IP Design - Efficient and Fast Prototyping and Porting to ASIC using Synopsys Tools.

Thomas Varghese
Prasanth R I

Mindtree Ltd
Bangalore, India

www.mindtree.com

ABSTRACT

The article summarizes our IP design lifecycle and some of the IP design strategies we practice - describes the various design strategies, optimizations and techniques we have used for keeping a check on the power consumption challenge and hence achieving industry best numbers in current consumption. - How Synopsys tools were efficiently used in different phases of the IP development cycle.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Mindtree Short Range Wireless group is a part of the Engineering R&D Service Line which offers various wireless IPs targeted to ASICs on various technology nodes.

With increased SoC complexities it becomes extremely important that any functional defects are detected as early as possible. One relatively cheaper and efficient way achieving this is the FPGA prototyping which also provides a relatively more accurate performance with respect to the ASIC.

In this paper we highlight the importance of IPs in an SoC, the development cycle of an IP and advantages of FPGA prototyping for IP validation as well as system validation, good design practices and steps which will be helpful for any IP development and could be adopted to shorten the IP development cycle and hence more time for system validation. We take a reference design and explain how efficiently we can use the Synopsys Tool chain in various stages of the development lifecycle and how it can make a positive impact on the time to market (TTM) and the quality of results (QOR).

2 IP Development

Before talking about IPs we shall talk about SoCs where we can get a complete system integrated on to a single chip. A typical SoC might contain more than one processor core, different storage memories (ROM, RAM, EEPROM, FLASH), Clock sources (PLLs), Power circuits (Voltage Regulators, DC-DC level shifters), Peripherals (RTC), External Interfaces (USB, Ethernet, UART), Analog Interfaces (ADC, DAC) and Embedded Software. Some types of SoCs offer customizable solutions using specialized, reconfigurable processors.

2.1 SoC Architecture

Complex SoCs with a multitude of supported protocols and communication interfaces are seen as a common design part even in smartphones now a days. Getting such a complex design into production has also become more challenging not only because of the stricter time to market guided by the highly competitive end market but also a higher demand on the non-functional metrics like power, area and pin count. Underestimation of the complexity of the design will end up in schedule over-runs and hence will affect success of the product. This will include setting up realistic plans and deadlines and sufficiently prepared human resources.

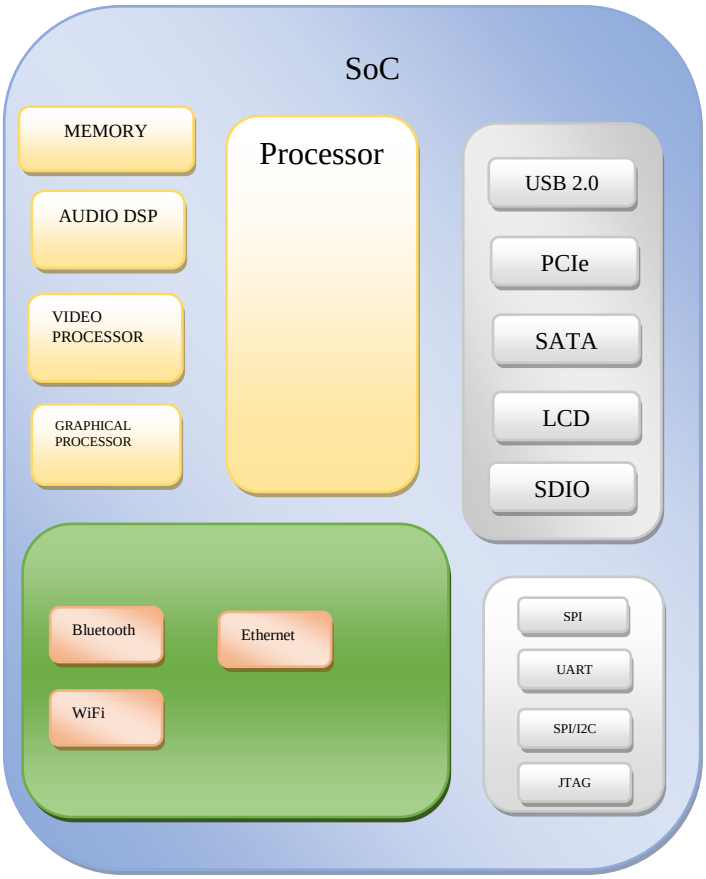


Figure 1 SoC Architecture

2.2 IP as a building block of a SoC

It becomes a difficult task for the SoC designer to validate the individual IPs in a SoCs. Also putting more effort on the validation of the non-strategic IP modules inside an SoC may not help to differentiate your product from the others in the market. The product is going to sell becuase of its end functionality and is not becauseof having an excellent peripheral.For example if we take a camera the camera is going to sell because of its ability to take pictures with very fine details , its performance in the low light conditions and its ability to enhance the image using the various image processing algorithms.The camera is not going to sell just because of having an excellent USB or Wi-fi connectivity alone.So it makes complete sense for an SoC designer to put effort in the integration aspects of the system and on the system validation rather than concentrating on developing individual IPs from scratch.

This suggests IP usage as a powerful way to speed up the product launch. These IPs can be Soft IPs (Synthesizable RTL, process independent) or a Hard IPs (synthesized netlist with timing information, easy to integrate and has predictable performance.) Any IP taken for SoC integration is expected to have some of the good characteristics like configurability and availability of a standard interface for easier integration, compliant to good design practices. IP integration can be easy if complete set of deliverables are provided – RTL, testbenches, synthesis scripts and documentation.

2.3 SoC validation – challenges

To better visualize the validation challenge of an SoC we will refer to the below table where the whole validation cycle is spread across different milestones in 3 or 4 macro stages for the multiple IPs included in the design.

	UART	USB	PS2	I2C	PCI	SDIO	HDMI	JTAG I/F	FLASH I/F	Image sin	Video pro	GPU	MMU	DMA	Ethernet	CAN	Audio Enc	AES/3DES	VITERBI D	FEC	LCD	SENSOR	FM	LTE	ZIGBEE	ANT	3G Mode	GPS	Bluetooth	WiFi
SPECIFICATION	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
RTL DESIGN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
SIMULATION-based VERIFIC	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
FPGA based VERIFICATION	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
HW/SW Co-VERIFICATION	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
DRIVER DEVELOPMENT	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
APPLICATION TESTING	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
IC DESIGN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 2 Possible IP components in a SoC

Today's SoCs designs are highly complex because of the following factors

- High density circuitry as more features are included but with least impact on size.
- Heterogenous components (often from different vendors) are integrated into the same SoC.
- Time to market.
- High speed requirement – routing and timing closure.
- Power grid design – multiple voltage domains
- Complex clock tree – a large number of gated, generated clocks.

2.4 IP Development Life cycle

The main steps in an IP development life cycle are,

1. Requirement analysis and understanding.
2. Design Partitioning.
3. Reset, clock, IO and bus interface requirements understanding.
4. Prototyping platform development.
5. Analysing the Tool and License requirements.
6. System Test plan creation.
7. Verification environment creation.
8. FPGA prototyping and signoff.
9. Low power mode support.
10. ASIC porting.

