International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 11, November-2014

Design Guidelines for FPGA Based Design

Vaibhav Taraate,
Senior Design Engineer RV-VLSI Design Center Bangalore,
Karnataka, India

***Abstract*—Over the past decade FPGAs are primarily used for SOC based design and for ASIC prototyping. The architecture of modern FPGA is complex and consists of up to few mega logic cells or logic blocks. An FPGA consists of finite number of resources and it is essential for a designer to implement the design by estimating the logic requirement that is the resource or device utilization. Modern FPGA architecture consists of sea of array of Logic blocks or logic cells, Block RAMs, Hard and soft IP cores, embedded multiplier, DSP blocks, Processor cores and other glue logic. Modern FPGAs are used as prototyping device for SOC based design and development. The major design constraints are area, speed and power. By using proper design and coding practices the large or medium FPGAs can be used for SOC prototyping. The coding and design guidelines are used to improve the design performance by optimizing design for glitch free behavior. Good design practices always aid in successful design migration between FPGA and ASIC for both prototyping and production. The paper presents most of the efficient coding and design guidelines by using Verilog HDL for FPGA based designs.**

***Keywords*—FPGA, Verilog, ASIC, HDL, Logic Block, STA, FSM, SOC, Lint, CDC, PLL, DLL, IP, LUT, IOB, ,mux, SDF,DSP.**

I. INTRODUCTION

Most of the complex design fails during implementation. The design fails due to timing violations, or due to the violation of area or power constraints. The main reason behind this is violation of design rules and the poor RTL logic design and even negligence in using the proper design guidelines. FPGA is register rich logic and the capacity of FPGA is always estimated in the form of how many Logic blocks it has? While writing an RTL code if designer refers coding and design guidelines then, chances of failure during implementation stage can be minimum. This saves the overall design and development time and also improves the performance, reliability and productivity of design team!

For FPGA based design the information about the FPGA architecture with physical interpretation of the architecture of the design always plays very crucial role during the design and development cycle. While designing by using HDL it is essential to understand how synthesis tool interprets different coding styles? Coding styles used can affect the logic utilization and design performance.

During the design cycle many times it has been observed that by using proper design guidelines the performance of the design can be improved. The use of design guidelines simplifies the static timing analysis and even matching of RTL behavior with gate level netlist becomes very easy. Another important goal for using design guidelines is to improve the overall testability features. The major design guidelines are for use of efficient resources of an FPGA, use of proper IPs and memory cores, and efficient use of low power cells for achieving the lower dynamic power.

It is very essential that Lint tool can be used at the various stages in the front-end FPGA design flow. Lint tools are useful to point the design rule violations at the gate, block or even at the system level.

The paper is organized in the following manner: The section I describes the introduction, section II describes the Verilog coding guidelines, the section III describes the design guidelines for area optimization, the section IV discusses on guidelines for clock, the section V describes the synchronous vs asynchronous design, the section VI describes the guidelines for use of reset, the section VII describes the guidelines for the CDC, the section VIII discusses on the design guidelines for low power design, section IX describes the guidelines for the use of vendor specific IP blocks and finally summary in section X.

IJERT

II. VERILOG CODING GUIDELINES

Guidelines for using Verilog to implement efficient RTL are listed in this section and it is always recommended to use these guidelines during RTL design phase. Among these, few guidelines are mainly described with reference to Verilog Stratified Event Queue [1].

A. Blcoking Vs Non-Blcoking Assignments:

I. It is recommended to use ***blocking assignments*** while modelling the ***combinational design***.

II. It is recommended to use ***non-blocking assignments*** while modelling ***sequential design.***

III. It is recommended to use the ***non-blocking*** assignments while modelling the ***latches.*** While implementing RTL design, it is essential to overcome the potential unintentional latches. Unintentional latches are inferred due to missing ***else*** or due to incomplete ***case*** conditions.

