Centre of Excellence in Semicon

PROJECT:

Physical design implementation of SPI Protocol Using Qflow

NAME: J VENKATESH

COURSE: Maven Silicon Physical Design Internship

BATCH: B2B PDI 02

INDEX

1.INTRODUCTION

Qflow is an open-source digital synthesis flow designed to transform digital circuit designs—written in high-level behavioral languages like Verilog or VHDL—into netlist structures. These netlists can then be used as configuration data for FPGA targets, such as Xilinx or Altera chips, or adapted into formats compatible with chip fabrication technologies, ultimately enabling the creation of standard-cell CMOS layouts. While many digital synthesis flows exist, most are proprietary tools provided by EDA (Electronic Design Automation) vendors. Although these proprietary tools are not open-source, they are often distributed at no cost, typically on the condition that they are used with the vendor's specific FPGA hardware.

Qflow offers a complete, open-source alternative for the full synthesis of digital circuits, beginning with Verilog source code and ending in a physical layout through a streamlined and accessible process. It is tailored to support a simplified subset of Verilog features, making it well-suited for educational and entry-level design applications. The Qflow toolchain incorporates a combination of components from the CellLibrary project, graywolf, and various academic tools, as well as a few commercial-quality tools and wrappers. Some of these commercial tools do require licensing and may have limited availability. Nonetheless, all tools within the Qflow ecosystem adhere to an open standard interface, making the platform especially beneficial for students and academic engineering courses.

Qflow consists of the following tools:

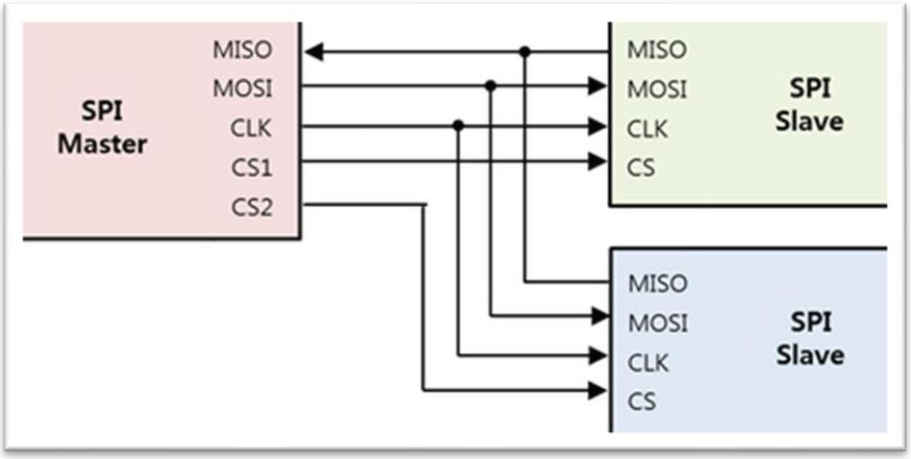
- Yosys: RTL Sythesis
- Qrouter: Routing
- Grayflow: Placement
- Magic:VLSI Layout Tool
- Netgen:LVS Layout vs Schematic
- OpenSta:Static Timing Analysis Tool

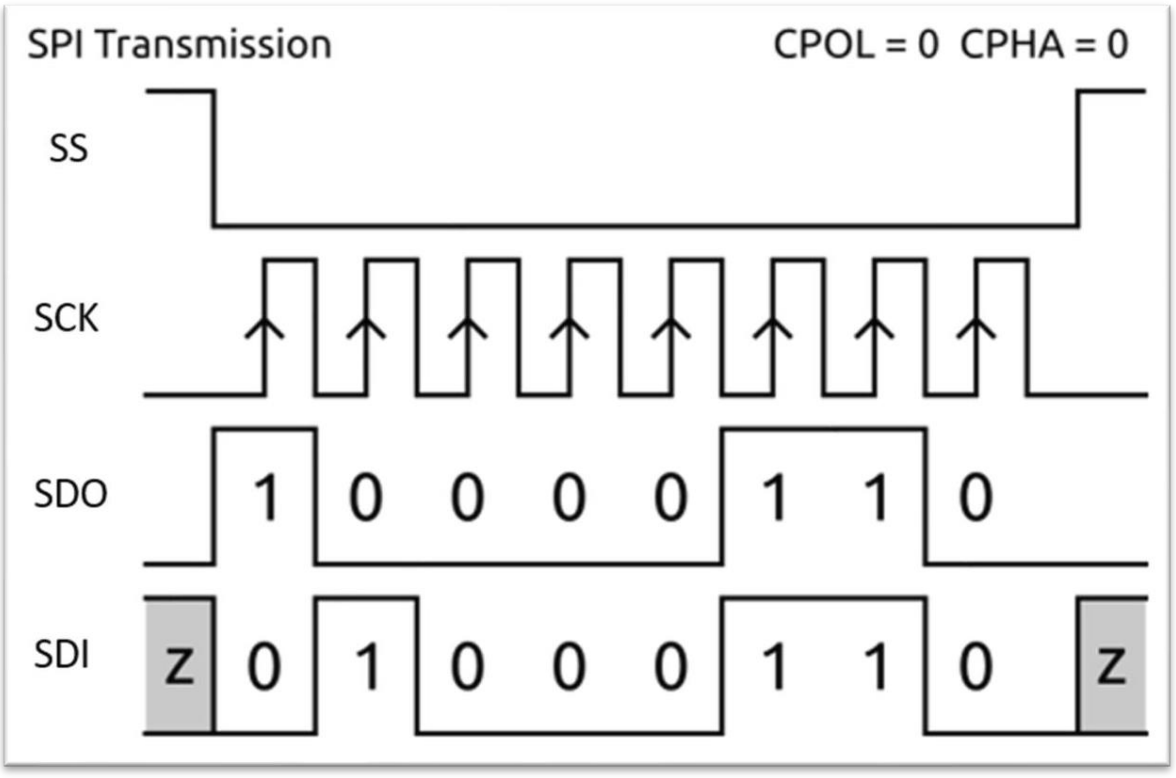
2.SERIAL PERIPHERAL INTERFACE(SPI)

