Lattice Prompt - Automating Radiant with MCP Server and Claude Skills User Guide

September 2026

Copyright

Trademarks

Disclaimers

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
`.` `.` `.`	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

Overview

Lattice Prompt is an AI-assisted FPGA development solution that connects supported AI assistants to Lattice Radiant® software through Model Context Protocol (MCP). It provides access to Lattice-specific design knowledge and automation capabilities that enable users to perform FPGA development designs using natural language interactions.

This guide describes the configuration and use of the Radiant MCP Server and Skills, specialized components within Lattice Prompt that automate Lattice Radiant. This AI extension package lets you drive Lattice Radiant FPGA tools with natural language instead of memorizing Tcl commands or CLI flags. It comprises an MCP server and Claude Code skills:

Piece	What It Is	What It Gives an LLM
Radiant MCP Server (`lattice-radiant-mcp`)	An MCP server that keeps a persistent `radiantc` Tcl session alive, locally or over SSH/LSF.	Fine-grained, individually-callable tools for project-, UDB-based designs, compilation, timing analysis, programming file generation and others - usable by a MCP-capable client (Claude Code, Claude Desktop, Cursor, etc.).
Claude Code Skills (`/lattice-radiant-automation`, `/lattice-radiant-webdocs`)	A guided, persona-aware workflow for simulating and compiling a Radiant project, checking timing closure, and answering Lattice device and tooling questions from authoritative, version-matched sources.	An interactive "simulate and compile my design" experience – asks setup questions, writes a Tcl compile script, runs simulation, synthesis, place & route, bitstream generation, and reports pass/fail timing. Documentation questions are answered from the local Radiant install and official data sheets.

Usage Notes

- Available flow: simulation > synthesis > map > place & route > timing analysis > bitstream generation.
- Simulation applies to Radiant `.rdf` projects only.
- Concurrent projects are isolated. Each Claude session maintains its own independent Radiant process and job state, so multiple users or multiple Claude sessions belonging to the same user can compile different projects simultaneously without interfering with one another.

How to Install

Prerequisites

- Windows 10 / 11
- Python 3.10 or later
- Claude Desktop (latest version)
- Lattice Radiant Software installed locally or reachable via SSH / LSF

Package Contents

Note:

If you do not have rights to set/modify environment variables on Windows, set them in a PowerShell prompt (e.g. `$env:LM_LICENSE_FILE = "C:\licenses\license.dat"`) before running the target commands.

File	Description
`lattice-prompt-installer_<version>.zip`	Overall installer package
`lattice-radiant-skills.plugin`	Claude skills: batch simulate / compile automation and web documentation search
`lattice-radiant-mcp.plugin`	Claude plugin that registers the Radiant MCP server
`install.py`	Python installer script
`lattice_radiant_mcp-<version>.zip`	MCP server source code
`Lattice-Prompt-User-Guide.pdf`	This user guide

Recommended: Installer Package

1. Extract the Lattice Prompt installer package into a folder of your choice.
2. Run the installer from a command prompt or PowerShell window in that folder:

   ```
   cd lattice-prompt-installer/
   python install.py
   ```

   The installer performs all steps automatically:

   a. Step 1: Python check – If Python is not on the system PATH, invoke it by its full path.

   b. Step 2: MCP server – Installs the `radiant_mcp` Python package from `lattice_radiant_mcp-*.zip` via `pip install --upgrade`.

   c. Step 3: Skills – Copies skills into `~/.claude/skills/`.

   d. Step 4: Plugin registration – Writes plugin entries and MCP server config directly into Claude's configuration files (`~/.claude/`).
3. Restart your AI client e.g. Claude fully (quit and reopen) to load the installed plugins and skills.

If the installer fails, install the `.plugin` files manually via **Claude Desktop > Settings > Plugins > Upload plugin**, one file at a time.

Plugin File	What It Installs
`lattice-radiant-skills.plugin`	`/lattice-radiant-automation` and `/lattice-radiant-webdocs` skills
`lattice-radiant-mcp.plugin`	`lattice-radiant-mcp` MCP server entry and tool permissions

Upgrading

Re-run `install.py` from a new installer package; the plugins will be overwritten with the new versions.

Prompt Examples

1. **RDF project**

   ```
   "Simulate and compile the Radiant project C:/path/to/project1/proj.rdf using
   Radiant 2026.1 on this machine."
   ```

2. **UDB design**

   ```
   "Compile the Radiant UDB design C:/path/to/design1/top.udb using Radiant 2026.1 on
   this machine."
   ```

Radiant MCP Server

Note:

This section is provided for reference only. No user action is required to use the tool.

