EE/CprE 465 -- Fall 2015

Lab 9: System Power Optimization

In the last lab, we learned how to trade timing for area. In this lab, we will optimize power for your design. Please recall how CMOS circuits consume power. There are two kinds of power in general: dynamic power (i.e., switching power and short-circuit power) and static power (i.e., leakage power and contention power of ratioed circuit). In this lab, we are going to minimize switching power, which is the dominating component of dynamic power. (We have already learned in classes that short-circuit power is not a significant component of dynamic power. To save leakage power, a rule of thumb is to minimize area.)

First, let us review how switching power is calculated:

$$P_{switching} = \alpha C V_{DD}^2 f$$

where α is the switching activity factor, C is the total load capacitance, V_{DD} is the supply voltage, and f is the system clock frequency. Based on the equation above, to save switching power, we may reduce capacitive load, Vdd, clock frequency, or switching activity. Many techniques have been proposed to minimize switching power consumption at system level, logic level, circuit level, layout level and manufacturing process. In this lab, we will focus on reducing switching activity of a design. At each clock edge, the values stored inside the flip-flops may change and it will cause some of the signals in the combinatorial logic driven by the flip-flops to switch as well. A lot of dynamic power will be consumed as a result. To save power, it is a good idea to turn off the clock when it is not needed.

Please recall the example for area saving given in Lab 8. The example is shown again in Fig. 1 below. We have many registers inserted. We continue to use it as an example.

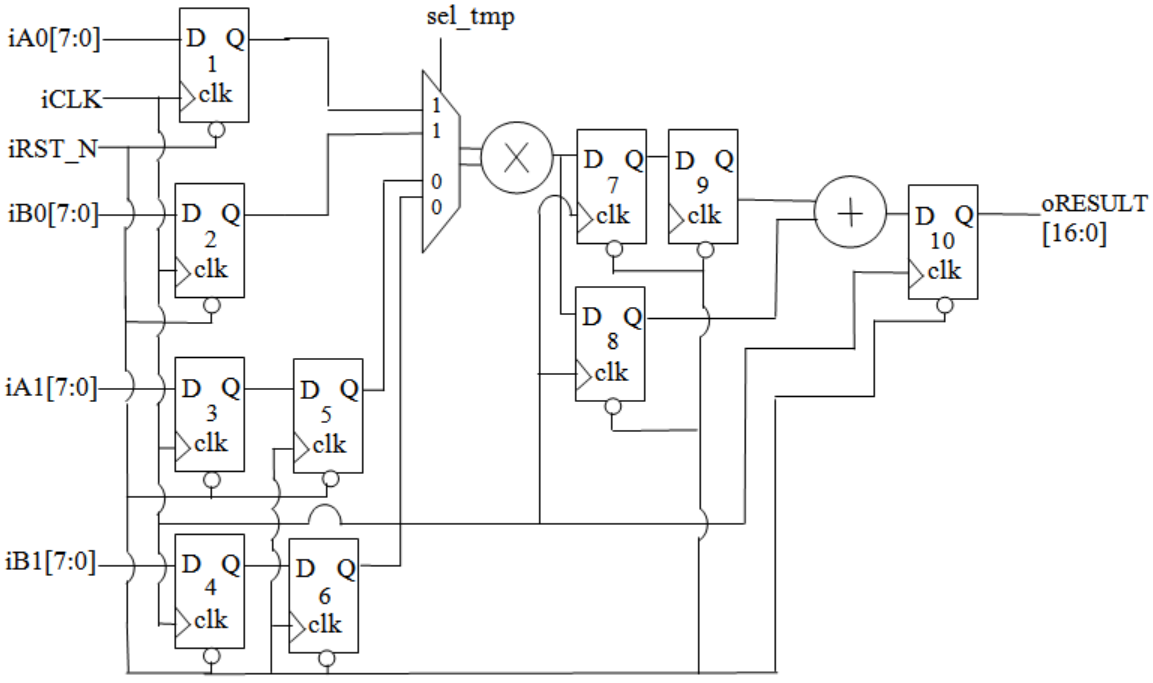


Fig. 1: Example design in Lab 8.