2.4.1 Requirement analysis and planning.

The input at this stage would be the product specification shared by the end user or it could be a specification released by a standard body. A feasibility study has to be done for the features keeping in mind about the TTM and use cases. Also sometimes it makes sense to have a roadmap illustrating the various milestones and the various features which will go with the associated milestone. This will help to incorporate any issues in the previous versions to the current versions and in fact the solution will be more robust and mature by the final version release. This is widely adopted practice especially after the smart phone revolution we have the smart phone vendors making their new version release almost in a span of less than six months.

2.4.2 Design Partitioning

This is a very critical step after the requirement freeze. The design has to be partitioned into software and hardware. We have to see the computation intensive blocks which will benefit from the parallelism in the hardware. Also time critical blocks or procedures where latency is unacceptable are other candidates to be implemented in hardware. Parallel computations are best suited to be put in hardware. Some of the constraints will come from the protocol itself where protocol dictates the turnaround time for a specific procedure. For example some of the protocols where we have the frame structure the protocol will give the time available for the frame timing which includes many signal processing tasks like error coding, FFT ,filtering and CRC. This will give an idea to take a decision whether the procedure can be jointly implemented in hardware and software or it has to be completely implemented in hardware. Another important factor would be the cost factor. The cost of the processor and the hardware should not overshoot the cost factor for the complete solution. The cost factor would be another decisive parameter to decide on the partitioning. In one of our system we had implemented the turbo decoder using multiple DSP processors. We could reduce the number of DSP processors as well as improve the performance and could get better timing when we moved the turbo decoder to hardware. Other examples are RF front end of any system which includes lots of filters is best put in hardware. If the algorithm to be included has lot of context switching and conditional branching it is best implemented in a processor. Also if you have very less time for prototype development and the performance is not a matter of concern implementing using a processor is the fastest option.

Another very important aspect would be the throughput requirements and power. These aspects should be the key pointers used to define the architecture for the system. Both throughput and power aspect would be driven by the final use case or final ecosystem where the product will be used. If it is going to be sensor like application throughput may not be a towering requirement but it will be governed by power consumption. Especially with plethora of wearable devices nowadays in market power consumption is the governing factor in such cases. Also design partitioning should not cause frequent interactions between the software and hardware. For example consider a case where our device needs to send a fixed length packet with a periodic interval till we get an acknowledgement from the receiver and inform the software (through interrupt) once this is done. Now one of the best approaches for achieving this with minimal hardware and firmware interactions will require the following blocks to be put in the hardware

- have a data FIFO in hardware to hold the fixed data to be sent. Software has to program this only once at the start of the procedure.

- have a timer in the hardware itself to keep track of the timing interval. Once triggered till the procedure is stopped the timer shall be running.
- Let the hardware handle the packetisation/depacketisation and error checks during reception.

2.4.3 Reset, clock, Memory, bus interface requirements analysis.

We have to decide on the type of reset required for the system and the frequency at which the system will be clocked. Also the memory requirements for the system should be analysed. We have to decide if we need multiple clocks for the system. To conform to the protocol timings certain part of the logic or system will have to clock at a higher frequency. Specifying the multi cycle constraints wherever applicable and using proper handshaking in clock domain crossing paths is very important.

Also for the low power modes we might have requirement for an RTC clock. Also depending on the architecture, power and throughput requirements appropriate bus interface has to be chosen between the hardware and software if your solution comprises of both. If the solution is offered as hardware IP it is better to give standard and widely used interface as connectivity option.

2.4.4 Prototyping Platform Development.

Appropriate platform has to be selected or developed for prototyping. The prototyping platform has to be selected based on the final application. Also we should have an initial estimate of the special resources like the DSP blocks and high speed transceivers required along with the approximate gate size to choose the FPGA.Recently there is availability of prototyping platforms which have a SoC FPGA which integrate an ARM core and many peripherals to the FPGA. This kind of platforms is very ideal for applications which require very high performance. This saves the board space since both the processor and FPGA is integrated into one. This also saves a lot of power since the interconnects between the processor and FPGA share the same silicon and hence the data rate between the FPGA and processor could be very high. Also we have to see the memory performance provided by the particular prototyping platform. The memory controller which is required to access the external memory could be shared between the processor and FPGA. But if the application requires very high performance it might be better to choose a system which has dedicated memory controller for the processor and FPGA.Appropriate platform has to be chosen from the available variants which are close to our requirements. Also we have to pay attention to the vendor roadmap for the prototyping platform in terms of their plans to support future revisions of processor core, tools etc. The debug mechanisms available and the tools required for development with the platform are other factors which need key consideration.

It is good to develop our own development platform but we need to budget good amount of time for this activity and have to start this activity well ahead of the IP development. Also there is a risk of certain components becoming obsolete so it is sensible to make the development platform in good numbers if we anticipate several versions or enhancements in the product or specification. Also there are many platforms which have the FPGA with embedded processors inside it.

2.4.5 Analysing the Tool and License requirements.

Depending on the tool and component flow chosen we have to ensure that appropriate development tools and licenses are available.

2.4.6 System Test plan creation and Verification environment creation.

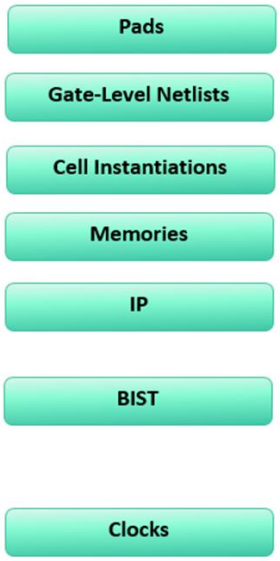
System test plan have to be created for verifying and validating the IP. It has to be decided whether a directed verification or a high level verification language will be used for verification. Also System level validation setup has to be defined for the prototype validation. If the IP consists of hardware and software it is important that the hardware software interface is modelled in the verification environment similar to the actual interface. The low level interface drivers used in the verification environment and the software should be exactly the same. If these are different it will allow many issues to creep through to the validation stage.

2.4.7 FPGA Prototyping

In this stage the design prototype has to be signed off against the test vectors created from the System Test plan created in the beginning from the requirements. Some cases for some of the protocols there would be test specification released by the standard body which can be used for sign off. FPGA prototyping offers a faster and realistic verification strategy. It facilitates the software development instead of waiting for the actual silicon. Even though FPGA prototyping offers a "close to final silicon" solution it still differs in some aspect;

- Implementation is different due to different clock tree, reset distribution, power management, memories and performance.
- Some modules cannot be ported "as is" but has to be replaced / modified with FPGA friendly structures. One of such component we had was the ADC which needed to be replaced by equivalent control logic. This ADC module though not a part of the core IP was still required for a more identical FPGA prototype.
- FPGA prototype represents a limited subset of overall device operational conditions.

One of the limitations with the FPGA prototyping is the limited visibility. The debugging with external logic analysers requires FPGA I/Os which are generally limited in number. We can overcome this limitation by using tools like Identify which gives debug visibility almost similar to RTL simulations.



