Lab 2: Comparators

By Si Mai

CSC 342/343 Computer Organization

Professor Izidor Gertner

Spring 2017

# Objective

# Functionality and Specification

## One-Bit Equal Comparator

The purpose of the one-bit equal comparator is to compare two bits. If the two bits are both either 0 or 1, then the output is 1, which indicates that they are both equal. Otherwise, the output is 0 and they are not equal. Table I shows the truth table for the one-bit equal comparator. The EqualNot column is the complement of the Equal column.

Table I. Truth Table for One Bit Equal Comparator.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| I0 | **I1** | **Equal** | **EqualNot** |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Using the truth table in Table 1, we can extract the switching function for the output Equal:

(1)

Fig. 1 shows the circuit implementation of the one-bit equal comparator using the equation above.

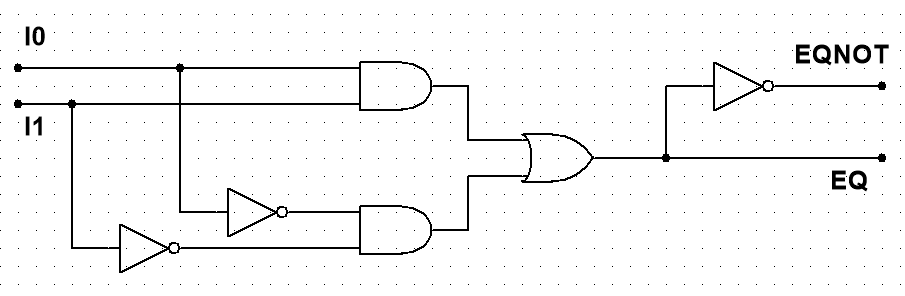


Fig. 1. Circuit diagram of the one-bit equal comparator.

Next, we can write the VHDL (which stands for **V**ery High-Speed Integrated Circuit **H**ardware **D**escription **L**anguage) for the one-bit equal comparator. The code uses a data-flow style architecture, which means that the one-bit equal comparator is implemented based on how data flows from the input signals to the output signals. By comparing either the truth table, switching function, or circuit diagram, we can see that the VHDL code mimics the behavior of them. Intermediate signals p1 and p0 (on lines 19 and 20) are the outputs of the top and bottom AND gates in the circuit diagram respectively. These signals are used as inputs to the OR gate and determines the output of Equal and EqualNot.

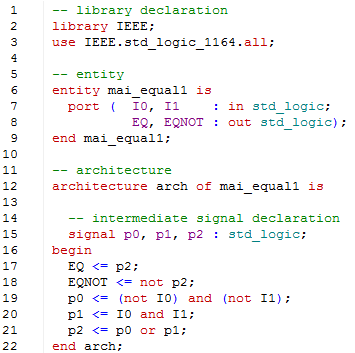


Fig. 2. VHDL code for one-bit equal comparator.

VHDL code consists of the library declaration, entity, and architecture. The library declaration allows us to use certain data types, such as std\_logic. The entity declares what the inputs and outputs to the circuit. The architecture is the implementation of the circuit. In C, the entity and architecture equivalents are the interfaces and implementations.

## Two-Bit Equal Comparator

The two-bit equal comparator takes two two-bit binary numbers and compares them. The corresponding 1’s and 2’s places in each binary number are compared. If they are both equal, then the two binary numbers are equal. Otherwise they are not equal. Like the 1 bit binary number, the outputs for Equal and EqualsNot are 1 and 0 respectively. Table II shows the inputs and outputs of the two-bit equal comparator. The inputs are A and B and the outputs are EQ and EQNOT. A0 and A1 corresponds to the 1s and 2s place of the binary number. The same holds for B0 and B1.

Table II. Truth Table for Two-Bit Equal Comparator.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| B1 | **B0** | **A1** | **A0** | **EQ** | **EQNOT** |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

From the truth table we can obtain the switching function for the output EQ:

(2)

The switching function is a canonical sum of the four bits B1, B0, A1, and A0. We can map this switching function directly on our circuit diagram, which is shown on Fig. 3. Moreover, we can model the data flow architecture using the Eq. 2. The VHDL using the gates is show on Fig. 4. Instead of explicitly labeling related groups of bits, such as A1 and A0, we use the data type std\_logic\_vector to indicate that the variable A is a vector of related bits. The statement 1 downto 0 specifies that the left most bit and right most bit are the most and least significant bits respectively. To access the individual bits, we call the variable and enclose the index with parenthesis. Thus A(1) returns the bit in the 2s place.

