

Power Methodology Guide

UG786 (v14.5) April 10, 2013



Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2009-2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
4/10/2013	14.5	Minor updates.

About This Guide

This power estimation and analysis methodology guide covers in a single document all power effects you may encounter while designing your FPGA logic and integrating it onto your system. It presents the electrical and physical factors, internal or external to the FPGA, which affect power. It then provides flows and techniques you may use at any stage in the design cycle to monitor and minimize device power.

The document is organized with the following chapters:

[Chapter 1, Power in FPGAs: Introduction](#) presents the different components necessary to understand power consumption in an FPGA and implications of the larger system onto that FPGA. It provides definitions for the different terms, explains the physics, and describes the contributing factors.

[Chapter 2, Software Power Analysis](#) presents the different approaches to calculate or estimate power in an FPGA. It presents the benefits, required inputs, assumptions and trade-offs with accuracy and complexity so you get a better understanding of how the presented results are calculated.

[Chapter 3, Power Estimation Methodology](#) presents, for each step in your design cycle, which tool and methodology to employ to assess the FPGA power and understand the implications on the larger system being designed.

[Chapter 4, Tips and Techniques for Power Reduction](#) provides practical guidelines you can use to minimize power for all aspects of the system. It covers considerations on the system in which the FPGA will be implemented, HDL coding techniques, software settings, and power optimization algorithms.

[Chapter 5, Summary](#) summarizes the information presented in the guide.

[Appendix A, Additional Resources](#) lists documents that are related to the information presented in this guide.

Table of Contents

Revision History	2
About This Guide	3
 Chapter 1: Power in FPGAs: Introduction	
FPGA Power Aspects and System Dependencies	7
FPGA Power and the Overall System Design Process	12
Xilinx Power Estimation and Analysis Tools	13
 Chapter 2: Software Power Analysis	
Power Calculations	17
Thermal Calculations	18
Power Models Accuracy	19
Activity Estimation	19
Utilization Estimation	22
 Chapter 3: Power Estimation Methodology	
Before Implementation	23
Monitoring During Implementation	26
Power Closure Stage	31
Power and Temperature Measurements	36
 Chapter 4: Tips and Techniques for Power Reduction	
System Level	39
Device Level	40
Design Level	40
Software Settings and Algorithms Level	44
Comparing Devices or Architectures Effectively	47
 Chapter 5: Summary	
 Appendix A: Additional Resources	
Xilinx Resources	51
Hardware Documentation	51
ISE Documentation	51
PlanAhead Documentation	52
Power White Papers	53

Power in FPGAs: Introduction

This chapter provides an overview of the different aspects and steps to consider when implementing an FPGA on a board. It puts the FPGA development in the greater context of the system being designed and provides a high level description for what to expect at each stage of the design flow. Flow details, tips, and techniques are provided in [Chapter 3, Power Estimation Methodology](#), and [Chapter 4, Tips and Techniques for Power Reduction](#).

FPGA Power Aspects and System Dependencies

Modern FPGAs are very capable chips integrating a large array of customizable logic plus memory, DSP, processor, and many hardened blocks to process data and communicate with other chips. As with many complex chips on a printed circuit board (PCB), FPGA power depends on multiple factors. On one hand, the FPGA and the user design creates system power supply and heat dissipation requirements, while on the other hand the system's physical and electrical factors can affect supply and cooling of the FPGA. Before we dive into more details on these aspects, the following sections provide the terminology and concepts we will use through the remainder of the document.

Power Supply Paths

Multiple power supplies are required to power an FPGA. The separate sources provide the required power for the different FPGA resources. This allows different resources to work at different voltage levels for increased performance or signal strength while preserving a high immunity to noise and parasitic effects.

For logic resources typically available in Xilinx FPGAs, [Table 1-1](#) presents the voltage source that typically powers them. This table is provided only as a guideline because these details may vary across Xilinx device families.

Table 1-1: FPGA Resources and the Power Supply that Typically Powers Them

Power Supply	Resources Powered
V_{CCINT} & $V_{CCBRAM}^{(3)}$	<ul style="list-style-type: none"> All CLB resources All routing resources Entire clock tree, including all clock buffers Block RAM/FIFO⁽¹⁾ DSP slices⁽¹⁾ All input buffers Logic elements in the IOB (ILOGIC/OLOGIC)⁽¹⁾ ISERDES/OSERDES⁽¹⁾ PowerPC™ processor⁽¹⁾ Tri-Mode Ethernet MAC⁽¹⁾ Clock Managers (DCM, PLL, etc.) (minor) PCIE and PCS portion of MGTs
V_{CCAUX} & $V_{CCAUX_IO}^{(3)}$	<ul style="list-style-type: none"> Clock Managers (MMCM, PLL, DCM, etc.)⁽¹⁾ IDELAY/IDELAYCTRL⁽¹⁾ All output buffers Differential Input buffers V_{REF}-based, single-ended I/O standards, e.g., HSTL18_I Phaser
V_{CCO}	<ul style="list-style-type: none"> All output buffers Some input buffers Digitally Controlled Impedance (DCI) circuits, also referred to as On-Chip Termination (OCT)⁽²⁾
$V_{CCPint}^{(3)}$	<ul style="list-style-type: none"> Zynq processor, memory interface, I/O interfaces and AXI interfaces
$V_{CCPAUX}^{(3)}$	<ul style="list-style-type: none"> Zynq memory and I/O interfaces
$V_{CCODDR}^{(3)}$	<ul style="list-style-type: none"> Zynq memory interface
$V_{CCO_MIO}^{(3)}$	<ul style="list-style-type: none"> Zynq I/O interfaces
$V_{CCPLL}^{(3)}$	<ul style="list-style-type: none"> Zynq PLL blocks
$V_{CCADC}^{(3)}$	<ul style="list-style-type: none"> XADC block
MGT*	<ul style="list-style-type: none"> PMA circuits of transceivers

Notes:

- These resources are available only in certain device families. Refer to the appropriate data sheets and user guides for more information.
- V_{CCO} in bank 0 (V_{CCO_0} or V_{CCO_CONFIG}) powers all I/Os in bank 0 as well as the configuration circuitry. See the applicable [Configuration User Guide](#).
- Xilinx 7 series FPGAs only.

Types of Power

Three components make up the total required power for each supply source.

- **Device static (leakage) power** – Represents the power required for the device to operate and be available for programming. A large portion of it is due to leakage in the transistors used to hold the device configuration.
- **Design static power** – Represents the additional continuous power drawn when the device is configured and there is no activity. This includes static current from I/O terminations, clock managers, and other circuits which need power when used and regardless of design activity.
- **Design dynamic power** – Represents the additional power resulting from the design activity. This power varies over time with the your design activity. It also depends on voltage level and logic and routing resources used.

Power Consumption Paths

The total power supplied to the device flows in then out of the FPGA through multiple paths:

- **Thermal power** – Represents the power consumed internally within the FPGA. It enables the different elements in the device and toggles the device logic. These actions generate heat, which contributes to raising the device junction temperature. This heat is then transferred to the environment. As with any other chip on the PCB, FPGA designers must provide heat dissipation paths to ensure the junction temperature remains within the device operating range.
- **Off-chip power** – Represents the current that flows from the supply source through the FPGA power pins then out of the I/Os and dissipated in external board components. The currents supplied by the FPGA are generally consumed in off-chip components such as I/O terminations, LEDs, or the I/O buffers of other chips and therefore do not contribute to raising the device junction temperature.

Power Modes

An FPGA goes through several power phases from power up to power down with varying power requirements:

- **Power-On** – Transient spike current which occurs when power is first applied to the FPGA. This current varies for each voltage supply and depends on the FPGA construction as well as the ability of the power supply source to ramp up to the nominal voltage. This current also depend on the device's operating conditions, such as temperature, sequencing between the different supplies, etc. With modern FPGA architectures and following the Power-On sequencing guidelines these spike currents are no longer a concern.
- **Configuration** – Power required during the configuration of the device. Unless your application is extremely low power, configuration power is always lower then active power, so this transient stage does not affect power supply requirements.
- **Standby** – Power supplied when the device is configured with your design and there is no activity applied externally or generated internally. This represents the minimum continuous power the supplies have to provide while the design operates.
- **Active** – Power required while the device is running your application. This power includes the standby power (all static power) plus any power generated from the design activity (design dynamic power). This power is instantaneous and varies at each clock cycle depending on the input data pattern and the design internal activity.

- **Suspend** – Power required when all or part of the device is powered down. This varies with device families; however, the device configuration is always preserved. Wake up logic can trigger the return to an active state. This mode ensures a good compromise between power savings and fast return to full functionality.
- **Hibernate** – Power required when one or more power supply sources are turned off. This functionality also varies with each FPGA architecture but generally allows the most power savings for applications which do not need to operate all the time. Wake up time to return to the active power state is longer since it typically requires the device to be reprogrammed before returning to normal operation. Hibernate mode also clears all data within the chip including BRAM and returns the device to the same state as it did during power-up.

Environmental Power Contributing Factors

- **Supply strategies**
 - **Regulator technology** – Different regulator technologies exist to balance input to output voltage difference, response time, maximum currents, and output voltage accuracy constraints.
 - **Decoupling network performance** – In addition to supplying the FPGA during brief and high power demand periods, an efficiently designed decoupling circuit will reduce current surge requests from the regulator and improve overall regulator consumption.
 - **FPGA selection** – Different FPGA families require different power supply counts and voltage levels. Selecting a device which supports a lower core voltage or lower voltage I/O interfaces reduces power.
- **Cooling strategies**
 - **System environment** – Shape and dimension of the system enclosure along with the ambient air temperature are the primary aspect which you can consider for transferring the generated heat to the environment.
 - **Heat sink** – Dimension, shape and mounting of the heat sink and the eventual associated forced airflow system determine the amount of heat that can be extracted from the FPGA.
 - **Package selection** – In addition to cost and signal integrity, the package dimension, material, and connection to the board influence how the generated heat can be transferred to the environment from both the top and bottom level. The larger the contact surface area between the heat sink and board the lower the thermal resistance.
 - **Component placement** – Component placement relative to the system enclosure and other board material, assembly, and components affects how the heat is transferred to the environment. For instance, an obstacle may reduce the airflow near the FPGA. Other heat generating components in close proximity to the FGPA may heat up the air flowing above the device and reduce the heat sink efficiency or transfer heat into the FPGA via the board material.

Device Power Contributing Factors

- **Manufacturing parameters**
 - **Silicon technology** – Xilinx design teams perform detailed simulation and analysis to select manufacturing process parameters carefully. Finding the appropriate balance between the desired functionality, cost, performance and reliability call for the use of different transistors sizes, strengths, voltages, and arrangements. This optimizes how much energy transistors require to be polarized or to switch. Xilinx provides behavioral and transistor level models to simulate the device I/O electrical characteristics within your

system. These models (IBIS, AMI, HSPICE, ELDO) are available on the Device Models page of the Xilinx [Downloads](#) web page.

- **Package technology** – Xilinx carefully selects the package technology and physical and electrical layout parameters as they influence the flow of currents through to the device core and I/Os. In addition, parameters such as shape and material define how the heat generated within the device can be transferred out onto the environment from both the top and bottom level of the package. To evaluate the electrical properties of the package in your particular system, Xilinx provides IBIS formatted per-pin RLCG models. Compact thermal models are available to simulate the thermal behavior of the selected package. These Package Thermal models are available on the Device Models page of the Xilinx [Downloads](#) web page.
- Architectural parameters
Xilinx teams define, among other parameters, the type, functionality, amount, layout, and connectivity between the different FPGA resources. This too is a difficult balancing act between contradicting factors. The most power efficient architectural choices can impact the device cost and performance or make the software implementation algorithms more complex. These choices greatly influence the device static and dynamic power.

Recent device enhancements:

- Adjusted the slice or basic logic structure to fit more functionality. Packing more logic into a smaller area typically saves logic and routing dynamic power (6 input over 4 input LUTs).
- Hardened functionality commonly used in designs. Adding dedicated block reduces system power since less FPGA programmable logic or external component are needed (Clock generator, PCIe, Memory controller).
- Optimized routing structures to minimize number of hops to get from source to destinations.
- Added gating buffers or ports to allow you to disable high fanout nets when not in use (clock, reset, clock enable, etc.).
- Added power down modes applied to the entire device or to individual logic blocks. This allows you to turn off unused portions of the design and enable them only when needed.
- Added support for low voltage I/O interfaces. Voltage is a key aspect driving both the static and dynamic power of I/Os.

Design Power Contributing Factors

- Device selection
 - **Appropriate device family** – Different vendors and device families offer different logic and I/O capabilities. Selecting the device with block size, configuration, and resources that best fit your application will optimize architectural element usage and therefore reduce static and dynamic power.
 - **Appropriate family member** – Device size influences mostly the device static power. Selecting too large or too small a device will result in suboptimal mapping of your RTL description, which will be less efficient in terms of dynamic power.
- **RTL description**
The design description influences how logic equations get mapped onto the available resources. A good understanding of the architecture elements, ports, configuration options, and modes helps maximize the use of embedded resources.

- Tool constraints

By default implementation tools aim to achieve your performance objective, then minimize device utilization. Providing a realistic and complete set of constraints for performance and utilization, for both core and I/O logic, will drive the tool most efficiently. This minimizes the dynamic power.

- Implementation tool options

Implementation tools have different algorithms which optimize for power reduction. Most are optional or can be swept using multiple strategies. Understanding your constraints and what they do will help you decide which ones are needed.

FPGA Power and the Overall System Design Process

From project conception to completion there are many different aspects to consider that influence power. Omitting for a moment all your other constraints (functionality, performance, cost, and time to market), we can sort power related tasks into two separate classes.

- **Physical domain** – Enclosure, board shape, power delivery system, thermal power dissipation system.
- **Functional domain** – Area, performance, I/O interfaces signal integrity.

