Chapter 6
Conclusion and future directions

There has been a tremendous increase in the complexity of designs in the VLSI industry over the past few years. On one side, the constant and ever-changing demands for new gadgets and appliances such as smartphones, cameras, connected vehicles, and infotainment are creating a need to shrink the design cycle time for SoCs. On the other side, FPGAs are getting popular in the VLSI industry owing to their fast speeds of operation. At the same time, more are more architects and designers in the industry are using HLS-based design flows owing to a higher abstraction level and the continued verification that they offer during the design flow. Two major optimization approaches for FPGA-based implementations of digital designs using HLS flows are major contributions of this work. They are discussed in Sections 6.1.

1) Application of optimization schemes offered by existing HLS tools (Section 6.1.1):

We have applied the same schemes using HLS directives on multiple application designs such as bandpass DSP filter, QPSK modulator, MIPS processor core design, AES algorithm, and YOLO v2 deep learning algorithm as indicated in Section 6.1.1.1 to Section 6.1.1.5. Depending on the type of application and its architecture, a particular set of directives help in area, speed or power optimization. As an example, loop unroll can help optimize the speed in an application architected using design loops while it may not have any effect on other applications.

2) Application-specific bit width for intermediate data nodes (Section 6.1.2):

This is a novel HLS optimization technique that supports to all application designs and HLS tools but this study has verified it for two HLS tools namely MATLAB HDL Coder and Vivado HLS. Using an automated method to calculate the bit width for all internal, input, and output nodes, it helps to remove pessimism in bit width constraints for downstream synthesis flows.

Further, we have proved the efficacy of this methodology by applying the same on designs from multiple application areas (Sections 6.1.2.1–6.1.2.5). As explained in each of the sections, unlike HLS directives, this design methodology has helped to achieve optimal synthesis results for all applications, irrespective of the HLS platform used. In section 6.1.3, we present an extension of the work where we use the learnings from HLS optimization to a test application. We designed a RADAR signal processor and verified the same using an HLS assisted framework. The final implementation results were optimal as compared to literature and in terms of functionality, the results were compared against a taped out SoC model for accurateness. Section 6.2 proposes

some possibilities for future work in the same area.

6.1 Major contributions of the work

We discuss the 2 major contributions of this thesis in this section. One of them being the optimization of designs from diverse application areas using HLS. The other contribution of the thesis is a novel and generic HLS design method based on application specific bit widths.

6.1.1 Optimization of multiple application designs using HLS directives

We discuss the application of HLS directives for multiple application designs to optimize their FPGA implementations. Next 5 subsections highlight the different applications that were designed along with the HLS tool and directives used to optimize them. The directives are very specific to the tool and application being designed.

6.1.1.1 BandPass DSP filter design, optimization, and implementation using HLS

The design was of a band-pass filter with a sampling frequency of 48kHz, pass band of 9-12 kHz and stop band of 6-15kHz. We optimized the filter for a higher frequency of operation as well as lesser power dissipation on target FPGA. The design was created using MATLAB and Simulink. HDL coder HLS platform along with multiplier sharing, and distributed pipelining HLS directives were used to generate verilog code. The final RTL code was implemented on Kintex 7 FPGA and found to have better runtime throughput than a reference implementation built using hand coded RTL. However, due to the usage of distributed pipelining, design used slightly higher flip-flop count than the reference implementation (Section 3.1).

6.1.1.2 QPSK modulator design, optimization, and implementation using HLS

We used MATLAB HDL coder HLS platform to generate synthesizable Verilog code for a QPSK modulator created using MATLAB and Simulink. The Verilog code was synthesized and targeted on Xilinx Kintex 7 FPGA. We were able to achieve a higher throughput as well as lesser resource utilization for the final implementation as compared to hand-coded implementation. The optimization was achieved mainly by the use of distributed pipelining directive. Owing to a slight increase in the number of flip-flops due to distributed pipelining, there was a slight increase in power dissipation of the design as compared to hand-coded RTL implementation for the same architecture. The final, optimized implementation was able to run at 330 MHz on Kintex7 device dissipating 0.263 W of power (Section 3.2).

