docs-ee Documentation

Release

gradrat.com

January 18, 2017

CHAPTER 1

About

1.1 Mission

The mission of this documentation project is to provide better resources for the practical skills that are often the bottleneck of getting research (and work) done for graduate students in the EE field.

It is not uncommon for students to spend several years getting trained by their advisors in some fundamental skill sets before becoming productive. Often times the domain knowledge of senior students departs when they graduate, leaving advisors with the task of training a new set of students.

The goal of this project is to retain all of the 'common' knowledge typically required for such research and document it in a way that will hopefully accellerate students' ability to become productive and successful.

1.2 Contact

- Issues: https://github.com/gradrat/docs-ee/issues
- contact[at]gradrat[dot]com

CHAPTER 2

Chip Design

2.1 Chip Tapeout Design Flow

The contents here describe recommended procedures/requirements/guidelines that should be followed for taping out a chip.

2.1.1 Major Milestones

The major milestones for each project may differ slightly based on the goals of the tapeout, but the following list should be a basis to start from.

Architecture Design Review

- Choose architecture to address design needs/goals
- Provide high-level system simulation results to indicate goals are achievable
- Generate system and block-level specifications
- Document architecture, assumptions, and requirements

Schematic/RTL Design Review

This should happen for each block in the system. This milestone will not have been reached until every block (or an encapsulating block) in the chip passes a design review.

- Generate schematics or RTL to implement block-level specifications
- Simulation results show required performance or functionality is being met
- Document design including decisions, assumptions and key results

Post Layout Design Review

All requirements for the the Schematic/RTL review should be repeated for the layout design. Additionally, the following requirements should be met.

- Generate LVS/DRC clean layout
- Review of any special layout requirements (metalization, matching, power, signal access)

- Confirmation that the design meets timing and edge rate requirements

Top Level Verification

Once the entire chip design has been constructed (even if only as models), a suite of regression tests should begin being developed. These tests should continually be revised/improved and run as the design progresses.

2.1.2 Design Reviews

The guideline for things to review for each block can be found in the digital (Digital_Design_Review) and analog (Analog_Design_Review) design sections.

2.1.3 Final Tapeout Checklist

Clocks

- Manual/visual inspection of clock networks on synthesized clocks
- Are clocks shielded from/against other sensitive lines?
- Check to make sure that all supplies for any clock buffers are present when you need a clock –> simulate power up and check that clocks are present.
- Check fan-out of any custom clock networks

Resets

- Check loading/timing/dependencies of all asynchronous resets

Interfaces

- Have all interfaces between blocks & outside of the chip been simulated?
- At the chip level, there should at least be behavioral level simulations of all interfaces
- Check timing between custom and synthesized blocks
- Check the supply voltages at each interface. If they have different supply voltages, make sure there are appropriate level shifting circuits.

Power Supplies

- Simulate ramping of power supplies
  - Check reset timings
  - Check for leakage currents (floating inputs)
  - Check for clocks
  - Check for latch-up
- Run DC simulation of entire chip
  - Check bias points, leakage currents

- Check the amount and location of decap on each supply rail

Layout

- Check DRC
  - All required DRC rules should pass
  - Any waived DRC rules should be marked in a DRC waiver list and noted as to why the DRC rule is waived
- Check LVS
  - Pin labels at the top should be on pads/bumps
  - LVS should pass with NO virtual connections
  - All components that you care about should be LVS'd (e.g. if you have an inductor, or some other custom device, you should LVS it!)
  - Look for any floating guard rings (sealring is okay)
- Check Metal Integrity
  - Check that any high current nets (power, ground, high power inputs/outputs) have maximally connected metal. You can use a Via_Finder and Via_Insertion tool for this purpose.
  - Check the resistivity of any critical nets that must carry high current using a resistance visualization tool (e.g. R3D, P2P, etc...)

