ID	Description	Actions to Take	Relevant Information	Status and Notes
1.1	**Project Introduction**			
1.1.1	**General Project Overview**			
	Have you had an introductory design review and planning meeting with a Xilinx representative yet?	Setup a kickoff meeting with a Xilinx representative to discuss the design and implementation strategy.		
	Have you reviewed the FPGA design recommendations provided in the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949)?	Read the UltraFast Design Methodology for the Vivado Design Suite (UG949). This checklist links to sections of UG949 for more information about specific topics.	UltraFast Design Methodology Guide for the Vivado Design Suite (UG949)	
1.2	**Xilinx Documentation and Training**		**Accessing Documentation and Training**	
1.2.1	**Xilinx Training**			
	Are you and your team proficient with the Vivado Design Suite? Have you investigated the latest Vivado Training offerings? The Vivado Design Suite has evolved into a premier design environment with a lot of capabilities and user options. Proper training can ensure the design team quickly becomes efficient with the Vivado tools.	Attend a regularly scheduled Xilinx training class or request an onsite class. Perform Web-based online training classes.	Xilinx.com > Support > Training	
1.2.2	**QuickTake Video and Software Tutorials, User Guides**		UG949: Using the Vivado Design Suite	
	Have you explored the available software self-training collateral? Xilinx provides a host of Video and Software Tutorials aimed at quickly bringing users up to speed with the Vivado Design Suite. The Vivado User Guides provide detailed information about using the various features of the Vivado Design Suite.	View the relevant QuickTake Video Tutorials. Download the videos for the best viewing quality. Perform the software tutorials to walk through example exercises. Review the Vivado Design Suite User Guides.	UG949: Introduction > Accessing Documentation and Training > Accessing the QuickTake Video Tutorials Xilinx.com > QuickTake landing page Xilinx.com > Software Tutorials landing page Xilinx.com > User Guides landing page	
1.2.3	**Documentation Navigator**		UG949: Using the Vivado Design Suite	
	Are you using the Xilinx Documentation Navigator? Documentation Navigator provides an environment to view, search, and download Xilinx documentation and collateral. The environment can be filtered to quickly locate the desired information. Design Hubs are provided to quickly gather relevant information on specific design tasks.	The Documentation Navigator "DocNav" should be used to access all Xilinx documentation and collateral. The Documentation Navigator is installed with Vivado Design Suite.	UG949: Introduction > Accessing Documentation and Training > Using the Documentation Navigator UG910: Getting Started	
	Have you updated Documentation Navigator to access the latest documents?	Invoke Documentation Navigator and press the Update Catalog button to refresh the list of current documents.	UG949: Introduction > Accessing Documentation and Training > Using the Documentation Navigator	
	Have you explored Design Hubs in Documentation Navigator yet? Design Hubs provide direct access to information about specific design tasks including software and hardware documentation, training collateral, FAQs, answer records, and advisories.	Invoke Documentation Navigator and select the Design Hubs View tab. Explore the various information made available with each one.	UG949: Introduction > Accessing Documentation and Training > Using the Documentation Navigator	
1.2.4	**Device and IP Notifications**			
	Have you subscribed to appropriate Design Advisories? Have you read all Customer Notices, Errata, and Answer Records related to the target device and IP? Have you subscribed to the appropriate Xilinx Technical Forums?	Subscribe to the Design and IP advisories to get the latest information about the device and IP you are using in the design. Participate in the technical forums to seek help and to review past user experiences.	Xilinx.com > Support	
	Have you submitted any web cases for your design project yet? If so, have you included the project name in any web cases you have submitted?	This allows you and Xilinx to search web cases by project name.	Xilinx.com > Support	
1.3	**Software Design Flow**		**Vivado Design Suite Flows**	
1.3.1	**Design Flow Overview**		UG949: Introduction UG949: Using the Vivado Design Suite UG949: Implementation	
	Are you familiar with the overall Vivado Design Suite tool flow?	If you are new to designing with Xilinx or are migrating from an ISE Design Suite flow, it is best to familiarize yourself with the Vivado Design Suite recommended tool flows and features. There are options on how you want to interact with the tool.	UG949: Introduction > Design Process UG949: Introduction > Need for Design Methodology	
	Are you familiar with the different recommended design flow options for Vivado Design Suite? Vivado Design Suite users now have design flow options, for example, that allow for FPGA design analysis at each step of the design process vs. waiting until the end of the flow to evaluate the	Consider adopting the recommended Vivado design flow where results are analyzed at each step, rather relying solely on implementation results. It can help reduce debug cycles and achieve faster design closure.	UG949: Using the Vivado Design Suite UG949: Introduction > Rapid Validation UG949: Implementation > Timing Closure > Baselining the Design	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	design. This flow can help reduce debug cycles and achieve faster design closure.		QuickTake: Design Flows Overview UG888: Design Flows Overview Tutorial UG892: Design Flows Overview	
1.3.2	**Interacting with Source Revision Control Systems**		UG949: Using the Vivado Design Suite	
	Are you managing your source files with a revision control system? If so: Do you have a good understanding of the correct files to place under revision control? Do you have an acceptable work flow to check in/out your source file revisions? Do you also plan to place implementation results or reports under revision control?	You should familiarize yourself with the Vivado tools and select the files to place under revision control. There are options to be considered for storing the design as well as the IP. You may wish to configure and test the flow with a sample design.	UG949: Using the Vivado Design Suite > Source Management and Revision Control Recommendations	
1.3.3	**Configuring and Managing Reusable IP**		UG949: Using the Vivado Design Suite UG949: Design Creation	
	Do you have a good understanding of the various methods to configure and manage Vivado IP? Are you aware of the various IP output products and their use?	The Vivado Design Suite offers options for IP configuration and management flows. Familiarize yourself with the options and define an IP management strategy for your design. The Vivado Design Suite enables IP to be managed in two different ways; (1) managed from within a design project (2) managed remotely within a Manage IP Location project. Each method has advantages and disadvantages. Familiarize yourself with these various options and decide which one works best for you.	UG949: Using the Vivado Design Suite > Configuring IP QuickTake: Configuring and Managing Reusable IP UG896: Designing with IP UG939: Designing with IP Tutorial	
	Are you familiar with IP level constraints and how they are processed?	Most Vivado IP includes various constraint files that are used in different scenarios. The IP needs to be timing correct when implemented standalone and in the context of the top-level design. Some IP also contain physical I/O constraints that must be considered during I/O planning.	UG949: Using the Vivado Design Suite > Configuring IP > IP Constraints UG896: Designing with IP > Understanding IP Constraints	
	Are you using third-party synthesis with Xilinx IP?	Appropriate IP output products are generated to support using third-party synthesis with Xilinx IP. The Vivado Design Suite **only** supports using Vivado tools to synthesize the IP itself. Port stub files are provided for each IP to allow the IP to be black-boxed during third-party synthesis. Vivado IP is not intended to be synthesized with any tool other than Vivado synthesis.	UG949: Using the Vivado Design Suite > Logic Simulation UG896: Designing with IP > Using Xilinx IP with Third-Party Synthesis Tools QuickTake: Configuring and Managing Reusable IP	
	Do you intend to package any custom IP for use with the IP Catalog or IP Integrator? Are you migrating any existing pcores used in ISE to Vivado Design Suite?	Familiarize yourself with the data requirements to package custom IP. Use the Vivado Package IP capabilities to prepare your IP for use with Vivado tools.	UG949: Using the Vivado Design Suite > Packaging Custom IP and IP Subsystems UG1118: Creating and Packaging Custom IP UG911: ISE to Vivado Design Suite Migration Guide	
	Are you using an UltraScale^{TM} device Memory Interface IP (MIG)? Are you familiar with the I/O Planning aspects of using this IP with UltraScale devices?	Familiarize yourself with the I/O port placement capabilities for memory controllers using the Memory Bank/Byte Planner in the Vivado IDE.	UG899: I/O and Clock Planning > I/O Planning for UltraScale Device Memory IP	
1.3.4	**Creating IP Subsystems with IP Integrator**		UG949: Using the Vivado Design Suite	
	Are you familiar with the block design capabilities of the IP integrator? Do you intend to include any IP integrator block designs in the project?	The block designs created IP integrator also require a management strategy over the life of the design. Familiarize yourself with the various options to configure and manage block designs as well as the IP contained in them and decide on an upgrade strategy.	UG949: Using the Vivado Design Suite > Creating IP Subsystems with IP Integrator UG1118: Creating and Packaging Custom IP > Packaging a Block Design UG994: Designing IP Subsystems using IP Integrator UG895: System-Level Design Entry	
1.3.5	**Working with Debug Cores**		UG949: Using the Vivado Design Suite	
	Are you familiar with the options for instantiating debug cores? Do you intend to insert debug cores in RTL or use the netlist insertion flow?	For some debug cores such as the ILA, configuration and core insertion can be done after the design is synthesized. You should become familiar with the netlist insertion flow if that is desired.	UG949: Configuration and Debug > Debugging > Probing the Design UG908: Programming and Debugging > In-System Logic Design Debugging Flows	
1.3.6	**Migrating to new Software Releases**		UG949: Using the Vivado Design Suite	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Do you intend to upgrade to new software releases as they become available? Vivado Design Suite only supports customization of the most current IP version in any given release. Older versions of IP that have not been upgraded can still be used in the design, but are locked and prevented from being modified with the new software release.	Managing a design project across while migrating to newer Vivado software and IP releases can be challenging. Occasionally, IP interfaces change with major revisions. Tool options and results can also change. You have to evaluate the impact and make a decision to either stay on the software release the design is currently in or migrate the design to a new release. Xilinx recommends that you make every effort to stay current with the software.	UG949: Using the Vivado Design Suite > Upgrading Designs and IP to the Latest Vivado Design Suite Release UG896: Designing with IP > Tcl Commands for Common IP Operations	
	Have you thought about your IP upgrade strategy over the life of the design? Are you upgrading your IP to the version of the Vivado Design Suite you are using? Do you intend to upgrade your IP to the latest version available or lock it to a specific release?	The Vivado Design Suite enables you to lock your IP to a specific version or upgrade to the current version. It is always best to upgrade the IP to take advantage of RTL and constraints refinements. However, the upgrade may have a design impact. Read the IP Change Log and User Guide to understand changes made in the IP upgrades. Determine an IP upgrade strategy with each software and IP update.	UG949: Using the Vivado Design Suite > Upgrading Designs and IP to the Latest Vivado Design Suite Release UG896: Designing with IP > IP Basics > Upgrading IP > Reporting IP Status UG896: Designing with IP > IP Basics > Upgrading IP	
1.3.7	**Working with Third-Party Tools**		UG949: Using the Vivado Design Suite	
