

Faculty of Engineering & Technology
Electrical & Computer Engineering Department
Advanced Digital Design – ENCS3310
Course Project's report

Prepared by: Khaled Abu Lebdeh

ID Number: 1220187

Instructor: Dr. Abdellatif Abu-Issa

Section: 4

Date: December 24, 2024

Abstract

In this project, we are considering implementing a comparator supporting both signed and unsigned inputs. It should be implemented in both structural and behavioral form using Verilog HDL.

The comparator takes two inputs of 6-bit to be compared, and a selection line to determine whether the comparison is based on signed or unsigned inputs. Then, the comparator finds whether input_1 is either greater, equal or smaller than input_2.

Table of Contents

Abstract	i
Table of Figures	iii
Theory	4
1- Universal Structural Comparator.....	6
1.1- Unsigned Structural Comparator	7
1.2- Signed Structural Comparator	8
2- Universal Behavioral Comparator for Design Verification	9
3- Result Analyzer for Design Verification	9
4- Linear-feedback Shift Register (LFSR) as a Vector Generator	9
Results	10
1- Unsigned Input Numbers	10
2- Signed Input Numbers	11
Conclusion and Future Work	13
References	14

Table of Figures

<i>Figure 1: 2-bit comparator circuit</i>	<i>Figure 2: n-bit comparator block</i>	4
<i>Figure 3: 4-bit Magnitude Comparator Circuit</i>		5
<i>Figure 4: Block Diagram</i>		6
<i>Figure 5: Comparator's "greater" output's formula</i>		7
<i>Figure 6: 4-bit LFSR</i>		9
<i>Figure 7: 6-bit LFSR in Verilog</i>		9
<i>Figure 8: Results of Unsigned-Inputs</i>		10
<i>Figure 9: Results of Signed-Inputs</i>		11
<i>Figure 10: Two inputs are equal</i>		12
<i>Figure 11: Error Detection</i>		12

Theory

Comparator is a digital circuit that compares inputs to determine whether the first number is larger than, less than or equal to the second number. This circuit typically has two inputs, input A and input B. While it has three outputs, greater ($A > B$), smaller ($A < B$) and equal ($A = B$). Note that only one output will be active at a time (geeks for geeks, n.d.).

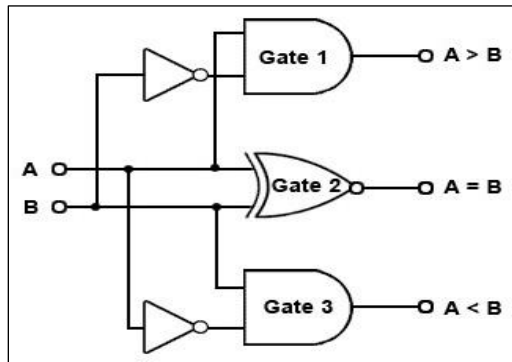


Figure 1: 2-bit comparator circuit

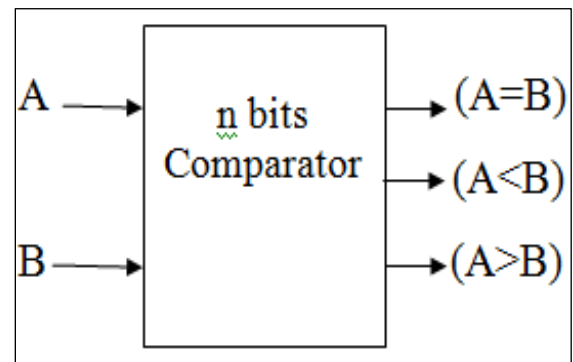


Figure 2: n-bit comparator block

Magnitude comparators support inputs with unsigned representation. However, we need a special signed comparator to compare signed inputs (usually in 2's complement representation).

Signed comparators focus on the most significant bit in inputs to be compared. The number is negative if the high-significant bit in its 2's complement representation is one. Otherwise, it is positive. The circuit should be able to determine the sign of each input, to find outputs based on it.

Even though, there are generic comparators named as “**Universal Comparators**” having a selection input to determine the type of comparison (i.e. signed or unsigned based).

If the comparator is used without synchronous elements (Flip-Flops, Registers, ...), the output could have glitches due to changes in gates' inputs/outputs. This could

lead to real-life issues if the comparator is connected to an important system. Some glitches may be avoided if we use gates with different delays. However, designers usually prefer to use synchronous elements to avoid unsafe scenarios. The synchronous clock used can be determined by finding the maximum latency in the circuit. (*Latency*: the maximum time needed to get the stable true output).