Serial Peripheral Interface (SPI) is a widely used communication protocol designed for short-distance data exchange between electronic devices. Unlike asynchronous serial communication, SPI operates synchronously, meaning it uses a clock signal to coordinate data transfer. A typical SPI bus consists of four main lines: Master Out Slave In (MOSI), Master In Slave Out (MISO), the Serial Clock (SCLK), and Slave Select (SS). In this setup, there is always one master device that controls communication and one or more slave devices that respond. Each slave is selected individually through a dedicated SS line, although in larger systems, address logic may be used to simplify wiring.

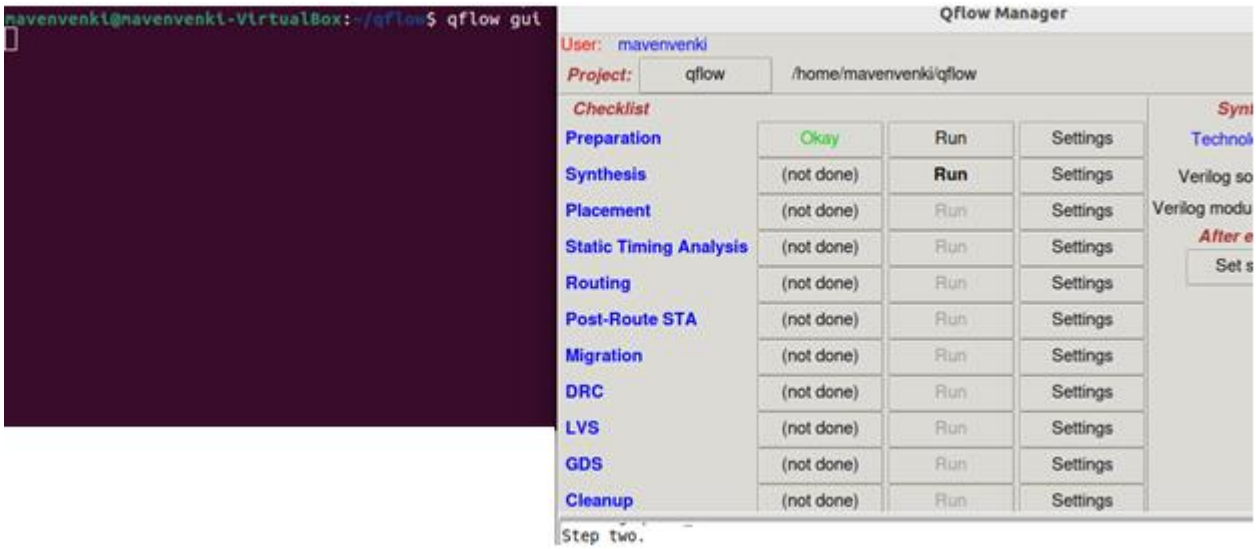
When the master wants to communicate with a particular slave, it begins by pulling that slave's SS line low. This action signals the start of communication, and data transfer begins. Only the master can initiate communication. During data transmission, both the master and the slave rely on the clock signal to synchronize the sending and receiving of data, with data being sampled on either the rising or falling edge of the clock. Proper timing is crucial, and the master's clock frequency must not exceed the maximum frequency supported by any connected device.

SPI supports four different modes of operation, based on two key parameters: CPOL (Clock Polarity) and CPHA (Clock Phase). CPOL defines the default idle state of the clock signal, while CPHA determines which clock edge is used to sample incoming data. For communication to be reliable, both the master and the slave must be configured with matching CPOL and CPHA settings. The most commonly used mode is CPOL = 0 and CPHA = 0, where the clock is idle low and data is sampled on the rising edge.