	Are you using third-party synthesis such as Synplify from Synopsys or Precision Synthesis from Mentor Graphics?	The Vivado Design Suite supports third-party synthesis tools. You should become familiar with the tool flows and options for handling IP and constraints.	QuickTake: Configuring and Managing Reusable IP UG901: Vivado Synthesis > Using Synthesis > Using Third-Party Synthesis Tools with Vivado IP UG896: Designing with IP > Using Xilinx IP with Third-Party Synthesis Tools	
	Are you using third party logic simulation such as VCS from Synopsys or NC-Sim from Cadence Design Systems? List the simulators used in the Notes section.	The Vivado Design Suite supports using third-party simulators for both behavioral and structural simulation. A target language can be specified for most IP to write out the simulation sources in either VHDL or Verilog. The ModelSim and QuestaSim simulators are integrated with the Vivado IDE. Output products to support both VCS and IUS simulators are produced along with the other IP output products.	UG900: Logic Simulation > Using Third-Party Simulators UG937: Logic Simulation Tutorial	
	Do you need to run structural simulation on any part of your design using a third-party simulation tool?	Structural netlists are automatically created for most IP in the target language specified. The Vivado Design Suite also enables writing structural simulation netlists for Verilog or VHDL for any module or IP. They can have different netlist types, std_logic and std_logic_vector, so you should ensure testbench compatibility. You should become familiar with the process to generate structural netlists.	UG949: Using the Vivado Design Suite > Configuring IP > Generating IP Output Products UG949: Using the Vivado Design Suite > Logic Simulation UG896: Designing with IP > Simulating IP UG939: Designing with IP Tutorial UG900: Logic Simulation > Understanding Vivado Simulator	

ID	Description	Actions to Take	Relevant Information	Status and Notes
2.1	**Design Architecture**			
2.1.1	**Board Planning**		UG483: 7 Series FPGAs PCB Design and Pin Planning Guide PCB Layout Recommendations UG949: Board and Device Planning	
	Have you planned your FPGA location and orientation on the PCB? Have critical interfaces to the FPGA been planned? Have you visualized the data flow through the PCB and FPGA?	Determine the critical interfaces between the PCB and the FPGA. Orient the FPGA on the PCB to best align the internal FPGA resources with their PCB components. Select initial target I/O banks for critical interfaces. Create a top-level floorplan of the FPGA design in the Vivado IDE to visualize the data flow. This ability to see the package pin to internal die relationship can help you make pin assignments that will streamline the critical connectivity.	UG949: Board and Device Planning > PCB Layout Recommendations QuickTake: I/O Planning Overview UG899: I/O and Clock Planning UG935: I/O and Clock Planning Tutorial	
	Are you familiar with the FPGA Power on requirements for the PCB? Has the Power Distribution system to the FPGA been properly planned?	The method in which the FPGA gets powered up on the PCB is important. You should familiarize yourself with the various options and requirements. Ensure the FPGA has adequate access to the power related components and voltage planes.	UG949: Board and Device Planning > FPGA Power Aspects and System Dependencies UG949: Board and Device Planning > PCB Layout Recommendations > Power Distribution System	
2.1.2	**Device Planning**		UG949: Board and Device Planning	
	Has the schematic review spreadsheet been completed?	Using the Schematic Review Spreadsheet will catch many system issues such as: physical interfaces configuration support supplies and power system monitor debug pins Ask your FAE for assistance with the schematic review checklist.	XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	Has a pinout compatible with the next smaller or larger device in a given package been considered?	If your design size may fluctuate, it's a good idea to ensure I/O pin compatibility to the next smallest or larger device in case unexpected events require a smaller or larger part. Use the Vivado IDE to set Alternate Parts when the FPGA device is expected to be heavily utilized, or if the design is still being modified.	UG949: Board and Device Planning > I/O Planning Design Flows > Defining Alternate Devices UG899: I/O and Clock Planning > I/O Pin Planning > Defining Alternate Compatible Parts	
	Does the PCB fabrication schedule dictate when the final I/O pinout assignment is due? Is the FPGA design mature enough to do pin assignments? Has the "as is" FPGA design been used to reduce risk and validate the pin out? Is there a requirement to revalidate when it is more mature?	The Vivado Design Suite enables interactive I/O exploration and assignment at each stage of the design process. An empty I/O Planning project can be created to perform early device exploration and I/O planning. However, performing I/O planning after synthesis ensures that the actual clock and I/O logic objects in the netlist can be placed and clock based DRCs are enforced. Some IP blocks such as memory controllers require careful alignment with I/O pins. Delaying pin assignment until after synthesis when the critical I/O blocks have been configured and potentially floorplanned can reduce routing challenges.	UG949: Board and Device Planning > I/O Planning Design Flows > Determine When the Final I/O Configuration is Required QuickTake: I/O Planning Overview UG899: I/O and Clock Planning > Introduction > I/O and Clock Planning Stages UG899: I/O and Clock Planning > I/O and Clock Planning Stages > I/O and Clock Planning Design Flow UG935: I/O and Clock Planning Tutorial	
2.2	**Clock and I/O Pin Planning**			
2.2.1	**I/O Planning**		UG949: Board and Device Planning UG949: Design Creation UG949: Implementation	
	Are all I/O ports defined at the top level of the RTL design? Note: A few Xilinx IP have I/O logic instantiated in submodules. Do not attempt to bring those up to the top level.	The RTL design should be crafted in such a way to bring all of the I/Os and global clock logic to the top-level of the design to facilitate easier constraints assignment and analysis. All Xilinx IP I/O and clock logic should be left in place at the respective levels of logic hierarchy within the IP.	UG949: Board and Device Planning > I/O Planning Design Flows	
	Are wide or critical buses assigned to facilitate streamlined data flow between PCB and FPGA resources? Has the data bus width caused bank-crossing situations? Have the GTs been assigned to facilitate data flow? Has the Connectivity IP and Memory Interfaces been implemented to influence I/O assignments?	The design data flow should be visualized in the Vivado IDE to ensure no obvious congestion or long routing requirements exist through the I/Os. This can be done by creating a top-level floorplan and arranging the floorplan and I/O assignments to facilitate the best data flow from the I/Os to internal FPGA resources.	UG949: Board and Device Planning > I/O Planning Design Flows > Pin Assignment QuickTake: I/O Planning Overview UG899: I/O and Clock Planning > I/O and Clock Planning Stages > I/O and Clock Planning Design Flow UG935: I/O and Clock Planning Tutorial UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Have you considered potential throughput bottlenecks such as BRAM, DDR, network traffic, etc.? Have you ensured there is sufficient headroom in design? Does the pinout allow the required speeds?	Define each external interface in terms of required performance. Ensure they expected performance is feasible for the design and device. There are IP available to help measure throughput such as the MIG testbench, and the AXI bandwidth monitor.	2.0 Board and Device Planning	
	Are you using stacked silicon interconnect (SSI) technology?	SSI technology requires special attention during I/O planning. Familiarize yourself with the SSI technology requirements and recommendations.	UG949: Implementation > Implementing the Design > Placement (place_design) > SSI Placement > Manual SLR Assignment Guidelines UG949: Board and Device Planning > I/O Planning Design Flows > Designing with SSI Devices UG949: Design Creation > RTL Coding Guidelines > Clocking > Additional Clocking Considerations for SSI Devices UG949: Design Creation > RTL Coding Guidelines > Clocking > Clock Skew for Global Clocking Resources in SSI Devices UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning > Floorplanning with SSI Devices	
	Was the Vivado GT Wizard or the ISE Transceiver Wizard used to create the pinout for the transceivers in this design?	Use the GT Wizard to ensure that the GTs have been correctly configured.		
	Are you using any Xilinx Connectivity IP? If so, has it been implemented prior to I/O pin assignment?	Xilinx Connectivity IP requires consideration during I/O planning. Some IP contain external I/O ports. Ensure you have implemented it early enough to help influence the pinout. Ensure that the example design for each IP is implemented to ensure that all I/O constraints are properly identified.	Refer to the IP Product Guide for more information	
	Have the memory-interface-related I/Os been assigned using MIG? Are all constraints and rules created through MIG included in the final constraints file?	Using MIG to configure memory interfaces involves an element of I/O planning. Physical constraints and configuration properties are generated while configuring the core. These I/O constraints and configuration properties should propagate through the flow to ensure complete and correct I/O pinout for the design. Verify that the I/Os have been assigned properly.	MIG Checklist UG586: 7 Series FPGAs Memory Interface Solutions User Guide UG899: I/O and Clock Planning > I/O Planning for UltraScale Device Memory IP PG058: LogiCORE IP Block Memory Generator Product Guide	
	Do any external interfaces have critical I/O timing requirements? If so, list them in the Status and Notes field.	Critical interfaces require extra attention during I/O and timing constraint assignment. List them all to ensure they are properly constrained and considered during I/O planning.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports UG903: Using Constraints > Constraining I/O Delay	
	Have you considered putting prohibits on VREF pins to ensure they don't get accidentally assigned?	Use the I/O Planning view layout in the Vivado IDE to interactively select the VREF package pins and place prohibits on them.	UG899: I/O and Clock Planning > I/O Planning > Defining and Configuring I/O Ports > Prohibiting I/O Pins and I/O Banks UG935: I/O and Clock Planning Tutorial	
	Have the I/O Standard constraints for all I/O been considered and explicitly assigned for each I/O Port? Are Drive Strength, Slew Rate, and Pull up/down/keepers set for each I/O port?	Use the I/O Planning view layout in the Vivado IDE to interactively assign I/O physical and I/O Standard constraints. Run the I/O and Clock rules of the *report_drc* command to check I/O Standard assignment and compatibility.	UG949: Board and Device Planning > I/O Planning Design Flows > Pin Assignment QuickTake: I/O Planning Overview UG899: I/O and Clock Planning > I/O Planning > Defining and Configuring I/O Ports > Configuring I/O Ports UG935: I/O and Clock Planning Tutorial	
	If used, have DCI Cascade constraints been analyzed for proper management?	The Vivado Design Suite enables interactive viewing and assignment of DCI Cascades in the design. Verify DCI Cascade assignments either using the Vivado IDE or manually.	UG949: Board and Device Planning > I/O Planning Design Flows > Pinout Selection > Internal VREF and DCI Cascade Constraints UG899: I/O and Clock Planning > Configuring the Device > Setting Device Constraints	
	Does the design use FFs for input and output I/Os? Are ILOGIC and OLOGIC attributes correctly applied?	Run the custom Tcl command *report_io_reg.tcl* to determine if ILOGIC and OLOGIC are assigned properly. Check the report file to ensure all ports have either an OLOGIC or ILOGIC assigned.		
	Have you identified pins with internal/external terminations and annotated the XDC file? Have you shared those with the PCB designer?	Identify pins with internal/external terminations and annotate XDC file to share with PCB designer.	Refer to the PCB Design and Pin Planning Guide for the FPGA architecture you are using. UG483: 7 Series FPGAs PCB Design and Pin Planning Guide UG583: UltraScale Architecture PCB Design and Pin Planning Advanced Specification User Guide	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Are any I/O of the FPGA driving power supplies? Have you taken into consideration the pullup/down that may occur during power up, configuration, and Flash programming?	FPGA I/Os have limited drive strength and should be considered when sinking/sourcing current to any active or passive component. Also the behavior of some pins may be different before, during and after FPGA programming and should be considered when connecting to any sensitive circuitry.		