MCP server for Lattice Radiant FPGA software (`lattice-radiant-mcp`). The MCP host where this server runs and the execution host (where `radiantc` actually runs) do not need to be the same machine or the same OS.

Configure Radiant's Location

The server needs to know where `radiantc` is installed. There are ways to set this (first match wins):

The easiest approach is to let Claude detect it automatically. In a Claude Code session, ask:

```
Use detect_radiant_installations to find Radiant on this machine,then
configure_radiant to save the path.
```

Or set it explicitly in `lscc-radiant-mcp.json` (create in the project directory):

```
{ "radiant_exe": "C:\\lscc\\radiant\\2026.1\\bin\\nt64\\radiantc.exe" }
```

Linux path example:

```
{ "radiant_exe": "/tools/lattice/radiant/2026.1/bin/lin64/radiantc" }
```

Note:

This section is provided for reference only. No user action is required to use the tool.

Priority	Method	How
1	Environment variable	Set `RADIANT_EXE` in the `mcp.json` env block
2	Config file	`lscc-radiant-mcp.json` in the project directory, or the user config path

Verify

Use `check_environment` to verify the `lattice-radiant-mcp` setup.

Expected: a response with `"ready": true`. If anything is misconfigured, `resolution_options` in the response will tell you exactly what to fix.

Troubleshooting

Symptom	Likely Cause	Fix
No radiant tools visible in Claude	`mcp.json` missing or session not restarted	Verify `~/.claude/mcp.json` contains the `lattice-radiant-mcp` entry; restart Claude Code
Permission prompt on every tool call	Allow-list not set	Add `mcp__lattice-radiant-mcp__*` to `permissions.allow` in `settings.json`
`radiantc` not found	Wrong or missing Radiant path	Run `detect_radiant_installations` in a session, then `configure_radiant(exe_path="...")`
`needs_user_input: true` from `detect_radiant_installations`	Radiant path or license vars not yet configured	Ask Claude to call `configure_radiant` with the missing values (`exe_path`, `lm_license_file`, `salt_license_server`) for the appropriate context
Installer warns a plugin failed to load	Wheel installed into a different Python environment, or a corrupted install	Re-run `install.py` from the intended (activated) environment; if the warning persists, capture the message and contact the release owner
MCP server fails to start, or tools target the wrong Python	`install.py` was run from a different environment than intended	Activate the correct environment and re-run `python install.py`
License checkout failure at Radiant startup	`LM_LICENSE_FILE` or `SALT_LICENSE_SERVER` not set	Call `configure_radiant(lm_license_file="...", context="local")` / `configure_radiant(lm_license_file="...", context="remote")`, or set the environment variable
SSH / bsub connection errors	Remote config incomplete	Run `check_environment(test_connectivity=True)` and follow `resolution_options`
Claude stops polling a long-running process (e.g., PAR)	Claude loses track of job status	Prompt Claude for the result via the chat interface and it should resume.

Configuration Reference

Environment Variables

Variable	Default	Description
RADIANT_EXE	(see priority table)	Path to the radiantc executable
LM_LICENSE_FILE	—	Lattice tool license file or server string (e.g. 1234@server)
SALT_LICENSE_SERVER	—	Questa/Mentor simulation license server - required only for simulation
RADIANT_MCP_CONFIG	—	Explicit path to a JSON config file
RADIANT_TIMEOUT	3600	Default command timeout in seconds
RADIANT_SEARCH_ROOT	—	Extra directory for detect_radiant_installations
RADIANT_REMOTE_MODE	—	Override remote mode: local, ssh, bsub, ssh_bsub
RADIANT_SSH_HOST	—	SSH host override
RADIANT_SSH_USER	—	SSH username override
RADIANT_SSH_IDENTITY	—	SSH private key path override
RADIANT_SSH_PORT	—	SSH port override
RADIANT_BSUB_QUEUE	—	LSF queue override
RADIANT_BSUB_PROJECT	—	LSF project override
RADIANT_MCP_LOG_LEVEL	INFO	Log verbosity: DEBUG, INFO, WARNING, ERROR

CLI Options

python -m radiant_mcp accepts these flags:

Flag	Default	Description
--transport	stdio	Transport: stdio, streamable-http, or sse
--host	127.0.0.1	Bind host (HTTP/SSE only)
--port	8080	Listen port (HTTP/SSE only)
--path	/mcp	URL path (streamable-http only)
--stateless	On	Stateless HTTP – no session IDs. Pass --stateful to opt out
--list-tools	Off	Print all registered tools and exit

Tools Reference

All tools return at minimum: `{ "success": bool, "output": str, "warnings": [str], "elapsed_ms": float, "timestamp": str }`. Several tools return additional fields documented below.