The next chapters will demonstrate interdependencies between these two classes however they are different in that the former involves hardware decisions while the latter mostly involves the FPGA logic design. Typically hardware selection and sizing occurs very early in the design flow to give time for prototype boards to be built. The FPGA functionality's effect on power consumption can be estimated early on, then refined as more and more of the design logic is completed. [Figure 1-1](#) illustrates a typical system design process and highlights power related decision points.

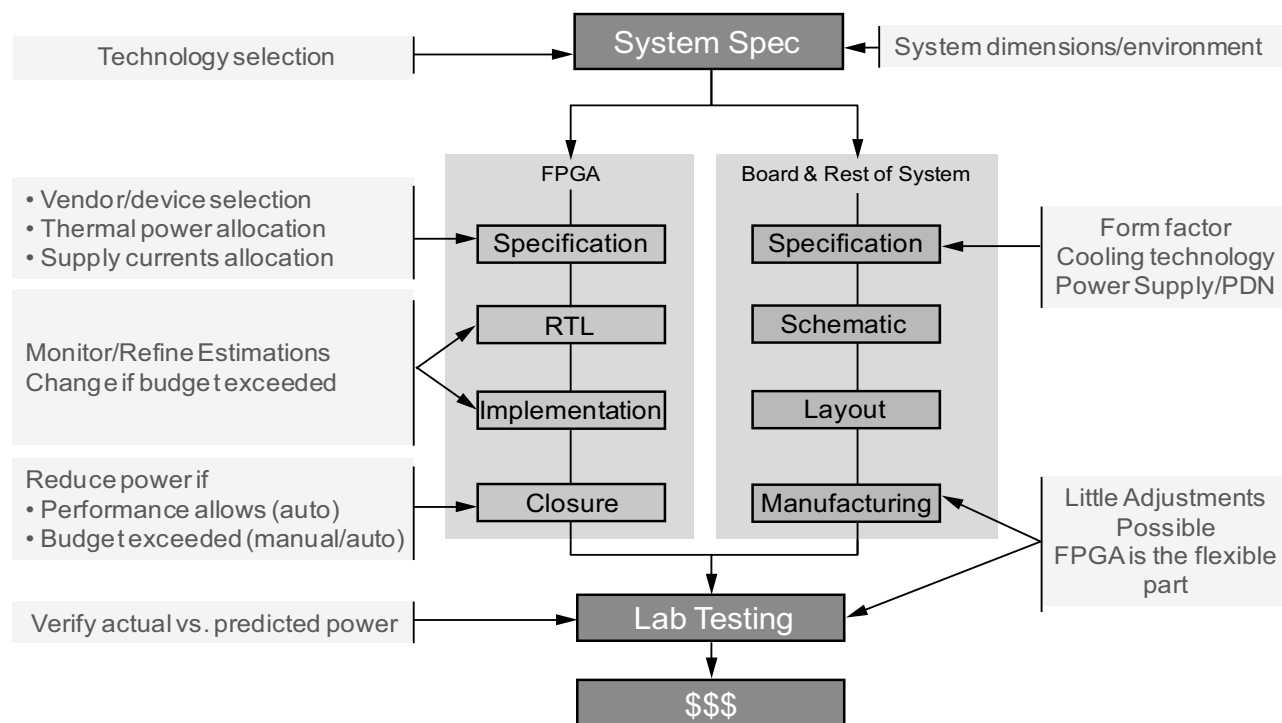


Figure 1-1: Managing Power Aspects in a System Design Process

Figure 1-1, page 12 shows that at the time you select your device and associated cooling parts, the FPGA logic is not yet available. Therefore a careful methodology to estimate the FPGA logic power requirements is needed. Methodologies are discussed in [Chapter 3, Power Estimation Methodology](#).

The next three chapters provide definitions and highlight the main contributor for each aspect influencing power.

Xilinx Power Estimation and Analysis Tools

Xilinx provides an extensive suite of tools and documentation to help you evaluate the thermal and supply requirements of your FPGA throughout the design cycle. [Figure 1-2](#) highlights the tools available at each stage of the FPGA design cycle. Some of the tools are standalone while others are integrated into the implementation software, to align with the environment and information available to you at each stage of the design process. All the tools have communication channels so you can exchange information and go back and forth to be most efficient with your analysis.

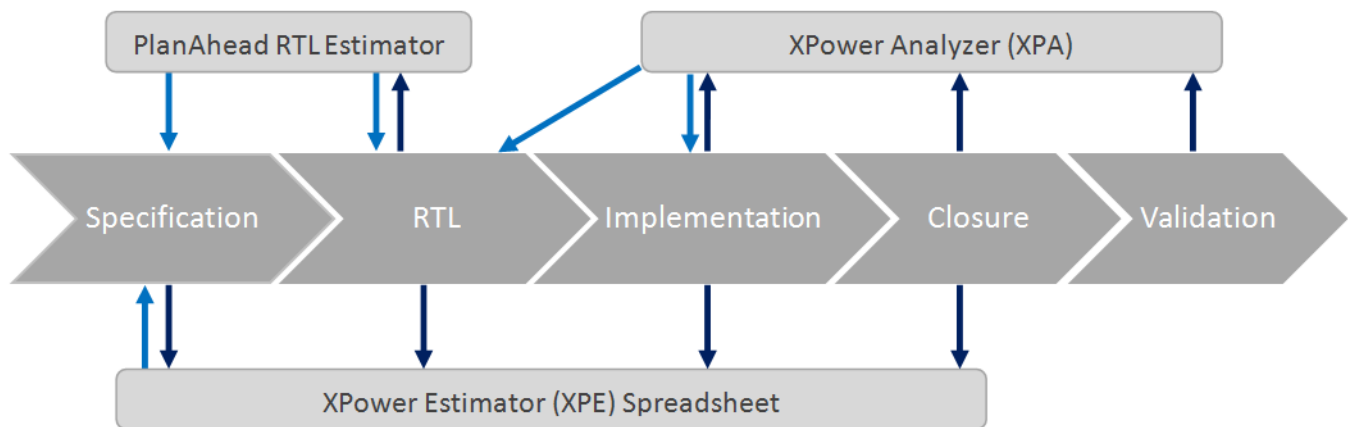


Figure 1-2: Xilinx Power Estimation and Analysis Tools in the FPGA Design Process

XPower Estimator (XPE)

The XPower Estimator (XPE) spreadsheet is a power estimation tool typically used in the pre-design and pre-implementation phases of a project. XPE assists with architecture evaluation and device selection and helps in selecting the appropriate power supply and thermal management components which may be required for your application. The XPE interface ([Figure 1-3](#)) lets you specify design resource usage, activity rates, I/O loading, and many other factors which XPE then combines with the device models to calculate the estimated power distribution.

XPE is also commonly used later in the design cycle during implementation and power closure to, for example, evaluate power implications of engineering change orders (ECO). For large designs implemented by multiple teams, the project leader can use XPE to import utilization and activity of each team's module, then monitor the total power and reallocate the power budget to ensure constraints are met.

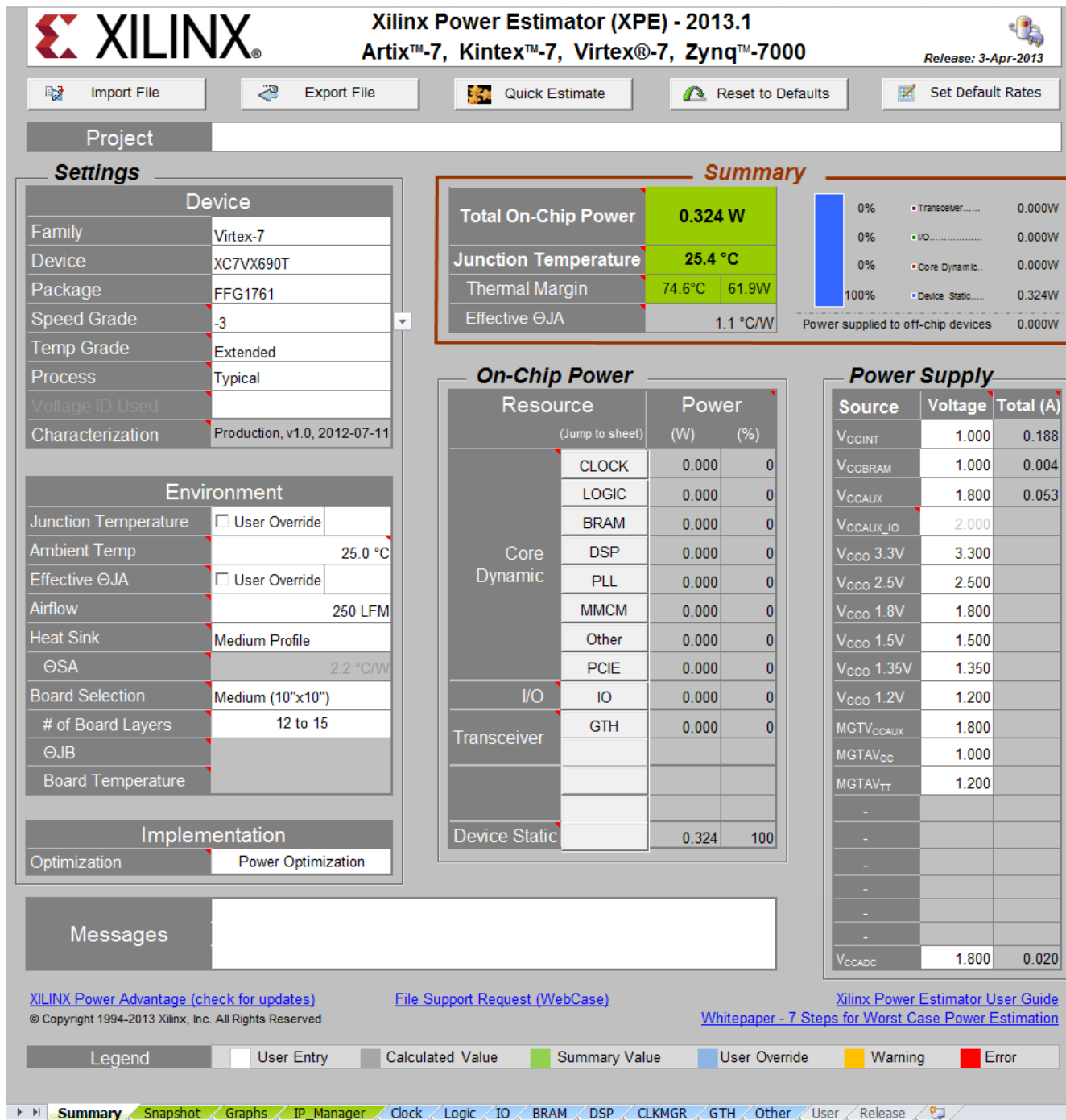


Figure 1-3: XPower Estimator Spreadsheet

XPower Analyzer (XPA)

The XPower Analyzer (XPA) tool performs power estimation post implementation. It is the most accurate tool since it can read from the implemented design database the exact logic and routing resources used. Figure 1-4 presents the summary power report and the different views you can navigate our design: by clock domain, by type of resource and by design hierarchy. XPA also allows you to adjust environment settings and design activity so you can evaluate how to reduce your design supply and thermal power consumption.

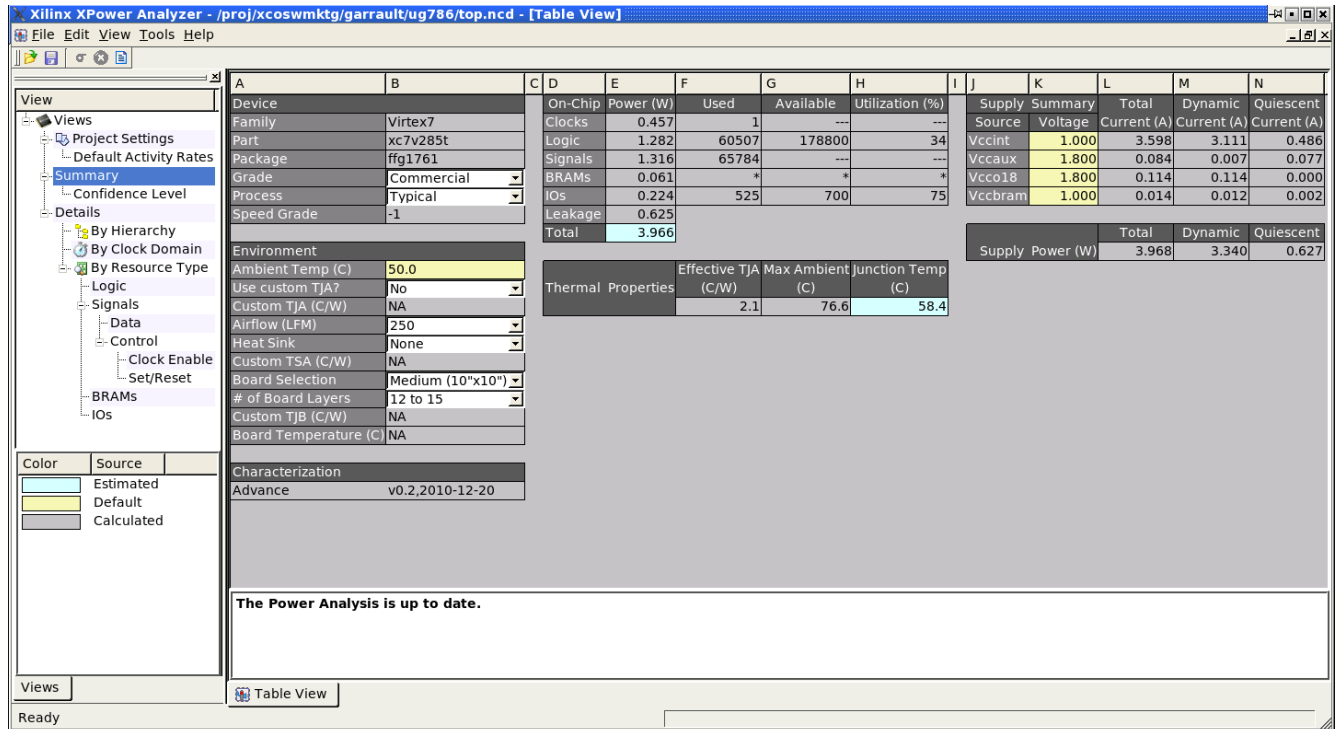


Figure 1-4: XPower Analyzer

PlanAhead RTL Power Estimator

The PlanAhead software performs power estimation to provide an early view of your design power distribution at the RTL level. You can specify the device operating environment, the I/O properties, and the default activity rates for the design using constraints or by interacting with the GUI. PlanAhead then reads your HDL code to estimate the design resources needed, and reports the estimated power from a statistical analysis of the activity of each resource.

