A Complete Digital Design Flow for SPI Protocol: From RTL to GDSII Using Qflow

A Partial Fulfillment of Mini project is submitted

to

MAVEN SILICON

Submitted by

PRAUDDHA CHOURE

Guide by

MR. ABHISHEK GADED

Course name

VLSI Physical Design Internship

Company Name

MAVEN SILICON

2024-25

Content of Mini Project report:

Introduction:

Digital design involves converting high-level hardware descriptions into physical circuits that can be implemented on silicon or FPGAs. One of the essential stages in this process is RTL (Register Transfer Level) synthesis, where a hardware description written in languages like Verilog or VHDL is transformed into a gate-level netlist. A netlist is a structural representation of a digital circuit in terms of logic gates and their interconnections.

In this report, we describe the complete synthesis flow of an RTL design using Yosys, an open-source synthesis tool, running on the Ubuntu Linux operating system. The purpose is to convert Verilog RTL code into a netlist, which can then be used for further steps such as simulation, placement, routing, or FPGA implementation.

Project 1: Synthesis of RTL code.

RTL (Register Transfer Level) Code:

RTL (Register Transfer Level) is a design abstraction that describes the flow of data between registers and the logical operations performed on that data. It is used in digital circuits to specify how data moves and how computations occur within clock cycles.

At the RTL level:

- You describe the behaviour and structure of digital circuits.
- It includes flip-flops, multiplexers, arithmetic operations, and control logic.
- Written typically in Verilog or VHDL.

RTL code: COUNTER DESIGN

```
module counter (clk, rst, en, count);
input clk, rst, en;
output reg [1:0] count;
always @(posedge clk)
if (rst)
count <= 2’d0;
else if (en)
count <= count + 2’d1;
endmodule
```

What is Synthesis?

Synthesis is the process of converting RTL code into a gate-level netlist. It involves:

- Parsing the HDL code (e.g., Verilog).
- Optimizing logic (removing redundant logic or simplifying expressions).
- Mapping the logic to standard cells or generic gates (like AND, OR, NOT).
- Generating a netlist, which represents how gates and modules are connected.

Output of synthesis:

- Gate-level netlist (a textual description of logic gates and their connections).
- Optional: reports like area, timing, and power (depending on technology).

Yosys: The Open Source Synthesis Tool

Yosys is a powerful open-source tool for RTL synthesis primarily targeting Verilog designs. It supports various backends including gate-level synthesis, formal verification, and netlist generation.

Key Features of Yosys:

- Open-source and free to use.
- Supports Verilog-2005.
- Can generate netlists in EDIF, BLIF, JSON, and other formats.
- Can be integrated with open-source FPGA toolchains (e.g., nextpnr).
- Custom scripting support using TCL or Yosys command language.

Ubuntu Environment

Ubuntu, a Linux-based open-source operating system, provides a stable and flexible environment for hardware design and simulation. Yosys and other EDA tools can be easily installed and run from the command line in Ubuntu, making it ideal for synthesis workflows.

RTL to Netlist Conversion Process using Yosys

Here's the basic step-by-step process followed during synthesis:

Step 1: Install Yosys on Ubuntu

sudo apt update

sudo apt install yosys

Step 2: Make a Project Directory

mkdir Project1/

```
prabuddha-choure@prabuddha-choure-VirtualBox: ~
prabuddha-choure@prabuddha-choure-VirtualBox:~$ mkdir Project1/
prabuddha-choure@prabuddha-choure-VirtualBox:~$
```

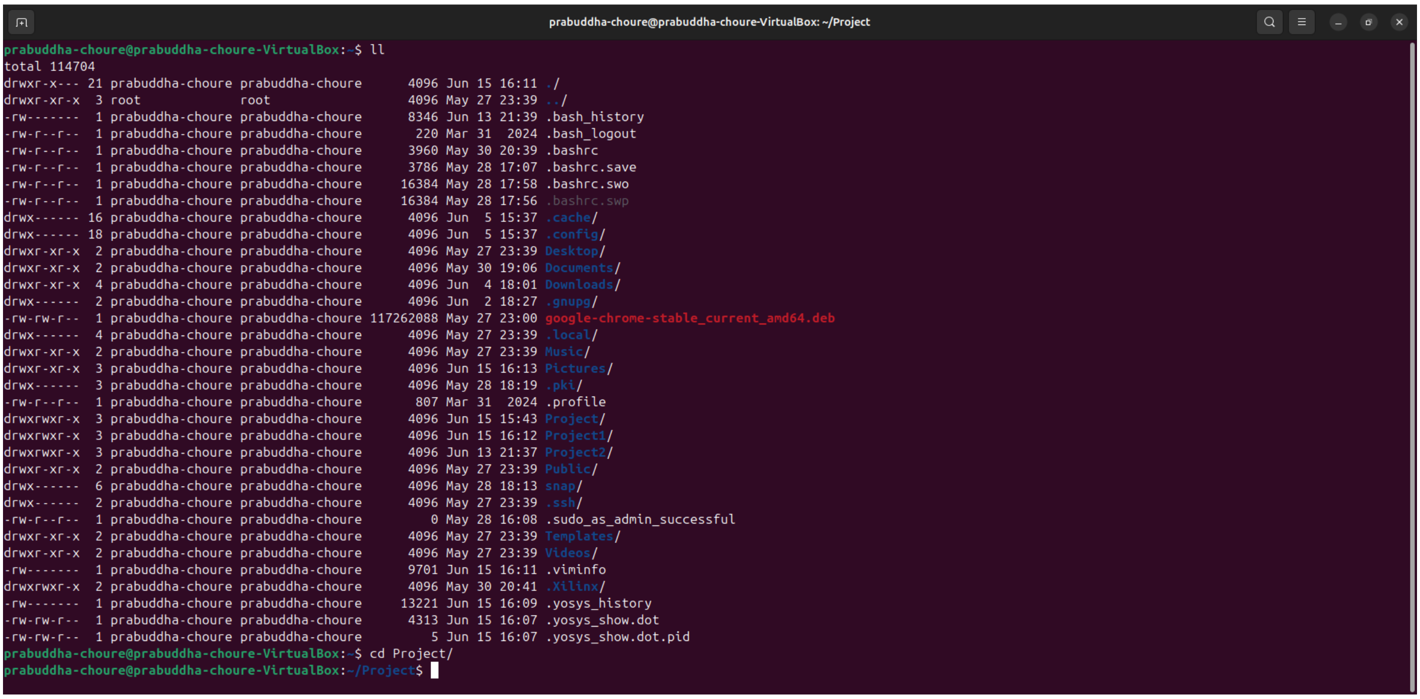
Step 3: Check whether the directory is made or not

ll

```
prabuddha-choure@prabuddha-choure-VirtualBox: ~
prabuddha-choure@prabuddha-choure-VirtualBox:~$ mkdir Project1/
prabuddha-choure@prabuddha-choure-VirtualBox:~$ ll
total 114716
drwxr-x--- 21 prabuddha-choure prabuddha-choure      4096 Jun 15 15:50 ./
drwxr-xr-x  3 root             root                  4096 May 27 23:39 ../
-rw-------  1 prabuddha-choure prabuddha-choure      8346 Jun 13 21:39 .bash_history
-rw-r--r--  1 prabuddha-choure prabuddha-choure       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 prabuddha-choure prabuddha-choure      3960 May 30 20:39 .bashrc
-rw-r--r--  1 prabuddha-choure prabuddha-choure      3786 May 28 17:07 .bashrc.save
-rw-r--r--  1 prabuddha-choure prabuddha-choure     16384 May 28 17:58 .bashrc.swo
-rw-r--r--  1 prabuddha-choure prabuddha-choure     16384 May 28 17:56 .bashrc.swp
drwx------ 16 prabuddha-choure prabuddha-choure      4096 Jun  5 15:37 .cache/
drwx------ 18 prabuddha-choure prabuddha-choure      4096 Jun  5 15:37 .config/
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 Desktop/
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 30 19:06 Documents/
drwxr-xr-x  4 prabuddha-choure prabuddha-choure      4096 Jun  4 18:01 Downloads/
drwx------  2 prabuddha-choure prabuddha-choure      4096 Jun  2 18:27 .gnupg/
-rw-rw-r--  1 prabuddha-choure prabuddha-choure 117262088 May 27 23:00 google-chrome-stable_current_amd64.deb
drwx------  4 prabuddha-choure prabuddha-choure      4096 May 27 23:39 .local/
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 Music/
drwxr-xr-x  3 prabuddha-choure prabuddha-choure      4096 Jun 15 15:44 Pictures/
drwx------  3 prabuddha-choure prabuddha-choure      4096 May 28 18:19 .pki/
-rw-r--r--  1 prabuddha-choure prabuddha-choure       807 Mar 31  2024 .profile
drwxrwxr-x  3 prabuddha-choure prabuddha-choure      4096 Jun 15 15:43 Project/
drwxrwxr-x  2 prabuddha-choure prabuddha-choure      4096 Jun 15 15:50 Project1/
drwxrwxr-x  3 prabuddha-choure prabuddha-choure      4096 Jun 13 21:37 Project2/
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 Public/
drwx------  6 prabuddha-choure prabuddha-choure      4096 May 28 18:13 snap/
drwx------  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 .ssh/
-rw-r--r--  1 prabuddha-choure prabuddha-choure         0 May 28 16:08 .sudo_as_admin_successful
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 Templates/
drwxr-xr-x  2 prabuddha-choure prabuddha-choure      4096 May 27 23:39 Videos/
-rw-------  1 prabuddha-choure prabuddha-choure      9495 Jun 15 15:42 .viminfo
drwxrwxr-x  2 prabuddha-choure prabuddha-choure      4096 May 30 20:41 .Xilinx/
-rw-------  1 prabuddha-choure prabuddha-choure     12972 Jun 15 15:42 .yosys_history
-rw-rw-r--  1 prabuddha-choure prabuddha-choure     19066 Jun 15 15:42 .yosys_show.dot
-rw-rw-r--  1 prabuddha-choure prabuddha-choure         5 Jun 15 15:42 .yosys_show.dot.pid
prabuddha-choure@prabuddha-choure-VirtualBox:~$
```

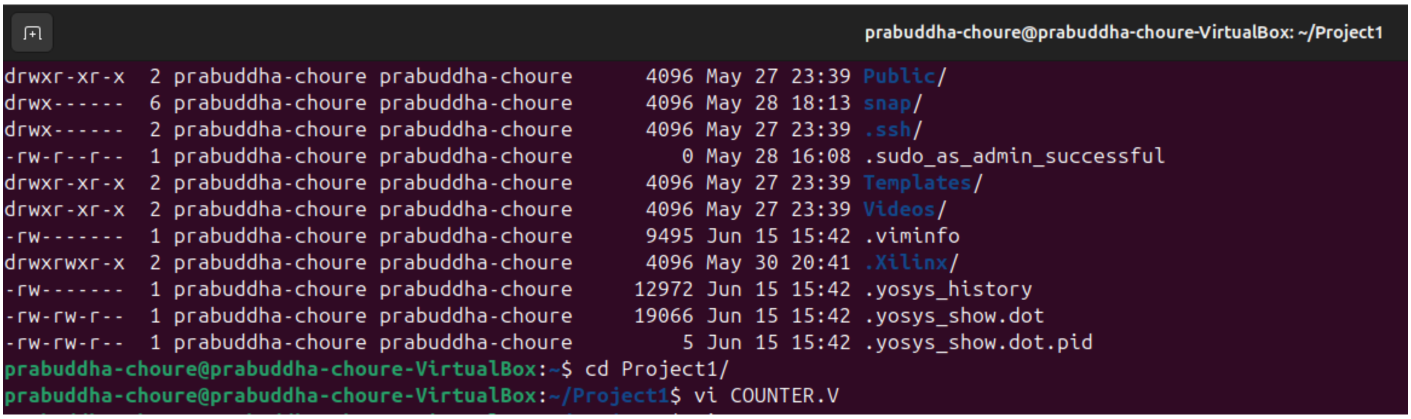
Step 4: Open the folder

cd Project/



Step 5. Now, write below command and press enter to open vi editor.

vi COUNTER.V



Step 6: Press “i” to on insert mode to write RTL code into it.

```
prabuddha-choure@prabuddha-choure-VirtualBox: ~/Project1
module counter (clk, rst, en, count);
input clk, rst, en;
output reg [1:0] count;
always @(posedge clk)
if (rst)
count<= 2'd0;
else if (en)
count<= count +2'd1;
endmodule
~
~
~
~
-- INSERT --
```

Step 7: Press “esc” button to come out from insert mode and write below command to save the RTL code.