SPI
Master
MISO
MOSI
CLK
CS1
CS2
MISO
MOSI
CLK
CS
SPI
Slave
MISO
MOSI
CLK
CS
SPI
Slave


SPI Transmission
CPOL = 0 CPHA = 0
SS
SCK
SDO
1 0 0 0 0 1 1 0
SDI
Z 0 1 0 0 0 1 1 0 Z

3.INVOKING THE TOOL



Qflow GUI is a graphical user interface built to work with the Qflow RTL (Register Transfer Level) toolchain, providing a more accessible and user-friendly way to design digital circuits on Linux systems. It simplifies the workflow by offering a step-by-step visual representation of each stage in the synthesis process. This interface is especially helpful for users who prefer graphical tools over command-line interaction, making the powerful capabilities of Qflow easier to use and navigate.

Within the Qflow GUI, each sub-tool in the flow is integrated and can be executed individually. To move through the flow, each tool must be run only after the successful completion of the previous one. You can run each tool by simply clicking the “Run” option, and the GUI will indicate whether the execution was successful with an "OKAY" message or report a "FAIL" if an error occurs.

To begin using the Qflow GUI for a project, a few setup steps are required. First, you must select the appropriate technology file—commonly `osu018`. Then, choose the top module of your Verilog source file. For example, in the case of an SPI project, this would be "spi_top.v" (with the .v extension). After that, you’ll need to enter the top module name again, this time without the extension—just "spi_top".

Once these steps are complete, go to the **Preparation** tab and click on **Check Project Directory**. This will automatically create four folders within your project directory: **‘layout’, ‘source’, ‘log’, and ‘synthesis’**. These folders organize the output and intermediate files generated throughout the flow, helping you track and manage your design process efficiently.

4.SYNTHESIS

Synthesis is a crucial phase in the digital design process where hardware description languages (HDLs) like Verilog or VHDL are translated into a gate-level representation that can be implemented on physical hardware such as FPGAs or ASICs. In the Qflow design flow, this step is handled by the Yosys tool, and Qflow GUI offers a convenient graphical interface to oversee and execute the synthesis seamlessly.

One of the most important aspects to keep in mind before running the synthesis process is setting the correct project path. This step is essential to avoid errors and ensures that the tool can correctly access the source files and generate the required output. The working path acts as the root directory where the source, output, and other relevant folders are located. Once the path is correctly set, you can run the synthesis tool through the GUI. After initiating the process, it's important to wait for the tool to finish and produce the output, which indicates that the synthesis has been completed successfully.

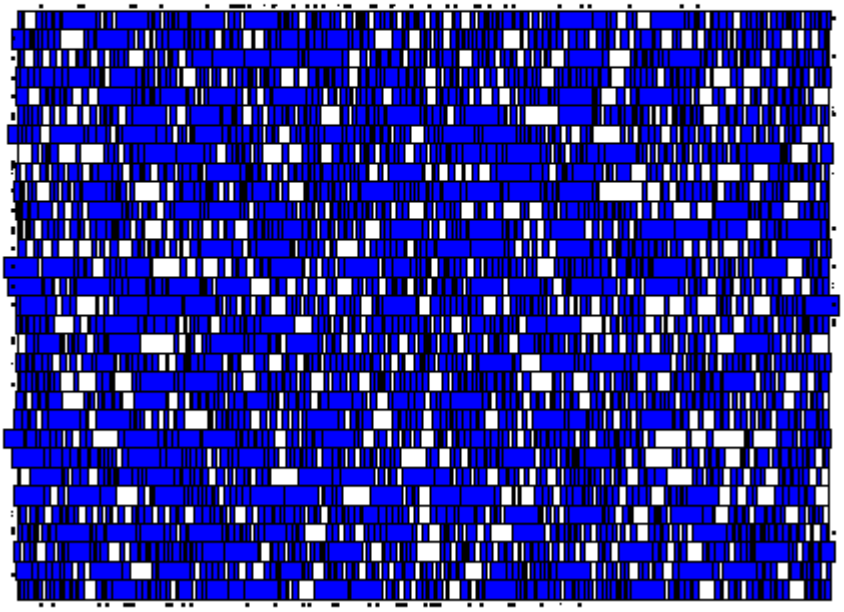
YOSYS

Yosys is a key open-source synthesis tool that plays a central role in the Qflow FPGA design toolchain. It is responsible for translating high-level hardware description languages, such as Verilog or VHDL, into a gate-level representation that can be used for logic placement and layout. As the synthesis engine within Qflow, Yosys not only handles the initial conversion but also performs essential tasks like optimization and technology mapping. This ensures that the resulting RTL designs are efficient and compatible with the next stages of the design flow.

One of the important outputs from Yosys is the BLIF (Berkeley Logic Interchange Format) file, which serves as a bridge to the later steps in the flow—such as logic placement, routing, and physical synthesis. Yosys is appreciated for its lightweight architecture, flexibility, and active development, which make it a powerful and cost-effective choice for FPGA and ASIC prototyping. The tool is also supported by a vibrant open-source community and enhanced through integration with plug-in tools like ABC, which complement Yosys by improving logic optimization and aligning output formats with specific design needs.

