PRACTICAS EMPRESARIAL

EMPRESA GRUPO BERNIER S.A.S

JUAN PABLO MUÑOZ VIDAL

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

PROGRAMA DE INGENIERÍA MECATRÓNICA

BUCARAMANGA

2019

PRACTICAS EMPRESARIAL

EMPRESA GRUPO BERNIER S.A.S

JUAN PABLO MUÑOZ VIDAL

INFORME FINAL PRACTICAS ACADÉMICAS PARA OBTENER EL TÍTULO DE INGENIERO MECATRÓNICO

DIRECTOR:

M. Sc. HERNANDO GONZÁLEZ ACEVEDO

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

PROGRAMA DE INGENIERÍA MECATRÓNICA

BUCARAMANGA

2019

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

1. INTRODUCCIÓN

Una buena elaboración de las practicas académicas le permiten a el estudiante tener una idea mas clara a lo que se va a enfrentar en su vida laboral en el momento de obtener su título profesional. En este informe se presenta las actividades elaboradas en el trascurso del desarrollo de las practicas académicas en la empresa Grupo Bernier S.A.S.

Grupo Bernier S.A.S, tiene como objetivo cubrir las necesidades y dar respuesta a los retos en el área de la tecnología de las pequeñas y medianas industrias, en las que los dueños son actores operativos y no estratégicos, utilizando técnicas de tecnologías inteligentes e innovadoras como lo son los chatbots, análisis predictivo de datos, inteligencia artificial, market eye y computación cuántica.

En el desarrollo de las practicas al estudiante se le asignaron cursos de computación cuántica para obtener un dominio del tema y se le dio la tarea de desarrollar algoritmos en el área de computación cuántica. Las practicas tuvieron un tiempo de desarrollo de 4 meses con un horario de medio tiempo 5 días a la semana.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

- Desarrollar un código de encriptación de datos basado en computación cuántica a un estado de producto mínimo viable o piloto en dos tecnologías de computación cuántica.

3. COMPUTACIÓN CUÁNTICA

3.1. ORDENADOR CUÁNTICO

Si comparamos un ordenador clásico con un ordenador cuántico, encontraremos la primera diferencia en algo tan fundamental como la unidad de información. Como ya sabemos, un ordenador maneja la información en forma de bits, que pueden tomar dos valores, 0 y 1, tomando únicamente uno de estos valores en un momento dado. La información se codifica en cadenas con varios de estos bits, que sería con lo que trabajen los ordenadores. Para representar estas cadenas los ordenadores disponen de varios dispositivos físicos que solo se pueden encontrar en dos estados físicos no ambiguos y perfectamente distinguibles que se corresponderán con los estados 0 y 1. Estos sistemas físicos pueden ser, por ejemplo, interruptores abiertos (0) o cerrados (1).

La unidad de información con la que se trabaja en computación cuántica es análoga al bit, pero se diferencia en que, en lugar de ser un sistema con dos posibles estados, es un sistema de dos niveles. Por ejemplo, tomemos como base el modelo atómico y supongamos que tenemos un átomo con un electrón que, a su vez, se puede hallar en dos estados de energía distintos. Sabemos que, si le aplicamos luz con la energía adecuada y durante el periodo de tiempo apropiado, podemos hacer que el electrón cambie de orbita y pase, por ejemplo, del estado de reposo al estado excitado y viceversa. De la misma manera, pero reduciendo el tiempo de aplicación de la luz, podemos conseguir que el electrón se quede en un estado intermedio entre estos dos estados de energía [1], en ambos niveles al mismo tiempo. Si llamamos a estos dos posibles estados 0 y 1 o, como es más común al hablar de computación cuántica [2], $|0\rangle$ y $|1\rangle$, la unidad de información podrá estar en el estado $|0\rangle$, en $|1\rangle$ o bien en un estado de superposición cuántica, es decir, un estado formado por la combinación lineal de ambos, de la forma de la ecuación 1:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad (1)$$

siendo α y β números complejos. Dicho de otra manera, el estado de esta unidad de información es un vector en un espacio vectorial bidimensional complejo. A esta unidad de información se le llama bit cuántico, qbit o qubit, que es la denominación más extendida. Para denotar el estado de un sistema cuántico se emplea la notación de Dirac. Esta notación emplea la estructura $|\rangle$, llamada ket. Cuando se utiliza esta notación, los estados vienen representados como $|0\rangle$, $|1\rangle$, $|2\rangle$ ... ... $|k\rangle$, donde $k \leq 2^n - 1$, siendo n el número de qubits del sistema. En ocasiones se emplea la expansión binaria de n dígitos. Por ejemplo, si empleamos la expansión binaria de $n = 3$ dígitos, el estado $|1\rangle$ también se puede representar utilizando la notación de Dirac como $|1\rangle_3$ o como $|001\rangle$.Por último, en ocasiones se emplearán letras para representar los qubits de forma simbólica. Así, los estados $|x\rangle$ e $|y\rangle$ representados de nuevo mediante notación de Dirac pueden representar, por ejemplo, el estado del qubit antes y después de que se le aplique una operación cuántica. Si bien la notación de Dirac es una de las maneras más comunes de representar qubits,

también se puede trabajar con ellos en forma de vectores columna. Los estados $|0\rangle$ y $|1\rangle$ se representan como vector (ecuación 2):

$$|0\rangle \leftrightarrow \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle \leftrightarrow \begin{bmatrix}0\\1\end{bmatrix} \quad (2)$$

Lo que implica que el estado general de un qubit (ecuación 1) se puede expresar mediante el siguiente vector (ecuación 3):

$$\begin{bmatrix}\alpha\\\beta\end{bmatrix} \quad (3)$$

Para finalizar con las características de los qubits, vamos a comentar una particularidad de los mismos que no encuentra analogía en los bits clásicos. Mientras que en la computación clásica podemos examinar el bit para determinar si este se encuentra en el estado 0 o 1, no podemos realizar la misma operación para averiguar en qué estado cuántico se encuentran los qubits, es decir, no podemos calcular el valor de α y β al mismo tiempo [3].
La operación que se emplea para examinar el qubit se denomina medida. Aunque más adelante hablaremos de la medida con más detalle, por ahora nos basta saber que si medimos el estado (ecuación 3) en la base $(|0\rangle, |1\rangle)$, obtendremos o bien 0 con probabilidad $|\alpha|^2$, o bien 1 con probabilidad $|\beta|^2$, con $|\alpha|^2 + |\beta|^2 = 1$.

