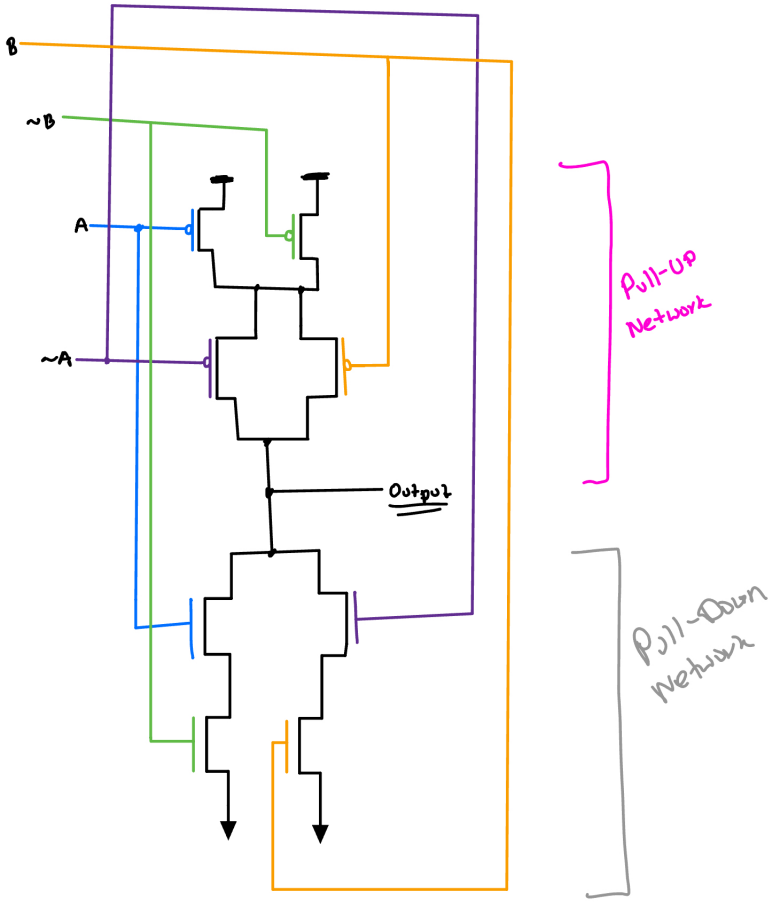
CIS 2620 Fall 2024

Pulkith Paruchuri

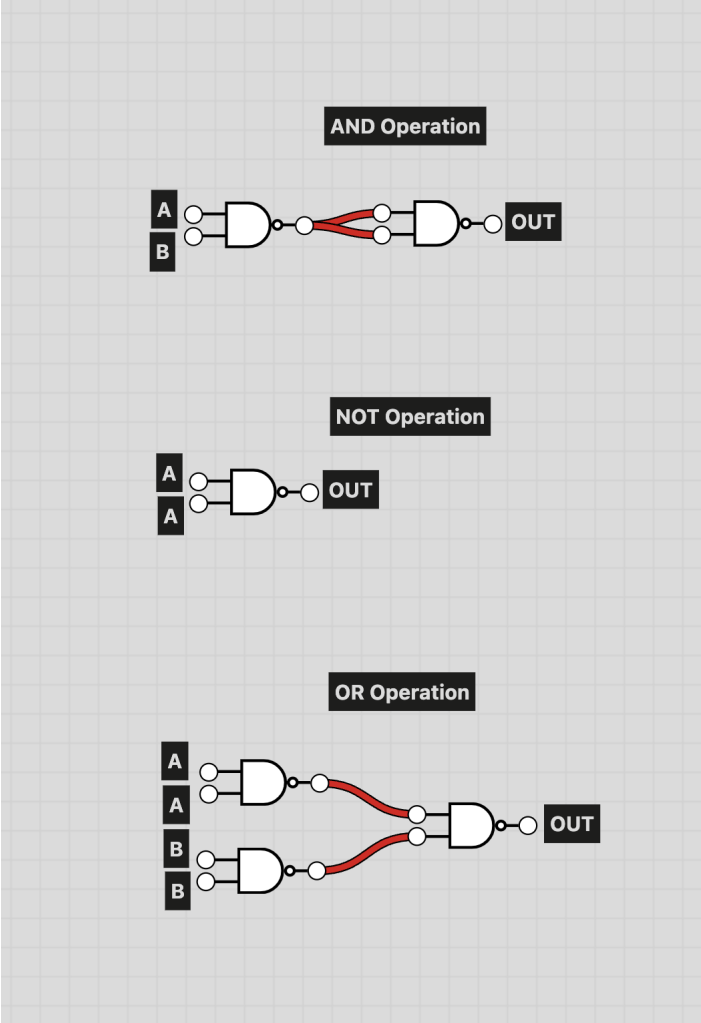
Homework 4

PennKey: 36386044

Q1. The circuit is below. The inputs and output are not part of either the Pull-Up or Pull-Down Networks.

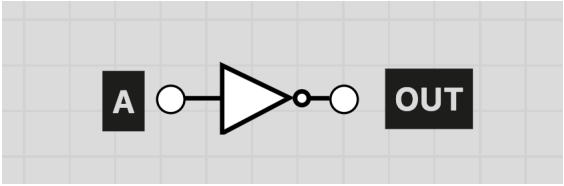


Q2. Pt 1 Logic Gates Below:



Pt 2 The "NOT" gate is not universal:

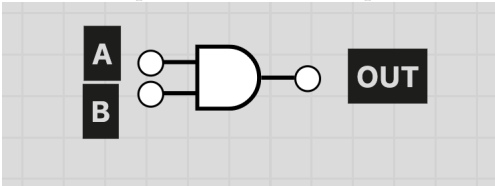
- You can implement the NOT operation as below



- You cannot implement the AND operation. This is because the NOT operation can only take one input, but the AND operation requires 2. To represent an AND operation, we need to combine the inputs in some way (like AND does), but we cannot since NOT only takes one input.
- We cannot implement the OR operation by the exact same rationale as the AND operation (we need to combine 2 inputs, but we cannot).

Pt 3 The "AND" gate is not universal:

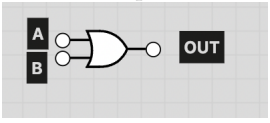
- The NOT operation needs only one input, so if we were to try to use AND operators to make a NOT operator, we would need both inputs to any AND operator to be the same (we can't have different inputs by inverting one an input, since we are trying to define the NOT operator to flip an input here). We can label any such input as A. Obviously, by definition of the AND operator, any use of the AND operator would just return A, so we will always have A, and can never flip it. Thus, you cannot implement the NOT operation.
- You can implement the AND operation as below



