CHAPTER 9

Design Constraints and Optimization

9.1 Overview

Constraints are used to influence the FPGA design implementation tools including the synthesizer, and place-and-route tools. They allow the design team to specify the design performance requirements and guide the tools toward meeting those requirements. The implementation tools prioritize their actions based on the optimization levels of synthesis, specified timing, assignment of pins, and grouping of logic provided to the tools by the design team. The four primary types of constraints include *synthesis*, *I/O*, *timing* and *area/location constraints*.

Synthesis constraints influence the details of how the synthesis of HDL code to RTL occurs. There are a range of synthesis constraints and their context, format and use typically vary between different tools.

I/O constraints (also commonly referred to as pin assignment), are used to assign a signal to a specific I/O (pin) or I/O bank. I/O constraints may also be used to specify the user-configurable I/O characteristics for individual I/Os and I/O banks.

Timing constraints are used to specify the timing characteristics of the design. Timing constraints may affect all internal timing interconnections, delays through logic and LUTs and between flip-flops or registers. Timing constraints can be either global or path-specific.

Area constraints are used to map specific circuitry to a range of resources within the FPGA. Location constraints specify the location either relative to another design element or to a specific fixed resource within the FPGA.

9.2 Design Constraint Management

One of the most important constraint implementation issues is the wide range of potential configuration overlap and interference. ***Effective design constraint implementation requires a solid knowledge and understanding of both the system requirements and the current design implementation approach***. Even with solid knowledge of the design, there are a broad range of design constraint combinations that can be applied to the design. Complex inter-relationships can and do occur between the different constraint types. This inter-relationship

may cause a change in one requirement group to require changes in other design constraints as well, even when the changes may be relatively minor. This complex interaction leads to some challenges in implementing and managing design constraints.

It can be beneficial to develop a design constraint plan in the early stages of a project. An organized plan can help keep the design from becoming over constrained. The design constraint plan may be as simple as an outline with bulleted entries. The constraint plan should be viewed as an informal document with an open format that supports efficient updates as the project matures.

Working to achieve timing closure is a challenging constraint task. The process of achieving timing closure can be improved by following an organized design optimization flow. The second part of this chapter presents a generalized design optimization flow and addresses important topics within each process stage. The selected design optimization flow and other text should be incorporated into the design constraint plan.

9.2.1 Avoiding Design Over-Constraint

Effective design constraint requires design analysis and restraint to develop and maintain the correct constraint balance. Over-constraining a design will cause the tools to work harder to resolve conflicting or unreasonable requirements with limited resources. Design over-constraint can occur in several different ways. Some of the most common include simply assigning too many constraints, constraining noncritical portions of the design, and setting constraints beyond the required level of performance. An example of design over-constraint may occur when path-specific timing constraints have been set to a minimum path delay value far exceeding the required circuit performance. The principle "if a little is good then more must be better." is seldom an appropriate philosophy when constraining an FPGA design.

Over-constraining a design can result in a significant increase in the time required to place, route and analyze a design. The result is a longer design implementation time. Since the design implementation phase potentially occurs many times during a design cycle this can have a significant impact on design efficiency. A more serious design over-constraint consequence occurs when the place-and-route process can no longer successfully implement the design within the specified FPGA architecture. This may force an upgrade to a larger or faster speed-grade FPGA component if the over-constraint conditions are not adjusted.

To avoid design over-constraint a few simple guidelines should be followed. Start by constraining only the highest performance circuits and then add additional constraints as required in an iterative approach. Additionally try to leave significant margin within area constraints and avoid constraining lower performance circuits unnecessarily. A more detailed design optimization flow will be presented later in this chapter.

9.2.2 Synthesis Constraints

The types, syntax and context of synthesis constraints generally vary between tools. Table 9.1 lists some of the synthesis constraints the Xilinx Synthesis Tool (XST).

Table 9.1 XST synthesis constraints

BOX_TYPE	LOC	REGISTER_POWERUP
BUFFER_TYPE	LUT_MAP	RESOURCE_SHARING
BUFG (CPLD)	MAP	RESYNTHESIZE
BUFGCE	MAX_FANOUT	RLOC
CLK_FEEDBACK	MOVE_FIRST_STAGE	ROM_EXTRACT
CLOCK_BUFFER	MOVE_LAST_STAGE	ROM_STYLE
CLOCK_SIGNAL	MULT_STYLE	SHIFT_EXTRACT
DECODER_EXTRACT	MUX_EXTRACT	SHREG_EXTRACT
ENUM_ENCODING	MUX_STYLE	SLEW
FSM_ENCODING	OPT_LEVEL	SLICE_PACKING
FSM_EXTRACT	OPT_MODE	SLICE_UTILIZATION_RATIO
FULL_CASE	PARALLEL_CASE	TIG
INCREMENTAL_ SYNTHESIS	PERIOD	TRANSLATE_OFF
IOB	PRIORITY_EXTRACT	TRANSLATE_ON
IOSTANDARD	RAM_EXTRACT	USELOWSKEWLINES
KEEP	RAM_STYLE	XOR_COLLAPSE
KEEP_HIERARCHY	REGISTER_BALANCING	SLICE_UTILIZATION_RATIO_ MAXMARGIN
EQUIVALENT_REGISTER_ REMOVAL	REGISTER_DUPLICATION	

Synthesis constraints are used to direct the synthesis tool to perform specific operations. As an example, consider the synthesis constraint CLOCK_BUFFER. This constraint is used to specify the type of clock buffer used on the clock port. Two important synthesis constraints that can be used to optimize a design implementation are REGISTER_BALANCING and INCREMENTAL_SYNTHESIS.