:wq

```
prabuddha-choure@prabuddha-choure-VirtualBox: ~/Project1
module counter (clk, rst, en, count);
input clk, rst, en;
output reg [1:0] count;
always @(posedge clk)
if (rst)
count<= 2'd0;
else if (en)
count<= count +2'd1;
endmodule
~
~
~
:wq
```

Step 8: Now again below code to check that COUNTER.V file is made or not.

ll

```
prabuddha-choure@prabuddha-choure-VirtualBox:~/Project1$ vi COUNTER.V
prabuddha-choure@prabuddha-choure-VirtualBox:~/Project1$ ll
total 12
drwxrwxr-x  2 prabuddha-choure prabuddha-choure 4096 Jun 15 15:55 ./
drwxr-x--- 21 prabuddha-choure prabuddha-choure 4096 Jun 15 15:55 ../
-rw-rw-r--  1 prabuddha-choure prabuddha-choure  171 Jun 15 15:55 COUNTER.V
```

Step 9: Active YOSYS tool by writing below command in the project directory.

yosys

```
prabuddha-choure@prabuddha-choure-VirtualBox:~/Project1$ yosys

 /----------------------------------------------------------------------------\
 |                                                                            |
 |  yosys -- Yosys Open SYnthesis Suite                                       |
 |                                                                            |
 |  Copyright (C) 2012 - 2020  Claire Xenia Wolf <claire@yosyshq.com>         |
 |                                                                            |
 |  Permission to use, copy, modify, and/or distribute this software for any  |
 |  purpose with or without fee is hereby granted, provided that the above    |
 |  copyright notice and this permission notice appear in all copies.         |
 |                                                                            |
 |  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES  |
 |  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF          |
 |  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR   |
 |  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES    |
 |  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN     |
 |  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF   |
 |  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.            |
 |                                                                            |
 \----------------------------------------------------------------------------/

 Yosys 0.33 (git sha1 2584903a060)

yosys>
```

Step 10: To generate RTL representation by reading Verilog source file.

read_verilog ../Project1/COUNTER.V

```
/----------------------------------------------------------------------------\
|                                                                            |
|  yosys -- Yosys Open SYnthesis Suite                                       |
|                                                                            |
|  Copyright (C) 2012 - 2020  Claire Xenia Wolf <claire@yosyshq.com>         |
|                                                                            |
|  Permission to use, copy, modify, and/or distribute this software for any  |
|  purpose with or without fee is hereby granted, provided that the above    |
|  copyright notice and this permission notice appear in all copies.         |
|                                                                            |
|  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES  |
|  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF          |
|  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR   |
|  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES    |
|  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN     |
|  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF   |
|  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.            |
|                                                                            |
\----------------------------------------------------------------------------/

 Yosys 0.33 (git sha1 2584903a060)


yosys> read_verilog ../Project1/COUNTER.V
1. Executing Verilog-2005 frontend: ../Project1/COUNTER.V
Parsing Verilog input from `../Project1/COUNTER.V' to AST representation.
Generating RTLIL representation for module `\counter'.
Successfully finished Verilog frontend.

yosys>
```

Step 11: This command is used to elaborate the design hierarchy. (This is the mandatory and first command to write after finishing Verilog frontend.

hierarchy -check -top counter

```
yosys> read_verilog ../Project1/COUNTER.V
1. Executing Verilog-2005 frontend: ../Project1/COUNTER.V
Parsing Verilog input from `../Project1/COUNTER.V' to AST representation.
Generating RTLIL representation for module `\counter'.
Successfully finished Verilog frontend.

yosys> hierarchy -check -top counter

2. Executing HIERARCHY pass (managing design hierarchy).

2.1. Analyzing design hierarchy..
Top module:  \counter

2.2. Analyzing design hierarchy..
Top module:  \counter
Removed 0 unused modules.

yosys>
```

(The word “counter” is taken from the RTL code which in the first line and it acts like a class of object)

Step 12: Now we need to write “help”. This is used to learn all commands in the Yosys tool. (There are many commands and arranged in alphabetical order)

help

```
prabuddha-choure@prabuddha-choure-VirtualBox: ~/Project1

yosys> help

    abc                  use ABC for technology mapping
    abc9                 use ABC9 for technology mapping
    abc9_exe             use ABC9 for technology mapping
    abc9_ops             helper functions for ABC9
    add                  add objects to the design
    aigmap               map logic to and-inverter-graph circuit
    alumacc              extract ALU and MACC cells
    anlogic_eqn          Anlogic: Calculate equations for luts
    anlogic_fixcarry     Anlogic: fix carry chain
    assertpmux           adds asserts for parallel muxes
    async2sync           convert async FF inputs to sync circuits
    attrmap              renaming attributes
    attrmvcp             move or copy attributes from wires to driving cells
    autoname             automatically assign names to objects
    blackbox             convert modules into blackbox modules
    bmuxmap              transform $bmux cells to trees of $mux cells
    bugpoint             minimize testcases
    bwmuxmap             replace $bwmux cells with equivalent logic
    cd                   a shortcut for 'select -module <name>'
    check                check for obvious problems in the design
    chformal             change formal constraints of the design
    chparam              re-evaluate modules with new parameters
```

Step 13: Now, run the following command to convert the process into a netlist.

proc

```
 proc                      translate processes to netlists
```

```
yosys> proc
3. Executing PROC pass (convert processes to netlists).

3.1. Executing PROC_CLEAN pass (remove empty switches from decision trees).
Cleaned up 0 empty switches.

3.2. Executing PROC_RMDEAD pass (remove dead branches from decision trees).
Marked 1 switch rules as full_case in process $proc$../Project1/COUNTER.V:4$1 in module counter.
Removed a total of 0 dead cases.

3.3. Executing PROC_PRUNE pass (remove redundant assignments in processes).
Removed 0 redundant assignments.
Promoted 0 assignments to connections.

3.4. Executing PROC_INIT pass (extract init attributes).

3.5. Executing PROC_ARST pass (detect async resets in processes).

3.6. Executing PROC_ROM pass (convert switches to ROMs).
Converted 0 switches.
<suppressed ~2 debug messages>

3.7. Executing PROC_MUX pass (convert decision trees to multiplexers).
Creating decoders for process `\counter.$proc$../Project1/COUNTER.V:4$1'.
     1/1: $0\count[1:0]

3.8. Executing PROC_DLATCH pass (convert process syncs to latches).

3.9. Executing PROC_DFF pass (convert process syncs to FFs).
Creating register for signal `\counter.\count' using process `\counter.$proc$../Project1/COUNTER.V:4$1'.
  created $dff cell `$procdff$8' with positive edge clock.

3.10. Executing PROC_MEMWR pass (convert process memory writes to cells).

3.11. Executing PROC_CLEAN pass (remove empty switches from decision trees).
Found and cleaned up 2 empty switches in `\counter.$proc$../Project1/COUNTER.V:4$1'.
Removing empty process `counter.$proc$../Project1/COUNTER.V:4$1'.
Cleaned up 2 empty switches.

3.12. Executing OPT_EXPR pass (perform const folding).
Optimizing module counter.

yosys>
```

Step 14: Run the following command to perform simple optimization and clean-ups on the design.

opt

```
opt                    perform simple optimizations
```

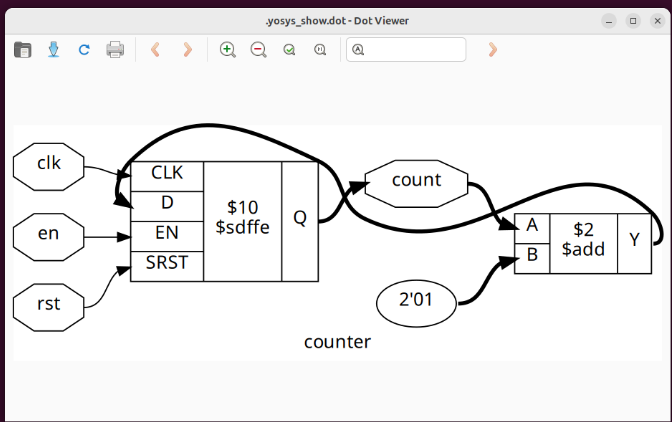
```
yosys> opt

4. Executing OPT pass (performing simple optimizations).

4.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module counter.

4.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module `\counter'.
Removed a total of 0 cells.

4.3. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
Running muxtree optimizer on module \counter..
  Creating internal representation of mux trees.
  Evaluating internal representation of mux trees.
  Analyzing evaluation results.
Removed 0 multiplexer ports.
<suppressed ~1 debug messages>

4.4. Executing OPT_REDUCE pass (consolidate $*mux and $reduce_* inputs).
  Optimizing cells in module \counter.
Performed a total of 0 changes.

4.5. Executing OPT_MERGE pass (detect identical cells).
```

Step 15: Run the below command to visualize the representation of blocks. (This command can be used several times to view the changes in design)

show

```
show                    generate schematics using graphviz
```

```
<suppressed ~3 debug messages>

4.8. Executing OPT_EXPR pass (perform const folding).
Optimizing module counter.

4.9. Rerunning OPT passes. (Maybe there is more to do..)

4.10. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
Running muxtree optimizer on module \counter..
  Creating internal representation of mux trees.
  No muxes found in this module.
Removed 0 multiplexer ports.

4.11. Executing OPT_REDUCE pass (consolidate $*mux and $reduce_* inputs).
  Optimizing cells in module \counter.
Performed a total of 0 changes.

4.12. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module `\counter'.
Removed a total of 0 cells.

4.13. Executing OPT_DFF pass (perform DFF optimizations).

4.14. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \counter..

4.15. Executing OPT_EXPR pass (perform const folding).
Optimizing module counter.

4.16. Finished OPT passes. (There is nothing left to do.)

yosys> show

5. Generating Graphviz representation of design.
Writing dot description to `/home/prabuddha-choure/.yosys_show.dot'.
Dumping module counter to page 1.
Exec: { test -f '/home/prabuddha-choure/.yosys_show.dot.pid' && fuser -s '/home/prabuddha-choure/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $$ >&3; exec xdot '/home/prabuddha-choure/.yosys_show.dot'; ) 3> '/home/prabuddha-choure/.yosys_show.dot.pid' &
```



Step 16: Run the following two command one by one. The feature of these commands is already done in optimization. But to visit every corner of the code, it is used. (After performing these command, rum "opt" one more time to clean-up)

fsm

```
fsm                          extract and optimize finite state machines
```

```
yosys> fsm

6. Executing FSM pass (extract and optimize FSM).

6.1. Executing FSM_DETECT pass (finding FSMs in design).

6.2. Executing FSM_EXTRACT pass (extracting FSM from design).

6.3. Executing FSM_OPT pass (simple optimizations of FSMs).

6.4. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \counter..

6.5. Executing FSM_OPT pass (simple optimizations of FSMs).

6.6. Executing FSM_RECODE pass (re-assigning FSM state encoding).

6.7. Executing FSM_INFO pass (dumping all available information on FSM cells).

6.8. Executing FSM_MAP pass (mapping FSMs to basic logic).
```

memory

```
memory                       translate memories to basic cells
```

```
yosys> memory

7. Executing MEMORY pass.

7.1. Executing OPT_MEM pass (optimize memories).
Performed a total of 0 transformations.

7.2. Executing OPT_MEM_PRIORITY pass (removing unnecessary memory write priority relations).
Performed a total of 0 transformations.

7.3. Executing OPT_MEM_FEEDBACK pass (finding memory read-to-write feedback paths).

7.4. Executing MEMORY_BMUX2ROM pass (converting muxes to ROMs).

7.5. Executing MEMORY_DFF pass (merging $dff cells to $memrd).

7.6. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \counter..

7.7. Executing MEMORY_SHARE pass (consolidating $memrd/$memwr cells).

7.8. Executing OPT_MEM_WIDEN pass (optimize memories where all ports are wide).
Performed a total of 0 transformations.

7.9. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \counter..

7.10. Executing MEMORY_COLLECT pass (generating $mem cells).

7.11. Executing MEMORY_MAP pass (converting memories to logic and flip-flops).
```

Step 17: Run the following command to dump the module code. It will create a file of Verilog code which can be used to compare the final netlist code. (This should be executed before "techmap")

write_verilog

```
write_verilog         write design to Verilog file
```

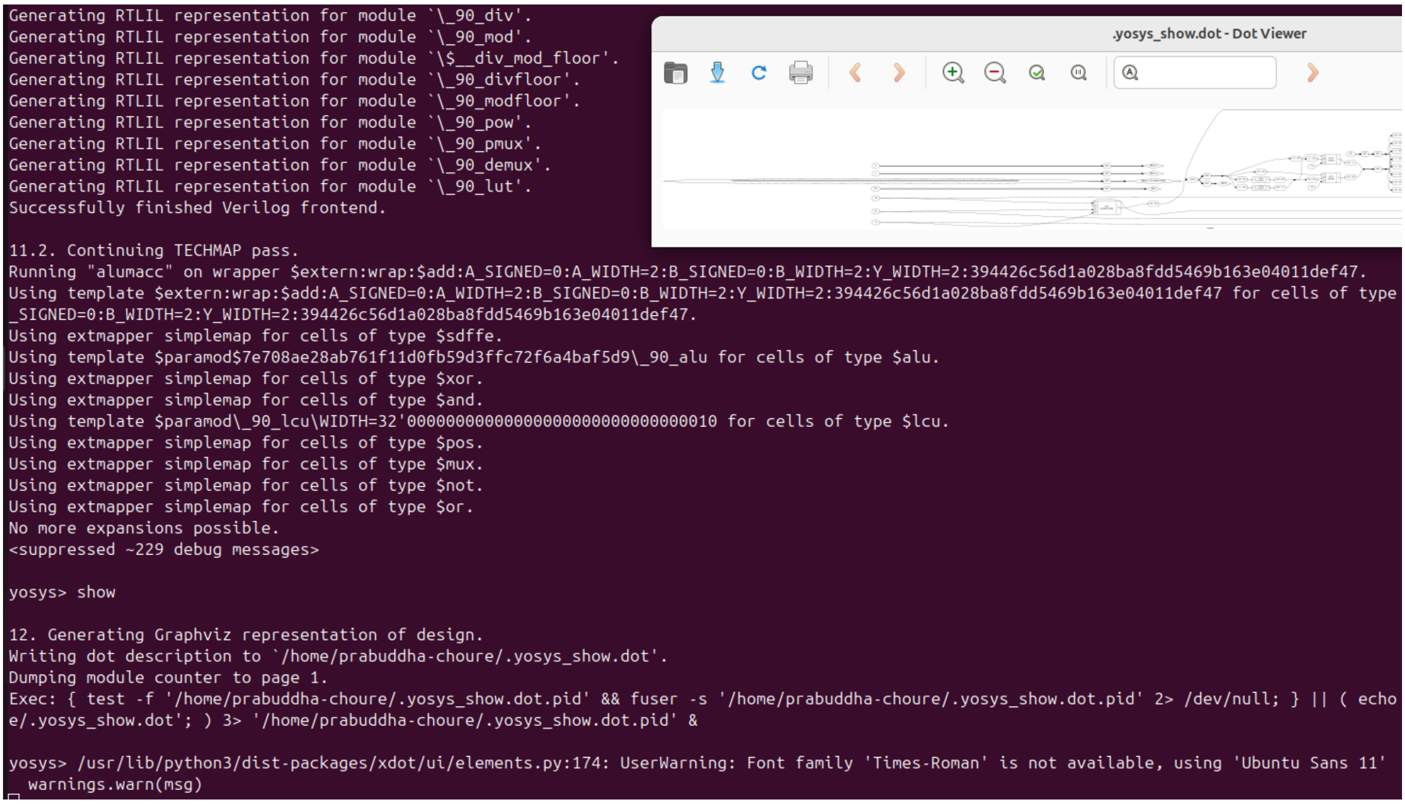
```
yosys> write_verilog synth1.v

10. Executing Verilog backend.

10.1. Executing BMUXMAP pass.

10.2. Executing DEMUXMAP pass.
Dumping module `\counter'.

yosys>
```