3.2. EL QUBIT Y LA ESFERA DE BLOCH [4]

El conjunto de todos los estados posibles de un qubit corresponde a la superficie de una esfera de radio unidad en el espacio tridimensional $\mathbb{R}^3$. Esta esfera se conoce como esfera de Bloch, en honor al físico Felix Bloch, y nos facilita la visualización de los estados de un qubit.
Partiendo de que α y β son números complejos, pueden ser expresados usando la notación de Euler: $\alpha = r_\alpha e^{i\varphi\alpha}|0\rangle, \beta = r_\beta e^{i\varphi\beta}|1\rangle$ entonces la ecuación 1 resulta como se muestra en la ecuación 4.

$$|\psi\rangle = r_\alpha e^{i\varphi\alpha}|0\rangle + r_\beta e^{i\varphi\beta}|1\rangle \quad (4)$$

Dado que, a efectos observables, las probabilidades de medir un cierto estado son $|\alpha|^2$ y $|\beta|^2$, $|\psi\rangle$ se puede multiplicar por un factor cualquiera $e^{i\gamma}$ (fase global), ya que estas probabilidades no van a cambiar.
Demostración:

$$|\alpha e^{i\gamma}|^2 = (\alpha e^{i\gamma}) * (\alpha e^{i\gamma}) = \alpha * e^{-i\gamma}\alpha e^{i\gamma} = \alpha * \alpha = |\alpha|^2 \quad (5)$$

Entonces, si multiplicamos la ecuación 4 por $e^{-i\varphi\alpha}$no afectaría al resultado dando como resultado la ecuación 6:

$$|\psi\rangle = r_\alpha|0\rangle + r_\beta e^{i\varphi}|1\rangle \quad (6)$$

Donde $\varphi = \varphi_\beta - \varphi_\alpha$. Hemos pasado de tener cuatro parámetros a tres. Ahora podemos expresar $r_\beta e^{i\varphi}$ en coordenadas cartesiana (ecuación 7):

$$|\psi\rangle = r_\alpha|0\rangle + (x + iy)|1\rangle \quad (7)$$

Dado que $|\psi\rangle$ tiene que cumplir la condición de normalización (ecuación 8) $\langle\psi|\psi\rangle = 1$,

$$r\alpha 2{+}x{+}iy2{=}1 \rightarrow r\alpha 2{+}x2{+}y2{=}1 \quad (8)$$

La ecuación 8 es una esfera de radio unidad en el espacio tridimensional $\mathbb{R}^3$, lo que sugiere una representación en coordenadas esféricas. Teniendo en cuenta que el cambio de coordenadas cartesianas a esféricas es (ecuación 9):

$$x = \sin(\theta)\cos(\varphi)\,, y = \sin(\theta)\sin(\varphi)\,, Z = \cos(\theta) \quad (9)$$

Por lo tanto, obtenemos que (ecuación 10):

$$|\psi\rangle = \cos(\theta)\,|0\rangle + [\sin(\theta) * \cos(\varphi) + i * sin(\theta) * \sin(\varphi)]|1\rangle = \cos(\theta)\,|0\rangle + \sin(\theta)e^{i\varphi}|1\rangle. \quad (10)$$

Observamos que para $\theta = 0$ el estado $|\psi\rangle$ es el $|0\rangle$, y para $\theta = \frac{\pi}{2}$ el estado es $e^{i\varphi}|1\rangle$, lo que hace que sólo nos haga falta los valores $0 \leq \theta \leq \frac{\pi}{2}$ para obtener todos los estados posibles de superposición. Es más, si tomamos un estado $|\psi'\rangle$ en el lado opuesto de la esfera con coordenadas $(1;\ \pi - \theta;\ \varphi + \pi)$, tenemos que $|\psi'\rangle = -|\psi\rangle$.

Demostración.

$$|\psi'\rangle = \cos(\pi - \theta)\,|0\rangle + \sin(\pi - \theta)\,e^{i(\varphi+\pi)}|1\rangle = -\cos(\theta)\,|0\rangle + \sin(\theta)\,e^{i(\varphi+\pi)}|1\rangle = -\cos(\theta)\,|0\rangle - \sin(\theta)\,e^{i\varphi}|1\rangle = -|\psi\rangle \quad (11)$$

Por tanto, observamos que las semiesferas superior e inferior son iguales, pero con signo opuesto. Es por esto por lo que reemplazamos la ecuación 10 por (ecuación 12):

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)e^{i\varphi}|1\rangle \quad (12)$$

o lo que es lo mismo (ecuación 13),

$$|\psi\rangle = \left(\cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)e^{i\varphi}\right) \quad (13)$$

con $\theta\epsilon[0,\pi]$ y $\varphi\epsilon[0,2\pi]$. Si ahora calculamos un estado ortogonal a este, sabiendo que $\langle\psi|\psi_\perp\rangle = 0$ (ecuación 14),

$$|\psi_\perp\rangle = \left(-\sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)e^{i\varphi}\right) \quad (14)$$

obtenemos un par de vectores ortonormales, conocidos como *estados de Bloch*. Como veremos más adelante, los estados $|\psi\rangle$ y $|\psi_\perp\rangle$ corresponden, respectivamente, con los autoestados asociados a los autovalores $+1/2$ y $-1/2$ de la proyección S_n del operador de espín. La figura 1 muestra una representación gráfica de la esfera de Bloch de un qubit.

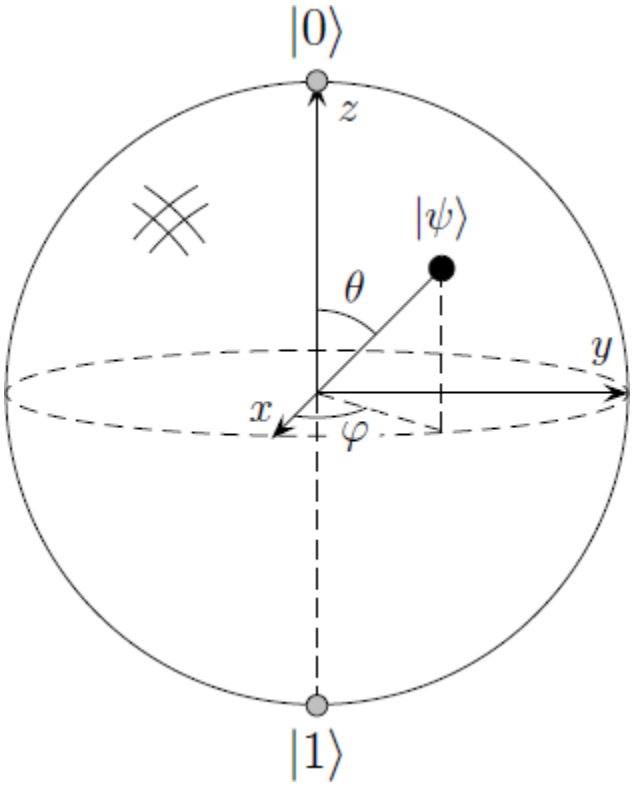

Figura 1. Representación gráfica1de la esfera de Bloch de un qubit [4].