	Do you have IOs that will be driven externally before the device is powered up or is configured?	For configured but unused pins, verify that any PCB connections are not in any possible contention or left floating.	Xilinx Answer Record 45985	
	Have you considered building system-level baseline design to check component level interfaces on the FPGA?	You can build a small sample design with just the I/O interfaces and clocks in the design. This ensures proper handling of complex I/O configurations and can give you a sanity check before committing to the I/Os.	UG949: Board and Device Planning > I/O Planning Design Flows > Interface Bandwidth Validation	
2.2.2	**Clock Resource Planning**		UG472: 7 Series FPGA Clocking Resources UG949: Board and Device Planning UG949: Design Creation	
	Are all the clocks and derivatives (e.g. phase shifted, multiples, divisions, etc.) for this design known? Has the intended clock topology been mapped to clock components (e.g. MMCM, PLL, BUFG, BUFR, etc.) for the given device architecture? Are there sufficient resources and connectivity to accomplish this assignment? □	Clock usage and placement have a great impact on results. Make sure to properly plan each global and generated clock to be used in the design. Once the design is synthesized, you can run the *report_clock_networks*, *report_clock_utilization,* and *report_drc commands* to analyze and validate clocking structures and rule adherence. This step is covered in Design Creation and Implementation.	UG949: Board and Device Planning > Clock Resource Planning and Assignment	
	Does the total number of global clock resources used in the design exceed the device limit of 32 for 7-Series devices? The limit for UltraScale is high, but special design considerations are required when using large numbers of global clocks. Are you familiar with when to use SRCC I/O pins vs. MRCC pins (7 series devices)? Are all internally generated clocks distributed on a dedicated clock resource (e.g. BUFG, BUFR, etc.)? List any clocks that are on local routing resources in the Status and Notes field.	There are clock placement ramifications for designs that require more than 16 global clocks in 7 series devices and 24 global clocks in UltraScale devices. Further clock planning may be required to either share some global clocks or move some of the clocks to non-global clocking resources. All clocks should be defined on either global or regional clock resources. Once the design is synthesized, you can run the *report_clock_utilization* command and analyze the resulting report. Look for over utilized clock resources. This step is covered in Design Creation and Implementation.	UG949: Board and Device Planning > Clock Resource Planning and Assignment > Selecting Clocking Resources UG949:Design Creation > RTL Coding Guidelines > Clocking UG903: Using Constraints > Defining Clocks	
	Are you using Source Synchronous Interfaces? Has the clocking been properly configured?	Source synchronous interfaces have special clocking requirements. Familiarize yourself with these requirements. The Vivado Design Suite includes constraint templates to assist with complex constraint creation.	UG949:Design Creation > RTL Coding Guidelines > Clocking UG903: Using Constraints > Defining Clocks	
	If over utilized, can and should any of the Vivado IP share clock resources?	Sharing clocking resources can address possible timing issues while reducing overall clock power dissipated. The Vivado IP has been constructed to enable sharing of top-level clock logic across multiple IP. This is configured within the IP customization wizard. Consider consolidating IP clock resources if additional clock resources are needed.	UG896: Designing with IP > IP Basics> Creating an IP Customization UG895: System-Level Design Entry	
	Are tradeoffs between MMCM and PLL usages considered? Have you used the Vivado Clocking Wizard to configure the MMCMs and PLLs?	The MMCM/PLL have different features and characteristics for jitter, performance, accuracy and power depending which is used and on how they are configured. Understand the tradeoffs between the two. Analyze your design usage of MMCMs and PLLs and determine if they are the optimal choices. Use the Clocking Wizard to configure MMCM/PLL settings.	UG949: Board and Device Planning > Clock Resource Planning and Assignment > Selecting Clocking Resources UG949: Design Creation > RTL Coding Guidelines > Clocking > Controlling the Phase, Frequency, Duty-Cycle, and Jitter of the Clock	
	Have you considered using BUFGMUX clock muxes?	If a clock MUX or advanced clock gating is needed, using the dedicated BUFGMUX provides improved results over general logic. Understand the advantages of using BUFGMUXs and apply where possible.	UG949: Design Creation > RTL Coding Guidelines > Clocking	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Are the BUFGMUX/BUFGCTRL clock inputs located in the right region?	For 7 series devices, the clock inputs when coming from an MMCM/PLL or I/O should both originate from either the upper half or lower half of the device. For UltraScale devices, it is suggested to place both inputs when sourced by an MMCM/PLL or I/O into the same clock region or bank.	UG949: Board and Device Planning > Clock Resource Planning and Assignment > Selecting Clocking Resources	
	Are all single-ended I/O standard based clocks placed on the P-side of a GCLK or CCLK pin?	Single-ended clock must use the p-side of a CCIO for proper connection. Verify that all single ended IO standard clocks are assigned properly.	UG949: Board and Device Planning > Clock Resource Planning and Assignment > Selecting Clocking Resources > 7 Series Devices > Single or Multi Region Clock Pin Selection	
2.2.3	**Non-signal and Power Pin Connectivity**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Will the XADC or SYSMON be used to monitor internal die voltage/temps, current draw on power supplies, or external sampling ADC mode?	The XADC is valuable to understand the on-chip characteristics of power supplies and die temperature. Having it properly connected and available allows additional insight to the chip operation. The XADC must be powered and connected properly to allow reliable operation.	UG949: Board and Device Planning > FPGA Power Aspects and System Dependencies > Measuring Power and Temperature > Power Measurement Techniques > Performing On-Board Monitoring UG480: 7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide UG580: UltraScale Architecture System Monitor User Guide	
2.3	**Power Planning and Analysis**			
2.3.1	**General Analysis**		UG949: Board and Device Planning	
	Has the Xilinx Power Estimator (XPE) or *report_power* been run? Has the design exceeded its Power budget?	Perform early power analysis using Xilinx XPE and final analysis with the Vivado *report_power* command. Measure power at each stage of the design and make power reducing design modifications, if needed.	UG949: Board and Device Planning > Worst Case Power Analysis Using Xilinx Power Estimator (XPE) UG907: Power Analysis and Optimization > Estimating Power – Initial Evaluation Stage	
	Can outputs be set to a lower voltage, lower drive strength, or slower slew rate (IBIS simulation may be required)?	Examine any outputs that can use lower power settings. Assign new constraint values and run the *report_drc* command to validate the new assignments.	UG949: Board and Device Planning > Worst Case Power Analysis Using Xilinx Power Estimator (XPE) UG907: Power Analysis and Optimization	
	Has it been determined that terminations are absolutely needed (IBIS simulation required)? Note: DCI termination can always be selectively enabled (assuming Vrn/Vrp connections are made) if there are unforeseen issues.	DCI termination consumes additional on-chip power that contributes to both current draw and junction temperature increase. Evaluate all terminations for possible removal.	UG949: Board and Device Planning > Worst Case Power Analysis Using Xilinx Power Estimator (XPE) UG907: Power Analysis and Optimization	
	Can the FPGA-to-FPGA communications that use LVDS be implemented via a single-ended interface to reduce power?	Single-ended I/O interfaces consume less power in general. Examine using single-ended I/O interfaces. Make the appropriate design changes, assign new constraint values and run the report_drc command to validate the new assignments.		
2.4	**Configuration**			
2.4.1	**General Hardware**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Is the targeted memory device and memory solution supported by Xilinx for FPGA configuration?	Verify the desired memory device is supported by Xilinx.	UG949: Configuration and Debug > Configuration Product family Configuration User Guide UG908: Programming and Debugging XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	Are the configuration signals accessible for debug and are status indicators (LED) for pins brought out (i.e. INIT/DONE) to the PCB?	INIT and DONE are the most important status signals available and are key to understanding the configuration status and failures. Easy access to the pin will accelerate the debug process if a configuration failure is experienced.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	Has the PUDC_B pin been tied appropriately?	If you do not want to use the internal pull-ups during configuration, then PUDC_B needs to be tied High. If internal pull-up resistors are desired on the SelectIO pin after power-up and during configuration, then verify that PUDC_B is Low. This pin must not be left floating before and during configuration.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	Have you made sure that the pins on an unpowered FPGA are not driven by an external source?	Review the schematic to verify.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist	

ID	Description	Actions to Take	Relevant Information	Status and Notes
			XTP344: UltraScale Architecture Schematic Review Checklist	
2.4.2	**Voltage Compatibility**		UG949: Board and Device Planning	
	Is the CFGBVS pin set appropriately and have the cautions been reviewed for handling of this signal?	Verify that the pin selection matches the targeted configuration mode bank voltage. For details on this pin refer to the Configuration User Guide.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	If using external SPI or parallel NOR flash, have the bank voltages been verified to ensure they are compatible?	Verify the SPI or parallel NOR support the flash bank voltage I/O selected.	SPI or Parallel NOR Flash Data Sheets Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
2.4.3	**JTAG (Recommended for All Boards)**		UG949: Board and Device Planning	
	Do you have the appropriate JTAG cable for your device?	Ensure the board includes a supported USB for Digilent cable or ribbon cable connector for the Xilinx Platform Cable USB II JTAG cable interface. JTAG cable access is valuable for debug and device status regardless of the configuration mode selected.	Product family Configuration User Guide DS593: Platform Cable USB II Data Sheet	
	If third-party devices are in the JTAG chain, have the test logic reset (TRST), compliance enable, or other JTAG control signals for all devices in the target chain been properly connected?	Verify the design contains the proper connections for configuration and debug.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist Third-Party Device Data Sheet	

ID	Description	Actions to Take	Relevant Information	Status and Notes
3.1	**Design Hierarchy**			
3.1.1	**General Analysis**		UG949: Design Creation	
	Was the design hierarchy defined to accommodate module function or clock isolation and floorplanning?	The logic hierarchy of the design should be planned with forethought to isolate clock regions, or specific functions where possible. Consider restructuring hierarchy where possible to group related logic.	UG949: Design Creation > Defining a Good Design Hierarchy > Address Floorplanning Considerations UG949: Design Creation > Defining a Good Design Hierarchy	
	Are all data paths registered at logical boundaries?	Register all data path signals at hierarchical boundaries to better facilitate floorplanning, timing analysis/debug and out-of-context implementation.	UG949: Design Creation > Defining a Good Design Hierarchy > Register Data Paths at Logical Boundaries	
	Are all clocking elements at the top level of design? Note: IP clock logic is located at the top level of the IP logic hierarchy and should not be brought up to the top-level.	It is much easier to share clocking resources when they are at the top-level of the design, thus improving P&R, timing, and/or power in the design. Constraining and debugging clocks is also generally easier when placed at the top-level of hierarchy compared to when buried in the design.	UG949: Design Creation > Defining a Good Design Hierarchy > Place Clocking Elements Toward the Top Level	
	Have you run the RTL DRCs to determine if your code have conditions that may limit functionality of performance?	The Vivado Design Suite has an ever-expanding set of DRC checks that can be run on the Elaborated RTL design. Elect to elaborate your design and analyze the Messages view to find any potential RTL code issues.	UG949: Design Creation > RTL Coding Guidelines > Running RTL DRCs UG906: Design Analysis and Closure Techniques > Logic Analysis Within the IDE > Validating Design Methodology Logic DRCs	
3.2	**IP Integration**			
3.2.1	**Configuring IP**		UG949: Using the Vivado Design Suite UG949: Design Creation	
	Is the IP interface fully compliant with AXI4 protocol?	Vivado IP is delivered with AXI4 compliant interconnect. Any custom IP should also follow these guidelines. Run the *check_bd_axi_interface.tcl* custom Tcl command to report any discrepancies. ARM provides a set of SystemVerilog assertions that can be used identify AXI4 violations during simulation. They are free and can be quite useful. A link is provided in the Relevant Information column.	UG949: Design Creation > Working with Intellectual Property (IP) > AMBA AXI QuickTake: Configuring and Managing Reusable IP UG896: Designing with IP UG939: Designing with IP Tutorial AMBA® 4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions User Guide	
	Are you getting messages about missing clocks when using an IP?	Each IP may contain timing constraints to support the IP. Review the Synthesis and Implementation Messages and log files to ensure the IP constraints are being properly applied.	UG949: Using the Vivado Design Suite > Configuring IP > IP Constraints UG896: Designing with IP > IP Basics > Understanding IP Constraints	
3.3	**RTL Coding Guidelines**			
3.3.1	**Coding Style: RTL General Guidelines**		UG949: Design Creation	
