Informatique Quantique – Cours 2

Nicolas Boutry[1]

[1] Laboratoire de Recherche et Développement de l'EPITA (LRDE), France

Outline

Outline

Dans ce programme, comme le NOT est sur 0*x*4, on l'appliquera sur les couples ($|0\rangle, |4\rangle$), ($|1\rangle, |5\rangle$), ($|2\rangle, |6\rangle$), ou ($|3\rangle, |7\rangle$). Cela à condition que les bits de contrôle *a* et *b* soient à 1. Autrement dit, seuls les états $011 = |3\rangle$ et $111 = |7\rangle$ sont concernés. Ici, comme ces deux états sont à 0, on ne voit rien se produire.

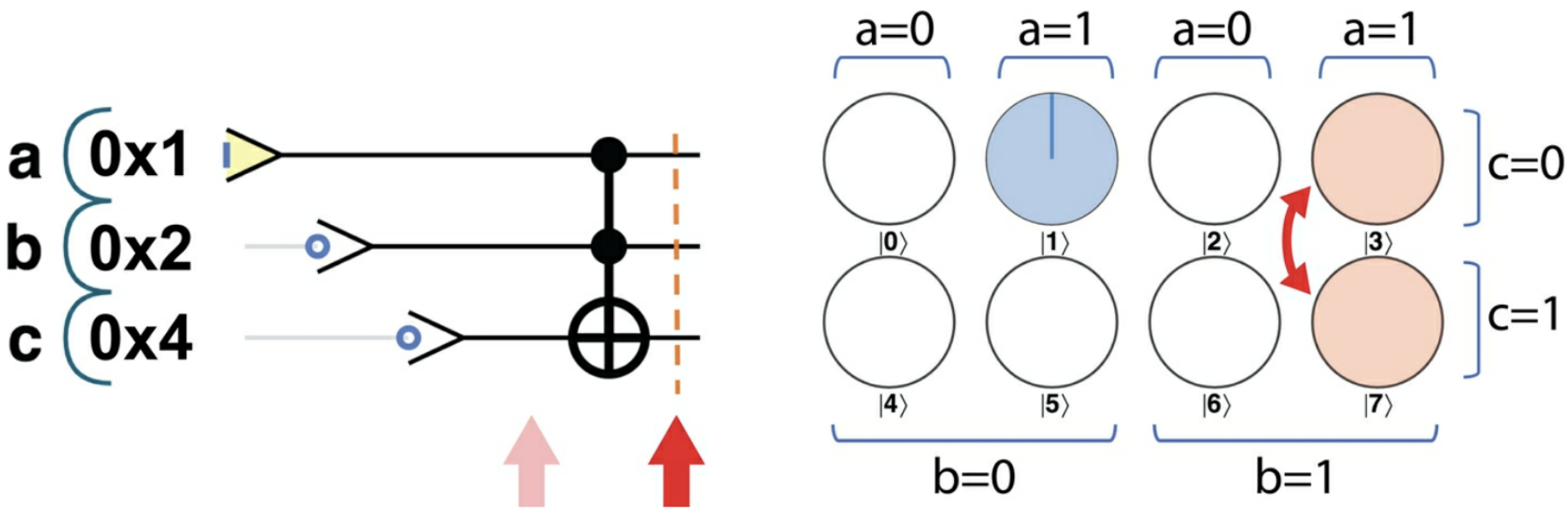


Figure 5-1. When b=0, this operation has no effect

Là, grâce a HAD, on est en même temps dans les états $001 = |1\rangle$ et $011 = |3\rangle$ avant l'application du NOT. Puis le NOT s'applique à $(|3\rangle, |7\rangle)$, on active donc l'état $|7\rangle$ et on désactive l'état $|3\rangle$.

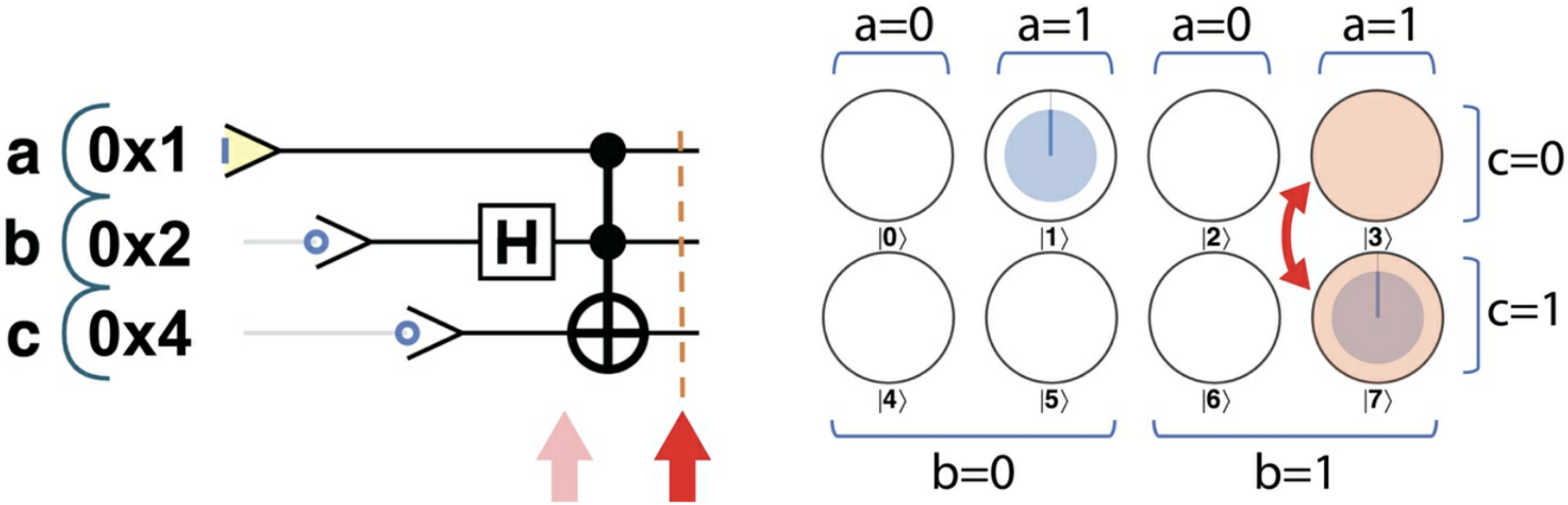


Figure 5-2. A single gate performs two operations at the same time

En combinant de telles opérations, on peut arriver à faire des opérations d'addition (incrémentation de 1, ou décrémentation de 1):

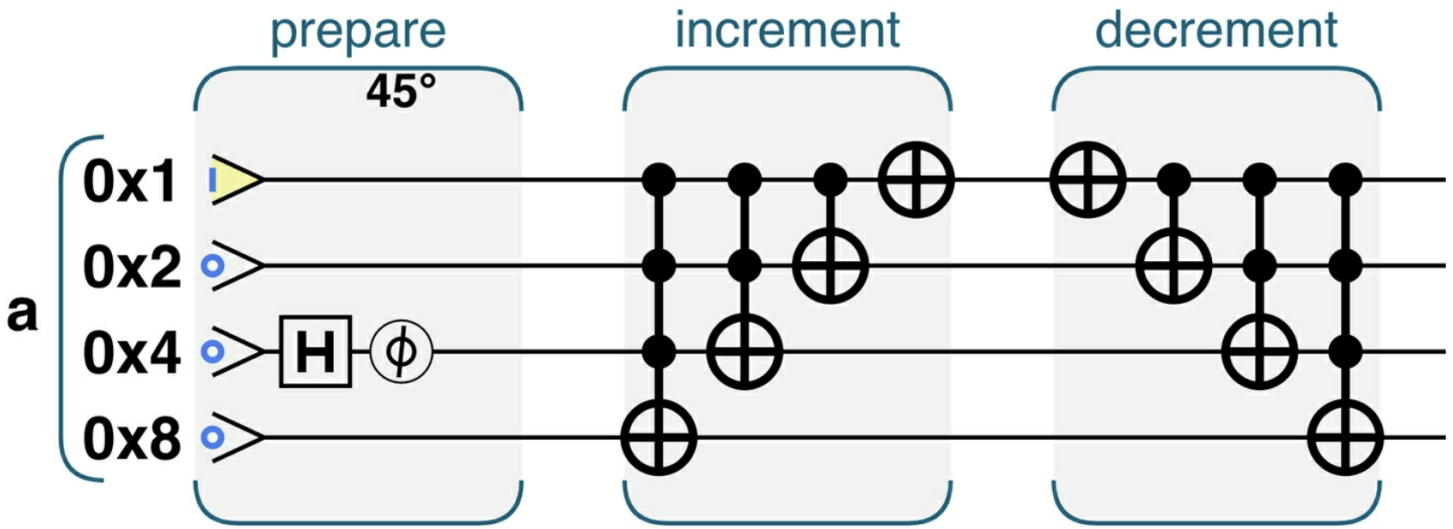


Figure 5-3. Operations performing increment-by-1 and decrement-by-1

Notez que l'on à décrémenté après avoir incrémenté, c'est la notion de réversibilité (vue au slide suivant). On aura eu besoin de 4 portes quantiques pour incrémenter de 1 !

Exercice : montrer que 0 devient bien 1, 1 devient bien 10, etc. par l'opérateur increment.

L'idée est qu'à chaque opération, on appliquera tôt ou tard son opération inverse, pour remettre le qubit dans son état initial. C'est la réversibilité.

Reversibility

> First, and most obviously, the decrement operation is simply the increment with its constituent operations reversed. This makes sense, but may not be obvious if you're used to conventional logic. In conventional logic devices gates tend to have dedicated inputs and outputs, and simply running a device in reverse is likely to damage it, or at least fail to provide a useful result. As we've noted, for quantum operations reversibility is a critical requirement.

Observons le déroulement sur la notation groupée :

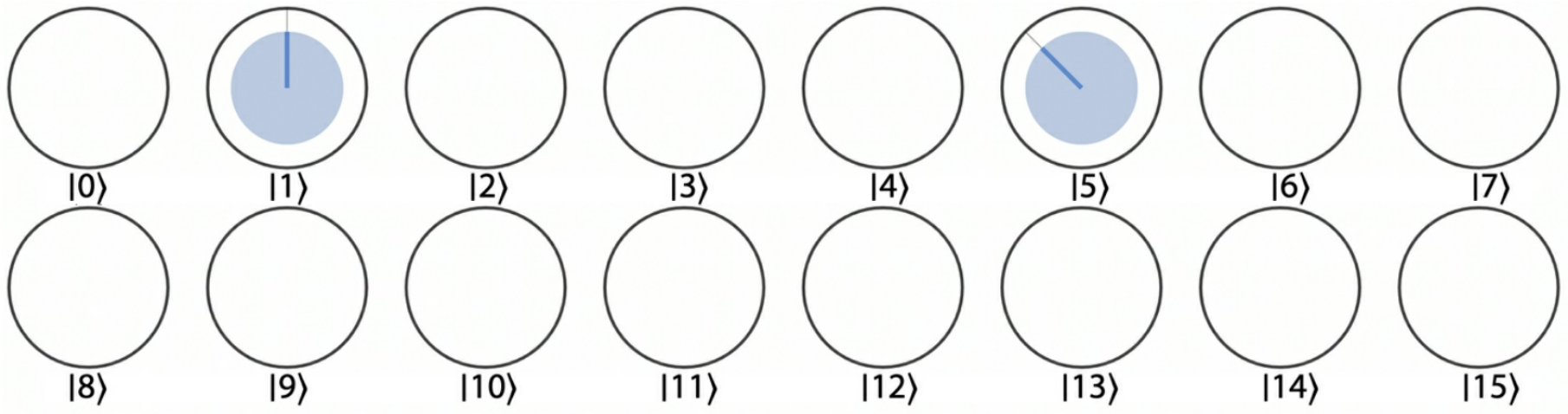


Figure 5-4. The prepared superposition, before incrementing

On incrémente de 1 tous les états en même temps : l'état $|1\rangle$ devient donc l'état $|2\rangle$ et l'état $|5\rangle$ devient l'état $|6\rangle$. Donc oui c'est beaucoup 4 portes pour une incrémentation, mais on a parallélisé l'opération !

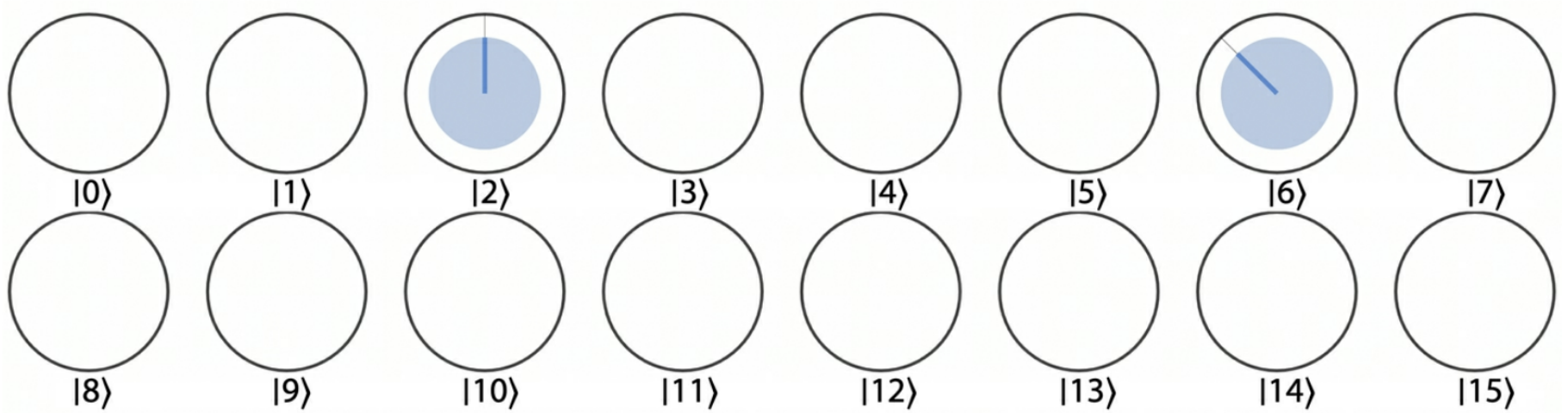


Figure 5-5. The resulting superposition, after incrementing

Attention, on va boucler lors des additions: lorsqu'on fait une addition de 12, $|1\rangle$ devient $|13\rangle$ et $|5\rangle$ deviendra $17 = 1$ donc $|1\rangle$ car 4 qubits. On parle d'overflow.

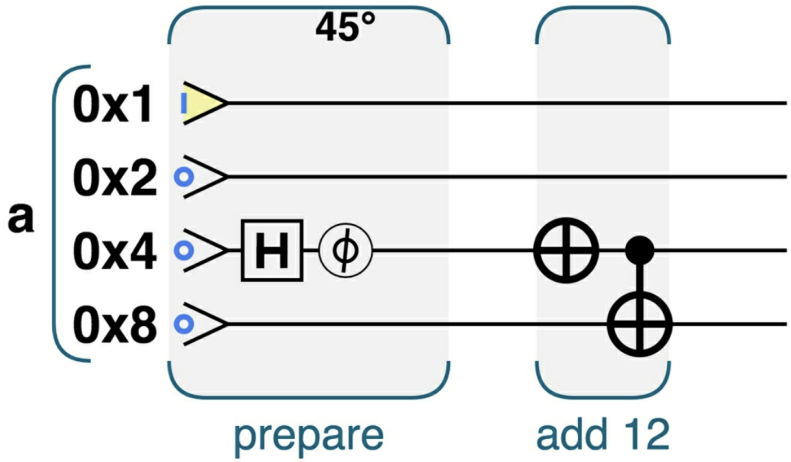
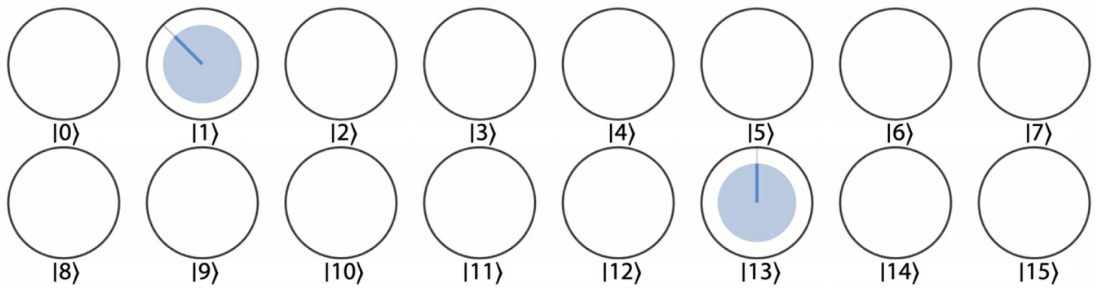


Figure 5-6. A program to add 12 to a quantum integer

In this case, Figure 5-7 shows that the input values $|1\rangle$ and $|5\rangle$ will become $|13\rangle$ and $|1\rangle$, since this program will wrap on overflow (just like conventional integer math).



On a vu l'ajout à une variable d'une constante, comment ajoute-t-on deux qubits ?
Rappel : on n'a pas le droit de lire ni *a* ni *b* !

Adding Two Quantum Integers

Suppose we have two QPU registers `a` and `b` (bearing in mind that each of these could potentially store a superposition of integer values), and we ask for a simple `+` operation that would store the result of their addition in a new register `c`. This is analogous to how a CPU performs addition with conventional digital registers. However, there's a problem — this approach violates *both* the reversibility and the no-copying restrictions on QPU logic:

- Reversibility is violated by `c = a + b` because the prior contents of `c` are lost.

Solution : on va utiliser des portes quantiques qui vont les faire interagir sans qu'on ait besoin de les lire:

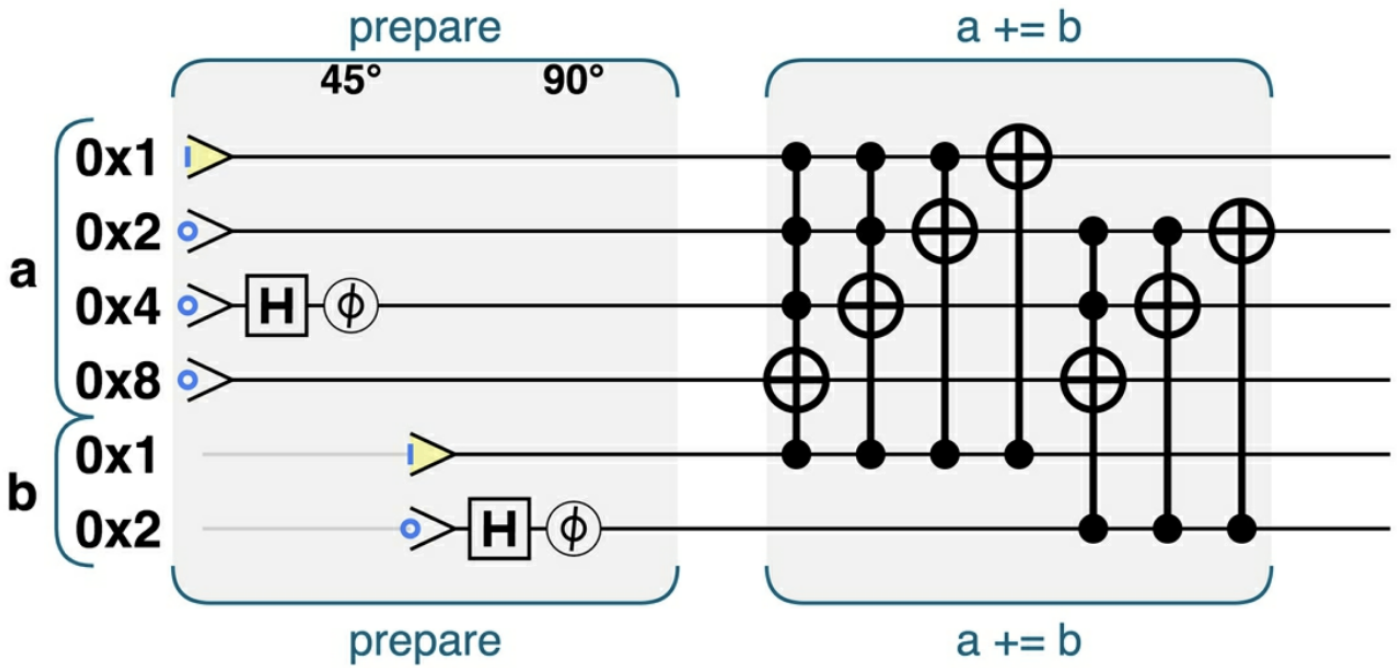


Figure 5-8. Operations assembled to perform a += operation

Exercice : comprendre comment $a = 1$ plus $b = 1$ donnerait 10 (attention, il s'agit d'une autre initialisation que celle décrite ci-dessus, regardez juste la zone $a+ = b$).

Rappel : il existe une notation appelée complément à 2, qui permet de représenter des valeurs négatives. Si le qubit de poids fort est à 0, c'est comme d'habitude, par contre quand il est à 1, on compte ainsi:

Table 5-1. Two's complement for binary representation of negative integers

0	1	2	3	–4	–3	–2	–1
000	001	010	011	100	101	110	111

La somme en binaire de $-a$ et de a fait 0, sauf pour la valeur négative minimale où l'on met le qubit de poids fort à 1 et le reste à 0. On obtient donc cette forme de symétrie dans le tableau.