Step 18: Run the following command to convert RTL cells into basic logic gates. After executing it, run "show" command to visualize graphic representation of fine-grain gates.

techmap

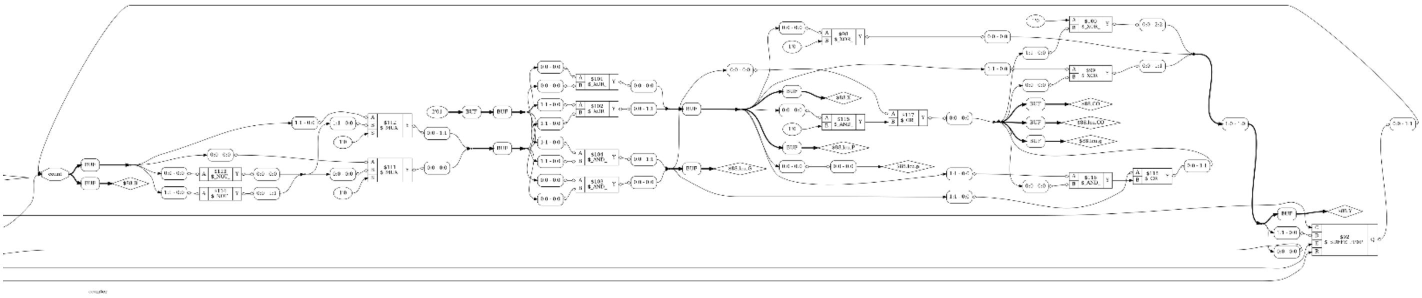
```
techmap                  generic technology mapper
```

```
yosys> techmap

11. Executing TECHMAP pass (map to technology primitives).

11.1. Executing Verilog-2005 frontend: /usr/bin/../share/yosys/techmap.v
Parsing Verilog input from `/usr/bin/../share/yosys/techmap.v' to AST representation.
Generating RTLIL representation for module `\_90_simplemap_bool_ops'.
Generating RTLIL representation for module `\_90_simplemap_reduce_ops'.
Generating RTLIL representation for module `\_90_simplemap_logic_ops'.
Generating RTLIL representation for module `\_90_simplemap_compare_ops'.
Generating RTLIL representation for module `\_90_simplemap_various'.
Generating RTLIL representation for module `\_90_simplemap_registers'.
Generating RTLIL representation for module `\_90_shift_ops_shr_shl_sshl_sshr'.
Generating RTLIL representation for module `\_90_shift_shiftx'.
Generating RTLIL representation for module `\_90_fa'.
Generating RTLIL representation for module `\_90_lcu'.
Generating RTLIL representation for module `\_90_alu'.
Generating RTLIL representation for module `\_90_macc'.
Generating RTLIL representation for module `\_90_alumacc'.
Generating RTLIL representation for module `\$__div_mod_u'.
Generating RTLIL representation for module `\$__div_mod_trunc'.
Generating RTLIL representation for module `\_90_div'.
Generating RTLIL representation for module `\_90_mod'.
Generating RTLIL representation for module `\$__div_mod_floor'.
Generating RTLIL representation for module `\_90_divfloor'.
Generating RTLIL representation for module `\_90_modfloor'.
Generating RTLIL representation for module `\_90_pow'.
Generating RTLIL representation for module `\_90_pmux'.
Generating RTLIL representation for module `\_90_demux'.
Generating RTLIL representation for module `\_90_lut'.
Successfully finished Verilog frontend.

11.2. Continuing TECHMAP pass.
Running "alumacc" on wrapper $extern:wrap:$add:A_SIGNED=0:A_WIDTH=2:B_SIGNED=0:B_WIDTH=2:Y_WIDTH=2:394426c56d1a028ba
```



```
Generating RTLIL representation for module `\_90_div'.
Generating RTLIL representation for module `\_90_mod'.
Generating RTLIL representation for module `\$__div_mod_floor'.
Generating RTLIL representation for module `\_90_divfloor'.
Generating RTLIL representation for module `\_90_modfloor'.
Generating RTLIL representation for module `\_90_pow'.
Generating RTLIL representation for module `\_90_pmux'.
Generating RTLIL representation for module `\_90_demux'.
Generating RTLIL representation for module `\_90_lut'.
Successfully finished Verilog frontend.

11.2. Continuing TECHMAP pass.
Running "alumacc" on wrapper $extern:wrap:$add:A_SIGNED=0:A_WIDTH=2:B_SIGNED=0:B_WIDTH=2:Y_WIDTH=2:394426c56d1a028ba8fdd5469b163e04011def47.
Using template $extern:wrap:$add:A_SIGNED=0:A_WIDTH=2:B_SIGNED=0:B_WIDTH=2:Y_WIDTH=2:394426c56d1a028ba8fdd5469b163e04011def47 for cells of type
_SIGNED=0:B_WIDTH=2:Y_WIDTH=2:394426c56d1a028ba8fdd5469b163e04011def47.
Using extmapper simplemap for cells of type $sdffe.
Using template $paramod$7e708ae28ab761f11d0fb59d3ffc72f6a4baf5d9\_90_alu for cells of type $alu.
Using extmapper simplemap for cells of type $xor.
Using extmapper simplemap for cells of type $and.
Using template $paramod\_90_lcu\WIDTH=32'00000000000000000000000000000010 for cells of type $lcu.
Using extmapper simplemap for cells of type $pos.
Using extmapper simplemap for cells of type $mux.
Using extmapper simplemap for cells of type $not.
Using extmapper simplemap for cells of type $or.
No more expansions possible.
<suppressed ~229 debug messages>

yosys> show

12. Generating Graphviz representation of design.
Writing dot description to `/home/prabuddha-choure/.yosys_show.dot'.
Dumping module counter to page 1.
Exec: { test -f '/home/prabuddha-choure/.yosys_show.dot.pid' && fuser -s '/home/prabuddha-choure/.yosys_show.dot.pid' 2> /dev/null; } || ( echo
e/.yosys_show.dot'; ) 3> '/home/prabuddha-choure/.yosys_show.dot.pid' &

yosys> /usr/lib/python3/dist-packages/xdot/ui/elements.py:174: UserWarning: Font family 'Times-Roman' is not available, using 'Ubuntu Sans 11'
  warnings.warn(msg)
```


continue



(Again, perform optimization – “opt”)

Step 19: Run the following command for technology mapping to convert high level logic in to gates available in a specific hardware library. (After typing each path ex. ../Project1, just press tab, you will get the next files)

abc -liberty ../Project1/Sky130/sky130_fd_sc_hd__tt_025C_1v80.lib

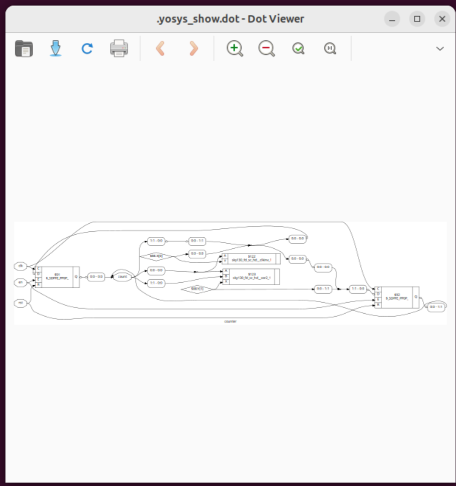
```
abc                       use ABC for technology mapping
```

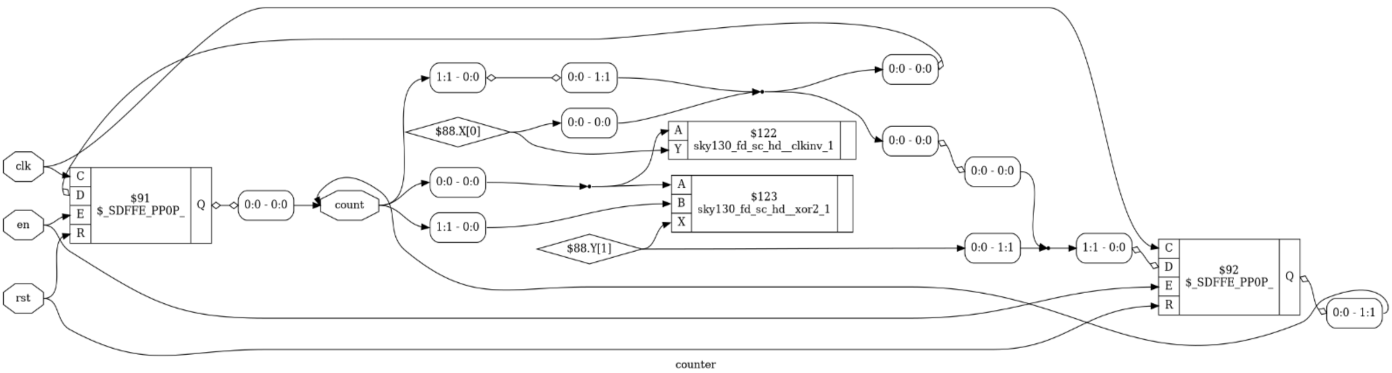
```
yosys> abc -liberty ../Project1/Sky130/sky130_fd_sc_hd__tt_025C_1v80.lib

14. Executing ABC pass (technology mapping using ABC).

14.1. Extracting gate netlist of module `\counter' to `<abc-temp-dir>/input.blif'..
Extracted 2 gates and 4 wires to a netlist network with 2 inputs and 2 outputs.

14.1.1. Executing ABC.
Running ABC command: "<yosys-exe-dir>/yosys-abc" -s -f <abc-temp-dir>/abc.script 2>&1
ABC: ABC command line: "source <abc-temp-dir>/abc.script".
ABC:
ABC: + read_blif <abc-temp-dir>/input.blif
ABC: + read_lib -w /home/prabuddha-choure/Project1/../Project1/Sky130/sky130_fd_sc_hd__tt_025C
ABC: Parsing finished successfully.  Parsing time =     0.07 sec
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_12" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_3" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_4" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_6" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_8" without logic function.
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbn_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbn_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrtn_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrtp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrtp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrtp_4".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfsbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfsbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfstp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfstp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfstp_4".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfxbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfxbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfxtp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfxtp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfxtp_4".
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__diode_2" without logic function.
```

```
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sdfxtp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sdfxtp_4".
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_1" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_2" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_4" without logic function.
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sedfxbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sedfxbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sedfxtp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sedfxtp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__sedfxtp_4".
ABC: Library "sky130_fd_sc_hd__tt_025C_1v80" from "/home/prabuddha-choure/Project1/../Project1/Sk
; 18 no func; 0 dont_use).  Time =     0.11 sec
ABC: Memory =   19.80 MB. Time =     0.11 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd__fa_1").
ABC: + strash
ABC: + &get -n
ABC: + &fraig -x
ABC: + &put
ABC: + scorr
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + dc2
ABC: + dretime
ABC: + strash
ABC: + &get -n
ABC: + &dch -f
ABC: + &nf
ABC: + &put
ABC: + write_blif <abc-temp-dir>/output.blif

14.1.2. Re-integrating ABC results.
ABC RESULTS:   sky130_fd_sc_hd__clkinv_1 cells:        1
ABC RESULTS:   sky130_fd_sc_hd__xor2_1 cells:        1
ABC RESULTS:        internal signals:        0
ABC RESULTS:           input signals:        2
ABC RESULTS:          output signals:        2
Removing temp directory.

yosys>
```