Top-level pads	Instantiations of SoC pads will not be understood by the FPGA tool flow.
Gate-level netlists	The design is not available in RTL form, but only as a mapped netlist of SoC library cells. These will not be understood by the FPGA tool flow.
SoC cell instantiations	Leaf cells from the SoC library are instantiated into the RTL, for whatever reason, and they will also not be understood by the FPGA tool flow.
SoC memories	Instantiations of SoC memory will not be understood by the FPGA tool flow.
SoC-specific IP	From simple DesignWare macros up to full CPU sub-systems, if the source RTL for the IP is not available then we will need to insert an equivalent.
BIST	Built-in self test (BIST) and other test-related circuitry is mostly inferred during the SoC flow but some is also instantiated directly into the RTL. This is not required for the prototype and may not be understood by the tools.
Gated clocks	As with BIST, clock gating can be inferred by SoC tools but is often written directly into the RTL. This generally overflows the clock resources available in the FPGAs.
Complex generated clocks	As with gated clocks, generated clocks might require simplification or otherwise handling in order to fit into the FPGA.

Figure 3 FPGA Adaptation - Extract from FPMM

2.4.8 Low power mode support

After the normal flow of the prototype is validated we need to focus on the possible power optimizations. Power optimizations include clock gating, frequency scaling and multi voltage domains. Power consumption being a differentiating factor for most of the IPs it is a pre-requisite for most of the IPs to support the various low power modes. So basically the validation of the IP is not limited to the functionality testing but it should validate the various power modes combined with the functionality.

It is better to keep the power intent of your design in a separate power intent file (may be UPF format). There are many advantages of following a UPF flow for your design compared to having the power description being added in RTL itself. Having a UPF flow will ease porting of the IP into designs with different power architectures.

Table 1 Low Power Design- UPF Flow vs Non UPF Flow

Feature	Non-UPF Flow	UPF Flow
RTL content	ISO, level shift	Automatically inserted
Synthesis	create power domain	Creating the power intent is the only work
DFT	Manual insertion	DFT insertion simplified
Equivalence	RET flops	No specific constraints required

2.4.9 ASIC porting

The FPGA design prototype will have to be taken to the SoC flow once prototype testing is completed. The prototype has to be customized for porting to ASIC. The clock gating components in FPGA have to be replaced by the appropriate component available in the target library. Similar is the case with block RAMs used in the design. Also scan chain insertion has to be performed and it has to be ensured that the RTL and scan inserted netlist are logically equivalent. To avoid any maintainability related issues it is always better to use the same database across the FPGA as well as ASIC flow. The usage of appropriate compile time switches for selecting the FPGA or ASIC flow helped to solve this problem. The SoC specific elements will be selected will be included if the ASIC flow is selected and corresponding FPGA equivalents will be included if FPGA flow is chosen.

3 FPGA Prototyping Flow

We have followed the Synopsys tool flow for FPGA prototyping and for the ASIC flow.

Table 2 Synopsys Tool chain

Process	Tool used
Lint	Leda
Simulation/Coverage	VCS (Also Vendor2 simulator)
Logic Equivalence	Formality
FPGA Synthesis	SynplifyPro (Also XST, Quartus II)
ASIC Synthesis	Design Compiler

The below figure illustrates the various steps in FPGA prototyping.

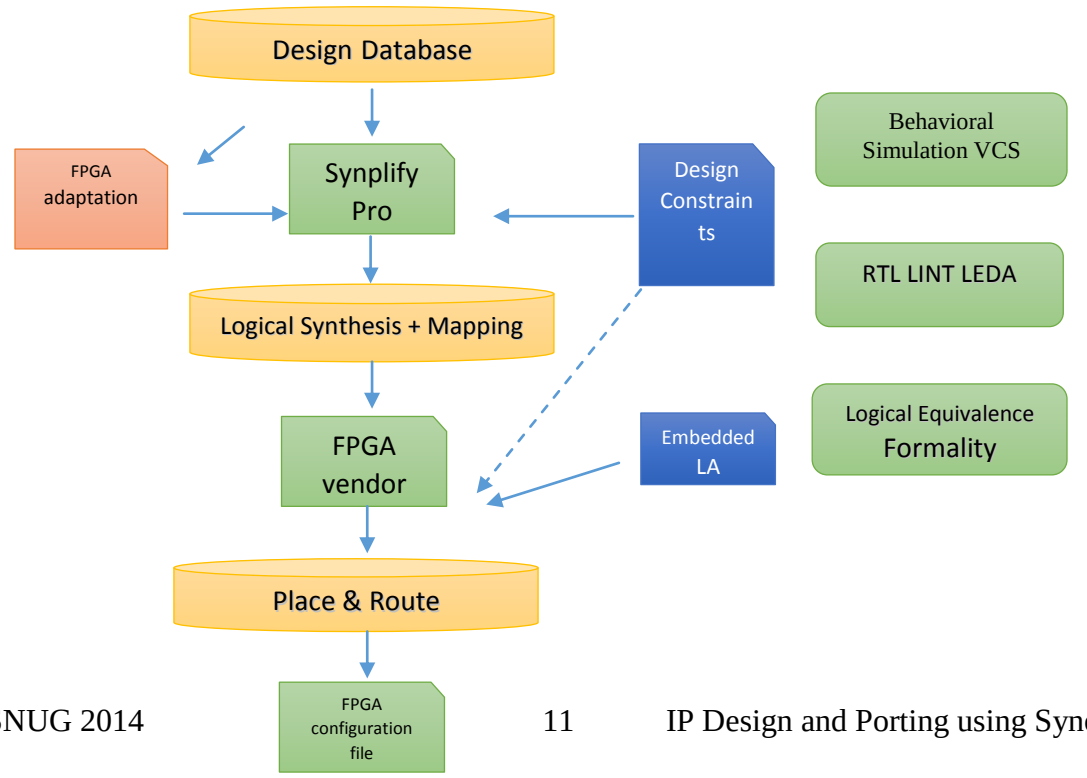


Figure 4 FPGA Prototyping Flow – Reference design

The design database which is in verilog RTL is targeted to ASIC. Hence it will have many ASIC specifc structures like BIST, Test ports, Clock Gates. Many of these are either irrelevant (BIST, Test ports) for FPGA prototype or are not FPGA friendly (Clock Gates). For the mentioned reasons the RTL code targeted to ASIC has to be "adapted" to FPGA before starting the prototyping. There are several steps that we can do at the time of RTL development to ease this process which are discussed in sections *6.5.2 FPGA Synthesis challenges of ASIC design* and *7.1 Good practices that helped ASIC to FPGA porting* . The SynplifyPro project is created with the FPGA adapted RTL along with the design constraints and the FPGA constraints. The verilog quartus mapped file generated in the previous step is fed as input to the quartus file along with the constratints in *.sdc and the Embedded LA information (*SignalTap*). The FPGA bitmap is generated after the P&R step by Quartus tool.

3.1 Low Power Support

Power consumption is an increasing challenge in the new designs. An ASIC might contain several power optimization techniques such as the *clock gating* which is the most widely used technique to save dynamic power, Mutli Vt cells to reduce leakage power. However it is now common to find more drastic power optimization methodologies like *Dynamic Voltage Frequency Scaling* (DVFS), Multiple V_{DD} and Power gating. A brief description of these methods is followed.