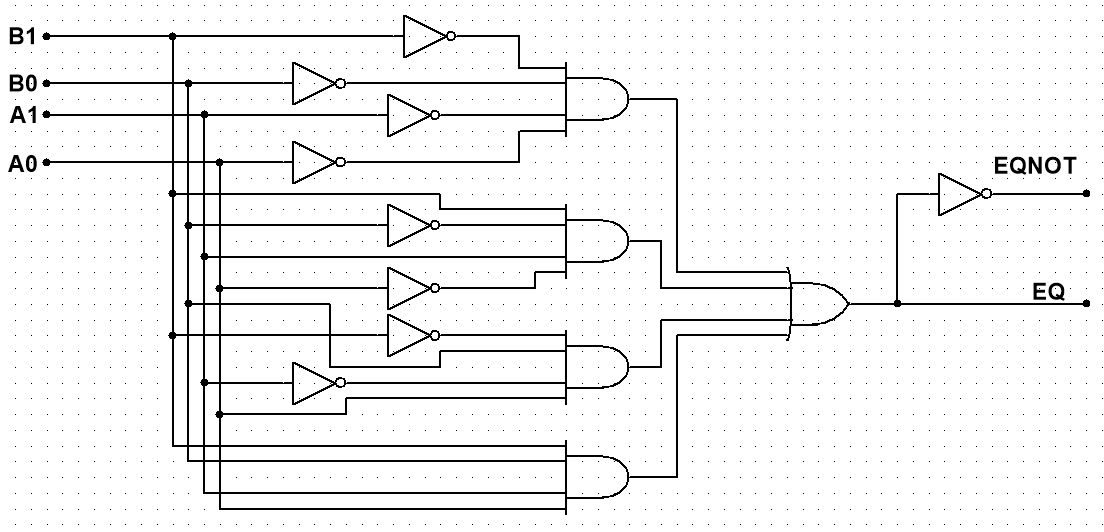


Fig. 3. Circuit diagram for the two-bit equal comparator.

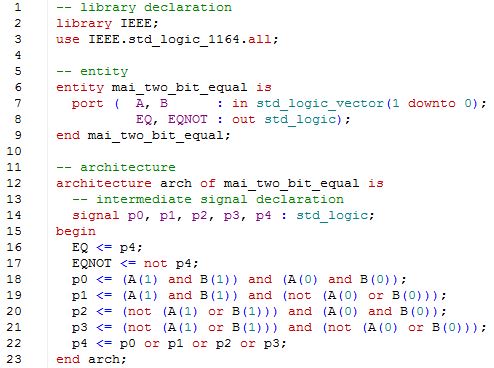


Fig. 4. VHDL code for two-bit equal comparator. The code models the data flow architecture using only gates.

When writing VHDL architecture, there are three different approaches: the data-flow, structural, and behavioral. The structural style describes the modular design of VHDL. In Fig. 4, the implementation of the two-bit equal comparator was written from scratch. An alternate way of writing the architecture is to notice that the two-bit equal comparator consists of 2 one-bit equal comparators. Therefore, we do not have to write many lines of code to get the same results. Fig. 5 shows the architecture implemented using structural modeling. Before the begin statement on line 24, we declare the one-bit equal comparator (on line 16) so that we can use it. The architecture in Fig. 5 is easier to read and less prone to error than the one in Fig. 4. Two one-bit equal comparators are used (on line 25 and 26). The actual mapping of the inputs and outputs follows the keyword port map. The approach shown below is known as direct mapping. Each signal in the component is mapped to either the entity or intermediate signals. This approach leaves little room for ambiguity and allows the signal to be listed in any order.

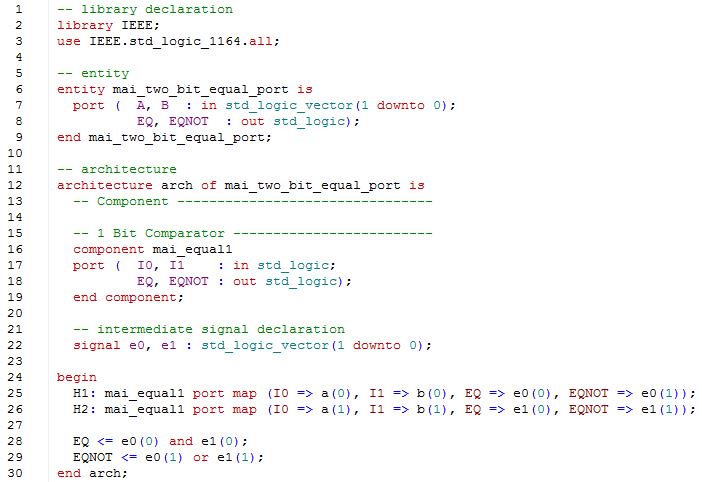


Fig. 5. Architecture of Two-Bit Equal Comparator using Structural Modeling.