IV. It is recommended to use the ***non-blocking*** assignments while modelling both ***sequential and combinational logic.***

V. It is recommended, ***not to mix the blocking and non-blocking assignments*** in the same always block.

Figure 1 is the hardware inference of Verilog code for blocking assignments. And Figure 2 is the hardware inference of Verilog code for non-blocking assignments.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

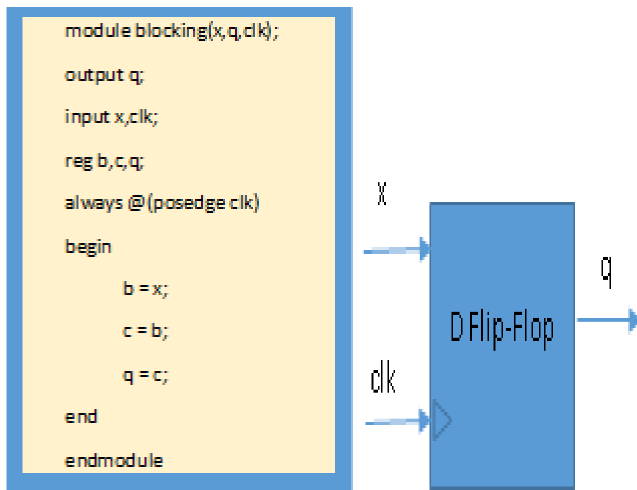


Figure 1: Hardware Inference: Blocking Assignment

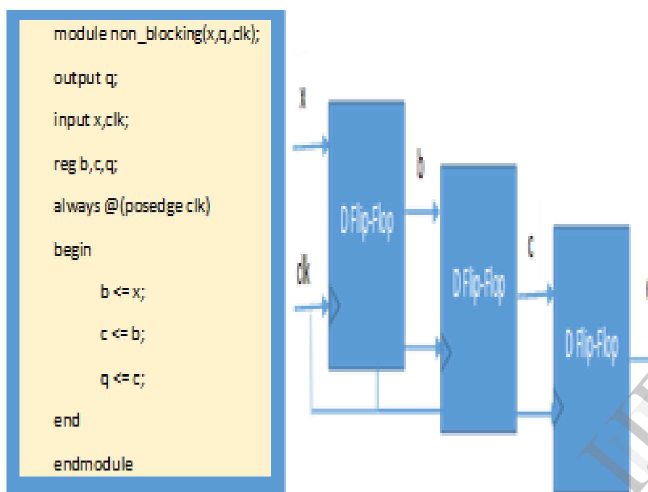


Figure 2: Hardware Inference: Non-Blocking Assignment

From Figure 1 and 2 it is observed that the blocking assignment generates single register as it truncates the other subsequent assignments. But non-blocking assignment generates the pipeline structure like shift register.

B. Priority Vs. Prallel Logic:

I. It is recommended to use ***if-else*** statement for designing priority logic. Priority encoder or priority interrupt control logic can be modelled by using the ***nested if-else*** statements.

Figure 3 is hardware inference of priority logic using nested ***if-else*** statement

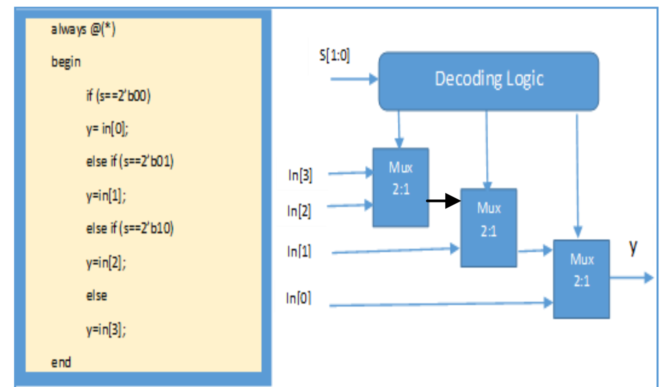


Figure 3: Hardware inference using if-else statement

II. It is recommended to use ***case*** statement for designing parallel logic. Priority logic generates the longer combinational path due to ***nested if-else statements***, so it is always recommended to use ***case*** statement to generate parallel logic. Figure 4 is hardware inference of parallel logic using ***case*** statement