Register balancing is used to optimize performance, and incremental synthesis is used to reduce synthesis runtime. Register balancing is used to meet design timing requirements by moving the placement of Boolean logic functionality across register boundaries. Register balancing can increase circuit clock frequency. This improved performance is gained by adjusting the relative path delays. There are two categories of register balancing and they are referred to as *forward* and *backward balancing*. Forward register balancing seeks to move a set of registers located at a LUT's input to a single register at the LUT's output. Backward register balancing is based on the opposite principle. The synthesis tool works to move a register located at a LUT's output to a set of flip-flops at the LUT's input. At the end of the process, the total number of registers in the design may be increased or decreased.

The primary objective of incremental synthesis is to reduce the total time it takes to compile the design. This is performed by synthesizing only the portion of the design that has changed. Synthesis tools may have different switches or constraints within the synthesis

phase to support this approach. Two other factors that can significantly influence the synthesis phase include preservation of the implemented design hierarchy, and the proper use of design constraints.

9.2.3 Pin Constraints

The first question that comes to mind when considering pin assignment is, "Why not let the FPGA tools assign pins?" This is a common question for designers to ask, since the FPGA tools are trusted to place and route the design. However, there are several factors that influence software-controlled resource location assignment. One of the primary FPGA placement directives is to spread functionality out to avoid routing congestion. With no clear guidance to the contrary, the tools will typically work to spread functionality out across the available resources. As an example, FPGA tools can have difficulty identifying the pins that make up a signal bus and can also have difficulty identifying the control signals associated with the bus. Without knowledge that the signals form a group, the tools do not seek to co-locate the signals even though they may benefit from closer placement. While it may be possible to increase the global constraints of the design so that the bus signals and related control signals will be located as a group, the design team then runs the risk of over-constraining the design. This can significantly increase the place-and-route time for the FPGA software.

Ultimately, the design team knows more about the desired data flow through the design than the tools. The design team should be in a better position to guide and influence the design implementation through informed pin assignments. A design team using a rapid design development flow may need to begin I/O assignments very early in the design cycle. The process of I/O assignment is more involved than simply assigning signals to available package pins. The following paragraphs will present some of the considerations that affect the pin assignment decisions.

Assigning board-level signals to FPGA I/O can have a large impact on system performance. In an ideal world, the critical FPGA functionality would have already been captured, compiled and simulated multiple times before the pin assignment step, allowing the design team to determine an optimized pin assignment. However, in a typical rapid system development, device pins are assigned early in the design cycle. The early assignment may be necessary to support early PCB layout. It is possible for the PCB board to have already been routed and in the process of being built before a significant percentage of the FPGA functional design has been captured. This "pin-locking" may be required to meet aggressive design schedules and allow the FPGA development to occur in parallel with the board build effort. This has the effect of maximizing schedule progress, while also increasing risk.

It is important to note that pin assignment is not critical for all designs, or all the pins in a design. Designs with significant I/O margins or slow operational speeds may not require careful pin assignment. However, pin assignment may become a critical factor if the design margin is limited by any of the following FPGA design factors:

- I/O pin availability
- FPGA fabric-level logic resources

- On-chip routing resources
- Required logic speed versus maximum FPGA speed
- Required logic speed versus layers of logic required to implement the design

Pin assignment can also become critical at the board level when signals require special routing considerations such as short signal trace length, matched line length, or controlled impedance. These requirements might be a result of signal loading or speed requirements or EMI requirements.

Most designs fall into a crossover group where pin assignment is not quite critical but also not an insignificant factor in design performance. Almost any design can benefit from a well-implemented pin assignment. It is possible to affect and improve design performance through considered pin assignment. The design factors that may influence pin assignment include:

- The size of the device
- The device package required
- The speed grade of the device
- The maximum speed that the FPGA can run
- The amount of time required to run place-and-route routines
- The number of layers in the PCB
- The number of vias required to implement signal crossovers in the PCB
- The trace width and spacing of the PCB
- The placement and orientation of components on the PCB
- The difficulty and time required to route the PCB

Pin assignment is often not given the time or attention required to implement an optimized design. A few important pin assignment concepts follow.

The pin assignment process is iterative, and pin assignments are often assigned multiple times during the life of a project as design changes and updates occur.

