There are many possible solutions – we present 3 plausible ones. There are two preliminary points which are important in all solutions:

- **s=1.** Since this is a combinational circuit, not a sequential one, we do not need the flip flop. In all cases, the output multiplexor should pass on the signal which bypasses the flip flop (i.e. s=1).

- **Decompose the 8-input OR.** A single CLB cannot possibly implement the 8-input OR which generates $y$. You must decompose that OR gate. This is easy, since OR is associative: $a+b+c = (a+b)+c = a+(b+c)$. That means a wide OR can be implemented by a sequence of smaller ORs as well as by a tree (as in Homework #2, problem 2, which did the same for a wide XOR). To minimize total CLB usage, the particular OR decomposition you choose should depend on the partitioning of the rest of the circuit.

Recall that this is our CLB:

![Configurable Logic Block](image)

The easiest way to use the CLB is as a 4-LUT. We compose the 4-LUT from the pair of 3-LUTs by feeding the same 3 inputs into \{b,c,d\} as into \{e,f,g\} and by using $a$ for the fourth input. The first 3-LUT implements the 4-LUT function assuming $a=0$ while the second assumes $a=1$. This technique is demonstrated in Homework #3, problem 3.
A very simple partitioning can be done using 11 4-LUTs. Each of the 8 AND-AND terms uses one CLB, and the OR is partitioned into a tree of 3 CLBs.

Implementation in 11 CLBs (4-LUT configuration).
The following partition in 5 CLBs is due to Drew Pertula. The 8 AND-AND terms can be grouped into 4 pairs, where the difference between each pair is the inversion of the x2 input. This guarantees that, in any such pair, one AND-AND term is forced to zero. Since the AND-AND terms are subsequently OR-ed, the value of x2 effectively selects which among each pair of AND-AND terms will pass to the output. Now we can pack each pair of OR-ed AND-AND terms into a single CLB – each AND-AND (now without x2 as an input) gets a 3-LUT, and x2 controls the multiplexer to select one of them. This requires 4 CLBs (2 AND-ANDs in each), and one additional CLB to OR their outputs (using a 4-LUT configuration). Total: 5 CLBs.

![Implementation in 5 CLBs.](image-url)
We can do one better – 3 CLBs. You must first realize that the circuit is an 8-to-1 multiplexor. It chooses one of \( \{x_3, \ldots, x_{10}\} \) according to the select bus \( \{x_2, x_1, x_0\} \). The first column of ANDs implements a decoder, converting the binary number \( \{x_2, x_1, x_0\} \) into a one-hot representation \( \{t_1, \ldots, t_8\} \) – only one of \( \{t_1, \ldots, t_8\} \) will be 1, the rest 0. The second column masks out the unselected inputs from \( \{x_3, \ldots, x_{10}\} \) by AND-ing them with 0 – only one of \( \{t_9, \ldots, t_{16}\} \) will actually copy an \( x \) input, the rest will be 0. The final OR simply passes-on whichever result was selected.

An 8-to-1 multiplexor can be implemented by a tree of 2-to-1 multiplexors. A 2-to-1 multiplexor fits in a 3-LUT. A CLB combines 2 2-to-1 muxes into a 4-to-1 mux. A pair of such CLBs fed through a 2-to-1 multiplexor in a third CLB forms an 8-to-1 mux.

![Diagram of 8-to-1 multiplexor implementation in 3 CLBs]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>s</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>x_3</td>
<td>x_4</td>
<td>x_2</td>
<td>x_5</td>
<td>x_6</td>
<td>x_2</td>
<td>1</td>
<td>t_4</td>
</tr>
<tr>
<td>x_1</td>
<td>x_7</td>
<td>x_8</td>
<td>x_2</td>
<td>x_9</td>
<td>x_{10}</td>
<td>x_2</td>
<td>1</td>
<td>t_5</td>
</tr>
<tr>
<td>0</td>
<td>t_4</td>
<td>t_5</td>
<td>x_0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>y</td>
</tr>
</tbody>
</table>

Implementation in 3 CLBs.