Figure 1-5 shows a report and resource and hierarchy views which you can navigate to analyze the power distribution. With its access to more detailed information about the design intent the RTL power estimator should be more accurate than the XPower Estimator spreadsheet and less accurate than post Place and Route analysis done with XPower Analyzer.

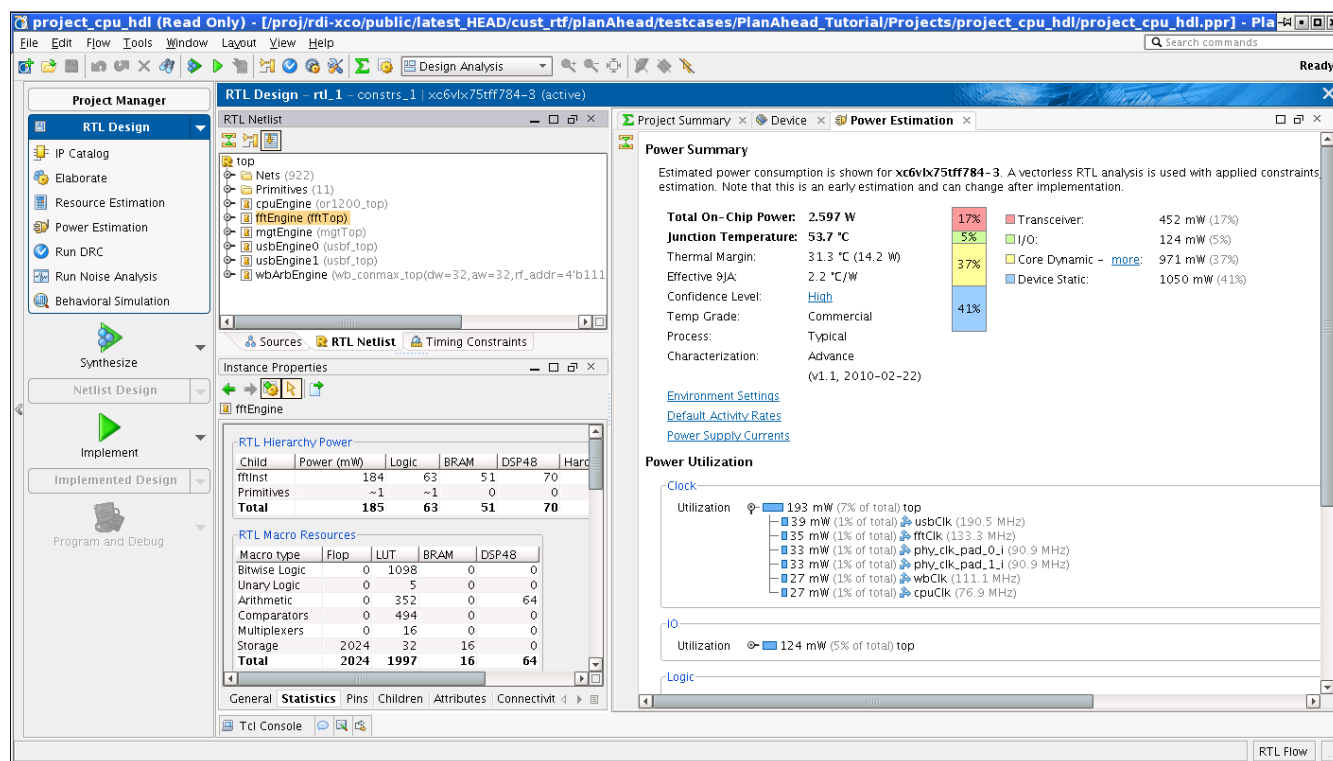


Figure 1-5: PlanAhead RTL Power Estimator

Software Power Analysis

In the previous chapter, we saw many of the different factors which influence the total power requirements in an FPGA. This chapter describes how these factors are determined and used in the software tools' calculations. Since information about the device resource usage, configuration and activity is unclear early on in the design cycle, the software will use default values. As the design implementation progresses, more of this information can be provided to the software, which will increase the accuracy of the power estimations. Xilinx uses an extensive suite of user designs plus anonymous data collected by the Webtalk program to determine statistically realistic tool default values. These values are applied when parameters cannot yet be computed automatically or data was not provided by the user.

Power Calculations

The total power for each voltage supply source of the FPGA is:

$$\text{Active Power} = \text{Device Static} + \text{Design Static} + \text{Design Dynamic}$$

Where the components that make up the total power are:

- **Device Static**
Device static power depends mostly on manufacturing, process properties, applied voltage, and the device junction temperature. The junction temperature itself depends on the ambient temperature, voltage level, and total current supplied. But the total current supplied includes the device static as a component, so we have a clear circular dependency. The tools use a series of successive iterations to converge on an approximation of the true static power for the specified operating conditions.
- **Design Static**
Some blocks in an FPGA are disabled by default then enabled, depending on your design requirements (for example, I/O termination, transceivers, block RAM, and clock generators). When these blocks are enabled they consume a set amount of power regardless of user design activity. This power varies with the configuration of the circuit. The software is able to model each circuit's contribution to the overall design static power. The models take into account a number of resource level configuration settings and also scale with voltage and the device's external environment settings. The external environment settings determine how heat dissipates into the environment.
- **Design Dynamic**
Design dynamic power mostly depends on the capacitance and activity of the resources used, and also scales with the applied voltage level. The software device database models the capacitance of each resource according to its configuration and connectivity. Then, if the user doesn't supply the activity, the software algorithms can predict the activity of each node in the netlist before calculating and adding up power for all components.

Note: Device static power plus design static power correspond to the standby power defined in [Power Modes in Chapter 1](#).

Thermal Calculations

The device junction temperature or temperature of the silicon is calculated as:

$$\text{Junction Temperature} = \text{Ambient Temperature} + \text{Thermal Power} * \text{Effective Thermal Resistance to Air}$$

Where the variables in the equation are:

- Junction Temperature (°C)
Temperature of the silicon. When selecting the device you choose a temperature grade. This grade defines a temperature range where Xilinx guarantees the device will operate as specified. If your operating conditions are above the Grade Maximum but remain below the Absolute Maximum temperature then the device operation is no longer guaranteed. Exceeding the Absolute Maximum operating conditions may cause damage to the device.

Working this equation backward you may want to set the junction temperature to the device maximum and determine the maximum power the device can generate for your environment. More precisely, since the tools present both static and dynamic power you can determine the worst case device leakage and maximum dynamic power your application is allowed to generate.
- Ambient Temperature (°C)
Temperature of the air immediately surrounding the device under the expected system operating conditions.
- Thermal Power (W)
Represents the power consumed internally within the FPGA. This generates heat, which contributes to raising the device junction temperature.
- Effective Thermal Resistance to Air - Θ_{JA} (°C/W)
This coefficient defines how power is dissipated from the FPGA silicon to the environment (device junction to ambient air). It includes contributions from all elements, from the silicon chip dimensions to the surrounding air, plus any material in between, such as the package, the PCB, any heatsink, airflow etc. Typically this combines thermal resistance and interdependencies from the two main paths which the generated heat can escape onto the environment:
 - Upward from the die to the air (junction to air or Θ_{JA}),
 - Downward from the die through the board and into the air (junction to board or Θ_{JB}).

Power Models Accuracy

The accuracy of the characterization data embedded in the tools evolves over time to reflect the device availability or manufacturing process maturity. This accuracy designation is presented under the **Characterization** field. Device family characterization data evolve with the following sequence: Advance, Preliminary, and Production.

Advance

Devices with this designation have data models primarily based on simulation results or measurements from early production device lots. This data is typically available within a year of product launch. Although the data with this designation is considered relatively stable and conservative, some under or over-reporting may occur. Advance data accuracy is considered lower than the Preliminary and Production data.

Preliminary

This designation is based on complete early production silicon. Almost all the blocks in the device fabric are characterized. The probability of accurate power reporting is improved compared to Advance data.

Production

This designation is released after enough production silicon of a particular device family member has been characterized to provide full power correlation over numerous production lots. Device models with this characterization data are not expected to evolve further.

Activity Estimation

Two parameters represent the activity of each node in a netlist:

- **Signal transition rate** – Defines the number of times the considered element changed state for the duration of the analysis. This sums up the number of positive and negative edges that occurred. For calculations Xilinx tools express this number as a number of millions of transitions per seconds (Mtr/s).
- **Signal static probability rate** – Defines the percentage of the analysis duration during which the considered element is driven at a high logic level. Also referred to as *Signal percentage high rate*.

Design node activity is a critical aspect for getting accurate power estimation. Depending on the level of completion of your design you may know the value, or the tool may be able to calculate or estimate it as described in the following sections. As a general rule, given the influence of this factor in power calculations, you should specify any activity information that is known. When the activity is not known, however, the best practice is to let the tool estimate activity.

User input

In any design, users typically know the activity of specific nodes since they are imposed by the system specification or the interfaces with which the FPGA communicates. Providing this information to the tool, especially for nodes which drive multiple cells in the FPGA (Set, Reset, Clock Enable, or clock signals) will help guide the power estimation algorithms.

These nodes encompass:

- **Clock activity** – Users typically know the exact frequency of all FPGA clock domains, whether externally provided (input ports), internally generated, or externally supplied to the printed circuit board (output ports).
- **I/O data ports** – With your knowledge of the exact protocols and format of the data flowing in and out of the FPGA, you can usually specify signal transition rate and/or signal percentage high rate in the tool for at least some of the I/Os. For example, some protocols have a DC balanced requirement (Signal percentage high rate = 50%) or you may know how often data is written or read from your memory interface, so you can set the data rate of strobe and data signals.
- **I/O and internal control signals** – With your knowledge of the system and the expected functionality you may be able to predict the activity on control signals such as Set, Reset and Clock Enable. These signals typically can turn on or off large pieces of the design logic, so providing this activity information will increase the power estimation accuracy.

Simulation

In parallel with all stages of the design development you will generally perform simulations to verify that the design behaves as expected. Different verification techniques are available depending on the design development state, the design complexity, or company policy. The following paragraphs highlight the valuable data you can capture, and common pitfalls related to using this data to perform power analysis. An important factor for getting an accurate power estimation is that activity needs to be realistic. It should represent the typical or worst case scenario for data coming into the simulated block. This type of information is not necessarily provided while performing verification or validating functions. Sometimes invalid data is given as input to verify that the system can handle and remain stable even when invalid data or commands are given to it. Using such test cases to perform power analysis may result in inaccurate power estimation since the design logic is not stimulated as it would be under typical system operation.

- **System transaction level** – Very early in the design cycle, you may have created a description of transactions which occur between devices on a PCB or between the different functions of your FPGA application. You can extract from this the expected activity per functional block for certain I/O ports and most of the clock domains. This information will help you fill in the XPower Estimator spreadsheet.
- **FPGA description level** – While defining the RTL for your application you may want to verify the functionality by performing behavioral simulations. This helps you verify the data flow and the validity of calculations to the clock cycle. At this stage the exact FPGA resources used, count, and configuration is not available. You can manually extrapolate resource utilization and extract activity for I/O ports or internal control signals (Set, Reset, Clock Enable). This information can be applied to refine the XPower Estimator spreadsheet information. You can also read your HDL into the PlanAhead RTL Power estimator as described in the “Estimating Power” section of the *PlanAhead User Guide (UG632)*, referenced in [Appendix A, Additional Resources](#). This tool will estimate device utilization and activity to provide quick power estimation. Your simulator should be able to extract node activity and export it in the form of a SAIF file. You can save this file for later use in the design

flow, for example after Place and Route, if you do not plan to run post-implementation simulations.

- **FPGA implementation level** – Simulation may be performed at different stages in the implementation process with different outcomes in terms of the power-related information which can be extracted. This additional information may be used to refine the XPower Estimation spreadsheet. It may also save I/O ports and specific module activity, which can later be reused in XPower Analyzer after the design is completely placed and routed.
 - **Post synthesis** – The netlist is mapped to the actual resources available in the target device.
 - **Post placement** – The netlist components are placed into the actual device resources. With this packing information the final logic resource count and configuration becomes available and you can update the XPower Estimator spreadsheet for your design.
 - **Post routing** – After routing is complete all the details about routing resources used and exact timing information for each path in the design are defined. In addition to verifying the implemented circuit functionality under best and worst case gate and routing delays, the simulator can also report the exact activity of internal nodes and include glitching. Power analysis at this level will provide you the most accurate power estimation before you actually measure power on your prototype board.

