Documentation of 1-Bit Full Adder in Verilog

Class 1i

Contents

1 Introduction

This document provides a detailed description of the 1-bit full adder module implemented in Verilog. A full adder is a digital circuit that performs the addition of binary numbers. In this design, the module takes three inputs: two single-bit binary values, `a` and `b`, and a carry-in bit, `carry_in`. It produces two outputs: the sum (`sum`) and a carry-out bit (`carry_out`).

2 Module Description

The 1-bit full adder module is defined in Verilog using the following interface:

```
module full_adder (
    input wire a,         // Input A
    input wire b,         // Input B
    input wire carry_in,  // Carry-in
    output wire sum,      // Sum output
    output wire carry_out // Carry-out
);
```

2.1 Inputs and Outputs

- **Input a**: The first binary input (single bit).
- **Input b**: The second binary input (single bit).
- **Input carry_in**: The carry-in bit, representing any carry from the previous addition stage.
- **Output sum**: The sum result of inputs `a`, `b`, and `carry_in`.
- **Output carry_out**: The carry-out result, which is passed to the next stage if multiple bits are added.

3 Operation

The 1-bit full adder performs binary addition using the logic operations XOR, AND, and OR. The outputs are calculated as follows:

$$\text{sum} = a \oplus b \oplus \text{carry\_in}$$
$$\text{carry\_out} = (a \wedge b) \vee (\text{carry\_in} \wedge (a \oplus b))$$

3.1 Truth Table

The truth table for the 1-bit full adder is shown below:

a	b	carry_in	sum	carry_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4 Implementation

The Verilog implementation of the full adder uses logical operations to compute the sum and carry-out as shown below:

```
module full_adder (
    input wire a,
    input wire b,
    input wire carry_in,
    output wire sum,
    output wire carry_out
);
    assign sum = a ^ b ^ carry_in;
    assign carry_out = (a & b) | (carry_in & (a ^ b));
endmodule
```

5 Parameterization

To make this module more versatile, we can parameterize it to allow the user to define different bit widths. Here is an example of a parameterized full adder that allows for a multi-bit input:

```
module full_adder #(
    parameter WIDTH = 1
) (
    input wire [WIDTH-1:0] a,
    input wire [WIDTH-1:0] b,
```

```
    input wire carry_in,
    output wire [WIDTH-1:0] sum,
    output wire carry_out
);
    assign {carry_out, sum} = a + b + carry_in;
endmodule
```

In this parameterized version, `WIDTH` is a parameter that specifies the number of bits. The module can handle inputs of any width by changing the `WIDTH` value when instantiating the module.

6 Compilation and Synthesis Instructions

To compile and synthesize the Verilog code for the iCEBreaker FPGA, follow these steps:

1. **Save the Verilog file**: Save the Verilog code as `sum.v` and the pin configuration as `sum.pcf`.

2. **Synthesize with Yosys**:

   ```
   yosys -p "synth_ice40 -top full_adder -json sum.json" sum.v
   ```

   This command synthesizes the Verilog code for the iCE40 FPGA architecture and outputs a JSON netlist.

3. **Place and route with nextpnr**:

   ```
   nextpnr-ice40 --up5k --package sg48 --pcf sum.pcf --json sum.json --asc
   ```

   This command places and routes the design for the UP5K model of the iCE40 FPGA.

4. **Generate a binary file with icepack**:

   ```
   icepack sum.asc sum.bin
   ```

   This converts the ASCII file (`.asc`) to a binary file (`.bin`) for programming the FPGA.

5. **Program the FPGA with iceprog**:

```
iceprog sum.bin
```

   This command uploads the binary file to the iCEBreaker FPGA board.

7 Testing

To verify the correctness of the full adder, the module can be tested with all combinations of inputs (as shown in the truth table) to ensure that the sum and carry-out values are produced correctly. A testbench in Verilog can be created to apply these inputs and observe the outputs.

8 Conclusion

This document provides a detailed overview of the 1-bit full adder module implemented in Verilog, including its interface, operation, and logic. This module is fundamental in digital systems, especially for implementing multi-bit adders and arithmetic operations in larger circuits.