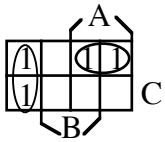
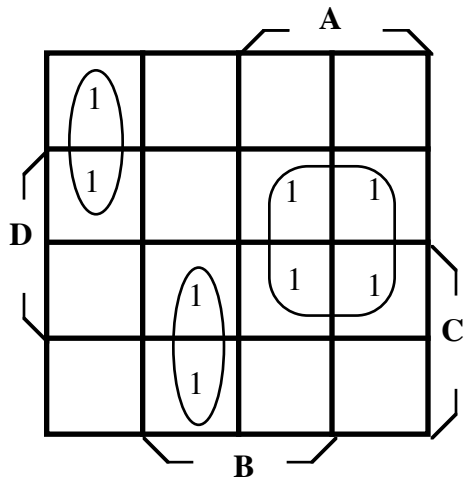
HW #7 SOLUTIONS

NT-1. For each of the following two expressions, identify any static combinational hazards that exist in the corresponding 2-stage logic circuits, assuming SIC operation.
(a) $Z=\bar{A}\bar{B}+A\bar{C}$
(b) $Z=(\bar{A}+\bar{B}+\bar{C})(\bar{A}+B+C)(A+D)$
(c) $Z=A\bar{B}+\bar{A}\bar{C}D+A\bar{D}+AC$

(a) $Z=\bar{A}\bar{B}+A\bar{C}$ From the algebraic expression, we can see that the two terms have a consensus, $\bar{B}\bar{C}$, which is not part of the expression . This means that there is a static 1-hazard in the region corresponding to this term, which is where B=C=0, when A changes in either direction. We could also see this from the K-map with the p-terms of the expression ovalled, as below. Clearly, the adjacent 1-points 000 and 100 are in different ovals, and no one oval includes both, so that moving between these points entails a 1-hazard.



(b) $Z=(\bar{A}+\bar{B}+\bar{C})(\bar{A}+B+C)(A+D)$. This is a POS expression, so the hazards we must be concerned with are static 0-hazards. One approach is to specialize (set) values of the variables other than A to reduce the expression to AA†. This can be done in two ways. In both cases, in order to preserve the one A-term, we must set D=0. Then to capture the leftmost $\bar{A}$-term, we set B=C=1. This identifies hazardous transitions between 0110 and 1110. Then we focus on the second $\bar{A}$-term, by setting B=C=0, to obtain the pair of points 0000 and 1000.
An alternative approach is to map the DUAL expression, which is, of course, a SOP expression. Then we can use the map method to identify 1-hazards, which will correspond to 0-hazards for the original expression, but at the dual points. This map is shown below:

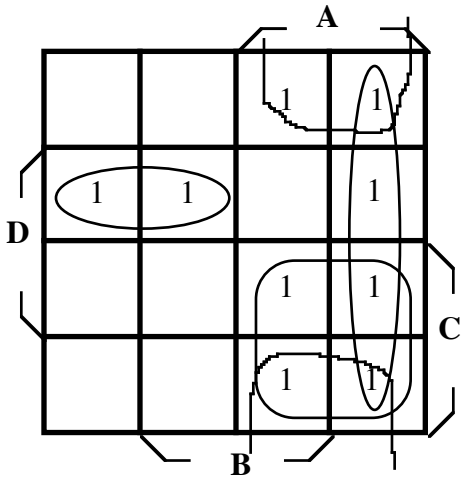


The 1-hazards for the mapped expression are between 0001 and 1001, and between 0111 and 1111. The DUAL pairs, are (map 0 to 1 and 1 to 0) are 1110 and 0110 (the first pair) and 1000 and 0000 (the second pair).

(c) $Z=A\bar{B}+\bar{A}\bar{C}D+A\bar{D}+AC$

We might notice the missing consensus term (between the first two terms), namely $\bar{B}\bar{C}D$, indicating as 1-hazard involving an A-change between 0001 and 1001.