I/O and ESD

- Are all outputs sized to drive the anticipated board/output load?
- Are all inputs designed to handle the expected voltage ranges?
- Do all inputs have some kind of weak pull-up/down to insure their state?
- Is there an ESD path for all I/Os? (See ESD_Guidelines)

2.1.4 Final Tapeout Procedure

After checking all of the pre-tapeout checklist items we are ready to send the final GDS to the foundry.

1. Stream out the layout design to GDS. If there are additional non-silicon layers (e.g. RDL), make sure to alter the layermap file to remove these layers (since the gds layers that they map to may collide with internal foundry layers).
2. Stream in the output GDS into a new library and DO NOT include any reference libraries during the stream in.
3. Run LVS using the streamed-in library layout and the schematic we had for the top level chip.
4. Run DRC on the streamed-in library layout. The errors should match, or be similar (depending on the type of errors).
5. Tab the revision control database
6. Freeze/disable access to the revision control database
7. Freeze/check-in the versions of any dependencies (cad/ versions, tools)
8. Tar and zip the gds database
9. Run a checksum on the *.tar.gz file

10. Send the database to the foundry
11. Add Info to project module about tool versions used during tapeout

CHAPTER 3

Foreword

This section of documentation describes the mechanics of using each EDA tool, but not **HOW** the tool should be used or applied in design.

CHAPTER 4

EDA Tools

4.1 Cadence Tools

4.1.1 Installation

Getting source files

- Download the *.sdp download configuration files and the Installscape tool which downloads the actual source files from Cadence
- Unpack the Installscape tool along with all of the *.sdp files in one directory
- Before using Installscape, you need to install the appropriate library support (it is/was a 32-bit tool):

```
>> yum install glibc.i686 libXext.i686 libXtst.i686
```

- glibc.i686 - ld-linux.so.2
- libXext.i686 - libXext.so.6
- libXtst.i686 - libXtst.so.6

License Installation

- Install the cadence license tools (LCU or under the tools/bin directory of most packages)
- Install the necessary libraries to run the licensing tools:

```
>> yum install libgcc.i686 redhat-lsb-core.i686
```

- If you want, use the "config_lic" tool under the /share/license directory or else just edit the license file so that it has a line like so:

```
SERVER <servername> <Ethernet MAC Address> <port>
```

where, it's MAC address and port to access the licenses at are filled in appropriately.

- Start the license server:

```
>> lmgrd -c <license file>
```

- Set either CDS_LIC_FILE or LM_LICENSE_FILE environment variables to point to @ to gain access to the tools.

Automatically starting the license server on boot

You can also have the license server run automatically by editing the rc.lic (might be rclic.sample) file and having it executed by the /etc/rc.d/rc.local script.

So for example, the rc.lic file would look like (this can be anywhere you choose):

```
#!bash

#!/bin/sh
# Start Cadence license daemons and specify the debug log file
#
# If editing this file manually, be sure to set values for INSTALL_DIR,
# LICENSE_FILE, and optionally for LOG_FILE and LMGRD_OPTS
#
INSTALL_DIR='<Cadence tool root directory where you have INSTALL_DIR/tools/bin/lmgrd>'
LICENSE_FILE='<path to license file>'
LOG_FILE='<path to log file>'
LMGRD_OPTS='<license manager options>'
LOG_DIR='<path to log file>'

echo
echo "Starting Cadence license daemons"
echo

if [ -r ${INSTALL_DIR}/tools/bin/lmgrd ]; then
  rm -f /usr/tmp/lockcdslmd

  if [ -r ${LOG_FILE} ]; then
    time_stamp=`ls -ol ${LOG_FILE} | awk '{printf "%s.%d.%s\n",$5,$6,$7}'`
    mv ${LOG_FILE} ${LOG_FILE}.$time_stamp
    (echo "     Old debug log files in ${LOG_DIR}:")
    ( ls -l ${LOG_FILE}.* )
  fi

  # so boot can be run in the background and log file can be moved
  # while daemons are running

  ( ${INSTALL_DIR}/tools/bin/lmgrd ${LMGRD_OPTS} -c ${LICENSE_FILE} \
    2>&1 ) | ksh -c "while read line; do echo \"\$line\" >> ${LOG_FILE}; done" &
  sleep 2
else
  echo ""
  echo "Cannot locate the license manager daemon (lmgrd)."
  echo "Please verify that the necessary symbolic link exists before proceeding."
  echo "For more information about licensing utilities, see the 'Cadence"
  echo "License Manager' in CDSDoc."
  echo ""
fi
exit 0
```