3.3. ENTRELAZAMIENTO CUÁNTICO

Una propiedad responsable de la potencia de los métodos de cálculo cuánticos es el entrelazamiento. El entrelazamiento es un recurso aprovechado en los métodos de factorización, aunque también es responsable de la decoherencia. Una característica tan importante como esta desearíamos ser capaces de cuantificarla. El criterio que utilizaremos es el de Bennett y Shor [5].
Partamos de que α y β comparten un sistema cuántico (cada uno accede a una parte). Sabemos que hay entrelazamiento entre las dos partes cuando el estado del sistema no se puede representar por una superposición de productos tensoriales. Formalmente, hay entrelazamiento cuando la matriz densidad del sistema no se puede escribir como (ecuación 15):

$$\rho^{\alpha\beta} = \sum iPi\rho_i^{\alpha} \otimes \rho_i^{\beta} \quad (15)$$

Donde los estados de cada subsistema no interferirían entre sí. Tanto α como β pueden preparar cualquier estado no entrelazado por medio de transformaciones locales y de comunicación clásica: primero escogiendo el índice 1, para el que deciden probabilidad p_i, después se preparando los estados locales $p_i^{\alpha}\ y\ p_i^{\beta}$. A nivel cuantitativo, la definición de entrelazamiento de formación involucra el número de pares necesarios para crear copias de un cierto estado con alta fidelidad, y la del entrelazamiento destilable, el número de pares casi perfectos que se pueden obtener con elevada fiabilidad de las copias del estado [6].

3.4. COMPUERTAS CUÁNTICAS

Así como una computadora clásica utiliza circuitos lógicos que permiten la manipulación de información mediante compuertas, en el cómputo cuántico existen mecanismos para manipular quregistros. El análogo a los circuitos lógicos son las compuertas cuánticas o bien qucompuertas. Las compuertas cuánticas y las composiciones de ellas transforman linealmente un estado inicial ψ_1 en un estado final ψ_2, conservando siempre las normas de los vectores, es decir, su ortogonalidad. Cualquier transformación lineal en un espacio vectorial complejo puede ser descrita por una matriz. Sea A^* la conjugada transpuesta de la matriz A. Una matriz M es unitaria si se cumple que $AA^* = I$. Las compuertas son transformaciones unitarias en espacios de Hilbert [7]. Dado este hecho, una de las consecuencias más importantes de las compuertas cuánticas es que son reversibles.

Para entender mejor el concepto de compuerta cuántica, veremos algunos ejemplos muy sencillos que operan sobre qubits. Estas transformaciones tienen un nombre convencional. I es la transformación identidad, X es la negación, $Y = ZX$ es la combinación de Z y de X, y Z es un corrimiento de fase. Cualquiera de éstas cumple la propiedad de ser transformaciones unitarias. Otra compuerta muy común es la compuerta C_{not} (Controlled-NOT gate), la cual aplica la compuerta de negación al segundo qubit siempre que el primero se encuentre en estado $|1\rangle$. En la literatura el primer qubit es conocido como bit de control y el segundo como un qubit de salida o bien target. La transformación C_{not} es unitaria, ya que se cumple que $C_{not}^{\ *} = C_{not}$ y además $C_{not} * C_{not} = I$. Esta compuerta no puede ser descompuesta en el producto tensorial de dos transformaciones a bits.
Otra transformación importante que opera sobre un bit es la compuerta de Hadamard esta tiene importantes aplicaciones. Cuando ésta es aplicada a $|0\rangle$, H crea una superposición del estado $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Aplicada a n bits genera una superposición de 2^n posibles estados [8]. A continuación, mostramos las matrices

de las compuertas cuánticas, las cuales pueden generar cualquier transformación lineal de dimensión 2, a partir de una combinación lineal de ellas con coeficientes complejos, en la primera columna aparece cada matriz y en la segunda la transformación lineal correspondiente:

Tabla 1. Matrices de Pauli

Matriz		Transformación lineal
$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	I:	$\|0\rangle \to \|0\rangle$
		$\|1\rangle \to \|1\rangle$
$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	X:	$\|0\rangle \to \|1\rangle$
		$\|1\rangle \to \|0\rangle$
$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Z:	$\|0\rangle \to \|0\rangle$
		$\|1\rangle \to -\|1\rangle$
$Y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	Y:	$\|0\rangle \to -\|1\rangle$
		$\|1\rangle \to \|0\rangle$
$C_{not} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	C_{not}:	$\|0x\rangle \to \|0\rangle \otimes \|x\rangle$
		$\|1x\rangle \to \|0\rangle \otimes X\|x\rangle$
$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	H:	$\|0\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$
		$\|1\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$

Por otro lado, en la computación clásica se puede realizar la copia de información sin ningún problema, sin embargo, en la computación cuántica la accesibilidad a la información tiene una limitante. De acuerdo al Teorema 1, la mecánica cuántica no permite la copia exacta de estados cuánticos desconocidos, lo que implica tener ciertas limitaciones en la aproximación de copias.