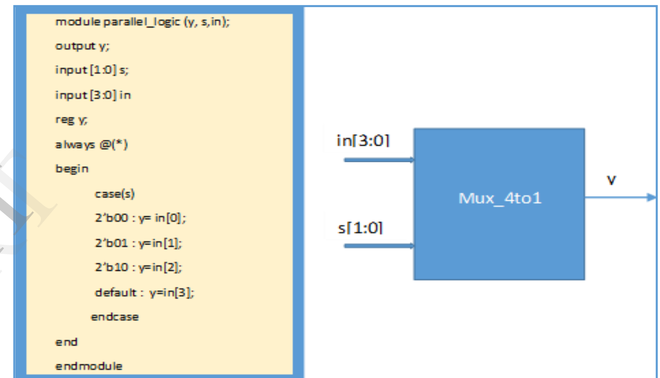


Figure 4: Hardware inference using ***case*** statement

C. FSM Guidelines:

I. Binary encoding techniques are efficient for a design having 16 or fewer states. As number of states increases the next state combinational logic performs slower operation.

II. One-hot encoding technique is very efficient and reliable as compare to the binary encoding due to glitch free behavior. One hot encoding requires low density nest state logic and useful in design of larger FSM blocks. But the main drawback of one-hot encoding is; it uses more registers!

III. While designing FSM, designer need to take care of following key points

a. Don't leave any undefined states. Initialize the unused states to reset value or use the default statements.

b. Don't implement the FSM with combination of registers and latches. Avoid the unintentional latches in the FSM design to improve the reliability.

c. Model the FSM blocks by using ***case*** statements to infer the parallel logic.

IJERT

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

d. Separate the next state, output combinational logic and state register logic in different always blocks to improve the speed of FSM and for better synthesis results.
e. Register FSM output as it preserves the hierarchy.
f. Use the look ahead mealy machines for better design performance.

D. Combinational Design and combinational loops:

I. It is recommended to use ***continuous assignment*** statement for design of ***combinational logic.***

II. While designing the combinational logic it is essential to avoid the combinational loops. Combinational loop causes instability and unreliability in digital designs as it violates the synchronous design concepts due to infinite looping.
The combinational loop generates the oscillatory output and the period of the oscillatory output signal is mainly dependent on the delay introduced by combinational logic in the feedback path.
For example as shown in Figure 5 the procedural block infers the combinational loop during synthesis. It is always treated as the undesirable behavior due to oscillatory nature of output. By using the proper Lint tool in the FPGA design flow at various stages the combinational loops can be detected [2],[3],[4] and can be avoided.

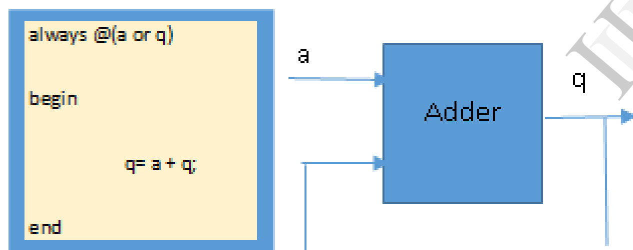


Figure 5: Combinational Loop

III. Use the signal grouping to improve the performance of FPGA based design. For example if the expression q= (x+y+z+w) can be represented as shown in the Figure 6 and the Figure 7 is hardware inference with grouping by using expression q= (x+y) + (z+w). Due to grouping the timing performance of design is improved.