Then the contents of your /etc/rc.d/rc.local might look like:

```
#!bash

#!/bin/sh
# This script will be executed *after* all the other init scripts

touch /var/lock/subsys/local
```

```
sh ./<path to rc.lic file>
```

Specific Tool Dependencies

For each of the following tools, you'll first need to install the following packages:

Virtuoso (IC)

```
>> yum install libXp xorg-x11-fonts*dpi.noarch
```

ADE

```
>> yum install compat-readline5 libgcc.i686
```

Encounter Digital Implementation (EDI)

```
>> yum install ksh
```

PVS / QRC Extraction

```
>> yum install compat-libtermcap
```

For QRC tech generation:

```
>> yum install libXft.i686 libXmu.i686
```

Other Cadence tools relying on 32-bit packages

You may have to install these packages too depending on your tool

```
>> yum install elfutils elfutils-libelf libXp
>> yum install compat-libstdc++-296
>> yum install libXext.so.6
>> yum install libXt.i686 libXt.x86_64
>> yum install gcc-c++
>> yum install libXp.so.6
>> yum install libXtst.so.6
>> yum install glibc-devel.i686
>> yum install libXcursor.so.1
>> yum install openmotif
>> yum install libelf.so.1
>> yum install libXss.so.1
>> yum install libXft.so.2
>> yum install libGL.so.1
>> yum install libGLU.so.1
>> yum install libXrandr.so.2
>> yum install libXi.i686 ,libSM.i686, elfutils.libefl.i686, libXrender.i686, zlib.i686
```

Troubleshooting

ADE-XL hangs or won't start

If you see that ADE-XL doesn't run (but ADE-L does), you might encounter the (ADE-XL Message 1921) error. If so, it's likely that you didn't install all of the required packages for your OS. To see what you are missing,

1 Go to your run directory, which should look something like: /.tmp_/.cmddir0/

2 Look at the **runICRP** file while the simulation is trying to run. There should be a shell command in that file to the tune of 'virtuoso '. Copy or record that command somewhere.

3 When the simulator finally stops, that file might be gone, but run the recorded command in your shell. This should produce some error log to the screen. Follow the path of the error message (maybe something to do with VNC) to see what libraries are missing

References

- http://alexcooper.co.uk/blog/2013/05/installing-cadence-tools-2012-2013/

4.1.2 Virtuoso

Schematic Usage

Schematic Setup

Customizing Sheet Info When editing a schematic in Cadence you can (should) add in a built-in sheet element border to each schematic view:

```
Edit->Sheet Size...
```

If you want to customize what is displayed, you can change the contents by editing (as the CAD/root user) the **Asize/Bsize** symbol views in the US_8ths library AND the **Title** symbol view.

Some possible labels to add include:

ilInst~>cellView~>cellName This provides the name of the cell that the sheet is instantiated in.

SosGetCurrentVersion(ilInst~>cellView) If you are using Cliosoft, this provides the revision/managed state of the cellview.

ilInst~>cellView~>fileTimeStamp This provides the last date/time that the schematic was edited & saved.

ilInst~>cellView~>createTime This provides the creation date for the schematic.

if(ilInst~>cellView~>modifiedButNotSaved then "True" else "False") Indicates whether the schematic has been modified but not saved.

dbGetPropByName(ilInst~>cellView "lastSchematicExtraction")~>value This returns the date that the schematic was last "checked".

