }
            foreach s $statusList {
                omitted: show colored ball
            }
        }
    }
```

```
}
```

showProjectSummary

The `showProjectSummary` procedure displays the summary page. First we get the list of users of the project (really the list of workspaces) by looking at the sets with name beginning with "CDT:", and then we call `showUserSummary` on each.

edademo.cgi: The project summary page

```
proc showProjectSummary { reloadMs } {
    set listOfUsers {}

    foreach setName [vtk_trace_list_sets] {
        if [regexp {CDT:(.*):step} $setName all user] {
            lappend_no_dup listOfUsers $user
        }
    }
    set listOfUsers [lsort $listOfUsers]

    EDADEMOPAGE "Project Summary  $env(PROJECT)" $reloadMs {
        omitted
    } {
        TABLE WIDTH="100%" border=0 CELLPADDING=10 {
            TR BGCOLOR="#DDDDFF" {
                foreach head {
                    Workspace Duration Jobs Invalid
                    Running Valid Failed Actions
                } {
                    TH { OUT $head }
                }
                foreach user $listOfUsers {
                    showUserSummary $user [vtk_set_find "CDT:$user"]
                }
            }
            TR BGCOLOR="#DDDDFF" {
                TD COLSPAN=9 { OUT "Totals" }
            }

            showUserSummary ""  [vtk_set_find "System:nodes"]
        }
    }
}
```

showUserReport

The procedure `showUserReport` generates the page showing the jobs in a user workspace. Notice the use of EDADEMOPAGE and showResults, which have been explained above.

edademo.cgi: The user workspace report

```
proc showUserReport { user mode reloadMs } {
    global argv

    EDADEMOPAGE "EDA-Demo Report Workspace $user" $reloadMs {
        if { $mode != "hlf" && $mode != "llf" } {
            omitted: autoreload control
        }
        showMenu $user $mode
    }  {
        switch $mode {
```

```
            "llf"   { showLowLevelFlow "$user"     }
            default { showResults $use      }
        }
    }
}
```

Legal Notices

Intellectual Property Rights Notice

Copyrights, trademarks, trade secrets, patents and third party software licenses.

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions. In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair Simulation Products

Altair® AcuSolve® ©1997-2023

Altair Activate®©1989-2023

Altair® Battery Designer™ ©2019-2023

Altair Compose®©2007-2023

Altair® ConnectMe™ ©2014-2023

Altair® EDEM™ © 2005-2023

Altair® ElectroFlo™ ©1992-2023

Altair Embed® ©1989-2023

Altair Embed® SE ©1989-2023

Altair Embed®/Digital Power Designer ©2012-2023

Altair Embed® Viewer ©1996-2023

Altair® ESAComp® ©1992-2023

Altair® Feko® ©1999-2023

Altair® Flow Simulator™ ©2016-2023

Altair® Flux® ©1983-2023

Altair® FluxMotor® ©2017-2023

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2023

Altair® HyperLife® ©1990-2023

Altair Packaged Solution Offerings (PSOs)

Altair HPC & Cloud Products

Third Party Software Licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

For a complete list of Altair Accelerator Third Party Software Licenses, please click here.

Technical Support

Altair provides comprehensive software support via web FAQs, tutorials, training classes, telephone and e-mail.

Altair One Customer Portal

Altair One (https://altairone.com/) is Altair's customer portal giving you access to product downloads, Knowledge Base and customer support. We strongly recommend that all users create an Altair One account and use it as their primary means of requesting technical support.

Once your customer portal account is set up, you can directly get to your support page via this link: www.altair.com/customer-support/.

Altair Training Classes

Altair training courses provide a hands-on introduction to our products, focusing on overall functionality. Courses are conducted at our main and regional offices or at your facility. If you are interested in training at your facility, please contact your account manager for more details. If you do not know who your account manager is, e-mail your local support office and your account manager will contact you

Telephone and E-mail

If you are unable to contact Altair support via the customer portal, you may reach out to the technical support desk via phone or e-mail. You can use the following table as a reference to locate the support office for your region.

When contacting Altair support, please specify the product and version number you are using along with a detailed description of the problem. It is beneficial for the support engineer to know what type of workstation, operating system, RAM, and graphics board you have, so please include that in your communication.

Location	Telephone	E-mail
Australia	+61 3 9866 5557 +61 4 1486 0829	anz-pbssupport@altair.com
China	+86 21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr

Location	Telephone	E-mail
Malaysia	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0) 46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
United Kingdom	+44 (0)1926 468 600	pbssupport@europe.altair.com

See www.altair.com for complete information on Altair, our team and our products.

Index

E