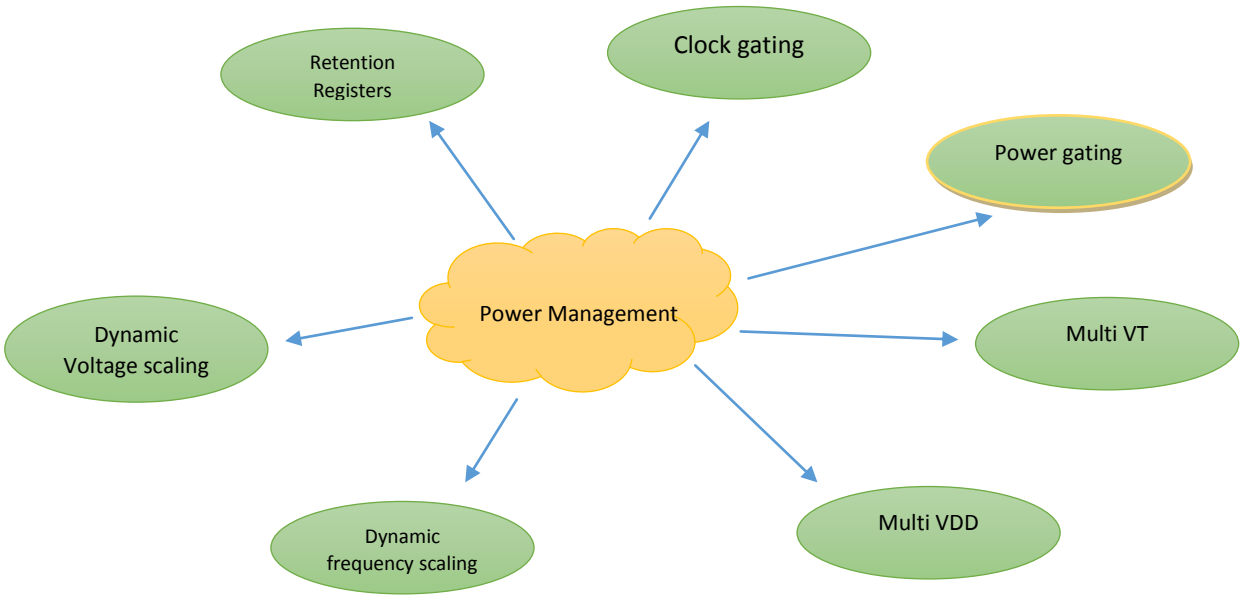


Figure 5 Power Optimization techniques in ASIC

3.1.1 Clock gating

This is a widely used technique for reducing power consumption. The idea is simple – to turn off the clock to block/blocks based on a control signal. This control signal can be selected by the designer (clock gate inserted in RTL) or inferred by the synthesis tool (Automated Clock Gating). Clock Gating is very efficient as the clock network in a chip can consume 30-50% of the total *dynamic power*. It is obvious that gating is more effective (in terms of power saving) if the **Clock Gate cells** are **placed higher up in the clock tree**. Clock gating might look simple but the **effectiveness** of the technique depends on the **choice of the right control signal** for gating. More number of clock gates does not ensure more power saving but it can also lead to increased dynamic power consumption. Hence the challenge with clock gating is that a very deep understanding of the design is required.

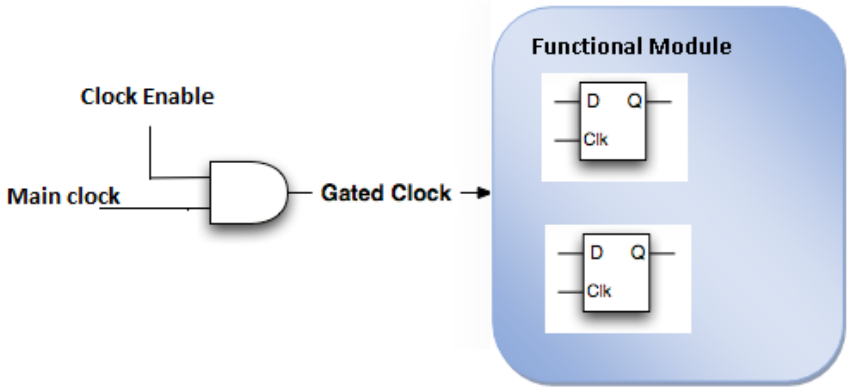


Figure 6 Clock Gating

3.1.2 Dynamic Voltage and Frequency Scaling

The DVFS technique takes advantage of the dependence of dynamic power on the Voltage and Frequency. This is generally used when workload is not CPU bound- In such a case we can operate the CPU at a scaled down voltage/frequency by providing "just-enough" computation power. Choosing the frequency can often be back calculated from the work load (number of CPU clock cycles) and deadline for the task completion. If the voltage is constant, then scaling down the frequency might not be a good idea, instead "race-to-idle" can be used.

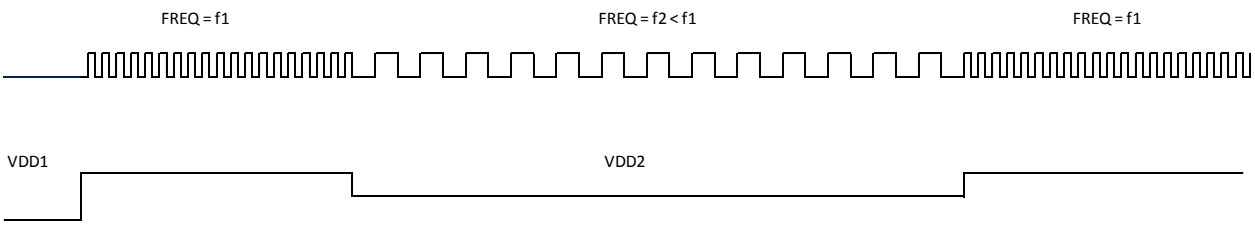


Figure 7 Dynamic Voltage and Frequency Scaling

3.1.3 Power Gating

Gating the power itself is a powerful technique to save power. Again a very good understanding of the operational modes is required for deciding the power shutdown possibility. A few registers might need to be implemented using special retention cells. We have to note that certain register values which can be derived based on other registers need not be part of the retention. A challenge with the power gating is the wakeup time required for the system to be completely operational. If we can define appropriate policies when gating will be enabled only specific values which can have non reset values at the time of gating needs to be part of retention cells. Some systems may not use retention cells but instead might chose to store and restore the registers. Before gating, the registers have to be stored to a non-volatile memory and after the gating is disabled the registers have to be restored with the corresponding register values stored in the non-volatile memory before enabling the power gating.

