Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS3310

Report

Signed-Unsigned Comparator in Verilog

Prepared by: Qusay Bider 1220649 **Se:2**

Dr.Elias Khalil

Date: 19/12/2024

Abstract

In digital systems, value comparison is a crucial operation, especially when assessing the relationship between signed and unsigned integers. This project focuses on the implementation of a Signed-Unsigned Comparator in Verilog. The design is capable of comparing 6-bit numbers in both signed and unsigned formats, with the mode selected via a control signal. The objective is to determine whether the first input number is equal to, greater than, or less than the second input number. The comparator design adopts a modular approach, featuring two main modules: one for signed comparison and another for unsigned comparison, along with a register to manage input/output synchronization.

Table of Contents

Table of Figures

Theoretical Overview

2.1 Signed and Unsigned Numbers:

Unsigned Numbers: These are non-negative values represented solely by their magnitude. In a 6-bit unsigned number, all six bits contribute to the overall value, with no designated sign bit.

Signed Numbers: Signed numbers encompass both positive and negative values, with the two's complement method being the most prevalent for representation. In a 6-bit signed number, the most significant bit (MSB) indicates the sign: 0 represents positive values, while 1 signifies negative values.

The comparison operation in digital systems involves determining whether one number is greater than, equal to, or less than another. This process becomes somewhat more intricate for signed numbers, as the MSB must be interpreted as a sign bit.

2.2 Comparison Logic:

Unsigned This entails directly comparing the bit patterns of the numbers, where larger values are indicated by higher bit values in the most significant positions.

In signed comparison, the most significant bit (MSB) represents the sign of the number. If the MSB of a number is 1, it indicates a negative value, necessitating special considerations when comparing it to positive numbers or other negative values.

2.3 Registers:

The design utilizes 12-bit registers (modules twelve_bit_register_in and twelve_bit_register_out) to store the concatenated input values and synchronize the outputs with the clock. These registers play a vital role in ensuring that the comparison operation occurs at the appropriate time during the simulation.

Design Philosophy

The design is organized to facilitate seamless switching between signed and unsigned comparison modes using the control signal S. The primary design considerations are as follows:

Modular Design: The project is segmented into separate modules for both signed and unsigned comparisons. This modular structure enhances debugging, testing, and future scalability of the design.

Clocking and Synchronization: The register modules ensure that inputs are captured and outputs are synchronized with the clock, providing stability during simulations and in real-world hardware implementations.

Comparison Modules: The Unsigned_comparator module handles unsigned comparisons, while the Signed_Unsigned_Comparator module manages signed comparisons. The latter utilizes the unsigned comparison when S = 0 and adjusts its behavior for signed numbers when S = 1.

Both comparison methods are implemented using bitwise operations such as AND, OR, NOR, and NOT gates, offering efficient mechanisms to assess the relationship between input values.

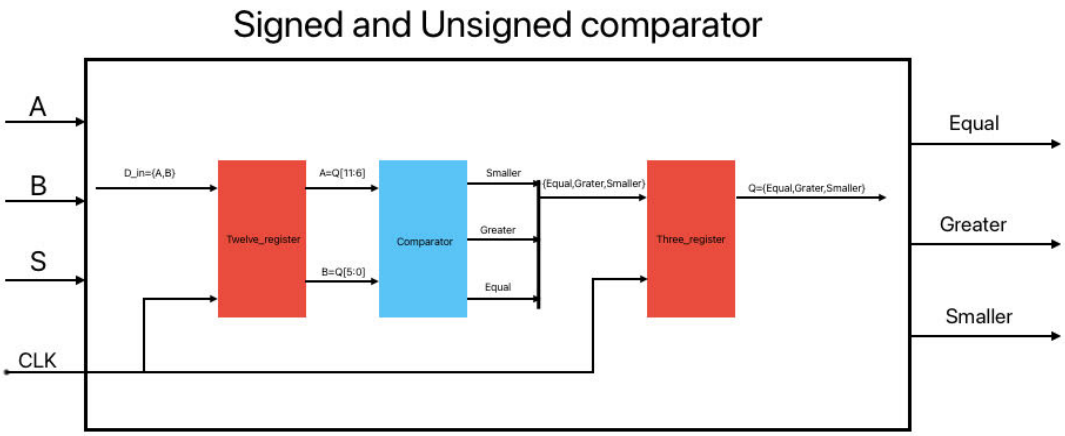

Figure 1:Design of Comparators

Simulation Results

The simulation was conducted using a comprehensive testbench, `Signed_Unsigned_Comparator_tb`, which verifies the module's functionality. Key observations:

1. **Unsigned Mode (S = 0):**

   The comparator accurately identifies the greater, smaller, and equal conditions for all 6-bit combinations of `A` and `B` in the range [0, 63].

2. **Signed Mode (S = 1):**

   Correctly handles two's complement values in the range [-32, 31].