Un façon équivalente de calculer l'opposé de *a* est d'inverser tous ses bits (penser aux notations circulaires monobits), et d'ajouter 1 (reconnaissez le pattern d'incrémentation ici):

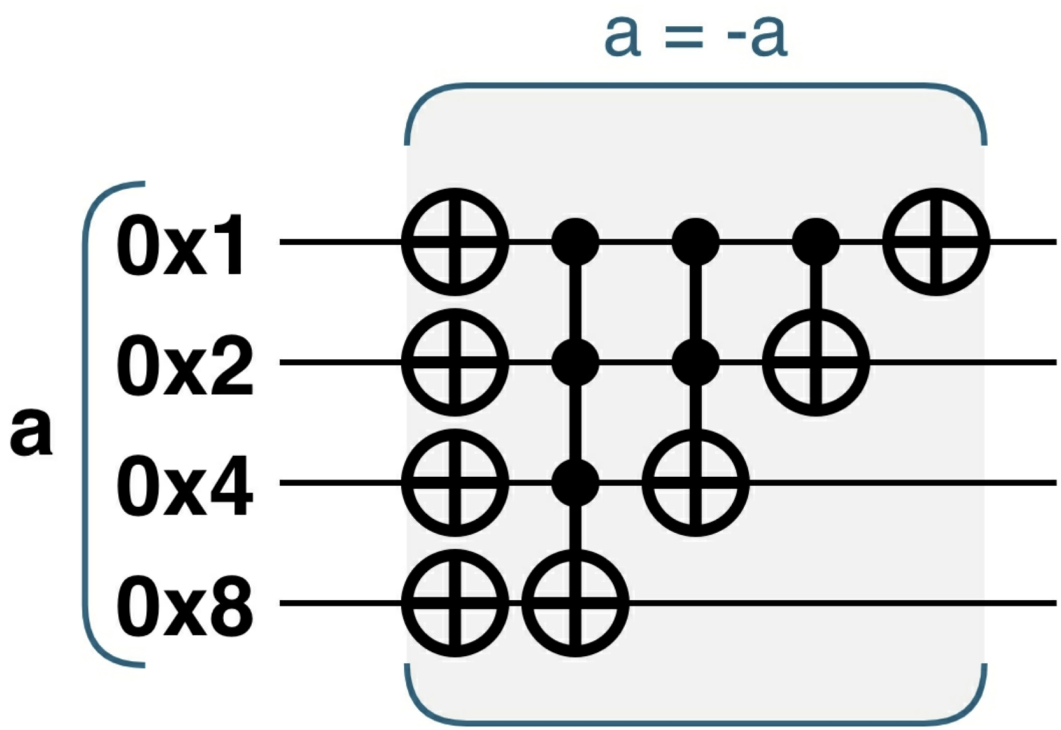


Figure 5-9. Two's complement negation: flip all bits and add 1

En effet, 000 inversé donne 111 et additionné à 1 on obtient à nouveau 000, etc.

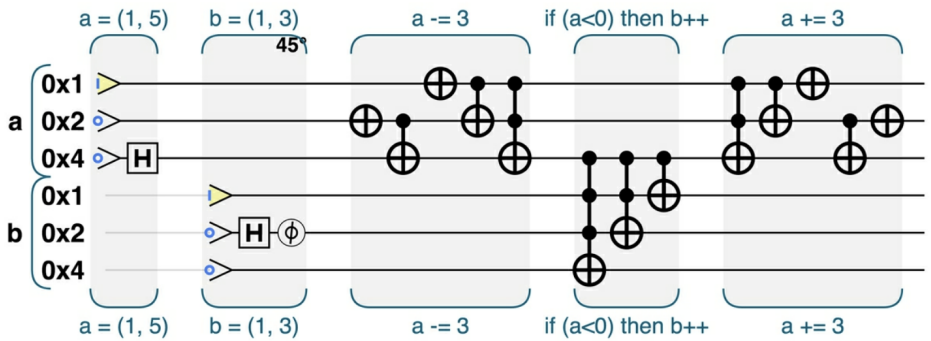


Figure 5-11. Conditional execution

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=5-4*.

Example 5-4. Conditional execution

```
a.subtract(3);
// if high bit of a is set then b += 1
b.add(1, a.bits(0x4));
a.add(3);
```

Voici un exemple de programme avec réversibilité : on prépare les données avec certaines initialisations plus des HAD, on retire 3, on fait notre petit traitement (ici on incrémente *b* si *a* est inférieur à 0, par interactions grâce à des portes quantiques, sans lecture), puis on remet *a* dans son état initial pour l'aspect réversibilité.

Running Example 5-4, circle notation reveals that only parts of the quantum state where a is less than 3 or greater than 6 are incremented.

Only circles in columns 0, 1, 2, *and* 7 in Figure 5-12 are affected by the incrementing logic.

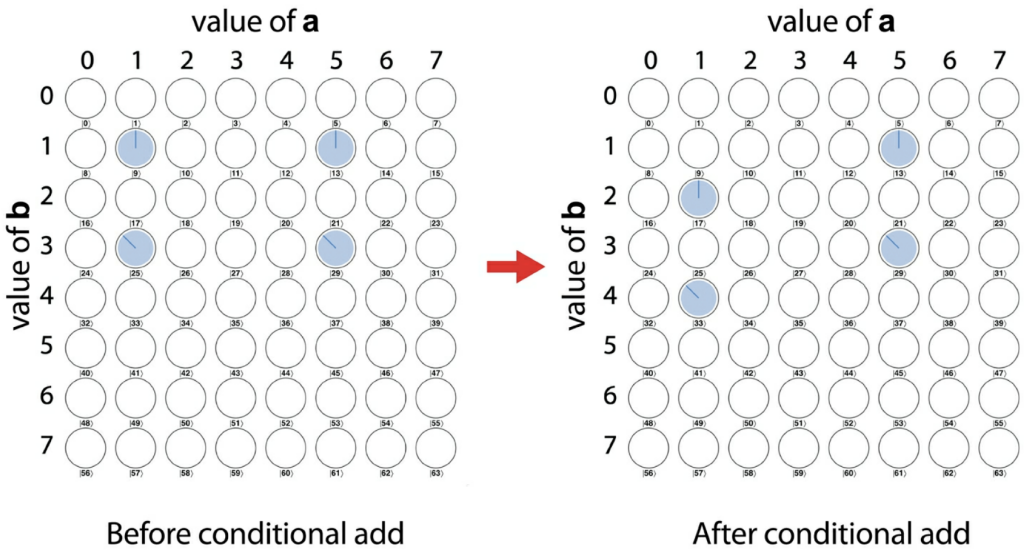


Figure 5-12. Conditional addition

Exercice: faites les étapes intermédiaires par écrit (la solution est au slide suivant).

Running Example 5-4, circle notation reveals that only parts of the quantum state where a is less than 3 or greater than 6 are incremented.

Only circles in columns 0, 1, 2, *and* 7 in Figure 5-12 are affected by the incrementing logic.

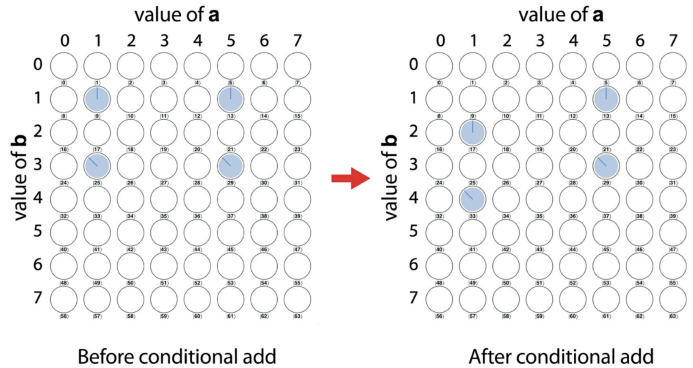


Figure 5-12. Conditional addition

Solution : prenons le cas $a = |1\rangle = 001$ et $b = |1\rangle = 001$. Alors on décrémente a de 3 pour obtenir $a = |6\rangle = 110$, le bit de poids fort (donc de signe ici) est à 1, car en réalité $6 = -2$, on ajoute donc 1 à b qui devient $|2\rangle = 010$, puis on revient à l'état d'avant pour a donc à $|1\rangle = 1$ (vous pouvez vous imaginer vous déplacer sur la grille au fur et à mesure des opérations).

Voici un nouvel exemple, avec un `if then` mais sur les phases. On garde toujours autant que possible l'aspect réversible.

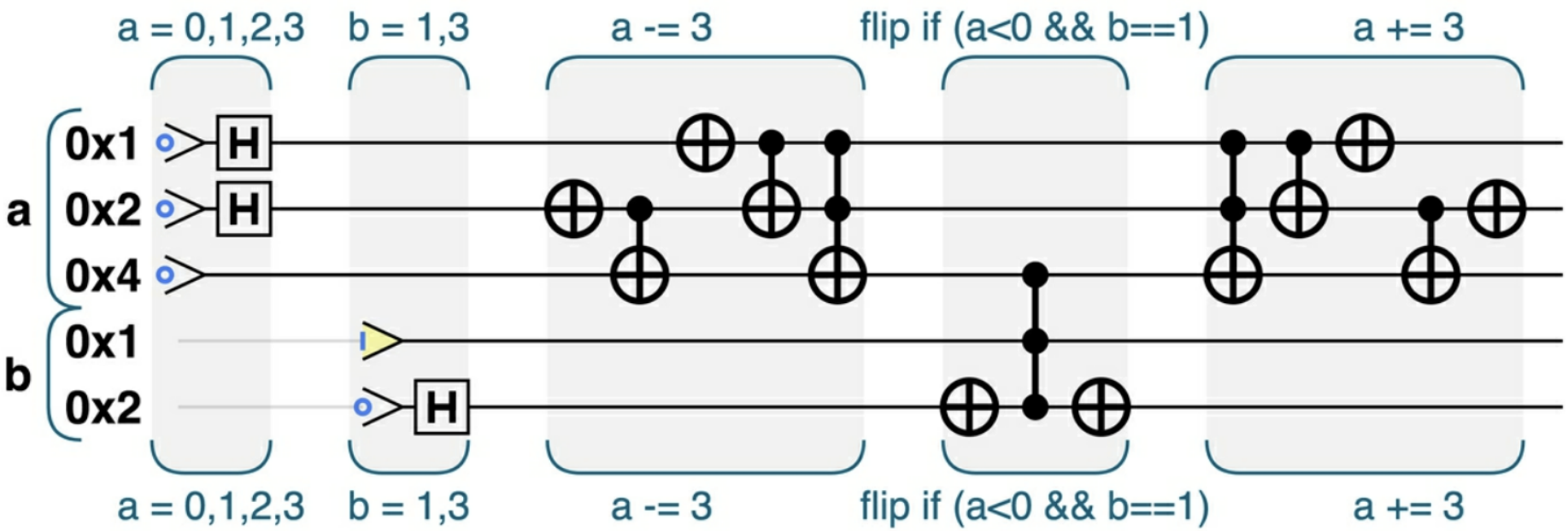


Figure 5-13. Phase-encoding the result

Les états initial et final seront donc :

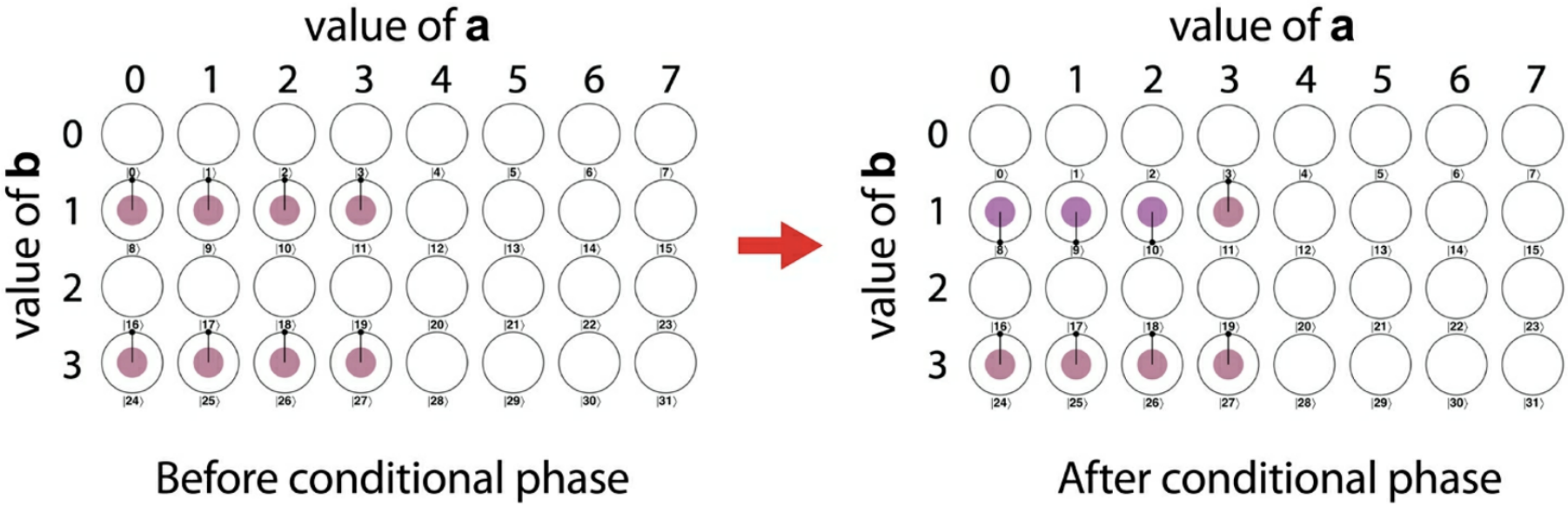


Figure 5-14. The effect of phase encoding

Repensez au décalage sur la gauche puis sur la droite comme précédemment.

Question : comment calculer une valeur absolue, sachant que l'on a pas le droit de lire le signe ?

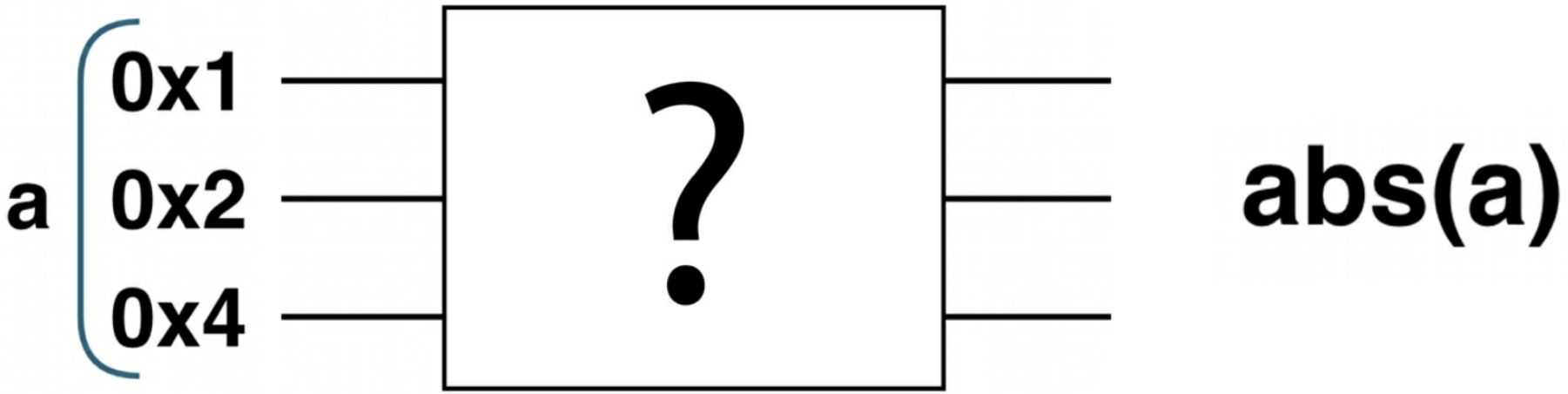


Figure 5-15. What QPU operations can compute the absolute value?

On va rajouter des qubits temporaires qu'on appelle les scratch qubits:

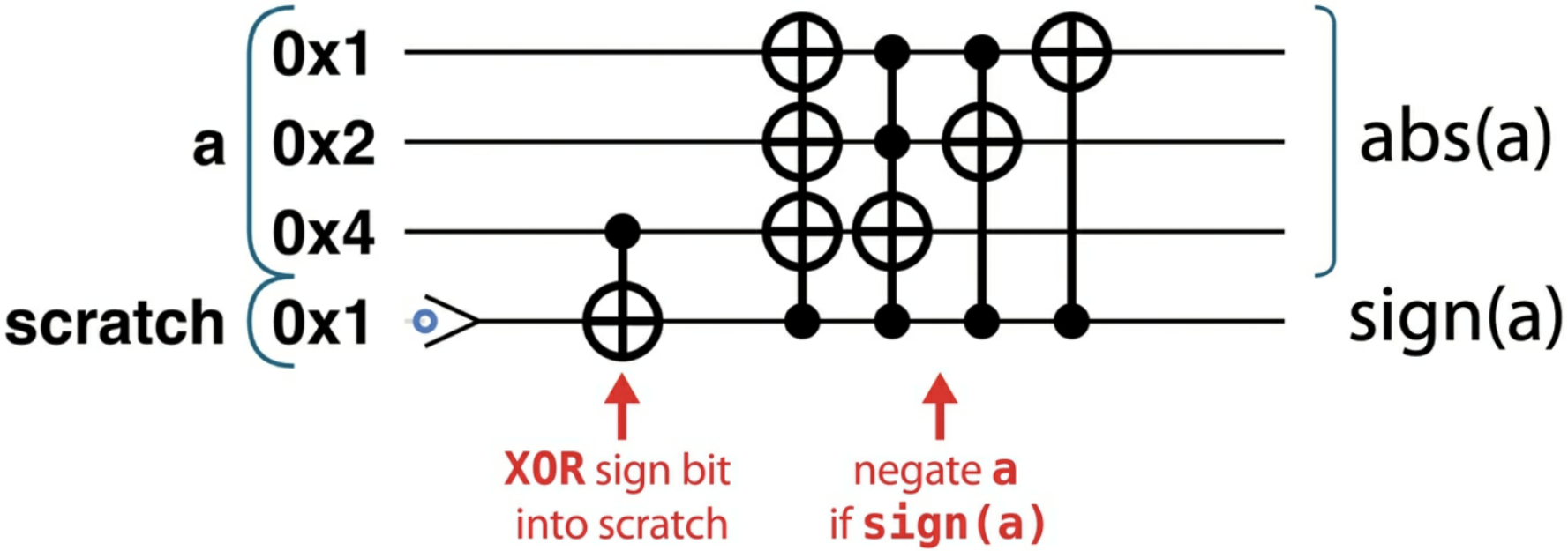


Figure 5-16. Scratch qubits can make an irreversible operation reversible

Ils sont bien souvent indispensables, on cherchera tout de même à en utiliser le moins possible ... L'astuce ici est donc que le scratch qubit passe a 1 si le bit de signe est a 1, et alors il commandera (en tant que bit de contrôle donc) le changement de signe.

> With the scratch qubit involved, we now *can* find a set of gates that will ultimately implement `abs` on our `a` register. But before we verify in detail that the circuit in Figure 5-16 achieves this, note that the `CNOT` operation we use between our scratch qubit and the `0x4` qubit of `a` (which tells us the sign of its integer value in two's complement) does not *copy* the sign qubit exactly. Instead, the `CNOT` *entangles* the scratch qubit with the sign qubit. This is an important distinction to note, since we also know that QPU operations cannot copy qubits.

Autrement dit, attention ! On a intriqué le scratch qubit avec *a* par cette opération, ils ne sont plus indépendants (observez le déroulement plus bas).

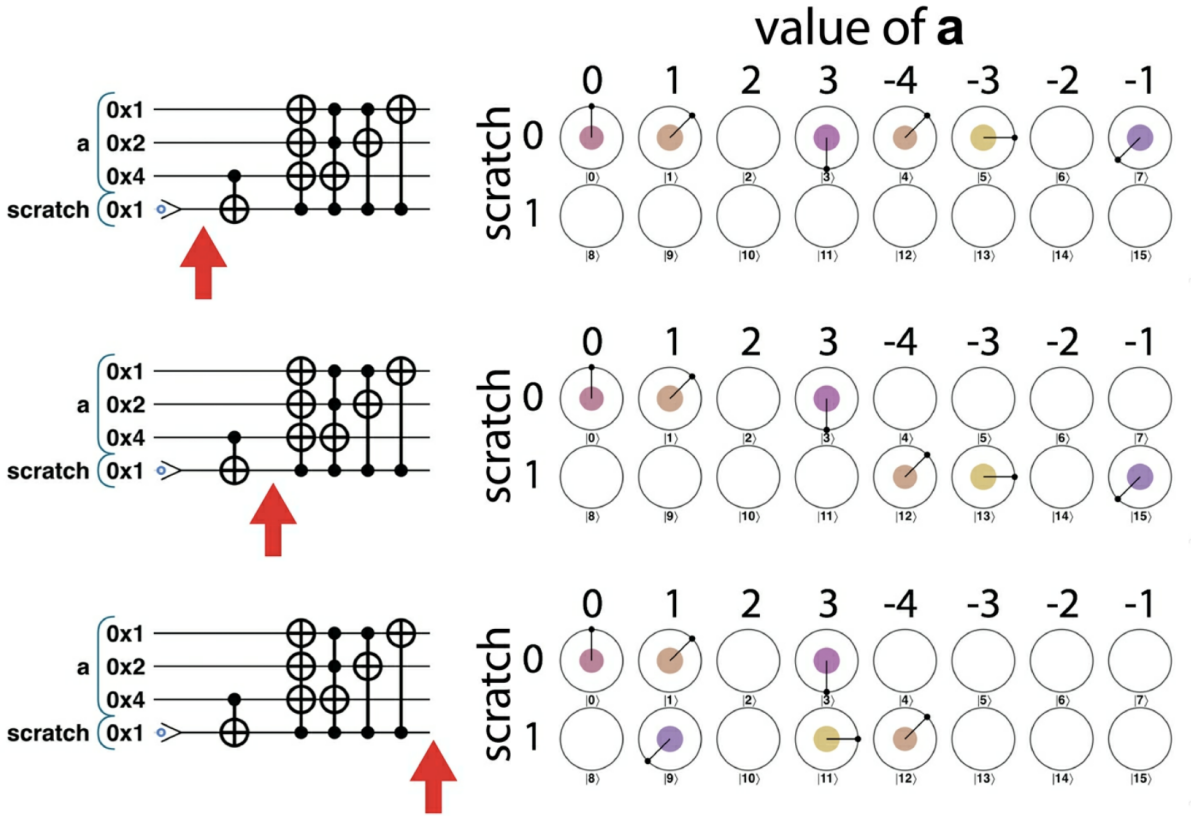


Figure 5-17. Circle notation steps for the absolute value

Attention on rappelle que l'inverse de -4 est -4 (ce sont les 2 exceptions avec le 0 en codage en complément à 2). On observe aussi que a la fin du programme, les deux variables (le scratch qubit et a) sont en effet intriqués ! (On ne peut plus retirer l'un sans nuire à l'autre.)