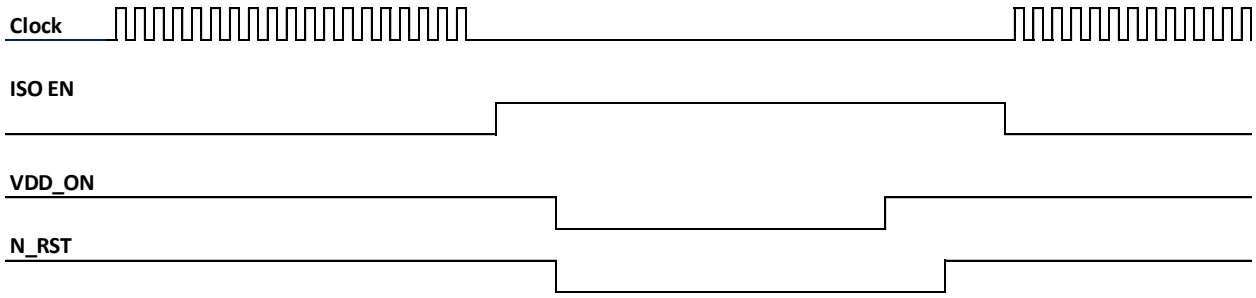


Figure 8 Power down/up sequence

4 Reference Design Overview

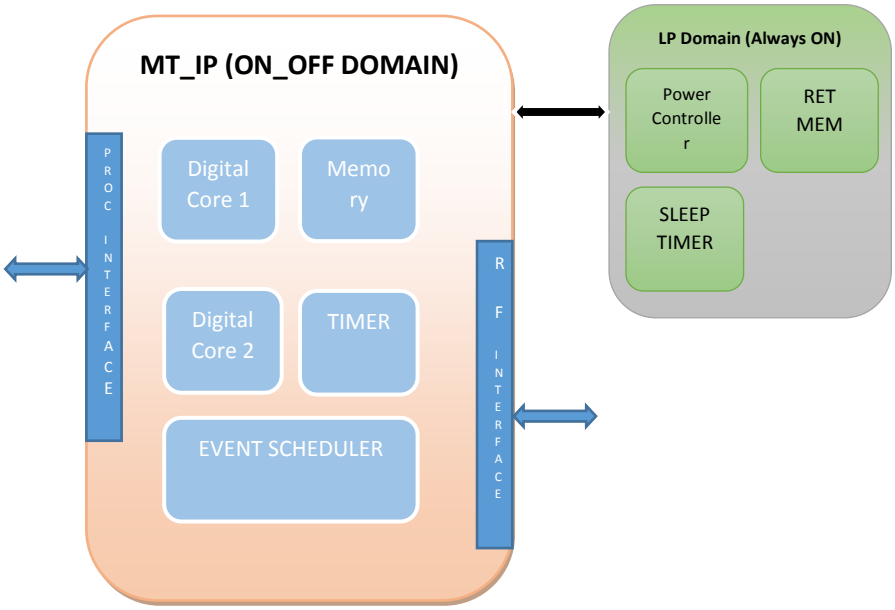


Figure 9 Reference Design Blocks

The figure above shows the reference design we have used for some tool benchmarking and measurements. This has a processor interface (typically a processor with ARM core) and a Radio interface. The design has two power domains – one is the *on_off domain* and the other is the *always_on domain*. The *always_on domain* has a power controller which controls the power down and power up sequence for the SoC. The context restore is done thorough save and restore in a RETENTION RAM placed in *always_on domain.* There are no retention cells in the *on_off domain.*

5 Challenges

We discuss here mainly two challenges here related to the low power design.

- Clock Gating
- Power Gating in FPGA

5.1 Clock Gating

Clock gating is a very common power optimization technique found in ASICs. The clock gating can be done at multiple levels. If different procedures are supported by the IP the clock to unused proedures can be gated OFF. There could be instances when the clock to complete IP can be gated and the IP will be just retaining the states and registers. During this time the entire clock can be gated OFF and the synchronization(timings) can be maintained by the timers in the

always ON domain. The timers in the always ON domain will typically use the RTC clock for this purpose.

However the clock gating structure is not a good choice for FPGA where dedicated low-skew clock lines are used to deliver un-gated clocks to all registers. ASICs will have different clocks like gated clocks, generated clocks, generated gated clocks and clock multiplexers.FPGA synthesis needs to take care of these combinational logic in clock paths and convert these into functionally equivalent FPGA friendly structure.

For the gated clocks, SynplifyPro has the feature "*Gated Clock Conversion*" to automate the conversion of gated structures in the design. We discuss some common clock structures found in ASICs and how SynplifyPro will take care of the same with „GCC" option enabled and also enabling the master clock. SynplifyPro supports scripting in TCL and also GUI mode through SCOPE to enter constratins for the clocks.

All the clock gate structures are not replacable by clock gate conversion. Follwing are the conditions to qualify a clock to be identified as a gated clock.

- The gated clock output can be disabled for some combinations of gating signals.
- Whenever enabled, the gated clock shall be either equal to the base clock or its inverted value.

A common clock gate structure found in ASICs is the glitch-free latch based clock gate strucutre which cannot be converted by the Synthesis tool. Alternate option is to replace this strucutre with an AND gate and to enable clock conversion by the Synthesis tool. The structure shown below in Figure 10 would introduce a combinational gate in the clock path. This will cause the tool not to use the dedicated low skew global lines and will end up in consuming lot of routing resources in FPPGA. This will cause skew between the registers and will cause setup and hold violations in the system. For avoiding the skew in the clock path we should remove the combinational gates from the clock path to the data path.

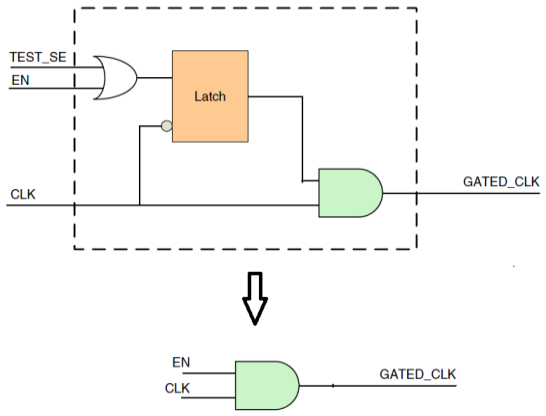


Figure 10 Glitch Free latch based clock gate

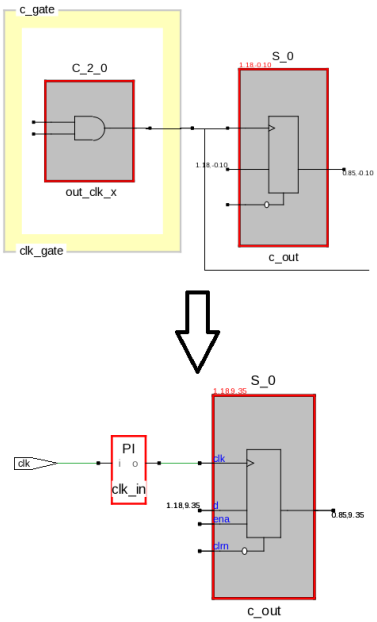


Figure 11 Clock Gate conversion by SynplifyPro

The FPGAs generally have register elements with the clock enables. One possible way is to modify the RTL and write the code using enables so as to make the synthesis tool to infer the clock gates. But the clock gates could be distributed in the design and this could introduce some errors. Another possible way is to make the tool do this job by enabling the gated clock conversion. Once this is done the gating logic is shifted to the clock enables of the registers and this will ensure that both the ungated and gated versions of register elements are clocked by the same source and the gating is performed by controlling the clock enables only.