Step 20: Again, run the “show” command for graphical representation of netlist code.

```
ABC: Library "sky130_fd_sc_hd__tt_025C_1v80" from "/home/prabuddha-choure/Project1/../Project1/Sky130/sky130_fd_sc_hd__tt_025C_1v80.lib" has 334 cells (94 skipped: 63 seq; 13 tri-state
; 18 no func; 0 dont_use).  Time =     0.11 sec
ABC: Memory =   19.80 MB. Time =     0.11 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd__fa_1").
ABC: + strash
ABC: + &get -n
ABC: + &fraig -x
ABC: + &put
ABC: + scorr
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + dc2
ABC: + dretime
ABC: + strash
ABC: + &get -n
ABC: + &dch -f
ABC: + &nf
ABC: + &put
ABC: + write_blif <abc-temp-dir>/output.blif

14.1.2. Re-integrating ABC results.
ABC RESULTS:   sky130_fd_sc_hd__clkinv_1 cells:        1
ABC RESULTS:   sky130_fd_sc_hd__xor2_1 cells:        1
ABC RESULTS:        internal signals:        0
ABC RESULTS:           input signals:        2
ABC RESULTS:          output signals:        2
Removing temp directory.

yosys> show

15. Generating Graphviz representation of design.
Writing dot description to `/home/prabuddha-choure/.yosys_show.dot'.
Dumping module counter to page 1.
Exec: { test -f '/home/prabuddha-choure/.yosys_show.dot.pid' && fuser -s '/home/prabuddha-choure/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $$ >&3; exec xdot '/home/prabuddha-chour
e/.yosys_show.dot'; ) 3> '/home/prabuddha-choure/.yosys_show.dot.pid' &

yosys> /usr/lib/python3/dist-packages/xdot/ui/elements.py:174: UserWarning: Font family 'Times-Roman' is not available, using 'Ubuntu Sans 11'
  warnings.warn(msg)
```





Step 21: Run the below command to clean up the design.

clean

```
yosys> clean
Removed 0 unused cells and 4 unused wires.

yosys>
```

Step 22: Run the following command to generate netlist file. This is the same command used in the ‘Step 17’, but the file name should be different.

```
 write_verilog          write design to Verilog file
```

```
yosys> write_verilog SYNTH1.V

16. Executing Verilog backend.

16.1. Executing BMUXMAP pass.

16.2. Executing DEMUXMAP pass.
Dumping module `\counter'.

yosys>
```

Step 23: Run the following command to exit from the yosys tool.

exit

```
yosys> exit
```

Step 24: Run the following command to see the raw netlist code before technology mapping.

vi synth1.v

```
prabuddha-choure@prabuddha-choure-VirtualBox:~$ cd Project1/
prabuddha-choure@prabuddha-choure-VirtualBox:~/Project1$ vi synth1.v
```

```
/* Generated by Yosys 0.33 (git sha1 2584903a060) */

(* top =  1  *)
(* src = "../Project1/COUNTER.V:1.1-9.10" *)
module counter(clk, rst, en, count);
  (* src = "../Project1/COUNTER.V:8.9-8.20" *)
  wire [1:0] _0_;
  (* src = "../Project1/COUNTER.V:2.7-2.10" *)
  input clk;
  wire clk;
  (* src = "../Project1/COUNTER.V:3.18-3.23" *)
  output [1:0] count;
  reg [1:0] count;
  (* src = "../Project1/COUNTER.V:2.17-2.19" *)
  input en;
  wire en;
  (* src = "../Project1/COUNTER.V:2.12-2.15" *)
  input rst;
  wire rst;
  assign _0_ = count + (* src = "../Project1/COUNTER.V:8.9-8.20" *) 2'h
  (* src = "../Project1/COUNTER.V:4.1-8.21" *)
  always @(posedge clk)
    if (rst) count <= 2'h0;
    else if (en) count <= _0_;
endmodule
~
~
~
~
~
~
~
~
~
~
~
~
"synth1.v" 25L, 736B
```

Step 25: Run the following command to see the final netlist code which will be used in partitioning.

vi SYNTH1.V

```
/* Generated by Yosys 0.33 (git sha1 2584903a060) */

(* top =  1  *)
(* src = "../Project1/COUNTER.V:1.1-9.10" *)
module counter(clk, rst, en, count);
  (* force_downto = 32'd1 *)
  (* src = "../Project1/COUNTER.V:8.9-8.20|/usr/bin/../share/yosys/techmap.v:270.23-270.24" *)
  wire [1:0] _0_;
  (* force_downto = 32'd1 *)
  (* src = "../Project1/COUNTER.V:8.9-8.20|/usr/bin/../share/yosys/techmap.v:270.26-270.27" *)
  wire [1:0] _1_;
  (* src = "../Project1/COUNTER.V:2.7-2.10" *)
  input clk;
  wire clk;
  (* src = "../Project1/COUNTER.V:3.18-3.23" *)
  output [1:0] count;
  reg [1:0] count;
  (* src = "../Project1/COUNTER.V:2.17-2.19" *)
  input en;
  wire en;
  (* src = "../Project1/COUNTER.V:2.12-2.15" *)
  input rst;
  wire rst;
  sky130_fd_sc_hd__clkinv_1 _2_ (
    .A(count[0]),
    .Y(_0_[0])
  );
  sky130_fd_sc_hd__xor2_1 _3_ (
    .A(count[0]),
    .B(count[1]),
    .X(_1_[1])
  );
  (* src = "../Project1/COUNTER.V:4.1-8.21" *)
  always @(posedge clk)
    if (rst) count[0] <= 1'h0;
    else if (en) count[0] <= _0_[0];
  (* src = "../Project1/COUNTER.V:4.1-8.21" *)
```

```
  (* src = "../Project1/COUNTER.V:8.9-8.20|/usr/bin/../share/yosys/techmap.v:270.23-270.24" *)
  wire [1:0] _0_;
  (* force_downto = 32'd1 *)
  (* src = "../Project1/COUNTER.V:8.9-8.20|/usr/bin/../share/yosys/techmap.v:270.26-270.27" *)
  wire [1:0] _1_;
  (* src = "../Project1/COUNTER.V:2.7-2.10" *)
  input clk;
  wire clk;
  (* src = "../Project1/COUNTER.V:3.18-3.23" *)
  output [1:0] count;
  reg [1:0] count;
  (* src = "../Project1/COUNTER.V:2.17-2.19" *)
  input en;
  wire en;
  (* src = "../Project1/COUNTER.V:2.12-2.15" *)
  input rst;
  wire rst;
  sky130_fd_sc_hd__clkinv_1 _2_ (
    .A(count[0]),
    .Y(_0_[0])
  );
  sky130_fd_sc_hd__xor2_1 _3_ (
    .A(count[0]),
    .B(count[1]),
    .X(_1_[1])
  );
  (* src = "../Project1/COUNTER.V:4.1-8.21" *)
  always @(posedge clk)
    if (rst) count[0] <= 1'h0;
    else if (en) count[0] <= _0_[0];
  (* src = "../Project1/COUNTER.V:4.1-8.21" *)
  always @(posedge clk)
    if (rst) count[1] <= 1'h0;
    else if (en) count[1] <= _1_[1];
  assign _0_[1] = count[1];
  assign _1_[0] = _0_[0];
endmodule
```

Project-II: SPI using Qflow.

Serial Peripheral Interface:

The Serial Peripheral Interface (SPI) is a synchronous serial communication protocol primarily used for short-distance communication in embedded systems. Developed by Motorola, SPI enables full-duplex data transmission between a master device and one or more slave devices. It is widely adopted for communication between microcontrollers and peripherals like sensors, ADCs, DACs, displays, and memory devices.

Key Features:

- Full-duplex communication: Simultaneous transmission and reception.
- Master-slave architecture: One master controls one or more slave devices.
- High-speed data transfer: Faster than I^2C due to simple protocol and no address overhead.
- Four-wire interface:
  - MOSI (Master Out Slave In)
  - MISO (Master In Slave Out)
  - SCLK (Serial Clock)
  - SS/CS (Slave Select / Chip Select)

Advantages:

- Simple hardware and protocol.
- Supports high-speed communication.
- Efficient for short-distance, high-throughput systems.

Limitations:

- Requires more pins than I^2C (especially for multiple slaves).
- No built-in acknowledgment or error checking.

Qflow:

Qflow is an open-source digital synthesis flow used for VLSI backend design of digital circuits. It provides a complete RTL-to-GDSII flow using a collection of open-source tools. Qflow is ideal for academic and research purposes, allowing users to understand and implement the physical design of digital integrated circuits from Verilog source code.

Key Features:

- Supports RTL synthesis, technology mapping, placement, routing, DRC, LVS, and GDSII generation.
- Uses standard-cell libraries for design.
- Based on widely used open-source tools:
    - Yosys – for synthesis
    - Graywolf – for placement
    - Qrouter – for routing
    - Magic – for layout visualization, DRC, and GDSII export
    - Netgen – for LVS (Layout vs Schematic) checks

Typical Flow Steps:

1. RTL Design – Write the digital design in Verilog.
2. Logic Synthesis – Using Yosys to convert Verilog to gate-level netlist.
3. Technology Mapping – Map generic gates to standard cells from a selected technology.
4. Placement – Determine locations of standard cells using Graywolf.
5. Routing – Interconnect placed cells using Qrouter.
6. DRC & LVS – Use Magic and Netgen for rule checks and verification.
7. GDSII Generation – Export the final chip layout in GDSII format for fabrication.

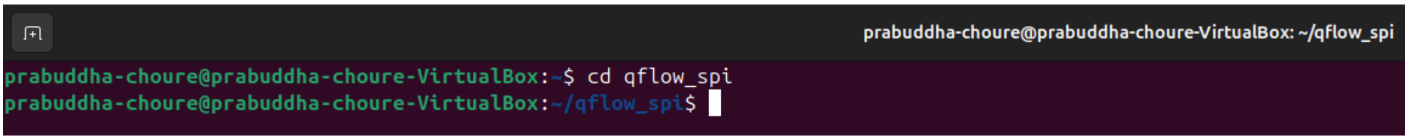
Advantages:

- Fully open-source, customizable.
- Suitable for learning and prototyping VLSI physical design.
- Integrates well with custom or academic cell libraries.

Invoke The QFLOW Tool.

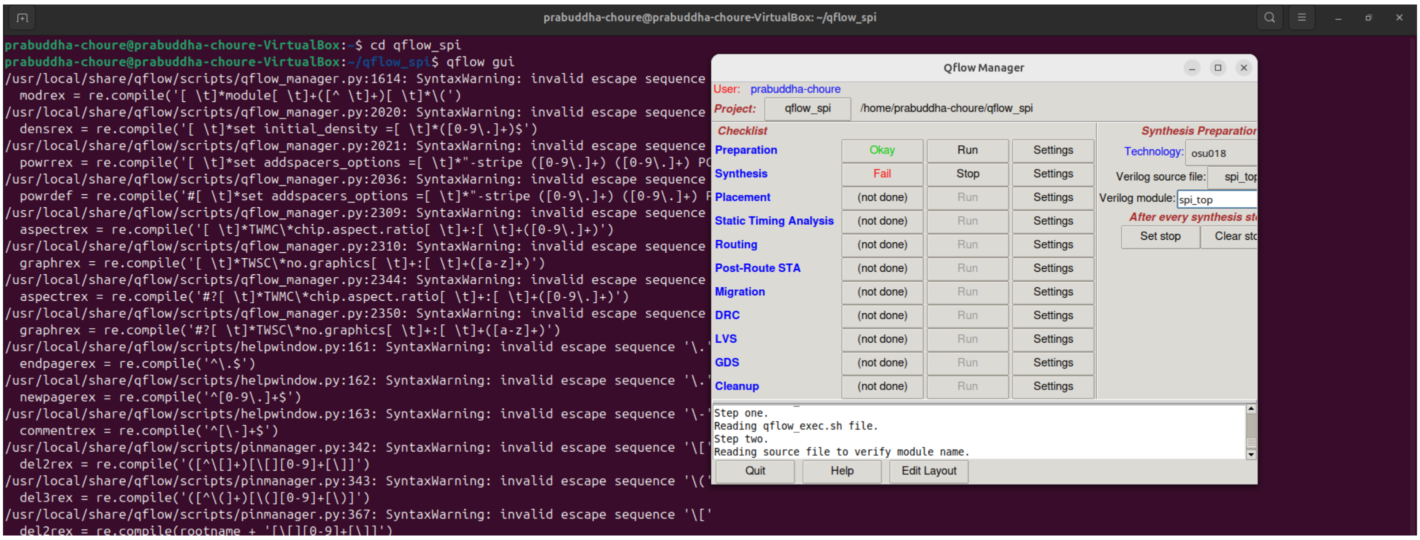
Basic steps to invoke qflow.

Step 1: Open the project directory in terminal.