The K-map is shown below:



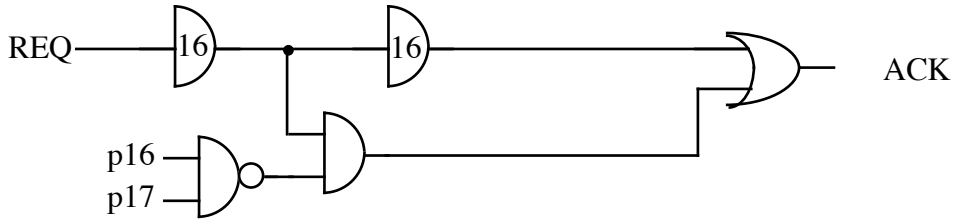
The only 1-hazard is between 0001 and 1001, i.e., with A changing while B=C=0, and D=1.

NT-2. Design 32-bit speculative completion adders meeting each of the following specifications. Assume the body of the adder is of the ripple-carry type and that there is 1 unit of delay for each adder stage.

(a) The completion signal is generated with delay 32 (worst case) or with delay 17. Use 2 p-signals.

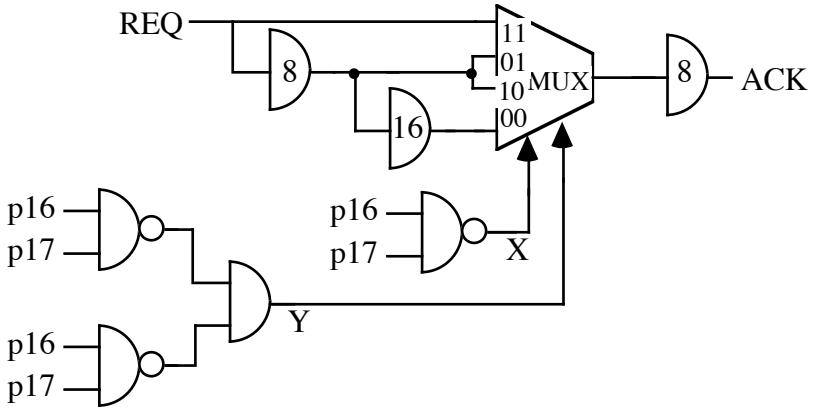
If input pair 16 is NOT a propagator (p16=0) then the longest carry ripple is from 17 thru 32 (length 16). If input pair 17 is NOT a propagator (p17=0) then the longest carry ripple is from 1 thru 16 (length 16). So, if p16=0 or p17=0, no carry ripple can be longer than 16. Then we can use the signal $\overline{p16p17}$ to set the completion signal delay to 16. (We can do a little better than the 17 in the problem statement.)

$p16=A16\oplus B16$ and $p17=A17\oplus B17$. Note that both of these signals can be used to generate sum bits S16 and S17, so generating them for the completion signal doesn't cost anything extra. The circuit for the acknowledge signal is shown below. We neglect delays in the logic gates, since these can be absorbed in the delay elements.