Notion de décalcul ou uncomputation en Anglais, servant à désintriquer les qubits pour pouvoir après coup les lire (donc les détruire) sans qu'ils n'interfèrent les uns avec les autres :

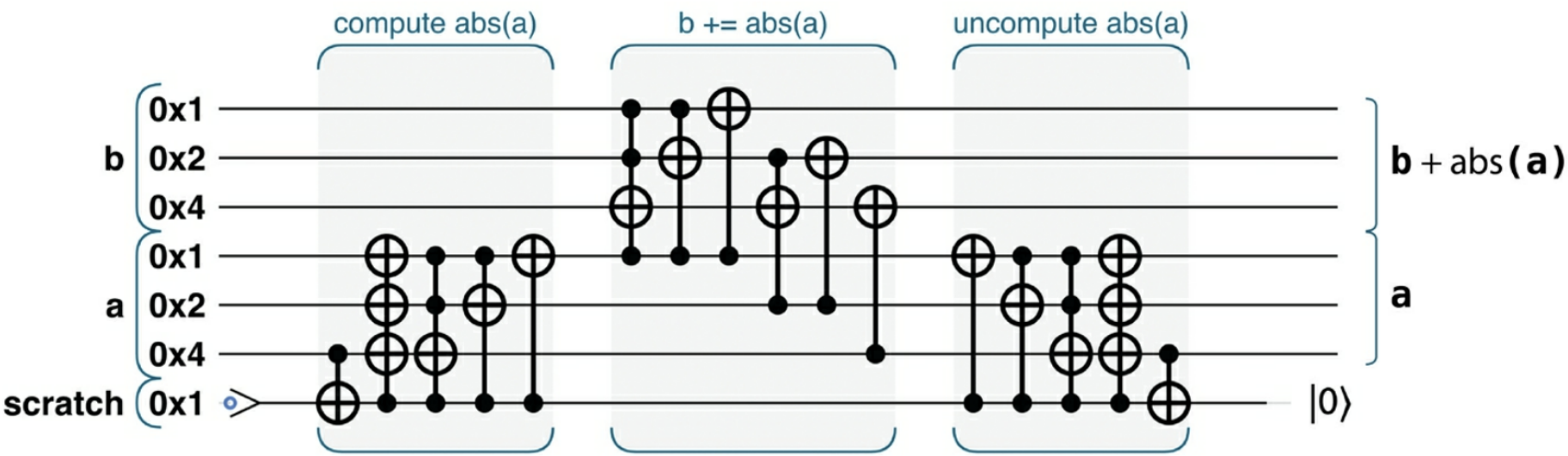


Figure 5-18. Using uncomputation to perform b += abs(a)

Ici, on va donc calculer $abs(a)$ (en utilisant/intriquant un scratch qubit), faire des opérations avec $abs(a)$ (ici on l'ajoute à b question d'en fait quelque chose ...), puis on va restituer a et donc désintriquer le scratch qubit (revenu à l'état initial $|0\rangle$) et on pourra lire a ou b tranquillement.

Même raisonnement pour faire *b xor abs*(*a*) :

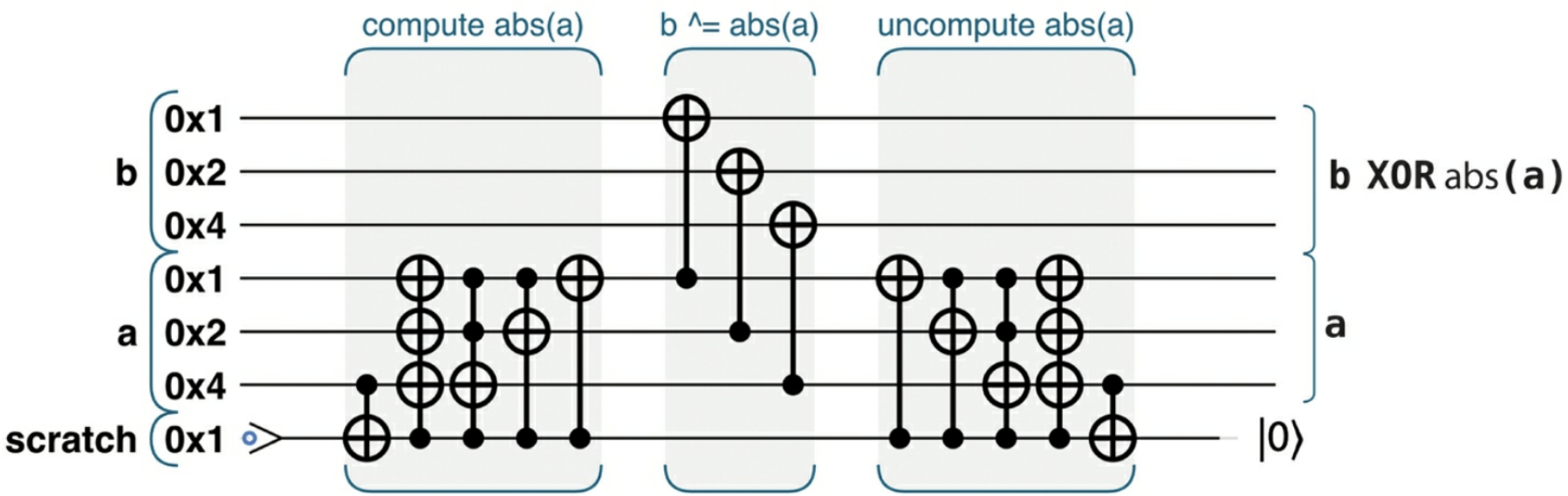


Figure 5-19. Using uncomputation to produce b xor abs(a)

Et ainsi de suite :

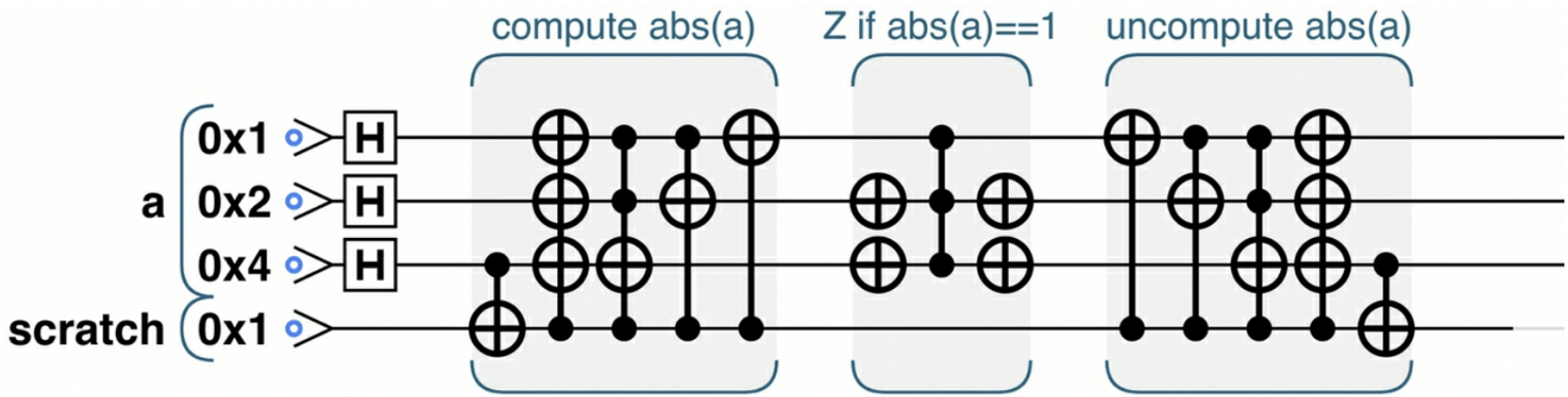


Figure 5-20. Using uncomputation to perform phase(180) if abs(a)==1

Observez combien la réversibilité est donc importante.

Déroulement du programme détaillé ci-dessus:

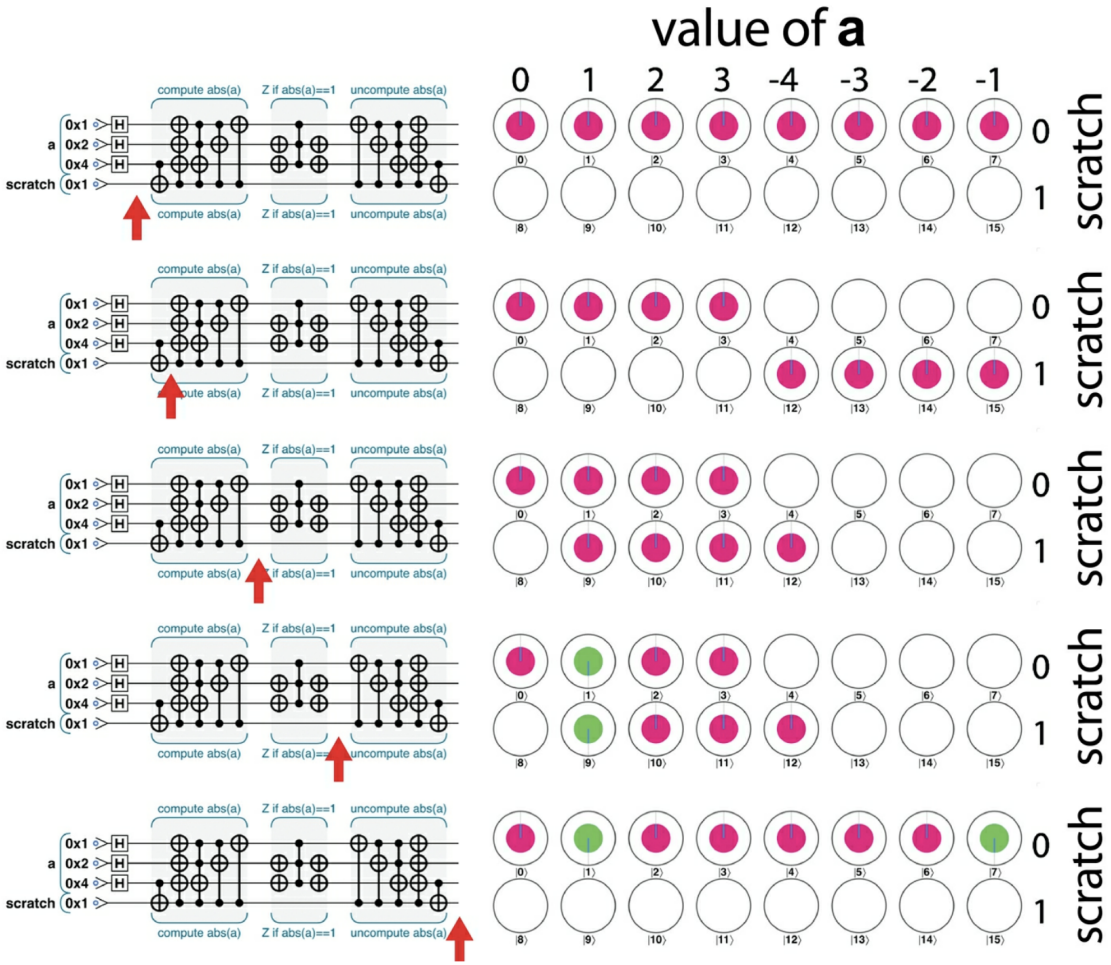


Figure 5-21. Step-by-step walkthrough of using uncompute for conditional phase inversion

Le scratch est revenu à $|0\rangle$ et est donc désintriqué du reste.

Outline

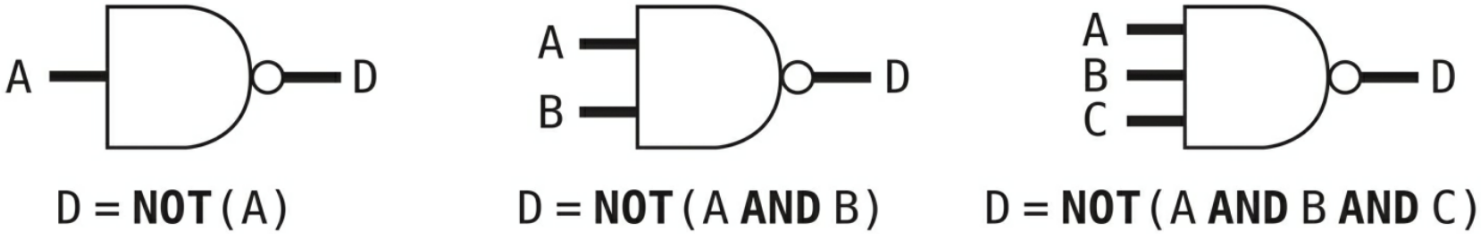


Figure 5-22. Digital NAND gates in different sizes

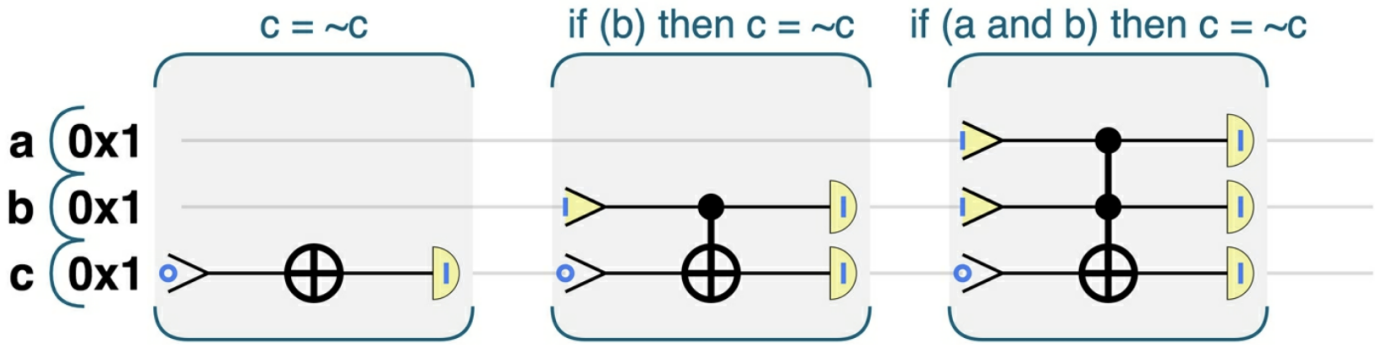


Figure 5-23. Quantum CNOT gates in different sizes

Un exemple de programme avec ces portes quantiques :

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=5-6*.

Example 5-6. Logic using CNOT gates

```
// c = ~c
c.write(0);
c.not();
c.read();

// if (b) then c = ~c
qc.write(2, 2|4);
c.cnot(b);
qc.read(2|4);

// if (a and b) then c = ~c
qc.write(1|2);
qc.cnot(4, 1|2);
qc.read(1|2|4);
```

La Toffoli permet de coder plusieurs portes logiques en une porte quantique :

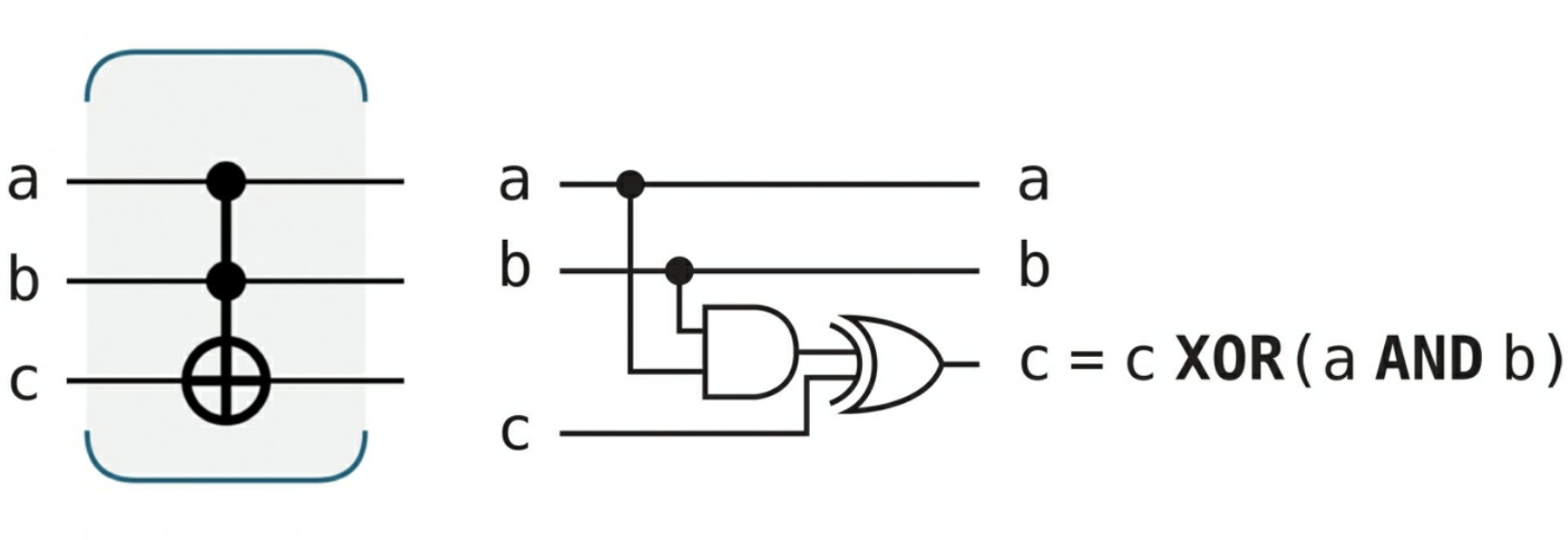
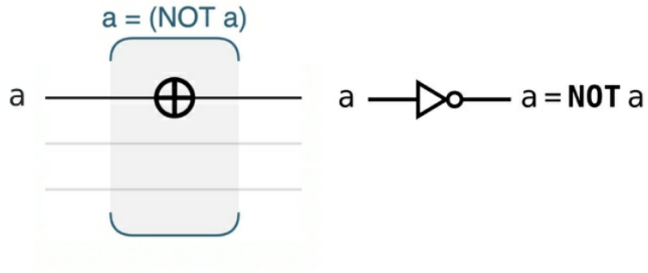
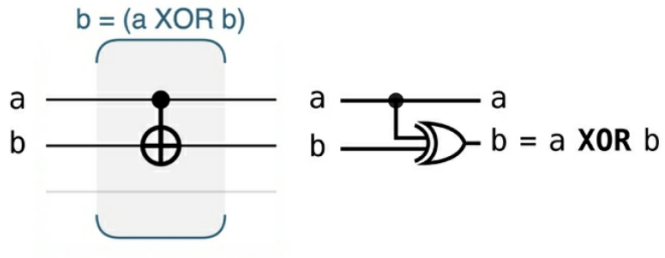
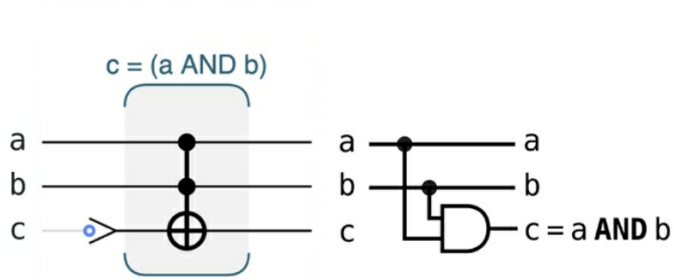


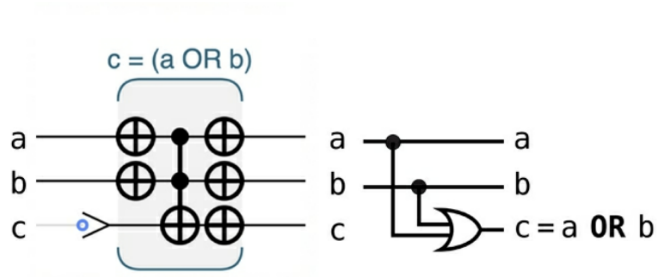
Figure 5-24. The exact digital logic equivalent of our multicondition CNOT gate

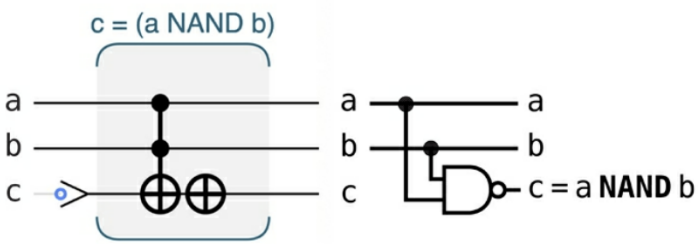
Voici d'autres équivalences :

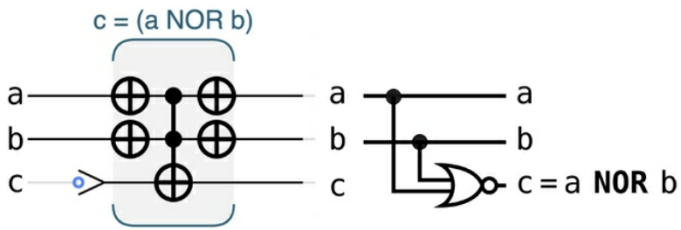












Outline

Imaginons 3 variables A, B, C chacune sur 4 qubits donc avec 16 états; un seul état est retourné pour chacune de ces variables, comment alors les différencier au moment de la lecture ? (On a 1 chance sur 16 d'avoir chaque état pour chaque variable ...)