Default Shortcuts

m Move selection

c Copy selection

r Rotate selection

i Create an instance

p Create a pin

e Descend into a selected instance

Ctrl+e Return up one level of hierarchy

9 Highlights a net

Layout Usage

Layout XL Usage

Layout Tips

Marking Nets

- To change how Cadence marks connectivity on nets, you can choose Connectivity–>Nets–>Mark and then hit the F3 button for options
- To preset certain options for yourself, you can modify your .cdsenv file. For example, to choose which layers you want excluded in the connectivity you could add:

```
layout markNetExcludePurposes boolean t
layout _markNetExcludePurposesName string "boundary annotate error label flight warning"
```

- You can also save the mark net options from the options form that appears when you hit F3. If you save this to the tech library, then those options become the default for all libraries that use that tech lib. The file should be saved with the name *markNetOptions* and placed in this path:

```
<tech library directory>/.cadence/dfII/markNet/markNetOptions
```

This will be the default file if there isn't a similar path in the current directory (where virtuoso was launched) or in the user's home directory.

Creating a new multi-part path Multipart paths can be useful for things like creating guard rings or seal rings or any connected structure that requires relative spacings/sizings.

1. Open a layout view
2. Go to Create->Multipart Path
3. Hit the 'F3' key to bring up the Multipart Path options menu
4. Choose a template name to create or edit
5. Click the Subpart... options button to create the multipart path definition

Creating pins from labels Depending on how you normally define pin/net connectivity, you sometimes may want to create pin database objects from text labels so that the layout can be better used in subsequent layouts in tools such as Layout-XL.

- If you are using labels (on layers M1TXT, M2TXT, etc.), go to the **Tools** menu and select **Create Pins from Labels...**.
- A window will pop up with options to put pins on locations of labels.

- **The default settings should be:**
  - Objects Within: Cellview
  - Object Type: Both
  - Objects: All
- If you just have M3TXT labels, then make sure to check the **ignore** checkbox on the right of all the other ones. If you have M2TXT, then make sure to not ignore those, etc.
- The column with **Objects Layer** is where it is looking for the labels. The **Pin Layer** is where it will add a rectangular *pin*.
- For something like M3TXT, you should change the **Pin Layer** to **M3**, etc.
- After pressing **OK**, there will be little metal rectangles at every label. You may have to go to each one to make sure they are all at the appropriate places (and not producing DRC errors).

Creating a P&R Boundary Sometimes you will want to do this if the layout is intended to be used in an automated place & route tool or perhaps for estimating a design size in Layout-XL.

1. Go to the **Create** menu and select **P&R Objects** and then **P&R Boundary**.
2. Point to the two opposite corners of the boundary of your cell and then adjust the shape by chopping, etc, if needed.

Shortcuts

Ctrl+d Unselect all

z Zoom in on area

Ctrl+z Zoom out

Shift+z Zoom in

m Move selection

c Copy selection

s Stretch selection

C Chop selection

p Create a path in the current layer

r Create a rectangle shape in the current layer

4.1.3 Physical Verification System (PVS)

Setting up PVS Menus (LVS/DRC)

It's often desirable to have the options needed to run an LVS, DRC, or extraction (QRC) run to be automatically populated based on the project or technology. This helps to speed up DRC/LVS/Extraction checks as well as avoid problems where designers/users either can't find the appropriate rules or are using the incorrect set of rules.

Getting the PVS menu to appear in a Layout Menu

Manually The PVS menu is not available by default in Cadence. To make it available, you should:

1. Open the layout view of a cell
2. Go to Launch->Plugins->PVS in the cell view's menu
3. Now a PVS menu item should be present in the layout cell view

Automatically To get the PVS menu to appear by default when you open a layout view, you should add the following lines (or similar) to your .cdsinit file.

```
printf( strcat("-- Registering PVS menu triggers... \n"))
deRegUserTriggers("maskLayout" nil '_pvsMaskLayoutMenuTrigger)
deRegUserTriggers("maskLayoutXL" nil '_pvsMaskLayoutMenuTrigger)
```

The first line, is just informational for users. The next two lines launch PVS whenever Layout, or LayoutXL tools are used.

Using DRC/LVS Preset Files

You can either manually fill out the LVS/DRC forms, or you can load the with a set of pre-defined values using preset files.