6.1.1.3 MIPS processor core design, implementation, and optimization using HLS

We used MATLAB function blocks in Simulink to design a MIPS core application. After the design in a high-level platform (MATLAB and Simulink), we used MATLAB HDL coder to generate synthesizable Verilog code. We targeted the same Verilog code on Virtex7 FPGA using Xilinx Vivado. The model in MATLAB and Simulink model was optimized using pipeline and loop unrolling HLS directives. Even though functional verification results were identical (confirmed using Verilog simulation on xSim), our FPGA implementation results were superior, in terms of the area and power dissipation, as compared to two other implementations available in the literature. The proposed implementation took 30 to 40% lesser resources as compared with others. The maximum frequency of operation achieved (400MHz on Virtex 7) is also slightly higher when compared with other implementations available in the literature (Section 3.3).

6.1.1.4 AES encryption algorithm optimization using HLS directives

We designed and implemented the AES algorithm using CTR mode of operation employing 10 rounds, and a 128 bits block length. Vivado HLS was used to produce the baseline design. The HLS simulation was done using C++ testbench and the results were matching the theoretical results for the AES algorithm. We optimized the high-level model in C++ using HLS directives namely pipelining, loop unrolling, and function in-lining. The final implementation achieved a throughput of 54.2 Gbps on Kintex7 and a throughput of 276.5 Gbps on Virtex 6. The generated Verilog code was implemented on Kintex7 FPGA using Xilinx Vivado and Virtex 6 device using Xilinx iSE. Even though implementation was functionally correct (confirmed using RTL simulation results using xSim), our proposed implementation is superior in terms of runtime throughput over others available in the literature (Section 3.4).

6.1.1.5 YOLO v2 deep learning algorithm optimization using HLS directives

We used Vivado HLS platform for designing a YOLO v2 deep learning algorithm. We applied Vivado HLS directives, namely pipelining, loop unrolling, and latency, for optimizing the implementation for a low area and high throughput. The final implementation was targeted on Xilinx Zynq FPGA device and comparison presented with other implementations available in the literature. Our proposed implementation was able to run at faster runtime throughput but utilizing order of magnitude lesser FPGA resources as compared to other implementations available in the literature. Further, in terms of functionality, a mean average precision of 0.48 at intersection of union of 0.5 was achieved for our implementation. Similar data was not available for other implementations available in the literature. The comparison was done against three reference

implementations available in the literature on Ultrascale+ and Virtex 7 FPGAs (Section 3.5).

6.1.2 Application-specific bit widths for intermediate data nodes (novel method for HLS optimization)

We proposed a technology and HLS platform independent methodology for optimizing application designs. The proposed method works without manual intervention and could be automated with the deployment of multiple applications to the target area, speed, or power optimization, depending on the application of the design. The proposed methodology also enables the redesign if simulation results do not adhere to threshold values. Such values could be mandatory for the design to be considered verified, or if synthesis results are beyond the area and timing budgets. We have proved the efficacy of this novel methodology using multiple application designs (Section 4.2).

6.1.2.1 HLS optimization of bandpass DSP filter using ASBWIDN

The design was of a band-pass equiripple filter with a sampling frequency of 48 kHz, pass band of 9–12 kHz and stop band of 6–15 kHz (Same design as used in Section 3.1).

The filter designed using MATLAB and Simulink environment had an order of 40. The same was optimized using the novel methodology based on application-specific bit widths. The functionality of the filter was verified using a chirp signal as input. The generated Verilog code from MATLAB HDL coder was implemented on Kintex 7 FPGA. The final implementation was able to run at 52 MHz on target FPGA device. In addition to speed, an improvement in area utilization as well as power dissipation for our implementation as compared to other implementations available in the literature was observed. We also did a comparison of implementation results with and without the use of our novel optimization techniques (Section 4.3) in an attempt to prove its efficacy.