To get the most out of Yosys, proper installation and configuration are essential. Fortunately, Yosys is backed by comprehensive documentation and active forums, offering a wealth of resources to help users troubleshoot issues, optimize performance, and explore advanced features.

5.PLACEMENT



Placement is a vital step in the physical design process where all the standard cells from the netlist are positioned within the defined core area of the chip layout. This stage is not just about placing cells—it also involves design optimization to ensure better performance, reduced area, and efficient connectivity. In many cases, the tool performing placement also supports trial routing, which provides an early estimate of how the routing might look. This helps in identifying potential congestion issues or design inefficiencies before moving on to the final routing phase.

Placement in digital design is carried out by the tool using three distinct techniques, each focusing on a different design priority.

- The first is **timing-driven placement**, where the primary goal is to meet timing requirements, ensuring that data paths are optimized for speed.
- The second is **congestion-driven placement**, which prioritizes routing by minimizing wire congestion and making the layout more routable.
- The third technique is **power-driven placement**, where the focus is on reducing power consumption by optimizing the placement of cells accordingly.

The overall goals of placement are centered around achieving a high **Quality of Results (QoR)**. This includes optimizing timing, area, and power, as well as minimizing routing congestion and eliminating potential optimization hotspots. It is also essential to reduce **cell density** and **pin density** to ensure that the design remains manageable and efficient. A key objective during placement is to avoid any **Design Rule Check (DRC)** violations, ensuring that the layout adheres strictly to fabrication constraints.

In terms of placement-specific objectives, designers focus on metrics such as the **Physical Half Periphery (Hp)** and generating a **Placement Database File (Db File)** that records the cell positions.

The placement process itself is typically divided into three main stages:

- **Pre-Placement**, where initial preparation and checks are performed;
- **Placement**, where the actual positioning of standard cells takes place;
- **Post-Placement**, which involves refinement and verification to ensure the design is ready for the next steps in the flow.

GRAYWOLF

In the Qflow FPGA design toolchain, Graywolf serves as the dedicated placement tool responsible for arranging logic cells onto the FPGA fabric or ASIC layout. It uses a force-directed algorithm to optimize the spatial positioning of these cells, aiming to improve proximity and reduce signal delays. The core objective of Graywolf is to strategically position each cell in a way that minimizes both signal delay and routing congestion.

To achieve this, Graywolf analyzes the wire lengths and signal interactions between cells, determining the most efficient locations for placement. It also takes into account layout balance by managing cell density and interconnect distances, which ensures that the design remains routable with minimal congestion. Additionally, Graywolf supports region constraints and respects design hierarchy, which helps maintain structure and consistency throughout the layout process. By providing an optimized and balanced placement, Graywolf lays the groundwork for smooth and efficient routing in the later stages of the design flow.

6.STATIC TIMING ANALYSIS

Static Timing Analysis (STA) is a fundamental technique in digital design used to ensure that circuits operate correctly, even under varying conditions. Unlike traditional simulations, STA does not require specific input vectors. Instead, it systematically analyzes all possible signal paths within a design to evaluate delays and timing behavior. By examining these paths, STA checks critical parameters such as setup and hold times, clock skew, and propagation delays.