The other approach is known as implied mapping. The mapping of the external signals to the signals of the components is done through the order that they appear. This leaves less source code but requires that the signals be placed correctly.

## Eight-Bit Equal Comparator

The eight-bit equal comparator takes 2 eight-bit binary numbers and compares them for equality. Unlike the one-bit and two-bit equal comparator case, we will not create a truth table for the eight-bit equal comparator. A truth table with 16 bits would result in 65536 rows, which would take too long to complete. Moreover, the circuit diagram would be cluttered with many gates and wires. Consequently, writing the VHDL code for the eight-bit equal comparator using data-flow modeling would be lengthy and prone to errors. Fig. 6 shows the implementation using structural modeling.

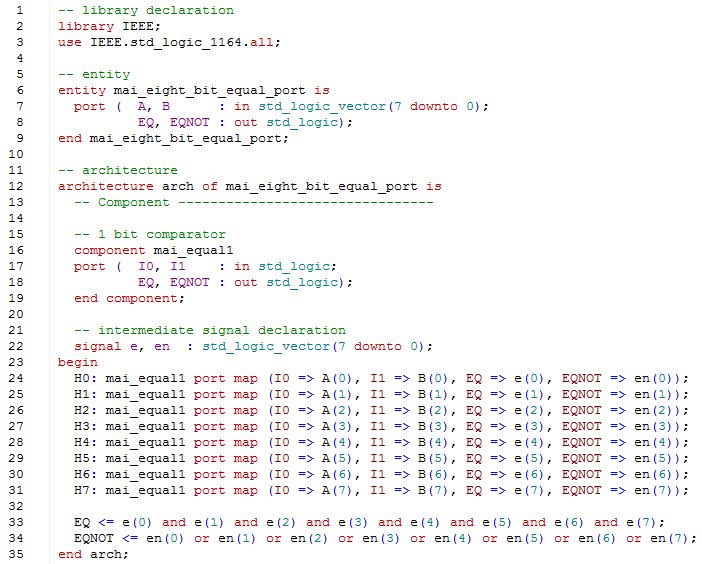


Fig. 6. Architecture of Eight-Bit Equal Comparator using Structural Modeling.

The code is similar to the two-bit equal comparator with structural modeling, except we have added 6 more one-bit equal comparators, and adjusted the external and intermediate signals to hold eight bit vectors instead of two.

# Simulation

## One-Bit Equal Comparator

In order to test the correctness of the code, we simulate the possible outcomes based on two bit inputs. Fig. 7 shows the waveform of the four possible input combinations and the corresponding output EQ and EQNOT. The simulation was done manually by forcing each input and running for 100 ns.

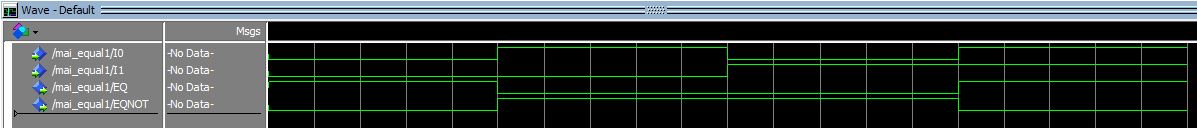


Fig. 7. Manual testing of the one-bit equal comparator.

While testing manually for the one-bit equal comparator was simple, changing the inputs one by one becomes tedious when we increase the number of bits to be tested. We therefore use test benches check the correctness of the VHDL code. Test benches are VHDL files that are written to test components. The test benches are shown in the appendix. Fig. 8 shows the results of the test bench used to test the one-bit equal comparator. P0, P1, Pout, Poutc are the intermediate signals that correspond to the external signals I0, I1, EQ, and EQNOT. The error signal is 1 if there was an error and 0 otherwise. As we can see in Fig. 8, the error signal is 0 throughout the test, which indicates that the one-bit equal comparator coded is correct.

