Quantum Weirdness in Your Browser

Originalartikel

Backup

<html> I’ll be brutally honest. When I set out to write this post, I was going to talk about IBM’s Q Experience — the website where you can run real code on some older IBM quantum computing hardware. I am going to get to that — I promise — but that’s going to have to wait for another time. It turns out that quantum computing is mindbending and — to make matters worse — there are a lot of oversimplifications floating around that make it even harder to understand than it ought to be. Because the IBM system matches up with real hardware, it is has a lot more limitations than a simulator — think of programming a microcontroller with on debugging versus using a software emulator. You can zoom into any level of detail with the emulator but with the bare micro you can toggle a line, use a scope, and hope things don’t go too far wrong. So before we get to the real quantum hardware, I am going to show you a simulator written by [Craig Gidney]. He wrote it and promptly got a job with Google, who took over the project. Sort of. Even if you don’t like working in a browser, [Craig’s] simulator is easy enough, you don’t need an account, and a bookmark will save your work. It isn’t the only available simulator, but as [Craig] immodestly (but correctly) points out, his simulator is much better than IBM’s. Starting with the simulator avoids tripping on the hardware limitations. For example, IBM’s devices are not fully connected, like a CPU where only some registers can get to other registers. In addition, real devices have to deal with noise and the quantum states not lasting very long. If your algorithm is too slow, your program will collapse and invalidate your results. These aren’t issues on a simulator. You can find a list of other simulators, but I’m focusing on Quirk. <h2>What Quantum Computing Is</h2> As I mentioned, there is a lot of misinformation about quantum computing (QC) floating around. I think part of it revolves around the word computing. If you are old enough to remember analog computers, QC is much more like that. You build “circuits” to create results. There’s also a lot of difficult math — mostly linear algebra — that I’m going to try to avoid as much as possible. However, if you can dig into the math, it is worth your time to do so. However, just like you can design a resonant circuit without solving differential equations about inductors, I think you can do QC without some of the bigger math by just using results. We’ll see how well that holds up in practice. QC doesn’t use bits, but qubits. Unsurprisingly a qubit can take a state of 0 or 1. If that was all, then there wouldn’t be much reason to continue the conversation. However, qubits are quantum, so they have two unique properties that QC can exploit. First, a qubit can be some combination of 0 and 1 at the same time. There are famously multiple interpretations of uncertainty in quantum mechanics, but you can think of it as a probability of the qubit being a 0 or a 1. Those probabilities will always add up to 1 and the idea extends to groups of qubits, too. So four qubits have some probability that their state is 0010. If you add up all the probabilities for all the states, you get one. When a qubit is in a combination of states, each with a given probability, that’s called superposition. You probably know that in quantum mechanics, the state appears to change based on observation. Qubits are the same. While they can be in a superposed state, the instant you measure that state, it becomes either 0 or 1. If you know about Schrödinger’s cat, it is the same principle. In the thought experiment, until you peek, the cat is half dead and half alive just as our qubits can be both 0 and 1. The other

key concept is entanglement. The idea is you can take one qubit and make its state dependent on another qubit’s state even though we don’t know the state of either of them. As a simple example, we can create a qubit that we don’t know the state of, but we know it is opposite of some other qubit that we might also not know about. Confused? Good. If this makes sense to you, you are probably ahead of where this post is going anyway. Even [Niels Bohr] once said (paraphrasing from Danish), “Anyone who is not shocked by quantum theory has not understood it.” <h2>What Quantum Computing Isn’t</h2> Quantum computing isn’t going to run wordprocessing applications any time soon, if ever. Like I said, it is more like analog computing where you set up a problem and get an answer. Part of that may be the early stage we are in. Part of it is just how it works. Speaking of how it works, QC is usually probabilistic. So when you search for something, for example, you get the answer to a certain probability. Again, this is like an analog computer. Maybe that probability is 0.99999, but it isn’t going to be 1.0000. There aren’t many practical quantum algorithms known, and the ones that are known are sometimes difficult to put into practice. For example, if you read hand-waving explanations of QC, you’ll hear that Grover’s algorithm lets you search an unstructured database using fewer queries. A little thinking would probably convince you that this is unlikely. If you are searching a database of animals for “dog” and there’s no way to access it except by index, how could you mysteriously know which index items to skip? You can’t, and neither does Grover’s algorithm. We’ll talk more about Grover’s in another post, but what the algorithm actually does is match its state to another state that you don’t know in advance. For example, suppose you had a C program that makes a library call that reads like: <pre> int oracle(int x) { if (x==9) return 1; else return 0; } </pre> However, because it is in a library, we don’t know what x has to be to get the 1 return value. If you wrote a program to find out, you might have to call many times. Even if you restricted x to be between 1 and 16, you might have to call sixteen times. You might have to call one time. Grover’s algorithm would call it four times and have a pretty good guess at the answer. This is still not a perfect analogy because the oracle function in QC provides information about which answer is right, so the practicality of it isn’t “unstructured database” as you often hear. A better analogy might be if the oracle looked like this: <pre> int oracle(int x) { if (x==9) return 0; if (x<9) return -1; else return 1; } </pre> Now, even a classic computer could make smarter guesses. <h2>Enough Theory… For Now</h2> We’ll have some more theory later, but your eyes should be good and glazed over by now. I had thought about doing a video about how to use Quirk, but [Craig] did a fine one, so watch that first. When [Craig] compared his simulator to IBM’s he talked about UI design principles he uses and it shows when you use Quirk. <iframe class=„youtube-player c2“ type=„text/html“ width=„800“ height=„480“ src=„https://www.youtube.com/embed/aloFwlBUwsQ?version=3&rel=1&fs=1&autohide=2&showsearch=0&showinfo=1&iv_load_policy=1&start=311&wmode=transparent“ allowfullscreen=„true“>[embedded content]</iframe> Each of the horizontal lines each represents a qubit and they start in state 0. Time goes from left to right along the qubit and doesn’t go backward, so you read the circuit that way. All of the boxes in the top and bottom gray zones are gates or components that affect the qubits. To the right of the qubits are several ways to visualize the final state of the system. The green boxes show if they are one or zero (or the chances of it, at least). The other two displays will take a little more theory that we’ll talk about next time. <h2>Straight Logic</h2> For now, I want to focus on just the 0 and 1 (technically, |0> and |1>) states. This is just like regular logic, but you can’t use most of the familiar gates you are used to. One that you can use is the NOT gate: the circle with a plus sign in it. Sometimes this also shows up as an X and is known as an X gate. Try