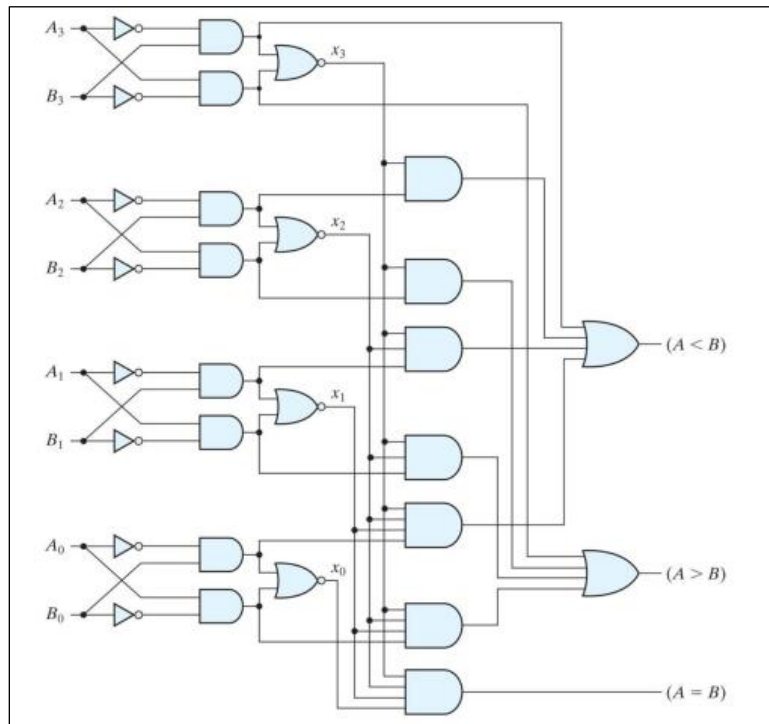


Figure 3: 4-bit Magnitude Comparator Circuit

Comparators are typically used in many different modules, such as Arithmetic and Logic Units (ALUs), hardware implementations of sorting algorithms (e.g., bubble sort, merge sort) and Digital Signal Processing (DSP) for analyzing and comparing signal values. Moreover, they could be used to implement Multiplexers (Mux) and Decoders

It is obvious that comparators are essential wherever decision-making, or value comparisons are needed in digital or analog systems.