	Are you using the Vivado Design Suite HDL Templates?	The HDL templates provide recommended coding styles and instantiation templates for targeting Xilinx FPGAs. It is suggested to use these to ensure best results.	UG949: Design Creation > RTL Coding Guidelines > Using Vivado Design Suite HDL Templates UG895: System-Level Design Entry > Working with Source Files > Editing Source Files	
	Are Verilog blocking statements used for unregistered logic and non-blocking for registered logic?	Misuse of blocking and non-blocking statements can result in synthesized results not matching RTL simulation behavior. Following this guideline helps prevent this from occurring. Review the synthesis log file for warnings.	UG949: Design Creation > RTL Coding Guidelines > Basic Functionality > Blocking Statements vs. Non-Blocking Statements	
	Are VHDL/Verilog Sensitivity lists complete?	Incomplete sensitivity lists can lead to synthesis results not matching RTL simulation. Review the synthesis log file for warnings.	UG949: Design Creation > RTL Coding Guidelines > Basic Functionality > Incomplete Sensitivity List	
	Have you ensured that there are no Delays in the RTL code?	Delays in HDL code are not synthesizable. Improper use of delays in synthesizable RTL code can result in synthesis results not matching RTL behavior. It is suggested to review and/or remove all delays in synthesized code.	UG949: Design Creation > RTL Coding Guidelines > Basic Functionality > Delays in RTL Code	
	Have you ensured that there are no incomplete if/else clauses?	Incomplete if/else statements can infer latches in combinatorial blocks and infer clock enables in synchronous blocks. This can result in unneeded timing issues when that is not required behavior for functionality and/or power savings. Ensure if/else clauses are complete unless determined necessary.	UG949: Design Creation > RTL Coding Guidelines > Basic Functionality > Latch Inference	
	Are you using for/while loops in RTL code?	For/while loops are not always synthesizable and even when deterministic, does not always create the best results in terms of logic and synthesis runtime. It is generally suggested to avoid such constructs in synthesizable code.	UG949: Design Creation > RTL Coding Guidelines > HDL Coding for Efficiency > Use of Loops in Code	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Are State-machine type, style and encoding properly selected?	Understanding and controlling state-machine implementation often results in the best trade-offs of performance, area, power and reliability of a circuit. Proper thought and guidance up front often leads to best characteristics for a given design.	UG949: Design Creation > RTL Coding Guidelines > HDL Coding for Efficiency > State Machine Guidance	
	Does any hierarchy contain mixed clock edges?	Mixing clock edges may lead to more critical paths in the design. Use this with discretion and understanding of timing impacts to the design.	UG949: Design Creation > RTL Coding Guidelines > HDL Coding for Efficiency > Avoid Mixing Edges of a Flip Flop	
	Was debug logic coded for easy removal?	Debug logic serves a valuable purpose in the bring up of the design but can lead to more difficult timing closure and increased power. For best end results, it is suggested to code any temporary debug logic in a way to facilitate easy removal.	UG949: Design Creation > RTL Coding Guidelines > HDL Coding for Efficiency > Use of Debug Logic	
	Are there any arrays declared in port declarations?	Defining arrays in port declarations can lead to difficulties in mixed-language design as well as difficulty in verification/debug. It is generally suggested to avoid arrays in port declarations.	UG949: Design Creation > RTL Coding Guidelines > HDL Coding for Efficiency > Arrays in Port Declarations	
	Are there a large number of Control Sets in the Design?	Large numbers of control sets (unique clock enable, clock and resets) can lead to less dense placement and thus longer wire-lengths resulting in more difficult timing and higher power. It is suggested to understand and where possible limit unique clock enables, resets and clock domains in the design.	UG949: Design Creation > RTL Coding Guidelines > Control Signals and Control Sets UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports > Report Control Sets	
3.3.2	**Coding Style: Fanout Guidelines**		UG949: Design Creation	
	Are High Fanout Nets Avoided?	High fanout nets can lead to timing issues in critical paths of the design. Run the *report_high_fanout_nets* and *report_timing* commands and analyze the high fanout nets. Determine if any can be reduced.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Performance > High Fanouts in Critical Paths	
	Are flip-flops duplicated to reduce high fanout nets?	Run the *report_fanout* command to identify candidates for replication.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Performance > High Fanouts in Critical Paths > Use Register Replication	
	Are high fan-in nets (associated with large muxes and buses) constrained and accommodated in the design?	Evaluate high fan-in logic for possible timing and/or routing issues and ensure best coding practices are applied including applying additional pipelining if possible.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Performance > Managing Wide Buses	
3.3.3	**Coding Style: Resets**		UG949: Design Creation	
	Does the reset functionality follow Xilinx guidelines?	You should: minimize the number of resets; reset control logic only, and avoid resetting the datapath; avoid active low resets. Run the *report_reset_signals.tcl* custom Tcl script to ensure proper use of resets. Global resets are often not necessary in FPGA design and can consume large amounts of routing and can cause timing issues both inside and outside the reset timing paths. It is suggest to only code resets in the portions of the design that are necessary for proper operation and avoid resets in all other areas of the design.	UG949: Design Creation > RTL Coding Guidelines > Control Signals and Control Sets > Resets > When and Where to Use a Reset	
	Can the block be put into a deterministic reset condition either synchronously or asynchronously?	Xilinx generally recommends using synchronous resets when a reset is necessary for functionality. Using synchronous resets gives more flexibility to resource mapping, generally providing improved results in performance, area, and power. Use synchronous resets when a reset is necessary. Use the *report_reset_signals.tcl* custom Tcl command to identify control signals and reset.	UG949: Design Creation > RTL Coding Guidelines > Control Signals and Control Sets > Resets > Synchronous Reset vs. Asynchronous Reset	
	Are mixed reset polarities used?	Mixing reset polarities can result in sub-optimal timing on the reset path, which in many designs can be critical for timing. Mixed polarities also limit resource mapping to elements like global routing for improved implementation particularly when using bottom-up or out-of-context design flows. It is suggested to choose active high or active low resets and use them consistently in the design.	UG949: Design Creation > RTL Coding Guidelines > Control Signals and Control Sets > Resets > Control Signal Polarity (Active-High vs. Active-Low)	
	Have you put high fanout reset nets onto BUFGs (total number less than 3), or replicated the driver for fanout limitation?	In some cases, improved timing and routability can be seen when high-fanout nets like resets are mapped to BUFGs. It is suggested to evaluate this in situations where very high fanout nets exist to see if results can be improved.	UG949: Design Creation > RTL Coding Guidelines > Clocking > Clock Resource Selection Summary	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Does each clock domain have an independent reset line?	In general, it is suggested to avoid resets on logic that does not require it. On paths that do require resets, ensure resets are properly created and constrained particularly when a common reset crosses a clock domain.		
	Have you checked that the design has no FFs that have an asynchronous de-assertion of the preset or clear input?	Xilinx generally recommends no resets or, if needed, synchronous resets. In the event that asynchronous resets are used, ensure that the reset is synchronously deasserted and properly timed. Run the *report_reset_signals.tcl* custom Tcl script to ensure proper use of FFs.		
3.3.4	**Coding Style: Memories**		UG949: Design Creation	
	Were the Vivado Design Suite HDL templates used to infer memories?	Improperly coded memory arrays can result in sub-optimal implementation results. It is suggested to use the Vivado Design Suite HDL Templates for memory inference to ensure that Xilinx BRAM and/or LUTRAM is properly inferred.	UG949: Design Creation > RTL Coding Guidelines > Inferring RAM and ROM	
	Is the BlockRAM output register being used?	Using the output registers of the BRAM can result in over 1ns clock-to-out improvement from the BRAM at the cost of added latency during read. You can also gain an extra 500 ps with a 3 cycle read with one in fabric. If this is acceptable, it is highly suggested to use these registers.	UG949: Design Creation > RTL Coding Guidelines > Inferring RAM and ROM > Performance Considerations when Implementing RAM	
	Is the proper write mode selected for the RAM?	BRAM write modes can impact functionality, power and reliability depending on the selected mode and operation of the BRAM. It is suggested to understand and always choose the proper write mode to match desired functionality, power and operation goals.	UG949: Design Creation > RTL Coding Guidelines > Inferring ROM and RAM > Selecting the Proper Block RAM Write Mode	
	Was the hard FIFO selected for FIFO buffering functions?	The dedicated hard-FIFO has performance, power, and area benefits over using a soft implementation of a FIFO. It is also more easily migrated to UltraScale devices. It is suggested to evaluate and use the hard FIFO where possible.	UG949: Design Creation > RTL Coding Guidelines > FIFO Creation	
3.3.5	**Coding Style: DSP**		UG949: Design Creation	
	Does the DSP bit width match the target device for optimal area and performance?	The DSP block has specific, bounded input widths per block depending on the selected operation. Understand how the desired precision width of the coded function maps to one or more DSP blocks and therefore allows for proper expectations so that coding modifications such as additional pipelining can be properly accounted for.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
	Has signed vs. unsigned been considered both for algorithm performance and DSP48 usage?	The DSP multiplier natively performs signed operations. Using unsigned arithmetic reduces the precision of the block. Understand how this affects bit width selection and mapping into the DSP blocks.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
	Have the dedicated DSP48 routing resources and logic been optimally used?	The DSP block has built-in "cascade" logic that efficiently implements arithmetic/filter operations that extend beyond a single DSP block. Using cascades and getting the best implementation requires specific pipelining and coding style considerations. When inferring arithmetic/filter operations that extend beyond a single DSP block, understand the best coding practices to infer these paths.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
	Has the multiplier in the DSP48 been fully pipelined using the registers in the DSP48?	Proper pipelining for the DSP multiplier not only improves performance characteristics but also improves power for multipliers implemented in the DSP Blocks. Understand and, when possible, add additional pipelining to improve performance/power characteristics.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
	Is the arithmetic code reset synchronous, active high and initializes/resets to zeroes?	Best practice is to not use reset where not needed, but, when needed, it is suggested the reset to be synchronous, active high and reset to a zero to have it properly inferred into a DSP48 block. Follow this guideline for any register intended to be placed into a DSP block.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
	Is the code structured to take advantage of multiplication-addition flows for optimal performance?	It is suggested to understand the structure and capabilities of the DSP block so that coding styles, when feasible, match the datapath and pipelining paths in the block to ensure optimal inference.	UG949: Design Creation > RTL Coding Guidelines > Coding for Proper DSP and Arithmetic Inference	
3.3.6	**Coding Style: Clocking**		UG949: Board and Device Planning UG949: Design Creation	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Are all clock resources (i.e. IBUFGs, BUFGs, BUFRs, , PLLs, global/regional clock pins, etc.) instantiated? Have location constraints been created for the any of the clock resources?	Manual placement of clock logic objects can help improve performance. Understand the clocking requirements of the device before attempting to assign placement constraints for clock resources. Use the Vivado IDE to visualize clock regions, I/O banks, and clock logic and to interactively assign placement constraints. Run *report_drc* to validate the assignments. Ensure all clock inputs are placed on Clock Capable (CC) or Global Clock (GC) pins. Run the *report_clock_utilization* Tcl command to report clock resources with a location constraint, and run the *report_clock_networks* Tcl command to visualize the topology of each individual clock tree.	UG949: Board and Device Planning > Clock Resource Planning and Assignment UG949: Design Creation > RTL Coding Guidelines > Clocking UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning	
	Are all clock resources being properly selected? Are all internally generated clocks distributed on a dedicated clock resource (e.g. BUFG, BUFR, etc.)? List any clocks that are on local routing resources in the Status and Notes field.	All clock sources should be analyzed for possible consolidation as well as ensure proper construction and no possible conflicts could exist during design implementation. Run *report_clock_networks*, *report_clock_utilization* and *report_drc* to analyze and validate clocking structures and rule adherence.	UG949: Design Creation > RTL Coding Guidelines > Clocking UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning	