Note:

Always check `warnings` in addition to `success`. `success` matches a fixed set of known Radiant error patterns and can miss new failure messages. `warnings` extracts every line containing an error- or failure-sounding word – review it after any long-running command.

Knowledge Base

Tool	Description
`search_documents(query, ip_name?, radiant_version?, doc_type?, file_format?, include_visual?, limit?)`	Redirects to the `/lattice-radiant-webdocs` skill when that plugin is installed (preferred). Returns an empty result when the skill is not installed. Parameters are forwarded as context in the redirect.

Note:

Preferred path: Install the `lattice-radiant-skills` plugin for best results. `search_documents` will redirect every call to `/lattice-radiant-webdocs`, which provides richer, version-matched documentation lookup using the local Radiant install and official data sheets.

Without the plugin: `search_documents` returns an empty result and instructs you to install `lattice-radiant-skills`.

Parameter Guide for `search_documents`:

Parameter	When to Set	Notes
`query`	Always (required)	Natural language; include IP name and concept
`ip_name`	User names a specific IP	Lowercase with underscores: `ddr3_phy`, `mipi_dphy`, `pcie`, `serdes`, `usb`
`radiant_version`	User specifies a release	e.g. `"2026.1"`
`doc_type`	User names a document category	`"guide"`, `"datasheet"`, `"architecture"`, `"test_plan"`, `"specification"` – advisory only
`file_format`	User explicitly names a format	`"pdf"`, `"docx"`, `"xlsx"`, `"pptx"` – hard filter; omit unless specified
`include_visual`	Set `False` for text-only queries	Skips figure/diagram index; when redirecting to the skill this is forwarded as context
`limit`	Broad topics	Default 30; forwarded as context when redirecting to the skill

Session Management

Tool	Description
`start_radiant(exe_path?, timeout?)`	Start the Radiant Tcl session. Safe to call repeatedly — no-op if already running
`stop_radiant()`	Stop the session gracefully
`get_session_status(log_lines?)`	Health check + auto-restart if the session has died
`detect_radiant_installations (extra_search_roots?)`	Scan common install directories and the Windows registry
`get_radiant_config()`	Show how `radiant_exe` is currently resolved
`configure_radiant(exe_path?, install_dir?, config_file?)`	Persist the Radiant executable location
`configure_remote_execution( mode, ssh_*, bsub_*, config_file?)`	Persist SSH / bsub settings
`get_remote_execution_config()`	Show the active remote execution mode
`test_remote_connection(timeout?)`	Verify SSH and/or bsub connectivity
`check_environment(test_connectivity?)`	Pre-flight check of Radiant path, SSH, and bsub configuration

Project Flow

Tool	Description
`compile_project(project_path, force?)`	Full compile flow in one call. Starts Radiant if needed, then runs `open_project > run_synthesis > run_map > run_par > run_export` in sequence. Stops at the first failing stage and reports which one failed. On success, returns the same `utilization_summary`, `par_summary`, and `timing_summary fields` as `run_par`, plus a `stages` list with per-stage results. Pass `force=True` to force all stages even when already up-to-date.
`open_project(project_path)`	Open a `.rdf` project file
`close_project()`	Close the current project
`set_synthesis_tool(tool)`	Set the synthesis tool to `"lse"` or `"synplify"`. Call before `run_synthesis`.
`set_target_device(part)`	Change the target device for the active implementation. The `part` argument uses Lattice full part-number format, e.g. `LIFCL-40-7BG400I`. On failure, the response includes `compilation_errors` – every output line containing an error or failure keyword – so the caller can see exactly why Radiant rejected the part string (e.g.unrecognized device family, invalid package/speed-grade). All compiled results become invalid after a device change; re-run synthesis, map, and PAR.
`run_synthesis(force?)`	Run Synthesis
`run_map(force?)`	Run Map (technology mapping)
`run_par(force?)`	Run PAR (Place and Route). On success, returns `utilization_summary`, (device resource usage table), `par_summary` (unrouted connections, worst setup/hold slack, timing scores, PAR status), and `timing_summary` (verdict, Fmax, slack, constraint coverage, combinational loops, MPW errors, advisory notes).
`run_export(force?, stop_when_done?)`	Run Export to generate the bitstream. Pass `stop_when_done=True` to stop Radiant automatically after a successful export.
`run_timing()`	Report post-PAR timing summary. Always includes `overall_summary`, `constraint_coverage`, and `combinational_loops` sections.
`list_simulation_projects`	List pre-synthesis simulation projects registered in the open Radiant project
`run_simulation(f_file?, radiant_home?)`	Run a pre-synthesis RTL simulation using the simulator bundled with Radiant. When `f_file` is omitted, auto-discovers an existing simulation registered in the project and patches it for batch use; if none exists, generates a default testbench and `.f` file targeting the design top module. Returns `wlf_path`, `generated` (bool – whether the testbench was auto-generated), and `patched_for_batch` (bool – whether an existing `.f` was patched).