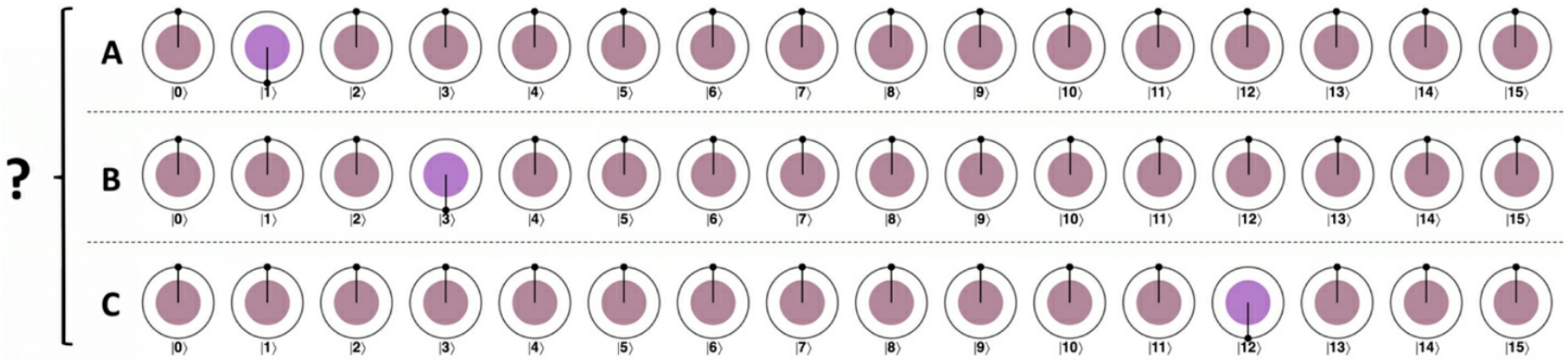


Figure 6-1. Each state has a single flipped value

Supposons l'opération flip-mirror décrite ci-dessous:

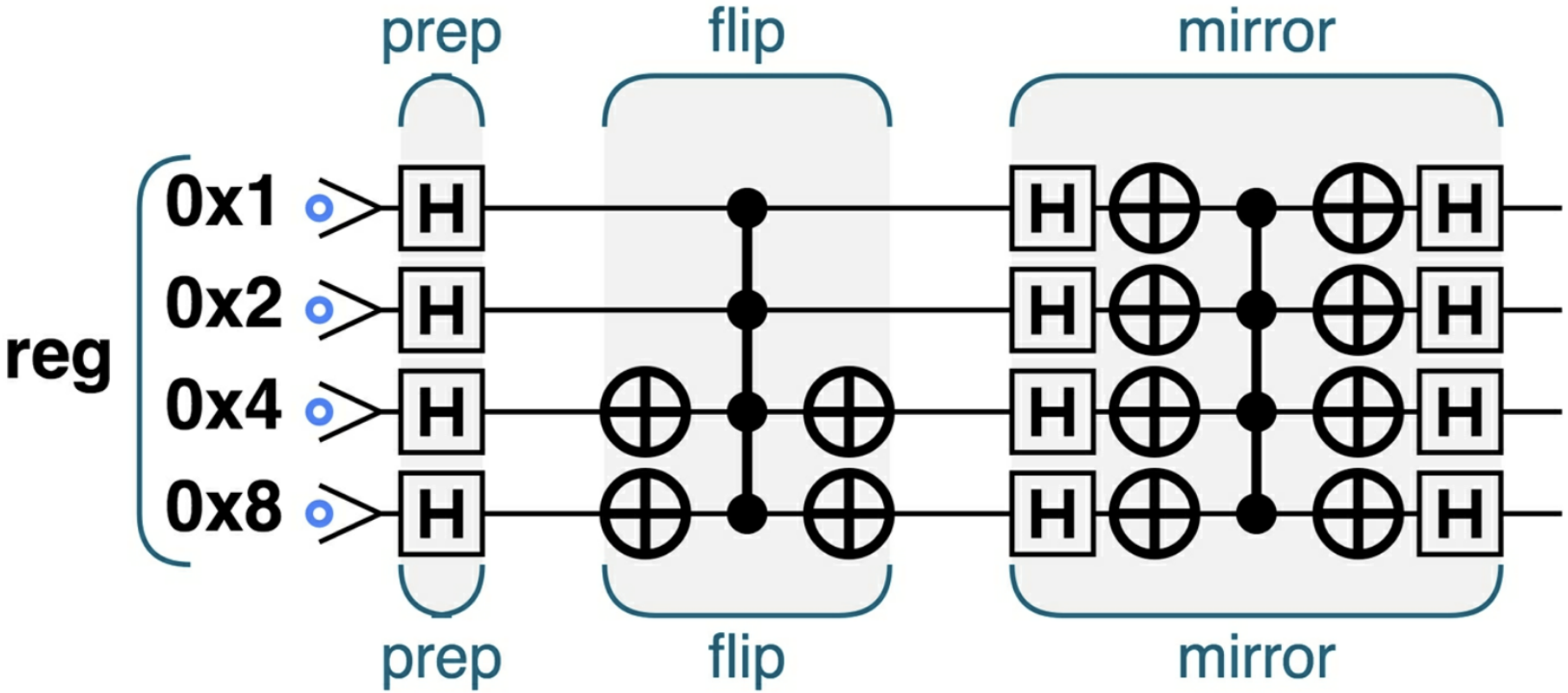


Figure 6-2. Applying the mirror subroutine to a flipped phase

Observez que chaque procédure est symétrique.

En appliquant le flip-mirror, on peut mettre en évidence les états dont la phase était inversée :

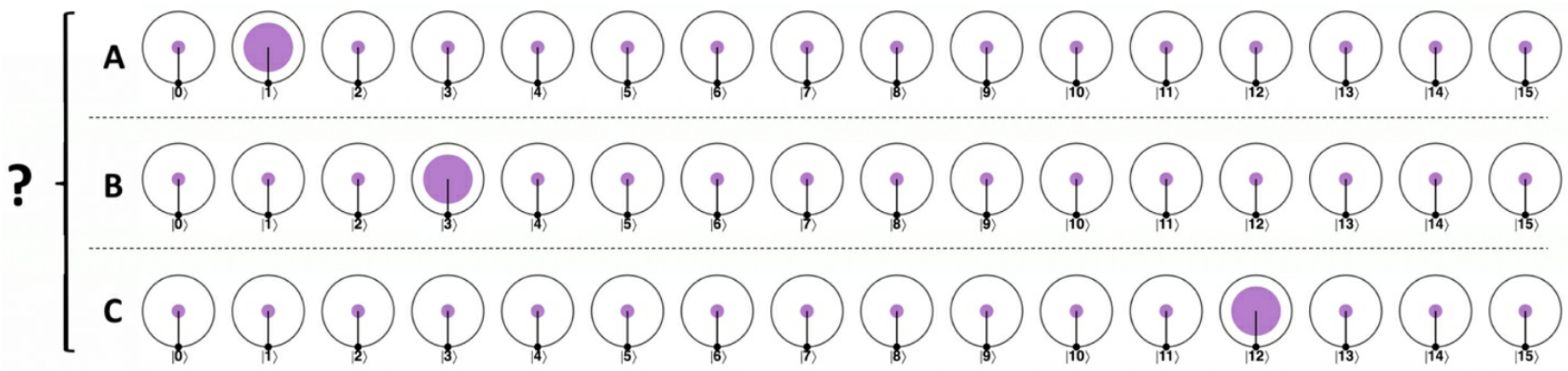


Figure 6-3. After applying the mirror subroutine, phase differences have been converted into magnitude differences

L'amplitude des phases inversées étant amplifiée, on aura une distribution de probabilité d'états différente au moment de la lecture, A, B, et C seront donc différentiables.

Supposons que l'on applique deux fois de suite cette opération quantique:

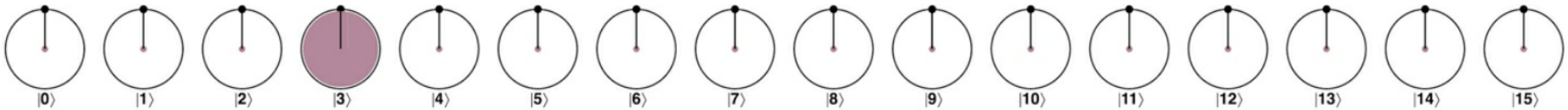


Figure 6-4. After applying flip-mirror a second time on state B

On a encore changé la distribution.

Le regroupement flip-mirror s'appelle AA pour amplitude amplification.

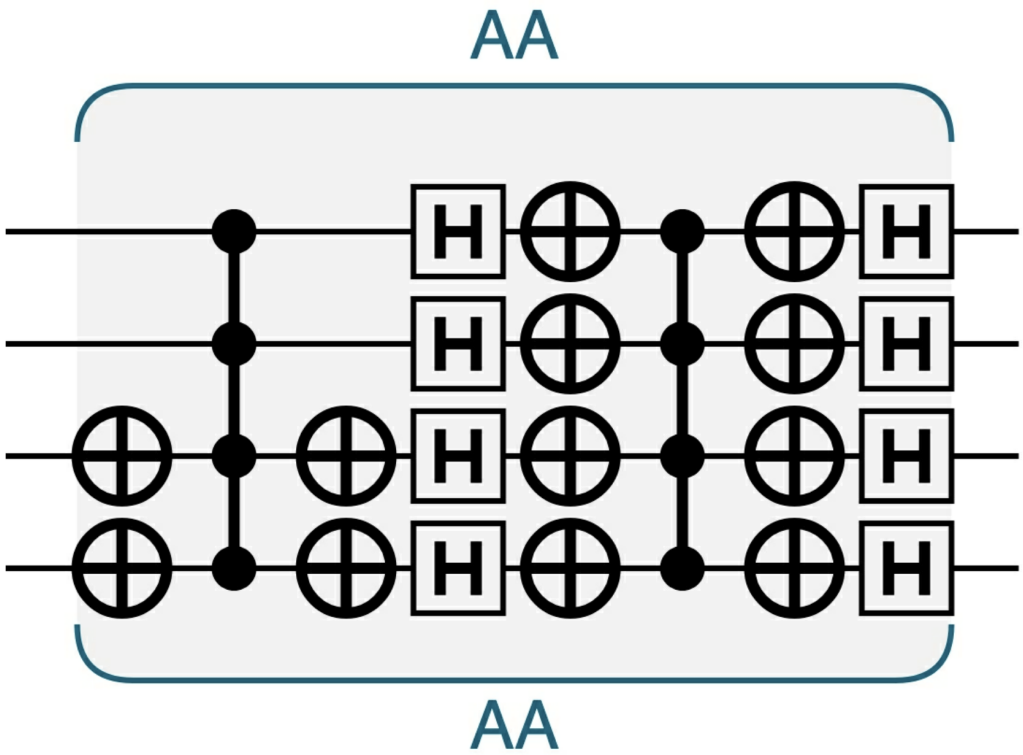


Figure 6-5. A single AA iteration

Voici un algo répétant autant de fois que nécessaire cette opération:

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=6-2*.

Example 6-2. Repeated amplitude amplification iterations

```
var number_to_flip = 3;
var number_of_iterations = 4;

var num_qubits = 4;
qc.reset(num_qubits);
var reg = qint.new(num_qubits, 'reg')
reg.write(0);
reg.hadamard();

for (var i = 0; i < number_of_iterations; ++i)
{
    // Flip the marked value
    reg.not(~number_to_flip);
    reg.cphase(180);
    reg.not(~number_to_flip);

    reg.Grover();

    // Peek at the probability
    var prob = reg.peekProbability(number_to_flip);
}
```

Notez que Grover correspond a l'opération mirror.

En réalité, l'AA ne renforce pas exactement l'amplitude de la phase retournée, observez ici :

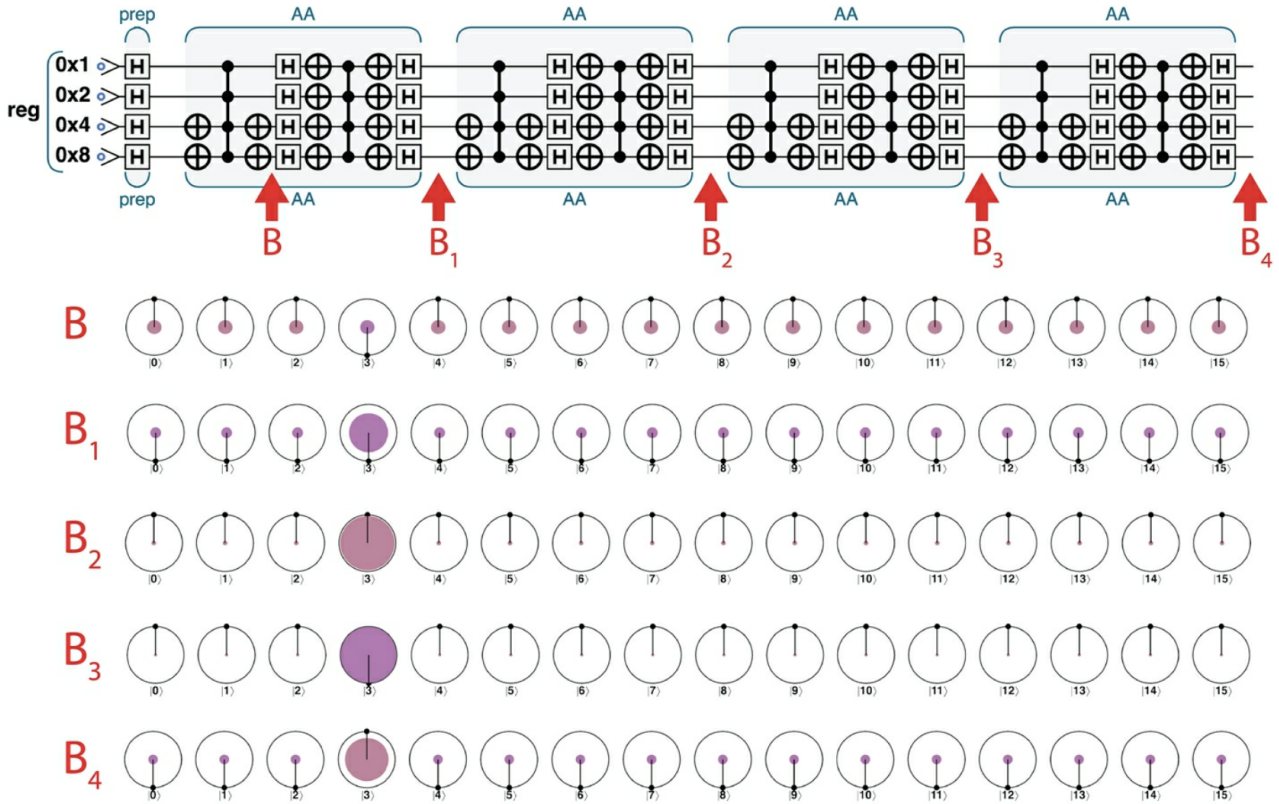


Figure 6-6. The result of applying AA 1, 2, 3, and 4 times to state B — the rows of circles show the state of our QPU register at the positions marked in the circuit

Attention : toutes les phases peuvent être changées simultanément !

L'AA est un phénomène pseudo-cyclique, il dessine une sinusoïde mais n'est pas exactement périodique:

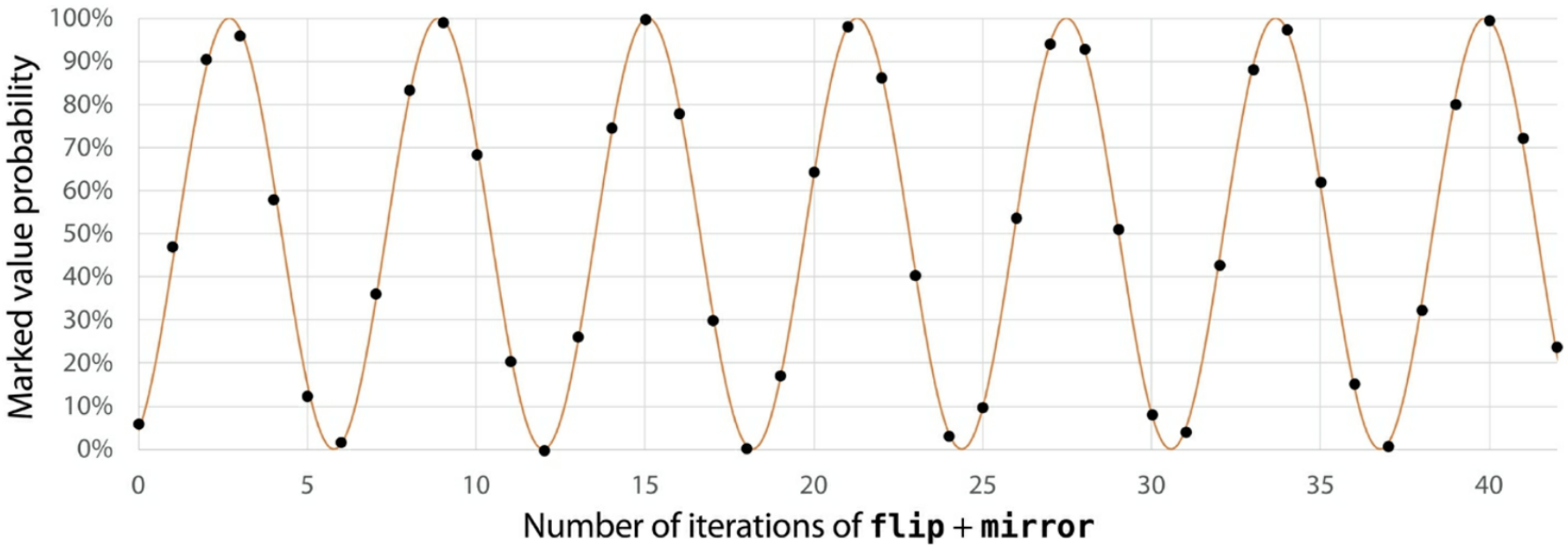


Figure 6-7. Probability of reading the marked value versus the number of AA iterations

Rappel : la probabilité est l'amplitude au carré.

On peut écrire sous forme de formule comment maximiser l'amplitude en fonction du nombre de qubits :

Equation 6-1. Optimal number of iterations in amplitude amplification

$$N_{AA} = \left\lfloor \frac{\pi\sqrt{2^n}}{4} \right\rfloor$$

Rappel : $\lfloor \, . \, \rfloor$ désigne la partie entière inférieure.

Si plus d'une phase est retournée, le changement d'amplitude va se passer différemment. Imaginons que l'on retourne les phases de deux états ainsi:

> But now let's instead flip *two* values (e.g., `n2f=[4,7]`, as shown in Figure 6-10). In this case we ideally want to end up with the QPU register configured so that we will `READ` either of the two phase-flipped values, with zero possibility of `READ`ing any others. Applying multiple AA iterations just like we did for one marked state (albeit with two `flip` subroutines run during each iteration — one for each marked state) yields the results shown in Figure 6-11.

Figure 6-10. Two values flipped

Rappel : "albeit" = cependant.

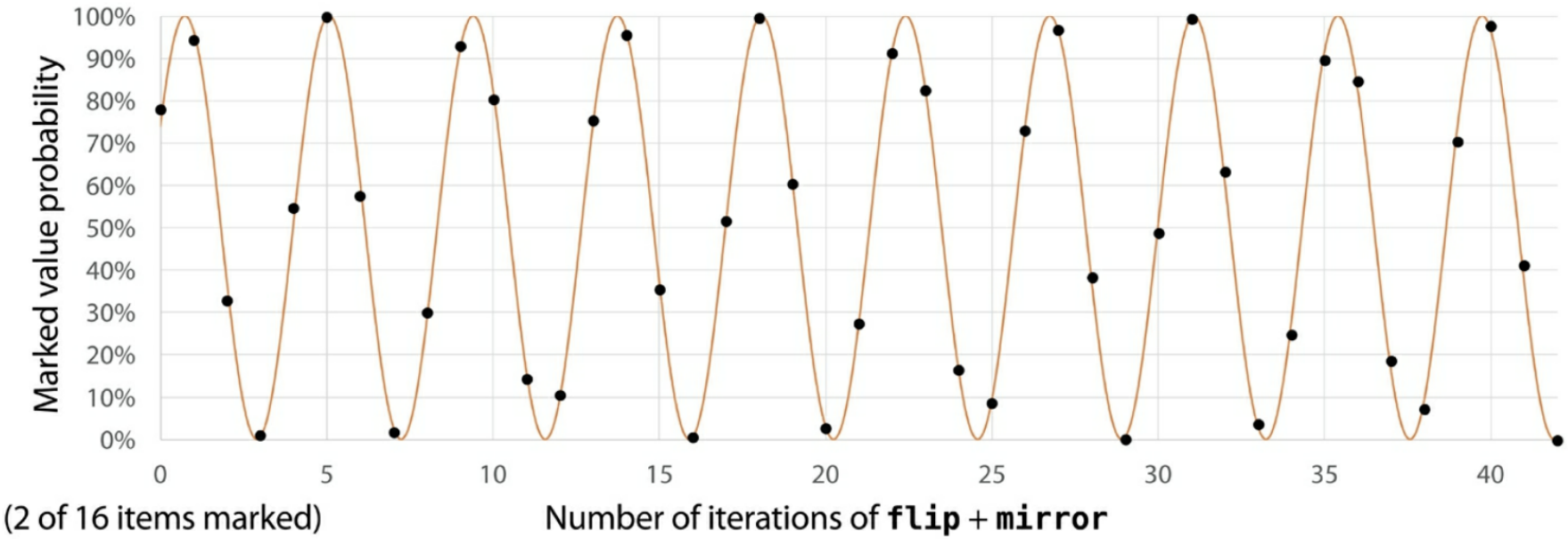


Figure 6-11. Repeated AA iterations with 2 values of 16 flipped

WARNING

Note that in Figure 6-11 the probability shown on the y-axis is the probability of obtaining *either one of the (two) marked values* if we were to READ our register.