Do you notice that with so many registers, the circuit consumes a lot of switching power at every rising clock edge? Is all the switching necessary in this circuit? In Fig. 1, we knew that we need to abandon every other result of the adder output to register #10 because only one data is valid in every two clock

cycles. So do we still need to capture the invalid data into the register, which wastes power? For this reason, we need to "shut down" its clock every other cycle and capture just the valid result into it. Now let us trace why we have invalid data at every other clock cycle. Please look at register #1 to register #6. Is it really necessary that we put so many registers here and let their clocks tick all the time? Please note that register #5 and register #6 are put here because iA0, iB0, iA1 and iB1 need to share the multiplier. Therefore, we just want to store the inputs iA1 and iB1 temporarily in them to wait for the multiplier to finish. Could we just "shut down" the clocks of these registers alternately so that we can eliminate register #5 and register #6? By eliminating register #5 and register #6, we save switching power because we decrease the switching activity of this part of the circuit. We also save some area as a byproduct.

Now we introduce a technique for "shutting down" the clocks for registers. The technique is called clock gating. Please refer to the circuit in Fig. 2 below. When the output of Control Logic, EN, is "0", it is not necessary for the CLK of Register Bank to keep ticking and wasting power. In other words, if EN=0, there is no new data coming in and it is not necessary for Register Bank to capture the old data again and again at every clock cycle.

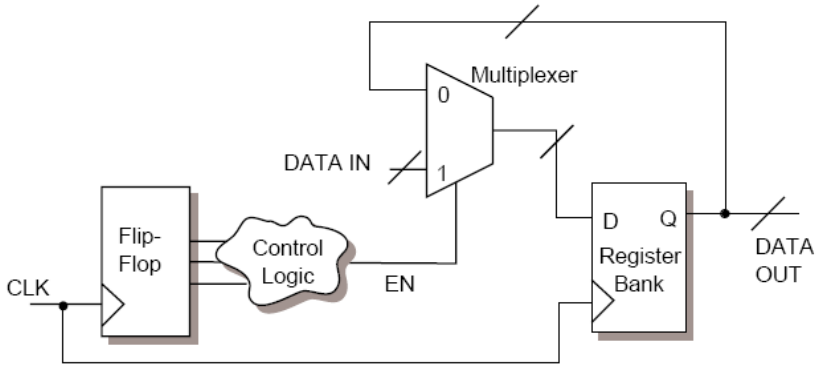


Fig. 2: A circuit without clock gating.

A little modification will solve this problem as shown in the circuit in Fig. 3 below. We "gate" (shut down) the CLK when EN=0. The gated clock is ENCLK. Using ENCLK as the clock signal for Register Bank, the old data is still stored in this Register Bank if EN=0. But there is a little problem, as the waveform in Fig. 3 shows. Because of the property of OR gate, a glitch will happen for ENCLK whenever EN is changing when CLK is "0". And this is a hazard for synchronize circuits.

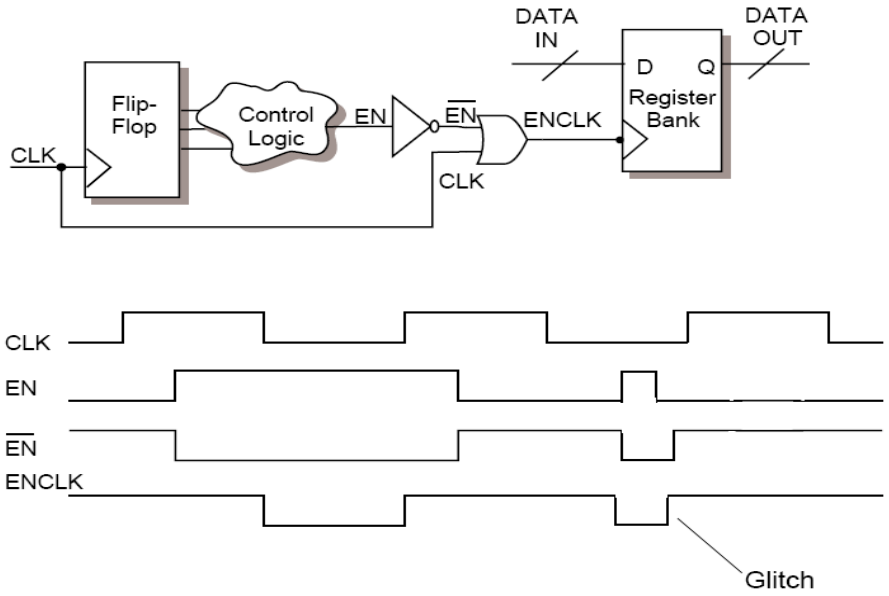


Fig. 3: The circuit in Fig. 2 with clock gating.