Non-Project (UDB) Flow

Chain: `load_udb` > `apply_constraints_udb` (optional) > `run_map_udb` > `run_par_udb` > `get_timing_report_udb` > `generate_bitstream_udb`

`load_udb` runs `des_report_flow_status` and returns `flow_status` so the caller knows which stages the UDB has already completed. Each subsequent `run_*_udb` tool re-queries `des_report_flow_status` at the start of its execution to get the current state before validating the prerequisite flag – ensuring the check reflects stages completed since `load_udb` was called rather than the snapshot taken at load time.

Tool	Description
`load_udb(udb_path, impl_dir?)`	Load a `.udb` checkpoint and query its flow status
`save_udb(out_path)`	Save the in-memory design to a `.udb` file
`apply_constraints_udb (sdc_path?, pdc_path?)`	Apply timing (`.sdc`) and/or physical (`.pdc`) constraints. At least one path must be provided.
`switch_to_nonproject(stage)`	Switch an open project to non-project flow
`reload_milestone(stage, udb_path, incremental?)`	Reload an earlier milestone.
`run_map_udb(out_path?)`	Run technology mapping. Requires `RUN_SYNTHESIS` flag.
`run_placement_udb(out_path?)`	Run placement. Requires `RUN_MAP` flag.
`run_routing_udb(out_path?)`	Run routing. Requires `RUN_PLACEMENT` flag.
`run_par_udb(out_path?)`	Run combined placement + routing. Requires `RUN_MAP` flag. On success, returns `utilization_summary`, `par_summary`, and `timing_summary` (same fields as `run_par`; timing is obtained from `sta_report_timing`.
`get_timing_report_udb()`	Get timing analysis (setup/hold summary and combinational loop report) from the in-memory design. Can be called after `load_udb` at any stage. Runs `sta_report_timing-summary` and `sta_report_loops`; both reports are concatenated in `output`.
`generate_bitstream_udb (outname)`	Generate the bitstream. Requires `RUN_ROUTING` flag.

Background Jobs

Background job tools spawn a separate `radiantc` process (or `qrun` for simulation) and return immediately with a `job_id`. The MCP server and interactive Radiant session remain available while the job runs. Jobs persist on disk and survive MCP server restarts and OS sleep/wake cycles. When a remote execution mode is configured, background jobs are dispatched to that remote host automatically - see Background Jobs and Remote Execution for per-mode behavior.

Remote Execution

When Radiant runs on a different machine (SSH server or LSF farm), configure the connection before starting a session:

```
Use configure_remote_execution to set up SSH or bsub access.
```

Supported modes:

Mode	When to Use
`local`	Radiant on the same machine as Claude (default)
`ssh`	Radiant on a remote server reachable by SSH
`bsub`	MCP is on an LSF login node; jobs go to compute nodes
`ssh_bsub`	MCP SSHs to a login node, then submits via bsub

Example config (`lscc-radiant-mcp.json`) – MCP on Windows, Radiant on a Linux LSF farm:

```
{
  "radiant_exe": "/tools/lattice/radiant/2026.1/bin/lin64/radiantc",
  "remote_execution": {
    "mode": "ssh_bsub",
    "ssh": {
      "host": "lsf-login.company.com",
      "user": "youruser",
      "port": 22,
      "identity_file": "C:\\Users\\youruser\\.ssh\\id_rsa"
    },
    "bsub": {
      "queue": "fpga"
    }
  }
}
```

`add_compilation_job` and `add_udb_compilation_job` automatically use the configured execution mode.The mode determines where `radiantc` runs and where its output log lives:

Tool	Where `radiantc` Runs	Where the Output Log Lives
`local`	MCP host	`~/.lscc_radiant_mcp_jobs/<job_id>/output.log` (local)
`ssh`	Remote SSH host	`/tmp/lscc_radiant_mcp_jobs/<job_id>/output.log` (remote)
`bsub`	LSF compute node	`~/.lscc_radiant_mcp_jobs/<job_id>/output.log` (local / NFS)
`ssh_bsub`	LSF compute node	`/tmp/lscc_radiant_mcp_jobs/<job_id>/output.log` (remote)

Note

bsub shared-filesystem requirement: For `bsub` mode the job's Tcl script and output log are written to the MCP host's `~/.lscc_radiant_mcp_jobs/` directory. LSF compute nodes must be able to read and write that path – which is the normal case when the home directory is NFS-mounted on compute nodes. If your site's home directory is not shared with compute nodes, use `ssh_bsub` mode instead (it stores job files in `/tmp/lscc_radiant_mcp_jobs/` on the remote host).

Mode	Probe Command
`ssh`	`kill -0 <remote_pid>` via SSH
`bsub`	`bjobs <lsf_job_id>` (local)
`ssh_bsub`	`bjobs <lsf_job_id>` via SSH on the login node

License Note:

`add_compilation_job` and `add_udb_compilation_job` each launch an independent `radiantc.exe` instance. Ensure your Radiant license supports concurrent instances if the interactive session is also active.

After an MCP server restart, remote jobs are temporarily set to `paused` (the server cannot probe remote host sat startup without the SSH/LSF config being available). The next `check_compilation_job` call re-probes the remote host and restores the job to `running` if still alive, or transitions it to `completed` or `failed` if it has ended.

Tool	Description
`add_compilation_job (project_path, synthesis_tool?, target_device?, impl_name?, configure_only?, force?)`	Submit a full compile pipeline (synthesis > map > PAR > bitstream) as a background job. Dispatches via the configured remote execution mode (`local`, `ssh`, `bsub`, or `ssh_bsub`). Pass `configure_only=True` to apply settings and save the project without running any flow stages. Pass `force=True` to force all stages to rerun even when already up-to-date. Returns `job_id`, `configure_only`, `tcl_script`, and `radiantc_path`.
`add_udb_compilation_job (udb_path, impl_dir, start_stage, outname?)`	Submit a non-project UDB compile pipeline as a background job. Dispatches via the configured remote execution mode. `start_stage` determines which stages remain: `"synthesis"` (map+PAR+bitstream), `"map"` (PAR+bitstream), `"placement"` (routing+bitstream), `"par"` (bitstream only). Returns `job_id`, `start_stage`, `outname`, and `tcl_script`.
`check_compilation_job(job_id, log_tail_lines?)`	Poll a background job. For remote jobs (`ssh`, `bsub`, `ssh_bsub`) performs a live liveness probe on each call `-kill -0` for SSH jobs, `bjobs` for LSF jobs – and persists the result. Returns `status` (`running`, `completed`, `failed`, or `paused`), `pid`, `start_time`, `end_time`, `log_file`, `log_tail`, and for remote jobs: `remote_mode`, `remote_host`, `remote_pid` (ssh), `lsf_job_id` (bsub/ssh_bsub), `remote_log_file`.

Tool	Description
`list_compilation_jobs (include_finished?, job_type?)`	List tracked background jobs. By default returns only active jobs. Pass `include_finished=True` for all historical jobs. Filter by `job_type` (`"compilation"` or `"simulation"`).
`add_simulation_job( f_file?, radiant_home?)`	Submit a built-in RTL simulation as a background job (local only - simulation requires local source files and a built-in simulator installation). Follows the same auto-selection logic as `run_simulation` when `f_file` is omitted: uses the first registered SPF simulation, or generates a default testbench. Returns `job_id`, `f_file_used`, `sim_dir`, `qrun`, `generated`, and `patched_for_batch`.

Command Fallback

Tool	Description
`run_tcl(command, timeout?)`	Execute any Radiant Tcl command not covered by a dedicated tool

Tcl Commands Used

Verified against Lattice Radiant Software 2025.1 User Guide, Chapter 9 (Tcl Command Reference Guide).