Et maintenant si on a 3 phases retournées :

Figure 6-12. Three values flipped

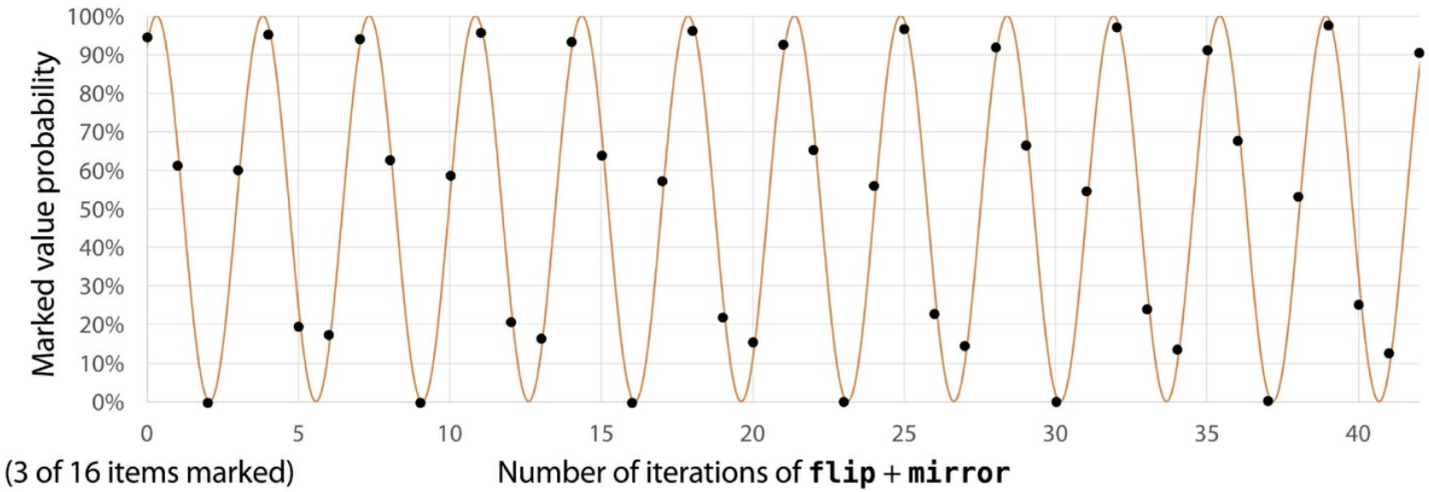


Figure 6-13. Repeated AA iterations with 3 values of 16 flipped

Dans le cas où 4 phases sont retournées, observons l'évolution avec les applications des AA's :

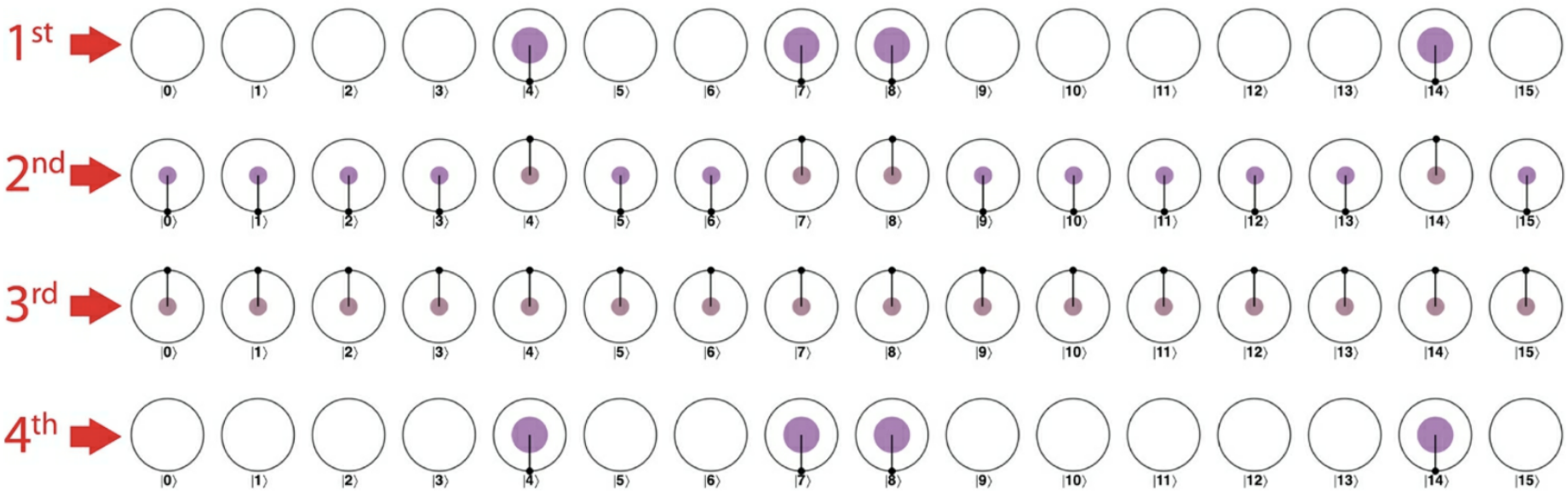


Figure 6-14. Four values flipped

C'est un cas "parfaitement cyclique" (cas particulier).

En effet on obtient ces changements d'amplitudes:

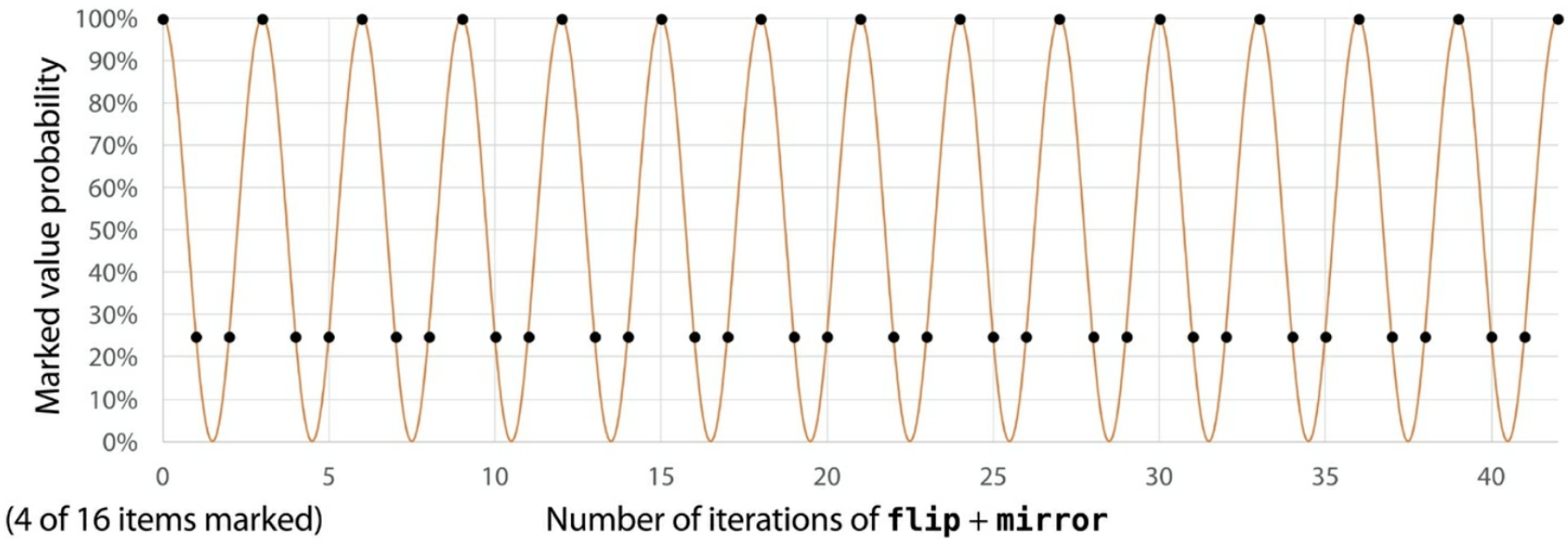


Figure 6-15. Repeated AA iterations with 4 values of 16 flipped

Dans le cas où 7 phases sont retournées :

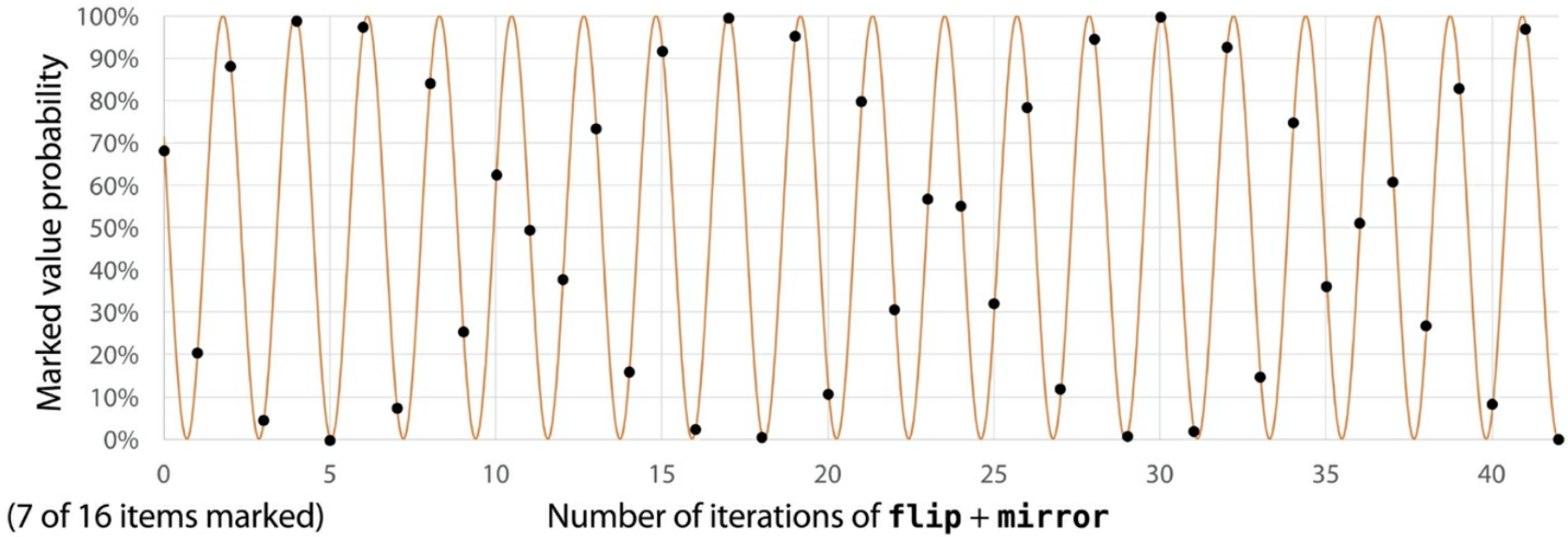


Figure 6-17. Repeated AA iterations with 7 values of 16 flipped

Cas encore plus étonnant: lorsque 8 phases sont retournées, on a préservation des amplitudes !

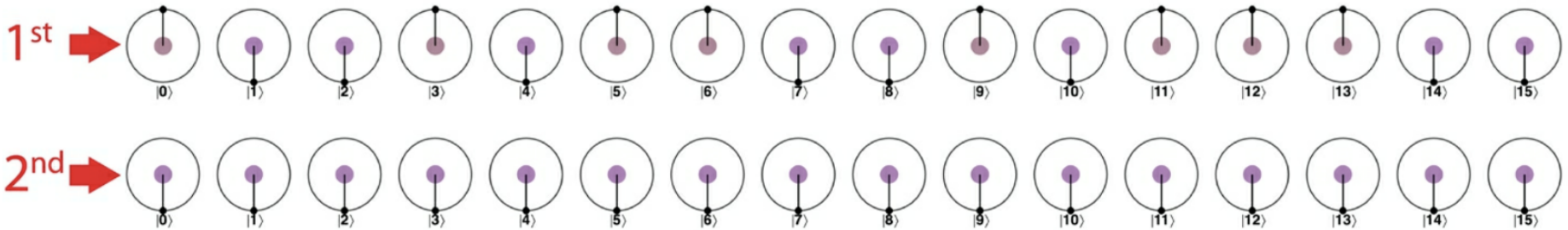


Figure 6-18. Eight values flipped

Note : on a 8 phases retournées, mais comme c'est relatif, peu importe le groupe de 8 phases choisi. Par extension, si on a 3 phases retournées sur 16, c'est donc équivalent à 13 phases retournées sur 16.

Observons en effet le déroulement avec les applications le l'AA :

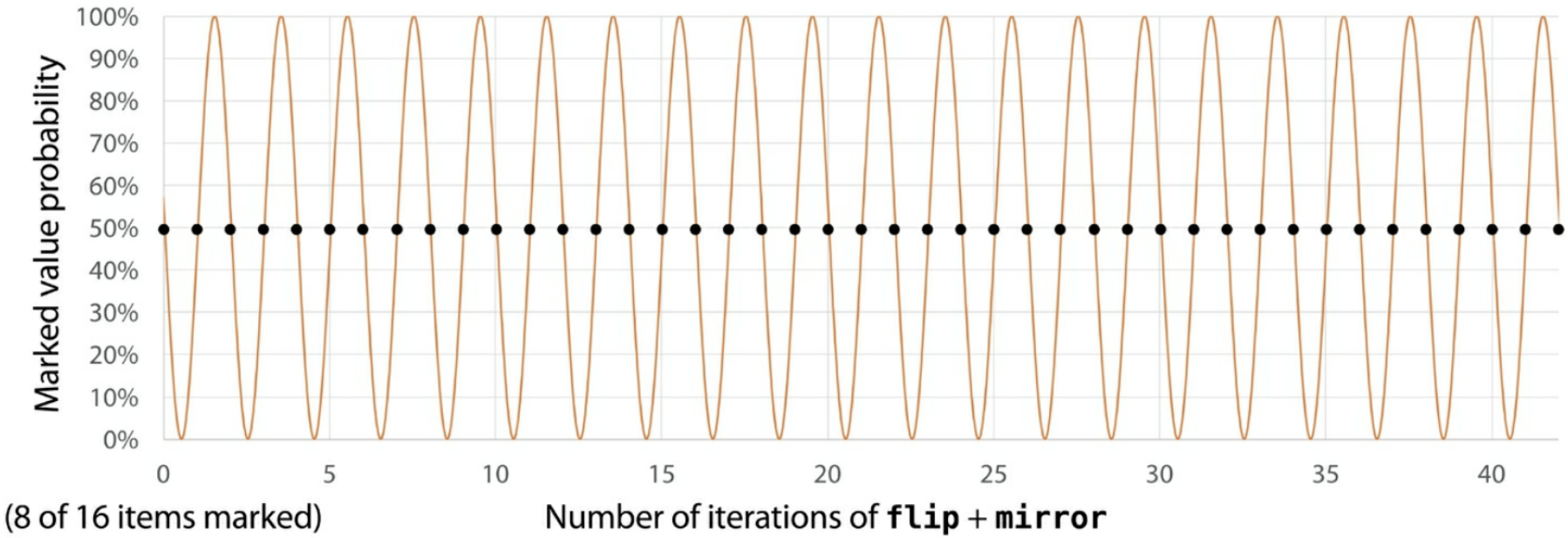


Figure 6-19. Repeated AA iterations with 8 values of 16 flipped

Et voici la formule du nombre optimal d'AA à appliquer lorsque l'on a *m* phases retournées et *n* qubits:

Equation 6-2. Optimal number of iterations for multiple flipped phases

$$N_{AA} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{m}} \right\rfloor$$

Conclusion

This chapter introduced one of the core operations in many QPU applications. By converting phase differences into magnitude differences, amplitude amplification allows a QPU program to provide useful output relating to phase information from a state that would otherwise remain invisible. We explore the full power of this primitive in Chapters 11 and 14.

Outline

Ici on va attaquer une des méthodes les plus puissantes de l'informatique quantique, la transformée de Fourier quantique ou QFT pour Quantum Fourier Transform qui servira dans plein d'autres fonctions étudiées plus tard dans ce cours.

Hidden Patterns

Let's make our state guessing game from Chapter 6 a little harder. Suppose we have a four-qubit quantum register containing one of the three states (A, B, or C) shown in Figure 7-1, but we don't know which one.

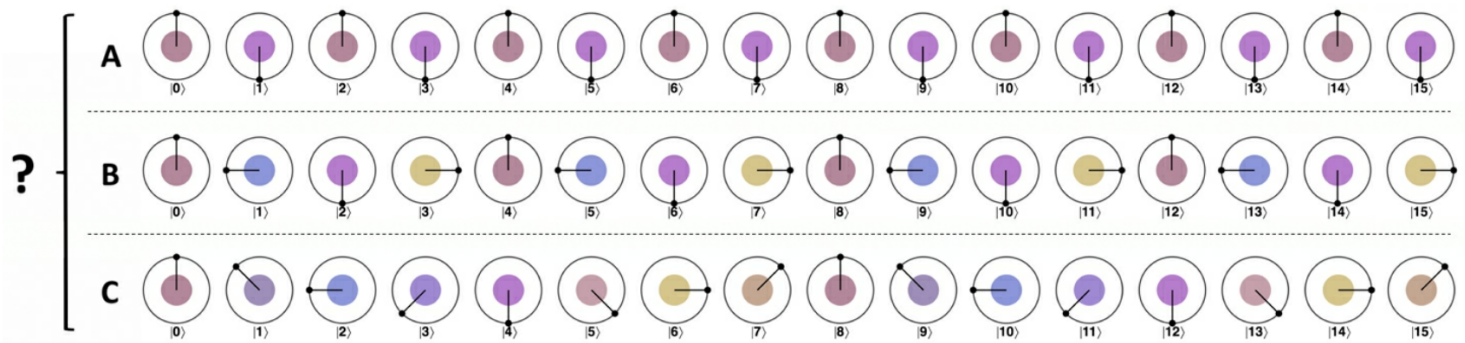


Figure 7-1. Three different states, before applying QFT

Au niveau amplitudes, les registres A, B et C sont indifférentiables, mais on observe des phases différentes.

Appliquons la QFT, on obtient alors :

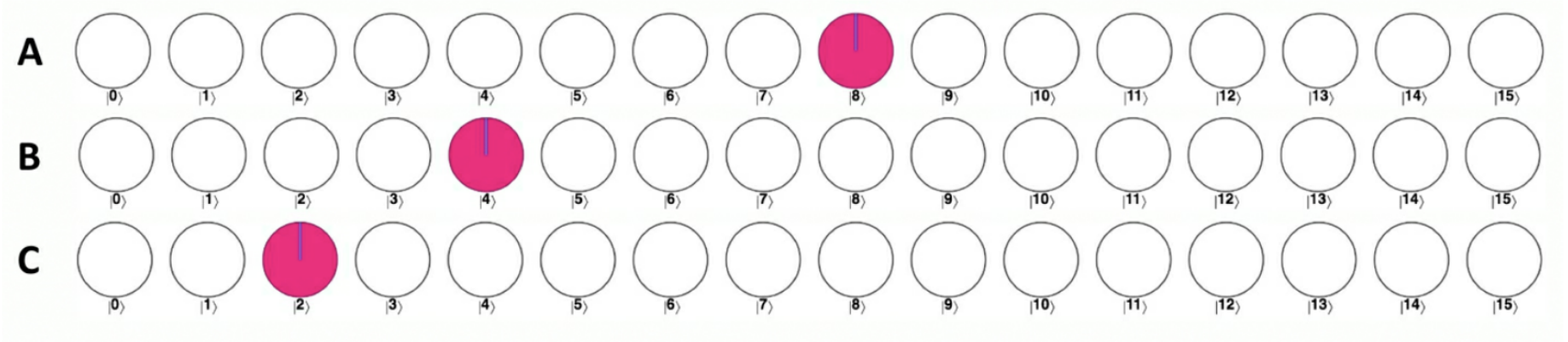
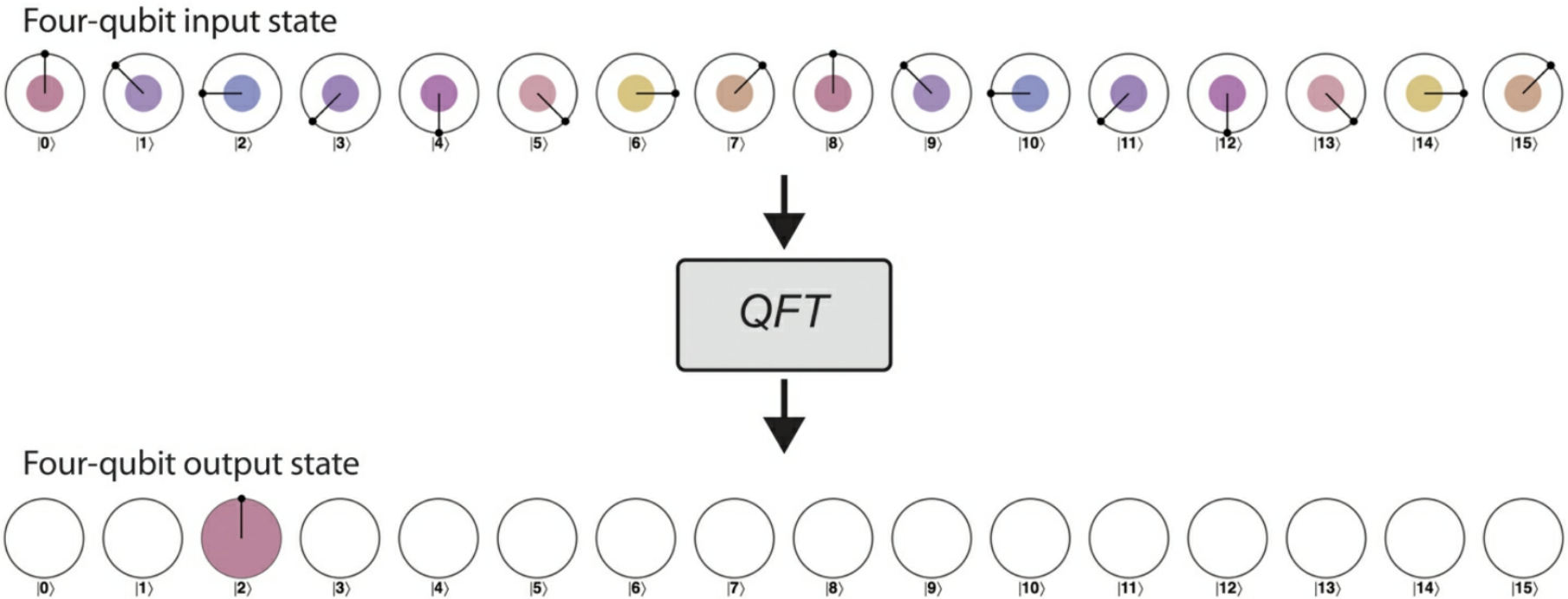


Figure 7-2. The same three states, after applying QFT

Si on se fie à ce que l'on connaît de la transformée de Fourier, ce sont peut-être les amplitudes correspondant à des harmoniques particuliers qui ont été amplifiés ?



On voit que l'on a 2 périodes décrites dans cette entrée au fur et à mesure que l'on augmente dans les états (les états représentent en quelque sorte le "temps"), et c'est l'état 2 qui est d'amplitude maximale.