Creating a new preset file To create/populate a DRC/LVS preset file:

1. Launch the DRC or LVS menu from the PVS menu PVS->Run DRC.. (or PVS->Run LVS..)
2. Fill out the Run directory, inputs, rule files, etc...
3. Save the settings for the run by choosing the File->Save Presets option from the DRC/LVS run menu

Manually Loading a preset file Once you have a presets file, you can simply:

1. Open the DRC or LVS run menu
2. Go to File->Load Presets

The tool is somewhat smart about replacing some of cell specific preset options with the current cell (e.g. input cell, output netlist/files, etc...).

Automatically loading a preset file To have a specific preset file loaded by default (e.g. if several users all want to use the same preset file), simply define the following environment variables:

```
setenv PVSUI_LVS_PRESETS_FILE <path to lvs preset file>
setenv PVSUI_DRC_PRESETS_FILE <path to drc preset file>
```

Customizing DRC/LVS Preset File Inputs

The 'smart' part of a preset file allows a single preset file to be generically used for most cells. However, there are times that you may want to customize the inputs/outputs based on the input cells (e.g. the run directory) or the type of check you are running.

The hierarchy of choosing how the LVS or DRC runs is populated is:

1. Presets file defined in a techRuleSet (highest priority)
2. PVSPreFormTrigger skill code (if a PVSPreFormTrigger procedure is defined)
3. Auto (or manually) loaded Preset file (lowest priority)

Using the PVSPreFormTrigger We can define a PVSPreFormTrigger to customize certain components of the LVS/DRC menu. An example of this procedure is provided below.

```
procedure( PVSPreFormTrigger(runType presetFile winId)
    let(
        (projDir user
        pvsTechLibPath
        runDir runDirExt
        cv cellName libName
        tmpFile f
        )

        cv = winId~>cellView
        user = getShellEnvVar("USER")
        projDir = getShellEnvVar("PROJECT")

        pvsTechLibPath = strcat( projDir "/cad/cds.lib")

        runDirExt = case(runType
                ("LVS" "lvs")
                ("DRC" "drc")
                ("FASTXOR" "fastxor")
                ("XOR" "xor")
                (t 'default)
            ); case

        cellName = cv~>cellName
        libName = cv~>libName
        runDir = strcat("/tmp_ssd/" user "/pvs/" libName "/" cellName "/" runDirExt)

        ; Write out the temporary presets file
        tmpFile = makeTempFileName( strcat(getTempDir() "/pvs"))
        f = outfile(tmpFile)
        fprintf(f "[ pvsgui.Run%s ]\n" runType)
        fprintf(f "RunDir \"%s\" \n" runDir )
        close(f)

        ; Return the name of the presets file
        tmpFile

    ); let
); procedure
```

This procedure re-defines the run directory based on the cell's name and library. If a preset file is loaded in conjunction with this procedure, the "smart" part of the tool automatically changes several other form inputs based on the run directory while keeping things like the rule file and tool options the same.

QRC Menu Setup

Setting up a QRC Techlib The output of the cds_qrc_techgen process is a technology directory(ies) that corresponds to a particular process corner/option. This technology directory should appear as rule or corner when running parasitic

extraction (PVS->Quantus QRC). To do this, you need to provide a path to that directory by defining a corner.defs file in the tech library path which might look like:

```
DEFINE TYP <PATH TO typical corner QRC run directory>
DEFINE SLOW <PATH TO slow corner QRC run directory>
DEFINE FAST <PATH TO fast corner QRC run directory>
```

Note: This corner.defs file must be placed inside of a library that is defined in Cadence (via cds.lib and/or pvtech.lib) by whatever file you specify as the Quantus QRC tech lib in the run menu.

Pre-loading the PVS->QRC menu When we want to run extraction based on a PVS-LVS run, we can use the PVS->QRC flow by launching the tool in the layout tool menu: QRC->Run PVS-Quantus QRC...