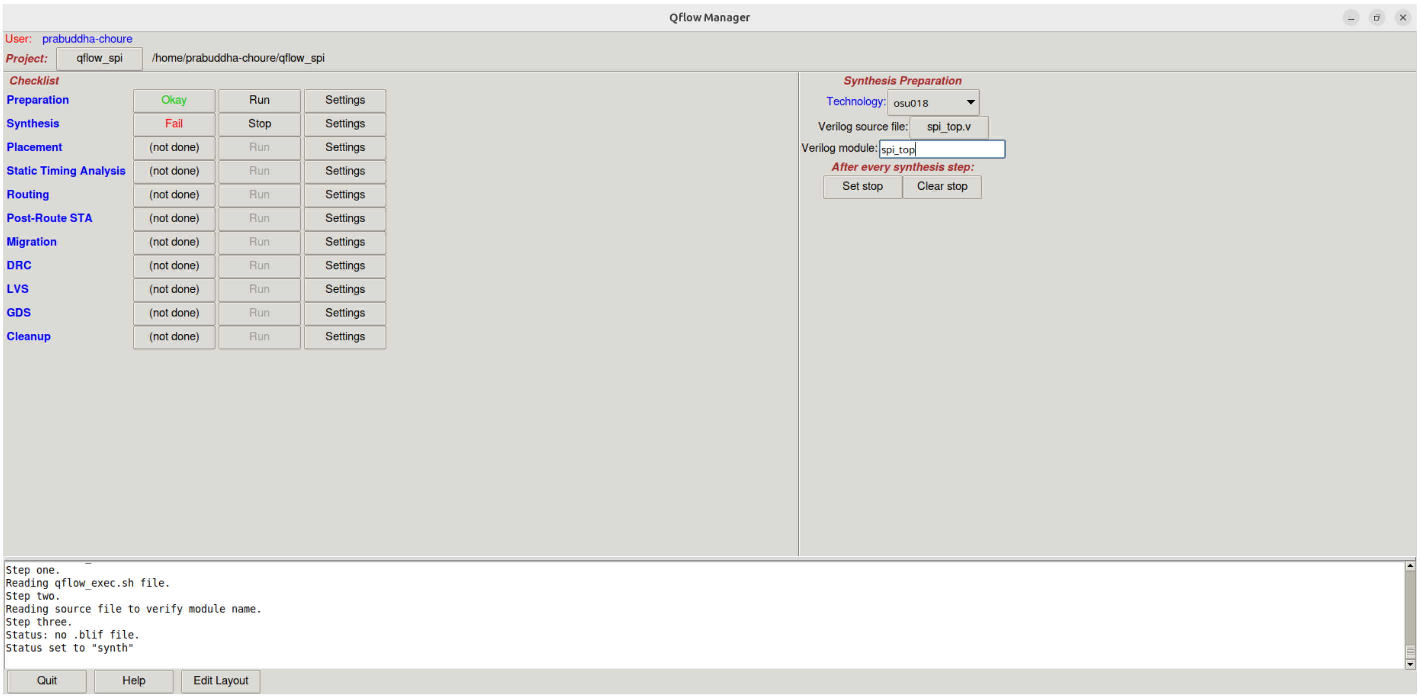
Step 2: Run below command to invoke qflow.

qflow gui



<u>Qflow gui:</u>

Qflow GUI is the graphical user interface version of the Qflow toolchain, which is traditionally command-line-based. It provides a visual interface to run and manage the various stages of the digital VLSI physical design flow — from Verilog synthesis to layout generation — without manually typing commands.



Key Features of Qflow GUI:

- Visual Workflow Control:
  Allows users to step through each stage of the Qflow design process (synthesis, placement, routing, DRC, etc.) using buttons or menus.

- Real-Time Status Updates:
  Displays logs and status messages in a structured, readable format.

- Easy Project Management:
  Simplifies selecting the project directory, top module, and standard cell library.

- Tool Integration:
  Internally calls the same backend tools used in the command-line flow:

  - Yosys (synthesis)
  - Graywolf (placement)
  - Qrouter (routing)
  - Magic (DRC/GDSII)
  - Netgen (LVS)

- Layout Visualization Support:
  Can launch Magic to view layout results at the push of a button.

Preparation:

Before running the digital design flow using Qflow, certain preparation steps must be completed to ensure a smooth synthesis-to-layout process. These steps set up the necessary environment, organize design files, and configure project settings.

1. Install Required Tools

   Ensure that Qflow and its dependencies are installed:

- Qflow
- Yosys (synthesis)
- Graywolf (placement)
- Qrouter (routing)

- Magic (layout viewer and DRC)
- Netgen (LVS)
- TCL/Tk (for GUI, if used)

2. Create Project Directory

A dedicated directory is created for each project. This directory contains:

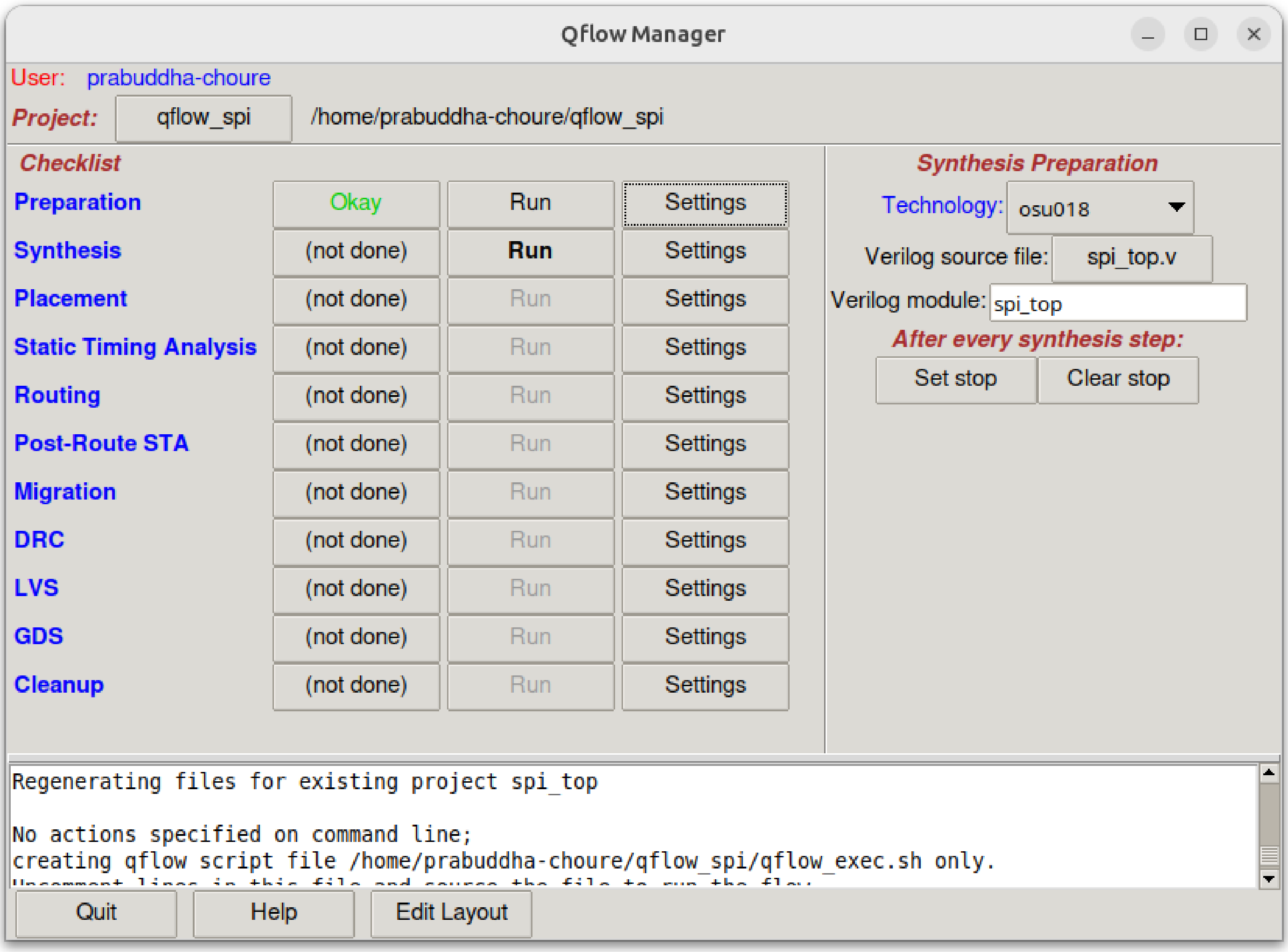
- Verilog RTL files
- Constraint files (optional)
- Standard cell library links
- Qflow configuration files

3. Add the RTL File

Place your Verilog (.v) source file (e.g., my_design.v) inside the project directory.

4. Select Standard Cell Library

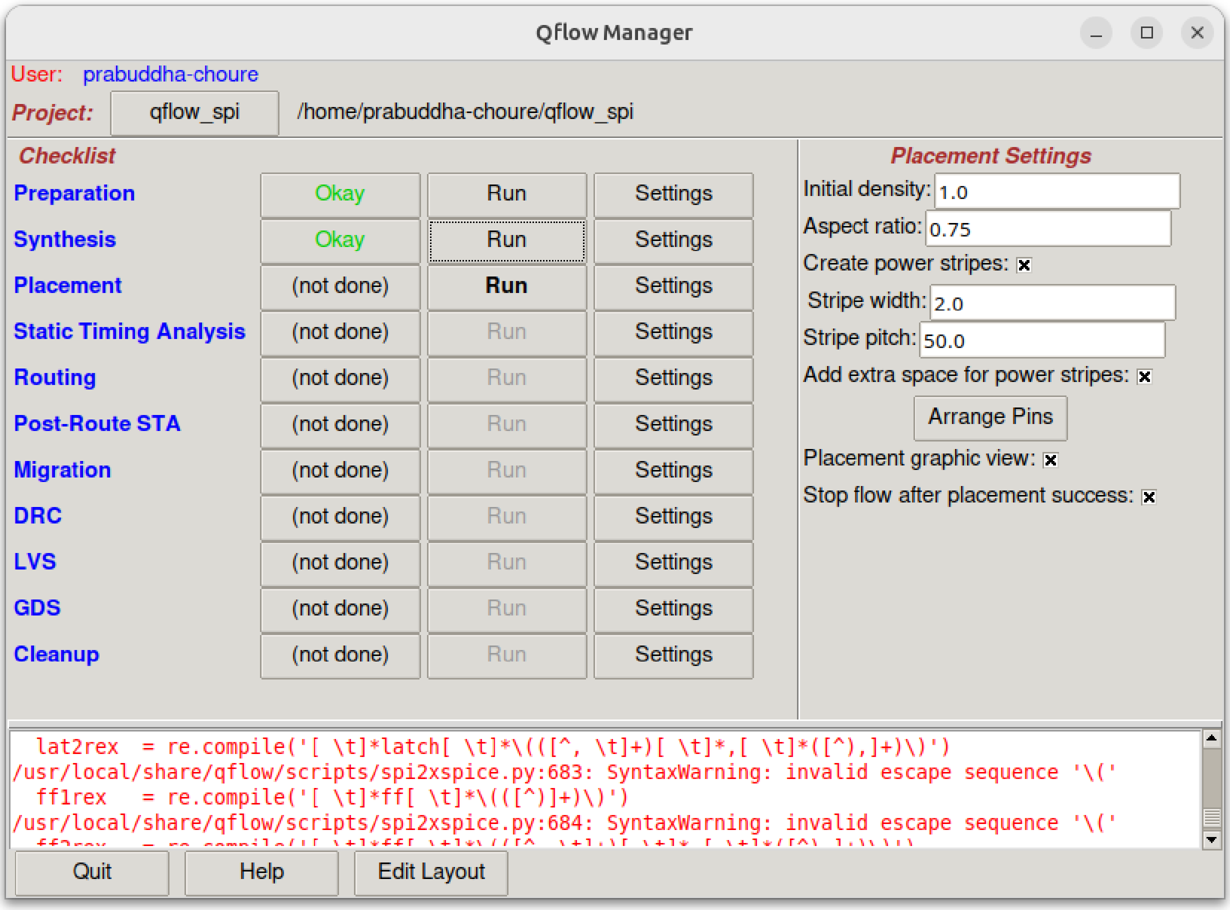
Choose a compatible standard cell library (e.g., osu018, sky130). Place or link the library in your project or configure Qflow to locate it.



Synthesis:

Synthesis is the first major step in the Qflow design process. It converts the RTL description written in Verilog into a gate-level netlist composed of logic gates from a specific standard cell library.

In Qflow, synthesis is performed using Yosys, an open-source synthesis tool that supports Verilog HDL.



<u>Objectives of Synthesis:</u>

- Translate high-level Verilog code into a netlist of standard gates.
- Optimize logic to reduce area, delay, and power consumption.
- Ensure the design is mappable to a given standard cell library.

<u>Steps Performed During Synthesis:</u>

1. Parsing RTL

   Yosys reads and checks the Verilog code for syntax and semantic errors.

2. Elaboration

   The hierarchical design is flattened for easier processing.

3. Logic Optimization

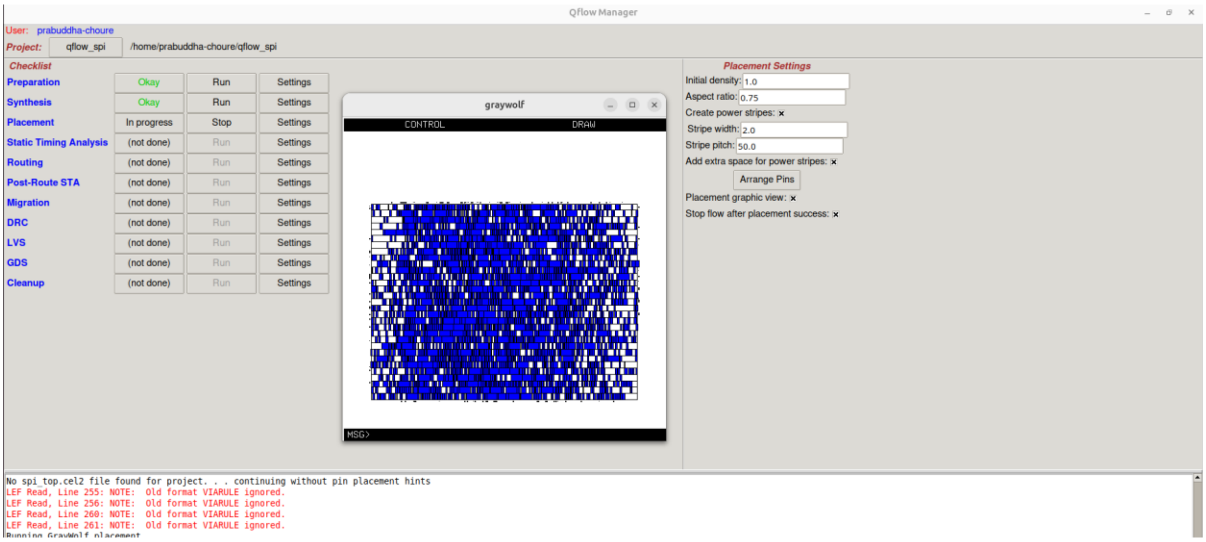
   Combinational logic is optimized using Boolean algebra and technology mapping.