In Figure 13 there is an illustration of the generated clock being gated. The top portion of the figure shows that the master clock is divided by 2 and the divided clock is further gated and used. In this case the source and destination register is not clocked by the same master clock. We see there is a flop and a gate in the clock path for the destination register. By enabling the gated generated clock conversion option in the SynplifyPro tool we can shift the flop and gate to the clock enable.

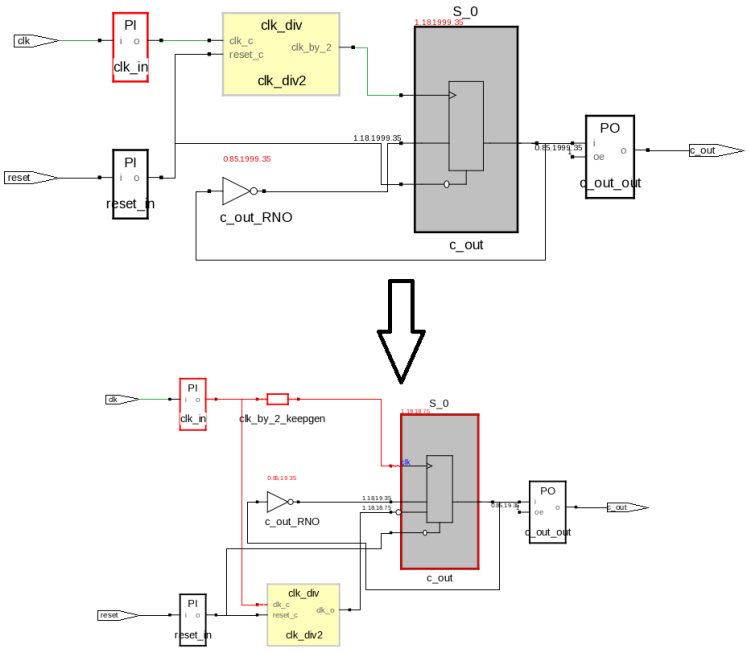


Figure 12 Generated Clock Conversion by SynplifyPro

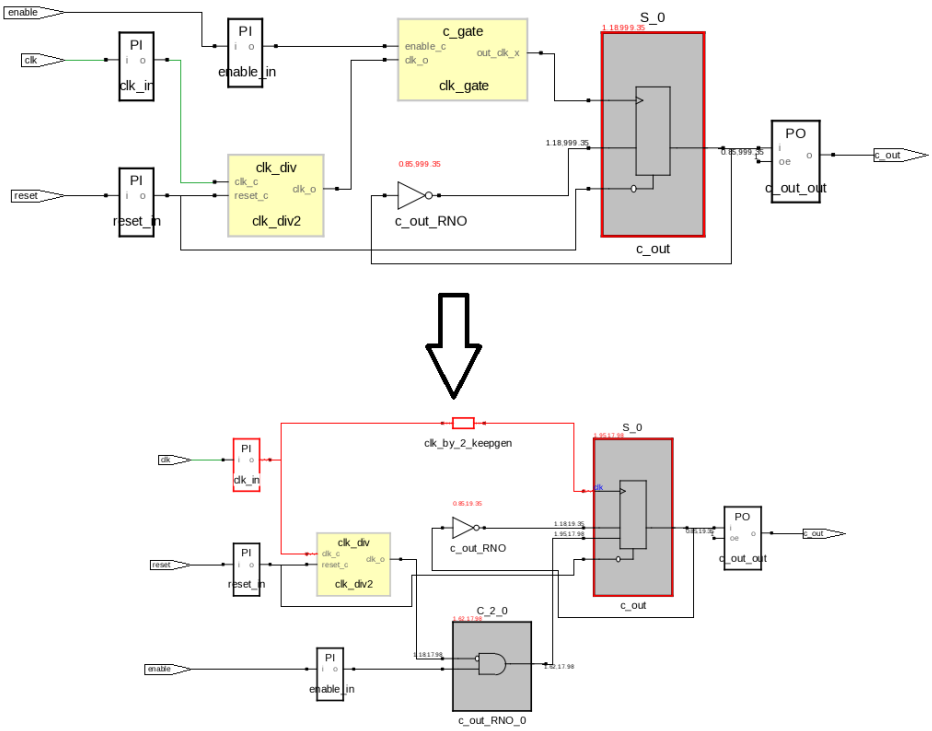


Figure 13 Gated Generated Clock converted by SynplifyPro

5.2 Power Gating in FPGA

Power Gating is an efficient way to reduce power consumption. The key steps in power gating are as mentioned below:

- Identifying the scenarios in the protocol/use case where the power can be gated.
- Describing the power architecture, the different power domains, and the power sequence.
- Estimating the threshold above which power saving can be achieved. Power gating for very brief intervals may not save significant power but might increase the power consumption because of the multiple interactions between the software and hardware which will come into picture for the context store and restore and the wakeup sequence.
- Defining the policies for power save and identifying the sequential elements which need to be stored and restored. Some of the register elements need not be stored and restored because its value can be re-calculated from the other elements which is stored and restored. The retention size is a key factor in power consumption. The retention block is going to be in the always ON domain and going to contribute to the leakage power consumption. If the retention contents are more it can be distributed to multiple RAMs with different supplies. Depending on the retention contents, only the relevant blocks of memory which have valid retention data needs to be supplied with power. The time taken to enter and exit the power save modes needs to be optimized as much as possible since this will maximize the time spent in the low power state and will help in saving power.
- The SoC wakeup and entry sequence need to be decided. If there are multiple power domains the sequence in which the power domains will be shutdown and woken up need to be defined.

The main elements of the "always ON" is the Retention RAM, the timers which use the sleep clock and the power sequence controller. The timers maintain the protocol timings using the low frequency clock. The power sequence controller generates the control signals necessary for controlling the various power switches, clock gates, isolation cells during the entry to the low power state and exit from the power state. Now from the ASIC point of view when one domain is SHUT DOWN the outputs going from this domain to always ON domain inputs will cause illegal voltages to appear in the always ON domain inputs and can excessive crowbar current. Hence these outputs from the power down domain need to be isolated. In FPGA we do not gate power but we need to ensure that the always ON domain does not accept or process the inputs from power down block once the power controller asserts the isolation enable signal which will be used to enable the isolation cells in the case of ASIC. The isolation enables can be used for accepting the inputs from the power down block and once the isolation enable is de-asserted the inputs are not further accepted from the power down block. Similarly the different voltages cannot be modelled in FPGA but in ASIC we have to use the voltage shifters for signals from one domain to another.

Now we cannot emulate the power down condition in FPGA. How do we do that?

The alternate way of emulating the shutdown is to reset the *on_off domain* at the time of power gating. This is done based on the fact that during power gating, all the registers in the *on_off domain* are lost. The control signal (vdd_off) generated by the power controller when in the power down state will be used to control the power switch in the SoC. In FPGA to emulate SHUTDOWN we can give this output to the reset control block and reset the on_off domain.