- Take the example $A = 1$, $B = 0$ as inputs. We need some combination of AND gates to output 1. We cannot invert A or B, as we cannot define a NOT operator. Any AND operator that takes both A and B will always output 0, and we cannot use this 0 output anywhere, since any subsequent AND operator will of course also output 0. We cannot rely on an AND operator that just accepts A because in the inverse case when $A = 0, B = 1$, this fails. We cannot have 2 separate sequences of AND gates where one accepts only A and one accepts only B, because this is equivalent to the original problem and leads us nowhere. Thus, we cannot create an OR gate.

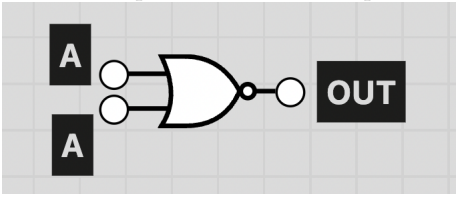
Pt 4 The "OR" gate is not universal:

- Since the NOT operator only takes one input, any sequence of OR gates to represent a NOT gate can only have 1 input, which we can denote as A. If $A = 1$, we need to invert this input for the output. However, any OR gate that takes A will of course still lead to an output of 1. Thus, since every OR gate will have an output of 1, we cannot have a NOT gate from only OR gates.
- To represent an AND gate, we need 2 inputs, A and B. Take $A = 1$, $B = 0$. We need some combination of OR gates to lead to an output of 0 to represent an AND gate. Any OR gate that takes A and B will always lead to an output of 1. Thus, we need to either (1) invert the output, which we have shown is not possible above, or (2) use OR gates where each gate only takes either A or B as input. However, in (2), each OR gate will obviously just output the same as the input (by logic of the OR gate), so we end up in the same situation as the start. Thus, we cannot represent an AND gate with OR gates.
- You can implement the OR operation as below