Teorema 1 (No clonación) *De acuerdo a las propiedades unitarias de la mecánica cuántica no es posible hacer una copia o clonación de un estado cuántico.*

Probaremos un ejemplo muy sencillo utilizando la linealidad de las transformaciones unitarias.
Supongamos que U es una transformación unitaria cuya función es clonar estados, es decir, $U(|a0\rangle) = |aa\rangle$ cualquiera que sea $|a\rangle$. Si en particular se considera un par de estados cuánticos ortogonales, digamos$|a\rangle\, y\, |b\rangle$, para el estado $|c\rangle = \frac{1}{\sqrt{2}}(|a\rangle + |b\rangle)$ resulta, por un lado la ecuación 16.

$$U(|c\rangle) = U\left(\frac{1}{\sqrt{2}}(|a0\rangle + |b0\rangle)\right) = \frac{1}{\sqrt{2}}(U(|a0\rangle) + U(|b0\rangle)) = \frac{1}{\sqrt{2}}(|aa\rangle + |bb\rangle) \quad (16)$$

.

y, por otro lado, siendo U de clonación la ecuación 17.

$$U(|c\rangle) = |cc\rangle = (|a\rangle + |b\rangle) \otimes (|a\rangle + |b\rangle) = \frac{1}{\sqrt{2}}(|aa\rangle + |bb\rangle) \quad (17)$$

lo que implica $\frac{1}{\sqrt{2}}(|ab\rangle + |ba\rangle) = 0$, lo cual no es posible. Esto prueba que es imposible clonar perfectamente un estado cuántico desconocido usando transformaciones unitarias. Además, la clonación no es posible mediante la toma de mediciones, ya que ésta última es probabilista y destruye estados.

Otra operación a tener en cuenta es la de medida, esta codifica un estado cuántico $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ usando un bit de información clásica M, obteniendo un resultado de 0 o 1 con una probabilidad de α^2 o β^2 respectivamente. Justo después de esta operación la función de onda colapsa hacia $|0\rangle$ si M es 0, o hacia $|1\rangle$ si es 1.

3.5. ALGORITMOS CUÁNTICOS

Un algoritmo, en general, es una sucesión finita de pasos para resolver una tarea. Un algoritmo cuántico consiste, así mismo, en la ejecución de una serie de compuertas cuánticas aplicado a entidades, que pueden ser qubits o quregistros, seguido de una posterior toma de mediciones para recuperar los estados resultantes. La figura 2 muestra un diagrama de un algoritmo cuántico básico [9]. Luego de una inicialización de la máquina de estados y de los qubits y quregistros, se aplican las transformaciones unitarias indicadas. Dado que la toma de mediciones es probabilista se ha de disponer de un generador de números aleatorios. Dependiendo de la toma de mediciones se decide si acaso se ha encontrado una solución, o bien, se reinicia el ciclo. En el rectángulo derecho aparecen concentradas las diferentes operaciones cuánticas y en el lado izquierdo una estructura típica de control (estructura selectiva).

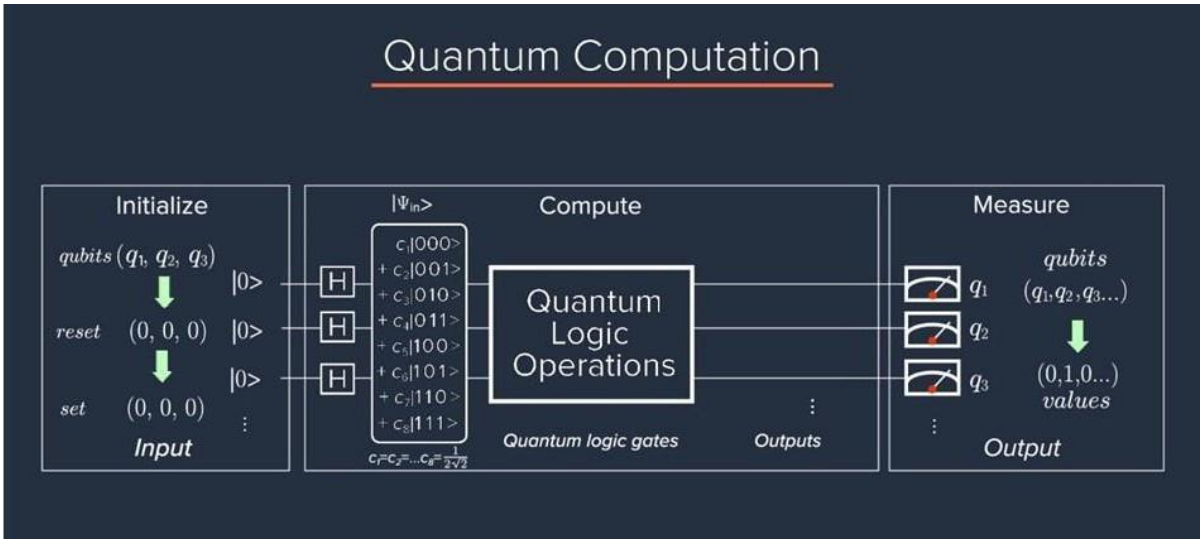

Figura 2. Diagrama de un algoritmo cuántico básico. [10]

4. DESARROLLO DE ACTIVIDADES

En el transcurso de las practicas se desarrollaron códigos en tecnologías de computación cuántica como un sumador, restador y generador de números pares, impares y primos. Estos códigos fueron generados en IBM quiskit y Q#, pero por cuestiones de privacidad de la información producida durante este tiempo en la empresa solo se hablará en este capítulo un poco del sumador generado y la metodología para programar en IBM quiskit y Q#.

4.1. CREAR EL CIRCUITO CON COMPUERTAS CUÁNTICAS CON IBM QUISKIT EN PYTHON.

Primero iniciamos importando la librería de quiskit

```
from qiskit import *
```

El paso a seguir es crear un registro cuántico con la cantidad de n qubits que vayamos a utiliza, en otras palabras viene siendo es seleccionar los qubits que vamos a utilizar y poniéndolos en estado que conocemos |0⟩.

- `q = QuantumRegister(n)`

Lo siguiente es crear un registro clásico que es donde se va a guardar las mediciones de los n qubits que será de m cantidad de mediciones.

- `c = ClassicalRegister(m)`

Ahora se crea un circuito cuántico que es donde se montaran las compuertas que utilizaremos en los qubits y le pasamos el registro cuántico y el registro clásico.

- `qc = QuantumCircuit(q, c)`

Una vez ya creado el circuito cuántico ahora si pasamos a posicionar las compuertas cuánticas para obtener el resultado esperado. Las compuertas cuánticas en este leguaje se escriben con qc. seguido de la letra de la compuerta a utilizar (x,h,z,r,y,i) si la compuerta va controlar el estado de otro qubits se utiliza una c antes de poner la letra de la compuerta, tenga en cuenta que solo se pueden usar como compuertas controladas la compuerta h y x. a continuación se presenta un ejemplo de como se escribiera los dos casos:

- `qc.x(q[n])`
- `qc.ch(q[n], q[n+1])`

Ya tenemos el circuito que queremos ahora lo siguiente es medir todos los qubits usando el comando measure sobre el circuito, se le pasa el qubit a medir y el bit clásico donde queremos que nos guarde la medición.