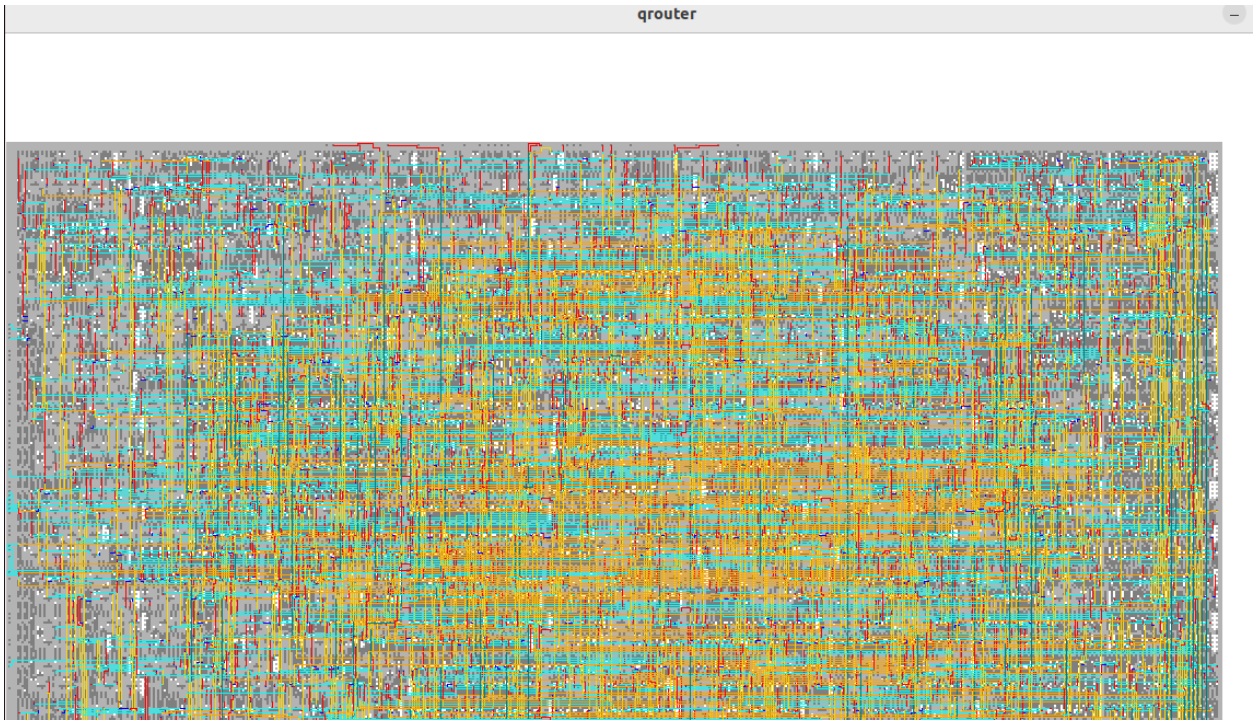
This process is essential for identifying and preventing timing-related issues that could lead to circuit malfunction, particularly in high-speed digital systems where even small delays can cause significant problems. One of the key strengths of STA is its ability to validate whether a design meets all necessary timing constraints without the need to simulate every possible input scenario. This makes it a powerful tool for ensuring both the performance and reliability of a digital design.

OpenSTA

OpenSTA is a free and open-source tool designed for performing Static Timing Analysis (STA) and is fully integrated into the Qflow toolchain. It allows designers to conduct thorough timing checks on their digital circuits, helping to ensure that the design meets required performance standards. OpenSTA offers both a graphical user interface (GUI) and command-line interface, making it flexible for different user preferences and workflows.

The tool includes several key features essential for effective timing analysis, such as path-based analysis, timing constraint checking, and delay estimation. These capabilities enable designers to evaluate whether the design meets critical timing requirements and identify potential issues early in the development process. By verifying that timing constraints are properly satisfied, OpenSTA plays a crucial role in optimizing a circuit’s overall performance and reliability.

7.ROUTING



In the Qflow FPGA design flow, routing is a critical phase that involves physically connecting the logic blocks with wires, much like building roads to link different cities. After the placement stage has determined where each logic element sits, routing takes over to establish the actual paths that electrical signals will follow across the chip. This step ensures that all required connections are made accurately and efficiently, without causing delays, interference, or signal conflicts.

The primary goals of routing are to avoid congestion, meet timing requirements, and ensure the layout remains efficient and error-free. It's a delicate balance that directly impacts the overall performance and functionality of the final hardware.

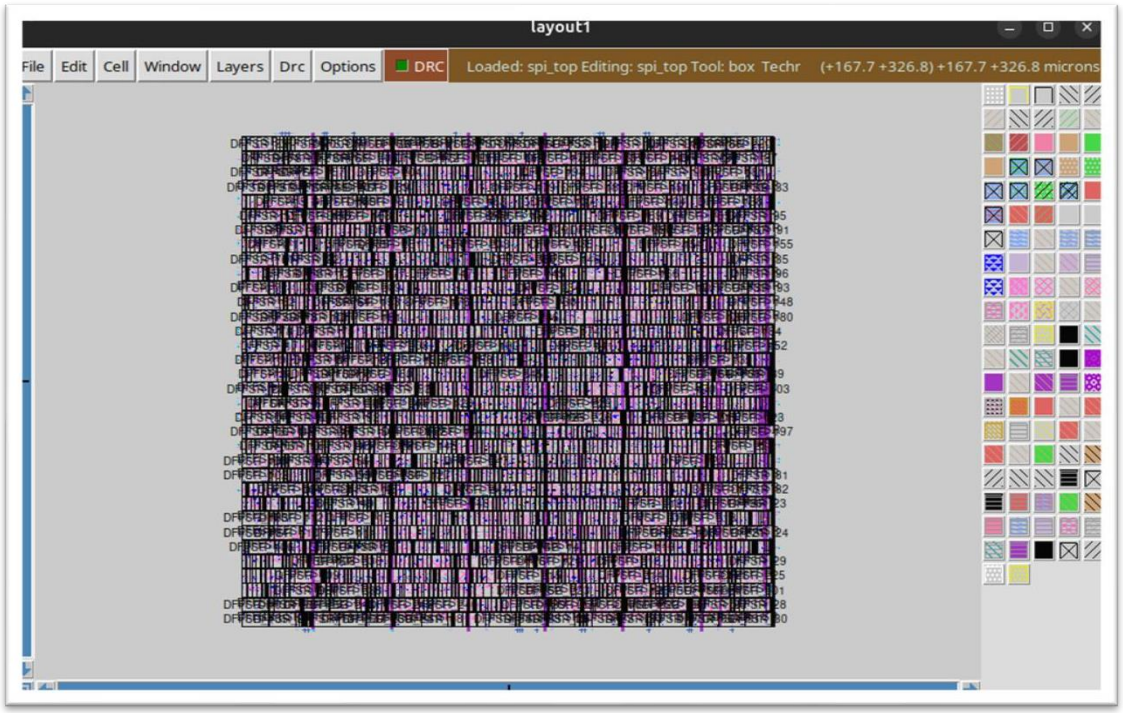
To carry out this task, Qflow relies on a tool called Qrouter. Qrouter is specifically designed to handle the routing process by creating wire paths that are both effective and within design constraints. It focuses on minimizing signal delays and parasitics-unwanted electrical effects that can degrade performance.