Regardons comment on a généré le signal :

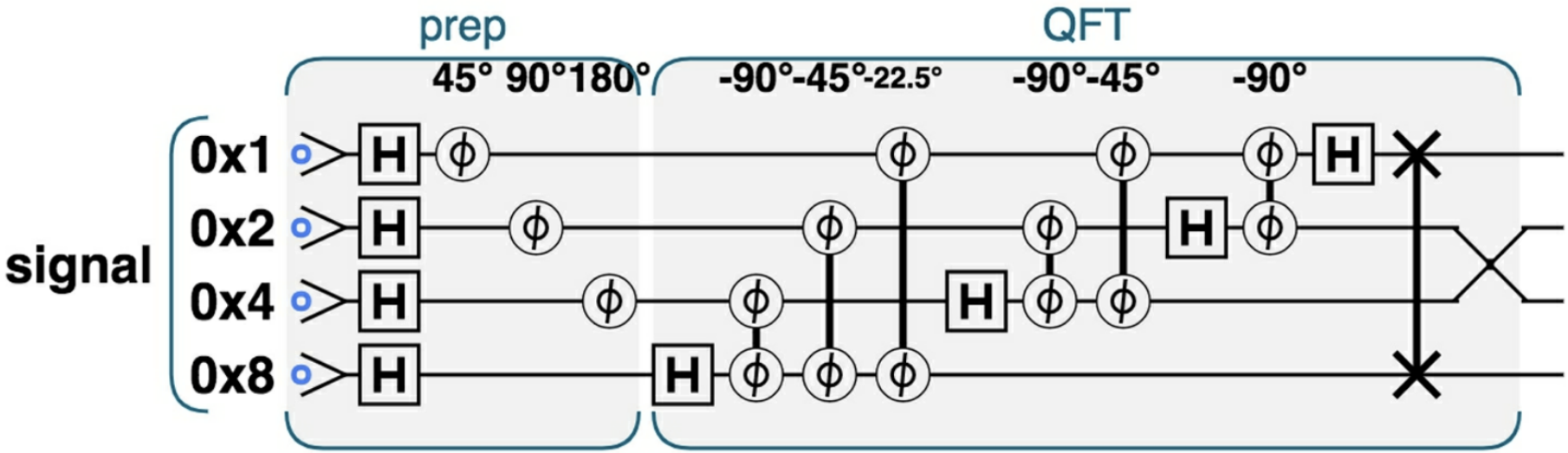


Figure 7-5. QFT of simple QPU register signal

On a imposé des décalages de 45, 90, et 135 degrés aux premier, deuxième et troisième qubits dans notre registre, ce qui explique qu'il y ait un décalage de 45 degrés par état. On applique ensuite la QFT, étrangement compacte (on la détaillera plus tard).

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=7-2*.

Example 7-2. QFT of simple QPU register signal

```
var num_qubits = 4;
qc.reset(num_qubits);
var signal = qint.new(num_qubits, 'signal')

// prepare the signal
signal.write(0);
signal.hadamard();
signal.phase(45, 1);
signal.phase(90, 2);
signal.phase(180, 4);

// Run the QFT
signal.QFT()
```

Notez dans le programme que l'on code les qubits selon leur numérotation hexadécimale (cf. le 1, 2, 4, ...).

Reprenons 4 signaux périodiques, donc a priori aisément différentiables par QFT:

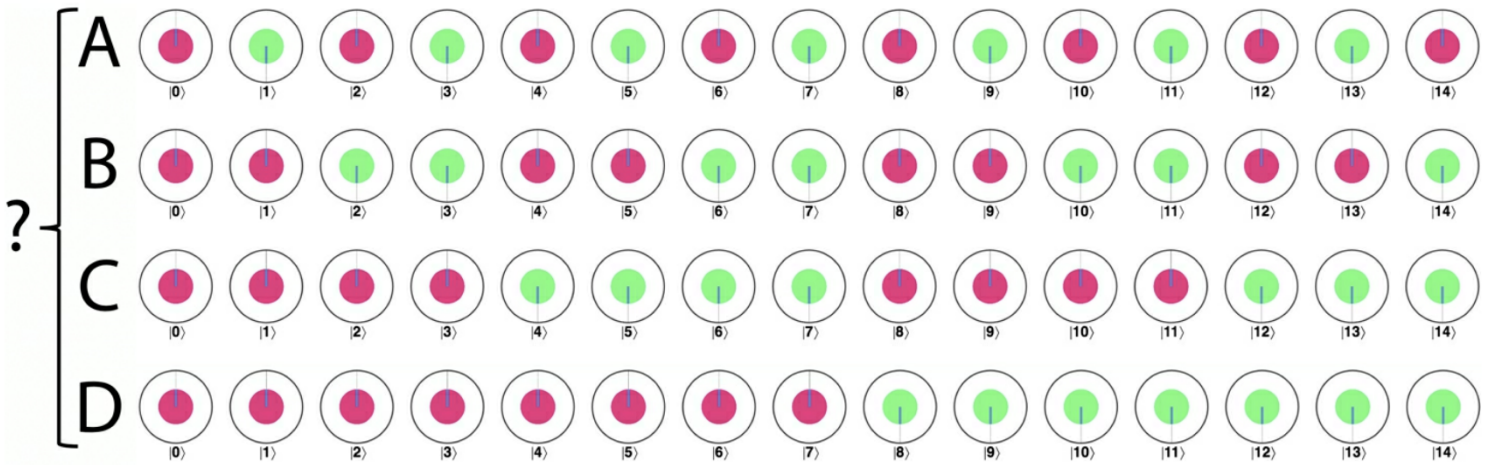


Figure 7-6. Four square-wave signals before applying QFT

En appliquant la QFT, on obtient ceci:

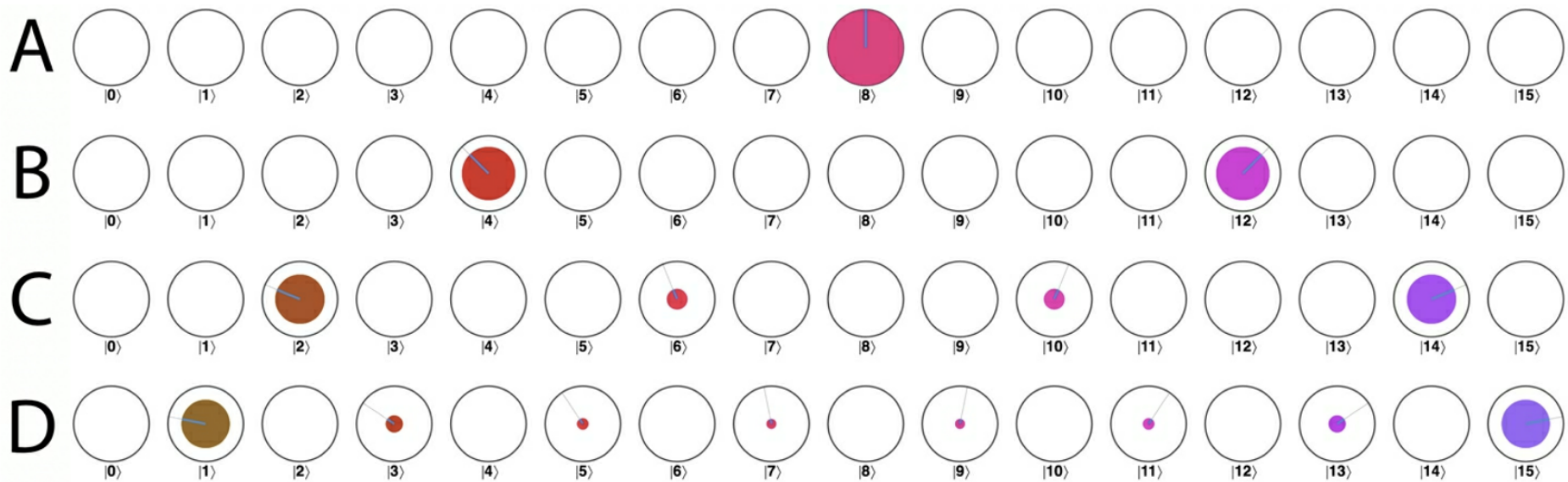


Figure 7-7. The same square-wave signals, after applying QFT

Notez la symétrie au niveau des amplitudes.

Supposons que l'on déphase de 180 degrés sur le deuxième qubit:

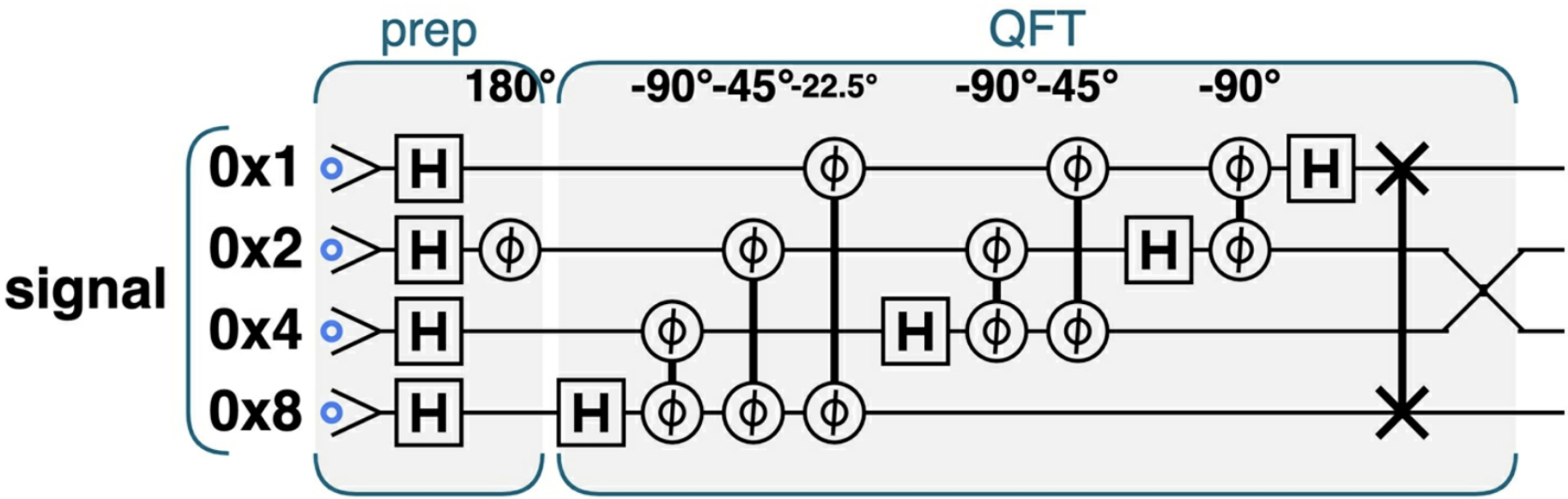


Figure 7-8. The quantum operations used by the QFT

Observons alors la transformée de Fourier discrète :

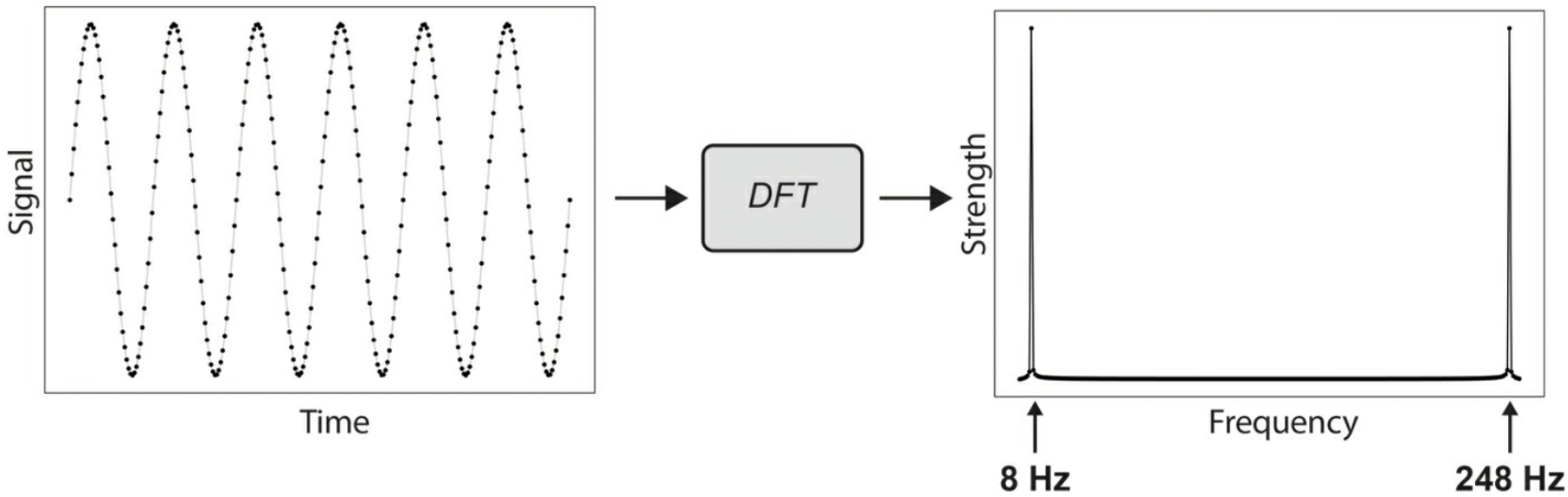


Figure 7-10. The DFT of a simple, single-frequency sine wave

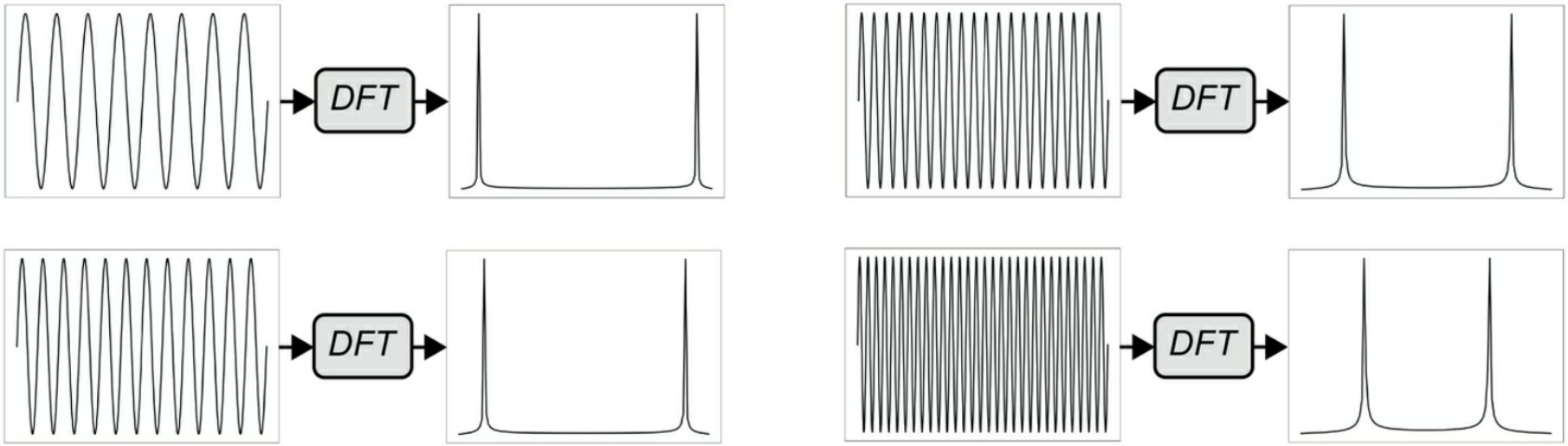


Figure 7-11. Further examples of DFT of real signals (actual sample points omitted for clarity)

On commence à gauche aux fréquences basses, puis jusqu'à la moitié du spectrogramme on augmente, puis on passe aux fréquences négatives.

Générons un signal, et observons :

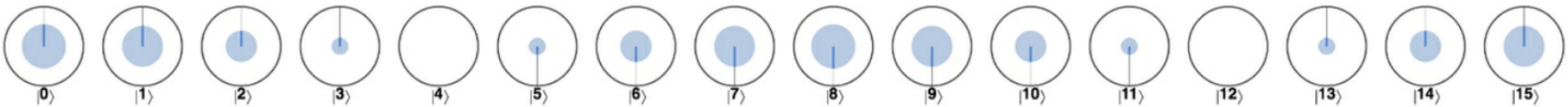


Figure 7-12. Quantum register with signal encoded in terms of magnitudes

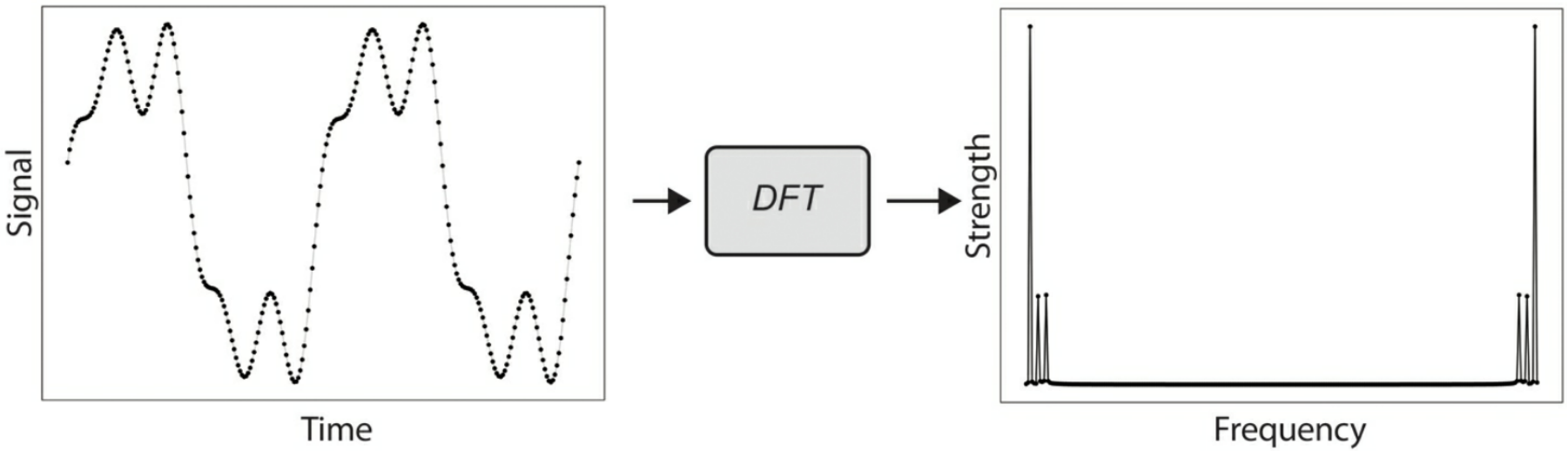


Figure 7-14. DFT on a signal consisting of a mixture of frequencies

La DFT décompose le signal en une somme de fréquences, quelque soit la complexité du signal.

Un autre exemple de signal, dit rectangulaire, et sa DFT:

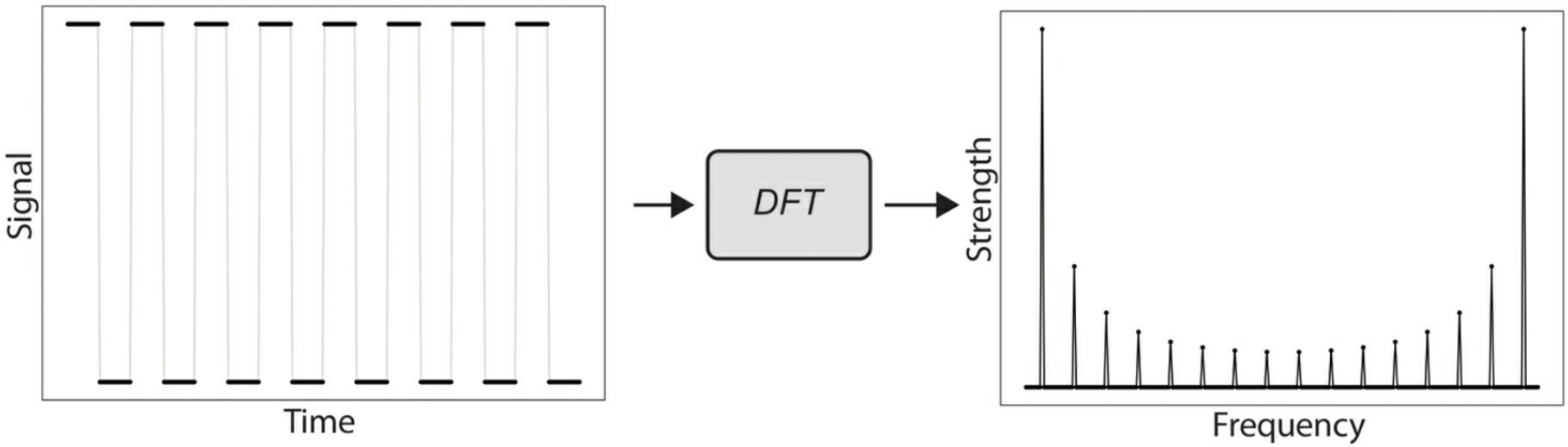


Figure 7-15. DFT of a square wave

Rappel : on a des HF à cause du manque de régularité de la fonction rectangle, même si sa période laisserait penser que l'on a que des MF ou BF.