- `qc.measure(q[n], c[m])`

Cargamos el simulador que vamos a utilizar para correr el circuito, en este caso utilizaremos qasm_simulato.

- backend_sim = BasicAer.get_backend('qasm_simulator')

Y Ejecutamos el circuito cuántico con el comando execute pasándole el circuito cuantico y el simulador que cargamos anteriormente y guardamos los resultados en result.

- result = execute(qc, backend_sim).result()

Ahora procedemos a imprimir la salida con el comando print lo que nos devuelve una cadena en forma de json.

- print(result.get_counts(qc))

Y por último si queremos dibujamos el circuito cuántico con el comando draw() sobre el circuito.

- qc.draw()

Siguiendo estos pasos usted tendrá un código base para general su código cuántico, si quiere obtener más información para general un algoritmo cuántico en IBM Quiskit puedes revisar [11].

4.2. CREAR EL CIRCUITO CON COMPUERTAS CUÁNTICAS CON Q#.

Esta guía está hecha para utilizar visual studio como compilador y el primer paso es descargar el kit de desarrollo de microsoft quantum este lo pueden encontrar fácilmente en la página de Microsoft y es total mente gratis. Una vez instalado el kit de desarrollo de microsoft quantum abrimos visual studio y creamos un nuevo proyecto con Q# Application.

En Q# trabajamos con dos archivos el Driver.cs que contendrán el controlador C # para su código cuántico, y Operations.qs, que contendrá el código cuántico en sí.

Q# entrega el siguiente código base del hola mundo:

```
namespace Quantum.Bell {
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;

    operation HelloQ () : Unit {
        Message( "Hello quantum world!" );
    }
```

}

Primero vamos a editar la operación para crear una operación la cual nos permita inicializar los qubits. Primero, reemplace la cadena HelloQcon Sety cambie los parámetros de la operación (el contenido de los paréntesis) para que contengan la cadena desired: Result, q1: Qubit. El archivo ahora debería verse como:

```
operation Set (desired: Result, q1: Qubit) : Unit {

    Message( "Hello quantum world!" );

}
```

Ahora, reemplace Message("Hello quantum world!");con el siguiente código entre las llaves que encierran el cuerpo de la operación:

```
if (desired != M(q1)) {
        X(q1);
    }
```

El archivo ahora debería verse como:

```
operation Set (desired: Result, q1: Qubit) : Unit {
    if (desired != M(q1)) {
        X(q1);
    }
```

Ahora se puede llamar a esta operación para establecer un qubit en un estado conocido (Zero o One). Medimos el qubit, si está en el estado que queremos, lo dejamos solo, de lo contrario, lo giramos con la compuerta X. Una operación es la unidad básica de ejecución cuántica en Q #. Los argumentos para una operación se especifican como una tupla, entre paréntesis. El tipo de retorno de la operación se especifica después de dos puntos. En este caso, la Set operación no tiene retorno, por lo que se marca como retorno Unit. Una operación tiene una sección de cuerpo, que contiene la implementación de la operación. Estos se utilizan para especificar variantes específicas de operaciones apropiadas. Vea la referencia de lenguaje Q # para más información.

Ahora creamos una nueva operación la cual nos va permitir ejecutar las compuertas cuánticas de nuestro algoritmo cuántico, esta es creada usando el siguiente código:

```
operation BellTest (count : Int, initial: Result) : (Int, Int) {

    mutable numOnes = 0;
    using (qubit = Qubit()) {

        for (test in 1..count) {
            Set (initial, qubit);

            let res = M (qubit)
```

```
            if (res == One) {
                set numOnes += 1;
            }
        }

        Set(Zero, qubit);
    }

// Devuelve el número de veces que vimos un |0> y el número de veces que vimos un |1>
    return (count-numOnes, numOnes);
  }
```

En esta operación usted podrá agregar mas qubits y asignarles las compuertas que crea necesarias para su lógica cuántica. Esta operación (BellTest) realizará un bucle para las count iteraciones, establecerá un valor inicial especificado en un qubit y luego medirá (M) el resultado. Recopilará estadísticas sobre la cantidad de unos y ceros que hemos medido y los devolverá a la persona que llama. Realiza otra operación necesaria. Restablece el qubit a un estado conocido (Zero) antes de devolverlo, lo que permite a otros asignar este qubit en un estado conocido.

Esto es requerido por la using declaración. Todas estas llamadas utilizan operaciones cuánticas que se definen en el Microsoft.Quantum.intrinsic de nombres. Por ejemplo, la M operación mide su argumento qubit en la Z base computacional () y X aplica un giro de estado alrededor del eje x a su argumento qubit.

La using declaración también es especial para Q #. Se utiliza para asignar qubits para su uso en un bloque de código. En Q #, todos los qubits se asignan y liberan dinámicamente, en lugar de ser recursos fijos que están ahí durante toda la vida útil de un algoritmo complejo.

La BellTest operación devuelve dos valores. El tipo de retorno de la operación es (Int, Int), y la return declaración tiene una tupla explícita que contiene dos variables enteras, como una forma de pasar varios valores, en lugar de usar estructuras o registros.

Ahora pasamos al controlador de C# llamado Driver.cs archivo en su entorno de desarrollo. Este archivo debe tener los siguientes contenidos:

```
using System;

using  Microsoft.Quantum.Simulation.Core;
using Microsoft.Quantum.Simulation.Simulators;

namespace Quantum.Bell
```

```
{
    class Driver
    {
        static void Main(string[] args)
        {
            using (var qsim = new QuantumSimulator())
            {
                HelloQ.Run(qsim).Wait();
            }
        }
    }
}
```

Tenesmos que reemplace el cuerpo del Main para leer las los datos programados en el código cuántico anterior, usando el método del siguiente código:

```
  using (var qsim = new QuantumSimulator())
        {

            Result[] initials = new Result[] { Result.Zero, Result.One };
            foreach (Result initial in initials)
            {
                var res = BellTest.Run(qsim, 1000, initial).Result;
                var (numZeros, numOnes) = res;
                System.Console.WriteLine(
                    $"Init:{initial,-4} 0s={numZeros,-4} 1s={numOnes,-4}" );
            }
        }

        System.Console.WriteLine( "Press any key to continue..." );
        Console.ReadKey();
```

El controlador de C# del código anterior se divide en cuatro partes las cuales son:

- Construye un simulador cuántico. En el ejemplo, qsim es el simulador.
- Calcule los argumentos necesarios para el algoritmo cuántico. En el ejemplo, count se fija en 1000 y inicializa los qubit.
- Ejecutar el algoritmo cuántico. Cada operación Q # genera una clase C # con el mismo nombre. Esta clase tiene un run método que ejecuta de forma asíncrona la operación. La ejecución es asíncrona porque la ejecución en el hardware real será asíncrona. Debido a que el run es asíncrono, obtenemos la respuesta, esto bloquea la ejecución hasta que la tarea se completa y devuelve el resultado de forma sincrónica.

