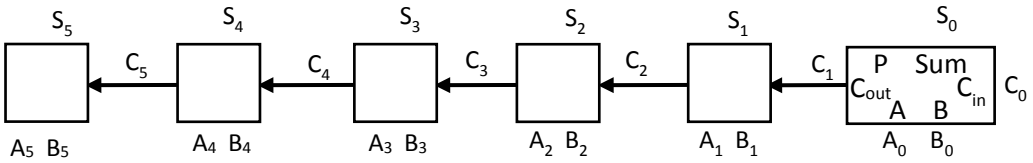
Homework #2 (static-timing analysis)

Turn in any four of these five problems.

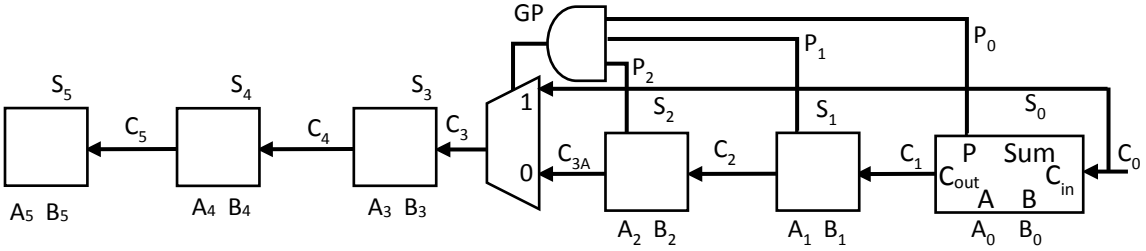
Problem #1. A *full adder* is a small circuit with the three inputs *A*, *B* and C_{in}. It has three outputs: *Sum* (which is *A xor B xor C*), C_{out} (which is *AB* | *BC* | *AC*) and *P* (which is *A xor B*). Essentially, it adds *A* and *B*, with a carry in, and produces a sum and the carry out. The "extra" output, *P*, stands for "propagate;" when it is true, then C_{out} will always equal C_{in}.

We could build a six-bit ripple-carry adder from six full adders as follows (not using the *P* output):



For simplicity, the inputs and outputs are labeled only in the rightmost full adder, but all six of them have their inputs and outputs in the same position on the full-adder box. Assume that within the full-adder block, the delay from any of *A*, *B* or C_{in} to *Sum* or C_{out} is one time unit (which is not quite realistic, but at least it's simple), and that the delay from *A* or *B* to *P* is also one time unit (and *P* does not depend on C_{in}). Assume that all of the *A* and *B* inputs are available at time 0, but that C_0 is not available until time=2. Label the latest arrival times on all of the nodes.

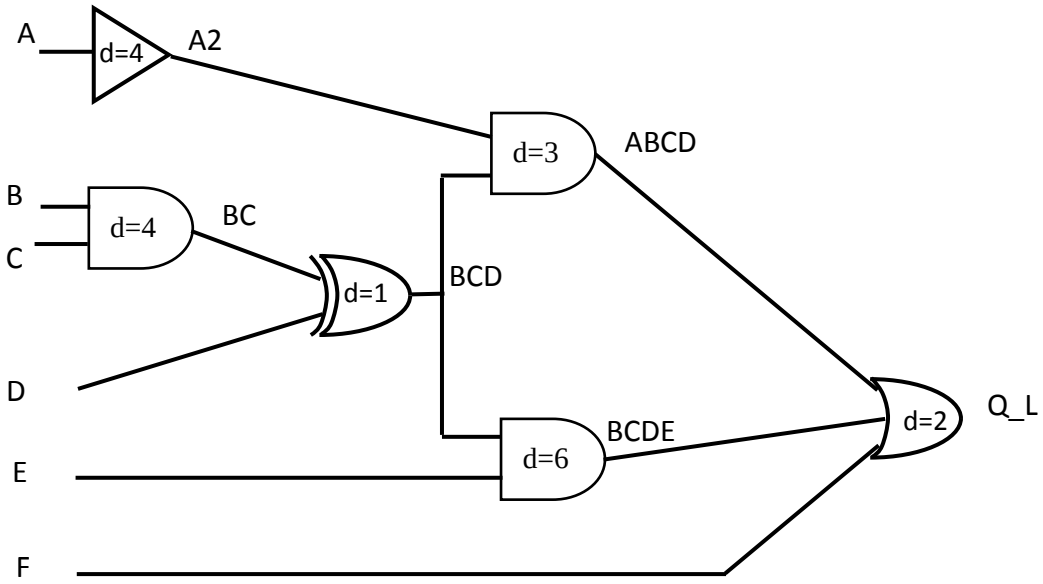
Ripple-carry adders are not a very fast way to do addition, since the chain of carry bits can get very slow. A somewhat faster method is a *carry-bypass* adder, as shown below.