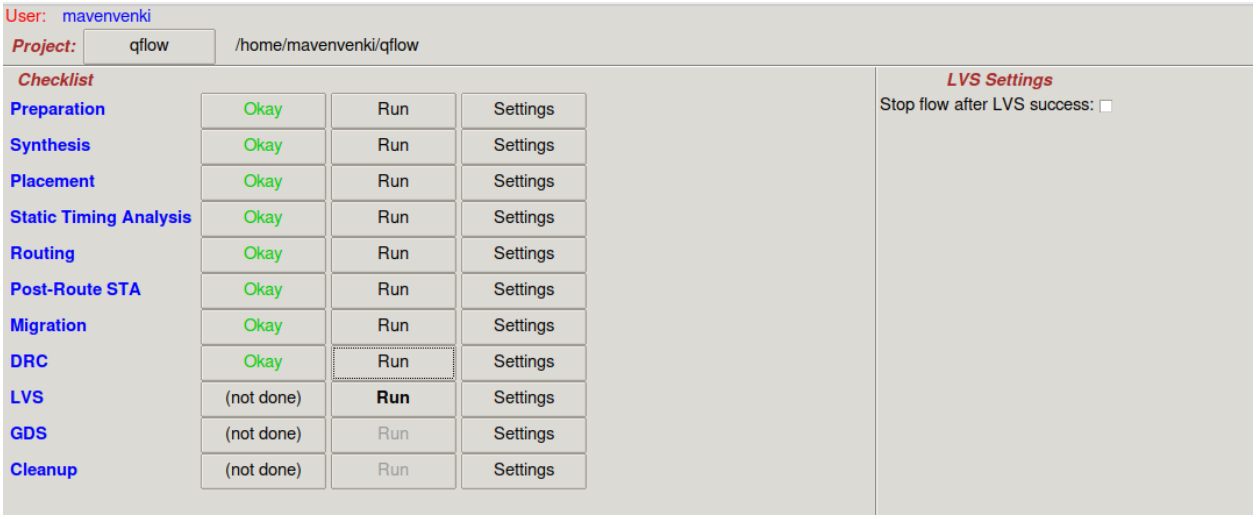
By ensuring that connections are made cleanly and correctly, Qrouter plays a vital role in transforming a logical circuit design into a fully functional physical implementation ready for real-world hardware.

8.DESIGN RULE CHECKING

```
mavenvenki@mavenvenki-VirtualBox: ~/qflow/layout
mavenvenki@mavenvenki-VirtualBox:~$ cd qflow
mavenvenki@mavenvenki-VirtualBox:~/qflow$ cd layout
mavenvenki@mavenvenki-VirtualBox:~/qflow/layout$ magic spi_top.mag
```

```
tkcon 2.3 Main
File  Console  Edit  Interp  Prefs  History  Help
    count [total]          count error tiles in each cell under box
    euclidean on|off enable/disable Euclidean geometry checking
    find [nth]      locate next (or nth) error in the layout
    help                   print this help information
    off                    turn off background checker
    on                     reenable background checker
    status               report if the drc checker is on or off
    style                set the DRC style
    printrules [file]      print out design rules in file or on tty
    rulestats              print out stats about design rule database
    statistics             print out statistics gathered by checker
    why                    print out reasons for errors under box
wrong # args: should be "magic::drcupdate option"
% drc why
No errors found.
```





Design Rule Check (DRC) is a crucial step in the FPGA design process, ensuring that the physical layout of a circuit complies with specific manufacturing rules. These rules are put in place to guarantee that the design can be reliably fabricated without defects. During DRC, the tool verifies critical aspects such as wire widths, spacing between components, and the alignment and dimensions of various layout elements.

Any violation of these design rules can lead to serious issues, including defects or complete circuit failure, especially when the design is fabricated. By identifying and flagging these problems early in the process, DRC plays a vital role in preventing costly errors and ensuring the overall success and reliability of the final hardware.