- Procesar el resultado de la operación. En el ejemplo, res recibe el resultado de la operación. Aquí el resultado es un grupo de número de ceros (numZeros) y el número de unos (numOnes) medidos por el simulador. Esto se devuelve como ValueTuple en C #, para obtener los dos campos, imprimir los resultados y esperar una pulsación de tecla.

Siguiendo estos pasos usted tendrá un código base para general su código cuántico, si quiere obtener más información para general un algoritmo cuántico en Q# puedes revisar [12].

4.3. SUMADOR DE 3 QUBITS

En la figura 3 se presenta la estructura equivalente al sumador de qubits generado este requiere de 12 qubits para su funcionamiento permitiendo sumar 3 qubits y dando una salida de 4 qubits. La lógica utilizada para la elaboración del sumador fue la misma implementada para realizar la suma de tres bits solo que en este caso lo miramos y programábamos de forma cuántica.

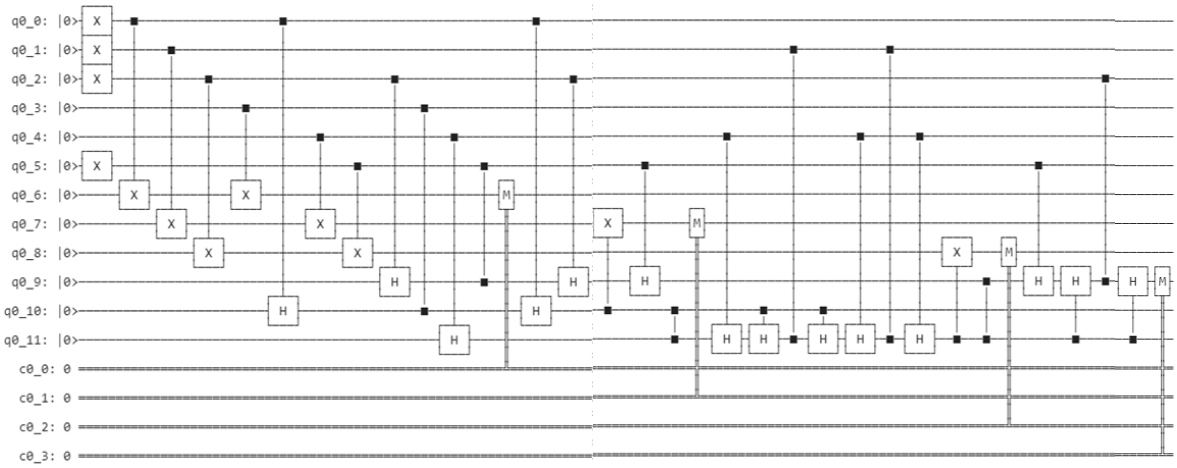

Figura 3. Lógica cuántica de un sumador de tres qubits [Autor]

para realizar la lógica del sumador de tres qubits tomamos como base la suma de tres bits el cual se representa por la operación de la ecuación 18,19,20 y 21.

$$Q_o = B_o \; XOR \; A_o \tag{18}$$

$$Q_1 = (B_1 \; XOR \; A_1) \; XOR \; (B_o \; AND \; A_o) \tag{19}$$

$$Q_2 = (B_2 \; XOR \; A_2) \; XOR \; ((B_1 \; AND \; A_1) \; OR \; (B_1 \; AND \; (B_0 \; AND \; A_0)) \; OR \; (A_1 \; AND \; (B_0 \; AND \; A_0))) \tag{20}$$

$$Q_3 = (B_2 \; AND \; A_2) \; OR \; ((B_2 \; AND \; (B_1 \; AND \; A_1)) OR \; ((B_1 \; AND \; A_1) \; AND \; A_2)) \tag{21}$$

Ahora lo que buscamos es escribir esta misma lógica, pero con compuertas cuánticas para el primer caso que es Q_o se puede representar con dos simples

compuertas como se presenta en la ecuación 22 y 23 lo cual sería igual a realizar $B_o\ XOR\ A_o$.

$$Controlada\ X(q[B_o], q[Q_o]) \quad (22)$$

$$Controlada\ X(q[A_o], q[Q_o]) \quad (23)$$

Para determinar la salida de Q_1 se tiene que llevar una combinación más amplia de las compuertas cuántica las cuales se presenta desde la ecuación 24 a la 29, donde toca usar un qubits (M_o) de más como memoria para un estado. Donde la ecuación 24,25 y 26 corresponden a realizar $(B_o\ AND\ A_o)$ y las ecuaciones 27,28 y 29 es realizar $(B_2\ XOR\ A_2)\ XOR\ (B_1\ AND\ A_1)$.

$$Controlada\ H(q[B_o], q[M_o]) \quad (24)$$

$$Controlada\ Z(q[A_o], q[M_o]) \quad (25)$$

$$Controlada\ H(q[B_o], q[M_o]) \quad (26)$$

$$Controlada\ X(q[B_1], q[Q_1]) \quad (27)$$

$$Controlada\ X(q[A_1], q[Q_1]) \quad (28)$$

$$Controlada\ X(q[M_o], q[Q_1]) \quad (29)$$

La salida Q_2 se puede representar con la siguiente combinación cuántica que va desde la ecuación 30 a la 41, donde vamos a utilizar un qubits (M_1) de más como memoria para un estado. Las ecuaciones 30 a la 38 corresponde a realizar $((B_1\ AND\ A_1)\ OR\ (B_1\ AND\ (B_0\ AND\ A_0))\ OR\ (A_1\ AND\ (B_0\ AND\ A_0)))$ y las ecuaciones 38, 39 y 40 es igual a el qubits de salida $XOR\ A_2)\ XOR\ ((B_1\ AND\ A_1)\ OR\ (B_1\ AND\ (B_0\ AND\ A_0))\ OR\ (A_1\ AND\ (B_0\ AND\ A_0)))$