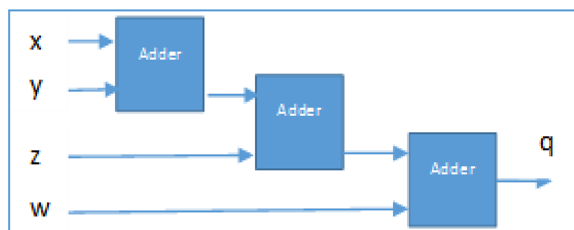


Figure 6: The hardware inference without grouping

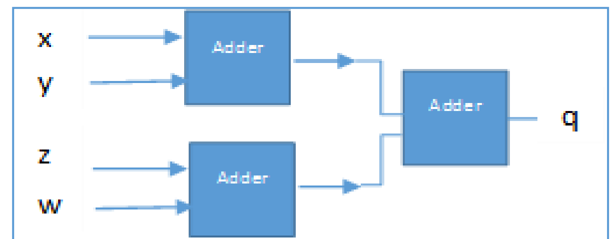


Figure 7: The hardware inference with grouping

E. Assignments:

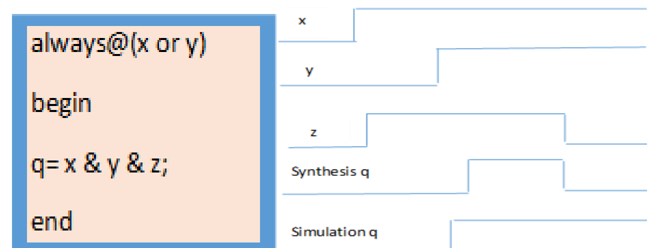
I. It is recommended ***not to make the assignments to same variable from multiple always block.*** It gives error as multiple drivers to the same net or wire.

II. It is recommended ***not to make assignments with #0 delay*** [1].

F. Simulation and Synthesis mismatch:

Most of the synthesis tools ignores the sensitivity list of combinational procedural block but simulator executes the procedural block, only when there is event on one of the signal in the sensitivity list parameters. Due to incomplete sensitivity list it creates the simulation synthesis mismatch. Consider the example shown in Figure 8. The synthesizer ignore the sensitivity list and generates the AND logic but due to incomplete sensitivity list the simulator generates the output which is different from required output.
The simulation synthesis mismatch is shown in the Figure 8 for the Verilog combinational block. It is recommended to use all the required signals or inputs as sensitivity list parameters.

IJERT



.Figure 8: Simulation Synthesis mismatch

G. Post synthesis Verification:

It is highly recommended to perform the post synthesis verification for the FPGA based design. Post synthesis verification with the SDF assures the correct behavior of the gate level netlist. There should not be mismatch between the functional verification of the design and the post synthesis verification!

III. GUIDELINES FOR AREA OPTIMIZATION

FPGAs have finite resources so it is recommended to follow the design guidelines to optimize the area. The area optimization techniques are: Resource Sharing, Logic Duplication. [Note: Many times it has been observed that, logic duplication can even increase area and the use of logic duplication technique is dependent on the design scenarios!].

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

A. Resource Sharing:

Always it is observed that, adders consumes more area as compare to multiplexers. The resource sharing is powerful technique to share the common resources to minimize the area. It is essential for the FPGA designer to consider resource sharing of arithmatic operators used in the same hierarchy.

The example shown in Figure 9 is hardware inference of Verilog code without resource sharing. Here the operation is addition (+) and it uses two adders and single multiplexer (mux). Figure 10 is hardware inference of Verilog code by using proper resource sharing. Due to use of only one adder and more number of mux the overall area is optimized

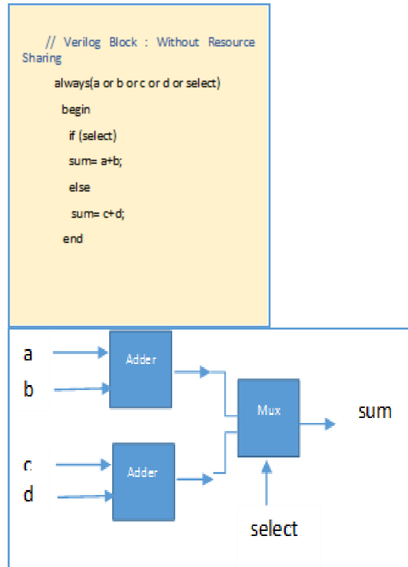


Figure 9: Hardware Inference: Without Resource Sharing

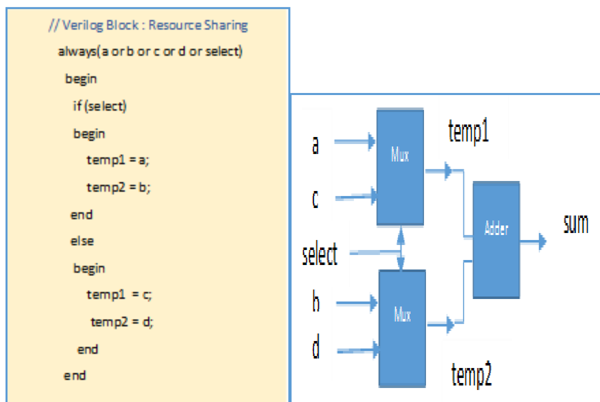


Figure 10: Hardware Inference: With Resource Sharing

Most of the time during design cycle it has been observed that the resource sharing is one of the powerful area minimization technique. But it is recommanded that not to share resources from different modules or from different hierarchy. Resources can be shared from the same module or from the same hierarchies.