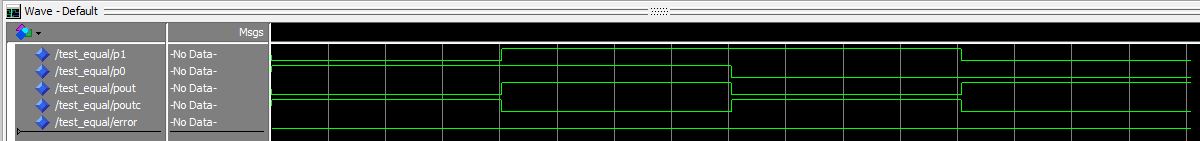


Fig. 8. Testing the one-bit equal comparator with a test bench.

## Two-Bit Equal Comparator

With the two-bit equal comparator, we used both the data-flow and structural model architecture. Fig. 9 shows the waveform of the test simulation on the data-flow implementation. This time P0 and P1 are bit vectors of size 2. The test bench uses 8 cases; the figure below shows that the error signal is 0 throughout, indicating that the data-flow architecture works as intended.

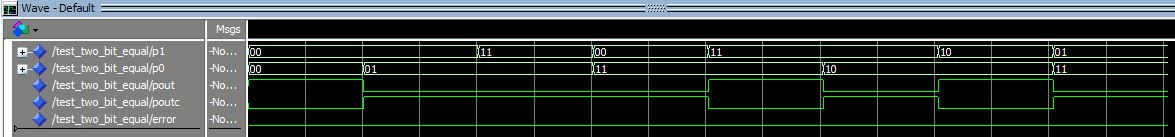


Fig. 9. Testing the data-flow model with test bench.

Fig. 10 shows the waveform of the test simulation on the structural implementation. The test file was adjusted for the structural model implementation by changing the names of the components. Again the error signal is 0 throughout the test, which shows that the structural model is correct.

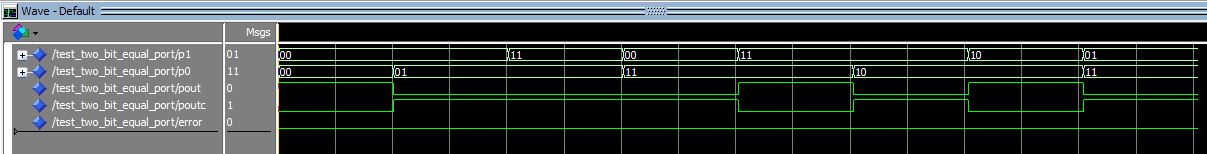


Fig. 10. Testing the structural model with test bench.

Since we only used eight cases, it could be possible that error could occur in the other eight cases. Fig. 11 shows the manual test of the structural model of all the possible 16 cases. Comparison of the truth

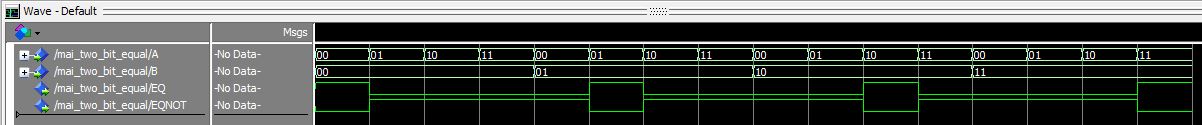


Fig. 11. Manual testing of the structural model.

table with the output in fig. 11 shows that the structural model works correctly.

## Eight-Bit Equal Comparator

The eight-bit equal comparator is tested for correctness using a test bench. The test bench also uses eight test cases to determine the correctness of the code. P0 and P1 are vector signals of eight bits. The pout and poutc are the EQ and EQNOT signals of one bit. They are named this way because they are intermediate signals since output signals cannot be used as inputs. The error signal is 0 throughout the test, so it indicates that the code is correct. However, it is possible that the code is incorrect because not all the possible combinations were tested.

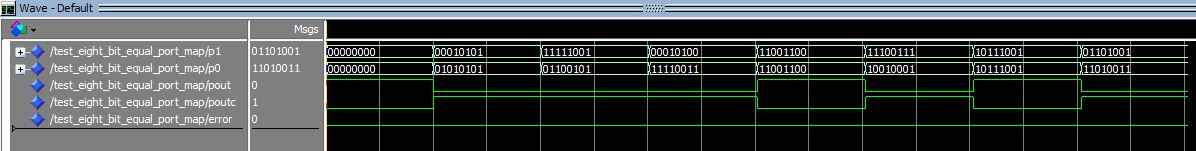


Fig. 12. Testing the eight-bit equal comparator (implemented with structural model) with a test bench.

# Implementation

## One-Bit Equal Comparator