This menu can be automatically populated by (re)defining the vuiUserDefinedRCXFormSetupCB skill procedure. An example of that procedure is shown below:

```
procedure( vuiUserDefinedRCXFormSetupCB(form)
    let(
        (projDir user
        pvsTechLibPath pvsTechLibName
        lvsRunDir svdbQRCDir
        cv cellName libName
        )

        cv = geGetWindowRep()
        user = getShellEnvVar("USER")
        projDir = getShellEnvVar("PROJECT")

        pvsTechLibPath = ddGetUpdatedLib()
        pvsTechLibName = "QRC_LIB"

        cellName = cv~>cellName
        libName = cv~>libName
        lvsRunDir = strcat("/tmp_ssd/" user "/pvs/" libName "/" cellName "/lvs")
        svdbQRCDir = strcat( lvsRunDir "/svdb" )

        ; Setting of the preQRCform
         when( form->hiFormSym == 'vuipreRcxForm
             form->runName->value = cellName
             form->runDir->value = svdbQRCDir
             form->assuraTech->value = pvsTechLibPath
             form->tech->value = pvsTechLibName
        ); end of when preQRC

        ; need to return a true for the QRC run to continue
        t

    ); let
); procedure
```

In this example, the cds.lib used to define the libraries is provided as the lib path definition for the tool. It also defines the run directory in the same way as the DRC/LVS example above.

Note: By populating/defining the assuraTech value, we no longer need a pvtech.lib file in the Cadence search path

(usually needs to be where you launched Cadence).

4.1.4 Quantus Extraction System (QRC)

QRC Usage

Techlib Setup

The output of the cds_qrc_techgen process is a technology directory(ies) that corresponds to a particular process corner/option. In order to use the extraction results, this technology directory should appear as rule or corner when running parasitic extraction (PVS->Quantus QRC). To do this, you need to provide a path to that directory by defining a corner.defs file in the tech library path which might look like:

```
DEFINE TYP <PATH TO typical corner QRC run directory>
DEFINE SLOW <PATH TO slow corner QRC run directory>
DEFINE FAST <PATH TO fast corner QRC run directory>
```

Note: This corner.defs file must be placed inside of a library that is defined in Cadence (via cds.lib and/or pvtech.lib) by whatever file you specify as the Quantus QRC tech lib in the run menu.

QRC Menu Setup

Pre-loading the PVS->QRC menu

When we want to run extraction based on a PVS-LVS run, we can use the PVS->QRC flow by launching the tool in the layout tool menu: QRC->Run PVS-Quantus QRC...

This menu can be automatically populated by (re)defining the vuiUserDefinedRCXFormSetupCB skill procedure. An example of that procedure is shown below:

```
procedure( vuiUserDefinedRCXFormSetupCB(form)
    let(
        (projDir user
        pvsTechLibPath pvsTechLibName
        lvsRunDir svdbQRCDir
        cv cellName libName
        )

        cv = geGetWindowRep()
        user = getShellEnvVar("USER")
        projDir = getShellEnvVar("PROJECT")

        pvsTechLibPath = ddGetUpdatedLib()
        pvsTechLibName = "QRC_LIB"

        cellName = cv~>cellName
        libName = cv~>libName
        lvsRunDir = strcat("/tmp_ssd/" user "/pvs/" libName "/" cellName "/lvs")
        svdbQRCDir = strcat( lvsRunDir "/svdb" )

        ; Setting of the preQRCform
         when( form->hiFormSym == 'vuipreRcxForm
```

```
            form->runName->value = cellName
            form->runDir->value = svdbQRCDir
            form->assuraTech->value = pvsTechLibPath
            form->tech->value = pvsTechLibName
        ); end of when preQRC

        ; need to return a true for the QRC run to continue
        t

    ); let
); procedure
```

In this example, the cds.lib used to define the libraries is provided as the lib path definition for the tool. It also defines the run directory in the same way as the DRC/LVS example above.

Note: By populating/defining the assuraTech value, we no longer need a pvtech.lib file in the Cadence search path (usually needs to be where you launched Cadence).