Project Flow

Tool	Radiant Tcl (2025.1)
compile_project	prj_run_synthesis + prj_run_map + prj_run_par + prj_run_bitstream (preceded by start_radiant + prj_open)
add_compilation_job	prj_open > prj_activate_impl (if impl_name) > prj_set_impl_opt synthesis <lse|synplify> / prj_set_device (if overridden) > prj_set_impl_opt {stop_if_timing_error} 0 > prj_run_synthesis + prj_run_map + prj_run_par + prj_run_bitstream > (skipped when configure_only=True) > prj_save > prj_close
open_project	prj_open <path>
close_project	prj_close
set_synthesis_tool	prj_set_impl_opt synthesis <lse|synplify>
set_target_device	prj_set_device -part <part>
run_synthesis	prj_run_synthesis
run_map	prj_run_map
run_par	prj_run_par
run_export	prj_run_bitstream
run_timing	prj_open_milestone -postpar + sta_report_timing -summary + sta_report_loops

Non-Project (UDB) Flow

Tool	Radiant Tcl (2025.1)
add_udb_compilation_job	des_read_udb + des_set_attribute -impl_dir + des_report_flow_status > stage commands per start_stage > bit_generate -w
add_simulation_job	*No Tcl – resolves .f file and spawns qrun -clean -f <file> directly as a background process*
check_compilation_job	*No Tcl – reads job metadata file and probes the OS process table; for remote jobs runs kill -0 <pid> via SSH or bjobs <id> via LSF)*
list_compilation_jobs	*No Tcl – reads job metadata files from ~/.lscc_radiant_mcp_jobs/*
load_udb	des_read_udb + des_set_attribute -impl_dir + des_report_flow_status
save_udb	des_write_udb -w
apply_constraints_udb	ldc_read <path>
switch_to_nonproject	prj_open_milestone -{stage}
reload_milestone	des_reload_milestone [-inc] -{stage}
run_map_udb	des_report_flow_status + map_run
run_placement_udb	des_report_flow_status + plc_run + plc_report
run_routing_udb	des_report_flow_status + rte_run + rte_report
run_par_udb	des_report_flow_status + par_run
get_timing_report_udb	sta_report_timing -summary + sta_report_loops
generate_bitstream_udb	des_report_flow_status + bit_generate -w

Claude Code Skills

Note:

This section is provided for reference only. No user action is required to use the tool.

Included skills:

- `/lattice-radiant-automation` - simulates and compiles a Lattice Radiant FPGA project from the command line, optionally retargeting to a different device and checks timing closure
- `/lattice-radiant-webdocs` - answers Lattice device and Radiant tooling questions from authoritative, version-matched sources: the local Radiant install (parts file, webhelp), official data sheets, and the Lattice knowledge base

Verify

`/lattice-radiant-automation`

Start a new Claude Code session and type:

```
/lattice-radiant-automation
```

Claude should greet you and ask for your user type.

`/lattice-radiant-webdocs`

Start a new Claude Code session and type:

```
/lattice-radiant-webdocs How do I constrain a clock in Radiant?
```

Claude should locate the local Radiant install, check `lattice_radiant_parts.txt` and the webhelp, and return version-matched results.

Note:

This section is provided for reference only. No user action is required to use the tool.

Troubleshooting

Symptom	Likely Cause	Fix
Skill not recognized (command does nothing)	Skill file missing or session not restarted	Verify the file path and restart Claude Code.
`lattice_radiant_parts.txt` not found	Radiant install not located	Run step 2 of the skill (locate install) or set the `RADIANT_EXE` environment variable.
`search_documents` returns empty	`lattice-radiant-skills` plugin not installed	Install via Settings > Plugins > Upload plugin.
MCP tools not available	`lattice-radiant-mcp` plugin not installed	Install `lattice-radiant-mcp.plugin` and restart Claude.
Claude stops polling a long-running process (e.g., PAR)	Claude loses track of job status	Prompt Claude for the result via the chat interface and it should resume.

Sample Session

Below is a full example session showing `/lattice-radiant-automation` compiling an LFCPNX-100 SoC design and `/lattice-radiant-webdocs` answering an RTL optimization question.

Step 1 – Persona and Environment Selection

Skill introduction:

This skill compiles a Lattice Radiant FPGA project from the command line – no GUI needed. It can optionally run a pre-synthesis RTL simulation using the built-in simulator. It handles environment setup, runs the full implementation flow (synthesis > map > PAR > bitstream), and reports timing results. For Junior FPGA Engineer and Experienced FPGA Engineer personas it also offers to change the synthesis tool, retarget to a different device, and update clock constraints before compiling.

When the selected project file is a `.udb` checkpoint the skill switches to the UDB non-project flow automatically – loading the checkpoint, determining which stages are already complete, running only the remaining stages, and reporting the same seven-item timing analysis.

