Synthesis of the Decoder Design Using Design Compiler

Deepali Chaudhary, Dr. C. S. Lamba

Abstract—Synthesis is an automatic method of converting a higher level of abstraction to a lower level of abstraction. In other words, the synthesis process converts Register Transfer Level (RTL) descriptions to gate-level netlists. A Register Transfer-Level description is a style that specifies a particular design in terms of registers and combinational logic in between. A good coding style would help the synthesis tool generate a design with minimal area and maximum performance. These gate-level netlists can be optimized for area, speed, testability, etc. The most commonly used synthesis tool in the ASIC industry is Synopsys Design Compiler. Design constraints and other synthesis options are given as commands or default settings to the tool.

Index Terms—Decoder, Compiler, RTL, VHDL.

I. INTRODUCTION

There are eight different types of objects categorized by Design Compiler.

Design: It corresponds to the circuit description that performs some logical function. The design may be stand-alone or may include other sub-designs. Although sub-design may be part of the design, it is treated as another design by the Synopsys.

Cell: It is the instantiated name of the sub-design in the design. In Synopsys terminology, there is no differentiation between the cell and instance; both are treated as cell.

Reference: This is the definition of the original design to which the cell or instance refers. For e.g., a leaf cell in the netlist must be referenced from the link library, which contains the functional description of the cell. Similarly, an instantiated sub design must be referenced in the design, which contains functional description of the instantiated sub-design.

Ports: These are the primary inputs, outputs or IO's of the design.

Pin: It corresponds to the inputs, outputs or IO's of the cells in the design.

Net: These are the signal names, i.e., the wires that hook up the design together by connecting ports to pins and/or pins to each other.

Clock: The port or pin that is identified as a clock source. The identification may be internal to the library or it may be done using dc_shell commands.

Library: Corresponds to the collection of technology specific cells that the design is targeting for synthesis, or linking for reference.

The top-down Design flow of Synthesis is shown below:

II. GENERAL GUIDELINES FOR GOOD PERFORMANCE SYNTHESIS

Following are given some guidelines which if followed might improve the performance of the synthesized logic, and produce a cleaner design that is suited for automating the synthesis process. Clock logic including clock gating and reset generation should be kept in one block to be synthesized once and not touched again. This helps in a clean specification of the clock constraints. Another advantage is that the modules that are being driven by the clock logic can be constrained using the ideal clock specifications.

No glue logic at the top: The top block is used only for connecting modules together. It should not contain any combinational glue logic. This removes the time consuming top-level compile, which can now be simply stitched together without undergoing additional synthesis.

Manuscript received July 22, 2014

Deepali Chaudhary, M.Tech Scholar, Department of Information Technology, Rajasthan College of Engineering for Women, Jaipur, Rajasthan

Dr. C.S. Lamba, Prof., Department of Information Technology, Rajasthan College of Engineering for Women, Jaipur, Rajasthan
Module name should be same as the file name and one should avoid describing more than one module or entity in a single file. This avoids any confusion while compiling the files and during the synthesis.

While coding finite state machines, the state names should be described using the enumerated types. The combinational logic for computing the next state should be in its own process, separate from the state registers. Implement the next-state combinational logic with a case statement. This helps in optimizing the logic much better and results in a cleaner design.

Incomplete sensitivity lists must be avoided as this might result in simulation mismatches between the source RTL and the synthesized logic. Memory elements, latches and flip-flops: A latch is inferred when an incomplete if statement with a missing else part is specified. A flip-flop or a register as it is referred to, is inferred when an edge sensitive statement is specified in the process body. A latch is more troublesome than a flip-flop or a register as it makes static timing analysis on designs containing latches. So designers try to avoid latches and prefer flip-flops more to latches.

Multiplexer Inference: A case statement is used for implementing multiplexers. To prevent latch inferences in case statements the default part of the case statement should always be specified. On the other hand an if statement is used for writing priority encoders. Multiple if statements with multiple branches result in the creation of a priority encoder structure.

The main importance of synthesizing the design is to implement it on hardware by knowing its area, delay, power consumption and resources. The basic RTL model is written in VHDL and is being simulated using test bench also written in vhdl. The synthesis of the RTL model is performed by running scripts either in tcl or dc-shell. The script are run and various files are generated like synthesized netlist either in vhdl or verilog, timing analysis information in sdf (standard delay format), area and timing report etc. After synthesis vhdl or verilog, timing analysis information in sdf (standard level netlist the sdf file should be added to ensure proper file generated by the synthesis. While simulating the gate level the sdf file is being simulated to generate vcd file while the verilog netlist is used as an input to the place and route step. The sdf file includes delay information for the simulation of the design.

IV. POST SYNTHESIS GATE LEVEL SIMULATION

The test benches used for RTL model validation can be reused with possibly some modifications for the VHDL gate level netlist. These VHDL models follow the vital modeling standard to ensure proper back-annotation of delays of the sdf file generated by the synthesis. While simulating the gate level netlist the sdf file should be added to ensure proper back-annotation delays.

V. CONCLUSION AND FUTURE WORK

The performance analysis and hardware implementation of the Non Binary LDPC codes have been done and it has been concluded that the codes perform better for medium codes as compared to long codes and is thus useful for short and medium packet transmission. If we increase the packet size the number of check node and bit node increases, again the decoding complexity increases and large number of hardware as well as memory is required. The advantage of using non-binary LDPC codes over Galois field is that the equivalent binary weight of parity check matrix is increased, while the number of short cycles may remain low. It can also outperform Reed Solomon codes even for burst error channels. The Hardware implementation of the decoding algorithm will be performed on the parity check matrix [H]. The future work can be hardware implementation of Non Binary LDPC codes using FFT method devised by Bernault, Declercq and Fossorier which reduces the number of operations and hence decoding fast.

The implementation of the same decoder by semi parallel architecture to achieve the advantage of both memory efficiency as well as better throughput. Extending the same project to a higher dimension H matrix to analyze the performance of Non Binary LDPC codes for long codes. The code designed using this method has large girth and shows similar performance in high signal to noise ratio(SNR) values and better performance in low SNR values than other codes. These codes have a low error floor. We simulated the Parallel decoder for various size codes and BER results are shown showing that this works well in designing block type LDPC codes.

REFERENCES

1959.