	Is the RTL/IP designed so that no LUT-gated clocks are used? List any gated clocks in the Status and Notes field.	Clock gating should be performed using the clock enables of synchronous elements and/or clock buffers enables and should never use LUT logic for gating functions. Ensure proper coding styles and synthesis inference behavior on clock lines to ensure no LUTs or other logic exists on the clock lines.	UG949: Design Creation > RTL Coding Guidelines > Clocking > Controlling the Phase, Frequency, Duty-Cycle, and Jitter of the Clock > Using Gated Clocks	
	Is MMCM and/or PLL LOCK signal monitored to ensure a clean clock on the input prior to the reset being released to the circuitry in that clock domain?	Run *check_pll_connectivity.tcl custom* Tcl command and examine *pll_io_check_report.txt* to report PLL and MCMM connectivity.	UG949: Design Creation > Coding Styles for Higher Reliability > Controlling and Synchronizing Device Startup	
	If using the mux function within the BUFGMUX, have clock transition requirements been considered?	The BUFGMUX has built in logic to ensure "glitch-free" transition of clock sources. Ensure proper considerations are made in terms of timing and input clock stability to ensure safe clock transition of clock sources.	UG949: Design Creation > RTL Coding Guidelines > Clocking > 7 Series Device Clocking > Global Clocking Resources	
	Does the design require that a multi-cycle path be used?	Use properly coded and constrained multi-cycle paths can relax timing, on related and unrelated paths in the design. Evaluate the use and constraint of multi-cycle paths to improve overall implementation results.	UG949: Design Creation > Working with Constraints > Adding Multicycle Path Constraints	
3.3.7	**Coding Style: Reliability**		UG949: Design Creation	
	Have all clock domain crossing been analyzed and classified as synchronous or asynchronous?	Synchronous CDCs should have both clocks properly related by phase. Asynchronous CDC should have proper timing exceptions and synchronization created. Run the *report_cdc* Tcl command and examine the results. Verify that all clock domain crossings have been closely analyzed.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Clock Domain Crossings UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports	
	Are all asynchronous crossings properly synchronized?	For reliable data capture across asynchronous clock domains, proper synchronization practices should be employed.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Clock Domain Crossings > Asynchronous Domain Crossing	
	Are phase differences between synchronous clock domains minimized for all synchronous domain crossings?	When crossing synchronous clock domains, large phase differences can cause large setup and/or hold times to occur resulting in sub-optimal timing and longer P&R times. Analyze and reduce any phase differences between clock domains in which data must be passed.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Clock Domain Crossings > Synchronous Domain Crossing, But Potentially Very High Skew	
	Is the design properly synchronized for safe startup?	Upon release of the global reset and enable for the device at start-up, some considerations should be made in order to ensure the design starts in a known and desired state. Ensure that the registers are initialized to desired values and the clocks are released in a controlled manner to ensure safe startup.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Controlling and Synchronizing Device Startup	
	Are there any untimed resets being used in the design?	Untimed resets can produce unpredictable behavior in the design when asserted or more importantly deasserted. Ensure safe and known behavior for all design paths including reset.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Use of Untimed Resets	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Has the design avoided the use of combinational feedback; (i.e. the looping of combinational processes)?	Combinatorial loops can create timing hazards that are difficult to predict or avoid and prevents optimizations to the design to improve power and other characteristics thus should be avoided. Run the *check_timing -verbose -override_defaults loops -file report.txt* command to analyze for any combinatorial loops in the design.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Avoid Combinational Loops	
3.3.8	**Coding Style: Performance**		UG949: Design Creation	
	Is logic connecting to hard resources (DSP48, BRAMs) pipelined to achieve required frequency?	Critical timing paths often occur from larger blocks like BRAMs and DSPs and in particular, the output paths from these components. It is suggested to pipeline the outputs of such components in order to improve the overall performance of these paths. Run the *report_design_analysis* Tcl command, which reports if BRAMs or DSPs are in the critical paths and if the optional pipeline registers are used.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Performance > Pipelining Considerations	
	Have registers been evaluated/optimized for implementation into SRL16 resources?	Shift Register LUTs (SRLs) often help reduce FF resource needs for data buffering and pipeline balancing of sections of the design. Evaluate when to use these resources and when to use standard registers to best meet goals for performance, area and power.	UG949: Design Creation > RTL Coding Guidelines > Coding Shift Registers and Delay Lines	
	Has the recommended coding style been followed for most efficient element inference?	Best results are seen when coding is performed with knowledge and understanding of how it will map into the underlying architecture. Understand how coding decisions will result in the target architecture.	UG949: Design Creation > RTL Coding Guidelines > Know What You Infer	
3.3.9	**Coding Style: Power**		UG949: Design Creation	
	Are areas of the device not in use at all times coded to be gated off?	If there are significant sections of the design that are not in use for sustained periods of time, coding in an enable at the clock domain for module level can gate off unnecessary switching thus saving power. Evaluate all blocks in the design for this potential power saving technique.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Power > Gate Clock or Data Paths	
	Have the low power options been considered for memory creation?	BRAM dynamic power is directly proportional to the time it is enabled. By disabling the BRAM when not needed, significant power savings can be seen. BRAM write mode also impacts power, and selecting the proper mode allows the BRAM to function most efficiently in terms of power. Review BRAM configurations and ensure the setting match functional needs while not dissipate unneeded power.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles to Improve Power > Best Practices for Block RAM in Design	
3.3.10	**Coding Style: Debug**			
	Have you thought about which debug cores you plan to use and what signals you plan to monitor? Have you placed the "MARK_DEBUG" attribute on the signals you wish to preserve and monitor? Have you instantiated any debug cores? If so list the in the Notes section.	On an RTL or Synthesized design set the MARK_DEBUG constraint on the nets to be debugged.	UG908: Programming and Debugging > In-System Logic Design Debugging Flows	
3.4	**Constraints Creation**			
3.4.1	**Organizing Design Constraints**		UG949: Design Creation	
	Are all the necessary constraint files listed in the project or in the script?	Run *report_compile_order –constraints* to review the constraint files processing sequence or visualize the same info in the Vivado IDE (see the Compile Order view in the Sources window).	UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Recommended Constraint Files UG903: Using Constraints > Constraints Methodology > Organizing Your Constraints UG945: Using Constraints Tutorial	
	Does the sequence of timing constraints applied to my design match the recommended one?	Use the Timing Constraints Editor in the Vivado IDE to visualize the timing constraints order loaded in memory or type write_xdc to write all the valid constraints to a file for review.	UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Recommended Constraints Sequence UG903: Using Constraints > Constraints Methodology > Ordering Your Constraints UG903: Using Constraints > XDC Precedence	
	Does the design need different timing constraints for synthesis and for implementation?	Implementation constraints usually need to be more specific than synthesis constraints (point-to-point exceptions, physical constraints). Also, when elaborated netlist and	UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Creating Synthesis Constraints	

ID	Description	Actions to Take	Relevant Information	Status and Notes
		physical constraints). Also, when elaborated netlist and synthesized netlist names do not match, separate constraints must be used for synthesis and implementation.	UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Creating Implementation Constraints UG903: Using Constraints > Constraints Methodology > Creating Synthesis Constraints UG903: Using Constraints > Constraints Methodology > Creating Implementation Constraints	
3.4.2	**Defining Clocks**		UG949: Design Creation	
	Have all the primary clocks been defined on netlist objects that correspond to the design boundary?	Primary clocks are typically defined on primary input ports or on the output of Gigabit Transceiver blocks (recovered clock with its own phase).	UG949: Design Creation > Working with Constraints > Defining Clock Constraints > Creating Primary Clocks UG903: Using Constraints > Defining Clocks QuickTake: Design Constraints Overview QuickTake: Creating Basic Clock Constraints	
	Is any primary clock defined in the transitive fanout of another primary clock? (to be avoided)	This type of clocking scheme should be avoided: the second clock blocks the propagation of the first clock, and the insertion delay (skew) of the clocks becomes inaccurate during timing analysis.	UG949: Design Creation > Working with Constraints > Defining Clock Constraints > Creating Primary Clocks UG903: Using Constraints > Defining Clocks	
	Have the generated clocks been defined in the transitive fanout of their master clock?	When there is no logical path between a generated clock and its master clock, the insertion delay of the generated clock cannot be computed and the skew becomes inaccurate during timing analysis.	UG949: Design Creation > Working with Constraints > Defining Clock Constraints > Creating Generated Clocks UG903: Using Constraints > Defining Clocks QuickTake: Creating Generated Clock Constraints	
	Are some generated clocks defined on pins where the Vivado Design Suite automatically creates some generated clocks (e.g. MMCM, BUFR, …)? If so, does their waveform match the one the tool would have automatically generated?	Validate that generated clock constraints match the auto-generated clock waveform.	UG949: Design Creation > Working with Constraints > Defining Clock Constraints > Creating Generated Clocks UG903: Using Constraints > Defining Clocks QuickTake: Creating Generated Clock Constraints	
3.4.3	**Constraining Input and Output Ports**		UG949: Design Creation	
	Are all the Input Ports constrained with an input delay?	Validate that all input ports have input delay constraints. Generate a check_timing report and review the no_input_delay/partial_input_delay sections.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Defining Input Delays UG903: Using Constraints > Constraining I/O Delay QuickTake: Setting Input Delay	
	Are all the Output Ports constrained with an output delay?	Validate that all output ports have output delay constraints. Generate a check_timing report and review the no_output_delay/partial_output_delay sections.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Defining Output Delays UG903: Using Constraints > Constraining I/O Delay QuickTake: Setting Output Delay	
	Have all the input and output delay constraints been defined relative to the appropriate clock?	Validate that all input and output constraints are defined relative to a clock used as a reference by the external device. This clock is also used as the internal IO clock in most cases.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Choosing the Reference Clock UG903: Using Constraints > Constraining I/O Delay	
	Are the input and output delays for synchronous interface matching the edge-aligned or center-aligned templates?	Validate that all input and output delay constraints match edge or centered aligned templates.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Verifying Delay Constraints	
	Are the active edges of the input and output delay constraints matching the specs of the board?	Validate that all input and output delay constraints are defined relative to the clock edges specified in the external device datasheet.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Choosing the Reference Clock > Rising and Falling Reference Clock Edges	
	Are the ports related to an internally modified clocks constrainted relative to a virtual clock?	Validate that ports related to internally modified clocks are constrained relative to a virtual clock.	UG949: Design Creation > Working with Constraints > Constraining Input and Output Ports > Choosing the Reference Clock	
3.4.4	**Defining Clock Groups**		UG949: Design Creation	
	Have clock groups been used instead of false paths to define clock domains that are exclusive or asynchronous to each other?	Clock groups should be used instead of false paths to identify asynchronous and exclusive clocks. Validate that clock groups have been used rather than false paths whenever possible.	UG949: Design Creation > Working with Constraints > Defining Clock Groups and CDC Constraints > Constraining Exclusive Clock Groups UG903: Using Constraints > Defining Clocks > Clock Groups	
	Have clock groups been used to define clocks that cannot logically or physically co-exist at the same time?	Exclusive clocks are clocks that propagate on the same hardware resource (clock path). By default, timing analysis reports paths between them even if it usually does not correspond to a realistic situation.	UG949: Design Creation > Working with Constraints > Defining Clock Groups and CDC Constraints > Constraining Exclusive Clock Groups	

ID	Description	Actions to Take	Relevant Information	Status and Notes
		correspond to a realistic situation.	gn Creation UG903: Using Constraints > Defining Clocks > Clock Groups	
