How Do You Compare Unsigned Numbers?

Let’s develop a bit-sliced design to compare two unsigned numbers. Which 8-bit unsigned number is bigger?

0 1 1 0 1 0 0 0
0 1 0 1 0 1 1 1

How did you know?
Did you start on the left or the right?

Humans Go from Left to Right

Usually, humans start on the left. Why?
As soon as we notice a difference, we’re done!

humans compare this way
0 1 1 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
0 1 0 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

Bit-sliced hardware cannot stop in the middle. The information flows from one end to the other. Output wires produce the answer (in bits).

Our Design Compares from Right to Left

Our comparator design will start on the right.

humans compare this way
a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

our design will compare this way
From least significant to most significant bit.
Three Possible Answers for Comparison of A and B

When comparing two numbers, A and B, we have three possible outcomes:
- A < B
- A = B
- A > B

To decide the answer for N+1 bits, we need:
- the answer for N (less significant) bits,
- one bit of A, and
- one bit of B.

An Abstract Model of the Comparator Bit Slice

A question for you:
How many bits must pass between slices?
Two!

This figure shows an abstract model of our bit slice.

We Need a Representation for Answers

Another question for you:
How do we represent the three possible answers?

Any way we want!

<table>
<thead>
<tr>
<th>C1</th>
<th>C0</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A = B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>not used</td>
</tr>
</tbody>
</table>

A Single Bit Requires Two Minterms on A, B

Let’s start by solving a single bit.

In this case, there are no less significant bits.

So we consider only A and B.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z1</th>
<th>Z0</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A = B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A = B</td>
</tr>
</tbody>
</table>

Note that Z1 and Z0 are minterms.
Comparing Two Bits is Fairly Easy

An implementation for a single bit appears below. This structure forms the core of our bit slice, since it compares one bit of \(A\) with one bit of \(B\).

When A and B are Equal, Pass Along the Answer

Now for the full problem. We’ll start with the case of \(A = 0\) and \(B = 0\).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_1</th>
<th>C_0</th>
<th>meaning</th>
<th>Z_1</th>
<th>Z_0</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(A = B)</td>
<td>0</td>
<td>0</td>
<td>(A = B)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A &lt; B)</td>
<td>0</td>
<td>1</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(A &gt; B)</td>
<td>1</td>
<td>0</td>
<td>(A &gt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>??</td>
<td>don’t care</td>
<td>x</td>
<td>x</td>
<td>don’t care</td>
</tr>
</tbody>
</table>

When A and B are Equal, Pass Along the Answer

Is there any difference when \(A = 1\) and \(B = 1\)?
No, outputs are the same as the last case.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_1</th>
<th>C_0</th>
<th>meaning</th>
<th>Z_1</th>
<th>Z_0</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(A = B)</td>
<td>0</td>
<td>0</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(A &lt; B)</td>
<td>0</td>
<td>1</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A &gt; B)</td>
<td>1</td>
<td>0</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>??</td>
<td>don’t care</td>
<td>x</td>
<td>x</td>
<td>don’t care</td>
</tr>
</tbody>
</table>

When A and B Differ, Override the Previous Answer

What about case of \(A = 0\) and \(B = 1\)?
Always output \(A < B\) (for valid inputs).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_1</th>
<th>C_0</th>
<th>meaning</th>
<th>Z_1</th>
<th>Z_0</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(A = B)</td>
<td>0</td>
<td>0</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(A &lt; B)</td>
<td>0</td>
<td>1</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A &gt; B)</td>
<td>1</td>
<td>0</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>??</td>
<td>don’t care</td>
<td>x</td>
<td>x</td>
<td>don’t care</td>
</tr>
</tbody>
</table>
When A and B Differ, Override the Previous Answer

And the case of \( A = 1 \) and \( B = 0 \)?

Always output \( A > B \) (for valid inputs).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_1</th>
<th>C_0</th>
<th></th>
<th>Z_1</th>
<th>Z_0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td>( A = B )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>( A &lt; B )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td>( A &gt; B )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>x</td>
<td>x</td>
<td>don’t care</td>
</tr>
</tbody>
</table>

\[ Z_1 \text{ is a Majority Function} \]

Let’s use a K-map to solve \( Z_1 \).

What are the loops?

\[
\begin{array}{c|c|c|c|c}
\text{AB} & \text{Z_1} \\
00 & 0 & 0 & 1 \\
01 & 0 & 0 & 1 \\
10 & x & x & x \\
11 & 1 & 0 & 1 \\
\end{array}
\]

So

\[ Z_1 = AB' + AC_1 + B'C_1 \]

\[ Z_0 \text{ is Also a Majority Function} \]

Now let’s use a K-map to solve \( Z_0 \).

What are the loops?

\[
\begin{array}{c|c|c|c|c}
\text{AB} & \text{Z_0} \\
00 & 0 & 1 & 0 & 0 \\
01 & 1 & 1 & 0 & 0 \\
10 & x & x & x & x \\
11 & 0 & 1 & 0 & 0 \\
\end{array}
\]

So

\[ Z_0 = A'B + A'C_0 + BC_0 \]

\[ Z_0 = \text{Full Implementation as SOP Expressions} \]

\[ Z_1 = AB' + AC_1 + B'C_1 \]

\[ Z_0 = A'B + A'C_0 + BC_0 \]

Notice the symmetry.

And the single-bit core.