Design philosophy

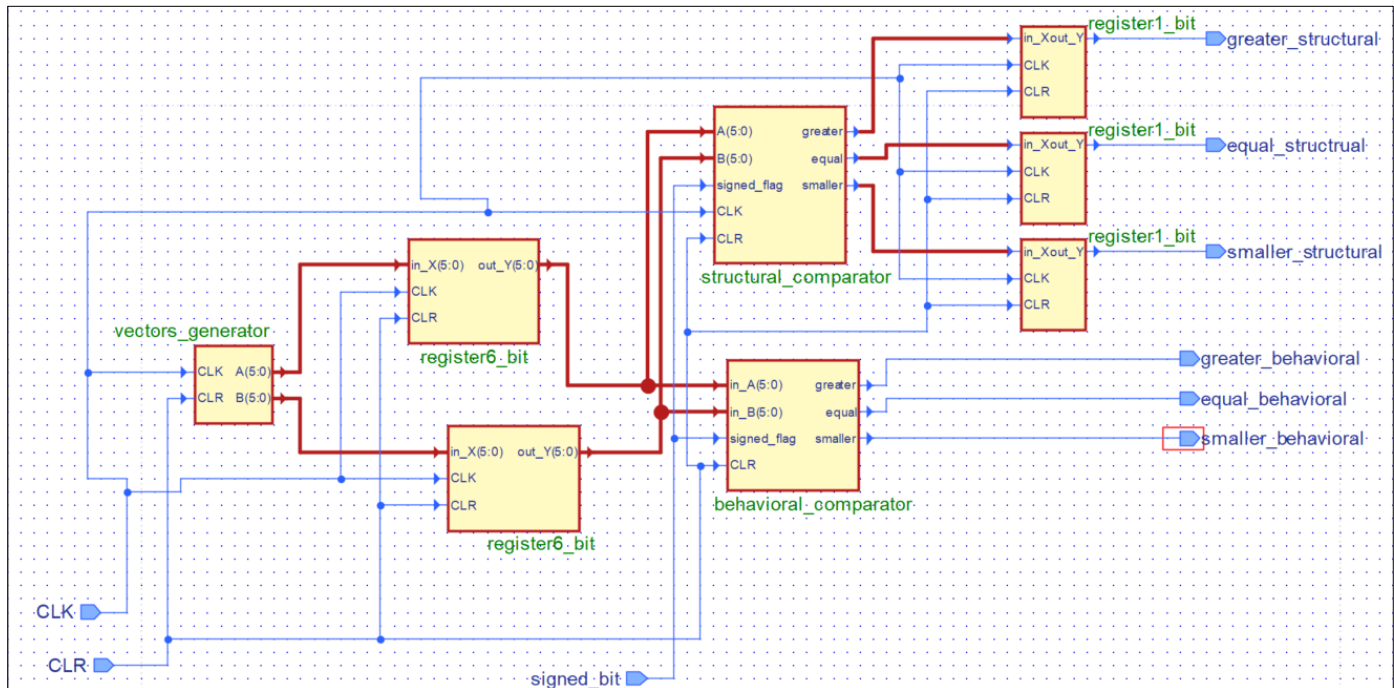


Figure 4: Block Diagram

- **No need** to add synchronized registers to the inputs/outputs of the behavioral comparator or even using a clock since that the inputs are taken from the “input registers” of the structural comparator which can’t be changed unless on the edge of the clock. The behavioral comparator has no glitchy outputs. So, the outputs will not change while the inputs were not changed (and they are changed by the clock) .

1- Universal Structural Comparator

A Top-down approach was used to implement the universal structural comparator using two separate modules (i.e. unsigned structural comparator module and signed structural comparator module).

The universal comparator has a selection line (i.e. flag) to determine whether the comparison is based on unsigned/signed numbers. Then, the appropriate (unsigned or signed) comparator will be activated (called) due to the value in the selection line then to find the output results.

Synchronous registers were used to avoid glitchy outputs (as discussed in [Theory](#)

Registers pass inputs to the comparator circuit at the rising edge of the clock. At the same rising edge, registers also return results of the comparator circuit.

Both unsigned and signed comparators were built from basic gates, with delays, to simulate a scenario like the real-life one. Comparators have two inputs to be compared with 6-bit length for each, and three outputs, greater, equal and smaller. It is obvious that the one output should be active at a time.

The **clock cycle** can be determined by finding the maximum latency of the comparator. Or, we try different values for the clock cycle until the circuit produces the expected output for all tested cases (this way was used in the project).

1.1- Unsigned Structural Comparator

All integer numbers can be represented in the binary numbering system. To compare two binary numbers, we check the most significant bit of them. A number with a larger most significant bit has the largest magnitude (value), while the other has the smallest one. If the most significant-bit in the two numbers are equal, the bit to be compared is the one having less significance (not necessarily the least one). The same approach is used until finding the largest number. If all corresponding bits are equal, then two binary numbers are equal.

❖ Given the 4-bit input numbers: A and B

1. If $A_3 > B_3$ then $GT = 1$, irrespective of the lower bits of A and B

Define: $G_3 = A_3B'_3$ ($A_3 = 1$ and $B_3 = 0$)

2. If $A_3 = B_3$ ($E_3 = 1$), we compare A_2 with B_2

Define: $G_2 = A_2B'_2$ ($A_2 = 1$ and $B_2 = 0$)

3. If $A_3 = B_3$ and $A_2 = B_2$, we compare A_1 with B_1

Define: $G_1 = A_1B'_1$ ($A_1 = 1$ and $B_1 = 0$)

4. If $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$, we compare A_0 with B_0

Define: $G_0 = A_0B'_0$ ($A_0 = 1$ and $B_0 = 0$)

❖ Therefore, $GT = G_3 + E_3G_2 + E_3E_2G_1 + E_3E_2E_1G_0$

Figure 5: Comparator's "greater" output's formula

1.2- Signed Structural Comparator

In signed comparators, inputs to be compared are represented in two's complement representation. Numbers are either positive or negative. The most significant bit in positive numbers is 0, while it is 1 for negative numbers.

To compare two signed numbers, we have four possible combinations.

- If they are both positive or both negative, we compare corresponding bits (**excluding** the most significant one) as unsigned numbers and get the results. (Two numbers can be equal).
- If they have different signs, the positive is always the largest, and the negative is always the smallest. (Two numbers can't be equal)

2- Universal Behavioral Comparator for Design Verification

To test and verify the *Universal Structural-Comparator*, a *Behavioral-Comparator* is needed to check output results. The behavioral implementation of designs is similar to high level languages' implementations. Behavioral code must be within *initial* or *always* statements. The execution is sequential within them.