Effective pin assignment requires detailed system-level design knowledge, including:

- ***Board-level component relationships and interface details***
- ***Targeted FPGA architecture details and proposed FPGA-level design implementation***

Pin assignment can be challenging because the designer must be knowledgeable about many aspects of the design. Pin assignment is affected by factors at both the board level and at the device fabric level. Assignments should be made based on a strong systems-oriented understanding of the data flow of the design at all levels. Effective pin assignment requires a detailed knowledge of the signal interfaces into and out of the FPGA at the board level, as well as an understanding of the proposed functional groups and interfaces within the FPGA. Assignments may also be affected by the details of the FPGA family's architecture and I/O structure and the I/O bank configuration set up by the design team.

Since FPGA components come in discrete sizes, FPGA designs may have "extra" I/O pins, which are not required to bring in or out system-critical signals. Rather than simply leaving these pins unused, every effort should be made to utilize each of these pins wisely.

I/O Pins that are "spare" after the required signals have been assigned should be evaluated for potential use as test points, auxiliary I/O or user-defined grounds. Consider the functionality of the board from a system viewpoint. What functionality might be added in the future? What signals will be required to implement future functions? Could board-level errors be fixed internal to the FPGA if the correct signals were accessible? Could additional status or control functionality be provided by routing specific signals into the FPGA? What are these additional signals?

Another critical use for unused pins is provision for access into internal nodes within the FPGA for testing and debugging. Routing a number of test points out to headers or a connector for easy hook-up to test equipment can greatly simplify the verification and debug phase of the design cycle. It can also be valuable to have a few pins routed out to pads. These pads enable easy connection to white wires that may be required to address future issues. Routing out signals for supporting design-for-test (DFT) functionality to support transition to an ASIC in the future should also be considered.

Consideration should be given to incorporating zero-ohm jumpers in-series with debug and expansion traces relatively close to the FPGA package. Placement of pull-up and pull-down resistor footprints, and power and ground connections close to the zero-ohm jumper pads may also be implemented to increase future design options. These additional pads support access to otherwise inaccessible I/O pins allowing simplified addition of white wires to implement design updates if FPGA interface changes are required. These options support simplified debug and potential future design expansion while maximizing future design flexibility. While these options can be very useful in prototype and development environments, they are less appropriate for volume production boards.

Design Clock Considerations

The implementation of clocking signals, routing, pin assignment and clock management can be particularly complex for FPGA design. We will discuss some design factors related to clock implementation in this section. For example, it is possible that bringing a clock in on a specific dedicated clock pin may limit the use or functionality of other dedicated clock pins or use of internal global resources. Similarly, clock feedback inputs to an FPGA component may be limited to a few specific clock input pins. It may be possible to assign a general-purpose signal to a clock feedback input pin blocking access to this FPGA feature unintentionally. A mistake in clock-related pin assignment can severely limit the functionality of a design implementation. ***It is critical that clock assignments be verified and double-checked against all available clock-related documentation***.

Effective clock implementation for high-performance FPGA-based systems benefit from the development of a well-defined clock implementation plan. FPGA designs generally require high input clock quality and careful clock management and implementation internal

to the FPGA. Factors that may degrade clock quality include clock jitter, clock skew and, duty cycle distortion.

Clock jitter effects can significantly degrade the performance of implemented systems. The effects of clock jitter include reduced timing budget margin and performance. Clock skew describes a difference between related signal and clock arrival times. The effects of signal and clock skew include hold time failures, data errors and reduced I/O timing margin. Clock duty distortion can result in reduced pulse widths, data errors and unreliable circuit performance. ***The effects of clock jitter, skew and duty cycle distortion can impact all levels of FPGA circuitry performance and should be carefully managed and controlled***.

The following paragraphs present some FPGA clock design guidelines.

(1) Separate FPGA clocks into priority groups. Use constraints to more clearly characterize clocks for the design tools. Constraints can be used to specify clock rates, phase relationships and duty cycles. Constraints can also be used to associate high-priority clocks with the circuitry they drive.

Clock Priority Groups

- High frequency with high fan-out
- Medium or low frequency with high fan-out
- High frequency with low fan-out
- Medium or low frequency with low fan-out

(2) Assign the highest priority clocks first. The two most significant FPGA clocking challenges are high speed and high fan-out. Clocks with these characteristics should be assigned to higher performance global resources. The number of high-performance buffers and routing resources are limited so they should be carefully managed.

(3) Assign clock block management resources. Clock blocks, such as Xilinx's Digital Clock Managers (DCMs) can implement advanced clock circuit functions including frequency division and multiplication, phase shifting, feedback-based adjustment and synchronous clock generation. Clock blocks are limited resources within FPGA components. The design team should monitor and control how these resources are assigned.

(4) Manage lower priority clocks. While lower priority clocks can be implemented on full-FPGA global resources if they are available they can also be routed through the standard FPGA routing fabric. It may be possible to break global clock routes into multiple smaller high-performance clock routes.

