The UltraFast™ Design Methodology is a set of best practices recommended by Xilinx to maximize productivity and reduce design iterations of complex systems, including embedded processor subsystems, analog and digital processing, high-speed connectivity, and network processing. See the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949) for more information.

The UltraFast Design Methodology Checklist (KTP301) includes common questions that highlight typical areas where design decisions have downstream consequences and draws attention to potential problems that are often unknown or ignored. It provides easy access to related collateral. The checklist is available within the Xilinx Documentation Navigator tool (DocNav).

This quick reference guide highlights key design methodology steps to achieve quicker system integration and design implementation and to derive the greatest value from Xilinx® devices and tools. Pointers to related collateral are also provided. The main design tasks covered in this guide include:

- Board and Device Planning
- Design Entry and Implementation
- Top-Level Design Validation
- Design Analysis
- Design Closure

Refer to the UltraFast Design Methodology – System-Level Design Flow available within the Xilinx Documentation Navigator tool (DocNav) for pointers to all design hubs and specific collateral.

See Also:
- UG949: Board and Device Planning
- PCB Design Checklist
- Memory Interface IP Design Checklists
- Schematic Design Checklists

See Also:
- UG949: Design Creation and Implementation
- Designing with IP Design Hub
- Using IP Integrator Design Hub
- Logic Synthesis Design Hub
- Applying Design Constraints Design Hub
- Implementation Design Hub
### TOP-LEVEL CONSTRAINTS VALIDATION

**Baseline the Design**
- Validate timing closure feasibility early in the design process after most blocks and key IP are available.
- Specify essential constraints only:
  - Use all IP constraints
  - Define realistic primary and generated clocks
  - Define all clock domain crossing constraints
  - Add multiclock paths where required
  - Do not use I/O constraints at this stage
- Ensure path requirements are reasonable
- Validate WNS = 0.0 ns using `report_timing_summary` at each stage of the flow:
  - After synthesis
  - Before placement
  - Before and after routing
- Address timing violations early in the flow
- Fix GDF issues in RTL and synthesis for the biggest impact

**Validate Timing Constraints**
- Run `report_timing_summary` or `check_timing` to ensure all clocks are defined and all registers, input ports, and output ports are constrained
- Run `report_methodology` and address all TIMING* and XDC* issues
- Run `report_clock_interaction` to ensure each clock pair is safely timed with reasonable path requirements
- Run `report_clock_cacc` to verify all asynchronous clock domain crossing paths are properly constrained and use safe synchronization circuitry
- Run `report Exceptions` to identify timing exceptions that overlap, are ignored, or are inefficient
- Ensure all Critical Warnings are resolved when the design is loaded and constraints are applied

### DESIGN ANALYSIS AND CLOSURE

**Identify Timing Violation Root Causes**
- Use `report_timing_summary` or `report_design_analysis` to find the root cause
- For setup paths, check for high datapath delay due to:
  - Large cell delay (7 series > 25%, UltraScale devices > 50%)
  - Large net delay (7 series > 75%, UltraScale devices > 50%)
  - For hold paths, check for positive hold requirement instead of usual 0 ns
- Check for high clock skew (> 500 ps), high clock uncertainty (> 200 ps), or both

**Reduce Logic Delay**
- Modify RTL to use parallel or efficient operators
- Add pipeline registers and use synthesis retiming
- Add registers on DSP or BRAM outputs
- Pull registers out of the SRL on the SRL input, output, or both
- Remove KEEP/DONT TOUCH/MARK_DEBUG to allow all optimizations

**Reduce Net Delay**
- Review and adjust floorplan constraints
- Optimize high fanout nets
- Address congestion if level > 4 is reported in:
  - `report_design_analysis.placer congestion table`
  - Initial estimated router congestion in log file

**Reduce Clock Skew**
- Use parallel buffers instead of cascaded buffers
- Use CLOCK_DELAY_GROUP between synchronous clocks originating from the same input or PLL
- Add timing exceptions between asynchronous clocks

**Reduce Clock Uncertainty**
- Optimize MMC settings
- Divide clocks with BUFF_OLD in UltraScale™ devices

### Reduce Control Sets
- Avoid MAX_FANOUT on control signals
- Increase synthesis control set threshold
- Merge equivalent control signals with `opt_design`

### Optimize High Fanout Nets
- Use hierarchy-based register replication in RTL
- Replicate fanout based on hierarchy with `opt_design`
- Promote to global clocking if not critical
- Force replication with `phys_opt_design`

### Address Congestion
- Lower device utilization and balance SLR
- Try placer directives AltSpreadLogic* or SSL_Spread*
- Identify congested modules with `report_design_analysis -complexity -congestion`
- Try synthesis directive AlternateRouting on congested modules
- Promote non-critical high fanout nets in congested regions to global clocking
- Reuse DSP and BRAM placement from previous implementations with low congestion

### Tune the Compilation Flow
- Try several `place_design` directives
- Run several `phys_opt_design` iterations
- Overconstrain critical clocks during placement and physical optimization with `set_clock_uncertainty`
- Use incremental compile after minor design modifications to preserve QoR and improve runtime

### Analyze and Optimize Power
- Constrain activity, environment, and process
- Try `power_opt` to reduce power consumption
- Maximize use of BRAM cascading

---

**See Also:**
- UG949: Design Closure
- UG949: Implementation and Design Closure
- UG949: Baselining The Design
- UG949: Design Constrained Implementation Hub
- UG949: Design Constrained Initialization Hub
- UG949: Design Constrained Analysis Hub
- UG949: Design Constrained Design Hub
- UG949: Design Constrained Optimization Hub
- UG949: Design Constrained Implementation Hub
- UG949: Design Constrained Initialization Hub
- UG949: Design Constrained Analysis Hub
- UG949: Design Constrained Design Hub
- UG949: Design Constrained Optimization Hub