Now we de-assert the reset once the wakeup sequence starts (de-assertion of vdd_off).The registers to be preserved during power gating are saved to the retention memory in *always_on domain* and restored after power up.

6 Efficient use of Tools

6.1 RTL Lint

Leda is widely used as the RTL quality checker and many times the usage of this tool happens as part of the final checklist execution to make sure that the RTL quality is good. But ideally Leda should be used even before starting the design simulations. This will help to find issues early enough in the design which otherwise would have taken considerable simulation hours to catch.

The truncation issues , multiple drivers issue which typically can happen when integrating modules developed by different persons are best examples which can be fixed in the linting process . Few other important issues which were reported by the linting tool were:

- Convergence in cross over path. There was a combinational logic before the synchronizer in the CDC path which got introduced during the design implementation.
- Absence of double flop synchronizer in one CDC path.

6.2 Simulation in VCS

Simulation offers a perfect debug environment as if offers maximum controllability and observability. Turn around time for validating a fix is less.

At some point where the design size increased considerably, we observed that the simulations in VCS (2013.06-SP1-2) was taking longer time to complete compared to another simulation tool.

With the help of profiling (using +prof option in command line) – different for different tests – will reveal useful information about the simulation of the design. This vcs.prof file has TOP LEVEL VIEW and MODULE VIEW which will help us understand the time spent in each module as well as the different stages of the simulation. Two state (+2state) simulation is another high performance option that can be used where signal values will be restricted to 1 and 0 in simulation. This option has to be enabled during compilation and can be applied to a part or the full design.

The simulation script that we had used had some legacy simulation switches which caused simulation to be slow. Use of the correct switches and also removal of redundant switches is highly recommended for speeding up simulation.

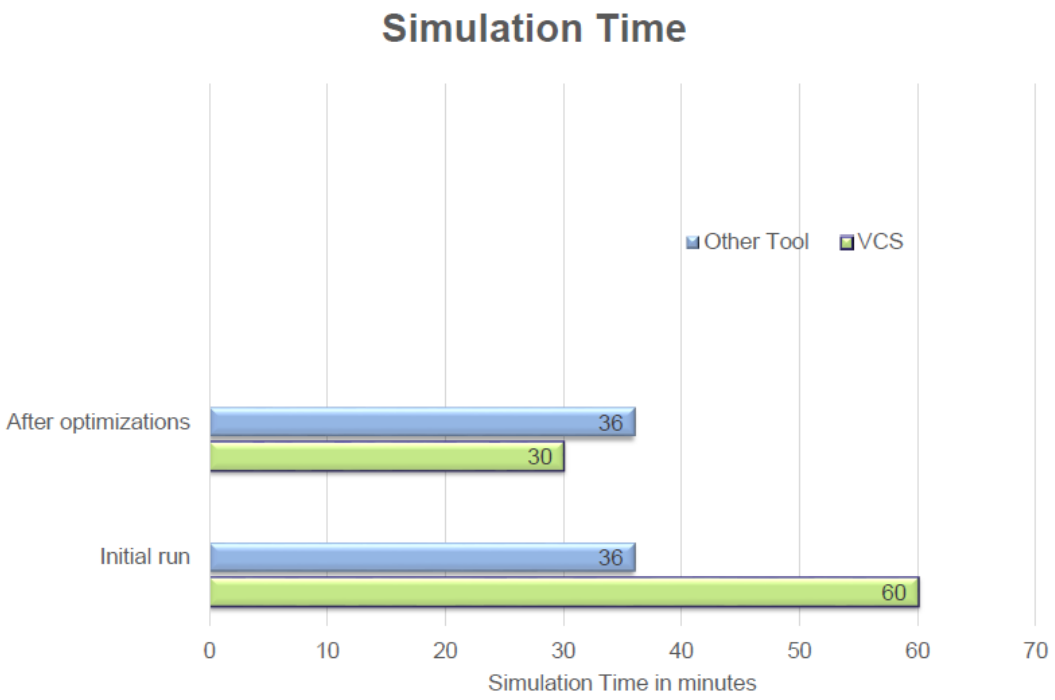


Figure 14 Simulation Benchmarking in VCS

With a very good support from Synopsys technical team we made some changes to our simulation script switches and could get results better/at par compared to the V2 Tool. This was definitely helpful as we had more Synopsys licenses available. The main changes were

- -full64 Enables compilation and simulation in 64-bit mode
- +evalorder option to evaluate the active events when limiting the exposure of race conditions present in the design.
- -parallel Enables parallel compilation and simulation for various applications.
- +vpd[=NCORES] Specifies one or more cores for VPD file dumping. You can change the number of cores at runtime.
- -hsopt improve gate-level and debug simulation speed
- VCS multicore technology enables parallel verification of a design and its verification environment on multicore compute servers.

6.3 Low Power Simulation in VCS with MVSIM

Today's SoC might have advanced low power techniques such as Power Gating, Retention, Low-Vdd Standby, and Dynamic Voltage Scaling (DVS) employ voltage control to enable fine-grained power management. Comprehensive verification of low power designs requires "voltage-level aware" verification of all the power states, but also of the specified transitions and the sequences between power states as the design moves from one operating mode to another.

Using MVSIM along with VCS will allow to run low power simulations of the design. In addition to the RTL/Netlist and the testbench we use for usual functional simulation, we also require a power intent specification of the design in UPF format.

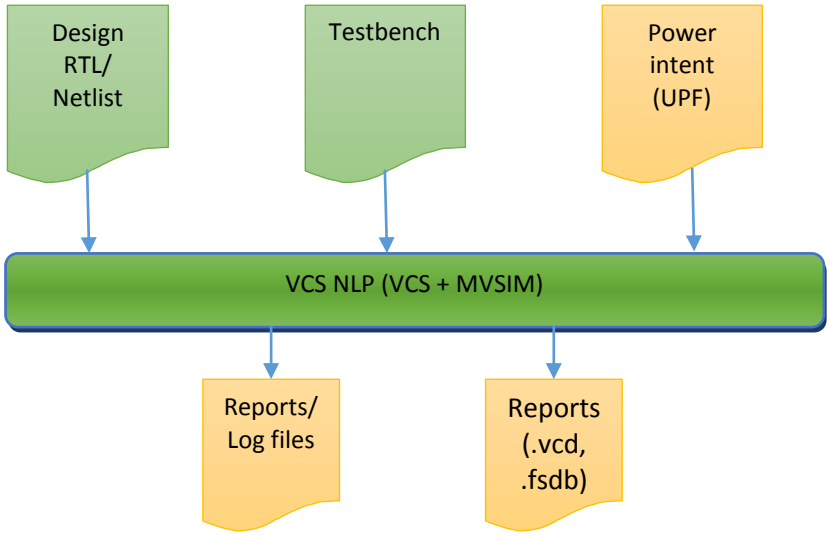


Figure 15 Low Power Simulation (NLP) - VCS and MVSIM

6.4 RTL Coverage analysis

Equally important step in the RTL development is the analysis of the coverage metrics. The VCS simulation has to be run in the coverage mode to get the various coverage metrics. There is also an option of getting the race conditions in the design from the VCS. This could be helpful to find out the possible race conditions in the test bench and design. The DVE tool can be used for analysing the various coverage metrics like line, fsm, toggle, and branch. The coverage analysis is very helpful to identify any redundant or unreachable code which can be removed. Also it helps to identify the areas where tests need to be added.