4. Technology Mapping
   The design is mapped to gates (e.g., NAND, NOR, Flip-Flops) available in the selected standard cell library (e.g., osu018 or sky130).

5. Netlist Generation
   A final gate-level netlist (.blif or .v) is generated for use in placement and routing.

Placement:

Placement is the process of determining the physical locations of the standard cells (logic gates) on the silicon chip. In Qflow, placement is handled by Graywolf, an open-source placement tool optimized for standard-cell-based digital design.



<u>Objectives of Placement:</u>

- Assign coordinates to each cell in the gate-level netlist.
- Minimize total wire length and routing congestion.
- Maintain timing and physical constraints.
- Avoid cell overlap and optimize layout area.

<u>Steps Performed During Placement:</u>

1. Read Netlist
   The gate-level netlist generated during synthesis is parsed.

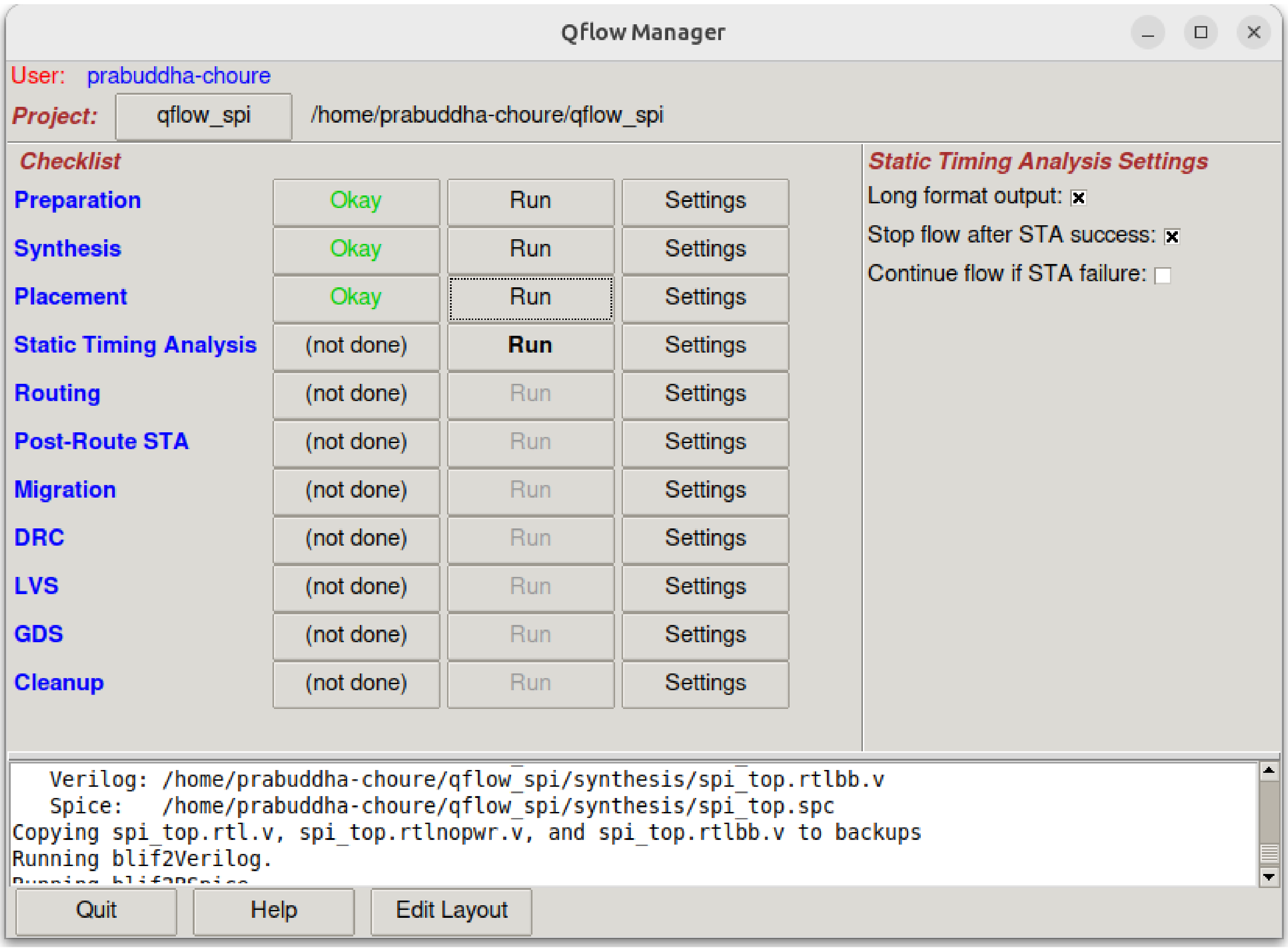
2. Initial Placement
   Graywolf performs a rough cell placement based on connectivity.

3. Optimization

   The placement is improved by minimizing estimated wire lengths and balancing cell density.

4. Row Assignment

   Cells are aligned into standard rows, respecting standard cell heights and layout rules.



Static Timing Analysis:

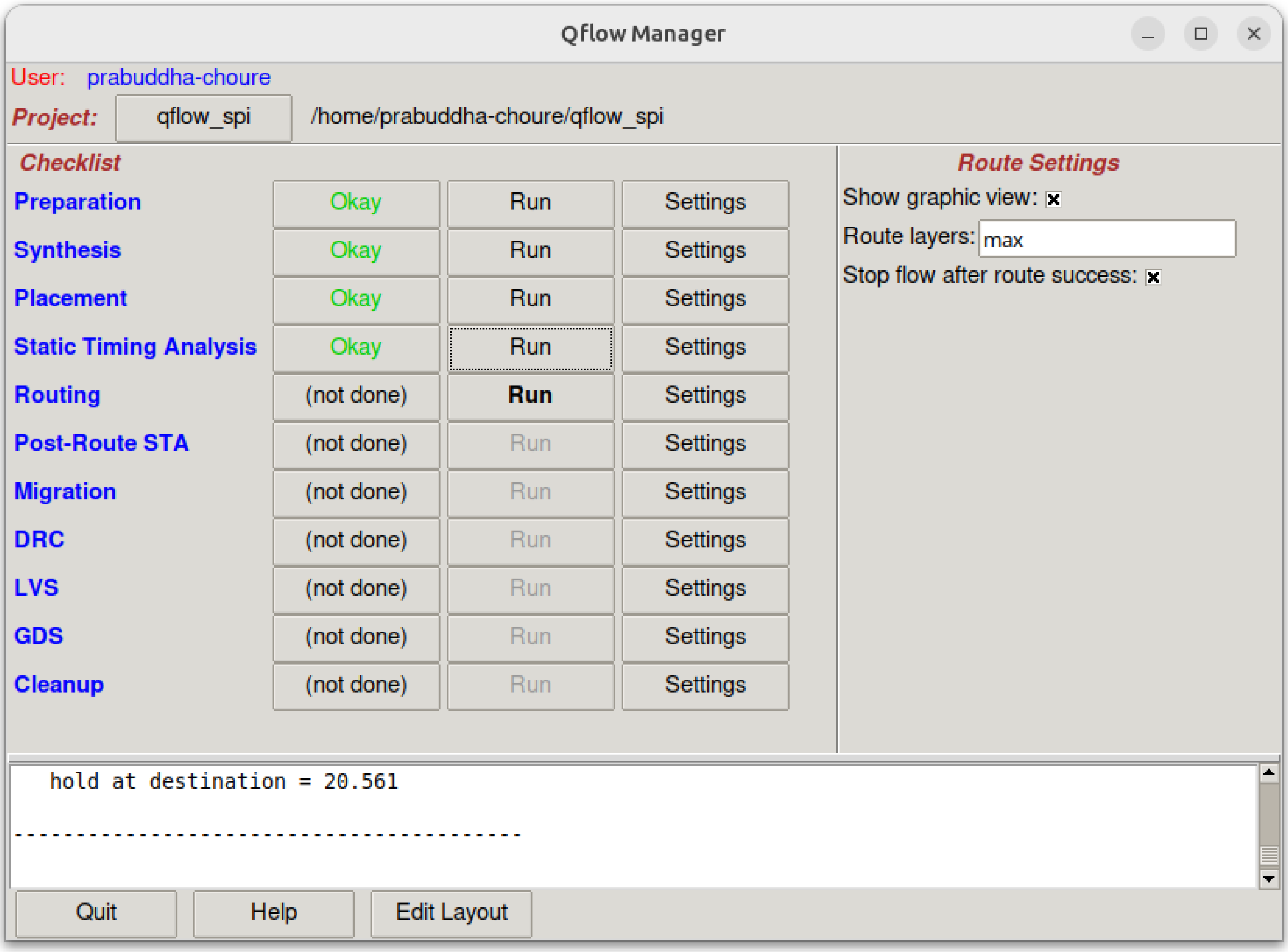
Static Timing Analysis (STA) is a method of validating the timing performance of a digital circuit without requiring input stimulus. It checks whether all data signals in the design meet the required setup and hold time constraints.

In Qflow, STA is performed using Qflow's built-in timing analysis step, which typically relies on data from synthesis and placement stages, combined with the delay models from the standard cell library (such as .lib files).

<u>Objectives of STA:</u>

- Ensure that all signal paths meet timing requirements.

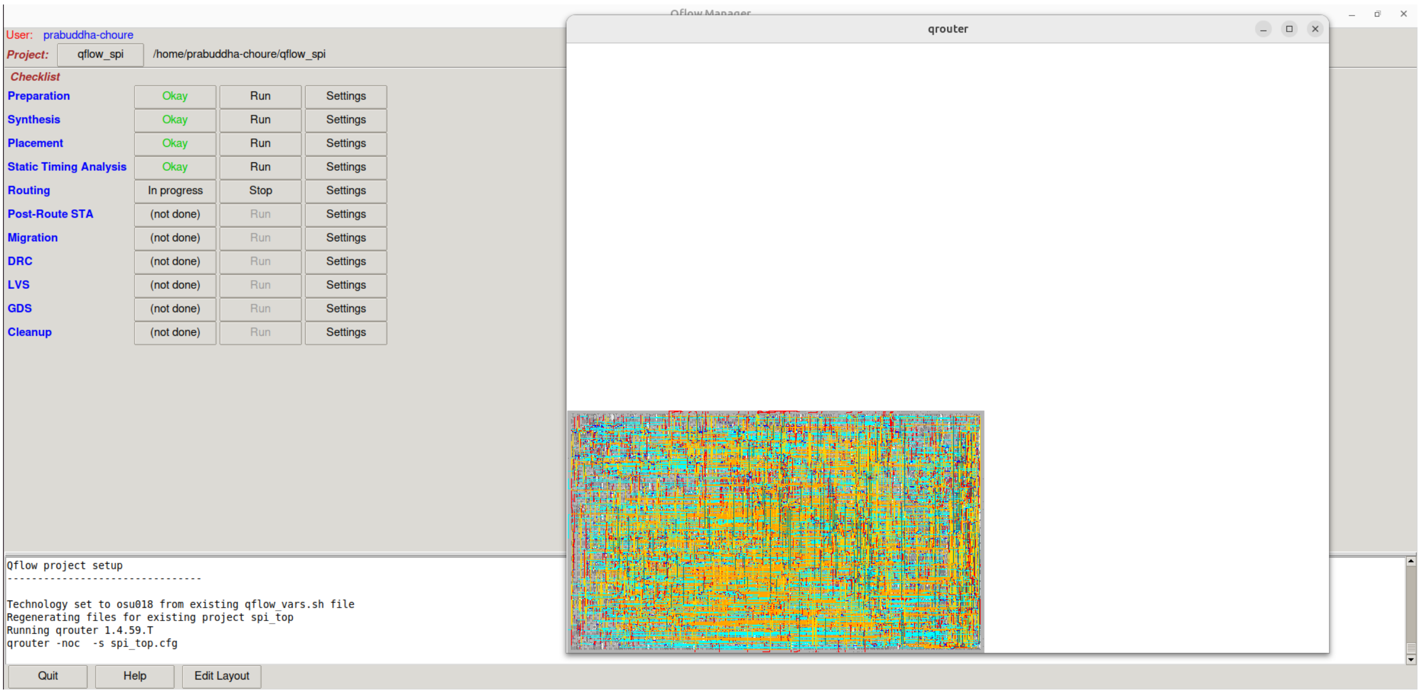
- Identify critical paths (longest delay paths).
- Check for setup and hold violations.
- Validate maximum and minimum delays across the design.



Routing:

Routing is the process of connecting the placed standard cells according to the netlist generated during synthesis. In Qflow, routing is performed using Qrouter, an open-source detail router designed for digital standard cell layouts.

Routing ensures that all logical connections (nets) between gates are implemented using physical metal wires while obeying design rules and avoiding shorts or opens.