$$Controlada\ H(q[A_1], q[M_1]) \quad (30)$$

$$Controlada\ Z(q[M_o], q[M_1]) \quad (31)$$

$$Controlada\ H(q[A_1], q[M_1]) \quad (32)$$

$$Controlada\ H(q[M_o], q[M_1]) \quad (33)$$

$$Controlada\ Z(q[B_1], q[M_1]) \quad (34)$$

$$Controlada\ H(q[M_o], q[M_1]) \quad (35)$$

$$Controlada\ H(q[A_1], q[M_1]) \quad (36)$$

$$Controlada\ Z(q[B_1], q[M_1]) \quad (37)$$

$$Controlada\ H(q[A_1], q[M_1]) \quad (38)$$

$$Controlada\ X(q[B_2], q[Q_2]) \quad (39)$$

$$Controlada\ X(q[A_2], q[Q_2]) \quad (40)$$

$$Controlada\ X(q[M_1], q[Q_2]) \quad (41)$$

Para la salida de Q_3 se representar con la siguiente combinación cuántica que va desde la ecuación 42 a la 50, estas ecuaciones representa a realizar $(B_2\ AND\ A_2)\ OR\ ((B_2\ AND\ (B_1\ AND\ A_1))OR\ ((B_1\ AND\ A_1)\ AND\ A_2))$.

$$Controlada\ H(q[B_2], q[Q_3]) \quad (42)$$

$$Controlada\ Z(q[A_2], q[Q_3]) \quad (43)$$

$$Controlada\ H(q[B_2], q[Q_3]) \quad (44)$$

$$Controlada\ H(q[A_2], q[Q_3]) \quad (45)$$

$$Controlada\ Z(q[M_1], q[Q_3]) \quad (46)$$

$$Controlada\ H(q[A_2], q[Q_3]) \quad (47)$$

$$Controlada\ H(q[M_1], q[Q_3]) \quad (48)$$

$$Controlada\ Z(q[B_2], q[Q_3]) \quad (49)$$

$$Controlada\ H(q[M_1], q[Q_3]) \quad (50)$$

4.3.1. SUMADOR EN IBM QUISKIT

```
#Importamos qiskit
 from qiskit import *
#Creamos un registro cuantico de 12 qubits
 q = QuantumRegister(12)
#Creamos un registro clasico de 4 bit
 c = ClassicalRegister(4)
# se crea un circuito que actue en el registro
 qc = QuantumCircuit(q, c)

# "Sumador"
# A1 = q0
# A2 = q1
# A3 = q2
# B1 = q3
# B2 = q4
# B3 = q5
# Q1 = q6
# Q2 = q7
# Q3 = q8
# Q4 = q9

# ac2 = q10
```

```
# ac3 = q11
# ac4 = q12

qc.h(q[0])
qc.h(q[1])
qc.h(q[2])
qc.h(q[3])
qc.h(q[4])
qc.h(q[5])
qc.cx(q[0], q[6])
qc.cx(q[3], q[6])

qc.ch(q[0], q[10])
qc.cz(q[3], q[10])
qc.ch(q[0], q[10])
qc.cx(q[1], q[7])
qc.cx(q[4], q[7])
qc.cx(q[10], q[7])
qc.ch(q[4], q[11])
qc.cz(q[10], q[11])
qc.ch(q[4], q[11])
qc.ch(q[10], q[11])
qc.cz(q[1], q[11])
qc.ch(q[10], q[11])
qc.ch(q[4], q[11])
qc.cz(q[1], q[11])
qc.ch(q[4], q[11])
qc.cx(q[2], q[8])
qc.cx(q[5], q[8])
qc.cx(q[11], q[8])
qc.ch(q[2], q[9])
qc.cz(q[5], q[9])
qc.ch(q[2], q[9])
qc.ch(q[5], q[9])
qc.cz(q[11], q[9])
qc.ch(q[5], q[9])
qc.ch(q[11], q[9])
qc.cz(q[2], q[9])
qc.ch(q[11], q[9])
qc.measure(q[6], c[0])
qc.measure(q[7], c[1])
qc.measure(q[8], c[2])
qc.measure(q[9], c[3])
backend_sim = BasicAer.get_backend('qasm_simulator')
```

```
result = execute(qc, backend_sim).result()
print(result.get_counts(qc))
```