3- Result Analyzer for Design Verification

The analyzer takes the expected result (from the behavioral comparator) and the current result (from the structural comparator) and checks whether they are the same. If two results are different, then the structural comparator is faulty.

4- Linear-feedback Shift Register (LFSR) as a Vector Generator

Some designs could take too long time if it is required to test all possible cases. Imagine, if we have 64 inputs, then 2^{64} cases are possible!

When too many possible cases, it is usually preferred to test random subset of possible cases. **LFSR** introduces (pseudo) random vectors which leads to high fault-coverage.

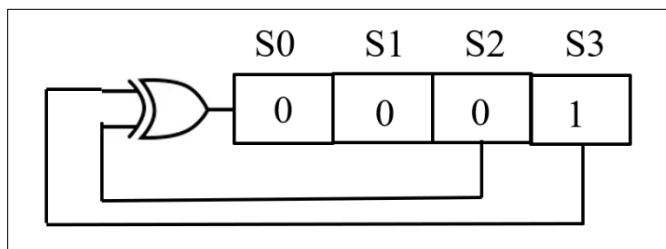


Figure 6: 4-bit LFSR

```
always @(posedge CLK, negedge CLR)
begin
    if (~CLR)begin
        //initial values
        A=6'b010101;
        B=6'b101010;
    end
    else begin
        XORed_bit_A= A[5]^A[4];
        XORed_bit_B= B[5]^B[4];
        A= {A[4:0], XORed_bit_A} ;
        B= {B[4:0], XORed_bit_B};
    end
end
```

