**EE115C Winter 2016 Final Project:**

**A 1.25GHz, <insert energy #> pJ 6-bit Absolute-Value Detector**

Matthew Quach (UID: 304159669)

Jeffrey Jiang (UID: 904255069)

Team



**Layout: (Replace with a screenshot of your design)**

|  |  |
| --- | --- |
| Specifications [Replace everything in [] with your numbers] |  |
| Critical Path ([Signal\_In] to OUT) Delay, tp, post layout | [] ps |
| tp\_X0-OUT, post layout | [] ps |
| tp\_X5-OUT, post layout | [] ps |
| X (horizontal dimension) | [] µm |
| Y (vertical dimension) | [] µm |
| Area (X×Y) | [] µm2 |
| Aspect Ratio | [] |
| Supply Voltage | [] V |
| Schematic Energy | [] pJ |
| Layout Energy | [] pJ |
| Functionality | Y / N |
| DRC Clean | Y / N |
| LVS Clean | Y / N |

Design Summary:

-Circuit topology, circuit style (ex. ripple-carry adder + comparator, static CMOS)

-Why did you make those choices?

The circuit that we implemented essentially combined the two components of generating the absolute value and doing the comparison in the same step. We decided to change the topology because we felt that we could get rid of many unnecessary steps as a result of using somewhat different logic. We noticed that finding () is the same as finding (). Imagine that both and are positive 2’s complement numbers. Therefore, the sign bit of both and are 0. Then we consider , the bit-wise inverse of T. We know that will be a negative number with 1 as the sign bit. Therefore, if we add these two values, we get something equivalent to . If the value we get from this addition is positive or zero (sign bit is 0), then we know that . Otherwise, However, since we know the original most significant bits of both and , we realized that if the 5-bit addition has a carry-out of 1, then the sign bit of the 6-bit addition would be zero. Therefore, we reduced the problem down to computation of the 5th bit carry. We noticed that the block in Appendix C actually mirrors this idea.

Another trick that we used was to remove the “add-by-one” adder completely when finding the absolute value of by just taking the sign bit as the carry-in into the carry compute block. This adds a bit of complexity to our comparator block in order to remove the extra adder stage, allowing us to just use if the sign bit was originally 1. Using the general logic from carry-lookahead logic, we saw the general overview of our logic would be given by the equation , where and and . Of course, since we do not want to have a single gate to have 6 different inputs, we split the logic into two stages: , . Then, we also do some additional optimizations due to the inverting logic of CMOS, to change our propagate/generate signals into NOR/NAND gates and our two carry generating gates into an inverting gate, with a single inverter in between them.

Finally, the other logic we had to implement was simply choosing the implementation to derive the “absolute value” of , which was simply an inverted signal and a non-inverted signal going into a MUX, selected by the sign bit. The MUX was implemented with pass-transistor logic, for simplicity and a lower number of transistors. Any additional logic was simply input buffers to ensure proper input capacitance and logical effort.

Critical Path Analysis

-Block diagram of design / critical path delay equation (ex. )

The critical path that we were able to identify was the path that passed through the most stages. This was the path from to the output, which passed through a total of 7 stages. As a result, we have the critical path delay equation of .

Design Optimization

-Gate level critical path, logical effort analysis, gate sizing strategy

Functionality Check

-Relevant waveforms

Layout Considerations

-Critical path, X&Y dimensions, Area, Aspect Ratio, Density, etc.

Important Features

Since we have a relatively large gate of “C5\_gen,” we took advantage of a large stage effort as the last stage in order to minimize our energy consumption. Using the fact that our design has relatively few stages, we were able to lower the energy significantly by making our sizing somewhat suboptimal for speed. After we minimized our two complex gates as much as we could, we just wanted to optimize the remaining stage efforts by letting their stage efforts to be about the same, increasing slightly if possible. This allowed for a good trade-off of low energy consumption with a decent delay.

-ex. minimized delay through transistor sizing

-ex. highly regular/optimized layout design

-ex. logic optimization for reduced area

Given another chance, I would do *this* differently

-ex. Change topology, because…

-ex. Not optimize the whole thing so much to save time…

-ex. Nothing, I nailed it!