Examples include breaking a global clock route with the potential to supply a clock to the entire FPGA into smaller circuits capable of routing a clock to half or a quarter of the FPGA. Routing a clock via a subsection global route may require the clock to be input to specific I/O pins. Once again, this stresses the importance of careful pin assignment.

9.2.4 Timing Constraints

Timing constraints may be used to influence and guide the placement of design elements and signal routes between placed elements in order to meet design performance requirements. The two general types of timing constraints are *global* and *path-specific*. Global timing constraints cover all paths within the logic design. Path-specific constraints cover specific paths. This section provides some guidelines on timing constraint of an FPGA design.

(1) Identify and constrain system clocks. The timing constraint process should start with the specification of the global timing constraints for all identified system clocks.

(2) Identify and create signal path groups. The two primary types of path groups are global and specific. A global group typically includes a group of paths between registers, input paths, and output paths. Ideally these paths should be within the same clock domain. Specific paths are mostly static or combinatorial paths, paths between clock domains, or multicycle paths. Multicycle paths are defined as paths between logic elements that have a timing requirement that is a multiple of the clock period for the logic elements. For example, if a series of logic functions require more than a singe clock cycle to complete, the data will be correct at the circuit output (the input to the next synchronous design element block) after the pipeline has been filled.

(3) Assign global constraints. The general rule of thumb when assigning constraints is to use global constraints for primary coverage of a majority of the design paths. Apply global period constraints to the design before the HDL synthesis phase. With access to timing constraints, synthesis tools may attempt to optimize the synthesized design to meet the specified timing requirements.

A common design optimization approach is to intentionally over-constrain the design period during the synthesis process. This approach will potentially reduce the amount of time required to meet timing objectives.

Within the design cycle, there is a trade-off between the synthesis phase and the implementation phase of the design flow. Increasing the length of the synthesis phase to reduce the length of the implementation is generally a good choice, since the implementation phase is executed far more often than the synthesis phase during a typical design development.

A goal of 1.5 or 2 times faster than the desired design period is a good rule of thumb during the synthesis phase. ***If the choice is made to over-constrain the synthesis design tools, make sure to prevent the higher-value constraints from being passed forward to the implementation tools. This can generally be accomplished via a tool switch option or by removing access to the synthesis constraint file.***

(4) Assign detailed group and individual path constraints. Use path-specific constraints for paths within the design that justify exceptions to the general constraints already assigned. ***Do not over constrain the paths; more is not better within the design implementation phase***. The more detailed design path constraints are:

KEY POINT

- Multicycle
- False path
- Critical path (for example, From:To)

To explore the finer points of adding time constraints to an FPGA design, two examples are given. The first example involves using timing constraints to specify timing between the system clock and data inputs.

The timing constraint shown in the first example specifies the clock and data signal relationships and timing to ensure that internal FPGA register setup and hold time requirements are not violated. The OFFSET_IN_BEFORE constraint is used to define how long the data signal should be valid before the system clock's rising edge arrives at the FPGA clock pin. The VALID constraint is used to specify both the amount of time the data signal is valid and the amount of time the data signal is valid after the rising edge of the system clock. The timing relationships of these two constraints provide the implementation tools the information required to optimally implement the design. Figure 9.1 illustrates the timing and relationships of the clock and data signals.

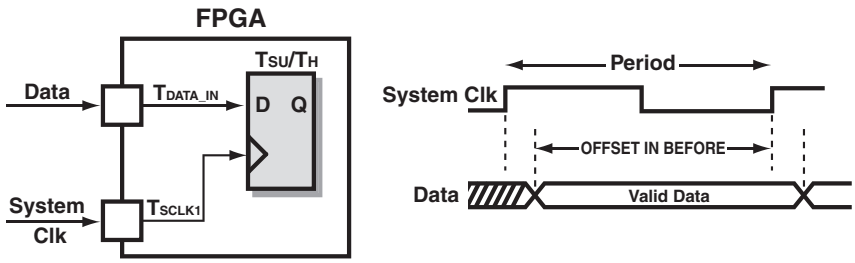

Figure 9.1 Input constraint example

The second example involves the routing of a signal from a register internal to an FPGA to an external component using a system synchronous timing approach. Understanding the FPGA to external device timing requirements is the first step in the constraint process. The external component interface I/O standard, the routing delay to the external component and the loading of the FPGA I/O pin must be determined. Knowing the detailed timing values supports the assignment of a timing constraint specifying the maximum time the data signal has to propagate from the output of the internal FPGA register to the FPGA output pin. The internal delay of the FPGA includes the clock path delay, register clock to output

time, and the data path delay from the register to the output pin. Based on these constraints, the implementation tools can determine a path route which will meet the specified timing requirements. Figure 9.2 illustrates the timing relationship for this constraint use.

