# Activity 01 Part 1 – Two-Bit Ripple Adder

Richard Groves

09/05/2023

Class: ECE4/530 - Digital Hardware Design

# 1. Introduction

## 1.1 Objective

## The objective of this activity was to design, synthesize, simulate, and implement a two-bit full adder circuit using Verilog. This involved: developing a modular design that started with a single-bit full adder and extended to a two-bit version, creating a functional testbench for simulation and verification, and assigning appropriate package pins through a constraint file.

# 2. Design

## 1.1 Design Approach

A modular design technique was used for this design.

### 1.1.1 Two-Bit Ripple Adder

The two-bit ripple adder was broken down into two one-bit full adder modules.

A diagram of a computer program

Description automatically generated

### 1.1.2 One-Bit Full Adder

RTL modeling was used to describe the one-bit full adder module.

## 1.2 Design Details

### 1.2.1 Module List

The individual module files are included with this report.

|  |  |  |
| --- | --- | --- |
| Module Name | Module File Name | Module Description |
| Full\_Adder | Full\_Adder.v | This full adder performs the addition of three one-bit binary numbers. The three inputs are a, b, and cin (Carry in), and it produces two outputs: s (Sum) and cout (Carry out). The sum output represents the sum of the inputs, while the carry out represents a carry-over value, if any. |
| Full\_Adder\_2Bit | Full\_Adder\_2Bit.v | This two-bit ripple carry adder is made of two cascading full adders, where the carry out of the first full adder serves as the carry in for the second full adder. This setup allows for the addition of two two-bit binary numbers. |

### 1.2.2 Schematics

#### 1.2.2.1 Full\_Adder\_2Bit Schematics

This schematic is structured as a composition of two single-bit full adder modules, interconnected to form a single two-bit ripple adder. In the initial stage, the least significant bits of the input numbers (a[0] and b[0]) are fed into the first full adder module along with the carry-in signal. This module computes the sum of these inputs, generating a sum output (s[0]) and a carry-out signal. This carry-out signal then serves as the carry-in for the next stage, where it is combined with the most significant bits (a[1] and b[1]) in the second full adder module. This stage produces the final sum output (s[1]) and a final carry-out signal. The combination of s[1] and s[0] forms the final 2-bit sum output, and the final carry-out signal serves as an indication of an overflow (if any).

A diagram of a circuit

Description automatically generated

A diagram of a circuit

Description automatically generated

## 1.3 Simulation

### 1.3.1 Testbench Module List

The individual module files are included with this report.

|  |  |  |
| --- | --- | --- |
| Module Name | Module File Name | Module Description |
| tb\_full\_adder\_2bit | tb\_full\_adder\_2bit.v | This testbench module is for a 2-bit full adder, specifying and simulating all possible input vectors and observing the sum and carry-out outputs. |
| tb\_loop\_full\_adder\_2bit | tb\_loop\_full\_adder\_2bit.v | This is a testbench module for a 2-bit full adder. It utilizes a loop structure to systematically simulate all possible input combinations (0 to 31) for the adder. |

### 1.3.2 Waveforms

#### 1.3.2.1 tb\_loop\_full\_adder\_2bit Waveform

A screenshot of a video game

Description automatically generated

#### 1.3.2.2 tb\_loop\_full\_adder\_2bit Waveform

A screenshot of a video game

Description automatically generated

## 1.4 Synthesis

### 1.4.1 Slices/LUTs

|  |  |
| --- | --- |
| Slices | LUTs |
| SLICE\_X43Y53 | fa2/o1 |
|  | S\_OBUF[1]\_inst\_i\_1 |

A computer screen shot of a diagram

Description automatically generated

## 1.5 Implementation and Testing

### 1.4.1 Implementation Files

These implementation files included with this report.

|  |  |
| --- | --- |
| File Name | File Description |
| Full\_Adder\_2Bit.bit | This is the bitstream file for the two-bit full adder project. |

### 1.4.2 Post-Implementation Timing

After running the post-implementation timing simulation, it was found that there are delays between changes in the inputs and changes in the outputs. In the image below, the measured delay between a change in input tb\_b and output tb\_s is 8.467ns. Additionally, the delay between a change in input tb\_cin and output tb\_s is 9.12ns. It is believed that there are a couple of contributing factors to these delays. Namely: propagation delay and routing delay. These are factors that Vivado considers, and the factors are unique to the FPGA the project is being designed for. In this instance, the propagation delay is believed to come from the LUTs used in the design, where the routing delay comes from the signal routing paths used on the FPGA.

A screenshot of a computer

Description automatically generated

### 1.4.3 Testing

The image below shows a test of adding 2+3 (a+b) on the ZYBO-Z7 board.

A green circuit board with many small black and silver objects

Description automatically generated