Statistical Estimation

When design node activity is not provided either from the user or from simulation results, the vectorless power estimation algorithms are capable of predicting this activity. The vectorless engine assigns initial “seeds” (default signal rates and static probability) to all undefined nodes. Then, starting from the design primary inputs it propagates activity to the output of internal nodes, and repeats this operation until the primary outputs are reached. The algorithm does not understand the timing or functional relationships between the various netlist components, however it understands the design connectivity and resource functionality and configuration. Its heuristics can even approximate the glitching rate for any nodes in the netlist. Glitching occurs when design elements change states multiple times in between active clock edges before settling to a final value. Overall the vectorless propagation engine is not as accurate as a post-Place and Route simulation with a reasonably long duration and realistic stimulus, but it is an excellent compromise between accuracy and compute efficiency.

Utilization Estimation

Resource usage is an important factor in determining the total power used by the targeted FPGA.

- **System specification level** – Before any code has been developed, and excluding for a moment code acquired, reused from a previous design, or generated from the wizard or core generator tools, then you will need to manually estimate usage for the different blocks to implement. You do this estimate by using your experience and understanding of the expected functionality. The accuracy of this estimate depends on the level of information available to you and the time you can devote to entering this information into the XPower Estimator spreadsheet.
- **RTL description level** – Xilinx provides an area estimator which can read your RTL code. You provide all your HDL and the IP cores and black boxes, and the tool will elaborate and resolve all parameters into a netlist. Then the area estimator will do pattern recognition to quickly estimate the resources which will be inferred and how these will be mapped to the targeted architecture. This is not as accurate as working with a synthesized or routed netlist; however it takes away a lot of the guesswork to estimate device resources used while you are developing the application code. This helps you monitor estimated power at an early stage in the development process when changes are less time consuming, more powerful, and less risky than later on.
- **Post-synthesis level** - The synthesis tool will infer logic structures from your description then map this RTL to specific device resources. It will optimize the netlist to meet your performance and area requirements. This increased level of detail for the device resources used can help you monitor and adjust your power estimations.
- **Implementation level** – Post-map and post-Place and Route power estimations become even more accurate since netlist optimizations affecting final logic resource utilization such as trimming/register replication or re-timing are taken into account. Actual routing resource usage and layout also become available to the power estimation tool.

Power Estimation Methodology

This chapter elaborates on the flow presented in [Chapter 1, Power in FPGAs: Introduction](#). For each step in a typical design cycle, the chapter describes a methodology to evaluate your application's power consumption, then continues with a description of tool features which automate or simplify power estimation. Once you generate the power estimation, go to the next chapter for techniques to investigate and eventually modify your system, to minimize the device power consumption. For details of the tools discussed here, refer to these documents in [Appendix A, Additional Resources](#):

- [XPower Estimator User Guide \(UG440\)](#)
- [PlanAhead User Guide \(UG632\)](#) (Estimating Power section in Chapter 5, RTL Design)
- XPower Analyzer documents:
 - Graphical interface: [XPower Analyzer \(XPA\) Help](#)
 - Command line tool (`xpwr`): [Command Line Tools User Guide \(UG628\)](#)

Before Implementation

Expectations

At this stage you have determined that an FPGA is the most effective technology for your application. Now you need to define which vendor, family, and package can best fit your functionality, performance, cost, and power budgets. In terms of power this means that you need to estimate the total device power requirements even before any logic is developed. Understanding the total power requirements will help you define your power delivery and cooling system specifications. Questions that you will typically ask yourself are how many voltage supplies are needed, how much power will each be drawing, and how much of the absorbed energy will generate heat. XPower Estimator is used to answer these questions. It helps you develop in parallel the FPGA logic and the Printed Circuit Board on which the device will be soldered. This exercise will also help you understand the margin you can expect to have and therefore gain confidence that your system will work within budget once implemented.

Figure [Figure 3-1, page 24](#) shows the XPower Estimator interface.

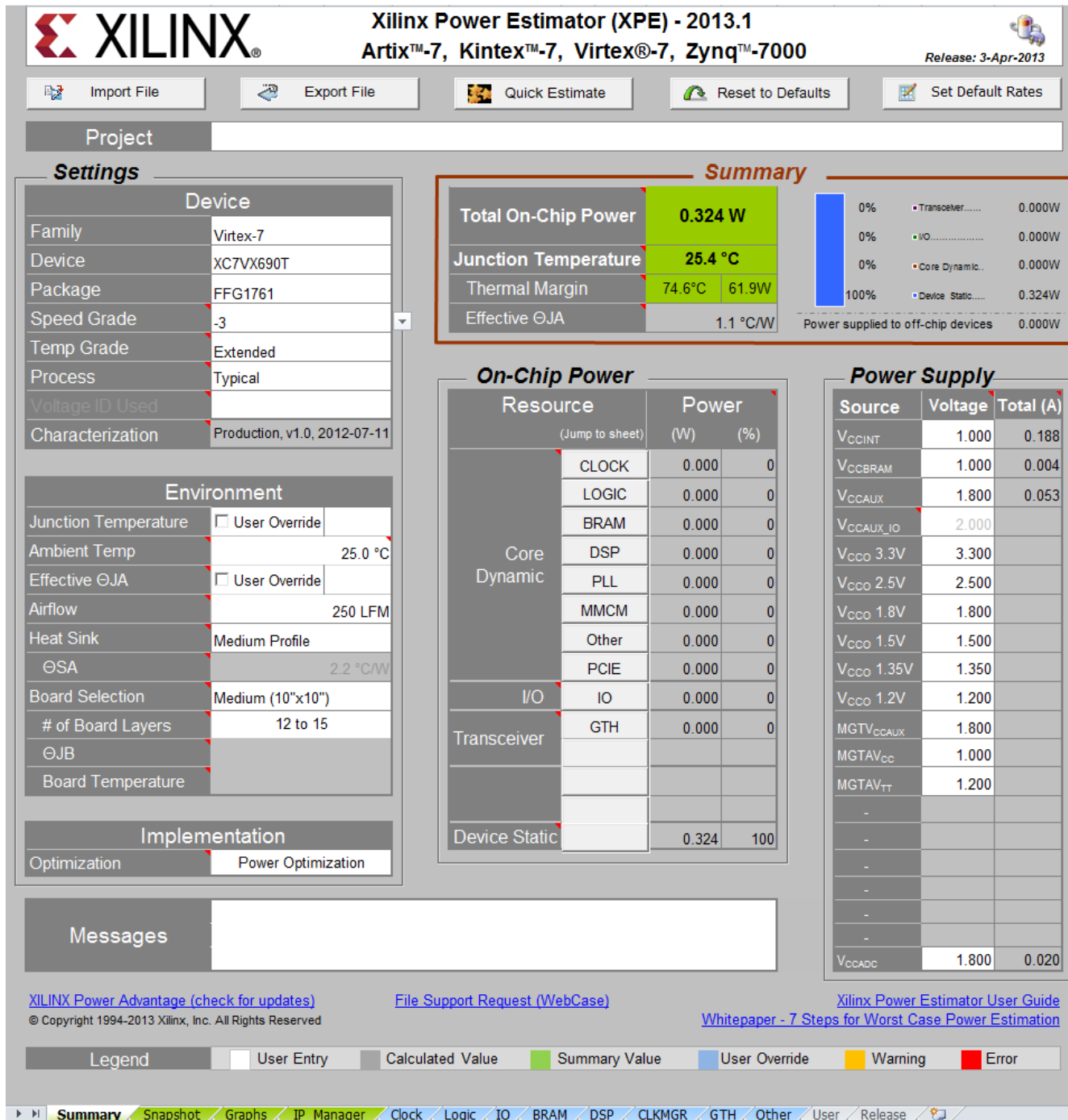


Figure 3-1: XPower Estimator (XPE) Methodology and Presentation of Power Information

Methodology

At this stage, the XPower Estimator spreadsheet will help you research and organize all known environmental and design information. Download the applicable spreadsheet for your device from the Xilinx [Power Solutions](#) webpage.

1. In the XPower Estimator spreadsheet, enter the device settings.

Select the device from the drop-down menus.

Note: The **Process** field on the Summary sheet accounts for typical or worst case power characteristics due to manufacturing variations.

2. Enter the environment settings.

When you open the spreadsheet, the Summary sheet has an **Environment** panel which lets you set all the environmental parameters which affect the device power. Changing these affects the device static power but it also determines how much the device junction temperature will rise for each additional unit of power to be dissipated by the FPGA (Effective ThetaJA).

If your design process allows you to perform thermal simulations on your system then you can download the package thermal model for your specific device from the Device Models page of the Xilinx [Downloads](#) web page. Use this information to calculate the specific junction to air (Θ_{JA}) and junction to board (Θ_{JB}) thermal resistance for your system. If you cannot perform thermal simulations use the value in the drop-down menus which most closely matches your system.

3. Enter the software settings.

Specify what you expect to be the most difficult aspect of your design, then XPE can adjust dynamic power calculations based on the implementation algorithm used and assumed placement and routing results.

4. Enter the voltage supply settings.

If this information is known, in the **Power Supply** table enter the exact voltage level for the different supply sources. Voltage is a large factor contributing to both static and dynamic power.

5. Enter the device resources used and expected activity.

For each design module or functionality block, extract the expected FPGA resource count and the configuration which will be required. Then estimate the average activity of these pieces of logic. Finally enter this information on the various device resource sheets in XPE.

Tool features and tips to estimate device resource usage and design activity:

- If your design includes IP blocks acquired or reused from previous designs which have already been implemented, then import the power information using an XPower Analyzer interoperability file. This saves a lot of guesswork and you can always adjust the imported data to account for the design specifics or differences in FPGA architecture.
- Instead of trying to guess for a large module, try to decompose each block to the finest granularity you can, then add up all resources.
- Enter realistic data and avoid being too conservative. Adding “padding” logic or an activity margin to each module is not necessary since there are other ways to calculate worst case scenarios.
- Use your experience from previous designs.
- The User sheet in XPE is not protected and is very useful to document intermediate calculations and assumptions.

Minimum set of data:

- Enter environment and voltage data that closely match the expected actual environment, since these parameters have a significant influence on the total power estimated.
- Enter as much information as is known on the I/O and transceiver sheets, since these also account for a large part of the total power
- For all the other sheets make your best educated guesses for the resource utilization and average activity columns without adding unnecessary margin, and then leave the unknown columns to their default values.

6. Perform What If? analysis scenarios.

Without having to code anything you can devise different implementation scenarios and review the implications in terms of resource usage and power. You can, for example, evaluate different I/O interface formats and see which standard, I/O configuration, and output termination would best meet your power budget. Using the on-chip termination will help you save on external components and board space. It will also increase the on-chip power, and you need to ensure that this power can be transferred out of the device and into the environment to keep the device junction temperature within the normal operating range.

The project leader can use this type of analysis to allocate the total device resource and power budget among the different team members. Even if these allocations are approximations and will be revised in the future, the analysis provides each development team with more independence and helps the teams develop in parallel.

7. Update the spreadsheet regularly.

As the design implementation progresses, more specific information becomes available and you can update this spreadsheet to refine your power estimations. For example, as soon as some part of the design has been created, then the project leader can import this data into XPE to replace the assumptions made earlier in the design flow. Commonly the project scope may evolve over time, with new features being requested; this can change device utilization and expected activity patterns. Refining the spreadsheet regularly will help you verify that the total thermal power budget has not been exceeded and catch potential issues early on.

Monitoring During Implementation

Expectations

As your design implementation progresses you will want to monitor and verify the power consumption regularly and make sure thermal dissipation remains within budget so that you can detect and act early on if any area gets close to your constraints. The tools and features available vary depending on how complete your logic is, as detailed below.

RTL Description Stage

As you write your HDL code and before running synthesis you can get an estimation of the resources used and dynamic power consumed from the RTL power estimation algorithm in PlanAhead. You do not need a complete project. You can create a project for the available portion of the design.

Figure [Figure 3-2, page 27](#) shows the PlanAhead presentation of power information.

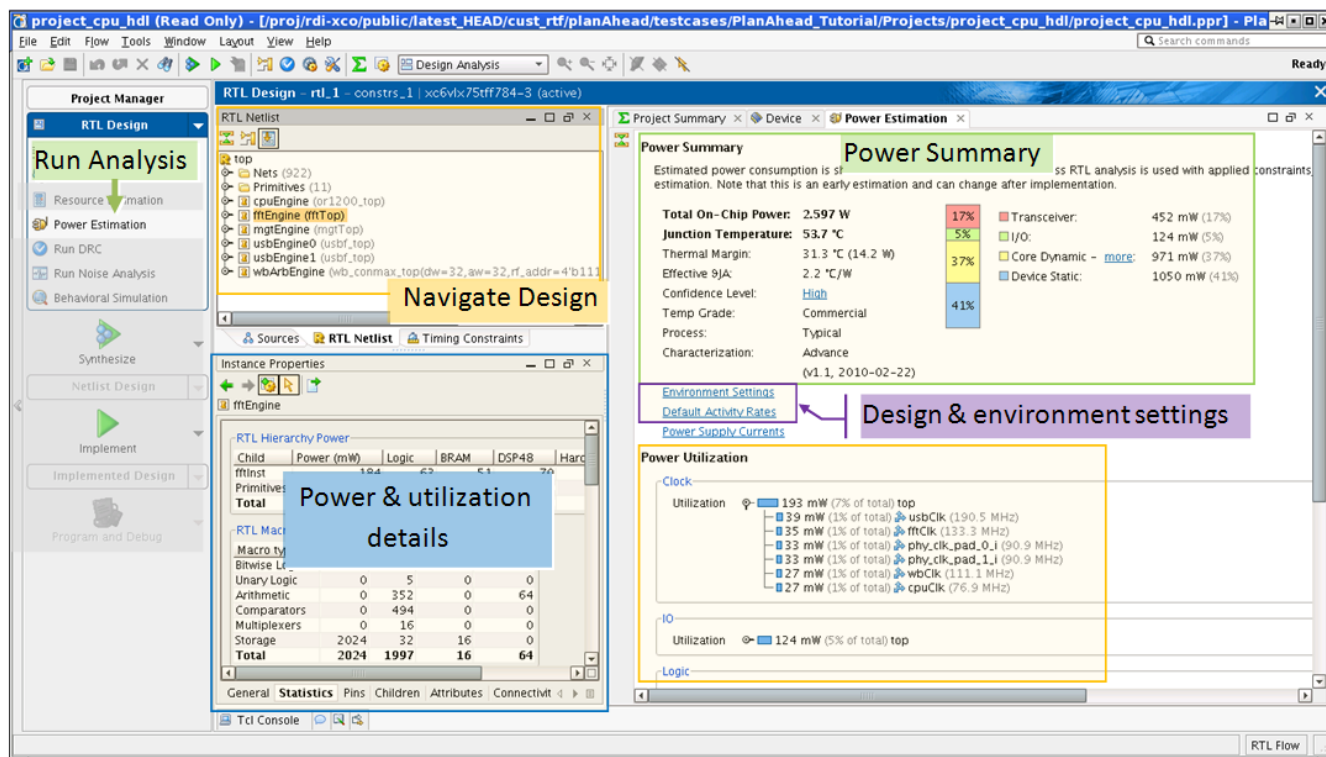


Figure 3-2: PlanAhead Methodology and Presentation of Power Information

Methodology

1. In PlanAhead, open the RTL view.

PlanAhead reads in all the specified HDL files. It resolves constants and parameters, then elaborates the description to generate an RTL netlist and estimate the resources used. This process is very fast.