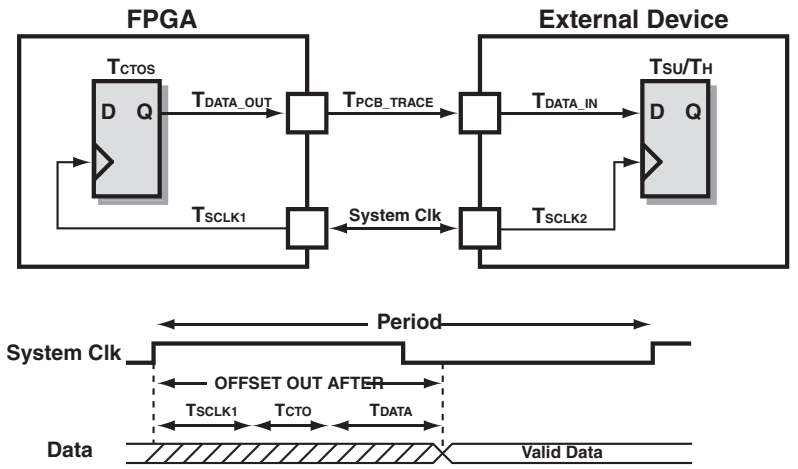

Figure 9.2 Output constraint example

9.2.5 Area Constraints and Floorplanning

Area constrains guide and control where the place-and-route tools may locate FPGA design elements. Area constraints may also define a potential placement region for design elements. A benefit of area constraining is the potential to reduce place-and-route tool implementation time. If the block element is area constrained, the place-and-route tool does not have to search for a location to place a block element. The process of laying out multiple design element blocks onto the target FPGA architecture is commonly referred to as *floorplanning*. Figure 9.3 illustrates the concept of FPGA floorplanning.

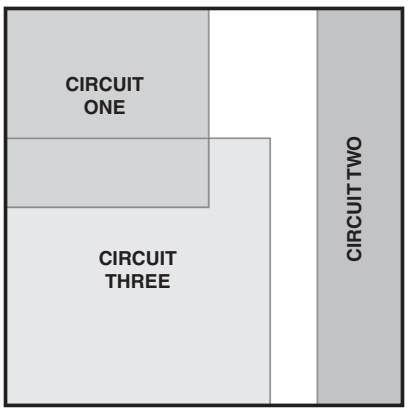

Figure 9.3 Example FPGA floorplan

Floorplanning also supports relationally placed macros (RPMs). Floorplanning is made easier if the design hierarchy is maintained. However, floorplanning may unintentionally cause the implemented design performance to be degraded. This is a consequence of the inability of the implementation tools to override placement constraints. Floorplanning can cause some design layout options to not be available to the design tools, and implemented performance can suffer as a consequence. In certain designs, it may be appropriate to implement the floorplanning effort early in the design optimization process. Taking this step requires a strong design and target architecture knowledge and sufficient available design margin.

Floorplanning may be used to place specific design elements, such as block memories, within the FPGA. ***The placement of design elements should be based on knowledge of which design blocks the elements will interface with and where those design elements (including hard IP functions) will or should be implemented***. Other design situations that may benefit from area constraint or location placement include interleaved logic from two or more design blocks and distributed memory implementations.

The primary objective of hierarchical block floorplanning is to guide the flow of data through the FPGA. Floorplanning will be heavily influenced by the location and distribution of clock resources and fixed functionality within the FPGA. As discussed in the hierarchical design section of Chapter 7, it is desirable to register the inputs and outputs of each major design block to be floorplanned. This provides the best timing margin possible and increases potential successful layout alternatives since the block-to-block interfaces will only require a routing path with no logic elements.

Area constraints are most effective when the design has been intelligently sectioned into functional blocks. Data path-oriented design blocks generally benefit from floorplanning. Place-and-route tools can typically place and locate state machines and other non-structured logic efficiently. The following list presents some considerations associated with area constraining and floorplanning an FPGA design.

Area Constraining and Floorplanning Considerations

- Depending on the tool, area constraints may not be recommended to overlap; refer to tool documentation for guidance
- Floorplanning is effective on data-path logic and hierarchical designs
- Floorplanning should be done with an awareness of the target FPGA architecture
- Develop a detailed understanding of intended design functionality
- Effective floorplanning may be an iterative process
- Avoid design over-constraint

9.2.6 Constraint Example

A functional implementation example will help demonstrate the relationships between the different design constraint categories. Consider a design team with a project requirement to implement a PCI bus interface that is PCI-compatible but not fully PCI-specification

compliant. The team will not be able to use a pre-verified IP core, but will need to develop a custom implementation.

The PCI specification defines the maximum signal trace length from the card-edge connector to any interface circuitry. This minimizes the bus loading at the system level, and limits the board-level signal propagation delays. Thus, the available signal assignments to I/Os at the FPGA package are limited by the placement and orientation of the FPGA component on the PCI-daughter card layout.