6.5 RTL Synthesis – Why SynplifyPro ??

One of the key requirements for us was to have the IP ported across different FPGAs. We had used SynplifyPro as the synthesis tool and used the respective FPGA tool for routing and bitmap generation. The choice of SynplifyPro as the synthesis tool is due to many reasons.

- A technology independent Synthesis tool was mandatory for the project. Multi-vendor support available with SynplifyPro is particularly useful for this requirement.
- SynplifyPro offers the flexibility to use the vendor specific mapping tools which gives better optimized results.
- Also most of our customers who wanted to port the IP to their own FPGA platforms were using Synplify Pro/Premier.

For automating Gated clock conversion in SynplifyPro, we have to enable the “GCC” option during synthesis and set the constraint to identify the main clocks in the design. For this the

SynplifyPro tool supports TcL scripting as well as a GUI mode called “SCOPE”. This enables even the new user to easily convert the gated clocks of his design.

6.5.1 Tips to reduce the Synthesis Time

The bitmap generation time starts to have an impact on the development schedule once the design size becomes larger. A few considerations which can be followed to reduce synthesis time are

- The FPGA should be chosen so that including the debug resources the utilization should not be more than 60-70 %.
- The prototype engineer should have an idea of the final performance requirements of the system and should only overconstrain only by 10 %. Initial phase of the development it is better to slightly underconstrain so that very less time is taken for bitmap generation.
- It is better to use incremental synthesis (specify compile points) and place and route option so that bitmap generation for small modifications is fast.
- Also the Fast synthesis option in the Synplify premier can be used if QOR is not a concern.

6.5.2 FPGA Synthesis challenges of ASIC design

Special care to be given to the clock tree implementation in FPGA to ensure clock skew are not present.

- Clock source had to be replaced with PLL
- Clock gate structures to be remapped.
- In the case of latch free clock gating we need to ensure that the gated clock is routed via global routing lines and also need to specify the STA tool.

6.6 Formality

Formality with its easy-to-use, flow-based graphical user interface and auto-setup mode enables users to check if two versions of a design are functionally equivalent. With the increasing size and complexity of today’s designs, coupled with the challenges of power, timing and area Formality is an inevitale tool that could help in addressing the time to market requirements of the design. Formality Ultra can be used to verify the correctness of the ECOs quickly even for big designs.

There could be multiple ways of implementing the same logic. The difference between the two implementations could be in terms of the area taken or in terms of the time taken. It will take considerable time to execute the full test suites after any logic optimization. In the IP roadmap we have various milestones defined. Each milestone would have undergone considerable verification and validation. The next milestone would focus on the area optimization without any functionality change. We have efficiently used the logic equivalence checking process for saving the time required for verification and validation of such optimizations. Also this will be very helpful to validate the various ECO fixes to confirm that the functionality is intact. The formality tool available from Synopsys was used for this purpose.

7 Results and Possible Improvements

We have discussed so far some of the challenges in the FPGA prototyping and also some challenges specific to a low power design. A strong collaboration with SoC team and the prototyping team and an efficient technical support from the EDA tool vendor are the must requirements for the successful porting of design into ASIC. We will also mention some guidelines that will help for easier porting of the RTL to FPGA.

7.1 Good practices that helped ASIC to FPGA porting

It is true that an RTL code targeted to ASIC is not usable as is in FPGA without any modifications.

1. Any ASIC/FPGA design will have custom blocks/ports which have to be kept protected under a macro which can be enabled during compilation.
    - Clock Gate structures (not FPGA friendly)
    - Memories.
    - SCAN ports (only for ASIC)
    - FPGA Debug Probes (only for FPGA).
    - Embedded LA (only for FPGA).
2. The design had a well-defined hierarchy (For example the Dig_Top, Clk_and_Reset_Top)
3. Modularize related logic and to register the outputs of the module. This is absolutely helpful in the case of memory instances. If all memories are under a single module (For example *top_mem*)if is easy to replace the same.
4. Having a synchornous design eases timing analysis. Use of proper constrainsts.
5. Take advantage by using FPGA specific structures (Block RAMs, BUFGCE/Clk_cntrl blocks etc)

7.2 Synthesis results

Table 3 Synthesis Results

Synthesis Run Time	Around 1 hour
P & R Run time	Around 2 hours
Utilization	Nearly 70% on Altera Stratix 3
Timing	Met

7.3 Power Estimation and Actual Silicon measurement

We have some results of power estimate done with PrimePower and the corresponding measurement in Silicon. The measurements has been accurate with a less than 10% error.

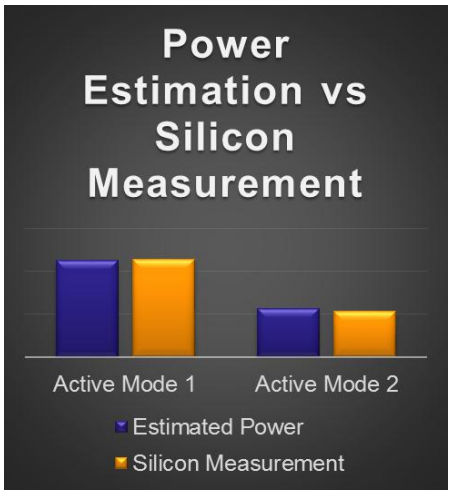


Figure 16 Power Estimate - Prime Power vs Silicon measurement

For accurate power estimation,

- Netlist data provided for analysis must accurately represent the design.
- By annotating parasitics, especially detailed parasitics, very accurate delay and transition times can be calculated. These accurate delays and transition times give more accurate dynamic power calculations.
- CCS Cell Library Models provide better accuracy in estimation.
- Accurate power analysis depends on accurate signal activity. For purely average power analysis, the toggle rage data can produce accurate leakage and average power results. If gate-level simulation activity data is unavailable, the support for RTL level switching or defaults switching activity propagation provides a good estimate of the dynamic power and state-dependent leakage.

7.4 Possible improvements

In many cases the SoC designer will have to integrate various IPs from different IP vendors and hence IP vendor shall ensure some steps to make easier porting for the design into SoC.

- Having good documentation/synthesis scripts will help the prototyping team to identify challenging areas of the design
- Start the design targeting for two technologies (SoC and FPGA)
- It is favourable for the prototyping engineer to be aware of SoC design methodology to some extent
- Follow RTL/Design guidelines strictly

Low Power requirements are becoming challenging in Today's ASICs and it becomes important to support Power gating prototyping in FPGA. The immediate step in this direction shall be the synthesis tool taking care of the automated insertion of power gating related logic by reading the power intent.

8 References

- FPGA-Based Prototyping Methodology Manual.
- Low Power Methodology Manual