Chapter 3: Digital Circuit Design and Implementation on FPGAs

3.1 Introduction to Digital Circuit Design on FPGAs

Digital circuits are the backbone of modern electronic systems, enabling everything from simple logic functions to complex signal processing. Field-Programmable Gate Arrays (FPGAs) provide an ideal platform for designing and implementing these circuits due to their flexibility, parallel processing capabilities, and high-speed performance.

This chapter introduces the process of digital circuit design and implementation on FPGAs using **VHDL** and **Verilog**. We will cover the steps required to translate design specifications into functional systems on an FPGA, from conceptualization and simulation to implementation and verification. Additionally, we will discuss practical tips for optimizing designs and improving performance.

3.2 Understanding the Design Flow for FPGA-Based Systems

The design flow for FPGA-based systems consists of several stages that ensure the correct implementation of a digital circuit. These stages include:

1. **Design Specification**:
    - Define the problem and the required functionality.
    - Create a detailed description of the digital circuit, including its inputs, outputs, and behavior.
2. **Hardware Description**:
    - Write the design using VHDL or Verilog to describe the circuit's behavior and structure.
3. **Simulation**:
    - Simulate the design to verify its functionality and check for any errors in the behavior or performance.
4. **Synthesis**:
    - Convert the hardware description into a netlist that represents the logical gates and components to be implemented on the FPGA.

5. **Implementation**:
    - Map the synthesized design to the FPGA architecture, considering constraints like timing and resource usage.
6. **Verification**:
    - Verify the design's functionality on the actual FPGA using testbenches and real-world stimulus.
7. **Programming the FPGA**:
    - Generate the configuration bitstream and program the FPGA to implement the design.
8. **Testing and Debugging**:
    - Test the FPGA-based system, troubleshoot any issues, and make necessary modifications.

3.3 Design Specification and Requirements

The first step in designing any digital circuit on an FPGA is to understand the functional requirements. The design specification should outline the circuit's functionality, input/output interfaces, and timing constraints.

Example: 4-Bit Binary Adder

Design Specification:

- Inputs: Two 4-bit binary numbers (`A` and `B`) and a carry input (`Cin`).
- Outputs: A 4-bit sum (`S`) and a carry output (`Cout`).
- Functionality: The circuit adds the two binary numbers along with the carry input and outputs the sum and carry output.

This simple example will help demonstrate the process of translating a high-level design into FPGA code.

3.4 Writing the VHDL/Verilog Code for FPGA Implementation

Once the design specifications are clear, the next step is to describe the circuit using VHDL or Verilog. The code must include the entity/module definition, port declarations, and logic to implement the desired functionality.

3.4.1 VHDL Example: 4-Bit Binary Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY ADDER_4BIT IS
  PORT (
    A     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    B     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    Cin   : IN  STD_LOGIC;
    Sum   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    Cout  : OUT STD_LOGIC
  );
END ENTITY ADDER_4BIT;

ARCHITECTURE behavior OF ADDER_4BIT IS
BEGIN
  PROCESS (A, B, Cin)
  BEGIN
    Sum <= A + B + Cin;
    Cout <= (A(3) AND B(3)) OR (A(3) AND Cin) OR (B(3) AND Cin);
  END PROCESS;
END ARCHITECTURE behavior;
```

In this VHDL example, the `ADDER_4BIT` entity takes two 4-bit input vectors `A` and `B`, along with a carry input `Cin`, and outputs a 4-bit sum `Sum` and a carry-out `Cout`. The architecture implements the addition operation and calculates the carry-out based on the most significant bits of the inputs.

3.4.2 Verilog Example: 4-Bit Binary Adder

```
module ADDER_4BIT (input [3:0] A, input [3:0] B, input Cin, output [3:0] Sum, output Cout);

  assign {Cout, Sum} = A + B + Cin;

endmodule
```

In this Verilog example, the `ADDER_4BIT` module also implements the same functionality as the VHDL example. The `assign` statement performs the addition of the input vectors `A` and `B`, including the carry input `Cin`, and assigns the result to the `Sum` output and the carry-out `Cout`.

3.5 Simulation and Verification

Before implementing the design on the FPGA, it is important to verify the functionality of the digital circuit using simulation. Simulating the design helps detect logic errors, race conditions, or timing violations early in the process.

3.5.1 Writing a Testbench for the 4-Bit Adder

A **testbench** is a VHDL or Verilog module that generates input stimuli and checks the outputs of the design. Here's an example of a simple testbench for the 4-bit adder:

VHDL Testbench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TB_ADDER_4BIT IS
END ENTITY TB_ADDER_4BIT;

ARCHITECTURE behavior OF TB_ADDER_4BIT IS
  SIGNAL A, B : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL Cin  : STD_LOGIC;
  SIGNAL Sum  : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL Cout : STD_LOGIC;
  COMPONENT ADDER_4BIT
    PORT ( A    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
           B    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
           Cin  : IN  STD_LOGIC;
           Sum  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
           Cout : OUT STD_LOGIC);
  END COMPONENT;
BEGIN
  uut: ADDER_4BIT PORT MAP (A => A, B => B, Cin => Cin, Sum => Sum, Cout => Cout);

  stim_proc: PROCESS
  BEGIN
    A <= "0001"; B <= "0010"; Cin <= '0'; WAIT FOR 10 ns;
    A <= "1111"; B <= "0001"; Cin <= '1'; WAIT FOR 10 ns;
    A <= "0101"; B <= "1010"; Cin <= '0'; WAIT FOR 10 ns;
    WAIT;
```

```
  END PROCESS;
END ARCHITECTURE behavior;
```

Verilog Testbench:

```
module TB_ADDER_4BIT;
  reg [3:0] A, B;
  reg Cin;
  wire [3:0] Sum;
  wire Cout;

  ADDER_4BIT uut (A, B, Cin, Sum, Cout);

  initial begin
    A = 4'b0001; B = 4'b0010; Cin = 0; #10;
    A = 4'b1111; B = 4'b0001; Cin = 1; #10;
    A = 4'b0101; B = 4'b1010; Cin = 0; #10;
    $finish;
  end
endmodule
```

Both testbenches stimulate the adder with different input values for `A`, `B`, and `Cin`, and verify that the output `Sum` and `Cout` are correct.

3.6 Synthesis and Implementation

After successfully simulating the design and ensuring its correctness, the next step is synthesis. The synthesis tool converts the hardware description into a netlist, mapping the design onto the FPGA's resources (logic blocks, I/O, etc.).

- **Synthesis Tools**: Vivado (Xilinx), Quartus (Intel), or other FPGA vendor-specific tools.
- **Implementation**: The synthesis tool performs placement and routing of the design, ensuring that the logic is correctly mapped to the FPGA.

3.7 Conclusion

This chapter covered the fundamentals of digital circuit design and implementation on FPGAs, focusing on how to translate design specifications into functional systems using **VHDL** and

Verilog. You learned how to define, simulate, synthesize, and implement digital circuits on FPGA platforms. By practicing these concepts, you will gain the skills necessary to design more complex FPGA systems and contribute to real-world digital solutions.