Once the FPGA device placement and orientation is defined in relation to the PCI card-edge connector, the design team may verify that the required PCB signal trace length requirements can be met. The PCI data bus and control signals need to be assigned within a select group of I/O pins on the FPGA. Defining the select group of pins on the FPGA package to assign the PCI interface signals to can be relatively involved. The selection process is complicated by the interrelationship between the available FPGA I/O banks, the available I/O pins and the relative relationships of the FPGA package's I/O pins to the location of the die-level I/O pads.

The design team knows the number of required I/O pins and the protocol standards the PCI interface requires (3.3V versus 5.0V, etc.). The design team will select one or more I/O banks with sufficient available I/O pins to support the required number of signals (with some project-defined margin). Each of these selected I/O banks will be configured to implement the required protocol standard. As discussed previously, only certain I/O standards can co-exist within an individual I/O bank. If other standards are required within the design that are not compatible with the PCI protocol, I/O banks must be set aside to support the other protocol standards.

With the I/O banks identified, the design team can assign the signals to the appropriate pins. After the I/O pins have been assigned, the design team can configure any other I/O pin-related characteristics the design requires. FPGA I/O configurable characteristics include faster signal slew rate, impedance matching, and weak pull-up or pull-down functionality.

Next, the design team may begin implementing the area and timing constraints that the FPGA design tools will use to guide the resource location assignments and signal routing for the critical functionality of the PCI interface. The area constraints will define the desired relationship between the group of pins selected for the PCI interface at the board level, and the implemented PCI circuitry within the FPGA. Since the performance of the PCI interface circuitry is critical, the area assignments should group the timing-critical parts of the design relative to both the selected I/O pin group and the location of any PCI functionality within the FPGA.

The area constraints must strike a balance between keeping the circuitry tightly clustered and providing enough margin to allow the placement and routing routines to route other functionality through the specified area. This allows the overall functional and timing requirements of the FPGA to be met. Similarly, timing constraints should be tight enough to guide the layout tools to achieve the required performance without driving the tools to seek a performance level beyond what the design requires.

If changes are made to any of these constraint groups, they must be evaluated to ensure that they won't cause changes in one or more of the other groups as well. Potential reasons for constraint changes include design functionality changes or an FPGA reorientation on the board. In this example, if the FPGA package needed to be rotated 180 degrees, each of the constraint groups will need to be re-evaluated and re-implemented. Each design is unique, and the relationships between the design constraint groups will be just as unique.

9.2.7 Constraints Checklist

The following list presents FPGA design constraint guidelines.

✔	*Design Optimization and Constraints Checklist*
❑	Develop and follow a design constraint plan
❑	Add constraints incrementally
❑	Constrain from general to specific
❑	Add only enough constraints to consistently meet functional and timing requirements
❑	Achieving higher performance requires a balanced mix of design constraints
❑	Designers need to be familiar with timing report context and analysis

9.3 Design Optimization

As applications implemented within FPGAs increase in speed, complexity and resource utilization, meeting performance requirements requires additional efforts. The following sections present a generalized FPGA design optimization flow. The process is based on the principle that the minimum amount of effort should be expended to get a design to meet its timing requirements. The individual design blocks should be captured and initially verified by simulation. The individual design blocks can then either be initially independently implemented or integrated, and then implemented as a system.

When it has been determined that the design does not meet timing, then the incremental changes discussed in this optimization flow should be made to hopefully ultimately enable the design to meet the required timing performance. Once the design consistently meets its timing performance requirements, no additional design constraints or design changes are required. Additional effort may be expended and performance may continue to improve; however, if the requirement is to achieve a certain level of performance, any effort expended to achieve performance beyond that level will not be a productive use of resources.

Optimization of an FPGA design can be a challenging design phase. There are many different approaches requiring different levels of effort. The order in which optimization efforts occur is important since some optimization activities can affect the results of previously applied efforts. ***Following an established optimization procedure can help make the optimization phase more efficient.***

Some optimization approaches can affect the results of previously applied optimization activities, so an established optimization methodology can make the design optimization

effort more efficient. The sequence of activities presented here is not absolute. It is intended as a guide; changes may be made based on prior experience, familiarity with the design, and personal preference.

This chapter presents an overview of a generalized incremental optimization design flow, starting with the lowest possible level of effort and working up through more involved optimization approaches. Additional design adjustments and modifications are iteratively applied to the design in an ordered sequence until the design consistently meets the desired performance requirements.

The objective of applying successive design optimization techniques is to avoid spending any more time or effort optimizing the design than necessary, while also reducing the risk of over-constraining the design.

9.3.1 FPGA Design Optimization Process

Timing closure can become difficult for large and complex FPGA designs. The process of obtaining timing closure will typically include multiple incremental HDL modification iterations, constraint refinement, design re-implementation (synthesize, place and route), and repeated timing analysis. ***A well-defined and organized design implementation flow is important to efficient design optimization***. Figure 9.4 presents a suggested design implementation optimization flow. The design optimization flow presented in this section is based on the flow presented in Rhett Whatcott's Xilinx TechXclusives article, *Timing Closure – 6.1i*.