	Have any set_max_delay constraint been overridden by clock groups or false paths constraints?	Use the report_timing and the report_exceptions command to identify set_max_delay constraints that have been overridden by set_clock_groups or set_false_path (higher precedence). Correct the constraints if the max delay constraint is needed.	UG949: Design Creation > Working with Constraints > Defining Clock Groups and CDC Constraints > Constraining Asynchronous Clock Groups and Clock Domain Crossings > Constraints on Individual CDC Paths	
3.4.5	**Specifying Exceptions**		UG949: Design Creation	
	Do all min and max delay constraints only use valid startpoints and endpoints to avoid path segmentation?	Review the log files and messages when the constraints loaded in memory to find the corresponding warnings. Fix any constraint that introduces path segmentation.	UG949: Design Creation > Working with Constraints > Specifying Timing Exceptions > Adding Min and Max Delay Constraints QuickTake: Setting False Path Exceptions UG903: Using Constraints > Timing Exceptions > Min/Max Delays	
	When defining a multicycle constraint, has the default hold relationship that is derived from the setup relationship been reviewed and adjusted accordingly?	Run hold timing analysis on the paths covered by the multicycle constraint and adjust the hold requirement if needed by using another multicycle constraint.	UG949: Design Creation > Working with Constraints > Adding Multicycle Path Constraints UG903: Using Constraints > Timing Exceptions > Multicycle Paths QuickTake: Setting Multicycle Path Exceptions	
	Has a multicycle constraint been considered to constrain the logic clocked by two clocks of the same period but having a phase shift relationship?	Timing analysis uses the closest launch and capture edges by default. When phase shift is introduced, multicycle path constraints need to be used to adjust the setup edges as needed.	UG949: Design Creation > Working with Constraints > Adding Multicycle Path Constraints UG903: Using Constraints > Timing Exceptions > Multicycle Paths	
	Do you use net names to find cells and pins used to describe some timing exceptions? Since nets can be optimized during the implementation flow, did you consider finding objects in a different way?	Avoid using net names to define constraints. Use logic objects when possible.	UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Creating Synthesis Constraints	
	Did you consider using the simplest patterns to find startpoints, through points and endpoints?	Complex patterns and long list of names are difficult to validate and debug. Plus they can impact runtime in some cases.	UG949: Design Creation > Working with Constraints > Specifying Timing Exceptions > Timing Exceptions Guidelines	
	Has Max Delay Datapath Only been used instead of Clock Groups or False Path? If so, is it purely on Clock Domain Crossing paths?	Ensure that Max Delay Datapath Only constraints are only used on paths between asynchronous clocks, and not on paths within a synchronous domain.	UG949: Design Creation > Working with Constraints > Specifying Timing Exceptions > Adding Min and Max Delay Constraints QuickTake: Setting False Path Exceptions UG903: Using Constraints > Timing Exceptions > Min/Max Delays	
	Did you consider the precedence between the timing exceptions and verified it by running timing analysis?	Verify timing exceptions by running timing analysis. Use the report_exceptions command to review all the exceptions loaded in memory and any potential conflicts.	UG949: Design Creation > Working with Constraints > Specifying Timing Exceptions > Timing Exceptions Guidelines UG906: Design Analysis and Timing Closure Techniques > Timing Analysis Features QuickTake: Advanced Clock Constraints and Analysis	
3.4.6	**Defining Block-Level or IP-Level Constraints**		UG949: Design Creation	
	Did you define placement constraints when developing block-level or IP-level constraints? If so, list them in the Status and Notes field.	Identify any physical constraints such as I/Os or Floorplanning assigned.	UG949: Design Creation > Working with Constraints > Creating Block-Level Constraints	
	Did you define timing exceptions between clocks that are not entirely bounded to the IP or block? If so, list them in the Status and Notes field.	Identify timing exceptions between clocks that are not entirely bounded to an IP or block. Such constraints must be avoided as they can impact paths outside the IP.	UG949: Design Creation > Working with Constraints > Creating Block-Level Constraints	
	Have all the IP constraints been reviewed?	Ensure that IP .xdc files are properly included in the design and are scoped to correct hierarchy IP locations.		

ID	Description	Actions to Take	Relevant Information	Status and Notes
4.1	**Running Synthesis**			
4.1.1	**Synthesis Best Practices**		UG949: Using the Vivado Design Suite UG949: Design Creation UG949: Implementation	
	Have you verified your timing constraints? Are the clock periods realistic? Are timing constraints from the .xdc file applied during synthesis?	Timing constraints syntax should be verified with the elaborated design. The quality of results should be analyzed after synthesis to gauge the feasibility of the clock performance specified. Consider using the "Baselining" validation technique to assign and validate constraints.	UG949: Implementation > Synthesis > Accurate Timing Constraints UG949: Design Creation > Working with Constraints > Organizing the Design Constraints > Creating Synthesis Constraints UG949: Implementation > Timing Closure > Baselining the Design	
	Are you using third-party synthesis?	Xilinx supports third-party synthesis tools. Appropriate IP output products are generated to support using third-party synthesis with Xilinx IP. Run the *link_design* command to prepare the design to run many of the commands described in the Validating Synthesized Design section below.	UG901: Synthesis > Vivado Synthesis > Using Synthesis > Using Third-Party Synthesis Tools with Vivado IP	
4.1.2	**Using Synthesis Attributes**		UG949: Implementation	
	Are synth_design settings needed or have been explored? Have you checked all of your synthesis attributes present in the RTL? Do you understand why each of them is needed and if it is relevant for Vivado Synthesis? Are attributes such as KEEP, DONT_TOUCH, MAX_FANOUT SHREG_EXTRACT, ROM_STYLE, RAM_STYLE, FSM_SAFE_STATE needed and used correctly?	Familiarize yourself with the Vivado synthesis attributes and their usage. Apply synthesis attributes where desired and revisit the need for attributes when moving to a new FPGA family or a new software release. Use the "Find in Files" capability in the Vivado Text Editor to search for Attribute names in your design. Ensure that all attributes are being applied properly.	UG949: Implementation > Synthesis Attributes UG901: Synthesis > Synthesis Attributes	
	Have you checked your tristates and made sure that you have not set any DONT_TOUCH or KEEP_HIERARCHY attributes on levels of hierarchy with tristates in lower levels?	Setting these attributes can hinder the tools ability to perform logic optimization across hierarchical boundaries. Ensure that you have not set them inadvertently.	UG949: Implementation > Synthesis Attributes	
4.2	**Validating the Synthesized Design**		UG949: Implementation	
	What needs to be checked to ensure the netlist for implementation is of good quality?	Analysis should be performed after synthesis to ensure the design doesn't have any major setup and hold violations or latch inferences before starting place and route.	UG949: Implementation > Moving Past Synthesis	
	Have you run DRC on the synthesized design? Has the post-synthesis DRC report been evaluated?	Run the *report_drc* command or IDE equivalent after synthesis. You must resolve all Critical Warnings in DRC. Not taking this step results in a failure of write_bitstream.	UG949: Implementation > Moving Past Synthesis > Review and Clean DRCs UG906: Design Analysis and Closure Techniques > Logic Analysis Within the IDE > Using Report DRC	
4.2.1	**Meeting Post Synthesis Timing**		UG949: Implementation	
	Does the design close timing post-synthesis?	Run the *report_timing_summary Tcl* command or IDE equivalent after synthesis.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing UG906: Design Analysis and Closure Techniques	
	Are all clocks properly defined with valid constraints in place? Are multi-cycle paths correctly constrained? Are all the Clocks Domain Crossing properly constrained?	Continually validating that constraints are defined properly and are met will help close timing during implementation. Validating the CDC constraints is very important for timing closure. Run the *report_clock_interaction* Tcl command or use the Clock Interaction report in the Vivado IDE to review the constraints.	UG949: Implementation > Timing Closure > Checking That Your Design is Properly Constrained UG949: Implementation > Timing Closure > Baselining the Design UG949: Implementation > Timing Closure > Understanding Timing Reports UG906: Design Analysis and Closure Techniques	
4.2.2	**Reviewing Device Utilization**		UG949: Implementation	
	Is the design post-synthesis utilization reasonable?	Run the *report_utilization* Tcl command or Vivado IDE equivalent after synthesis. Higher utilized designs may impact performance.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing > Reviewing Utilization UG906: Design Analysis and Closure Techniques > Design Analysis Within the Vivado IDE > Analyzing Device Utilization Statistics	
	Has the complete system been reviewed for resource utilization including clocks, BRAMS, DSP48s, FFs, LUTs? Will the complete design meet the targets?	Perform early resource estimation to validate the design will fit in the target device.	UG949: Implementation > Timing Closure > Timing Closure Criteria UG906: Design Analysis and Closure Techniques > Design Analysis Within the Vivado IDE > Analyzing Device Utilization Statistics	
4.2.3	**Reviewing Clock Trees**		UG949: Implementation	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Is the clock buffer utilization expected?	Run the *report_clock_utilization* Tcl command or Vivado IDE equivalent after synthesis.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing > Reviewing Clock Trees UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports	
	Is the clock tree topology expected?	Run the *report_clock_networks* Tcl command or *Vivado* IDE equivalent after synthesis.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing > Reviewing Clock Trees UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports	
	Are you using sync vs. async resets on your registers?	Run the custom script *report_reset_signals.tcl.*	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing	
	Have you had synchronization circuitry for asynchronous Clock Domain Crossing (CDC) paths?	Make sure the asynchronous Clock Domain Crossing paths safe. In general, there are two popular methods to allow data to cross asynchronous clock domains safely. If only a single bit is needed or if methods such as grey-coding are used to transfer more than one bit of related data, register synchronizers can be inserted to reduce the Mean Time Before Failure (MTBF) of the circuit. For multiple bits of data (that is, a bus), the generally recommended practice is to use an independent clock (asynchronous) FIFO to safely transfer data from one domain to another.	UG949: Design Creation > RTL Coding Guidelines > Coding Styles for Higher Reliability > Clock Domain Crossings > Asynchronous Domain Crossing	
	Have you fully pipelined your add/mult structures to infer pipelined DSP48s?	Run the *report_design_analysis* Tcl command.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing	
	Are the output Block RAM registers used in order to reduce the read time?	Run the *report_design_analysis* Tcl command.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing	
	Have you checked the inferred SRLs and made sure they are what you want?	Validate the SRL inference was handled properly in synthesis.	UG949: Implementation > Moving Past Synthesis > Meet Post-Synthesis Timing	
4.3	**Implementing the Design**			
4.3.1	**Implementation Steps and Options**		UG949: Implementation	
	Have run strategies been explored to improve performance or power?	Experiment with implementation strategies and directives to improve design targets.	UG949: Implementation > Implementing the Design > Project Mode vs. Non-Project Mode Options > Strategies UG949: Implementation > Implementing the Design > Project Mode vs. Non-Project Mode Options > Directives QuickTake: Vivado Implementation Directives and Strategies UG901: Synthesis > Vivado Synthesis > Using Synthesis UG904: Implementation > Preparing for Implementation > About the Vivado Implementation Process UG904: Implementation > Implementing the Design > About Implementation Commands	
4.3.2	**Logic Optimization**		UG949: Implementation	
	Are *opt_design* settings and/or directives needed or have been explored?	Run the *opt_design -help* Tcl command or Vivado IDE Implementation Settings to view the command options.	UG949: Implementation > Implementing the Design > Logic Optimization UG904: Implementation > Implementing the Design > Logic Optimization	
4.3.3	**Power Optimization**		UG949: Implementation	
	Are *power_opt_design* settings and/or directives needed or have been explored?	Run the *report_powe* r Tcl command before/after optimization and then run the *report_power_opt* Tcl command.	UG949: Implementation > Implementing the Design > Power Optimization UG904: Implementation > Implementing the Design > Power Optimization UG907: Power Optimization and Analysis	
4.3.4	**Placement**		UG949: Implementation	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Are *place_desig* n settings and/or directives needed or have been explored?	Run the *place_design -help* Tcl command or Vivado IDE Implementation Settings to view the command options.	UG949: Implementation > Implementing the Design > Placement UG904: Implementation > Implementing the Design > Placement	
4.3.5	**Physical Optimization**		UG949: Implementation	
	Is *phys_opt_design* along with settings and/or directives needed or have been explored?	Run the *phys_opt_design -help* Tcl command or Vivado IDE Implementation Settings to view the command options.	UG949: Implementation > Implementing the Design > Physical Optimization UG904: Implementation > Implementing the Design > Physical Optimization	
4.3.6	**Routing**		UG949: Implementation	
	Are the *route_design* settings and/or directives needed or have been explored?	Run the *route_design -help* Tcl command or Vivado IDE Implementation Settings to view the command options.	UG949: Implementation > Implementing the Design > Routing UG904: Implementation > Implementing the Design > Routing	
4.3.7	**Completing and Managing Runs**		UG949: Implementation	