it: As you’d expect, inverting a 0 once creates a 1 and inverting it twice makes a zero. No surprise. There’s also a controlled not (CNOT) gate that is really an exclusive OR gate. That is, if one input is true, the other input inverts. To create one, you just put a dot on any qubit in the same column as an inverter. Try putting a solid dot (from the Probes section) over the second inverter. Note the output and then delete (middle click) the top left inverter. You’ll see the states change where the bottom right inverter only flips the bit if the top qubit is in the 1 state. Remember that I said there are a lot of

oversimplifications that make things hard to understand? This is one of them. The idea that this is a gate with two inputs and one output is only correct in the case that the inputs are 0 or 1. But once we start dealing with qubits in superposed states, this isn’t true anymore. The “input” dot is perfectly capable of modifying the first qubit and — under the right circumstances, you can flip the inverter and the dot and it wouldn’t matter. QC is strange. However, for “computational states” (|0> and |1>) the gate analogy is a fair one. Just don’t forget that it is wrong. One note about the simulator. If you see NaN on the outputs (short for Not a Number), try using a different browser. I’ve seen some — but not all — versions of some browsers have trouble with the Javascript. That’s not quantum weirdness, a NaN is a normal software bug. <h2>Toffoli Gate</h2> If you are accustomed to normal logic gates, you know that you don’t need all of them. For example, a NAND gate is sufficient to make any other kind of logic gate you want. The NOT and CNOT gates get us part of the way there. The other gate that is easy to implement in QC is a Toffoli gate, sometimes known as a CCNOT gate. You can think of it as a 3-input gate (say, A, B, and C) that has three outputs (X, Y, Z). Two of the outputs are the same as the inputs (that is, X=A, Y=B). The Z output, however, is A AND B XOR C. That’s a bit rough, but you can make whatever gates you want out of that and the NOT/CNOT gate. To create a Toffoli gate in Quirk, you simply put two dots over an inverter as seen to the right. The two inverters on the left are there to set the inputs to a 1 state. Try it and you’ll see the output is 1 unless you remove either of the inverters. Putting an inverter on the bottom qubit (under the other two inverters) will invert the answer (that’s the XOR C part). Obviously, configured like this, we have a two-input AND gate. Inverting the inputs gives you an OR gate. Setting the bottom qubit would give you NAND or NOR. Setting B to 1 would give A XOR C. Just like the CNOT, this gate can affect its inputs when dealing with quantum states, but we aren’t looking at those yet, so, for now, it is OK to think of the top two qubits along with the bottom left as inputs and the right side of the bottom inverter as an output. Except it isn’t in the general case. By the way, you’ll see when we use the IBM simulator (which is more like the hardware) there’s no way to build this directly. You have to use many CNOT gates to build one (see below). This is one reason you want to start with Quirk. <img data-attachment-id=„290245“ data-permalink=„https://hackaday.com/2018/01/24/quantum-weirdness-in-your-browser/ibmtof/“ data-orig-file=„https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=800&h=192“ data-orig-size=„1501,360“ data-comments-opened=„1“ data-image-

meta=„{"aperture":"0","credit":"","camera":"","caption":"","created_timestamp":"0","copyright":"","focal_length":"0","iso":"0","shutter_speed":"0","title":"","orientation":"0"}“ data-image-title=„ibmtof“ data-image-description=„“ data-medium-file=„https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=800&h=192?w=400“ data-large-file=„https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=800&h=192?w=800“ class=„alignright size-full wp-image-290245“ src=„https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=800&h=192“ alt=„“ width=„800“ height=„192“ srcset=„https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=800&h=192 800w, https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=250&h=60 250w, https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=400&h=96 400w, https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png?w=768&h=184 768w, https://hackadaycom.files.wordpress.com/2018/01/ibmtof.png 1501w“ sizes=„(max-width: 800px) 100vw, 800px“/> <h2>Instant Gratification</h2> I think that’s enough for this post, even though we haven’t got to the quantum weirdness yet. However, if you want to jump ahead, select the menu from Quirk and click the Quantum Fourier Transform link. That should keep you dizzy until next time when we talk more theory, get into superposition, and tangle with entanglement. If you are wondering what’s the point? Hang on to that thought. Right now, we only know about how QC can look like regular logic gates. But that’s just a small part of what you can do with a quantum computer. </html>