8.2. Assumption: "two pipeline stages" = four latches.

Factor F: 
\[ F = (AB + BC) + D \]
\[ = (A + C) + D \]
\[ = A + C + D \]

First attempt at gate level design:

Wrong! Latches invert.

Solution: work backwards.

Static design:
- need to have an even # of inversions between latches.
  (see p 358 Fig)
- Static check logic

\[ F = (AB + BC) + D = (A + C) + D \]

\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
\[ \overline{B(A + C) + D} \]
Final design:

Note: is this really any faster than:

Dynamic logic implementation:

- need to be careful with any static inversions
  (e.g. domino logic inverter). We can forget about
  this constraint if we just do the whole thing
  with only dynamic gates. (See p. 359, 876)

- our logic function is non-inverting, and we have
  4 stages of inverting latches, so the logic between
  the latches must be non-inverting (when taken together)

- a single dynamic stage also inverts.
example:

\[ \text{\includegraphics[width=0.5\textwidth]{example-diagram.png}} \]

the total # of stages we allocate to \( F, G, \) and \( H \) must be even.

so we could have one stage implementing \( F \) and
one stage implementing \( G \) but then we would need
an even # of stages implementing \( H \), etc...

note: np-eoms doesn't force you to chain a p-block after
an n-block, i.e. we can have this:

\[ \text{\includegraphics[width=0.5\textwidth]{chain-diagram.png}} \]

\( q \) 0 \( \Rightarrow \) hold
\( q \) 0 \( \Rightarrow \) prec.
\( p-1 \) eval  eval

on a \( q=1, \overline{q}=1 \) overlap,
\( F \) may get pulled low but this
is normal evaluation behavior.

on a \( q=0, \overline{q}=0 \) overlap,
the places in the field are
\( H1 \) is on but \( H2 \) is off \( \Rightarrow \)
no race problems.

same technique: work backwards, but this time we don't
have to worry about the static inversion constraint.

\[ \text{\includegraphics[width=0.5\textwidth]{static-diagram.png}} \]

notice if we are allowed to use \( \overline{A}, \overline{C} \) as inputs the
speed would improve further (since we can then eliminate the
two inverts in the first stage).
Static: $t_{\text{max}} = t_{\text{clock}} + t_{\text{data}} + t_{\text{pin}} + t_{\text{prop}} = \frac{T}{2}$

dynamic: $t_{\text{max}} = t_{\text{clock}} + t_{\text{data}} + t_{\text{pin}} + t_{\text{prop}} = \frac{T}{2}$

They are the same, but this is misleading for at least two reasons:

1) Might be able to use $A$, $C$ can eliminate $t_{\text{pin}}$

2) Dynamic nor gate much faster than static nor gate.

Thanks to Engling Tea for helping with this problem!
### 8.3 Timing

a) Fill in the missing clock connections on the schematics (in the areas marked by the boxes), so that the circuit will operate correctly, and will have the maximum throughput. For each connection you may choose between \( \phi \) or \( \phi' \), which are the two phases of an ideal clock with no skew.

b) Determine for each stage in the figure the function and/or circuit type. You may choose between flip-flop, latch, static gate, dynamic gate, or others. For each stage in the circuit, also fill in the correct operating mode for each clock phase, as given in the table (choose between precharge, evaluate, hold, transparent).

<table>
<thead>
<tr>
<th>Circuit Type</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td>latch</td>
<td>not</td>
<td>prech.</td>
<td>hold</td>
</tr>
<tr>
<td>( \phi' )</td>
<td>transp.</td>
<td>prech.</td>
<td>prech.</td>
<td>hold</td>
</tr>
<tr>
<td></td>
<td>hold</td>
<td>eval</td>
<td>eval</td>
<td>transp.</td>
</tr>
</tbody>
</table>

http://bwrc.eecs.berkeley.edu/Classes/icdesign/ee141_s00/Homeworks/ee141hw8.htm

3/22/2000