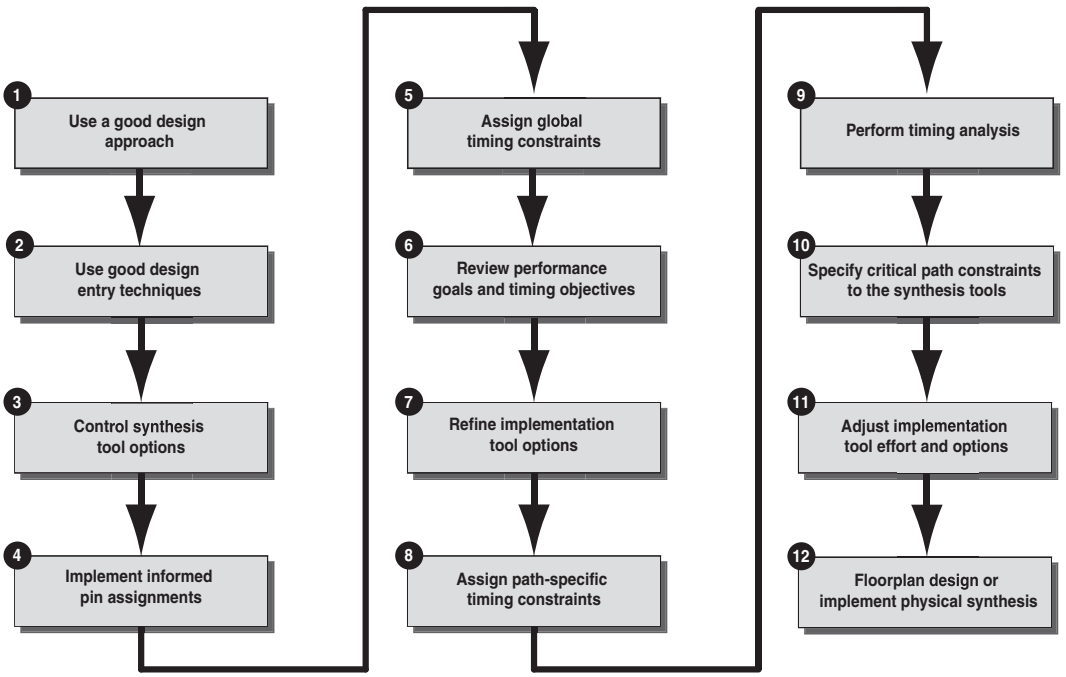

Figure 9.4 FPGA design optimization flow

(1) Use a good design approach. As always, synchronous design techniques are strongly recommended. Implement strong, organized hierarchical design structures. Keep design modules and blocks to a manageable size. Partition design blocks intelligently as discussed in the implementation chapter.

(2) Use good design entry techniques. Use an HDL design entry method following defined coding standards and styles. Adopt and use a common coding standard. Comment code to clarify intent. When appropriate, use cores and design instantiation rather than relying on inference. Implement code that will take advantage of the specific resources available within the targeted FPGA component hardware architecture (fabric and routing resources).

(3) Control synthesis tool options. Research and understand the available synthesis tool directives, switches, constraints and operational modes. Follow the synthesis tool vendor guidelines provided to obtain the best design results. In order to maintain design portability between different synthesis tools, enter synthesis constraints through the synthesis tool constraint editor.

(4) Implement informed pin assignments. In rapid system prototyping, pin assignment will occur early in the design cycle. Research and understand the details of the FPGA fabric and architecture. Make pin assignments and assign constraints that take into account the design signal and control flow, board component relationships and FPGA fabric architecture. It may be possible that the pin assignment occurs even earlier in the process before the design has been synthesized.

(5) Assign global timing constraints. The objective of this stage is to specify the global timing for each design clock. Path-specific constraints can be added to either the synthesis or implementation tools. Adding path-specific constraints to the synthesis design tools causes additional architectural optimization to occur. Adding path-specific constraints to the design forces the tools to increase the priority on the specified paths during the place-and-route cycle. A combination of these two approaches can leverage a design toward meeting timing requirements.

(6) Review performance goals and timing objectives. Review the design report files. Static timing analysis is used to evaluate how close the implemented design is to meeting or exceeding the required timing. Once the design has been implemented into logical design elements, the delay through the logic elements of the design will be defined. The logic delays will remain fixed through the final design implementation.

At this point in the design cycle, it is possible to evaluate the design's implementation against the 60/40 rule. This rule specifies that 60% or less of the timing budget should be consumed by the logic portion of a signal connection while the routing portion of the connection should take 40% or more of the budget. If this design guideline is met, the tools have a better chance to achieve the required timing performance. Having 40% or more of the available routing time (the clock period) available for signal routing is a general guideline, the appropriate ratio for each design may vary based on the target architecture. The 60/40 rule is intended to provide a measure of "goodness" at this stage of the design cycle.