	Has the design status been evaluated post-route to ensure completion?	Run the *report_route_status* Tcl command to ensure all signals are properly routed.	UG949: Implementation > Implementing the Design > Routing > Intermediate Route Results UG904: Implementation UG906: Design Analysis and Closure Techniques	
	Are you familiar with the process and commands to store intermediate results in both Project and Non-Project mode?	Understand how to save and manage intermediate results of the design.	UG949: Implementation > Implementing the Design > Intermediate Steps and Checkpoints UG904: Implementation > Preparing for Implementation > Using Design Checkpoints to Save and Restore Snapshots	
	Are you familiar with implementation techniques such as Incremental Compile and Re-Entrant Routing? Do you intend to use them?	Incremental compile enables you to leverage previously successful implementation results when performing minor logic changes. Understand the process and ramifications and decide whether you wish to use it. If routing problems exist, explore re-entrant routing.	UG949: Implementation > Implementing the Design > Incremental Flows UG949: Implementation > Implementing the Design > Routing > Using Re-Entrant Route Mode UG904: Implementation > Analyzing and Viewing Implementation Results > Modifying Implementation Results	
4.4	**Design Closure**			
4.4.1	**Power Estimation and Optimization**		UG949: Implementation	
	Have XPE or *report_power* been run on the 90% complete design? Does the power meet budget?	Verify the that power budget estimate is reasonable. Review the Power optimization documentation if power estimation is too high.	UG949: Implementation > Power QuickTake: Power Estimation and Analysis Using Vivado UG907: Power Optimization and Analysis > Estimating Power - Vivado Design Flow Stage	
	Are constraints in place to accurately analyze power?	Review high fanout signals like global reset and clock enable.	UG949: Implementation > Power > Best Practices for Accurate Power Analysis	
4.4.2	**Power Optimization During Implementation**		UG949: Board and Device Planning	
	Have you reviewed "clock enable" management as a solution for reducing/controlling dynamic power? Have you considered or attempted to use the Implementation options for power optimization?	For sections of the design that are not needed for sustained periods of time, providing either an internal or external disable can reduce unnecessary dynamic power in areas of the design not needed at a given time. Familiarize yourself with the various Vivado IDE and Tcl commands for power optimization. One of them is to use the *power_opt_design* command on either the entire design or specific modules.	UG949: Board and Device Planning > Worst Case Power Analysis Using Xilinx Power Estimator (XPE) UG949: Implementation > Power > Power Optimization UG907: Power Analysis and Optimization > Power Analysis and Optimization in the Vivado Design Suite > Power Optimization Feature	
	Have you examined the results of the intelligent clock gating performed by the *power_opt_design* command?	Experiment early in the design cycle with the Vivado IDE and Tcl commands for power optimization and evaluate power and performance impacts. Run the *report_power_opt* Tcl command or use *Tools->Report->Report Power Optimization* in the Vivado IDE.	UG949: Implementation > Power > Power Optimization UG907: Power Analysis and Optimization > Power Analysis and Optimization in the Vivado Design Suite > Power Optimization Feature	
4.4.3	**Preliminary Timing Closure**		UG949: Design Creation UG949: Implementation	
	Does the 90% complete design meet the target speed?	Review the timing closure criteria to ensure the design is feasible as defined.	UG949: Implementation > Timing Closure > Timing Closure Criteria UG906: Design Analysis and Closure Techniques > Performing Timing Analysis	
	Has the design been analyzed for trade-offs in technology choices?	Explore the device technology options to see if another speed grade would help.	UG949: Implementation > Timing Closure > Reviewing Technology Choices	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Have you run the Design Methodology Timing DRCs to determine if your code have conditions that may limit functionality or performance?	The Vivado Design Suite has an ever expanding set of DRC checks that can be run on the Implemented design. After implementation, run the Design Methodology timing DRCs and analyze the Messages view to find any potential issues.	UG949: Implementation > Implementing the Design > Project Mode vs Non-Project Mode Options > Running Methodology DRCs UG906: Design Analysis and Closure Techniques > Logic Analysis Within the IDE > Validating Design Methodology Logic DRCs	
	Have all warnings and error messages been addressed? List the Messages that were not resolved.	Review and address all Errors and Critical Warning messages. Review all Warning messages.	UG949: Implementation > Timing Closure > Checking That Your Design is Properly Constrained UG906: Design Analysis and Closure Techniques > Logic Analysis Within the IDE > Validating Design Methodology Logic DRCs	
	Are you familiar with the connection between clock period (waveform), setup relationships, and hold relationships?	Understanding how to configure the timing analysis and to interpret Vivado timing reports will help you identify design or constraint issues. Familiarize yourself with timing analysis and reporting options.	UG949: Implementation > Timing Closure > Understanding Timing Reports UG906: Design Analysis and Closure Techniques > Logic Analysis Within the IDE > Validating Design Methodology Logic DRCs	
	Have you baselined the design before moving further in to the timing analysis? Have you performed the Baslining and Constraints Validation procedure?	Baselining is a technique to define and validate timing constraints in a sequential manner. It can help ensure timing constraints are properly assigned and implemented. Perform the step-by-step procedure outlined in Appendix A of the *UltraFast Design Methodololgy Guide for the Vivado Design Suite* .	UG949: Implementation > Timing Closure > Baselining the Design UG949: Appendix A: Baselining and Constraints Validation Procedure	
	Have you validated again that the primary and generated clocks are completely and properly defined? Are the exclusive clock groups properly defined?	Run the *check_timing* Tcl command or validate by tracing the clocks in the Vivado Schematic viewer.	UG949: Implementation > Timing Closure > Checking That Your Design is Properly Constrained UG949: Implementation > Timing Closure > Baselining the Design UG949: Implementation > Timing Closure > Understanding Timing Reports UG906: Design Analysis and Closure Techniques > Timing Analysis Features UG906: Design Analysis and Closure Techniques > Viewing Reports and Messages > Creating Design Related Reports	
	Have you validated again that the asynchronous clock groups and multi-cycle paths are properly constrained?	Run the *report_clock_interaction* Tcl command or use the Clock Interaction report in the Vivado IDE. Review clock domain crossing constraints, WNS path requirements and WHS path requirements. Are they reasonable?	UG949: Implementation > Timing Closure > Checking That Your Design is Properly Constrained UG949: Implementation > Timing Closure > Debugging and Fixing Timing Issues > Defining Baseline Constraints	
	Are you familiar with how to identify the paths with the worst cell delay in the design and how to change a net delay model for timing analysis?	Run timing analysis with no net delay, with estimated delays only or with actual net delays. Review the RTL to reduce the logic delay, or identify appropriate constraints for meeting timing on these paths.	UG906: Design Analysis and Closure Techniques > Design Analysis Techniques > Identifying the Longest Logic Delay Paths in the Design	
	Have attribute settings for MMCL/PLL been checked for violations (e.g. performance setting violates input or output clock frequency range)?	Review the Messages and analyze the settings for MMCMs and PLLs to ensure they are correct.	UG949: Design Creation > RTL Coding Guidelines > Clocking > Controlling the Phase, Frequency, Duty-Cycle, and Jitter of the Clock > Using Clock Modifying Blocks (MMCM and PLL) UG949: Implementation > Timing Closure > Understanding Timing Reports UG906: Design Analysis and Closure Techniques	
	Are you familiar with how to perform and review hold timing analysis before and after route?	Vivado timing analysis can be run in a variety of ways to pinpoint specific issues, such as hold violations or impact of hold fixing on setup violations. Familiarize yourself with the various options and run timing analysis with options, if needed.	UG906: Design Analysis and Closure Techniques > Design Analysis Techniques > Determining if Hold-Fixing is Negatively Impacting the Design	
	Has the design been analyzed for Control Set Usage?	Excessive number of control signals can degrade the placement quality by reducing the number of valid placement solutions. Examine the control sets in the design and take steps to reduce them if needed. Synthesis provides some control on the minimum fanout of clock enable and synchronous set/reset signals to be preserved.	UG949: Implementation > Timing Closure > Control Signals and Control Sets	
4.4.4	**Final Timing Closure**		UG949: Implementation	
	Are your timing constraints signoff quality?	Review the check_timing report (also included in the timing summary report). Also review the User Ignored Paths and	UG949: Implementation > Timing Closure > Timing Closure Criteria	

ID	Description	Actions to Take	Relevant Information	Status and Notes
		summary report). Also review the User Ignored Paths and Unconstrained Paths section of the timing summary report to waive any ignored path.	lementation UG906: Design Analysis and Closure Techniques > Performing Timing Analysis	
	Does the design meet setup and hold for all signals?	Review the WNS/TNS/WHS/THS/WPWS/TPWS numbers displayed in the timing summary report. Review the various sections of the summary report for more details on the remaining violations.	UG949: Implementation > Timing Closure > Timing Closure Criteria UG906: Design Analysis and Closure Techniques > Performing Timing Analysis	
4.4.5	**Floorplanning**		UG949: Implementation	
	Are you considering floorplanning to try and improve timing results? If so, did you consider the data flow through the design? Is the design logic hierarchy set up well for your floorplanning strategy? If using SSI technology have you considered floorplanning to assist with design partitioning? List any floorplanned logic resources in the Status and Notes field.	The Vivado Design Suite enables you to floorplan critical modules or logic in your design. Verify that the design logic hierarchy been constructed well to support your desired floorplanning. Proper Floorplanning requires knowledge of the design, the key interfaces, as well FPGA resources and uses. A suboptimal floorplan can hinder performance more than help it. Take care when floorplanning logic. Look carefully at SLR crossings with SSI technology. Use the Vivado IDE to visualize the I/O connectivity and the top-level data flow as well as estimate the required resources.	UG949: Implementation > Timing Closure > Considering Floorplan UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning	
	Is there a need to manually place any of the big blocks like PCIe, BRAM and DSP48s? Have you placed any logic in the design?	The Vivado Design Suite enables you to lock down resources to specific FPGA sites. Consider hand placing critical logic such as BRAM and DSP to better group them and to align with other FPGA resources.	UG949: Implementation > Timing Closure > Considering Floorplan > Preserving Placement and Routing UG906: Design Analysis and Closure Techniques > Design Closure Techniques > Floorplanning > Locking Specific Logic to Device Sites	

ID	Description	Actions to Take	Relevant Information	Status and Notes
5.1	**Device Configuration**			
5.1.1	**JTAG Interface**		UG949: Board and Device Planning	
	Are the JTAG pins exposed to a connector that is compatible with a Xilinx-supported programming cable?	JTAG pins can be used for programming and debug of the FPGA. Having the JTAG interface available is important for design bring-up and debug. The connector should be compatible with the Xilinx Platform Cable USB II or supported Diligent USB cable.	DS593: Platform Cable USB II Data Sheet Product family Configuration User Guide	
	Can the Xilinx FPGA be isolated in the JTAG chain?	The capability to isolate the FPGA in a JTAG chain will accelerate the debug process should an issue be encountered. Not all devices support the optional pins or JTAG features (i.e. TRST is an optional signal and is not supported by Xilinx FPGAs).	Product family Configuration User Guide	
5.1.2	**General Configuration Pin Handling**		UG949: Board and Device Planning	
	Was the signal integrity for key configuration signals reviewed during PCB layout?	Run IBIS model simulation to validate signal integrity. (i.e. CCLK, TCK pins).	Product family Printed Circuit Board User Guide	
	Have shared banks and multi-function pins in use been reviewed to ensure there are no conflicts? (i.e.. DDR memory and parallel NOR voltage targets)	Some configuration interfaces use multiple purpose IO and care should be taken to ensure the pins being reused are targeting the same voltage after configuration as during. DDR devices can require 1.5V so care should be taken in these applications to align the bank voltages appropriately.	Product family Printed Circuit Board User Guide Product family Configuration User Guide	
	Have the selected configuration interface connections been verified, including the mode selection?	Verify that the proper interface connections have been made (address, data and control signals). Ensure the address alignment is verified, for example some parallel NOR devices start at A1 for the LSB where the FPGA starts with A0. Ensure the control signals are connected properly and flash reset or write protection signals are tied appropriately.	Product family Configuration User Guide Target flash memory data sheet	
	Are the M2, M1, and M0 mode pins held at a constant DC voltage level during and after configuration?	Verify that the pins are held at a constant DC voltage value that matches the desired mode selection.	Product family Configuration User Guide	
	Have you double-checked compliance with the Configuration checklist items in the Board and Device Planning section of this Checklist?	Review the Configuration section of the Board and Device Planning section of this Checklist.		