Générons un signal avec un déphasage sur le 5ème qubit (sur 8) :

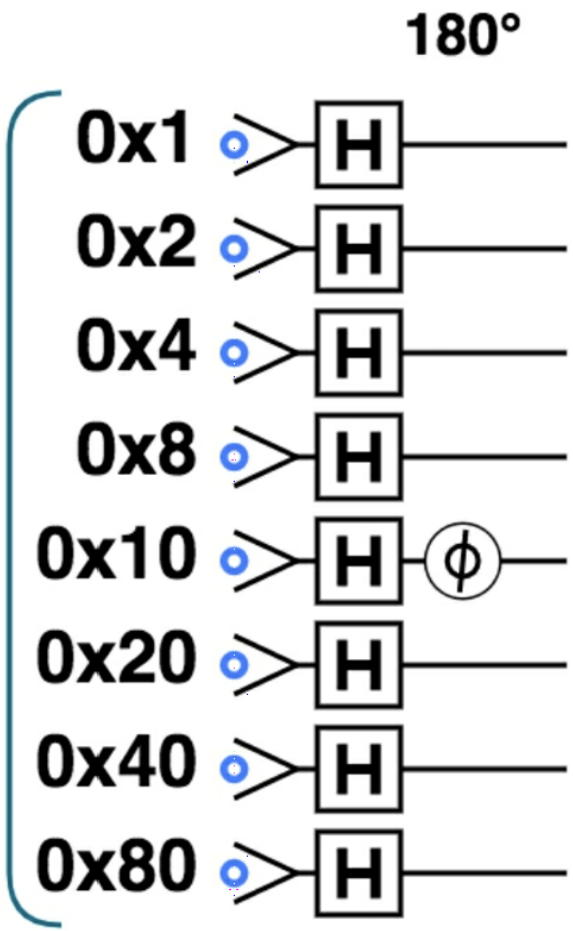


Figure 7-16. The quantum gates needed to prepare an eight-qubit square-wave signal

On change la phase des “1” du 5ème bit pour obtenir cela en représentation intriquée :

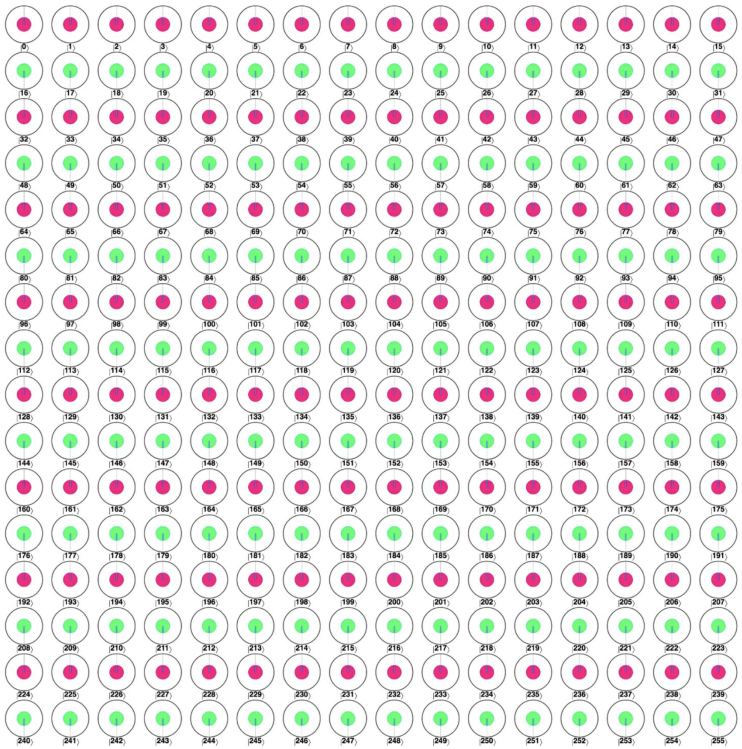


Figure 7-17. Square-wave signal in a qubyte

Rappel mnémotechnique : on déphase sur le premier qubit implique que l’on ait 2^0 qubit à 0 pour commencer puis 2^0 à 1 puis on boucle, donc si l’on déphase le 5ème qubit, on aura 2^4 qubits à 0, puis la même quantité à 1, puis on boucle.

La QFT est alors d'harmonique principal $|8\rangle$ et $|248\rangle$ sur 256 états.

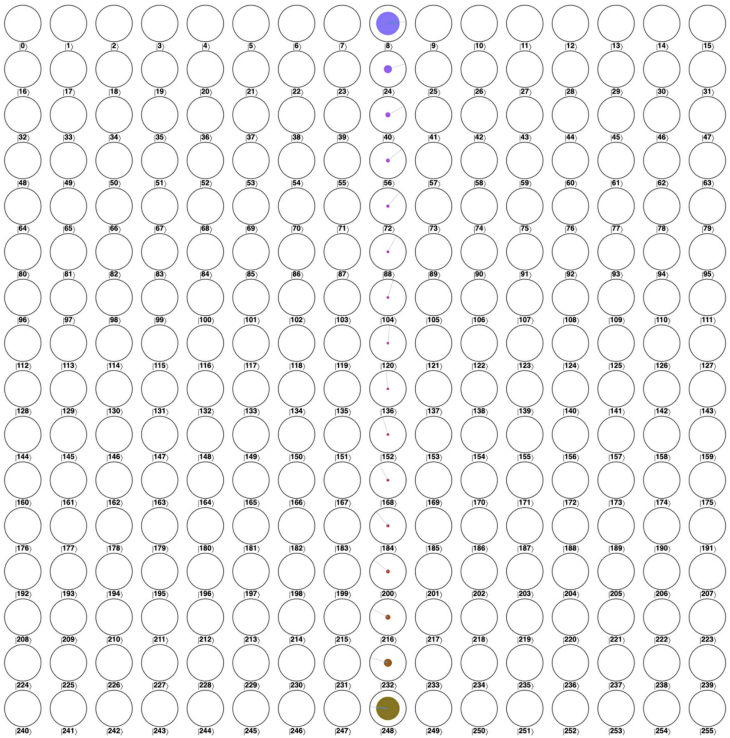


Figure 7-18. QFT output from a square-wave input

Astuce : penser proportionnellement à la fréquence d'échantillonage (256) ou à la fréquence max (128).

Comme les amplitudes après QFT correspondent à des amplitudes d'harmoniques, on peut les lire directement par READ à la nuance près que l'on lira des probabilités et non des amplitudes.

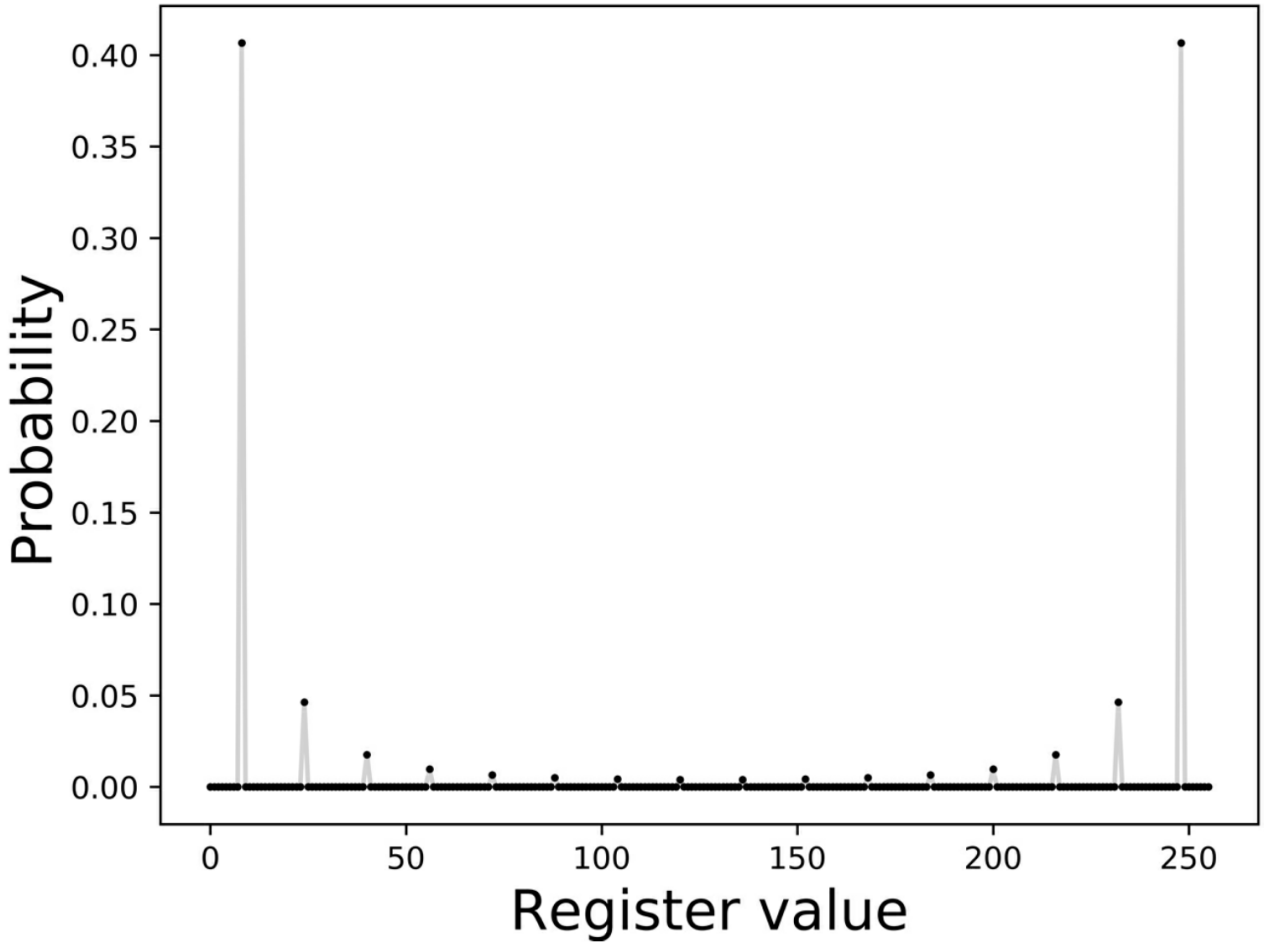
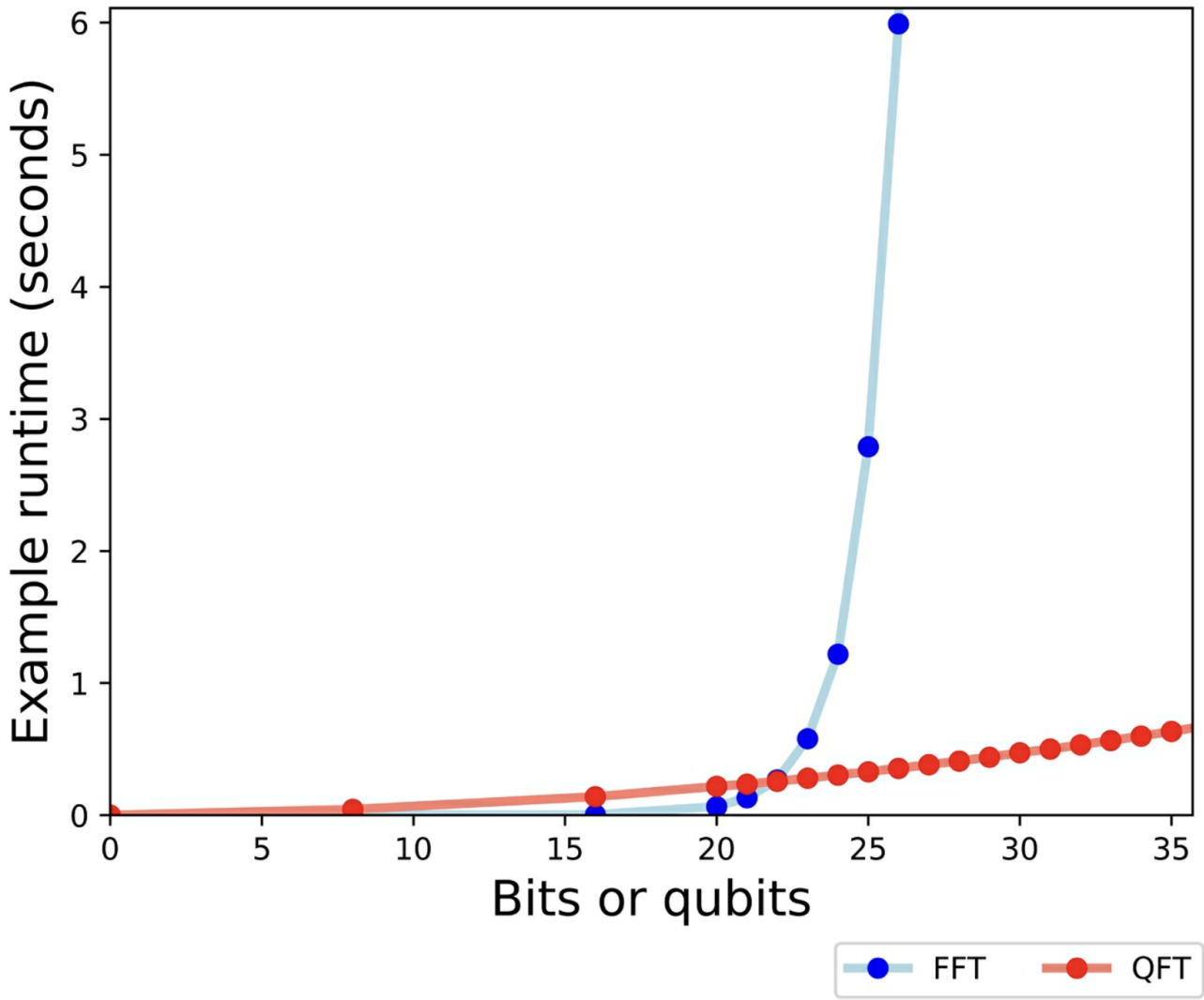


Figure 7-19. Readout probabilities for the QFT of a square-wave qubyte input

Complexités des FFT et QFT :

> The FFT requires a number of operations that grows with the number of input bits n as $O(n2^n)$. The QFT, however — by leveraging the 2^m states available in an m-qubit register — uses a number of gates that grows only as $O(m^2)$.

Quoi dire ? :)


Example runtime (seconds)
0
1
2
3
4
5
6
0
5
10
15
20
25
30
35
Bits or qubits
FFT
QFT

Sans grand étonnement, la QFT inverse est représentée par ce circuit :

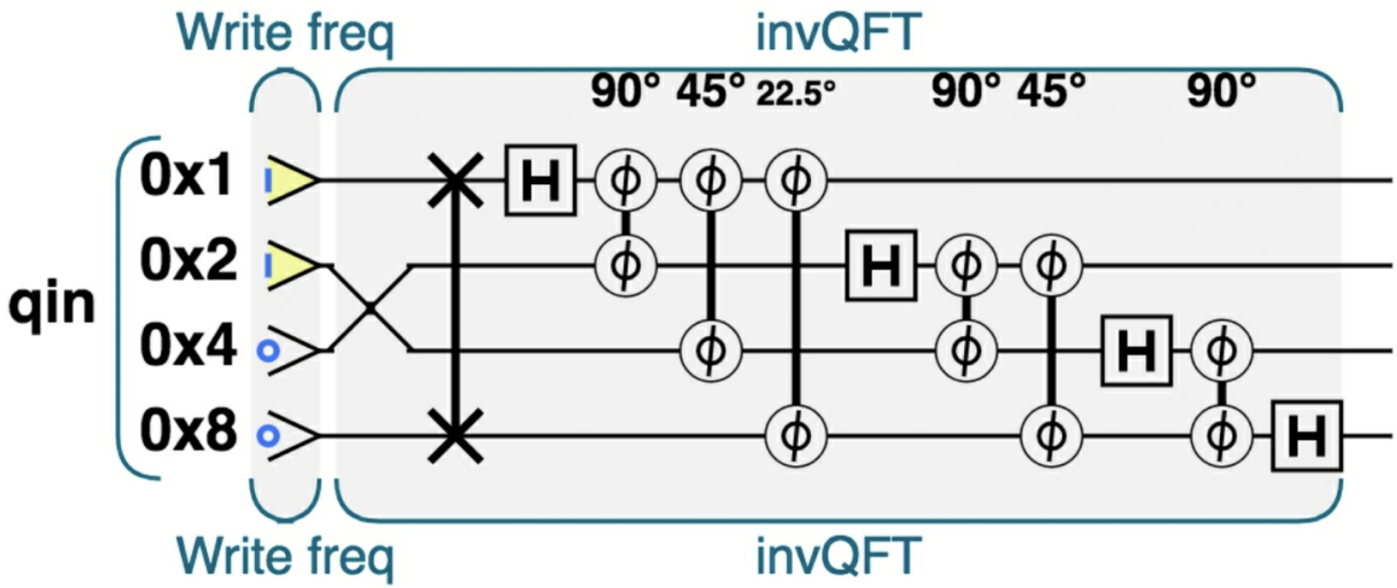


Figure 7-21. The quantum operations needed to produce a signal having periodically varying phase

Lorsque l'on écrira un programme où l'on utilisera la QFT inverse, on pensera donc fréquence et non plus échantillons temporels :

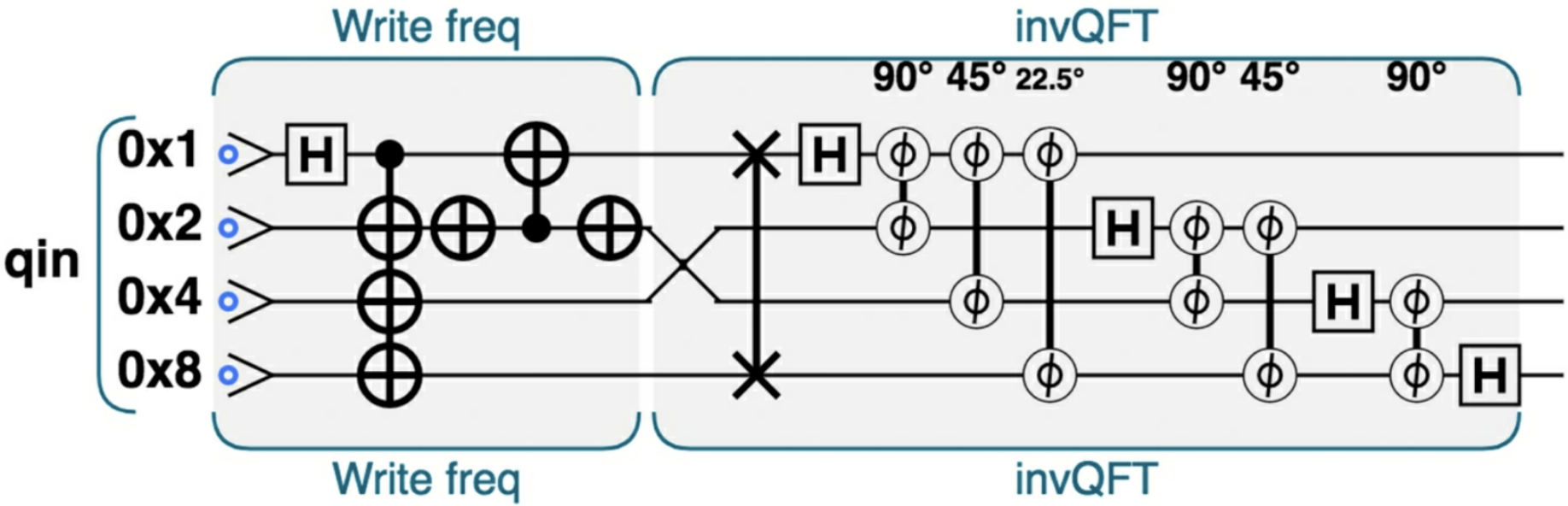


Figure 7-22. The quantum operations needed to produce a signal having periodically varying magnitude

Autrement dit :

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=7-6*.

Example 7-6. Prepare a state with invQFT

```
// Setup
var num_qubits = 4;
qc.reset(num_qubits);
var qin = qint.new(num_qubits, 'qin');
qin.write(0);

// Write the frequencies we want to register
qin.had(1);
qc.cnot(14,1);
qin.not(2);
qc.cnot(1,2);
qin.not(2);

//Inverse QFT to turn into a signal
qin.invQFT()
```

Si l'on veut par exemple faire du traitement de signal, on aura donc possibilité simplement de faire la QFT, traiter le signal au niveau spectral, puis revenir en temporel via la QFT inverse :

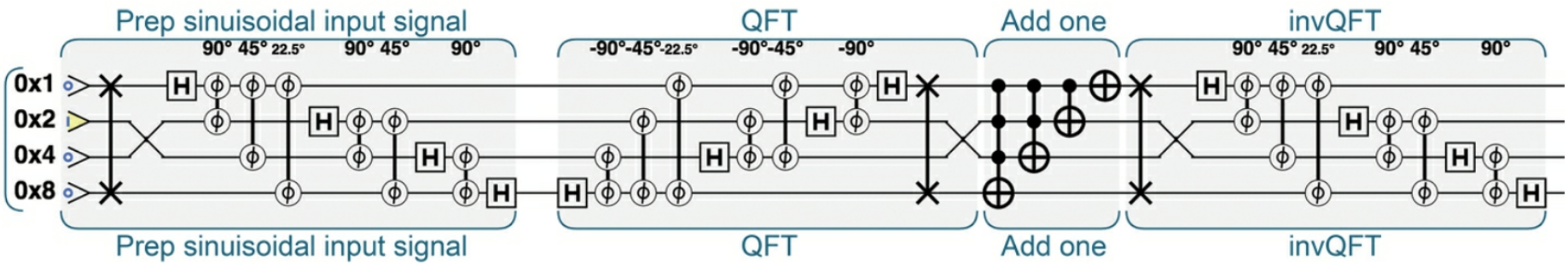


Figure 7-23. Simple example of using the QFT and inverse QFT to manipulate the frequency of a signal

Ce qui donne le programme suivant :

SAMPLE CODE

Run this sample online at *http://oreilly-qc.github.io?p=7-7*.

Example 7-7. QFT frequency manipulation

```
// Set up input register
var n = 4;

// Prepare a complex sinusoidal signal
qc.reset(n);
var freq = 2;
qc.write(freq);
var signal = qint.new(n, 'signal');
signal.invQFT();

// Move to frequency space with QFT
signal.QFT();

// Increase the frequency of signal
signal.add(1)

// Move back from frequency space
signal.invQFT();
```

Imaginons que l'on travaille avec un plus grand nombre de qubits, il faut évidemment tous les mettre en relation pour faire la décomposition fréquentielle, on obtient alors :

Figure 7-24 shows the fundamental QPU operations used for performing the QFT on a qubyte signal.

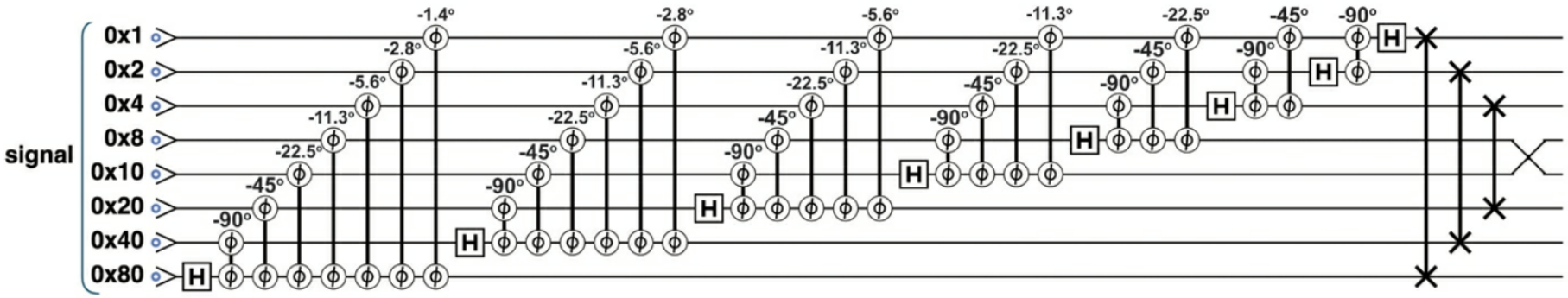


Figure 7-24. Quantum Fourier Transform, operation by operation

Notez le pattern qui se répète (en quelque sorte fractal).