B. Logic Duplication:

Logic Duplication is the powerful technique to reduce the net delay by enabling the placement tool to place the replicated logic in various areas of die [2]. The major drawback of this technique is, it increases the area of the design while replicating the register or sequential logic.

On the other hand, as per as area minimization is concern, logic duplication can act as very efficient tool but depends on the design specific scenarios! Consider example of implementing 8:256 decoder using single ***case*** statement. If FPGA architecture has logic block with two, 4 input LUTs and output generation LUT as shown in the Figure 11 [3] then to implement the single output it uses 3 LUTs. So for 256 bit output 768 LUTs are utilized. By splitting case statement to implement two 4:16 decoders, logic duplication can be achieved. By using logic duplication, if two 4:16 decoders are used with 256 AND gate array then the overall device utilization is just 288 LUTs for implementation of 8:256 decoder and it reduces the device utilization by around 480 LUTs. That is very huge reduction in the overall area. For the 8:256 decoder the logic duplication is accomplished by using the 4 input LUTs and 2 input LUTs; the structure of logic block is shown in the Figure 11 [3].

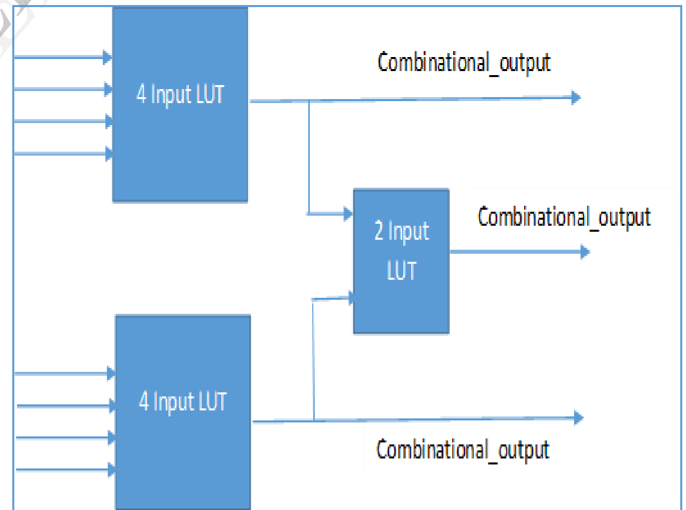


Figure 11: FPGA Logic Block used for Logic Duplication

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

IV. GUIDELINES FOR CLOCK

The performance and reliability of an FPGA based design is based upon the clocking schemes. For the FPGA based design and implementation it is recommended that:

a. *Use single global clock.*
b. *Avoid use of Gated clocks.*
c. *Avoid mixed use of positive and negative edge triggered flip-flops*
d. *Avoid use of internally generated clock signals.*
e. *Avoid ripple counters and asynchronous clock division*

It is recommended by most of the FPGA vendors that, do not use the internal generated clocks as it causes the functional and timing issues in the design. If internal generated clocks are required in the design then use DLL [3] or PLL [2] to generate the clocks.

The internal generated clocks by using combinational logic are prone to glitches and it create the functionality issues in the design. Due to the combinational delays it creates the timing issues in the FPGA designs.

The major problem for using the internal generated clocks is the issue during synthesis and timing analysis.

Xilinx [3] provides the library component global clock buffers BUFGCTL [3] and BUFGMUX [3] to generate internal clocks.

To avoid glitches it is recommended to register the output of the internal generated clocks. It is recommended to use the clock generation logic shown in the Figure 12.

