Introduction

TCL (Tool Command Language) is a powerful scripting language used to automate tasks in tools like Xilinx Vivado. It is a core component of the Vivado Design Suite, enabling users to control the tool flow, perform design analysis, and manage constraints efficiently, all through commands instead of using the graphical interface.

TCL integrates seamlessly with Xilinx-specific commands and supports both Synopsys Design Constraints **(SDC)** and Xilinx Design Constraints **(XDC)** formats, making it an essential tool for efficient and repeatable FPGA design workflows.

Key Features of Tcl in Vivado

- **Versatility**: Used for netlist queries, timing constraints, tool execution (e.g., `place_design`, `synth_design`), and custom report generation.
- **Compatibility**: Supports Tcl 8.5 commands and extends with Vivado-specific commands for FPGA design tasks.
- **Object Handling**: Allows direct manipulation of design objects (e.g., cells, pins) using Tcl commands.

Benefits

- **Design Constraints**: Essential for timing and performance optimization (e.g., `create_clock`, `set_input_delay`).
- **Debugging & Analysis**: Enables static timing analysis (STA) and custom reports.
- **Cross-Platform**: Works on both Linux and Windows.
- **Industry Standard**: Compatible with third-party EDA tools.

Workflows

1. **Project-Based Flow**:
    - Uses project infrastructure (e.g., `create_project`, `launch_run`).
    - Suitable for GUI and script integration.

2. **Project-Less Flow**:
    - Script-driven (e.g., `read_verilog`, `synth_design`).
    - No project files; manual report/checkpoint management.

Creating a New Vivado Project with TCL

Here is a simple TCL script to create a new project, add sources, and run synthesis and implementation**:**

- **Creating a new project**

```
create_project my_project ./my_project -part xc7a35ticsg324-1L
```

- **Set target language**

```
set_property target_language VHDL [current_project]
```

- **Add design source files**

```
add_files ./src/top.vhdl
```

- **Add constraint file**

```
add_files ./constraints/top.xdc
```

- **Set the top module**

```
set_property top top [current_fileset]
```

- **Run synthesis**

```
launch_runs synth_1
wait_on_run synth_1
```

- **Run implementation**

```
launch_runs impl_1
wait_on_run impl_1
```

- **Generate bitstream**

```
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
```

- **Open the implemented design**

```
open_run impl_1
```

(Make sure to replace the file paths (./src/top.vhdl, etc.) and the part number with the ones you're using).

Execution Methods

- **GUI**: Enter commands in the Tcl Console or run scripts via Tools > Run Tcl Script.
- **Batch Mode**: Execute scripts without GUI:

```
vivado -mode batch -source script.tcl
```

Key Commands

- **Design Queries**:

```
get_cells -hier -filter {lib_cell == BUFG}
```

- **Timing Constraints**:

```
create_clock -period 10 -name clk [get_ports clk_pin]
```

- **Implementation**:

```
launch_runs impl_1
wait_on_run impl_1
```

TCL Commands for Simulation

Add this to your TCL script after adding design files:

- **Add the testbench file**

```
add_files -fileset sim_1 ./tb/top_tb.vhdl
```

- **Set the simulation top module**

```
set_property top top_tb [get_filesets sim_1]
```

- **Set simulation language**

```
set_property target_simulator XSIM [current_project]
set_property simulator_language VHDL [current_project
```

- **Launch simulation (behavioral)**

```
launch_simulation
```

- **Run simulation for a certain time (e.g., 1 microsecond)**

```
run 1 us
```

Notes

- The simulation is typically done **before** synthesis to verify logic correctness.
- **Runs the simulation**

```
launch_simulation
```

- **Controls how long the simulation executes**.

```
run <time>
```

- **Logs all signals for waveform viewing**.

```
log_wave -recursive *
```

- **Launches the waveform viewer if you are in GUI mode.**

```
start_gui
```

Logging & Documentation

- **Journal/Log Files**: Record commands (vivado.jou) and tool messages (vivado.log).

Conclusion

- Tcl in Vivado enhances automation, customization, and debugging for FPGA designs.
- Project-based flow is recommended for beginners, while project-less offers flexibility for advanced users.