3. **Behavioral Test Results:**

   - The testbench dynamically verifies the outputs for all combinations of A and B.
   - All assertions passed, confirming the module's correctness.

Testbench Results

***** The results of the case are shown in the next case result line.**

- ### Case1:

When s=0 , A=111001 , B=111001 .

```
Time = 749010 | A = 111001, B = 111001 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is equal to B


Time = 749170 | A = 111001, B = 111010 | Equal = 1, Greater = 0, Smaller = 0
Behavioral Test
A is less than B
```

Figure 2:Testbench,Case1

- ### Case2:

When s=0 , A=110110 , B=00001 .

```
Time = 535810 | A = 110110, B = 000001 | Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is greater than B


Time = 535970 | A = 110110, B = 000010 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B
```

Figure 3:Testbench,Case2

- ### Case3:

When s=0 , A=110010 , B=111110 .

```
Time = 505890 | A = 110010, B = 111110 | Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is less than B


Time = 506050 | A = 110011, B = 000001 | Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is greater than B
```

Figure 4:Testbench,Case3

- ## Case4:

When s=1 , A=100100 , B=100100 .

```
Time = 19810 | A = 100100, B = 100100 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is equal to B


Time = 19970 | A = 100100, B = 100101 | Equal = 1, Greater = 0, Smaller = 0
Behavioral Test
A is less than B
```

Figure 5:Testbench,Case4

- ## Case5:

When s=1 , A=111100 , B=111001 .

```
Time = 138370 | A = 111100, B = 111001 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B


Time = 138530 | A = 111100, B = 111010 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B
```

Figure 6:Testbench,Case5

- ## Case6:

When s=1 , A=101001 , B=111110 .

```
Time = 47970 | A = 101001, B = 111110 | Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is less than B


Time = 48130 | A = 101010, B = 100001 | Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is greater than B
```

Figure 7:Testbench,Case6

Note: The system will pass for each possible case from A and B to test it and after check value will print if the system passes or fail.

```
Time = 148290 | A = 111110, B = 111011 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B


Time = 148450 | A = 111110, B = 111100 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B


Time = 148610 | A = 111110, B = 111101 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is greater than B


Time = 148770 | A = 111110, B = 111110 | Equal = 0, Greater = 1, Smaller = 0
Behavioral Test
A is equal to B



The Test Pass
```

Figure 8:Test the system pass

Now, we will creating the incorrect Design For each in order to ensure the results.

Transfer_Level2[5] ⟶ Transfer_Level2[1]

```
// level 3 L (A < B)
and #8 level3_L5(Transfer_Level_L3[5], Transfer_Level2[1], Transfer_Level_A1[4]);
and #8 level3_L4(Transfer_Level_L3[4], Transfer_Level2[5], Transfer_Level2[4], Transfer_Level_A1[3]);
and #8 level3_L3(Transfer_Level_L3[3], Transfer_Level2[5], Transfer_Level2[4], Transfer_Level2[3], Transfer_Level_A1[2]);
and #8 level3_L2(Transfer_Level_L3[2], Transfer_Level2[5], Transfer_Level2[4], Transfer_Level2[3], Transfer_Level2[2], Transfer_Level_A1[1]);
and #8 level3_L1(Transfer_Level_L3[1], Transfer_Level2[5], Transfer_Level2[4], Transfer_Level2[3], Transfer_Level2[2], Transfer_Level2[1], Transfer_Level_A1[0]);

// level 4 L (A < B)
or #8 A_Less_B(D_out[0], Transfer_Level_A1[5], Transfer_Level_L3[5], Transfer_Level_L3[4], Transfer_Level_L3[3], Transfer_Level_L3[2], Transfer_Level_L3[1]);
```

Figure 9:Incorrect Design

```
Time = 3920   A = 000000 , B = 010000 , Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is less than B


Time = 4160   A = 000000 , B = 010001 , Equal = 0, Greater = 0, Smaller = 1
Behavioral Test
A is less than B


Time = 4400   A = 000000 , B = 010010 , Equal = 0, Greater = 0, Smaller = 0
Behavioral Test


The Test Faill
```

Figure 10:Test Case Incorrect Design

Conclusion

The design effectively implements a signed and unsigned comparator for 6-bit inputs. It employs a modular approach, featuring distinct register modules for storing input values and comparator modules for executing the comparisons. The system can perform both unsigned and signed comparisons based on the selection signal (S).

Simulation results demonstrated that the design functions as intended, delivering accurate comparison outcomes for both signed and unsigned inputs. The testbench results validated that the system correctly identifies greater than, less than, and equal conditions.

Future Works

Optimizing for Larger Bit-widths:

The current design works for 6-bit values, but extending this to 8, 16, or even 32-bit values would improve the versatility of the comparator. This could be achieved by adjusting the bit-widths in the register and comparator modules.

Performance Improvements:

The performance of the comparator can be improved by optimizing the logic gate delays and reducing the clock cycles required for the comparison, especially for larger bit-width numbers.