# 1️⃣ Environment & System Info (Sanity Check)

```
if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz"    /proc/cpuinfo
}
puts "Hostname : [info hostname]"
```

**Why this matters**

- Prints **CPU model, frequency, hostname**
- Very useful when:
  - Comparing runtime between machines
  - Debugging multi-CPU or server issues
  - Attaching logs to reviews

✅ Good habit for **reproducibility**

---

# 2️⃣ Global Variables & Output Directories

```
set DESIGN sha256
set GEN_EFF medium
set MAP_OPT_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_${DATE}
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}
```

### What's happening

- `DESIGN` → top module
- Separate **time-stamped directories** for:
  - Outputs
  - Reports
  - Logs

👉 This avoids overwriting old runs (⭐ very important in real projects)

---

# 3️⃣ Search Paths & Tool Configuration

```
set_db / .init_lib_search_path {...}
set_db / .script_search_path {...}
set_db / .init_hdl_search_path {...}
set_db / .max_cpus_per_server 8
set_db / .information_level 7
```

### Key points

- `init_lib_search_path` → where `.lib` lives
- `init_hdl_search_path` → RTL location
- `max_cpus_per_server 8` → parallelism
- `information_level 7` → **detailed logs** (great for learning & debug)

---

# 4️⃣ Library & Physical Setup

```
read_libs "fast.lib"
read_physical -lef "gsclib045.fixed2.lef"
set_db / .cap_table_file {...}
```

### Meaning

- `fast.lib` → timing library (best-corner)
- LEF → physical info (cell sizes, pins)
- Cap table → **pre-route RC estimation**

⚠️ Note: You correctly **did NOT mix cap table + QRC** (only one should be used)

---

# 5️⃣ Power Optimization Hook

```
set_db / .lp_insert_clock_gating true
```

Even if you don't explicitly add clock-gating constraints:

- Genus is allowed to **insert integrated clock-gating (ICG) cells**
- Helpful for **power-aware synthesis**

---

# 6️⃣ RTL Read & Elaboration

```
read_hdl "$DESIGN.v sha256_core.v ..."
elaborate $DESIGN
check_design -unresolved
```

### What happens

- Reads all RTL

- Builds full design hierarchy
- `check_design -unresolved` catches:
    - Missing modules
    - Undeclared nets
    - Parameter issues

✅ Always do this **before constraints**

---

# 7️⃣ Constraints (SDC)

```
read_sdc ".../sha256_fast.sdc"
check_timing_intent
```

This is **the heart of your 750 MHz target**.

`check_timing_intent` validates:

- Clocks
- IO delays
- False / multicycle paths

If timing intent is broken → synthesis results are meaningless.

---

# 8️⃣ Cost Groups (Timing Buckets)

```
define_cost_group -name C2C
define_cost_group -name C2O
define_cost_group -name I2C
define_cost_group -name I2O
```

**Why cost groups are powerful** They separate timing paths into:

- **C2C** → register-to-register (most critical)
- **I2C** → input-to-reg

- **C2O** → reg-to-output
- **I2O** → pure combinational IO paths

This allows:

- Focused optimization
- Cleaner timing reports
- Better debugging

💯 This is **interview-level best practice**

---

# 9️⃣ Generic Synthesis

```
set_db / .syn_generic_effort medium
syn_generic
report_dp
write_snapshot -tag generic
```

**What happens**

- RTL → **generic logic**
- No real cells yet
- Datapath inference happens here

`report_dp` shows:

- Adders
- Shifters
- Comparators
- Mux structures

👉 Very useful for **datapath optimization analysis**

---

# 🔟 Technology Mapping

```
set_db / .syn_map_effort high
syn_map
```

**Now Genus**

- Maps logic to **real standard cells**
- Tries harder to meet **1.333 ns**

You also:

- Save snapshots
- Dump timing per cost group
- Generate **RTL → mapped LEC scripts**

Excellent flow discipline 👌

---

# 1️⃣1️⃣ Incremental Optimization

```
set_db / .remove_assigns true
set_db / .use_tiehilo_for_const unique
syn_opt -incremental
```

**This step fixes**

- Leftover assigns
- Constant propagation
- Gate sizing
- Buffer insertion

Incremental = faster + safer than full re-synthesis

---

# 1️⃣2️⃣ Final Reports & Outputs

```
write_hdl  > sha256_m.v
write_sdc  > sha256_m.sdc
```

```
write_db -to_file synthesized.db
```

Generated artifacts:

- ✅ Gate-level Verilog
- ✅ Post-synthesis SDC
- ✅ Genus database
- ✅ Timing, datapath, message reports
- ✅ LEC scripts (RTL ↔ gate)

This is **exactly what P&R / LEC / STA teams expect**

---

# ⭐ Overall Quality Assessment

**This script is:**

- ✓ Industry-grade
- ✓ Cleanly structured
- ✓ Debug-friendly
- ✓ Timing-driven
- ✓ Reusable

For a **750 MHz SHA256**, this is absolutely solid.

---

# 🔧 Optional Improvements (Advanced)

If you want to push it further:

1️⃣ Add **useful QoR dumps**

```
report_qor > $_REPORTS_PATH/qor.rpt
```

2️⃣ Enable **retiming** (if allowed by spec)

```
set_db / .optimize_retiming true
```

3️⃣ Add **multi-corner runs** (slow/typical/fast)

---