6.1.2.2 Sobel Edge Filter Design and Optimization using ASBWIDN

Sobel edge filtering is a commonly used design in image processing and computer vision applications. MATLAB and Simulink were used to design a Sobel edge image filtering algorithm. The design was verified using an input image with a resolution of 1920 × 1080. RTL (generated using HDL coder) simulation results were compared with the golden MATLAB model (Library block in Simulink) and found to be identical. Further, we quantized the bit width

by using application specific bit width methodology and created an optimized implementation. The generated RTL design was implemented on Kintex 7 FPGA and had less than 1 percent quantization error. This was confirmed by using FPGA-in-the-loop feature of MATLAB HDL Verifier and also by using RTL simulation on xSim software.

The final implementation (with optimal bit width constraints in synthesis) ran for multiple image formats and could run at 250 MHz on Kintex7 FPGA device. The proposed implementation achieved an area reduction of about 50 percent as compared to others available in the literature. Further, the proposed implementation was 30% faster and dissipated 20% less power than reference designs. Additionally, the results achieved with optimized implementation were also seen to be superior as compared to the implementation with pessimistic bit widths (Section 4.4).

6.1.2.3 Harris Corner Detection Algorithm – implementation using HLS and Optimization using ASBWIDN

We demonstrated a real-time implementation of Harris corner detection (an algorithm widely used in image processing applications) on Xilinx ZedBoard. MATLAB HDL coder was used for the high-level design, Xilinx xSim for functional simulation, and Vivado for FPGA synthesis. We used the same method for optimizing the design: application-specific bit width for intermediate nodes. Our implementation reported a power dissipation of 0.315 W and we were able to achieve a frame rate of 160 fps on ZedBoard. In terms of the target area, our implementation took 20% to 50% lesser resources than other available implementations in the literature (Section 4.5).

6.1.2.4 Digital Down Converter for Software-Defined Radio Applications :Optimization using ASBWIDN

This work was based on MATLAB and Simulink to design a DDC to down convert an intermediate frequency signal from 200 MHz to 2 MHz. The double-precision model was created using MATLAB and Simulink and internal signal bit widths were optimized using ASBWIDN. For functional verification of the design, we chose an input chirp centered at 200 MHz with a bandwidth of 20 MHz. Additionally, we also evaluated the quantization error for our implementation using FPGA-in-the-loop simulation and found it within 1% limit. The optimized verilog design was implemented on Kintex7 FPGA.

Our implementation was compared against a golden double-precision model (pessimistic bit widths) and three other reference implementations available in the literature. The proposed

design attained a maximum frequency of operation of about 500 MHz which is higher than reported by other works in literature. Also, our design had a power dissipation of 0.325 W which is lesser than other references (Section 4.6).

6.1.2.5 AES Algorithm optimization using ASBWIDN

Herein, we use Vivado HLS to design an AES algorithm for automotive applications and optimized the same using ASBWIDN. We targeted the generated Verilog code for Virtex6 and Kintex7 FPGAs and compared our implementation with three other implementations available in the literature. The throughput achieved on Virtex6 (562 MHz) and Kintex7 (300 MHz) is 10%–20% more than other similar implementations. The final power dissipation reported is 0.114 W (Section 4.7).

6.1.3 RADAR signal processor design, optimization, and implementation for automotive applications

We presented an HLS-assisted design and verification framework for RADAR processors used in automotive ADAS applications. The HLS directives (distributed pipelining and resource sharing) were applied to optimize the generated RTL and the FPGA implementation. The FMCW RADAR processor calculates distance, velocity, and angle of elevation for a target vehicle relative to the transmitter. The designed RADAR system had a center frequency of 77 GHz, sweep bandwidth of 150 MHz, and a chirp time of 7.33 microseconds. In addition to HLS directives, we used novel HLS optimization method based on bit width optimization, as explained above for this design application. RTL verilog implementation was targeted for both Virtex6 and Kintex7 FPGAs and implementation results compared with others in the literature.

In addition to RADAR processor design, we also proposed an HLS-based verification framework, which was used to verify the RADAR signal processor by modeling multiple targets in an environment model, namely a pedestrian, car, and truck. The velocity, distance, and angle of elevation of each target were calculated and compared with actual values. The error reported was within the permissible limits. Further the error reported in measurements was lesser as compared to the ones reported by two other implementations available in literature.