The skill asks each question individually and adapts its depth to the persona chosen:

Persona	Question Asked
SW Developer	Q1, Q2/Q2A, Q3, Q4, Q_SIM, Q7, Pre-auth
Junior FPGA Engineer	Q1, Q2/Q2A, Q3, Q4, Q_SIM, Q5, Q_SYNTH, Q_DEVICE, Q6, Q7, Pre-auth
Experienced FPGA Engineer	Q1, Q2/Q2A, Q3, Q4, Q_SIM, Q5, Q_SYNTH, Q_DEVICE, Q6, Q7, Pre-auth

Q_SYNTH – Synthesis tool (Junior / Experienced only); reads the synthesis tool configured for the selected implementation from the `.rdf` and offers to keep it or switch. Valid values are `lse` (Lattice Synthesis Engine, the Radiant default) and `synplify`. If switched, `set_synthesis_tool` is called when the `lattice-radiant-mcp` MCP server is active, or `prj_set_synthesis_tool -name <tool>` is added to the Tcl script.

Q_DEVICE – Target device (Junior / Experienced only); reads the current device part from the `.rdf` and offers to keep it or change it to a different Lattice part number (e.g. `LIFCL-40-7BG400I`). If changed, `set_target_device` is called when the MCP server is active, or `prj_set_device -part {<part>}` is added to the Tcl script. A device change forces a full re-compile regardless of Q7.

Timing Analysis – after every compile (project flow and UDB flow alike) the skill reports a seven-item checklist:

#	Item	Result
1	Overall pass/fail	`Timing Errors: 0 endpoints; Total Negative Slack: 0.000 ns`
2	Actual Fmax	Clock Summary – `Actual (all paths)` row
3	Worst-case setup slack	`Path Slack: ±X.XXX ns (Passed/Failed)`
4	Hold check	Hold errors = 0
5	MPW errors	Non-zero MPW on PLL input pads is a known report artifact; the bitstream is still valid
6	Constraint coverage	`Constraint Coverage: XX.XX%` – less than 100% means under-constrained paths
7	Combinational loops	`Number of combinational loops: 0` – any non-zero count is a design bug

Area / Resource Utilization – after timing, the skill reports resource utilization from the most complete report available, checked in this order:

Priority	Report File	Used When
1 (first)	`<impl-name>/<project>_<impl-name>.par`	PAR completed — post-route numbers
2	`<impl-name>/<project>_<impl-name>.mrp`	Compilation stopped after map, before routing
3 (last)	`<impl-name>/<project>_<impl-name>.arearep`	Compilation stopped before map (synthesis estimate only)

The output states which report was used and what compilation stage it reflects, so the numbers can be interpreted correctly (post-route utilization is authoritative; earlier-stage figures are estimates).

Question	Answer
Background	SW Developer
Execution environment	Local or Remote machine
Remote machine type	Login machine (bsub)
Connection string for remote machine	`<user>@<login machine hostname>`

(Remote machine) SSH verification:

```
SSH OK
<user>
```

(Remote machine) bsub discovered at:

```
<bsub path>/bsub
```

Step 2 – Radiant Installation and Project File

Question	Answer
Radiant installation path	`<Radiant path>`
Project file	`<project path>/soc_primary_gsrd.rdf`

Detected from `.rdf`:

- Device: `LFCPNX-100-9LFG672I`
- Default implementation: `impl_1`

Note:

Junior FPGA Engineer / Experienced FPGA Engineer personas only: After detecting the implementation name the skill asks two additional questions before Q6:

- Q_SYNTH – offers to keep the current synthesis tool (`synplify` in this project) or switch to `lse`. Skipped for SW Developer.
- Q_DEVICE – offers to keep the current target device (`LFCPNX-100-9LFG672I`) or change it to a different Lattice part number. If changed, `set_target_device` is called (MCP) or `prj_set_device -part {<part>}` is added to the Tcl script, and a full re-compile is forced. Skipped for SW Developer.

Existing bitstream found – both `.bit` and `.twr` present from a prior run.

User decision: Keep existing device (`LFCPNX-100-9LFG672I`). Increase target frequency by 10% and re-compile.