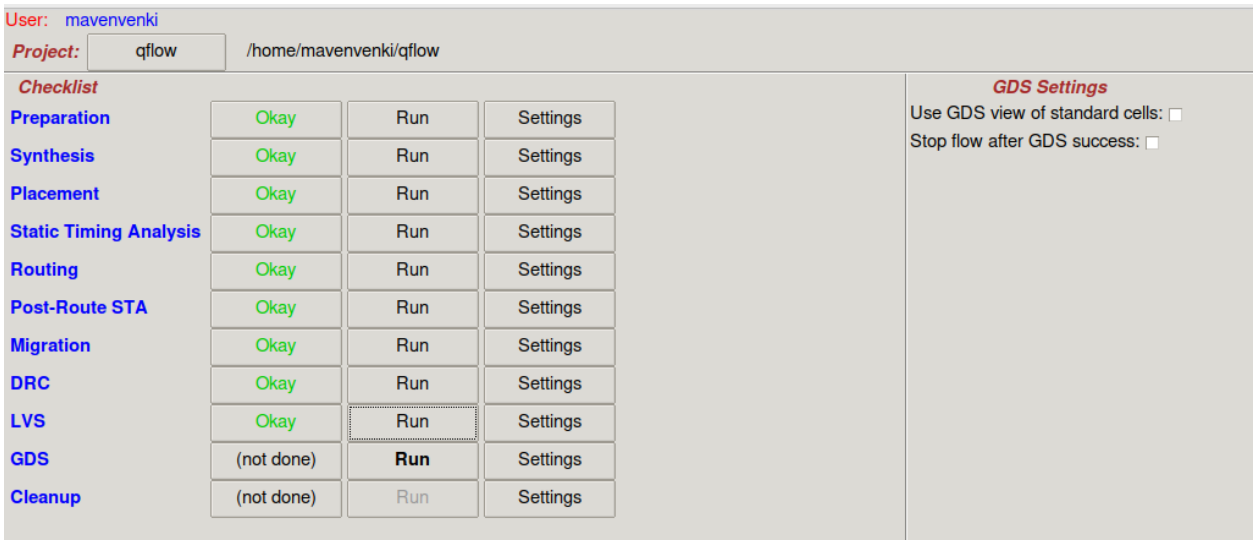
MAGIC TOOL

In the Qflow design flow, Magic is the primary tool used to perform Design Rule Checks (DRC). It plays a vital role in identifying layout violations by visually highlighting errors directly within the layout, making it easier for designers to understand and correct issues. Magic not only flags the problem areas but also provides detailed feedback through its console, helping to streamline the debugging process during the physical verification stage.

The workflow for fixing DRC errors in Qflow using Magic is quite straightforward and effective. First, you load the design into Magic, allowing you to view the layout visually. Then, by using the command ‘drc count’, you can quickly determine how many design rule violations are currently present. Once the violations are understood, you can navigate to the exact location of each error within the layout. From there, manual corrections can be made by adjusting wire widths, spacing, or realigning components as needed.

After making the necessary changes, it’s important to recheck the layout and rerun the DRC process to confirm that the violations have been resolved and no new ones have been introduced. This iterative process helps ensure the layout complies fully with fabrication rules, thereby reducing the risk of manufacturing defects. Additionally, designers should regularly save progress and take snapshots of working versions, as fixing one violation may occasionally introduce another elsewhere. Magic’s integration in Qflow makes this process more visual and interactive, offering a clear path to a clean, manufacturable layout.

9.LAYOUT vs SCHEMATICS (LVS)



Layout Versus Schematic (LVS) is a crucial verification step in the physical design process that ensures the physical layout of a circuit faithfully represents its original schematic, or logical design. This step is essential in catching errors that could compromise the functionality or manufacturability of the final chip.

LVS works by comparing netlists—which are lists of components and their interconnections—generated from both the schematic and the physical layout. It checks whether the layout implementation is logically equivalent to what was originally intended in the schematic. Through this comparison, LVS can identify a variety of issues such as missing or extra components, incorrect wiring, or wrong connections between logic elements.

This verification step is especially important because even a small mismatch between the schematic and layout can result in circuit failure or significant fabrication problems. Ensuring consistency at this stage guarantees that the design integrity is preserved all the way through to manufacturing.

In the context of the Qflow FPGA design flow, LVS tools play a vital role by comparing the layout's netlist with the original design netlist to confirm that everything is connected exactly as planned. This not only prevents costly post-fabrication errors but also helps catch any mistakes introduced during layout generation, making LVS an indispensable part of the digital design process.