2. Set your device and environment to match the data entered in the XPower Estimator spreadsheet.

These factors do not influence dynamic power significantly, but will help you do a fair comparison with current assumptions in XPE, especially towards the end of the RTL description when most of the design HDL code is available.

3. Set activity rates.

For known elements such as clocks you can define these nodes using constraints.

For unknown elements the tool has defaults which you can later adjust.

4. Run the power estimation and review the estimated area and associated power.

You can navigate through the elaborated design and review the estimated resources used and associated power for each element in the design hierarchy.

5. Adjust the XPower Estimator or implementation constraints after area and power analysis.

In the XPower Estimator spreadsheet review the resource and power estimated by the RTL power estimation algorithm. When you see differences from what you had originally planned, decide if a correction is necessary. In the RTL power estimation tool, on the other end, after reviewing the power distribution for your particular block you may want to consider changes in your description or adding constraints to guide the mapping of the downstream tools.

These estimates are not final since neither the synthesis mapping and transformations nor the Place and Route optimizations of this logic have been performed. However, comparing this RTL power estimation with the earlier manual estimations you did can help you confirm your estimate or make adjustments, which in both cases will improve your confidence that you will remain within your power budget.

Synthesis Stage

Once the design is synthesized, whether entirely or by module, device resources become fairly well defined, although further optimizations during Place and Route may affect final utilization or activity slightly. The synthesis tool tries to achieve your performance requirements while satisfying your other constraints, such as available resources on the target, power budget, and runtime. This means making mapping decisions, such as using block versus distributed memory or using a different encoding style for state machines. All these decisions affect the configuration and amount of the resources used, which you could only approximate before. You can compare your assumptions in the XPower Estimator spreadsheet with these synthesis results and adjust XPE where appropriate.

Place and Route Stage

Expectations

After Place and Route is done, the design database has all the logic configuration, packing, and routing structures used completely defined. The XPower Analyzer performs and presents you with the most comprehensive report. However, the tool still needs information from you to be most effective. The following Methodology section guides you through generating a power report at this stage and points out the information, unavailable from the project files, which you can provide to improve the accuracy of the power estimation.

Methodology

This section covers power analysis using the XPower Analyzer (XPA) GUI. We assume this is the first time you are setting up a power analysis after Place and Route. You will therefore provide the tool with activity information and refer to existing data files. For subsequent runs, you can choose whether to use the XPA GUI to navigate your design and analyze power or use the command line equivalent (**xpwr**) to bypass the GUI and review the text power report directly.

Figure [Figure 3-3, page 29](#) shows the XPower Analyzer interface.

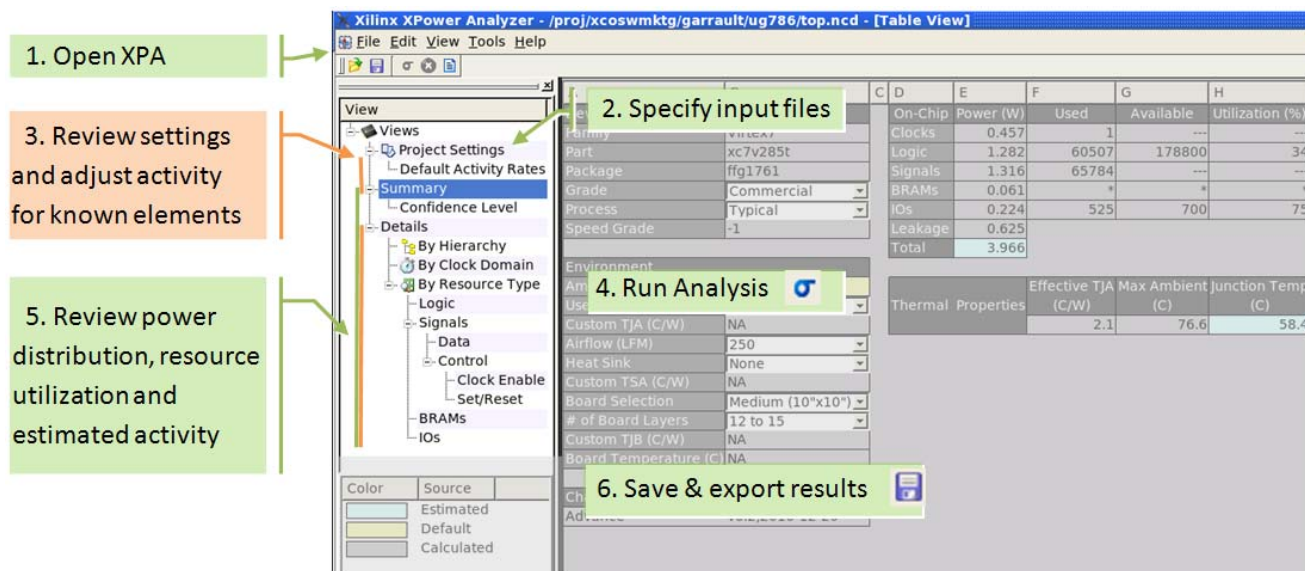


Figure 3-3: XPower Analyzer (XPA) Methodology and Presentation of Power Information

1. Open the XPower Analyzer graphical interface.
XPower Analyzer can be invoked from Project Navigator, from the PlanAhead interface, or by entering the **xpa** command line.
2. Specify the input files to use for the analysis.
 - **Placed and routed design database (NCD file)** – Contains all logic configuration and routing information.
 - **Physical Constraint (PCF file)** – Contains settings for all of the logic and I/Os of your design plus activity for specific nets such as clock networks.
 - **Simulation Results (SAIF or VCD file)** – XPA will match nets in the design database with names in the simulation results netlist. For all nets matched it will apply switching activity and static probability to calculate the design power. Simulation results may have been generated early in the design flow, before synthesis or Place and Route. In this case it is preferable to capture from the simulation results only module I/O ports activity and let the vectorless engine estimate internal node activity. Functional simulations do not capture glitch activity. In addition, XPA may not be able to match all nodes between the design and the simulation netlist because of logic transformations which happen during implementation (replications, gating, retiming, etc.). Nevertheless most primary ports and control signals will be matched and this information provides the tool with realistic activity for the matched nodes. The activity will be propagated by the vectorless engine onto the unmatched design portion and increase the accuracy of the power estimation. When you provide simulation results, make sure to use this type of simulation results:
 - Ensure test vectors and inputs to the simulation represent the typical or expected behavior of the design. Error handling and corner case simulations do not typically stimulate the logic in the way it would be stimulated under normal operation.
 - Post-implementation simulation results are preferred over behavioral simulation results.
 - **Settings (XPA file)** – Contains environment and activity data. This information may have been generated from an earlier run of XPower Analyzer. In this case it is an easy way to reopen XPA with the same conditions and results as on a previous run, so you can continue an analysis where you left off. You can also use this mechanism to save, retrieve,

and compare multiple scenarios. Finally, you can import a settings file created earlier in the flow from the XPower Estimator spreadsheet, and all your environment parameters will be imported into this XPA session, saving the effort of manually reentering them.

3. Review settings and adjust activity for known elements.

Review the different input fields to make sure they accurately represent your expected system.

- **Project settings** – In the Project Settings view, make sure all available input files have been read into XPA. When importing simulation results, review the matching rates to ensure the numbers correspond to your expectations. Unmatched hierarchy separator or format differences can reduce the number of elements for which activity is read from the simulation output result.
- **Environment settings** – Review the different user-editable cells in the Summary view. Make sure the process, voltage and environment data closely match your expected environment. These settings have a significant influence on the total estimated power.
- **Tool defaults** – In the Default Activity Rates review the tool's current defaults, estimate if your application departs significantly from these numbers, and decide if adjustments are required. By default, Xilinx recommends leaving these settings unchanged since they were derived from a suite of representative user designs. These numbers are used as “seeds” for any node for which the user did not provide activity, through GUI edits or an input file. The propagation engine will then adjust each node's activity depending on the activity propagated from the driving cone of logic.
- **Known elements** – This step is very important for calculating the design dynamic power since your specific knowledge of the application behavior can help define the activity of nodes undefined in any of the input files.
 - By Clock Domain view – Make sure all clocks are specified. Although not recommended, sometimes users overconstrain their designs to make the implementation tools work harder and to have more timing margin. For power calculation purposes you will want to use the exact clock frequencies your design will run on the board; otherwise the accuracy of the design dynamic power will be lower.
 - I/O view – If you know the data patterns of your I/O interfaces specify this activity in the applicable columns (**Signal Rate** and **% High**). Unless you are calculating the total power per supply in a separate tool, such as a spreadsheet, specify the termination technique for your outputs so XPA can include the amount of power the FPGA supplies to these external components.
 - Control signals view – XPA extracts and lists all the different control signals in the Clock Enable and Set/Reset views. You may know from the expected behavior of your application that some Set/Reset signals are not active in normal design operation so you may want to adjust the activity for these signals. Similarly, some signals in your application may disable entire blocks of the design when the blocks are not in use. Adjust their activity according to the expected functionality. Since synthesis tool and Place and Route algorithms can infer or remap control signals to optimize your RTL description, many of the signals listed in these views will be unfamiliar to you. When unsure of what these signals are, let the tool determine the activity.

4. Run the analysis.

Once you have provided XPA with the logic configuration and activity, then run the analysis. The tool will start by annotating the netlist with activity from files and user inputs, then apply the tool defaults for the remaining undefined nodes. Next, through an iterative process it will propagate this initial activity from the primary inputs to the primary outputs of your design to refine the activity estimate for the undefined nodes. Finally, it will calculate the dynamic power for each used resource and deduce the additional static power this switching activity generates, to compute the device's expected junction temperature and total power requirements.

5. Review your design power distribution.

Once the power analysis is complete you can open the Summary view to review the **Supply Power** and **Thermal Properties**. These tables highlight the total on-chip power and device junction temperature. These cells have a green background when the estimated junction temperature is within the nominal device operating range. They turn orange when the estimated value for junction temperature is above the Grade Maximum for the device but remains below the Absolute Maximum temperature. They turn red when the junction temperature is above the Absolute Maximum, as these operating conditions may cause damage to the device.

The **Supply Summary** section shows the current drawn for each supply source and breaks down this total between static and dynamic power. The **On-Chip** table shows the power dissipated in each of the device resource types. With this high-level view you can determine which parts of your design contribute most to the total power.

From there, you then get more details of the power at the resource level by opening the **Details** views. The different **Details** views are organized as tables (limited to 2000 entries to be more manageable). You can drag a column header to reorder the column arrangement. You can also click on a column header to change the sorting order. When the reported power exceeds your thermal or supply budgets you can refer to [Chapter 4, Tips and Techniques for Power Reduction](#), for a list of available techniques to reduce the device power. These techniques depend on how complete your design is and how tolerant to change your development process is.

6. Save or export the results.

Once you are confident with the results and have navigated through the different views to review the relevant information for your application you can do one or more of the following:

- **Save result as a text file** – For project documentation you may want to save the power estimation results. In other circumstances you may be experimenting with different mapping, placement, and routing options to close on performance or area constraints. Saving power results for each experiment will help you select the most power-effective solution when several experiments meet your requirements.
- **Save modified inputs as a settings file** – This is useful to save the analysis with its current settings so that you can reload these exact results some time in the future. XPA will save a file (.xpa) containing any information manually entered into the tool. This also proves useful when performing scenario analysis to estimate power under different environment corners or different modes/functions in which your application is expected to operate.
- **Export design for analysis in XPower Estimator** – Saves all environment information, device usage, and design activity in a file (.xpe) which you can later import into the XPower Estimator spreadsheet. This proves quite useful when your power budget is exceeded and you don't think that software optimization features alone will be able to meet your budgets. In this case, import the current implementation results into XPower Estimator, explore different mapping, gating, folding, and other strategies, and estimate their impact on power before modifying the RTL code or rerunning the implementation.

Power Closure Stage

Your design cycle will include a power closure phase under two main circumstances.

Constraints are met and you have time to optimize your design.

Admittedly, in today's complex system and with time-to-market pressure this is a rare situation. Typically at this stage in the development process you want to minimize changes to the RTL, board

power supply, and cooling parameters since, this involves a lot of verification or PCB respin costs. However you can use this time to experiment with different software options and constraints to optimize your logic and routing resource counts, configuration, and activity. This minimizes dynamic power and reduces static power at the same time. Depending on your design margins a 15% to 20% savings for your core dynamic power is a reasonable expectation, with some designs showing even more power reduction.