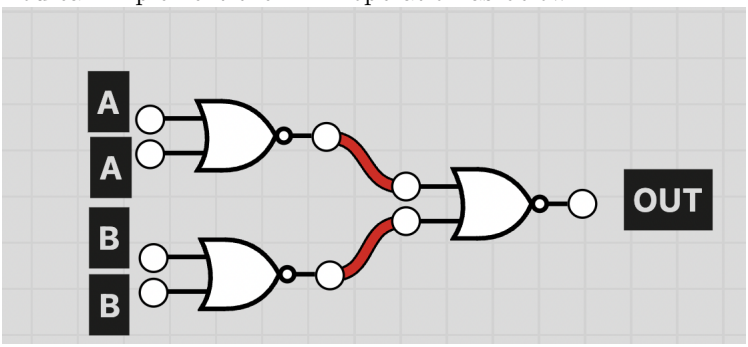


Pt 5 The "NOR" gate is universal:

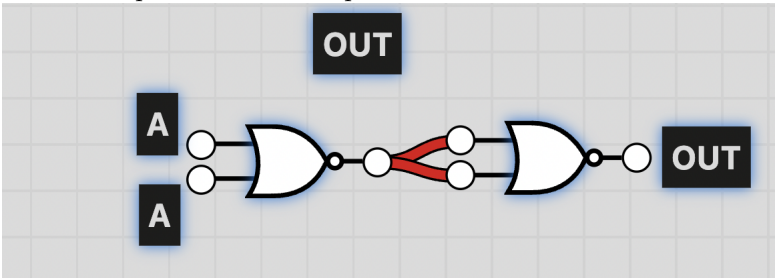
- You can implement the NOT operation as below



- You can implement the AND operation as below



- You can implement the OR operation as below



Pt 6 The "XOR" gate is not universal:

- Take a singular input $A = 0$. To create a NOT operation, we need to invert A. Because the XOR gate takes 2 inputs, any initial XOR gate must take A as both inputs, which results in 0. Then this 0 with any other 0 with any other 0 from another combination of XOR gates stays as 0, and this 0 with any other A as input also stays as 0. Thus, we can see, there is no way to get a 1, so a NOT operation is not possible.

- Let's look at two cases for the initial gate:

  (1) The initial gate XORs A with itself or B with itself. This leads to a guaranteed 0 for all values of A, B. If $A = 1, B = 1$, the only way to obtain a 1 towards the final solution is if we XOR this guaranteed 1 with A or B. However we need to reconcile these separate paths (0 XOR A, 0 XOR B), in the case only $A = 1$ or only $B = 1$, since we need a single output. However any XOR between one of the paths, and another duplicate of the path, A or B leads to 0. We can not just depend on one path, in the case either only A or B is 1. Thus, in this case, we cannot create AND from just XOR.

  (2) The initial gate XORS A with B. If $A = B = 1$, this gates creates a 0. The only way to create a 1 towards the output, is xoring this initial 1 with either A or B.

However, this again leads to two separate paths, that we need to reconcile, since we cannot just arbitrarily choose A or B (since if $A = 1$, $B = 0$, then the initial gate equals 1, and just choosing B leads to 1). Using $A = B = 1$, any XOR between the paths just remains as 0. Thus, we cannot create AND from XOR in this case.

Thus, its not possible to create AND from just XOR operators.

- It can be seen that the rationale for the OR operator is the same as the one for the AND operator, by replacing OR with AND, and replacing "since if $A = 1$, $B = 0$, then the initial gate equals 1, and just choosing B leads to 1", with "since if $A = 1$, $B = 0$, then the initial gate equals 1, and just choosing A leads to 0".

Q3. Because technology to dissipate heat from processors is lagging far behind the ability to thrust more transistors into a processor, oftentimes in mobile phones, only a fraction of the transistors are active at a time, running at a sustainable rate. Computational Sprinting is the idea that, because often computational demand occurs in short bursts, we can activate many more (or all) of the transistors at once (far above sustainable limits) for a very short period of time to address a burst in computational demand. Then, while waiting for user input, or any other low computational task, the processor can cool down, perhaps using a material like candle wax to dissipate heat, before the cycle repeats from another burst. This idea could also be transitioned to other technology like servers for improved performance.