10.GDS (GRAPHIC DATA SYSTEM)

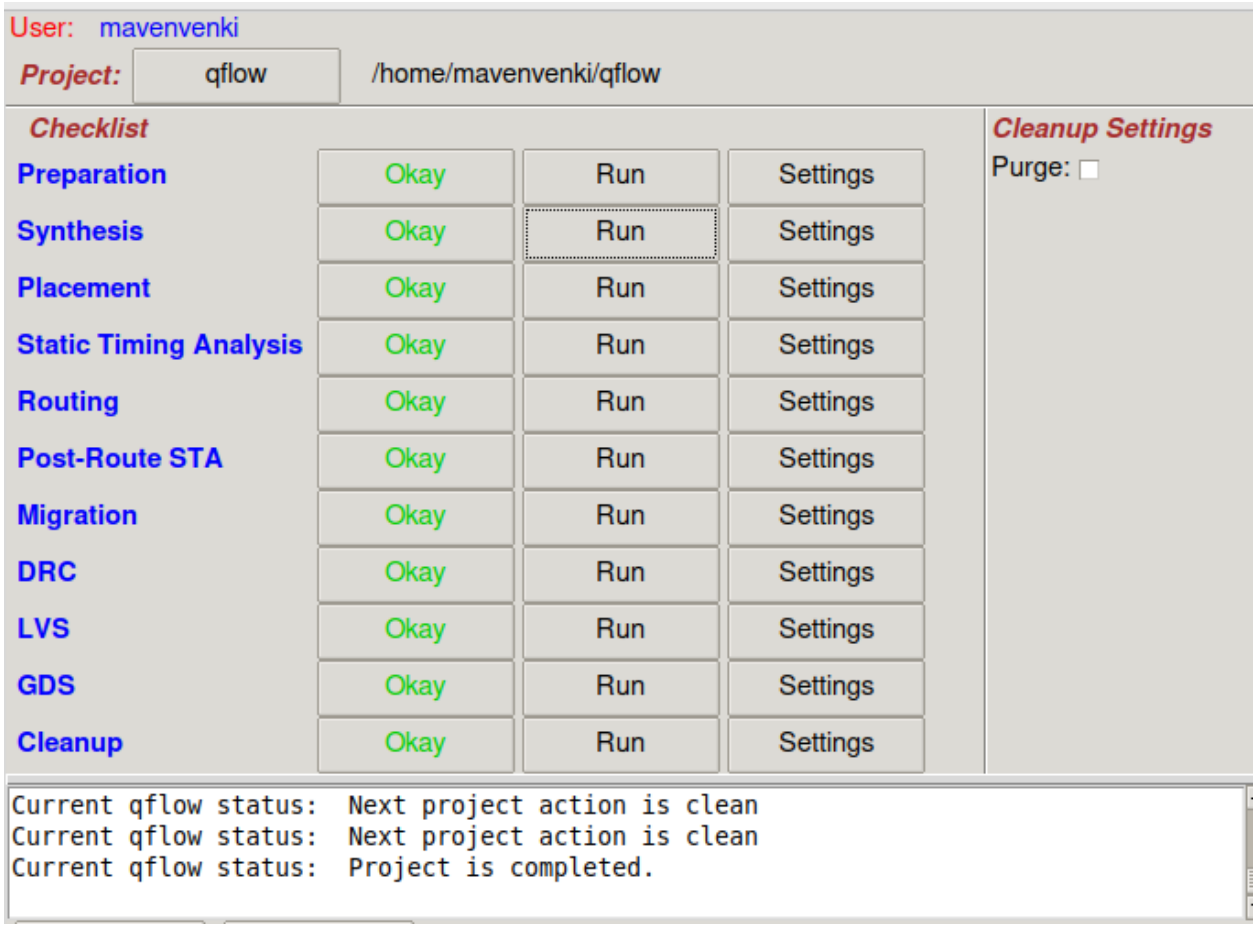
GDS (Graphic Data System) is a widely used file format in both FPGA and VLSI design, serving as the final representation of a circuit's physical layout. Essentially, it acts as the blueprint for fabrication, containing detailed data about the circuit's geometries, component placements, and interconnections. This file is crucial because it ensures that the design can be precisely interpreted by the tools used in chip manufacturing.

The GDS format plays a vital role in making the physical layout easily sharable across design teams and compatible with various EDA tools, streamlining collaboration and reducing the chances of miscommunication or error. It also helps minimize manufacturing defects by offering a standardized, accurate depiction of the final layout.

Within the Qflow workflow, tools like Magic are responsible for generating the GDS file. Once generated, this file becomes the basis for creating the photolithographic masks used during chip fabrication.

The importance of GDS cannot be overstated. It represents the final step before fabrication, effectively bridging the gap between digital design and physical manufacturing. By ensuring that the physical layout exactly reflects the logical intent, the GDS file provides the confidence needed to proceed with building actual hardware, making it a cornerstone of the VLSI and FPGA design process.

11.CONCLUSION-OUTPUT



This section highlights the completion status of the Qflow design flow, providing a clear overview of the entire process from start to finish. It includes a screenshot of the Qflow Manager, where all key stages—beginning with **synthesis** and ending with **GDS generation**—are visibly tracked. Each essential step in the flow, including **placement**, **Static Timing Analysis (STA)**, **routing**, **Design Rule Check (DRC)**, **Layout versus Schematic (LVS)**, and the final **GDS export**, has been executed successfully.