One simple way to perform this kind of experiment is to use the SmartXplorer feature in ISE or the Design Runs option in PlanAhead. These have a set of predefined strategies which adjust synthesis and Place and Route tool settings. You can also edit the existing strategies or even create your own. Once satisfied, run these strategies to explore the implementation solution, running either on a single machine or on a cluster of machines, to save on total runtime. You can then select the most power-effective options from among the successful runs and use the resulting netlist for your final application.

My power budget is exceeded! What can I do?

Typically at this stage the pressure to get the system to market is getting high and many parameters in your system are well defined, such as the board environment and cooling options. Even though this restricts the type of engineering rework you can do, the following methodology should help identify and focus on the highest potential areas for power reduction.

Step 1. Which power budget is exceeded?

GUI users can review the Summary view in XPower Analyzer, while command line users can use the **Summary** section of the **xpwr** command report file (.pwr). The **On-Chip** and **Supply Power** tables provide a high-level view of the power distribution. Navigate the Summary view to determine the type and amount of power that exceeds your budget.

Step 2. Identify the area on which to focus

Review the different detailed views in XPower Analyzer or XPower Estimator. Analyze the environment parameters and the power distribution across the different resources used, the design hierarchy, and clock domains. When you find an area of the design where power seems high, the following information should help you determine the likely contributing factors.

Thermal budget exceeded

If the estimated junction temperature exceeds the device specified operating conditions then you need to find ways to reduce the amount of power flowing into the device and dissipated as heat or improve the system thermal capabilities to facilitate heat extraction. Review the static versus dynamic power distribution ratio.

Reducing Static Power

- Power for supply sources such as V_{CCINT} or V_{CCAUX} is quite sensitive to process, voltage and temperature. Design resources contributing most to device static power are transceivers, I/Os, and Clock generation modules.
 - **Transceivers** – Consider using the power down and low power modes. Consider reducing the voltage swing of your transmitters. Maximize supporting circuits sharing across multiple channels such as PLL.
 - **I/Os** – Because I/Os have to drive and receive signals from longer distances, their transistors are much larger than the core transistors, resulting in a greater contribution to the total power per unit resource used.

- V_{CCAUX} – Typically supplies power to the input buffer. Review the I/O standard used. Some device families accept different voltage levels for this supply; evaluate if you can use a lower voltage.
 - V_{CCO} – Mainly used to supply power to the output buffer. Review the I/O standard, drive strength, and on-chip termination settings in the context of your performance needs and evaluate if you can use lower drive strength using tristatable DCI I/O standards (T_DCI), get by without terminations, or use external terminations.
 - V_{CCINT} – Powers the I/O interface with the device core logic. Make sure to enable only the minimum set of I/O features required by your application. Also consider using the low-power modes which some features provide, when your data rate allows.
 - Clock Generation modules – Typically the configuration for these modules is powered from the V_{CCAUX} supply and consumes more power than the interface to the device core logic. Focus your efforts to minimize the number of clock generation modules used. Since most blocks have many programmable outputs, frequencies, and phase shifts, often the same module can be used to generate the clock signals of unrelated IP blocks. Try to minimize the VCO frequency when selecting the multiply and divide factors to generate the desired clock rate.
 - Partial Reconfiguration – If your design encompasses multiple applications with different functionality depending on the environment, you may consider using Partial Reconfiguration to only program the device with the applicable functionality for each environment. This typically saves logic and routing resources and may allow you to use a smaller part.
- Device environment
 - **Voltage** – Xilinx guarantees a device will behave as expected for a supply range around the typical value. If your voltage supply regulators and other components attached to these rails support lowering the voltage level, then even a 1% or 2% reduction can have a significant influence on the transistor leakage and switching power. Voltage level influences both static and dynamic power.
 - **Thermal dissipation paths** – There are two main paths for the generated heat to escape the device. The heat can go up through the package and into the air or it can go down through the package balls, the board, and the air. Review each transfer coefficient between the device die and the surrounding environment. Lowering these thermal resistances allows more heat generated from the device to escape into the environment more quickly and lowers the device junction temperature, which in turn reduces transistor static power. You can evaluate if lowering the ambient temperature or increasing the airflow for your system or locally to the FPGA fan is possible. You can also consider adding a heat sink or changing the characteristics of the existing heat sink.

Reducing Dynamic Power

Factors contributing to the design dynamic power are: $\sum (\alpha \cdot f_{clk} \cdot C_L \cdot V^2)$

where the factors in the equation are:

- **Activity (α, f_{clk})**

The less often components and signals toggle the less power they consume. I/O, clocks, arithmetics, memories, and some bitwise logic typically are the design portions which generate the most activity in a design. Different options are available to reduce this activity.

- **User intervention** – From your understanding of the design behavior, you can turn off modules in your design when their outputs are unused. You can disable inputs so no new data is captured. You can also add clock enable signals to gate entire design blocks, I/O interfaces, or clock domains. Reducing clock frequency is rarely an alternative, but some

applications can be arranged to modulate the clock frequency depending on the availability of input data.

- **ISE power optimization algorithms** – At a very fine-grained level and across hierarchy boundaries, these algorithms are capable of detecting sequences in your design data flow where outputs are not used and gate the clock and/or logic during the unused cycles. Examples are: disabling unused multiplexer inputs by predicting the selector value, or disabling RAM ports when no read or write operation is needed.

- **Capacitance (C_L)**

The capacitance which needs to be driven at each toggling event varies with the type, fanout, and capacitance of logic and routing resources used in the design. First, review the design constraints. Efficient constraints (without overconstraining) guide the implementation tool to optimize only the timing-critical paths in your design for performance, while minimizing area and routing structures used for the other paths.

- **Signal routing** – it is usually best to let the Place and Route tool decide which specific routing resource to use since there are many factors to take into account for each path and the surrounding logic. But you can use floorplanning techniques to cluster logic into specific clock regions or bring highly interconnected logic closer together. The idea is to minimize the length of high fanout signals or highly active paths in the design.

- **Voltage (V^2)**

Voltage is the main external parameter on which you have leverage for dynamic power. Xilinx guarantees the device will behave as expected for a supply range around the typical value. If your voltage supply regulators and other components attached to these rails support lowering the voltage level then even a 1% or 2% reduction can have a significant influence on the FPGA total power, since voltage influences both static and dynamic power.

- **Number of elements (Σ)**

Reducing the number of elements will reduce the total capacitance to be switched. Efficient HDL code for the target and tool constraints help in this approach. Converting some glue logic to use hardened blocks such as DSPs or BRAMs also helps. Remember that Shift registers can be implemented in a special mode of the LUTs to reduce the number of register and routing structures used. Other techniques to reduce area, such as time multiplexing, partial reconfiguration, etc., can be pursued.

- **Balanced approach**

In most cases, after your analysis you will find that adjustments on both the static and dynamic aspects will be the easiest way to reduce the power enough to return within your allowed budget.

Supply power budget exceeded

You may run into a situation where your total power budget is exceeded since some applications or standards define the maximum power a single PCB can draw. There are other situations where your design may have reached the maximum current the selected voltage regulator can provide. Because an FPGA is programmable it is only natural to look into minimizing its power since this is one of the components in the system over which you have the most control. The total supply power includes the thermal power plus any power flowing from the supplies through the FPGA and dissipated in outside components. All the techniques and effects to reduce the internal power described in the [Thermal budget exceeded, page 32](#) section above can be explored. In addition, the main contributors to off-chip power are the external components the FPGA is driving. These are typically resistive termination loads or specific devices such as power transistors, LEDs, or other components. You can calculate the total power difference of your I/O interface when using on-chip terminations instead of off-chip terminations. The FPGA may disable on-chip terminations when not use. On the other

hand, bringing terminations on-chip will increase the power consumed by the device and therefore increase its static power. It is important to look at the total power for each scenario before selecting the best topology for your application.

Step 3. Experiment

With the list of candidate areas in your design for power optimizations derived from the previous step, you can now sort this list from easiest to most involved and decide which optimization or experiment to perform next. The power tools allow you to do What If? analysis so you can quickly enter design changes and estimate the power implications without having to actually edit any code or constraint or rerun the implementation tools.

- Experiment within the implementation tools

Review your synthesis and implementation tool options to turn on power and area reduction on your entire design or a portion of it. The Xilinx tools are unique in that power optimization algorithms can gate design logic automatically for an average 15% to 20% core dynamic power saving, which is significant and does not require any code change or additional functional verification from you.

- Experiment within XPA

In XPA you can make adjustments then rerun the analysis to review the power implications for these factors:

- **Environment** – Includes thermal parameters, process, or voltages.
- **Design Activity** – Adjust the activity of nets or instances in the design. Change one item or change multiple items at a time. You can also change:
 - Clock domains – Adjust the switching frequency.
 - Glue logic – Adjust the dynamic activity rate.
 - I/Os – Adjust both static and dynamic activity probabilities. You can also adjust parameters for the external components connected to the device outputs, such as the load capacitance or the near-end board termination details.
 - Signals – Adjust the dynamic activity rate for data signals. For control signals you can also adjust the static probability to evaluate power under different Clock Enable, Set, or Reset scenarios.
 - Specific blocks – In addition to the dynamic activity probability you can also adjust the activity of control ports such as port enables or write enables on block RAMs.

- Experiment within XPE

In XPE you can import XPower Analyzer results from modules developed by multiple sources to review the total power once these separate IP block are implemented in the device. You can also evaluate situations where you would have to change the netlist, and evaluate the power implications without having to actually make the code changes. For your design core logic XPE works at a coarser resolution than XPA, since you cannot adjust each logic element or signal individually in XPE.

In XPE, you can also experiment with:

- **Resource usage** – Explore reducing the resource count. Try remapping pieces of logic from slice logic to dedicated blocks such as BRAM or DSP, and vice versa.
- **Resource configuration** – Explore using different configuration settings for the design I/Os, block RAMs, clock generators, and other resources.

- Experiment within PlanAhead RTL Power Estimator

If you need to modify your RTL code to reduce power, the RTL power estimator provides features to navigate the power consumed by resources and the design hierarchy and to identify the design areas contributing the most to the device power. From this fast analysis you can quickly derive design constraints and tool options to guide the synthesis or Place and Route. Examples are: directives to map a particular state machine differently or the most effective power optimization options. You can experiment with adding a pipeline or performing power retiming around high-activity logic such as carry chains and XOR functions. Although long paths with carry chains tend to be on slower clock domains, they exhibit more glitching activity, which increases the design power. Retiming or pipelining these paths is often beneficial.

Step 4. Implement the changes and review the power saving

Once you have determined the best changes to make given your time, performance, and resource constraints, proceed with implementing them. It is worth mentioning that trying too many options or changes at once may not yield the best results because of potential conflicts or interactions between them. Best practice, if time allows, is to experiment with a few options at a time so you can evaluate their effect on power and other constraints before adding on other changes.

Power and Temperature Measurements

This section briefly describes different mechanisms available to measure FPGA power consumption or heat dissipation. Some of these techniques use internal FPGA resources; others use board or external components. Some applications require power and temperature to be actively monitored and adjusted after deployment; other applications would use these measurement techniques in the lab during prototyping and validation phases. Consider what most applies to your design.

Power Measurements

- **Current Sense Resistor** – Inserted in series between the regulator output and the FPGA, this small precision resistor will create a small voltage drop which, by Ohm's Law, is proportional to the flowing current. Measuring this voltage gives you the current being supplied to the FPGA.
- **Advanced Regulators and Digital Power Controllers** – The latest evaluation kits have advanced regulator and power controllers which you can use to capture regulator output currents and voltages, then send this information to a monitoring computer via a USB interface. This is the simplest and most convenient way to monitor the power rails. The ML605 and SP605 boards have integrated Texas Instrument UCD92xx controllers which can be accessed with the Fusions Digital Power Designer software on a PC via a PMBus (I2C) to USB interface module.
- **On-board Monitoring** – The latest Xilinx device families provide internal sensors and at least one analog-to-digital converter to measure supplied voltages and device temperature. The ChipScope utility provides real-time JTAG access to measure the different supply source voltages or device junction temperature before and after device configuration (see [Figure 3-4](#)). You can also instantiate a System Monitor or XADC component in your code to access these measurements from your FPGA application.

Thermal Measurements

- External Monitoring** – Junction Temperature cannot be measured directly since the device package prevents access to the silicon. Junction Temperature can however be approximated by measuring the temperature of the package, the heat sink, and other locations with a thermocouple. Thermal cameras are also used to visualize the device temperature and thermal dissipation interactions with neighboring components and the larger environment.
- On-board Monitoring** – Thermal measurements are possible and use the same mechanisms as power measurements. You can use ChipScope before and after device configuration (see [Figure 3-4](#)), or use the System Monitor primitive within your design to read the device junction temperature.

