Two integers $x, y$ are compared and result is $z$.

\[ y < x \]

\[ a = \begin{cases} 
1 & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]

\[ x = \begin{cases} 
2z & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]

\[ y = \begin{cases} 
2z & \text{if } y > x \\
0 & \text{if } y = x \\
1 & \text{if } y < x 
\end{cases} \]

\[ z \]

\[ a = \begin{cases} 
1 & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]

\[ a = \begin{cases} 
1 & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]

\[ a = \begin{cases} 
1 & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]

\[ a = \begin{cases} 
1 & \text{if } x > y \\
0 & \text{if } x = y \\
1 & \text{if } x < y 
\end{cases} \]
\[ E_0 = x_1 y_0 ar{y}_1 \bar{y}_0 + x_1 y_1 y_0 + x_1 y_0 y_0 + \bar{x}_0 \bar{y}_0 \]

\[ = x_1 (y_1 + \bar{y}_0 \bar{y}_1) + \bar{x}_0 y_0 \]

\[ = x_1 (y_1 + \bar{x}_0 y_0) + \bar{x}_0 y_0 \]

\[ = x_1 \bar{y}_1 + x_0 \bar{y}_0 + \bar{x}_0 y_0 \]

\[ Z = x_1 \bar{y}_1 \]

Assume inverse input is available.

\[ E_1 = x_1 \bar{y}_1 \bar{x}_0 \bar{y}_0 + x_0 \bar{y}_0 \bar{y}_1 \]

\[ = (x_1 \bar{y}_1)(\bar{x}_0 \bar{y}_0)(\bar{x}_1 \bar{y}_0) \]

To use XOR and AND gates.
\[ \begin{align*}
\text{I: } & \omega \bar{y}z + \omega x \bar{y} + \omega yz + \omega xy + [\omega yz] \\
& \text{Non-EPI} \\
\end{align*} \]
The PLA has a bigger network size due to its large grid network that is cheap to make, but takes up a lot of space. Also, the normal gate network can simplify gates into smaller ones like the XOR gate. The PLA is also done due to its inflexibility. The PLA is faster than the PLA, not really! in a gate network. The propagation delay is greater due to the above. A possibility to simplify gates further like the XOR and PLA can be used again and again for other circuits. While gate networks can't PLA only apply to simple circuits (2-layer) while other can handle more. PLA is more convenient while gate network is more efficient.
(4) abc def gh

\[ E(abc \text{ def gh}) = \text{a0b0c0d0e0f0g0h} \]
\[ E(abc \text{ def g}) = \text{a0b0c0d0e0f0g0} \]
\[ E(abc \text{ def g}) = \text{a0b0c0d0e0f0g0} \]

Simplify until shown on back:
\[ E(abc \text{ def g}) = \text{a0b0} \]
\[ E(a \ldots f 11) \]

\[ E(a \ldots f 01) \]

\[ E(a b c d e f 0 0) \]
\[ E(a b c d e f 1 0) \]
\[ E(a b c d e f 0 0) \]
\[ E(a b c d e f 1 0) \]
\[ E(a b c d e 0 0) \]
\[ E(a b c d e 0 1) \]
\[ E(a b c d 0 0) \]
\[ E(a b c d 0 1) \]
\[ E(a b c 1 0 0) \]
\[ E(a b c 1 0 1) \]
\[ E(a b 0 0 0) \]
\[ E(a b 0 0 1) \]

\[ E(0 b 0 0 0 1 1) = a b 0 0 0 \]
\[ E(0 b 0 0 1 1 1) = a b 0 0 1 \]
\[ E(0 b 0 1 1 1 1) = a b 0 1 \]

\[ a b 0 0 = 64 \text{ combinations of } b \Rightarrow h \]
\[ a b 0 1 = 64 \text{ combinations of } b \Rightarrow h \]
\[ \frac{128 \text{ combinations total}}{\text{total}} \]