In this adder, we have added circuitry to break the six-stage-long carry chain into two smaller pieces that run somewhat in parallel. The signal *GP* is a group propagate; it is one whenever all of stages 0, 1 and 2 are propagates (i.e., (P_0=P_1=P_2=1). When *GP* is high, that means that the carry out from the A_2,B_2 stage should just be C_0. We thus set up the mux so that when *GP*=1, C_0 gets copied directly to C_3. Otherwise, C_3 is driven from the long carry chain as usual.

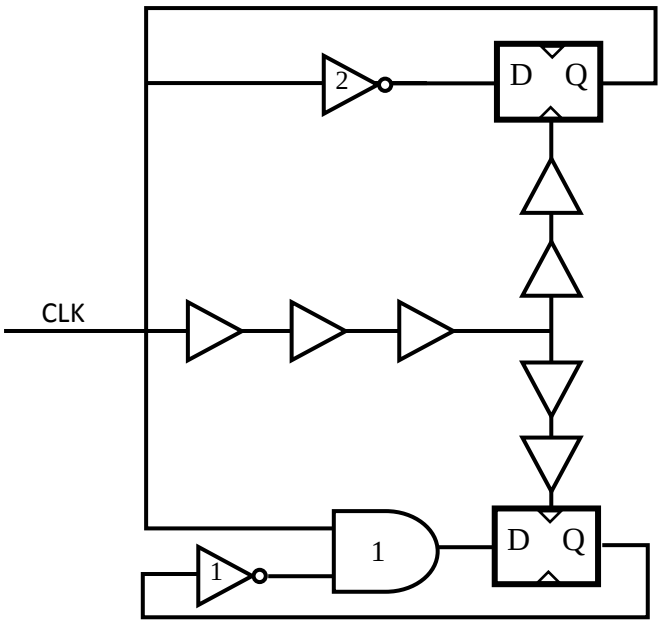
Just as in the ripple-carry adder, can you write down the latest arrival times on all of the signals? There is a false path in this circuit; what is it?

Problem #2. Consider the network below. Assuming that all inputs have an arrival time of t=0, show the latest arrival times at every node. Also, for each gate, draw an arrow to its output arrival time from whichever of its inputs caused that output arrival time.

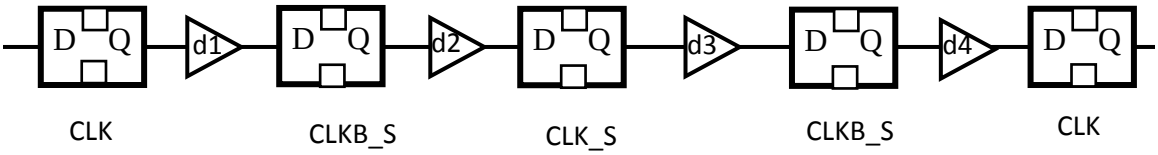


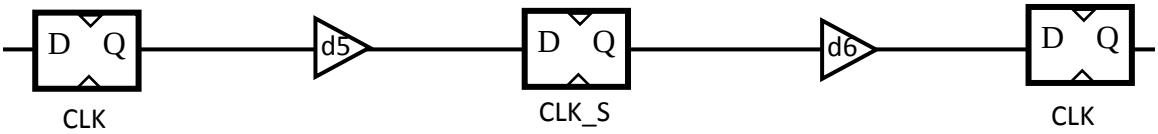
Problem #3. Consider the network below. For each of the three paths from clock to driver-flop through logic to receiver-flop, determine the minimum t_{cycle} for reliable operation. For the overall network, what is the minimum t_{cycle}?