To solve this problem, we introduce the integrated clock gating cell, which is formed by one LATCH followed by a AND gate as shown in the circuit in Fig. 4 below:

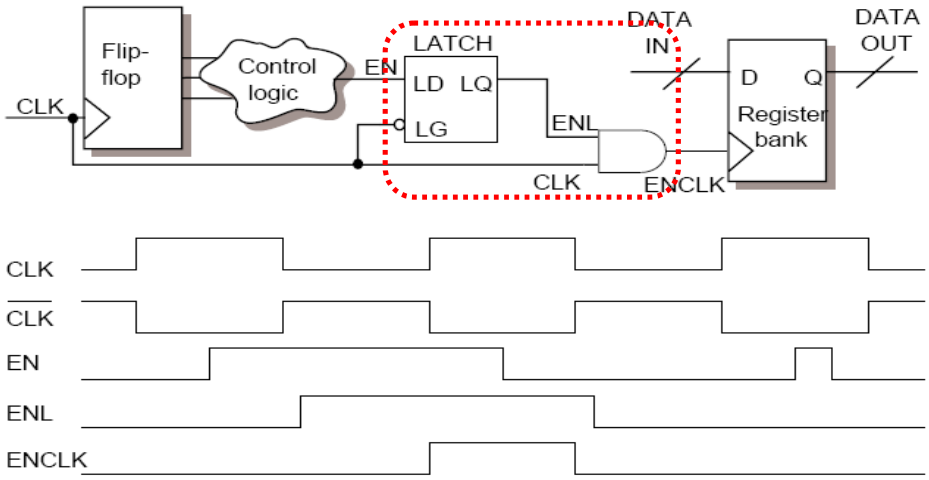


Fig. 4: The circuit in Fig. 2 with integrated clock gating cell.

But there is again a problem. Please notice that we use a latch here. Latch-based design is usually avoided in IC design except in processor design because latch-based design is level sensitive, entails time borrowing, complicates static timing analysis (STA), etc. The solution is to replace the integrated clock gating cell using a latch with one using a DFF and an OR gate as shown in Fig. 5:

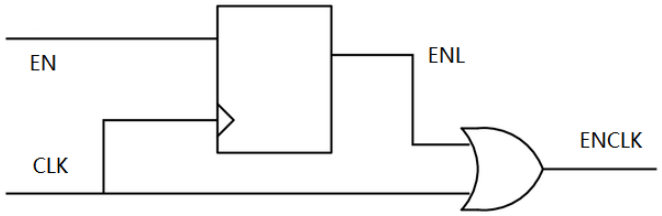


Fig. 5: Integrated clock gating cell using DFF.

Our synthesis software, Cadence RTL-Compiler, can recognize the circuit structure as shown in Fig. 2 and insert integrated clock gating cells automatically if we use the following command before "elaborate" in our script:

```
set_attribute lp_insert_clock_gating true
```