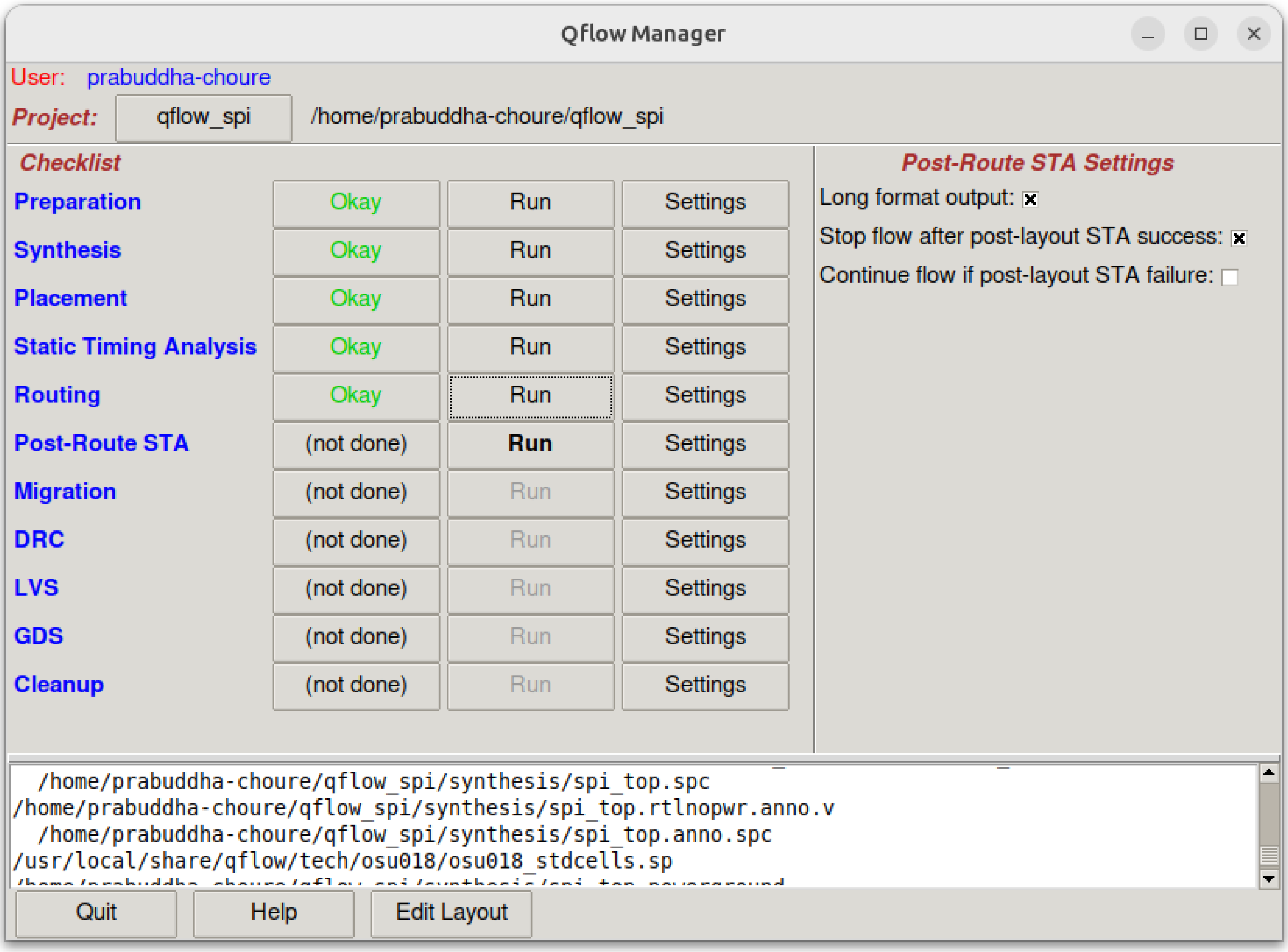
Objectives of Routing:

- Connect all signal nets between placed cells.
- Follow metal layer constraints and design rules (e.g., spacing, width).
- Avoid congestion and minimize routing delay.
- Prepare the design for DRC and layout generation.

Types of Routing:

1. Global Routing (Planning) – Estimates paths and allocates routing resources.
2. Detail Routing – Assigns exact metal tracks and vias to each net.

Qflow primarily uses detail routing with Qrouter after placement.

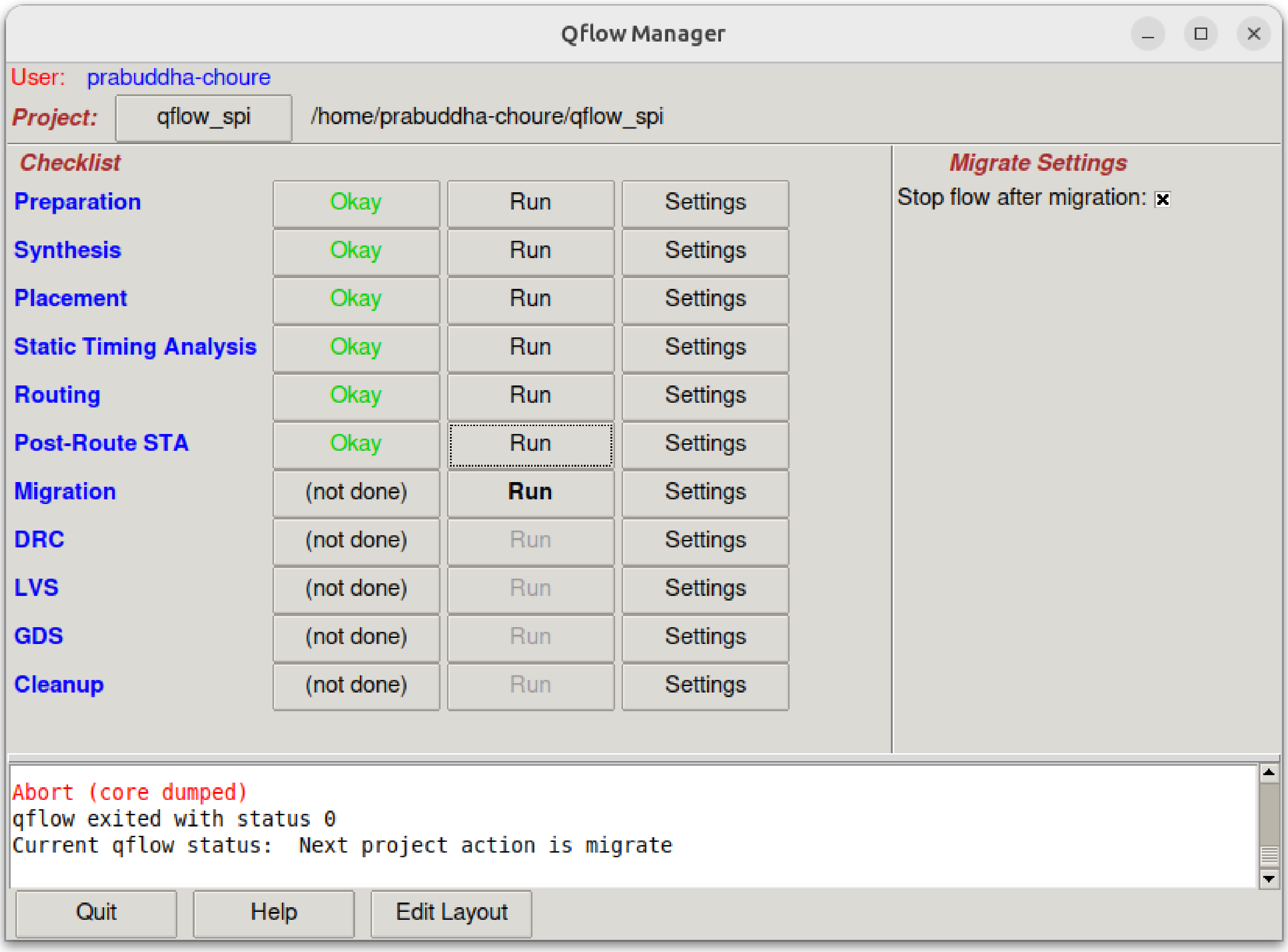


Post-Route STA:

Post-Route STA is the final timing verification step in the Qflow design flow, performed after routing is completed. It uses the actual routed netlist and layout information, which includes realistic wire delays and parasitic effects, making it more accurate than pre-route STA.

<u>Purpose of Post-Route STA:</u>

- Analyze timing with real interconnect delays after routing.
- Detect setup or hold violations that could not be seen earlier.
- Identify critical paths in the final physical layout.
- Ensure the circuit operates reliably at the target clock frequency.

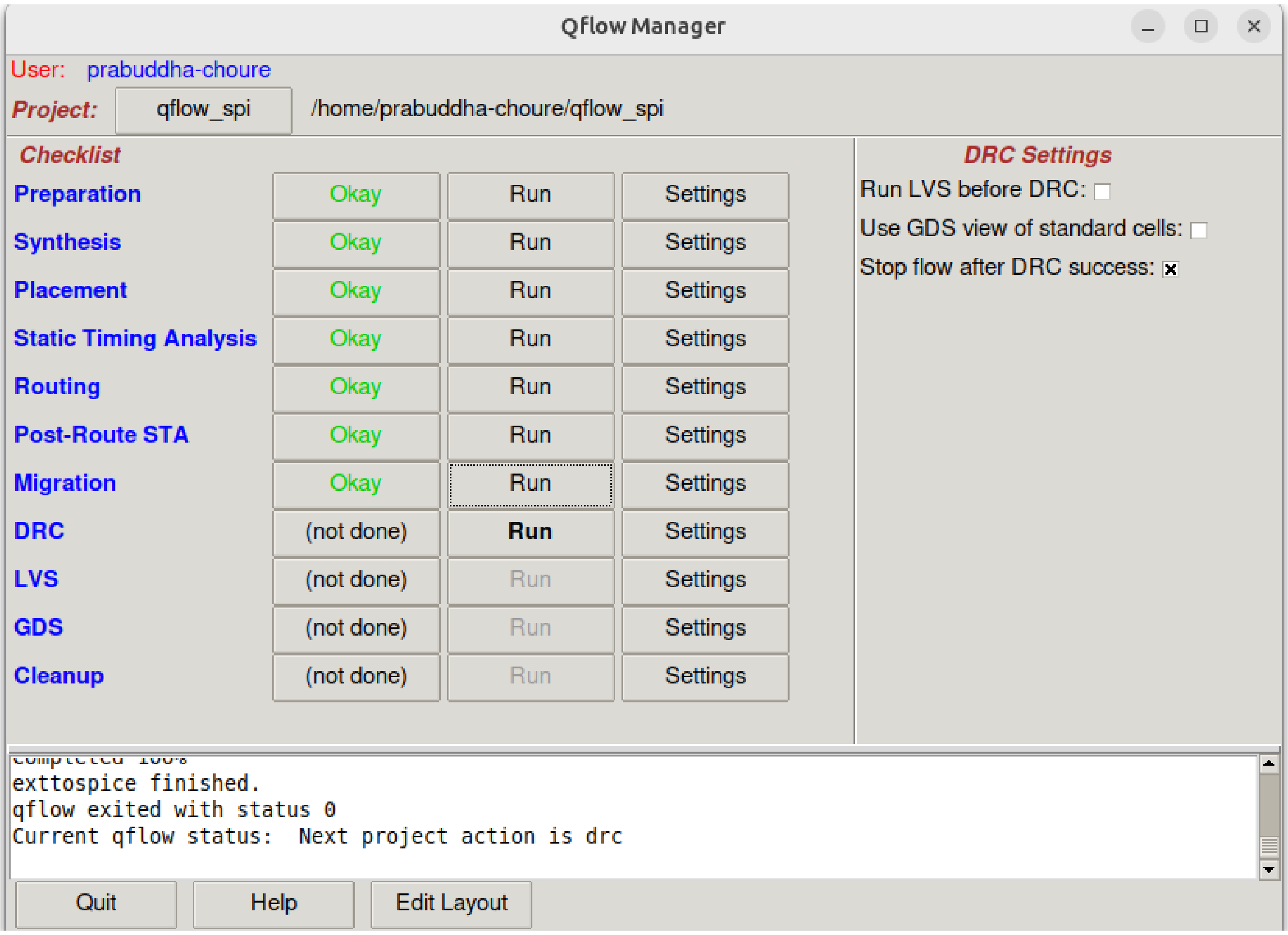


Migration:

In the Qflow digital design flow, Migration refers to the final stage where the physical layout and supporting files are prepared for fabrication or external tools. This step ensures that the design is fully DRC-clean, LVS-matched, and ready for export in standard formats like GDSII.