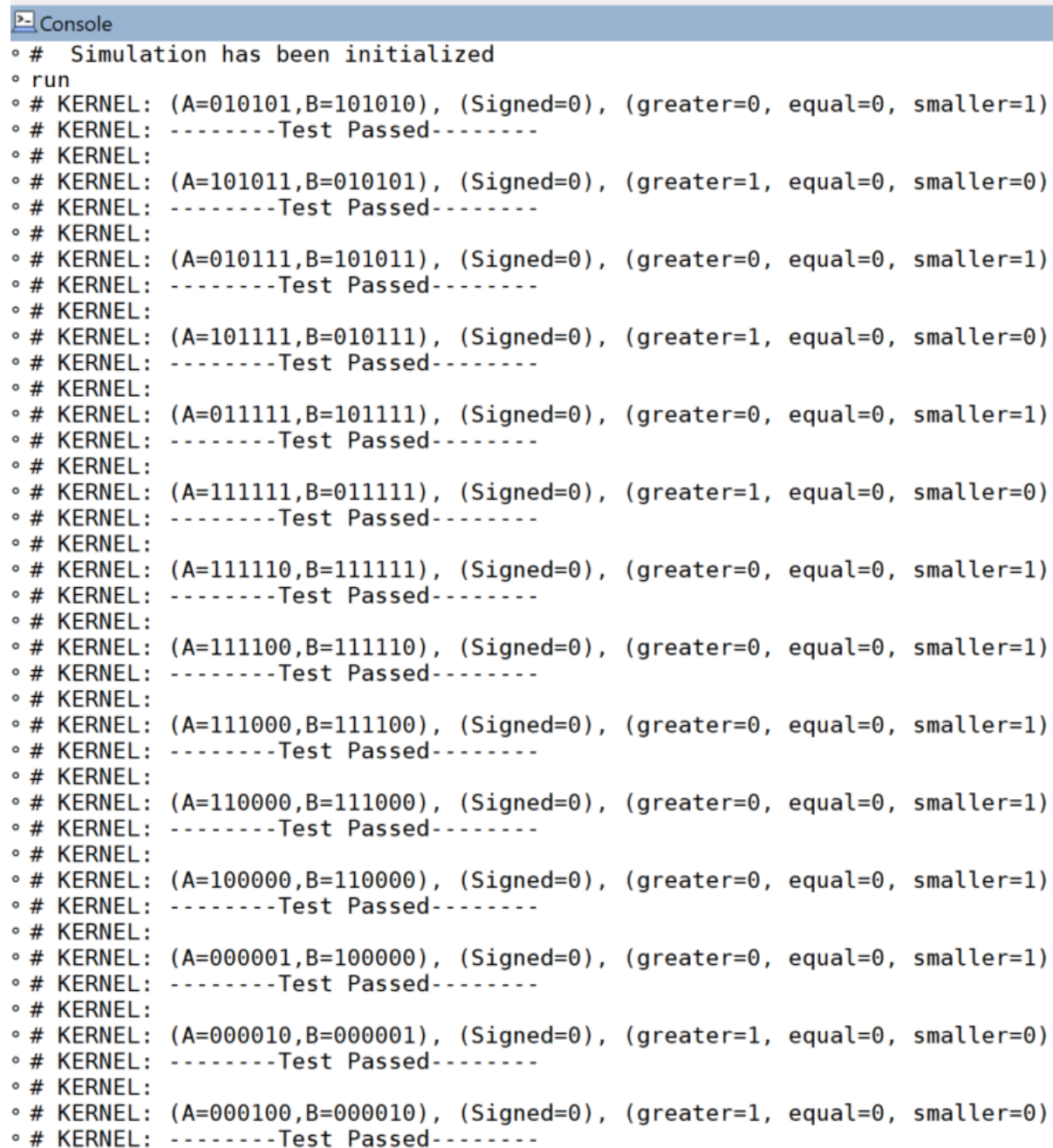
Figure 7: 6-bit LFSR in Verilog

- **LFSR** can't start with **zeros** as an initial value (i.e. seed).

Results

The design is valid for all possible inputs, and the following figures show output results for a sample of inputs.

1- Unsigned Input Numbers



```
Console
° # Simulation has been initialized
° run
° # KERNEL: (A=010101,B=101010), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=101011,B=010101), (Signed=0), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=010111,B=101011), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=101111,B=010111), (Signed=0), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=011111,B=101111), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111111,B=011111), (Signed=0), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111110,B=111111), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111100,B=111110), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111000,B=111100), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=110000,B=111000), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=100000,B=110000), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000001,B=100000), (Signed=0), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000010,B=000001), (Signed=0), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000100,B=000010), (Signed=0), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
```

Figure 8: Results of Unsigned-Inputs

2- Signed Input Numbers

```
Console
° run
° # KERNEL: (A=010101,B=101010), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=101011,B=010101), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=010111,B=101011), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=101111,B=010111), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=011111,B=101111), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111111,B=011111), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111110,B=111111), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111100,B=111110), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=111000,B=111100), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=110000,B=111000), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=100000,B=110000), (Signed=1), (greater=0, equal=0, smaller=1)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000001,B=100000), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000010,B=000001), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
° # KERNEL:
° # KERNEL: (A=000100,B=000010), (Signed=1), (greater=1, equal=0, smaller=0)
° # KERNEL: -----Test Passed-----
```

Figure 9: Results of Signed-Inputs

- Sample run without using LFSR, shows the case when two numbers are equal

```

Console
o # KERNEL:
o # KERNEL: (A=011100,B=011000), (Signed=1), (greater=1, equal=0, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=011100,B=011001), (Signed=1), (greater=1, equal=0, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=011100,B=011010), (Signed=1), (greater=1, equal=0, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=011100,B=011011), (Signed=1), (greater=1, equal=0, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=011100,B=011100), (Signed=1), (greater=0, equal=1, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=011100,B=011101), (Signed=1), (greater=0, equal=0, smaller=1)

```

Figure 10: Two inputs are equal

- An **error** was introduced in the design to find out whether the analyzer detects the error or not.

```

o # KERNEL:
o # KERNEL: (A=010101,B=101010), (Signed=1), (greater=1, equal=0, smaller=0)
o # KERNEL: -----Test Passed-----
o # KERNEL:
o # KERNEL: (A=101010,B=110101), (Signed=1),
o # KERNEL: Unexpected results!!
o # KERNEL: -----Test Failed-----
o # KERNEL:

```

Figure 11: Error Detection

Conclusion and Future Work

Comparators are essential elements in many integrated circuits (ICs). There are different ways to implement and design comparators. Optimized implementations should be chosen for better performance.

Comparators could have any length for inputs. When the length is relatively high, it leads to too many possible input cases that can't be verified all. We implemented **LFSR** to generate (sudo) random vectors to simulate real scenarios with too many possible cases. LFSR provides high fault-coverage, which enhances and eases testing and design verification.

To enhance verification, we could use “*Signature Registers*” to avoid checking the correctness of the output for each input vector separately. After all input vectors generated by LFSR are applied to the circuit, we check if the signature register has the good (expected) signature. It indicates that the circuit **mostly** has no errors.

To avoid glitches in outputs, we used synchronized registers to pass inputs/outputs to/from comparators. However, all outputs then need the same time to appear. In some cases, glitches don't occur, or the circuit finds results before the edge of the clock. The synchronized clock prevents the circuits from using those advantages to enhance performance. We may move towards choosing proper gate delays to achieve better performance without clocks or glitches.

References

(n.d.). Retrieved from geeks for geeks: https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/?ref=header_search