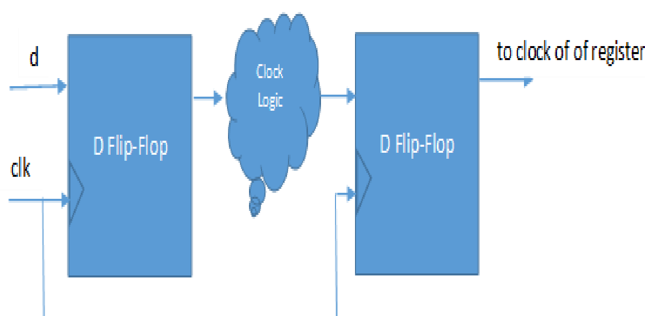


Figure 12: Clock Generation Logic

For low power designs it is essential to use the clock gating but it is prone to glitches. So it is recommended to use the clock gating cells for low power FPGA based design.

It is recommended not to use the asynchronous pulse generator circuit. Figure 13 represents the asynchronous way of pulse generation. This technique should be avoided as it is prone to glitches and very difficult to synthesize and place and route. Depending on the pulse width requirement replace the inverter shown in Figure 13; by chain of odd number of inverters.

Figure 13: Asynchronous pulse generator

Figure 14 represents the recommended pulse generator where the pulse width is dependent on the clock period. It is recommended to use two level synchronizer at the input of pulse generator to avoid the metastability issues.

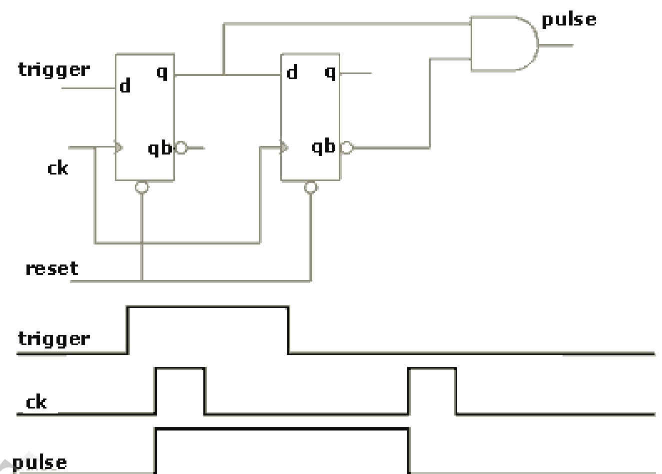


Figure 14: Synchronous Pulse Generator

IJERT

V. SYNCHRONOUS VS ASYNCHROUNOUS DESIGNS

In synchronous design the data input is sampled on every active edge of clock and clock signal controls the activities of inputs and outputs. Figure 15 represents the synchronous design where the combinational logic (CL) drives the data to the input of flip-flop. For the proper operation of the design it is essential that the data input should be stable for at least setup time of register and it should be stable for at least hold time of register. The propagation delay of combinational logic limits the operating frequency of the design. To meet the timing requirement it is essential to have synchronous relationship of all inputs and combinational inputs with the clock signal of the flip-flop.

Use the pipelining feature to improve the performance of synchronous design. As FPGA is register rich logic pipelining is used for improvement of the speed of the design at the cost of latency.

On the other hand an asynchronous design doesn't have common clock (Example Ripple counters) and are prone to glitches or spikes. It is very difficult to model the timing of asynchronous design by using timing constraints. Many times an asynchronous design generates the glitches or short time duration pulses shorter than the clock period. If the glitches are passed through the combinational logic then the output leads to an incorrect value. Figure 16 describes an asynchronous logic prone to glitches.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