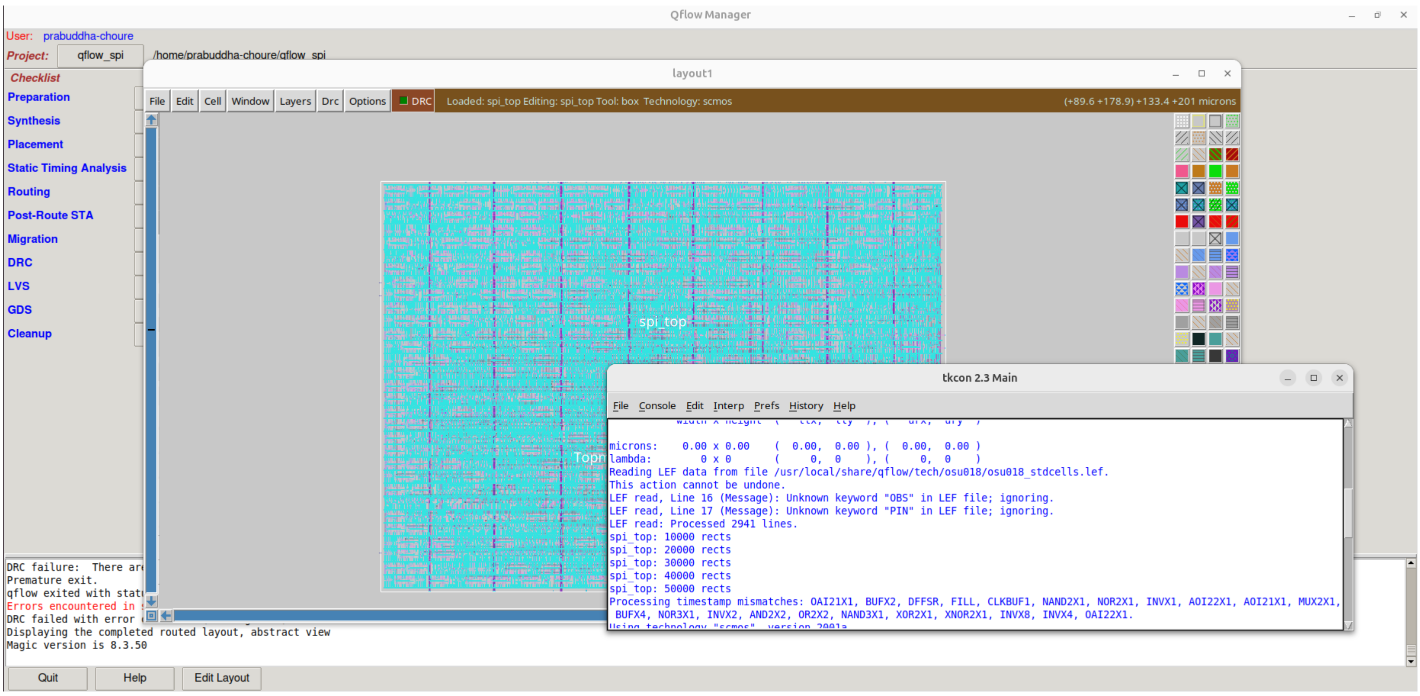
<u>Objectives of Migration:</u>

- Finalize the physical layout.
- Perform Design Rule Check (DRC) to ensure layout complies with foundry rules.
- Run Layout vs Schematic (LVS) to verify the physical layout matches the original netlist.
- Generate GDSII file for fabrication or further processing in commercial tools.



Design Rule Check (DRC):

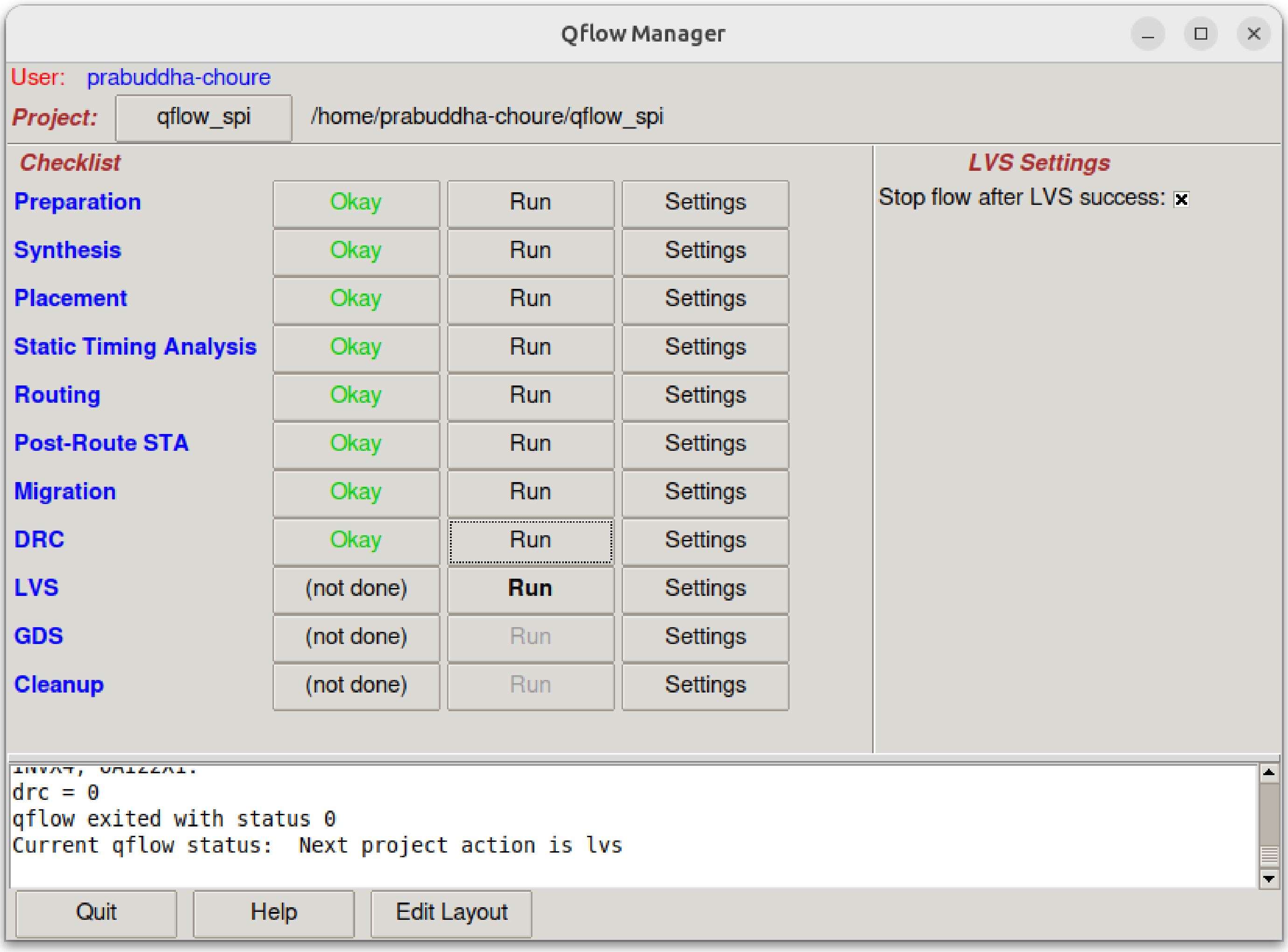
Design Rule Check (DRC) is a critical verification step in the VLSI physical design flow. It ensures that the layout follows the manufacturing constraints and geometrical rules defined by the semiconductor fabrication process.



In Qflow, DRC is performed using the Magic VLSI layout tool after the routing stage.

<u>Objectives of DRC:</u>

- Verify that the layout complies with foundry-specific design rules such as:
    - Minimum width and spacing of metal layers
    - Via enclosure rules
    - Overlap constraints
    - Well/implant layer spacing
- Identify and correct any physical violations before tape-out or fabrication.
- Ensure manufacturability and yield.



Layout Vs Schematic (LVS):

Layout vs Schematic (LVS) is a fundamental verification technique in the VLSI design process. It ensures that the physical layout of an integrated circuit (IC) exactly matches its intended logical design. The logical design is derived from RTL synthesis, while the layout is generated through placement and routing.

LVS compares two netlists:

1. Schematic Netlist – Generated from the RTL or gate-level description after synthesis.
2. Extracted Netlist – Obtained from the layout using a layout extraction tool (e.g., Magic in Qflow).

The LVS tool checks:

- Whether all devices (gates, transistors) in the schematic exist in the layout.
- If the connectivity (nets and pins) between those devices matches.
- That no extra or missing components are present in the layout.

A successful LVS indicates that the layout will behave functionally identical to the intended design.

Importance in VLSI Design:

- Detects errors introduced during physical design (e.g., routing mistakes, wrong pin connections).
- Validates design integrity before fabrication.
- Required by foundries as part of the design sign-off process.

Tool Used in Qflow:

In the Qflow open-source toolchain, Netgen is used for LVS comparison. It reads both netlists, matches elements, and reports mismatches, if any.

Graphic Data System (GDS):

GDSII, also known simply as GDS, is the standard file format used in the semiconductor industry to represent the physical layout of integrated circuits (ICs). It is the final output of the VLSI design flow and is used by foundries for mask generation and fabrication of chips.

GDSII stores information about:

- Geometries of layers (e.g., metal, polysilicon, diffusion)
- Cell definitions and hierarchy
- Coordinates and shapes of all layout features
- Text labels and layer types

Structure of GDSII:

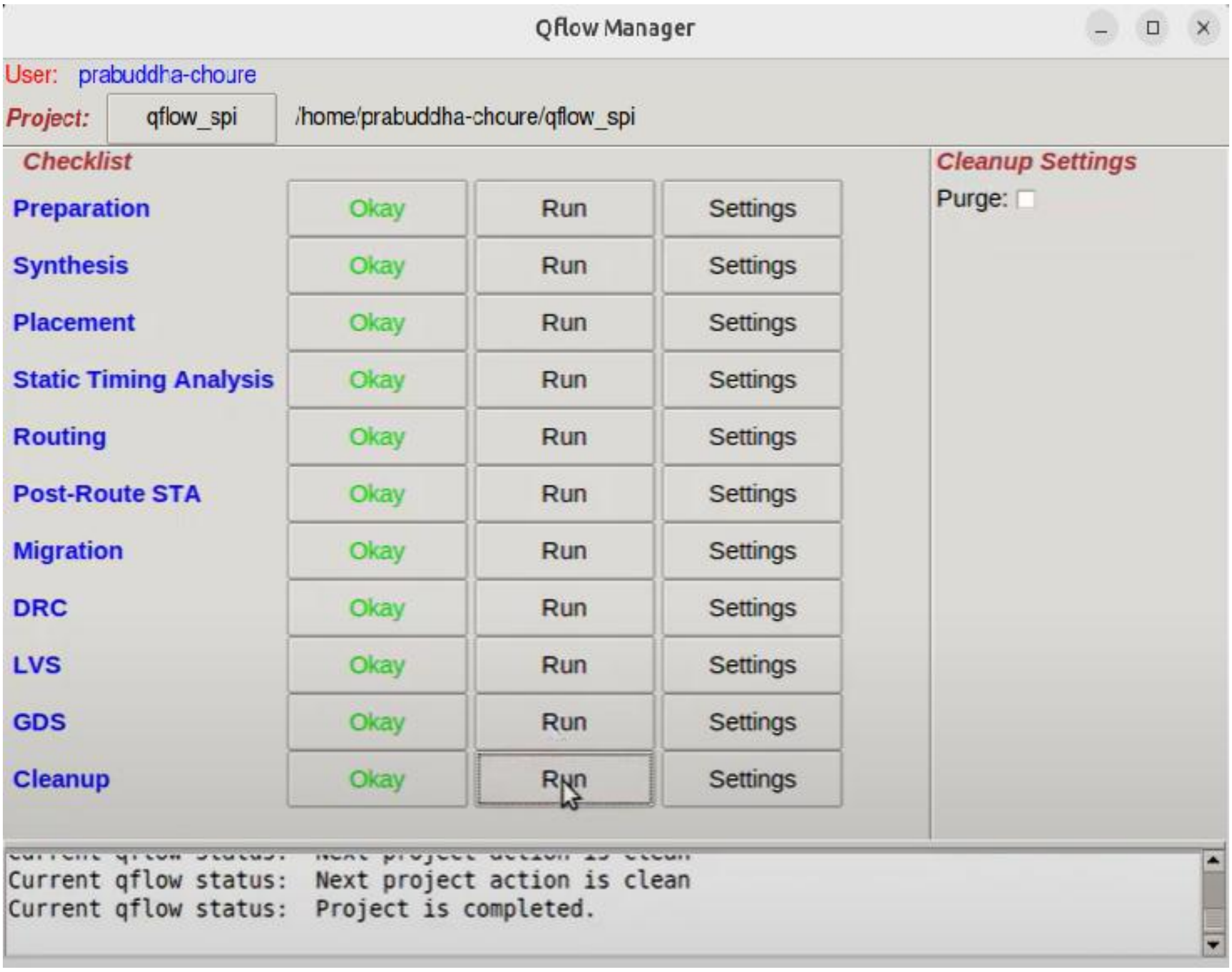
- Binary format (not human-readable)
- Layered format: each layer corresponds to a specific process level in fabrication
- Supports hierarchy: complex designs are built from reusable cell blocks (macros)

Purpose in VLSI Design:

- Acts as the final deliverable to fabrication facilities.
- Provides all necessary physical information to create photolithography masks.
- Ensures that the fabricated chip will match the designer's layout.

Final Output:

The screenshot below shows the Qflow Manager GUI at the final stage of the design process for the project qflow_spi. All stages in the digital VLSI flow have been successfully completed — including Preparation, Synthesis, Placement, STA, Routing, DRC, LVS, and GDS generation — as indicated by the green "Okay" status next to each step.



This confirms that the SPI module was fully processed from RTL to GDSII without any errors, and the design is now ready for fabrication or further integration.

Result & Discussion:

The project was successfully carried out in two phases, focusing on RTL design and physical implementation of a Serial Peripheral Interface (SPI) module.

In Project-I, the Verilog-based RTL was developed and synthesized using the Yosys tool in a Linux Ubuntu environment. The RTL-to-netlist conversion process was verified, and a correct, optimized gate-level representation was obtained.

In Project-II, the generated netlist was further processed through the Qflow toolchain to implement the physical design. Each stage of the flow — including synthesis, placement, routing, STA, DRC, LVS, and GDS generation — was completed successfully, as indicated by the Qflow GUI status. The final layout passed both DRC and LVS checks, ensuring functional and physical correctness.

The result demonstrates a full RTL to GDSII flow using open-source tools, validating the viability of Qflow and Yosys for academic and prototype-level VLSI design. The SPI module is now in a fabrication-ready state, with no violations or critical timing issues reported.