Alright, let's imagine we're in the lab and I'm mentoring you, my junior, from scratch. My goal is to take you from **VHDL zero** to being confident enough to design, simulate, and test complex RTL designs on **any FPGA board**, not just Artix-7. I'll break it into **phases**, with practical exercises, mindset, and best practices.

Phase 1: Understanding FPGA & VHDL Basics

Objective: Build strong conceptual foundation. You must understand *what you are controlling* before coding.

1. **FPGA Architecture Overview**
   - Learn about **LUTs, flip-flops, block RAM, DSP slices, IOs**.
   - Understand **synchronous vs asynchronous logic**, **clock domains**, and **timing constraints**.
   - Tools: Block diagram of Artix-7, Xilinx datasheets.
2. **VHDL Fundamentals**
   - Data types: `std_logic`, `std_logic_vector`, integers, arrays.
   - Constructs:
     - **Entity & Architecture** – Think of it as *function declaration* (entity) and *function body* (architecture).
     - **Signals vs Variables** – signals reflect hardware wiring; variables are local and temporary.
     - **Processes & Concurrent Statements** – processes for sequential logic, concurrent statements for combinational logic.
   - Lab Exercise:
     - Create a **2-bit counter** and simulate its waveform.
     - Write combinational logic for **AND, OR, XOR gates** using both **concurrent** and **sequential** style.
3. **VHDL Simulation Tools**
   - Introduce **Vivado Simulator**, **ModelSim**, or **GHDL**.
   - Teach writing **testbenches**:
     - Generate clocks, resets, and stimulus signals.
     - Observe waveform outputs.

> **Mindset:** Always simulate before programming the FPGA. If it doesn't work in simulation, it will never work on hardware.

Phase 2: Combinational & Sequential Logic Design

Objective: Master coding real hardware, starting simple.

1. **Combinational Logic**
   - Multiplexers, decoders, encoders.
   - Example Exercise: Implement a **4-to-1 multiplexer** using `with-select-when` and `if-else`.
2. **Sequential Logic**

- Flip-flops, counters, shift registers.
- Clock enable and synchronous reset.
- Exercise: Build an **8-bit synchronous counter with enable**.

3. **State Machines (FSMs)**
    - Moore vs Mealy machines.
    - Design a **traffic light controller** as a FSM.
4. **Best Practices**
    - Name signals meaningfully.
    - Avoid latches (unintentional inferred hardware).
    - Use **rising_edge(clk)** for synchronous logic.

Phase 3: Modular & Hierarchical Design

Objective: Teach you to design **large, maintainable designs**.

1. **Modularity**
    - Break designs into submodules (`entity` + `architecture` per block).
    - Example: Top-level module instantiates:
        - Counter module
        - Multiplexer module
        - Output module
2. **Port Mapping**
    - Teach positional vs named mapping.
    - Lab Exercise: Implement a **simple calculator**: input two 4-bit numbers, output sum/difference/product.
3. **Generic & Configurable Designs**
    - Use **generics** to parameterize widths, number of states.
    - Example: Create an `N-bit counter` using a generic for width.

Phase 4: Timing, Constraints & FPGA Tools

Objective: Prepare you for board-level implementation.

1. **Timing Concepts**
    - Setup & hold time, clock-to-Q, propagation delays.
    - Critical paths and timing analysis.
2. **FPGA Toolchain**
    - Vivado flow: synthesis → implementation → bitstream → programming.
    - Constraints (`.xdc`) for clock and IO pin assignment.
3. **Lab Exercise**
    - Blink an LED on Artix-7 with correct clock constraints.
    - Move to **switch-controlled outputs**, integrating combinational and sequential modules.

Phase 5: Testbenches & Verification

Objective: Make you **self-sufficient in verifying designs before deployment**.

1. **Testbench Structure**
   - Instantiate DUT (design under test).
   - Clock and reset generation.
   - Stimulus processes.
2. **Verification Techniques**
   - Assertions: ensure signals meet expected conditions.
   - Waveform checking.
   - Corner cases: simulate at max/min frequency, input combinations.
3. **Lab Exercise**
   - Write a testbench for an **8-bit ALU**, verify addition, subtraction, AND, OR.

Phase 6: Advanced Topics

Objective: Prepare for **any complex design or FPGA project**.

1. **Pipelining & Parallelism**
   - Introduce **dataflow and pipeline architectures**.
   - Exercise: 4-stage pipelined adder.
2. **Memory & DSP Usage**
   - Block RAMs, distributed RAM, FIFO.
   - Exercise: Build a **FIFO buffer** with write/read pointers.
3. **Clock Domains & Synchronization**
   - Cross-clock domain signals.
   - Synchronizers, metastability.
4. **Reusable IP & Integration**
   - Use Xilinx IP cores.
   - Create your own parameterized IP for reuse.

Phase 7: Full Project Flow

Objective: Simulate **real-life board design**.

1. **Pick a small project**:
   - Example: Simple UART + LED display.
2. **Steps**
   1. Write modular VHDL code.
   2. Simulate thoroughly.
   3. Implement on FPGA (synthesize & place/route).
   4. Test on board.
   5. Debug with **ILA core** (Integrated Logic Analyzer).
3. **Iterate & Improve**

- Review waveforms.
- Refactor for performance or readability.

Mentoring Approach

- **Daily 1-on-1 coding + simulation practice.**
- **Always simulate before programming FPGA.**
- **Pair programming:** I code, you debug; then swap.
- **Weekly mini-projects:** Build from LED blink → ALU → UART → small CPU.
- **Emphasize documentation:** Maintain **design notes** and **test plans** like a real engineering team.

By the end of this structured plan:

- You will understand **hardware at RTL level**.
- You can write **clean, modular VHDL** for any design.
- You can **simulate, synthesize, implement, and test** on **any FPGA board**.
- You can debug both **functional** and **timing issues** confidently.

If you want, I can make a **step-by-step 12-week roadmap** with **specific exercises for each week**, starting from zero to complex Artix-7 project, so you can train hands-on as if you're working on real products.

Do you want me to make that roadmap?