6.2 Limitations and Future directions

There are some limitations of this work that we have identified. These can be addressed as part of future works. Since HLS flows are getting increasingly popular in VLSI design flows, addressing these limitations through future research will be a good value addition.

A) Usage of directives and novel methods suggested for tools other than MATLAB HDL coder and Vivado HLS.

Our study for diverse applications was mainly focused on Xilinx Vivado HLS and MATLAB HDL coder due to their wide acceptance in industry and license availability for us. However, the efficacy of these directives and novel method developed in this study (application-specific bit widths for intermediate data nodes), has not been verified for other HLS tools. Other tools may have different compiler directives, inbuilt optimizations and hence may lead to different results. There are multiple HLS tools available in the industry and academia (open source). Some examples include Mentor Graphics Catapult-C, Cadence Stratus, and pyverilog (open source). As part of future work, one can extend these applications and optimization methods to these platforms and tools.

B) Compiler related optimizations for HLS tools.

The novel HLS based design methodology (described in chapter 4) relies on calculation of optimum bit widths for inputs, outputs and internal signals accessible to the user in high-level language code as in C or MATLAB. But in addition to these variables, compilers may have many intermediate signals which may hold temporary values like stack or temporary storage registers etc. So, there is a possibility to enhance the high level language parsers and compilers to optimize the code with help of logic re-ordering and changing the precedence order for computations. As a part of future work, one can enhance the HLS code compilers to support the same and hence make further optimizations to the generated RTL code.

C) Optimization of application specific bit width methodology for large designs having larger number of nets

The novel HLS design methodology described in Chapter 4 does the bit width calculation in an automated manner for all internal nets and external design inputs and outputs. For all the designs that were designed as part of this work, the number of such nets were limited to few hundreds or thousands. Since the time taken for bit width optimization algorithm is small as compared to overall HLS design cycle time, the exercise to measure the time required for bit width optimization algorithm to run is not done as part of this work. But for larger designs with high number of design nodes, the complexity of this algorithm may increase and it may take long time to converge. As a result, it may result into a larger design cycle time. Hence, as part of future work, one can work to optimize the algorithm complexity.

D) Bit width optimization for applications requiring real-time inputs (like CNN)

The novel HLS design methodology described in Chapter 4 is currently not applicable to scenarios or applications where we have real-time input data. Since the method relies on calculation of maxima and minima during a high level simulation, the optimization method would not work if the complete input stimulus is not available at the start of simulation. There have been numerous implementations proposed in recent literature for applications like CNNs where object detection happens in real time. Researchers have optimized the implementations for area, speed and power using bit width trade-offs and hence compromising the accuracy [6.1, 6.2, 6.3, 6.4]. Even though all these implementations are based on bit width trade-offs for detection accuracy, all of these are based on hand-coded RTL approach. As a part of future work, one can extend the novel method described in chapter 4 for real time input data and obtain interesting results for applications like CNNs. Once such optimization is successful, it is expected to give even better results as compared to implementations where a manual trade-off is done.

E) Extension of HLS optimization techniques to ASIC designs and other FPGAs

In our study, we have used the novel optimization methods and available HLS directives for optimizing the designs targeted on FPGAs like Kintex7, Virtex6, and Zynq (ZC 702 and Zedboard). However, the generated RTL was not taken through the ASIC synthesis flow all the way to tape out. In the future, efficacy of HLS flows and optimizations for ASIC designs can be explored. In fact, as all HLS optimization methods kick-in at a higher abstraction level (RTL level), hence they are independent of the target technology (ASIC or FPGA). Therefore, these optimizations are expected to give good results with ASIC synthesis flows as well. Hence, there is a definite potential to extend these optimizations for RTL to silicon flows in future studies. Moreover, because of technology independence, the implementation results for other FPGA targets (For example, Intel) are also expected to be optimal. Hence, there is a definite potential for research to prove efficacy of HLS optimization to other FPGAs.