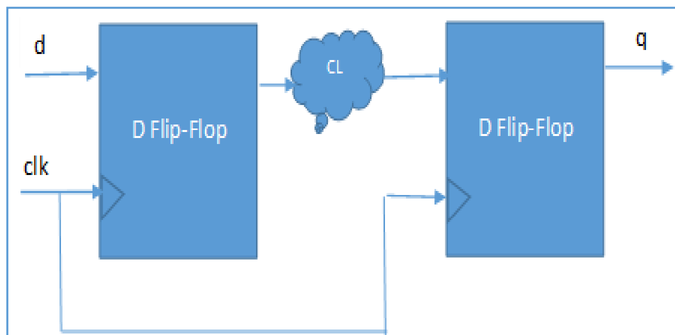


Figure 15: Synchronous Logic

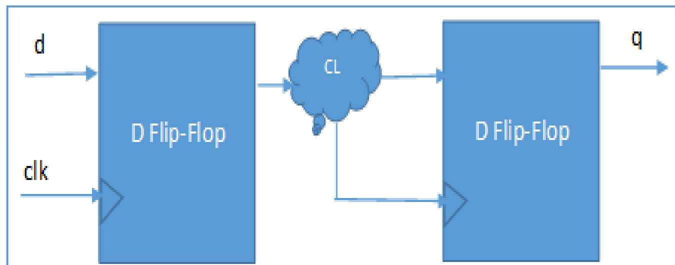


Figure 16: Asynchronous Logic

Many times it has been observed that an asynchronous logic reduces the device resources but prone to hazards. So it is recommended to use the synchronous logic while implementing the sequential design. Synchronous logic always makes STA easy [4]!

VI. Guidelines for use of reset

Resets are classified as synchronous and asynchronous resets. Asynchronous resets are easy to implement as they don't depend on the clock. But STA becomes difficult and complex while using asynchronous resets. At the same time automatic insertion of the test structure is difficult.
On the other hand synchronous resets are difficult to implement as it requires more resources and they are dependent on the clock. Synchronous resets slowdowns the design performance. It is recommended that FPGA designer should avoid internally generated conditional resets [2], [3].

It has been observed during FPGA based designs that, reset deasserted circuit is required while using asynchronous reset. If reset signal is deasserted and if does not pass the setup and hold timing check then flip-flop goes into metastable state and it can lead to potential functional issues in the design [5].

It is recommended to use the synchronized asynchronous resets. That is asynchronously asserted and synchronously deasserted. Figure 17 is the recommended representation of asynchronous active low reset (reset_n) passing through the two level synchronizer.

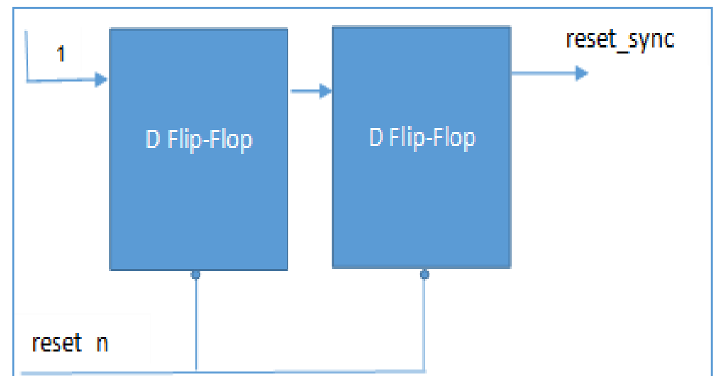


Figure 17: Reset Generation Logic

For very large density or complex FPGA based designs with multiple hierarchies it is essential to use the Linting tool which can provide proper information about the reset and clock trees [2], [3].

VII. Gudelines for CDC

It is impossible to verify the Clock Domain Crossing (CDC) by using functional verification and even it is impossible to verify CDC by using timing analysis tool due to asynchronous nature of clock path. The major problem encountered in CDC is functionality failure due to metastability.
To avoid a metastability it is recommended to use the dual or three stage synchronizers while transferring signals from one clock domain to another.
Linting tools are used to ensure the use of synchronizer chain on the clock domain crossing paths. Use two or three level synchronizer shown in Figure 18 to transfer the signals from one clock domain to another. This will avoid metastability in the design.

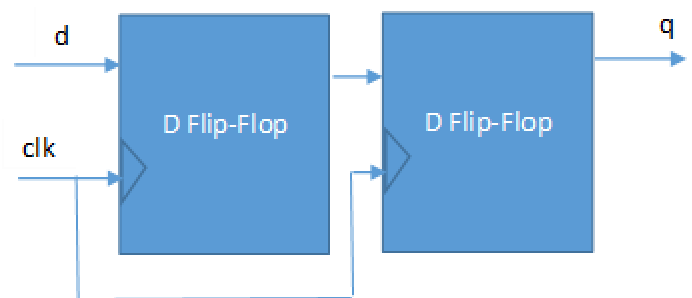


Figure 18: Two Level Synchronizer

VIII. Guidelines for Low power design

Reducing the power for many application is very critical and due to complexity of designs only use of power efficient FPGA devices or architecture is not sufficient. It is essential for designer to understand the features of EDA tools to optimize the dynamic power. The recommendation by many FPGA vendors is to reduce the switching activity in the sequential logic and clock routing [2], [3]. For the low power design it is recommended to use the gated clocks or the low power clock gating cells. Dynamic power of a cell is

IJERT

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

dependent on voltage, load capacitance and on clock frequency. Due to switching at the clock input it has been observed that the dynamic power increases. So to reduce dynamic power it is recommended to use clock gating cells. Figure 19 shows the clock gating cell.