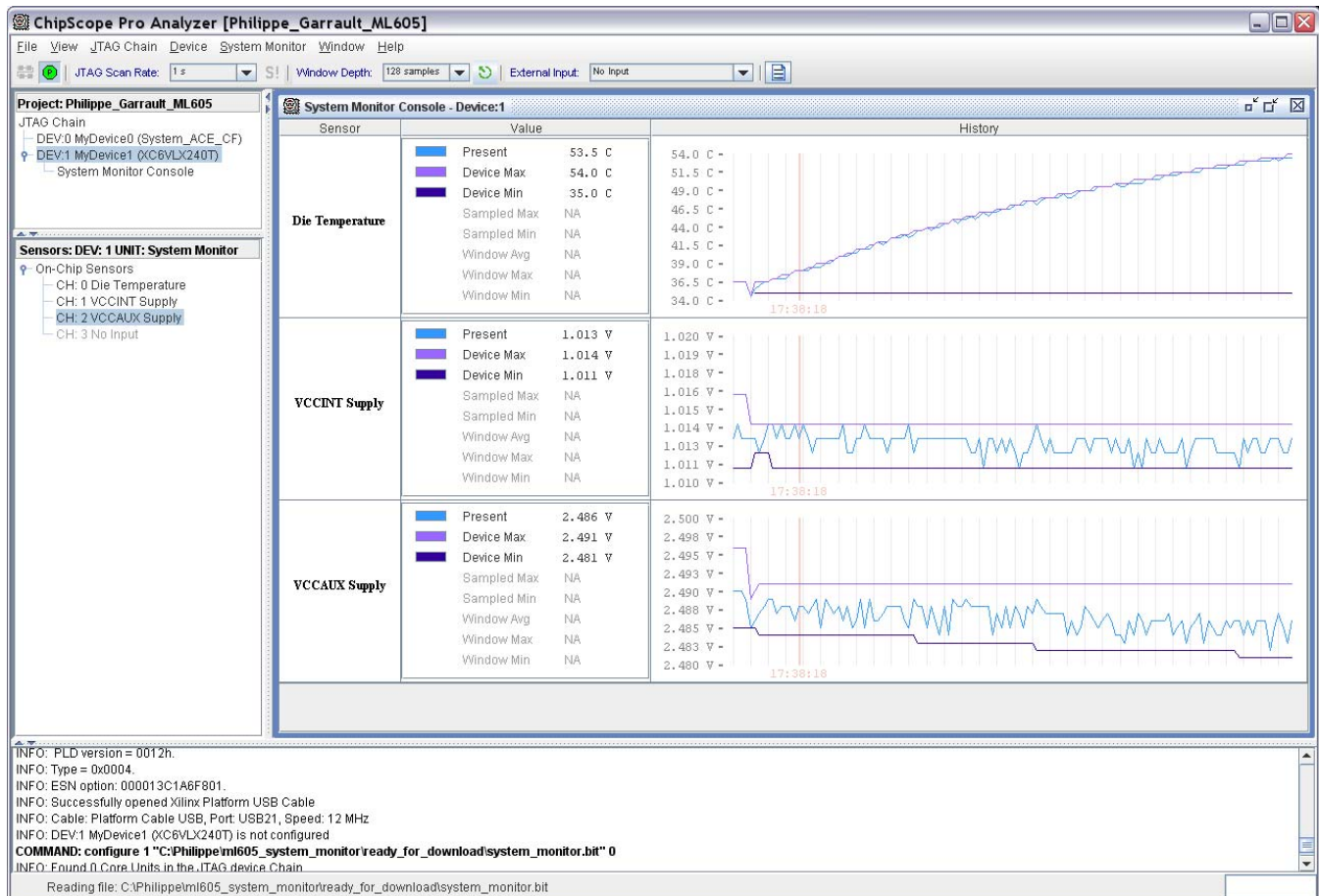


Figure 3-4: Voltage and Junction Temperature Monitoring with ChipScope

Methodology for Power and Temperature Measurement

To evaluate the three components contributing to the design total power it is important to control the device junction temperature and let it stabilize before making measurements. This is because the device and design static power is heavily dependent on the device junction temperature.

- **Device static** – First download a blank design to ensure no input noise is captured and all internal logic and configuration circuits are in a known state. A blank design is a design with a single gate or flip-flop which never toggles, and ensure all outputs are tristated. Then wait for the junction temperature to stabilize and measure V_{CCINT} , V_{CCAUX} and any other supply source of interest. With special equipment, a simple heat gun, or cold spray you can force temperature changes to evaluate the influence of the environment on the device static power.
- **Design static** – Download the design into the FPGA and turn off all input and internal activity (input data, external and internal clock generation). Wait for the device temperature to stabilize, then measure power on all supply rails of interest. Subtracting the device static measurement from these values gives you the additional static power from the specific logic resources and configuration used in your design (design static power).
- **Design dynamic** – Download the design into the FPGA and provide clocks and input stimulus representative of the design. Wait for the junction temperature to stabilize before measuring all supply sources of interest. This power represents the instantaneous total power of your design. It will vary with the change in activity at each clock cycle.

Tips and Techniques for Power Reduction

This chapter describes power reduction techniques and their expected effect on total device power. This information will help you evaluate your best options depending on your time, power budget, available resources, and freedom to change your design. A few techniques are mentioned here only briefly, for completeness. To avoid replication, refer to the “experimentation” sections in the previous chapter for more information.

System Level

Cooling strategy

Cooling strategy ensures that heat generated from the device is extracted and absorbed by the environment. These cooling strategies, which are generally available at the beginning of the design and become less feasible in the later stages, have a significant impact on the device static power:

- Increase the airflow,
- Lower the ambient temperature,
- Use a heat sink (or a larger heat sink), or select a different regulator.

Supply strategy

Voltage has a large effect on both static and dynamic power. Active control of the voltage level ensures the desired voltage is applied to the device.

- **Use switching regulators** – These are more power efficient than Linear regulators, at the expense of requiring a higher component count.
- **Use adjustable regulators** – Sense voltage as close as possible to the FPGA and to the highest consuming device if the same supply powers multiple FPGAs.
- Select regulators with tight tolerances.

Device selection

- **Select the best device for the product** – Vendor, density, functionality, and performance are no longer the primary factors for selecting a device. System level decisions for implementing the functionality can help minimize power consumption for the total product.
- **Minimize the number of devices** – This saves space, I/O interconnect power, total leakage, and other factors. Typically, replacing multiple components (for example, processor and FPGA) with a single larger FPGA consumes less static power.

- **Select the smallest device possible** – This reduces leakage. Typically in an FPGA family the same package may be available with different die sizes. You can, for instance, use a larger die during the prototyping and pre-series phase, then move to a smaller die for volume production.
- **Select the largest package possible** – This increases heat dissipation. A larger package has a larger area to dissipate the die heat into the environment. A larger heat sink can be attached to the package upper side and more heat can escape onto the PCB via the bottom ball grid array.
- **Use low voltage devices** – Some device families are available with a lower power option. The lower core voltage requirements translate into significant static and dynamic power savings.
- **Use low leakage devices** – Some device families are available with a lower leakage or static power options in the form of specific speed or temperature grades. These devices may cost a bit more to purchase but you or the end user may be able to more than offset this with savings on the electricity bill or cooling hardware and system maintenance.

Device Level

Xilinx designer teams have the task of finding innovative yet rational solutions to enhance device capabilities and meet your application challenges. In practice this translates into two main areas of extensive research and experimentation: manufacturing process and architectural parameters. These topics were covered in [Device Power Contributing Factors in Chapter 1](#).

Design Level

Get accurate power information

To identify the best areas on which to focus your power minimization efforts make sure to get realistic power estimates from the tools.

- **Specify the expected device operating conditions** – The device, its thermal specifications, and applied voltages are important factors in determining the device supply and thermal requirements.
- **Specify resource usage, configuration and activity.**
 - Provide the tools with the maximum information known, especially for resource usage and activity for clocks, control signals, and primary inputs.
 - Reuse data or knowledge from a previous design or use the import capabilities in the tool to minimize manual data entry.
 - When specifying activity, make sure it matches normal or worst case operation. If the design processes data by bursts followed by quieter periods, then make sure to normalize the activity over a longer time period. Thermal and supply effects tend to have a response time much slower than the internal switching logic.

Use resources more efficiently

Logic – As obvious as it is, sometimes we're too busy to spend time to fully understand the targeted architecture. This knowledge helps to:

- Optimize the design description
 - Write code which allows the synthesis tool to map logic to dedicated blocks. The synthesis tool can then decide which mapping best meets your timing and utilization requirements. For example, counters or state machines can be mapped to either distributed logic or block RAMs. Shift registers can be mapped to a special LUT mode to save area and power.

- Minimize asynchronous control signals which prevent logic optimization and uses more routing resources.
- Minimize the number of control sets. A control set consists of the unique grouping of a clock, clock enable, set, reset, and, in the case of LUT RAM, write enable signals. Control set information is important because count limits or sharing of signals within a slice may occur. This varies with the FPGA architecture, and when the limit is reached can prevent proximity packing of related logic, which would increase routing resources.
- Add pipeline levels to minimize the size of combinatorial logic cones. This minimizes the propagation of glitches between registers until signals reach their final state at each clock cycle.
- Use resource time sharing. These techniques minimize device resource usage by time multiplexing different functions to the same hardware resources. This allows you to use a smaller device or can reduce placement and routing congestion, which will lower both static and dynamic core power.
 - Processes which are slow and similar can be performed on the same resources instead of separate resources. This requires careful thinking for how to bufferize, multiplex, initialize, and control the data to be processed. Typical applications for such optimization are similar parallel processes, such as processing multiple input sensors. Instead of having as many processing units as inputs, you could use a single processing unit and make it run faster, so it processes input channels one after the other while ensuring the same response time for each output. An XPower Estimator What If? estimation can help you decide whether the power savings are worth the engineering effort.
 - Partial Reconfiguration. Use this technique to change functionality on the fly and eliminate the need to fully reconfigure and re-establish links. Partial Reconfiguration is particularly useful to minimize resources used, and therefore both static and dynamic power. It allows you to load the FPGA with only the functions that are needed for the system's current environment or for the particular time in the overall process of the application. More details and methodology guides for Partial Reconfiguration are available here:
<http://www.xilinx.com/tools/partial-reconfiguration.htm>
 - Use the DSP and BlockRAM optional registers. For example, in DSP blocks the multiplier or MREG registers, when enabled, are the most power efficient implementation as they minimize the propagation of internal glitches between clock cycles.

- **Minimize the number of resources** – Minimizing logic resources has the compounding effect of reducing the routing resources used. This in turn provides additional degrees of freedom to the implementation software to more efficiently Place and Route your design.
 - Remove debug logic from your final design, rather than just disabling it.
 - Minimize the number of clock generation components. Think about clock generation and management from the design top level perspective rather than from major IP blocks to prevent unnecessary replications of functionality. Clock components often have multiple outputs with separate programmable frequency and phase shifting capabilities. The project leader can evaluate if instead of instantiating one clock manager in two separate IP blocks, the different clocks domains required could be generated from a single clock generator.
- **Minimize activity** – Component and signal routing activity is a major contributor to the device core dynamic power.
 - Gate clock or data paths. This common technique stops paths when the results of these paths are not used. Gating a clock stops all driven synchronous loads; gating a data path keeps signal switching and glitching from continuing to propagate up to the next synchronous element. The software tools analyze your description and netlist to detect such conditions. Still, there are things the designer knows about the application, data flow, and dependencies which are not available to the tool, and can only be specified by the designer.

General guidelines for gating:

- Maximize the number of elements affected by the gating signal. For example, it is more power efficient to gate a clock domain at its driving source than to gate each load with a clock enable signal.
- When gating or multiplexing clocks to minimize activity or clock tree usage, use Clock Enable ports of dedicated clock buffers. Inserting LUTs or other methods to gate-off clock signals is not efficient for power and timing considerations.
- Minimize the number of control sets. Since adding gating signals to stop the data or clock path may require additional logic and routing it is important to minimize the number of such structures to avoid defeating the original purpose. Placement and routing of these additional resources may complicate the implementation of the existing logic. This may result in spread-out placement, additional replications, or increased routing congestion which would then increase the dynamic power.
- Disable BlockRAM ports. In your application, when not reading or writing in the array make sure to describe enable signals to disable memory ports when not in use.
- Use suspend or standby mode to disable the device when not in use. Capabilities vary with the particular FPGA family targeted. This technique is often used for battery applications, or when the data to be processed comes in bursts followed by long periods of inactivity. This mechanism is simple to implement and lowers both the device static and dynamic power consumption. Remember to make sure your application allows time for the device to boot up when it returns from the power down mode.
- Use power down modes for individual architecture components. Most of the device components have features which allow you to turn off circuits or disable the clock or data flow (for example, the block RAM port enable and the transceiver multiple power down modes). Learn about the architecture to maximize the use of these dedicated power saving structures.
- **Guide placement** – Generally it is best to let the software control the placement of your design logic. You may think that constraining logic into a small device area will save dynamic power, since routing resources will be shorter. But this may cause congestion and signals may use suboptimal resources just to get around this artificially created congestion. Yet there are many circumstances where guiding placement is useful, for instance:

- Force denser placement of high activity logic when the timing or other constraints are loose. Arithmetics and bitwise functions such as XOR can produce glitching activity between clock cycles, which increases the dynamic power. Forcing a closer proximity between these functions can shorten the routing resources used and keep paths within the same Slice or CLB, which is more power efficient.

I/O – Because I/O interfaces have to drive long distances with potentially more parasitic effects, they typically represent a large portion of the device power requirements.

- Use the lowest V_{CCAUX} possible. This minimizes both the static and dynamic power for this voltage supply.
- Inputs - Limit usage of internally referenced input standards.
- Outputs:
 - Use the lowest slew/drive/voltage level supported by the receiving chip(s).
 - No termination or series termination are preferred over parallel terminations. Signal integrity simulation tools can help with this determination.
 - Consider whether using on-chip or off-chip termination is the best option given your device thermal budget, system cost, and board real estate requirements.
 - Evaluate using lower voltage swing differential standards.
 - Evaluate if your application allows you to use transceivers instead of large parallel busses.
 - Evaluate the requirements of I/O features such as IBUF, IO DELAY, and others, and disable when performance allows.

Transceiver

- Use low power modes to disable some circuitry when not in use.
- Pack the maximum number of transceivers into a single tile to minimize duplicating supporting circuits.