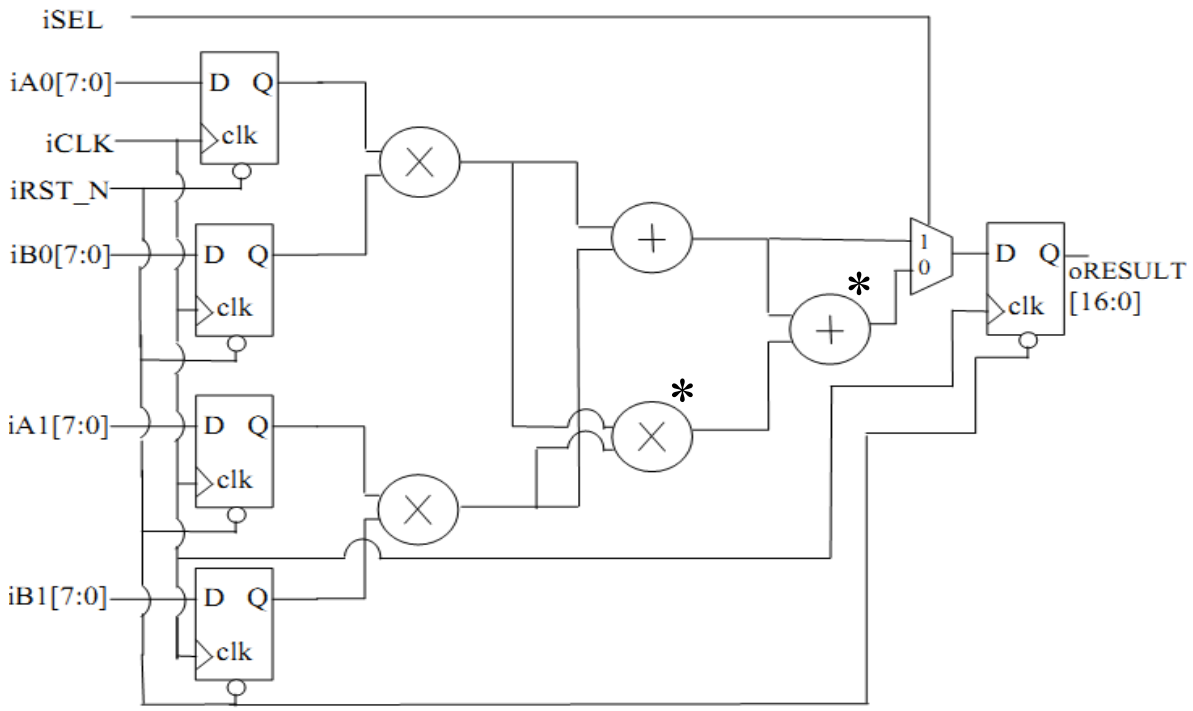
Unfortunately, the tool will not insert any integrated clock gating cell unless it recognizes there is such a structure as shown in Fig. 2 in your circuit. So sometimes, we have to manually insert a DFF followed by an OR gate to serve as an integrated clock gating cell if we want to gate some clocks.

At last, let us consider how our circuit will run in practice. If there is no active inputs (i.e., all inputs are unchanged) in two consecutive clock cycles, do we still want our circuit to keep calculating and wasting power? We want to save this part of switching power as well. At each clock cycle, we need to compare the new set of inputs with the previous set of inputs. If they are the same, we do not want our circuit to run. We just want to keep the current result at the output ports. So we need to gate the clocks of all DFFs except those in the comparison module. But please be careful. We cannot immediately gate them because our pipeline is still working on the current inputs. We need to figure out when to gate them. And if there are new active inputs coming in, we need to stop gating them.

Lab Tasks:

1. Please draw a circuit diagram like the one in Fig. 4 but replace the latch-based integrated clock gating cell with the DFF-based one as shown in Fig. 5. And please draw a timing diagram to show how DFF-based clock gating cell works and how it can also avoid glitches in the output gated clock.
2. Define the input clock signal, iCLK, as 5 MHz. Modify your design in Lab 8 and implement the whole idea discussed above to save switching power.
3. Please draw the modified circuit diagram of your design. You may just draw it by hand but draw it clearly.
4. Verify the function in ModelSim at least by 5 sets of input. Go through the synthesis flow and report the area, timing and power of your design.
5. Automatically insert integrated clock gating cell to your design in Lab 8 by using the command shown above in RTL Compiler and go through the flow of synthesis. Report how many clock gating cells are inserted automatically by RTL Compiler using the command: "report clock_gating –summary". Compare the area, timing and power based on the same constraints with your design in Lab 8. Please note that the power may not decrease so much because the power report in RTL Compiler is just an estimation. Without specific simulation input vector, it just uses some random input vectors to estimate power. Also, we introduced some logics such as comparison module. These parts of circuit are not so small comparing to the original circuit. So the extra power consumption of these parts of circuit may cancel out some of the power we saved. But for a larger and more complicated system, this kind of technique may save a significant amount of switching power.
6. Bonus (+25 points): Further improvement. Please notice that we have a MUX close to the output of the whole circuit and it is controlled by iSEL to determine which part of results needs to be outputted:

$$\begin{cases} \text{oRESULT= iA0*iB0+iA1*iB1+ iA0* iA1* iB0*iB1} & \text{when iSEL=0} \\ \text{oRESULT= iA0*iB0+iA1*iB1} & \text{when iSEL=1} \end{cases}$$



Now please consider that, if iSEL=1, do we still need the adder and the multiplier that are marked by "*" in the diagram above to work? Could we save this part of switching power? Please draw the circuit diagram and implement your idea.