Assume that the two flops have $t_{setup} = t_{clk \rightarrow Q} = 1$, that the seven clock buffers all have delay=1.5, jitter=±.1 and skew=±.2, and that the inverters and AND gate have the delays indicated inside of their picture.



Problem #4. Consider the following two networks. The clock period is 10 time units. Assume that CLK and CLKB are perfect true and complement clocks respectively (i.e., in each 10 time-unit clock period CLK rises at t=0 and falls at t=5 and that CLKB falls at t=0 and rises at t=5). Assume that CLK_S and CLKB_S are similar, but have *skew* – any given clock edge may rise or fall anywhere in the range 1 earlier





or later than the nominal time. The top network uses level-sensitive latches and the bottom network uses edge-triggered flops. In both networks, the first and last clocks are perfect (real-life clocks are never perfect, but this assumption makes our analysis easier) and the intermediate clocks have skew.

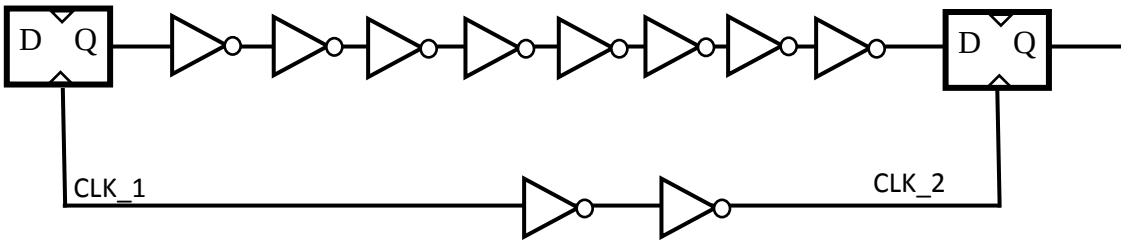
Your goal is to build circuits that do as much useful work as possible, but still meeting all timing constraints. The clock skew makes your life harder: you never know if the skew amount will be positive, negative, or 0 at any time, but you still must meet timing constraints.

Not knowing exactly when a clock edge will occur means that it's harder to meet timing constraints. Doing work takes time; when clock skew restricts the amount of time that your logic gates have to execute, it means that the logic can do less work.

For the network of latches, choose delays d1 to d4 such that you maximize work (i.e., maximize d1+d2+d3+d4) while meeting all timing constraints. For the network of flops, similarly pick d5 and d6. In which case (the flops or the latches), did you get more delay?

In which case (the flops or the latches) would you need to expend more power routing clocks?

***Problem #5*:** consider the following circuit.



There are eight inverters between the two flops. At the SS process corner, each inverter has a delay of 25ps, and the flops have t_{setup}=12.5ps and $t_{clk\text{-}to\text{-}Q}$=37.5ps. At the TT corner, each inverter has a delay of 20ps, and the flops have t_{setup}=10ps and $t_{clk\text{-}to\text{-}Q}$=30ps. What is the maximum operating frequency of this circuit at TT and SS? Which is worse?

We noted in class that the SS process corner is usually worst for the logic part of a critical path, while the TT or FF corners may make the clock delay worse. Which was the bigger effect for this circuit? Do you think that would be true for most circuits, and why?

Another effect that we did not discuss very much in class is *on-chip variation* (OCV). Many chips are so big that different parts of the chip can essentially be at different process corners! Luckily, the amount of variation on a single chip cannot be nearly as large as across chips; i.e., if a chip is nominally at TT, then all areas of the chip will be "nearly" at TT even with OCV, rather than being at SS or FF.

Assume that, for a chip that nominally came of the manufacturing line at TT conditions, OCV may cause delay values to vary by ±10% at 3 (i.e., roughly 99% of chips will fall into this range). Using these 3σ OCV conditions, what cycle time would you have to set to ensure correct operation?

Please turn in your assignment via the Provide cgi interface. You can use any reasonable format (including writing your answers by hand and taking a picture of the page with your phone).