	If EMCCLK is used for fastest performance has this signal been instantiated in the design since it is a multi-function pin?	The configuration internal CCLK has a large tolerance range. If configuration speed is critical the EMCCLK will help achieve the fastest times possible. Since the pin is on a dual-purpose I/O an I/O standard must be defined.	Product family Configuration User Guide	
	If using SelectMAP mode, have you ensured that the RDWR_B and CSI_B are not used after configuration or that the ABORT sequence is not entered?	Care should be taken not to enter the ABORT after configuration by sequencing the RDWR_B and CSI_B. See the Product family configuration user Guide for details.	Product family Configuration User Guide	
5.1.3	**Security**		UG949: Configuration and Debug	
	Is FPGA eFUSE programming required for prototyping or production?	If eFUSE programming is required, refer to the product family Configuration User Guide and the Xilinx software manual for options. Ensure that access for supported Xilinx cable solution is provided on-board.	Product family Configuration User Guide DS593: Platform Cable USB II Data Sheet	
	Is bitstream protection and security a priority?	If encryption is required review the product family configuration User Guide and application notes on the options available.	Product family Configuration User Guide XAPP1084: Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs XAPP1239: Using Encryption to Secure a 7 Series FPGA Bitstream	
	Are advanced security features planned to be used such as disabling the JTAG interface?	Review the feature carefully as special care should be taken if disabling the JTAG interface. This should only be used in advanced security applications.	Product family Configuration User Guide WP365: Solving Today's Design Security Concerns	
5.1.4	**Configuration Solution Selection**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Have the application configuration requirements been reviewed and trade-offs understood?	Ensure the configuration mode trade-offs have been reviewed so that the most optimal solution can be selected for pin count, configuration speed, and feature support.	Product family Configuration User Guide	
	Does the application need to store multiple configuration images?	MultiBoot is a popular option and there are different ways to implement a multiple image system. See the product family configuration User Guide for recommendations. Using the technique of partial reconfiguration for your design will also need to store multiple configuration images.	Product family Configuration User Guide UG909: Partial Reconfiguration > Configuring the Device > Configuration Modes	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	Have the power supply ramp time been considered to reduce configuration time?	Applications where configuration time is critical may want to consider using a restricted power supply ramp time to guarantee a faster configuration time.	Product family Configuration User Guide Product family Data Sheet for TPOR	
	How much storage is required for the configuration bitstreams?	Based on the bitstreams targeted calculate the storage size required. Ensure the flash selected or storage device has enough memory to accommodate the image(s) targeted. Some application require padding between images and this should be accommodated in the size estimation. See the product family configuration User Guide for details.	Product family Configuration User Guide UG835: Vivado Design Suite Tcl Command Reference Guide > Tcl Commands Listed Alphabetically > write_bitstream UG909: Partial Reconfiguration > Configuring the Device > Configuration Time	
	Is in-system flash programming required for proto-typing and debug or production? If so, has a supported flash device been selected?	For supported flash families with Xilinx in-direct programming, see the software manuals.	For SPI and BPI Configuration and Indirect Programming References, see: UG908: Programming and Debugging Product family configuration User Guide XAPP587: BPI Fast Configuration and iMPACT Flash Programming with 7 Series FPGAs XAPP586: Using SPI Flash with 7 Series FPGAs XAPP1220: UltraScale FPGA BPI Configuration and Flash Programming	
5.1.6	**File Generation and Programming**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Have the required file generation options been verified for bitstream generation (.bit) and flash programming file generation (i.e. .mcs) ?	Dependent on the configuration mode selected you will need to enable appropriate file generation options. These options include selecting the appropriate bus width (i.e. x1, x2, x4, x8, 16, or x32) dependent on the mode. In addition, you need to make selections for advanced features like encryption, synchronous bpi mode, 32 bit enabled spi mode etc.	UG908: Programming and Debugging UG936: Programming and Debug Tutorial QuickTake: How to Use the "write_bitstream" Command in Vivado QuickTake: Setting and Editing Device Properties Product family Configuration User Guide	
	Have the DRCs warnings received during configuration bitstream generation been reviewed and corrected?	Review the DRC warnings and ensure they are corrected.	UG908: Programming and Debugging QuickTake: Setting and Editing Device Properties Product family Configuration User Guide	
	If an external flash configuration mode (SPI or BPI) is used, will the flash be reused after configuration or is Persist being used?	If the flash is being reused after configuration take care to review the handling of dedicated configuration IO (such as the SPI signals that can be accessed after configuration using the STARTUP primitive) or if Persist is being used be certain to review the software manual and configuration User Guide for details.	UG949: Configuration and Debug > Configuration > Remote Update Product family Configuration User Guide	
	Does the FPGA need to be held off from configuring at power up to ensure power supplies are stable?	The INIT_B pin can be used to hold off configuration. If this is required (i.e. to ensure flash is powered up first before the FPGA tries to retrieve configuration data) then refer to the product family configuration User Guide.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
	Is an internal configuration clock desired? If so can a clock tolerance be accepted by application configuration time requirements?	If cost is a factor, using the internal configuration clock will prevent the need of an external clock. However, if the fastest configuration time is required, EMCCLK should be evaluated instead of the internal configuration clock. The internal configuration clock has a clock tolerance that will reduce the max freq.	Product family Configuration User Guide XMP277: 7 Series Schematic Review Checklist XTP344: UltraScale Architecture Schematic Review Checklist	
5.2	**Configuring and Debugging the Hardware Design**			
5.2.1	**Configuring the Design to Debug**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Does the design meeting timing prior to adding the debug core?	Ensure design meeting timing prior to adding the debug core. Verify Timing constraints. The Logic Debug Analyzer should not be used to debug timing issues in the design.	UG908: Programming and Debugging > Debugging Logic Designs in Hardware	
	Have you marked the nets to be debugged "MARK_DEBUG" ?	On a RTL or Synthesized design set the MARK_DEBUG constraint on the nets to be debugged.	UG908: Programming and Debugging > In-System Logic Design Debugging Flows	
	Did you ensure the clock domain associated with the debug core in synchronous to the nets in the debug core?	Ensure the clock domain associated with the nets in a debug core is synchronous to the nets in that core.	UG908: Programming and Debugging > In-System Logic Design Debugging Flows	

ID	Description	Actions to Take	Relevant Information	Status and Notes
	If you are using the Netlist core insertion flow, have you made sure you have run the Set Up Debug wizard after marking nets "MARK_DEBUG" prior to implementation?	Open the Synthesized Design. After marking all the nets considered necessary for debug you need to invoke the Tools > Set up Debug dialog. This 2 step process is necessary to set up the debug core inserted into the design.	UG908: Programming and Debugging > In-System Logic Design Debugging Flows	
	Have you specified any complex trigger conditions as part of the debug core?	Ensure the selection of "Advanced Trigger" in the Set up Debug Wizard dialog box.	UG908: Programming and Debugging > In-System Logic Design Debugging Flows	
	Do you need to add the VIO Debug capability? Do you know how?	Instantiate the VIO IP core from the IP Catalog->Debug section into your design.	UG908: Programming and Debugging > Debugging Logic Designs in Hardware	
	Do you need to add the JTAG-To-AXI Master Debug capability? Do you know how?	Instantiate the JTAG to AXI Master IP core from the IP Catalog > Debug section into your design.	UG908: Programming and Debugging > Debugging Logic Designs in Hardware	
	Do you need to use the IBERT Debug application?	Instantiate the IBERT IP core from the IP Catalog > Debug section and generate its example design.	UG908: Programming and Debugging > In-System Serial I/O Debugging Flows	
5.2.2	**Debugging the Hardware Design**		UG949: Board and Device Planning UG949: Configuration and Debug	
	Once you have programmed the device with the IBERT design, do you know how to interact with the IBERT application?	Change the Vivado IDE Layout to "Serial I/O Debug" by choosing "Serial I/O Debug" from the Layout menu or choosing "Serial I/O Debug" in the Layout selection in the icon panel of the Vivado IDE.	UG908: Programming and Debugging > In-System Serial I/O Debugging Flows	
	Have you specified the probes file as part of programming the design into the hardware device?	Ensure the probes file is provided in addition to the .bit file as part of programming the device. The probes file contains the debug probe information that is necessary to successfully debug the design.	UG908: Programming and Debugging > Debugging the Design	
	If using the IDE to debug the hardware design, have you changed the Vivado IDE Layout to Debug?	Open the Synthesized Design Change the Vivado IDE Layout to "Debug" by choosing "Debug" from the Layout menu or choosing "Debug" in the Layout selection in the icon panel of the Vivado IDE.	UG908: Programming and Debugging > Debugging Logic Designs in Hardware	