(7) Refine implementation tool options. With a design close to meeting timing requirements, some minor adjustments to the implementation tools (mapping and place and route) may allow the design to pass without having to add advanced timing constraints to the design.

Once global (and possibly high-level path-specific) constraints have been added to the design, the design team may make adjustments to the level of effort of the implementation tools. For example, the level of place-and-route effort can be adjusted from standard to a higher level. If multiple effort levels are available, it is advised that the effort level be increased one level at a time, rather than from lowest to highest all at once. Again, the objective is to apply only as much effort as required. Higher effort levels will naturally extend the time required to complete the place-and-route implementation effort, thus leading to a longer implementation cycle time.

Changing the implementation tool effort level has the advantage of avoiding the need to make changes to the design code. If the timing requirements cannot be met by increasing the level of implementation effort, other approaches must be applied.

(8) Assign path-specific timing constraints. If the design does not meet timing requirements with the application of global timing constraints and adjustments to the synthesis and implementation tools, it may be necessary to apply more detailed timing constraints. Applying constraints is typically an iterative process. Well-considered additional design constraints may help the implementation tools prioritize the place-and-route design efforts. Potential modifications to the code may also be required. The utilization of cores or FPGA architecture-specific coding structures may be required to improve performance.

(9) Perform timing analysis. Take time to review the design timing analysis reports. Ensure that all paths are fully optimized. If paths are identified that are not meeting timing, make changes to adjust the way these paths are implemented.

(10) Specify critical path constraints to the synthesis tools. Use constraints to identify critical paths to the synthesis tools to guide more targeted design implementation. This effort will likely be iterative. For maximum design portability, implement constraints via the synthesis tool constraint editor.

(11) Adjust implementation tool effort and options. Make adjustments to advanced implementation tool options. These adjustments will likely increase the implementation cycle processing time. This results in a trade-off between time and results. These design optimization efforts occur later in the optimization flow, since they tend to increase the length of the design implementation phase and make each design update cycle significantly longer, which can be a significant penalty in designs that must be implemented many times.

Changes can also be made to the design packing and placement tool efforts. Setting tool switches to force placement and routing of critical signals early in the optimization cycle can result in significant design performance improvements. Again, higher levels of effort will increase design implementation times.

Another option involves adjusting the number of place-and-route cycles run. Increasing the number of place-and-route cycles causes the design to be implemented with different design implementation priorities, increasing the odds of achieving a successful design placement and routing combination. One advanced implementation approach involves running many placements, and then only routing the "best" placements.

(12) Floorplan design or implement physical synthesis. The design place-and-route phase may be guided by specifying floorplanning constraints that direct design elements to specific locations on the FPGA fabric. This is saved as one of the last design optimization approaches, since floorplanning can unintentionally make certain timing paths worse. This factor is compounded by the fact that the tools cannot override the placement constraints.

Make sure to allow enough margin within each placement block or range to allow the implementation tools sufficient margin to implement the design efficiently in parallel with other design functionality which may need to be co-located within that specific area of the FPGA fabric. Make sure to review tool restrictions. For example, some tools do not encourage layout block overlap and may actually restrict placement within overlapping areas.

Another advanced option involves using a physical synthesis tool, which has a level of awareness of the target FPGA architecture, structure and available resources. Physical synthesis implements a design that co-locates related logic functionality in the physical design for reduced routing overhead. Physical synthesis is related to design floorplanning since it influences and guides the placement of logic to assist meeting timing objectives. Physical synthesis tools can provide a 10–20% improvement in system timing. Most physical synthesis tools are not provided as part of the basic manufacturer tool suite. Physical synthesis tools are more efficient when operating on synchronous designs.

9.4 Summary

The four types of constraints include *synthesis*, *pin*, *area* and *timing*. Synthesis constraints are used to instruct the synthesis tool on how to map the HDL code to RTL occurs. Pin constraints are used to specify the assignment of I/O. Area constraints are used to instruct where the place-and-route tool can locate a specified design block partition. Timing constraints are used to specify path delays. Timing constraints can be global or path-specific.

Floorplanning is the process of guiding the placement of multiple design partitions onto the FPGA fabric. Design constraints, floorplanning and tool options can influence the design optimization. For example, floorplanning can be used to optimize the FPGA fabric area. Constraining a design for a minimum area results in fewer routing resources used with smaller interconnect distances. This means faster signal paths and implementation times. The result is a design that takes up less FPGA fabric area and has increased performance, and faster implementation times. Floorplanning is a powerful speed and area optimization technique if done properly. However, there are no set rules for properly floorplanning a design.

Design optimization is an incremental process that applies increasing engineering effort and tool computational time to leverage the design to meet timing. Most of the effect on the ability of the design to meet timing is derived from the original design implementation. Synchronous design, design modularization, a formal design hierarchy with registered boundaries, and good HDL coding can all positively influence the ability of the design implementation tools to achieve the desired timing performance.