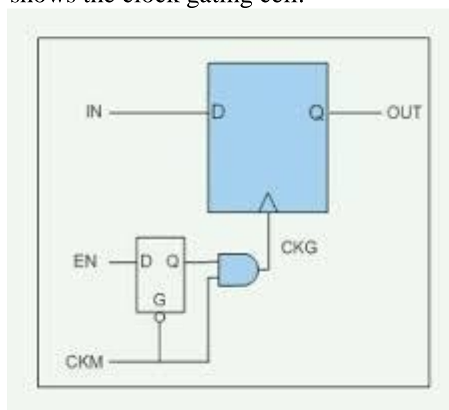


Figure 19: Recommanded Clock gating

IX. GUIDELINES FOR USE OF VENDOR SPECIFIC IP BLOCKS

It it always recommended by the FPGA vendor to have the brief and detail understanding of the FPGA device and the architecture of FPGA device.

It is recommended to use the vendor specific design and coding guidelines to improve the performance of design. It is highly recommended to encrypt the IP by using proper security standards.

During synthesis phase it is recommended to infer the micro-functions such as multipliers, shift-registers, memories and DSP blocks to ensure the optimal results [2],[3].

For the better performance it is recommended to use the proper timing constraints and analyze the timing constraints by using the timing analyzer [3]. It is even recommended to use the proper place and route effort level while implementing the design [3]. The place and route effort level allows the EDA tool to use the proper algorithm to improve the design performance and even it improves the design placement. It is also recommended to use the proper IOB resources and proper speed grade during design implementation stage [3].

While using the synchronous interface it is recommended to use the single clock synchronous RAM (read and write in the same clock domain) and while using asynchronous interfaces use the dual port RAM [2].

IJERT

X. SUMMARY

FPGA design engineer should follow the design and coding guidelines provided by the FPGA vendor for better reliability and performance of the design. It is better to ensure for the efficient, reliable, reusable and readable RTL design. Designer need to ensure about the proper design functionality to lead to the correct simulation and synthesis behavior. It is essential for the designer to ensure for the achieved coverage for the design. Designer need to ensure for the use of proper clocking mechanism, proper clock gating cells for the better design performance. Finally designer need to ensure for the optimized design performance without violation of any of the design constraints! It is always recommended to use the linting tool in the FPGA design flow at various stages.

ACKNOWLEDGMENT

The author would like to express sincere gratitude to design and research team of RV VLSI Design Center Bangalore, Karnataka (India) for frequent technical discussions on emerging and new technology.

REFERENCES

[1] IEEE standard http://standards.ieee.org/getieee/1800/download/1800-2012.pdf www.ieee.org

[2] Altera "Quartus II Handbook", www.altera.com

[3] Xilinx ISE simulation and synthesis guide www.xilinx.com/support/documentation/sw_manuals/xilinx14.../sim.pdf www.xilinx.com

[4] Wayne Wolf. 2005, "FPGA Based System Design", Prentice Hall

[5] Altera Quartus II documentation " www.**altera**.com/literature/hb/qts/**quartusii**_handbook.pdf" www.altera.com

ABOUT AUTHOR

VAIBBHAV TARAATE is senior design engineer at RV-VLSI Design Center Bangalore, Karanataka (India). He received his M.Tech. in Aerospace Control and Guidance from Indian Institute of Technology (IIT) Bombay(Powai) and B.E. in Electronics from Shivaji University. He has working experience of around 14 years in the field of FPGA based designs and semi custom ASIC designs. His area of interest includes SOC based designs, Parallel Processing and Computing, Low power ASIC designs and Configurable Computing Networks.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)