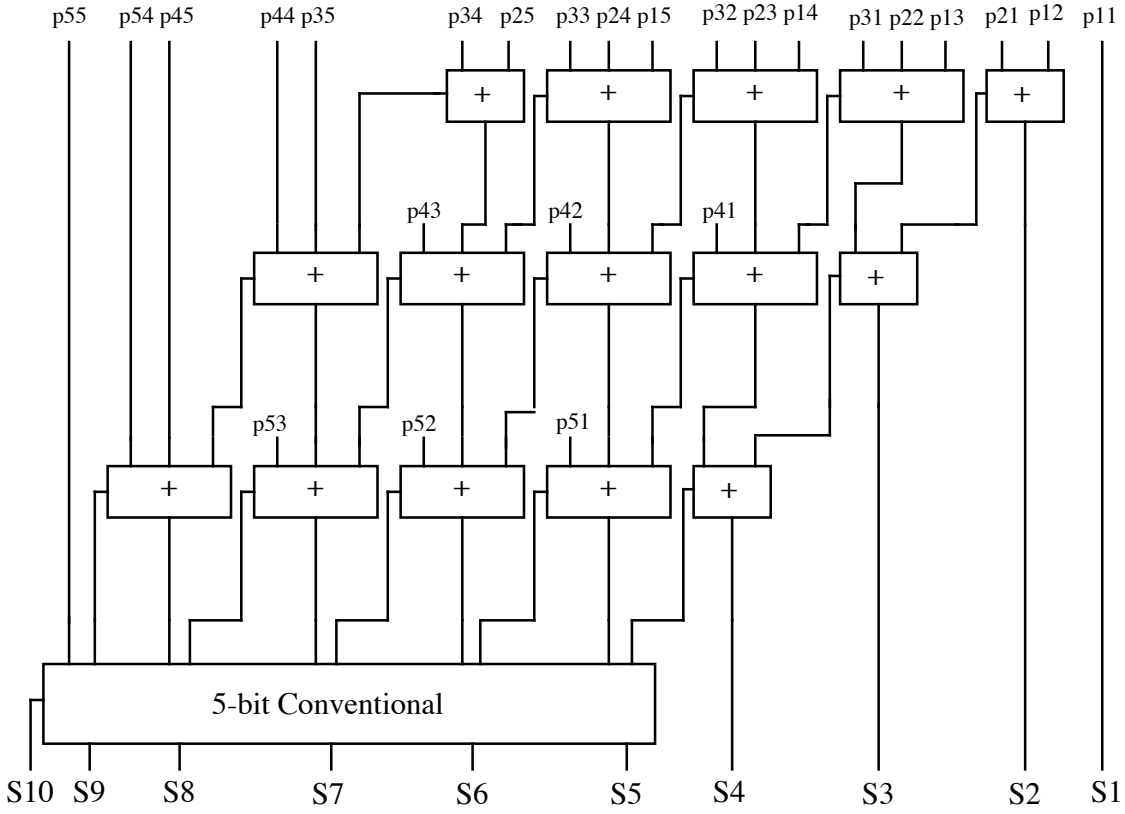
The adder is an ordinary ripple-carry adder. Assume that the adder inputs are valid at the time that the REQ signal is turned on, in which case the adder outputs will be valid as soon as ACK goes on.

(b) The completion signal is generated with delay 32, or 17, or 12. Use 2 p-signals to control the delay of 17, and 4 p-signals to control the delay of 12.
Use p16 and p17 as in part-a to generate a signal that cuts 16 units off the delay.
Now suppose that p8 or p9 =0, that p16 or p17 =0, and that p24 or p25 =0. Then the longest possible carry chain cannot exceed 8. For example, suppose p8=p17=p25=0 and all other p's=1. Then the carry chain lengths, starting at cell-1, are 7 (ending at 7), 8 (starting at 9 and ending at 16), 7 (starting at 18 and ending at 24), and 7 (starting at 26 and ending at 32). Also, note that if p8p9=0 and p25p26=0, no carry chain can be longer than 16.
So the carry chain cannot exceed 16 if X=$\overline{\text{p16p17}}$=1 or if Y=($\overline{\text{p8p9}}$)($\overline{\text{p24p25}}$)=1.
The carry chain length cannot exceed 8 if XY=1. The circuit below exploits these observations.
It does a little more than was asked for in this problem in that it shortens the delay to 16 for the case where X=0 and Y=1.



NT-3. Using as building blocks full adders and half adders, as well as a single block representing a fast 5-bit adder with carry out, draw a diagram of an array multiplier based on the carry save adder idea, that will multiply two 5-bit numbers. Assume as inputs, the 1-bit products, pij, where pij=AiBj.

```
                    b5   b4   b3   b2   b1
                     a5   a4   a3   a2   a1
                    ---------------------------
                     p15 p14 p13  p12  p11
                  p25 p24 p23 p22  p21
               p35 p34 p33 p32 p31
            p45 p44 p43 p42 p41
         p55 p54 p53 p52 p51
        ------------------------------------------------
S10   S9   S8   S7   S6  S5  S4   S3   S2    S1
```



NT-4. Construct a tree of CSA's to add 32 numbers. Don't worry about bit positions. Represent each adder (k-bits) by an oval. Terminate with a rectangle, representing a conventional adder that generates a single number for its output. There is no unique solution to this problem. The statement that this could be done with a tree of depth 8, including the conventional adder is wrong. The best that can be done for 32 inputs is a tree of depth 9. (Trees of this depth exist for up to 42 inputs.) I apologize for the error.

The tree below is one example of a depth-9 tree for n-32. It was built from the bottom up. Each oval represents a k-bit CSA, and the rectangle at the bottom represents a conventional adder--presumably a fast one--there is no point in using the array if we are going to use a slow adder at the end.