Note: Avoid over-engineering. Maybe designing for simultaneously worst case device, process, environment, and design activity is not necessary. Sometimes, other components in the system would stop functioning before such a condition occurs. In this case you need to make sure the absolute maximum junction temperature is not exceeded and that the device can restart from a known state. Processing data under such circumstances may not be a concern since upstream or downstream board components may not be operational.

Software Settings and Algorithms Level

Synthesis

Synthesis tools make many trade-offs between area, performance, runtime, and power constraints. Priority is given to meeting either the density or performance goals. All non-critical paths are then optimized to minimize area and power consumption. The following sections highlight methodologies, flows, and constraints to further optimize the synthesized netlist.

The options and techniques presented refer to Xilinx XST, however other FPGA synthesis tools such as Synopsys Synplify or Mentor Precision products offer similar functionality.

General

- Provide complete and realistic timing constraints.

This helps the tool focus performance optimizations on the critical paths and frees the remaining paths for area and power optimizations. Artificially over-constraining the performance requirements generates excessive replications and inefficiently maps the functional description to hardware resources. Under-constraining can lead to sub-optimal mapping, which would make it difficult for Place and Route to achieve the desired performance.

- Read black boxes.

This provides timing information for paths to and from the black boxes plus resource usage within the black boxes. This avoids unnecessary replications or suboptimal use of logic resources.

XST Power Optimizations

- Minimize the number of simultaneously active block RAM ports.

This optimization, enabled with the **-power yes** option in XST, modifies the decomposition of RAM or ROM descriptions which span multiple block RAMs. The optimization adjusts address lines, port enable and write enable control signals to minimize the number of active block RAM ports at each clock cycle, while ensuring timing constraints are met.

- Force the most power efficient mapping of block RAMs regardless of the impact on performance.

Use the **block_power2** option to the **ram_style** constraint when you know the timing paths related to this memory are not critical. Savings range from 15% to 75%.

XST Area Optimizations

- Use the **Area** optimization mode.

Whenever possible, use the **Area** optimization mode in XST. This will minimize the number of resources used.

- Force mapping decisions.

For specific logic blocks, after a careful estimation of the performance implications you can constrain the tool to map functions to dedicated logic resources. For example, you can force counters to be mapped onto available DSP blocks. Shift registers can be mapped to the SRL mode of slices. Another example: when you find shallow memories implemented as block RAMs with long input or output routes you may want to force mapping to distributed RAM since this may reduce the routing power.

- Use resource sharing.

Resource sharing minimizes the number of arithmetic operators, resulting in reduced device utilization. Similar arithmetic operators can be implemented with common resources on the device, provided their respective outputs are never used simultaneously. Resource sharing usually involves creating additional multiplexing logic to select between factorized inputs. This factorization minimizes logic usage and prevents logic replications. Synthesis tools perform these optimizations by default and control the potential performance side effect; therefore, you should rarely have to disable this optimization.

Miscellaneous XST options

- Use Register Balancing to reduce activity.

The **-register_balancing** option tries to forward or backward retime registers to equalize the length of timing paths in combinatorial logic. Although this is primarily done for design performance this can prove useful to reduce activity, since shortening the longest timing paths minimizes glitching propagation. Register Balancing is especially useful for paths which implement high activity logic such as arithmetics or wide bitwise structures.

- FSM encoding style.

You can experiment with the encoding style of your large state machines. Grey encoding, for instance, minimizes the number of bit changes between state transitions. The different encoding styles affect the amount of logic necessary to decode the next state compared to the logic necessary to generate the outputs.

Implementation

The following sections present different algorithms you can use separately or combine for additional gains.

Netlist optimization

- Enable activity aware optimizations (intelligent gating).

These algorithms analyze the logic equations to detect, for each clock cycle, sourcing registers that do not contribute to the result. The software then utilizes the abundant Clock Enable (CE) resources available in the FPGA logic to create fine-grained gating signals which neutralize useless switching activity (see Figure 4-1). This intelligent clock and data gating is controlled with the **map -power high** option. Total core dynamic power reduction in excess of 15% is possible and in most cases the additional gating logic inserted does not affect performance.

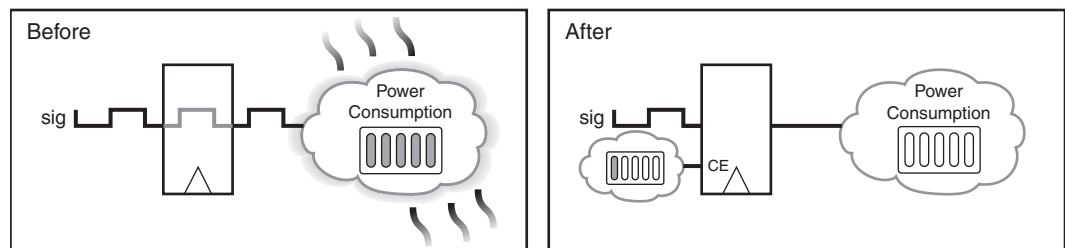


Figure 4-1: Intelligent Clock Gating Reduces Switching Power Consumption

- Resource reallocation.

This netlist optimization remaps high fanout nets to less power-consuming routing structures. For example, a reset signal which is distributed to a large number of loads may be mapped to a global net. Global nets are dedicated routes optimized for performance and also have a lower

capacitance. This also frees routing resources on which to Place and Route the remaining data paths, which reduces routing congestion and overall capacitance. All of these effects, which are controlled by the **map -power on** option, reduce the core dynamic power.

Placement

- Capacitance aware optimizations.
 - **Group clock loads** – This process reorganizes the placement of synchronous elements such as Flip-flop or DSP blocks to minimize the reach of each clock net. When clock loads are placed along a minimum number of horizontal or vertical clock spines the unused branches in the clock region can be disabled by the software. This reduces the clock resources used and buffering requirements, which saves core dynamic power. This process is controlled by the **map -power on** option.
 - **Group data loads** – This algorithm minimizes the total wire length in your design while ensuring performance requirements are met. Grouping data loads saves power since the dynamic power increases with the fanout and type and length of routing structures used, due to the associated increased capacitance. The grouping algorithm, which is also enabled with the **map -power on** option, achieves power reduction by placing related logic closer together.
- Activity aware optimizations.
 - **Provide design activity from simulation results** – This helps the placer to more efficiently prioritize and floorplan the netlist for minimum power consumption. In addition to voltage and capacitance, activity is an essential factor determining the dynamic power. By default the placer tries to meet the performance and routability goals of your design. The activity information from simulation results further guides the placer when placing logic and paths with high activity. This results in higher placement density with more internal CLB routing for these structures, which results in lower core dynamic power. Use this option to enable these algorithms:

map -activity_file *file_name.saif*

Make sure the netlist and simulation output components can be effectively matched (identical hierarchy separator, top-level name, etc.).

- **Important:** Consider using the **map -power xe** option, which enables both capacitance and gating algorithms. Experience shows this is the most effective option to reduce core dynamic power.

Miscellaneous

- Check for tool updates.

As for most IC components, power characteristics of an FPGA are refined over time as the manufacturing process matures. In the early stages of an FPGA, power characteristics are derived from simulations. Then, as engineering samples become available, measured data can be integrated into the power models. Finally, once the device is in full production the process variances across multiple manufacturing batches can fully be characterized, and these measurements can update the power models. Over time, the tools for power estimation and analysis will use this updated information to produce more accurate results.
- **Sweep software power options.**

One simple way to sweep software options affecting power is to experiment with the SmartXplorer feature in ISE or the Design Runs option in PlanAhead. These have a set of predefined strategies which adjust synthesis and Place and Route tool settings. You can even

add your own strategies or edit the existing ones and distribute the different runs on multiple machines to speed up the exploration.

Comparing Devices or Architectures Effectively

To compare devices or architectures:

- Match thermal dissipation factors.
Since different tools have different options and default settings, you must match the thermal dissipation factors when you do your comparison. This will help you accurately compare static power.
- Match resources used.
Since different architectures have different resources available (Shift registers, Memory controllers, Clock managers, etc.) and different sizes of resources (LUTs, BRAMs, DSP blocks, etc.), make sure to enter the equivalent number in both tools. For example, an architecture which does not support shift registers implemented with LUT SRLs will require many more flip-flops to achieve the same functionality as an architecture which does support it.
- Match I/O settings.
Different architectures may support different I/O standards, termination or data capture, and alignment blocks. Make sure to represent the same voltage level and functionality when you do comparisons.
- Match activity.
Tool defaults may vary between architectures and vendors. Ensure that this information is adjusted before making comparisons.
- Understand what is included in each result value presented.
Ask yourself questions such as: What is included? What is excluded? Are contributions from all supply voltages included? Is the power dissipated in off-chip termination included?

Summary

The principles covered in this Guide can help you perform design power estimations throughout the design cycle. The analysis methodology helps you identify the main contributors to the total power, and the tips and techniques help you decide how to most efficiently address excess power consumption. Of course each design is different, so you can pick and adjust this material according to your specific device, environment, processes, and looming deadlines.

Minimizing power will ultimately help your application do more within the same power footprint. It will reduce end product operating cost from both the energy bill and reliability and maintenance costs.

Additional Resources

Xilinx Resources

- **Device User Guides:**
http://www.xilinx.com/support/documentation/user_guides.htm
- **Xilinx Glossary:** <http://www.xilinx.com/company/terms.htm>
- **ISE Design Suite 14: Release Notes, Installation, and Licensing (UG631)**
<http://www.xilinx.com/cgi-bin/docs/rdoc?v=14.5;t=release+notes>
- **Product Support and Documentation:** <http://www.xilinx.com/support>
- **Xilinx Power Solutions:** <http://www.xilinx.com/power>

Hardware Documentation

- **7 Series Device Documentation:**
http://www.xilinx.com/support/documentation/7_series.htm
- **7 Series FPGAs SelectIO Resources User Guide (UG471):**
http://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf
- **Virtex-6 FPGA Configuration User Guide (UG360):**
http://www.xilinx.com/support/documentation/user_guides/ug360.pdf
- **Virtex-6 FPGA SelectIO Resources User Guide (UG361):**
http://www.xilinx.com/support/documentation/user_guides/ug361.pdf
- **Spartan-6 FPGA Configuration User Guide (UG380):**
http://www.xilinx.com/support/documentation/user_guides/ug380.pdf
- **Spartan-6 FPGA SelectIO Resources User Guide (UG381):**
http://www.xilinx.com/support/documentation/user_guides/ug381.pdf
- **Spartan-6 PCB Design Guide (UG393):**
http://www.xilinx.com/support/documentation/user_guides/ug393.pdf
- **Virtex-5 FPGA Configuration User Guide (UG191):**
http://www.xilinx.com/support/documentation/user_guides/ug191.pdf
- **Virtex-4 FPGA Configuration User Guide (UG071):**
http://www.xilinx.com/support/documentation/user_guides/ug071.pdf

ISE Documentation

- **Libraries Guides:**
http://www.xilinx.com/support/documentation/dt_ise14-5_librariesguides.htm

- **ISE Design Suite Documentation:**
http://www.xilinx.com/support/documentation/dt_ise14-5.htm
 - **Command Line Tools User Guide (UG628):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/devref.pdf
 - **Constraints Guide (UG625):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/cgd.pdf
 - **Data2MEM User Guide (UG658):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/data2mem.pdf
 - **ISim User Guide (UG660):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/plugin_ism.pdf
 - **Synthesis and Simulation Design Guide (UG626):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/sim.pdf
 - **Timing Closure User Guide (UG612):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug612.pdf
 - **Xilinx/Cadence PCB Guide (UG629):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/cadence_pcb.pdf
 - **Xilinx/Mentor Graphics PCB Guide (UG630):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/mentor_pcb.pdf
 - **XPower Estimator User Guide (UG440):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug440.pdf
 - **XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices (UG627):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst.pdf
 - **XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst_v6s6.pdf
- **ISE Methodology Guides:**
 - **Large FPGA Methodology Guide (UG872):** http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug872_largefpga.pdf
- **ISE Tutorials:**
 - **ISE In-Depth Tutorial (UG695):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ise_tutorial_ug695.pdf
 - **ISE RTL Technology Viewer Tutorial (UG685):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug685.pdf
 - **ISim In-Depth Tutorial (UG682):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug682.pdf
 - **Using Xilinx ChipScope Pro ILA Core with Project Navigator to Debug FPGA Applications (UG750):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug750.pdf
 - **Xilinx Power Tools Tutorial (UG733):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug733.pdf

PlanAhead Documentation

- **PlanAhead User Guide (UG632):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/PlanAhead_UserGuide.pdf
- **Floorplanning Methodology Guide (UG633):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/Floorplanning_Methodology_Guide.pdf

- ***Hierarchical Design Methodology Guide (UG748):***
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/Hierarchical_Design_Methodology_Guide.pdf
- ***Pin Planning Methodology Guide (UG792):***
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug792_pinplan.pdf
- ***PlanAhead Tcl Command Reference Guide (UG789):***
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug789_pa_tcl_commands.pdf
- **PlanAhead Tutorials:**
http://www.xilinx.com/support/documentation/dt_planahead_planahead14-5_tutorials.htm
 - ***Quick Front- to-Back Flow Overview (UG673)***
 - ***I/O Pin Planning (UG674)***
 - ***RTL Design and IP Generation (UG675)***
 - ***Design Analysis and Floorplanning (UG676)***
 - ***Debugging with ChipScope (UG677)***
 - ***Team Design (UG839)***
 - ***Design Preservation (UG747)***
 - ***Partial Reconfiguration (UG743)***
 - ***Reconfiguration with Processor Peripheral (UG744)***

Power White Papers

- ***Lowering Power at 28 nm with Xilinx 7 Series FPGAs (WP389)***
- ***Reducing Switching Power with Intelligent Clock Gating (WP370)***
- ***Seven Steps to an Accurate Worst-Case Power Analysis Using Xilinx Power Estimator (XPE) (WP353)***
-