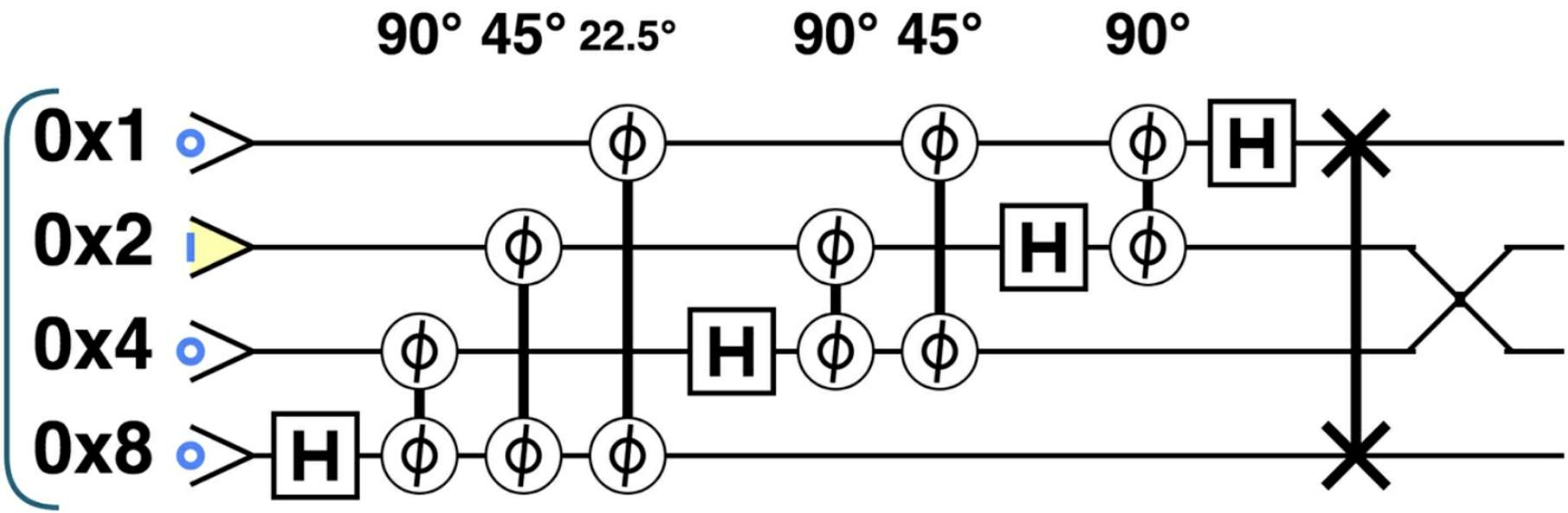
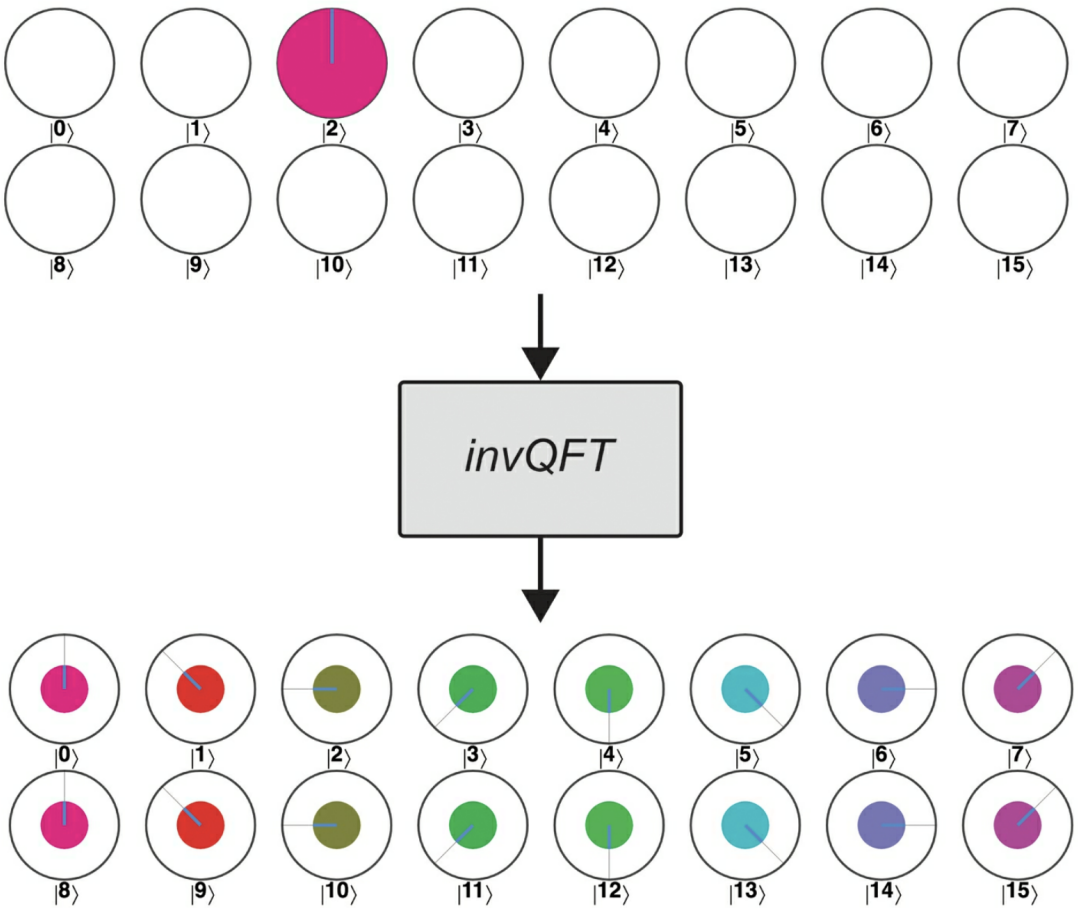


Figure 7-25. Four-qubit inverse QFT

Remarquons aussi qu'à la fin de la QFT inverse, on aura un échange entre le premier et le dernier qubit, entre le deuxième et l'avant dernier qubit, etc. On fait cela par commodité pour que la représentation graphique de la QFT (inverse) soit plus simple.

Remarquons un moyen super simple de créer le même signal que précédemment mais avec beaucoup moins de lignes de code :



On notera les sous-programmes a base de HAD et de PHASE (un par qubit) :

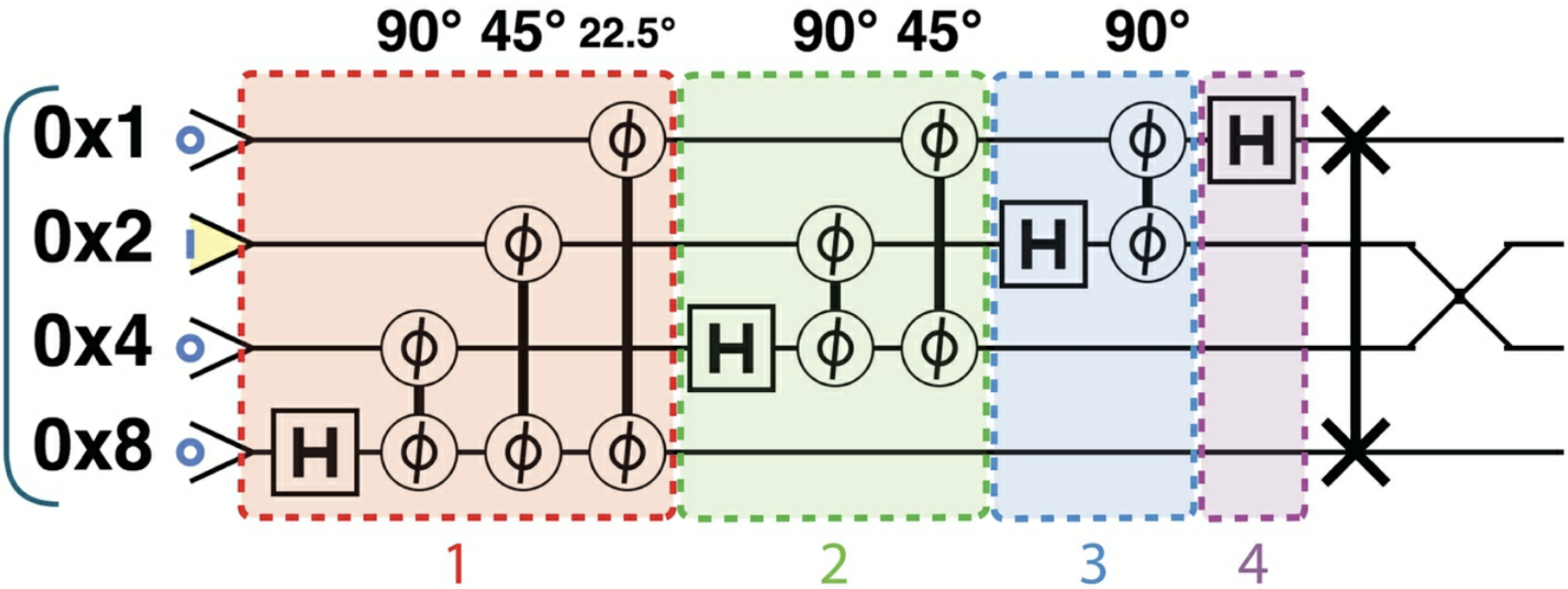


Figure 7-31. The four subroutines of invQFT

Conclusion

In this chapter you have learned about one of the most powerful QPU primitives, the Quantum Fourier Transform. While the AA primitive allowed us to extract information about discrete values encoded in the phases of our register, the QFT primitive enables us to extract information about *patterns* of information encoded in the QPU register. As we will see in Chapter 12, this primitive is at the core of some of the most powerful algorithms that we can run on a QPU, including Shor's algorithm, which first kick-started mainstream interest in quantum computing.

Outline

On va introduire la notion d'état propre:

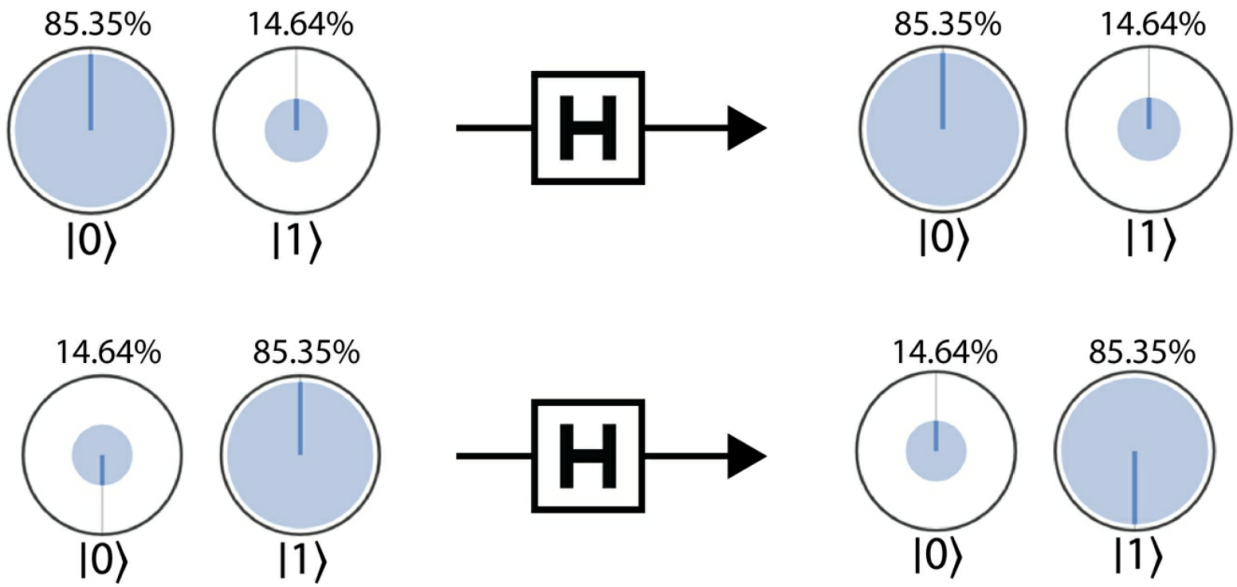


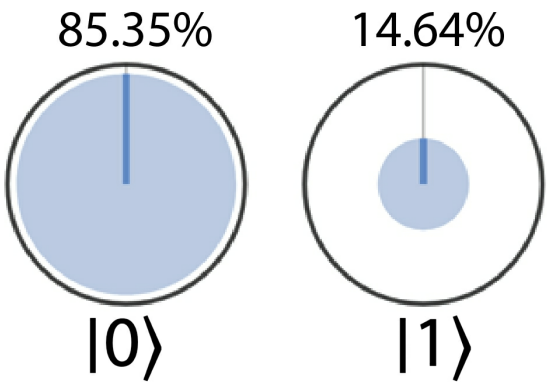
Figure 8-2. Action of HAD on eigenstates

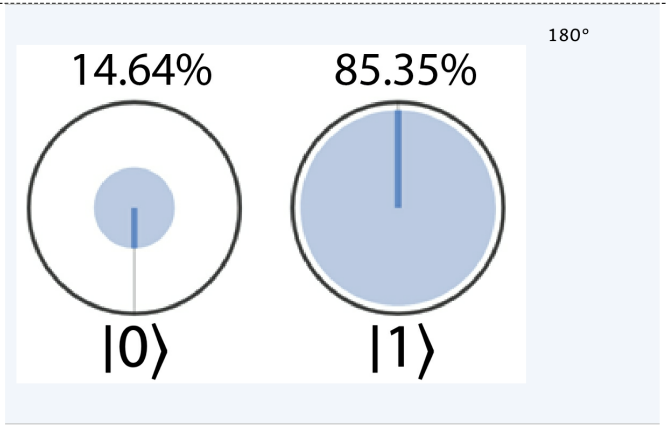
On voit que la sortie est "proportionnelle" a la sortie (dans le sens mutipliée par un complexe), on a donc au moins deux états propres pour HAD (en réalité on peut la coder en matrice 2*x*2 donc elle a 2 états propres au plus).

A un état propre, il correspond une phase propre

Table 8-1. Eigenstates and eigenphases of HAD

Eigenstate	Eigenphase
85.35% |0⟩ + 14.64% |1⟩	0°
14.64% |0⟩ + 85.35% |1⟩	180°



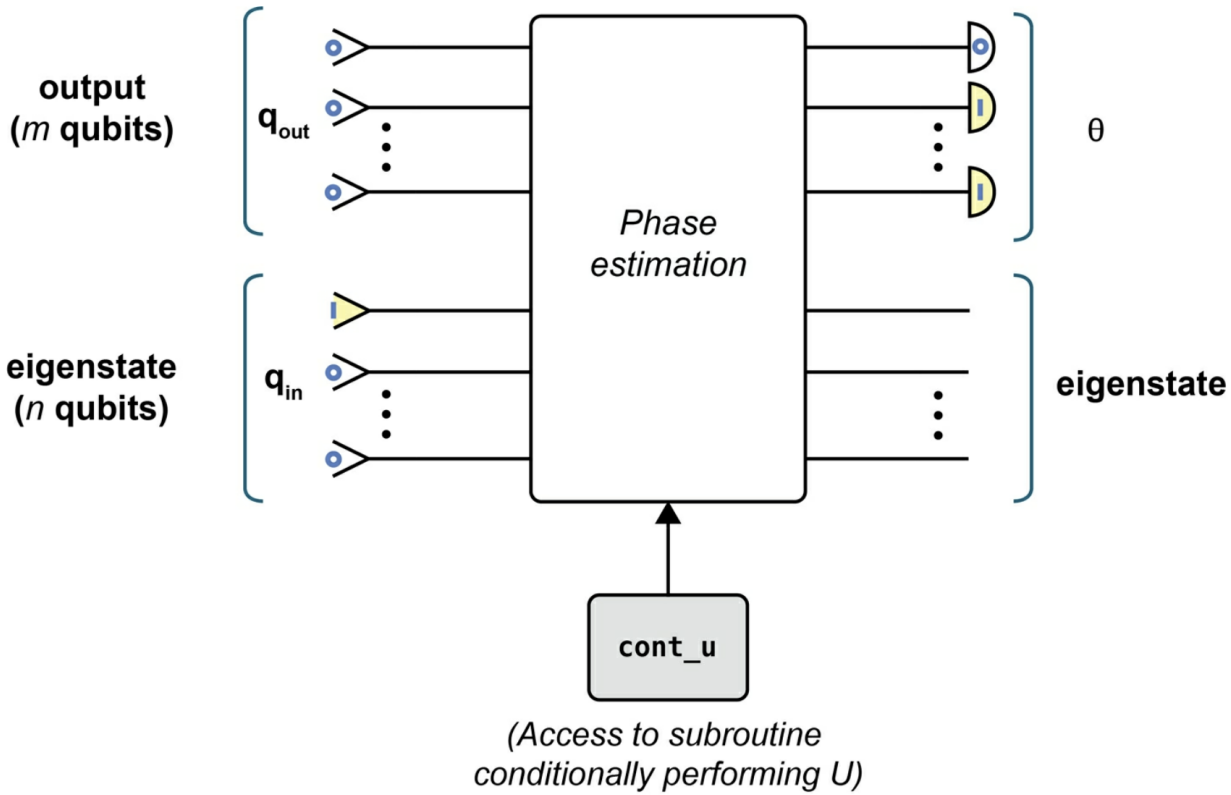


Attention, on ne parle pas de phase relative entre états, mais de phase absolue (changement de phase de l'ensemble des états).

Caractérisation d'un opérateur quantique par ses états/phases propres :

It's worth reiterating that the particular eigenstates (and associated eigenphases) shown in Table 8-1 are specific to HAD — other QPU operations will have entirely different eigenstates and eigenphases. In fact, the eigenstates and eigenphases of a QPU operation determine the operation entirely, in the sense that no other QPU operation has the same set.

On peut à l'aide de ce circuit calculer la phase absolue :



On rentrera une "output" initialisée à 0 (notez l'abus de langage), qui sera marquée par le circuit (auquel on donne l'état propre) par la phase propre. L'état propre sortira intact mais l'output sera valuée par ce qu'on espérait, d'où le nom.

Prenons un exemple :

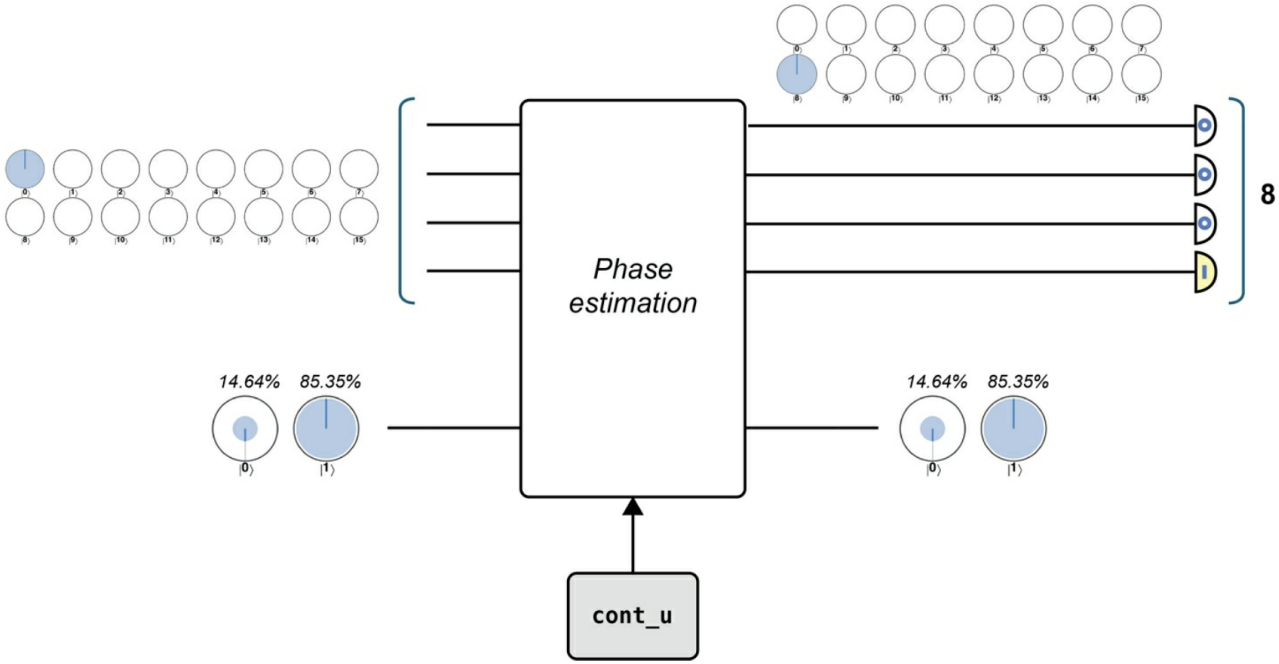


Figure 8-4. Overview of how to use phase estimation primitive

The state of the input register remains the same before and after we apply `phase_est()`, as expected. But what's going on with the output register? We were expecting 180° and we got 8!

On dénote que l'état $|0\rangle$ ne peut être transformé en sortie qu'en un autre état (ou une superposition d'états), codé de façon relative sur les 16 états possibles. Que représente donc ce 8 ?

Solution : il correspond en fait justement à 180 degrés (on pense en relatif où $2^4 = 16$ correspond à 360 degrés).

Autrement dit :

$$R = \frac{\theta_j}{360^\circ} \times 2^m$$

Cela donne dans le cas de 3 qubits :

Binary	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Fraction of register	0	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{7}{8}$
Angle	0	45	90	135	180	225	270	315

Note : il faut penser sens trigonométrique, ainsi 90 degrés et 270 degrés ne sont pas équivalents.

Complexity

The complexity of the phase estimation primitive (in terms of number of operations needed) depends on the number of qubits, m, that we use in our output register, and is given by $O(m^2)$. Clearly, the more precision we require, the more QPU operations are needed. We'll see in "Inside the QPU" that this dependence is primarily due to phase estimation's reliance on the invQFT primitive.