Step 3 – Clock Constraint Update (+10%)

Clock	Current Period	Current Freq	+10% Period	+10% Freq
`lpddr4_pll_refclk`	10.000 ns	100 MHz	9.091 ns	110 MHz
`system_pll_clk`	8.000 ns	125 MHz	7.273 ns	137.5 MHz
`sgmii_usr_clk`	8.000 ns	125 MHz	7.273 ns	137.5 MHz

Updated `.sdc` (excerpt):

```
create_clock -name {lpddr4_pll_refclk} -period 9.091  [get_ports
{lpddr4_pll_refclk_i}]
create_clock -name {system_pll_clk}    -period 7.273  [get_ports {pll_clk_i}]
create_clock -name {sgmii_usr_clk}     -period 7.273  [get_pins
{eth_tse_mac_sgmii_inst/.../CH2_PIPE_PCS_TXCLKOUT}]
```

Step 4 – Compile Script and Job Submission

Tcl script written to `<project path>/run_impl.tcl`. LSF job submitted:

```
Job <74988953> is submitted to queue <medium>.
```

Step 5 – Compile Progress

Time	Milestone
T+0 min	Job running – Synthesis analyzing IP source files
T+4.5 min	Synthesis COMPLETE
T+9 min	Map COMPLETE – PAR launched
T+16 min	Job DONE – `Successfully completed`

Step 6 – Timing Results

#	Item	What To Look For
1	Overall pass/fail	PASS - 0 timing errors, 0 ns TNS (setup and hold)
2	Actual Fmax	See clock table below
3	Worst-case setup slack	+0.196 ns (`system_pll_clk`, worst corner)
4	Hold check	PASS - 0 hold errors
5	MPW errors	2 MPW warnings on `lpddr4_pll_refclk` input pad – known artifact; bitstream valid
6	Constraint coverage	98.3% – 4 unconstrained async paths (no action required)
7	Combinational loops	0 – clean

Clock	Target	Actual Fmax	Worst Slack
`system_pll_clk`	137.5 MHz	141.3 MHz	+0.196 ns
`axi_clk`	137.5 MHz	160.9 MHz	+0.263 ns
`cpu_clk`	137.5 MHz	161.3 MHz	+0.332 ns

Result: PASS - Design meets all targets at +10% frequency.

Sample Session — UDB Non-Project Flow

When the file selected in Q4 is a `.udb` checkpoint, the skill skips the project flow questions (Q_SIM, Q5, Q_SYNTH, Q_DEVICE, Q6, Q7) and enters the UDB non-project flow directly.

UDB Step 1 – Load the Checkpoint

```
load_udb(udb_path="<impl_dir>/design_postsyn.udb")
```

The tool returns `flow_status` showing which stages are already complete:

Flag	Value	Meaning
`RUN_SYNTHESIS`	1	Synthesis complete
`RUN_MAP`	0	Map not yet run
`RUN_PLACEMENT`	0	—
`RUN_ROUTING`	0	—
`RUN_BITSTREAM`	0	—

This UDB is post-synthesis. The skill will run map > PAR > bitstream.

UDB Step 2 – Optional Constraints

The skill offers to apply `.sdc` / `.pdc` files before running the remaining stages. In this example the user keeps the constraints already embedded in the UDB.

UDB Step 3 – Run Remaining Stages

```
run_map_udb(out_path="design_map.udb")
run_par_udb(out_path="design_par.udb")
generate_bitstream_udb(outname="design")
```

Stage	Duration	Result
Map	3 min	Complete
PAR	11 min	Complete
Bitstream	1 min	`design.bit` written

UDB Step 4 – Timing Analysis

```
get_timing_report_udb()
```

The same seven-item checklist is reported:

#	Item	What To Look For
1	Overall pass/fail	PASS - 0 errors, 0 ns TNS
2	Actual Fmax	153.8 MHz (`sys_clk`)
3	Worst-case setup slack	+0.512 ns
4	Hold check	PASS – 0 errors
5	MPW errors	0
6	Constraint coverage	100%
7	Combinational loops	0 – clean

Result: PASS - Design meets timing from post-synthesis UDB checkpoint.

UDB Step 5 – RTL Optimization Tips (from `/lattice-radiant-webdocs`)

```
/lattice-radiant-webdocs What are some tips for optimising RTL to increase Fmax?
```

Technique	RTL Action	Tool Lever
Reduce logic levels	Pipeline insertion, expression balancing	Synthesis effort
Avoid priority encoding	`case` instead of nested `if-else`	—
Retiming	`syn_allow_retiming = 1`	Synthesis
FSM encoding	`syn_encoding = one-hot` or `safe`	Synthesis
Infer carry chains / DSPs	Standard arithmetic operators	Map
Reduce fanout	`syn_maxfan`, register duplication	Synthesis

Revision History

The following table provides the revision history for this document.

Date	Document Version	Description
August 2026	1.0	Initial release.
September 2026	1.0.1	Updated for Lattice Prompt branding.