Vincent Han

EE 4301

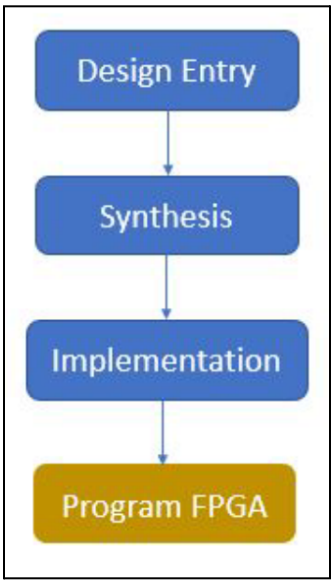
Lab Report 1

1 Introduction

1.1 FPGA Design

FPGA design is crucial in digital electronics and computer engineering for several reasons. FPGAs provide flexibility, adaptability, and customizable hardware implementations, enabling iterative refinement and enhanced performance. They also facilitate rapid prototyping and exploration of novel architectures and algorithms. FPGA design offers advantages in time-to-market and cost efficiency, allowing for pre-fabrication verification and post-deployment updates. Additionally, it bridges the gap between software and hardware development, promoting collaboration and integration. Overall, FPGA design plays a vital role in driving advancements and implementing cutting-edge technologies in various fields.

1.2 Design Flow



The FPGA design flow consists of several key steps. Design entry involves describing the functionality using an HDL. Synthesis converts the HDL code into a gate-level representation. Implementation maps the design onto the FPGA, including technology mapping, placement, and routing. Finally, the FPGA is programmed by loading the generated bitstream onto the device. This process enables efficient design and deployment of complex digital systems on FPGA platforms.

1.3 Xilinx Tools

Xilinx provides a comprehensive suite of tools that play a pivotal role in facilitating the FPGA design flow. The Vivado Design Suite, a robust platform, encompasses multiple stages of the design process, ensuring a seamless workflow and empowering engineers to maximize the potential of Xilinx FPGAs. Vivado HLS allows designers to describe designs using high-level languages, generating optimized RTL code. Vivado synthesis transforms RTL code into an optimized gate-level netlist, improving performance, power consumption, and area utilization. Vivado implementation handles technology mapping, placement, and routing, optimizing resource utilization. Vivado programming simplifies FPGA configuration, generating the bitstream for loading onto the FPGA. Xilinx's tools streamline the design and implementation process, enhancing efficiency and enabling engineers to harness the full capabilities of Xilinx FPGAs.

2 Body

2.1 Design Entry

During the design entry step, engineers use hardware description languages (HDLs) like Verilog or VHDL to describe the desired functionality of the FPGA design. Input files include the

engineer's HDL code, along with any required libraries or IP cores. The design entry step generates several important output files, including the HDL file that serves as the basis for synthesis and implementation. Timing constraints files specify the desired timing requirements for the design, ensuring proper operation and performance. Simulation files are generated for testing and debugging the design in a software-based simulation environment. The design entry step provides a foundation for subsequent stages of the FPGA design flow. It enables synthesis and implementation of the design, ensuring accurate representation of desired functionality. Through HDLs, engineers can capture the behavior and structure of the design. The input files provide the necessary information to define the FPGA design. Output files, such as the HDL file, timing constraints, and simulation files, facilitate the development, validation, and testing of the design. Design entry is a crucial step that bridges the gap between the engineer's specifications and the subsequent stages of FPGA design. It allows for the translation of functional requirements into a digital representation. This step lays the groundwork for synthesis, implementation, and verification, ensuring the successful realization of the FPGA design.

2.2 Synthesis

During the simulation step, the FPGA design is tested and validated using software-based simulation tools. Input files include the design's HDL files, testbench files, and simulation libraries. Simulation runs test scenarios, evaluates design behavior, and identifies flaws. Output files include waveform files for visualization, log files for debugging, and coverage reports for test thoroughness. Simulation helps ensure correct operation and optimize performance. It aids in identifying design flaws and verifying functionality. Input files provide necessary information for accurate simulation. Waveform files capture signal behavior over time. Log files offer detailed information on simulation processes and errors. Coverage reports assess test completeness.

Simulation enhances design quality and reliability. It precedes synthesis and implementation stages. It aids in troubleshooting and refining the design. Simulation is a crucial step in FPGA development, enabling thorough testing and validation. It ensures the design functions as intended before physical implementation.

2.3 Implementation

During the implementation step, the FPGA design is mapped to the target device, converting it into a physical realization. Input files include the synthesized netlist and design constraints. Implementation involves technology mapping, placement, and routing. Output files comprise the routed design, timing reports, power analysis reports, and resource utilization reports. The routed design specifies physical locations and interconnections. Timing reports analyze timing performance, critical paths, and potential violations. Power analysis reports estimate power consumption. Resource utilization reports detail resource usage. Implementation optimizes design placement and routing for efficient FPGA utilization. It ensures the design meets timing requirements and minimizes power consumption. Input files guide the implementation process, providing design and constraint information. Output files aid in optimizing, refining, and validating the design. Implementation is a critical step in FPGA development, transforming the abstract design into a physically realized circuit.

2.4 Program FPGA

The "program FPGA" step involves loading the configuration data, known as a bitstream, onto the FPGA device. Input files consist of the generated bitstream file. This file contains the specific configuration information for the FPGA design. The primary output of this step is the successful configuration of the FPGA device, enabling it to execute the desired functionality. The program FPGA step transfers the bitstream from a host device to the FPGA. It establishes

the logic and functionality of the FPGA based on the loaded bitstream. However, this step typically does not produce additional output files beyond the successful FPGA configuration. Its main objective is to configure the FPGA with the specified design rather than generating specific output files. Ultimately, the program FPGA step completes the FPGA design flow, making the device ready to operate according to the configured logic and functionality.

2.5 8-bit Adder/Subtractor

When creating the 8-bit adder/subtractor, it gave a good introduction to the design flow of working with a FPGA, as well as utilizing the tools that Xilinx Vivado provides. I had initially created a full adder, which had the capability to add two bits together. Then combining eight full adders together creates an 8-bit adder. I was able to test its functionality by using a testbench file and test cases. Looking at the waveform outputs, I was able to see if the 8-bit adder was working correctly. I then was able to create a constraints file, which assigns physical parts of the FPGA to the 8-bit adder. We can then generate a bitstream, and test out the FPGA and the Verilog code physically in order to test its functionality. I then was able to edit the code to add additional functionality with the button. This whole process gave me good exposure to the FPGA design process, Vivado, and of course Verilog.

3 Summary

The FPGA design flow consists of four steps: design entry, synthesis, implementation, and program FPGA. In the design entry phase, we are able to scheme a design and its functionality. We can create the base level parameters that will build up a whole implementation. Moving onto the synthesis, we can start adding together those base level components to create higher level functioning components. Additionally, we can start adding behaviors. We then can move onto the implementation phase, where we can start applying the components we have programmed and

assign it to a physical board. Then finally we can send our implementation over to the physical FPGA, in which we can test its functionality. While going through the process of designing an 8-bit adder/subtractor, we had good exposure and practice with the FPGA design flow.