4.3.2. SUMADOR EN Q#

```
namespace Quantum.Bell
{
open Microsoft.Quantum.Primitive;
open Microsoft.Quantum.Canon;
open Microsoft.Quantum.Extensions.Math;
open Microsoft.Quantum.Extensions.Convert;



    operation Set (desired: Result, q1: Qubit) : Unit
    {
        let current = M(q1);
        if (desired != current)
        {
            X(q1);
        }
    }
       operation BellTest (count : Int, initial: Result) : (Int, Int, Int, Int, Int,
Int, Int, Int)
    {
        mutable numOnes0 = 0;
              mutable numOnes1 = 0;
              mutable numOnes2 = 0;
              mutable numOnes3 = 0;
        using (q = Qubit[14])
        {
            for (test in 1..count)
            {
                Set (initial, q[0]);
                           Set (initial, q[1]);
                           Set (initial, q[2]);
                           Set (initial, q[3]);
                           Set (initial, q[4]);
                           Set (initial, q[5]);
                           Set (initial, q[6]);
                           Set (initial, q[7]);
                           Set (initial, q[8]);
                           Set (initial, q[9]);
                Set (initial, q[10]);
                           Set (initial, q[11]);


                // "Sumador"
                //# A1 = q0
                //# A2 = q1
                //# A3 = q2
                //# B1 = q3
                //# B2 = q4
                //# B3 = q5
                //# Q1 = q6
                //# Q2 = q7
```

```
//# Q3 = q8
//# Q4 = q9

//X(q[0]);
//X(q[1]);
//X(q[2]);
//X(q[3]);
//X(q[4]);
//X(q[5]);

Controlled X([q[0]], q[6]);
Controlled X([q[3]], q[6]);
Controlled H([q[0]], q[10]);
Controlled Z([q[3]], q[10]);
Controlled H([q[0]], q[10]);
Controlled X([q[1]], q[7]);
Controlled X([q[4]], q[7]);
Controlled X([q[10]], q[7]);
Controlled H([q[4]], q[11]);
Controlled Z([q[10]], q[11]);
Controlled H([q[4]], q[11]);
Controlled H([q[10]], q[11]);
Controlled Z([q[1]], q[11]);
Controlled H([q[10]], q[11]);
Controlled H([q[4]], q[11]);
Controlled Z([q[1]], q[11]);
Controlled H([q[4]], q[11]);
Controlled X([q[2]], q[8]);
Controlled X([q[5]], q[8]);
Controlled X([q[11]], q[8]);
Controlled H([q[2]], q[9]);
Controlled Z([q[5]], q[9]);
Controlled H([q[2]], q[9]);
Controlled H([q[5]], q[9]);
Controlled Z([q[11]], q[9]);
Controlled H([q[5]], q[9]);
Controlled H([q[11]], q[9]);
Controlled Z([q[2]], q[9]);
Controlled H([q[11]], q[9]);

let c0 = M (q[6]);
let c1 = M (q[7]);
let c2 = M (q[8]);
let c3 = M (q[9])
     if (c0 == Zero and  c1 == Zero and  c2 == Zero and  c3 == Zero)
{
    set p0000 = p0000 + 1;
}
if (c0 == One and  c1 == Zero and  c2 == Zero and  c3 == Zero)
{
    set p0001 = p0001 + 1;
}
     if (c0 == Zero and  c1 == One and  c2 == Zero and  c3 == Zero)
{
    set p0010 = p0010 + 1;
}
     if (c0 == One and  c1 == One and  c2 == Zero and  c3 == Zero)
{
```

```
        set p0011 = p0011 + 1;
    }
        if (c0 == Zero and  c1 == Zero and  c2 == One and  c3 == Zero)
    {
        set p0100 = p0100 + 1;
    }
        if (c0 == One and  c1 == Zero and  c2 == One and  c3 == Zero)
    {
        set p0101 = p0101 + 1;
    }
        if (c0 == Zero and  c1 == One and  c2 == One and  c3 == Zero)
    {
        set p0110 = p0110 + 1;
    }
        if (c0 == One and  c1 == One and  c2 == One and  c3 == Zero)
    {
        set p0111 = p0111 + 1;
    }
        if (c0 == Zero and  c1 == Zero and  c2 == Zero and  c3 == One)
    {
        set p1000 = p1000 + 1;
    }
        if (c0 == One and  c1 == Zero and  c2 == Zero and  c3 == One)
    {
        set p1001 = p1001 + 1;
    }
        if (c0 == Zero and  c1 == One and  c2 == Zero and  c3 == One)
    {
        set p1010 = p1010 + 1;
    }
        if (c0 == One and  c1 == One and  c2 == Zero and  c3 == One)
    {
        set p1011 = p1011 + 1;
    }
        if (c0 == Zero and  c1 == Zero and  c2 == One and  c3 == One)
    {
        set p1100 = p1100 + 1;
    }
        if (c0 == One and  c1 == Zero and  c2 == One and  c3 == One)
    {
        set p1101 = p1101 + 1;
    }
        if (c0 == Zero and  c1 == One and  c2 == One and  c3 == One)
    {
        set p1110 = p1110 + 1;
    }
        if (c0 == One and c1 == One and c2 == One and  c3 == One)
    {
        set p1111 = p1111 + 1;
    }
 }

Set(Zero, q[0]);
Set(Zero, q[1]);
Set(Zero, q[2]);
Set(Zero, q[3]);
Set(Zero, q[4]);
Set(Zero, q[5]);
```

```
            Set(Zero, q[6]);
            Set(Zero, q[7]);
            Set(Zero, q[8]);
            Set(Zero, q[9]);
            Set(Zero, q[10]);
            Set(Zero, q[11]);

        }

        // Return number of times we saw a |0> and number of times we saw a |1>
        return (p0000 ,p0001 ,p0010 ,p0011 ,p0100 ,p0101 ,p0110 ,p0111 ,p1000 ,p1001
,p1010 ,p1011 ,p1100 ,p1101 ,p1110 ,p1111 );
    }
}
```

4.3.2.1. DRIVER

```
using System;

using Microsoft.Quantum.Simulation.Core;

using Microsoft.Quantum.Simulation.Simulators;

namespace Quantum.Bell

{

    class Driver

    {

        static void Main(string[] args)

        {

            using (var qsim = new QuantumSimulator())

            {

                // Try initial values

                Result[] initials = new Result[] { Result.Zero };

                foreach (Result initial in initials)

                {

                    var res = BellTest.Run(qsim, 1023, initial).Result;

                    var (p0000, p0001, p0010, p0011, p0100, p0101, p0110, p0111,
p1000, p1001, p1010, p1011, p1100, p1101, p1110, p1111) = res;

                    System.Console.WriteLine(

                    $" 0000={p0000,-4}/1023 Numero: 0\n 0001={p0001,-4}/1023 Numero:
1\n 0010={p0010,-4}/1023 Numero: 2\n 0011={p0011,-4}/1023 Numero: 3\n 0100={p0100,-
4}/1023 Numero: 4\n 0101={p0101,-4}/1023 Numero: 5\n 0110={ p0110,-4}/1023 Numero:
6\n 0111={ p0111,-4}/1023 Numero: 7\n 1000={ p1000,-4}/1023 Numero: 8\n 1001={
p1001,-4}/1023 Numero: 9\n 1010={p1010,-4}/1023 Numero: 10\n 1011={ p1011,-4}/1023
Numero: 11\n 1100={p1100,-4}/1023 Numero: 12\n 1101={p1101,-4}/1023 Numero: 13\n
1110={  p1110,-4}/1023 Numero: 14\n 1111={ p1111,-4}/1023 Numero: 15\n");
```

```
                }
            }
            System.Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }
    }
}
```

5. CONCLUSIONES

- La computación cuántica es una tecnología innovadora, la cual puede brindar mucho potencial en el mundo de la tecnología informática. Actualmente se encuentra en una etapa de desarrollo, por lo tanto, los recursos relacionados a esta son limitados, haciendo que el objetivo propuesto no se cumpla a cabalidad ya que los recursos que se en encuentran no son suficientes e imposibilitan el cumplimiento de dicho objetivo.

- La costumbre de un trabajo en equipo, adquirida en el trascurso de mi desarrollo académico se ve aprovechada en un entorno laboral, consiguiendo el desarrollo de las actividades de una manera más eficiente.

- Los conocimientos adquiridos en el transcurso de mi desarrollo académico, fueron de gran importancia y utilidad, sobre todo a la hora de realizar una investigación exhaustiva para poder resolver cualquier dificultad o reforzar los conocimientos adquiridos.

6. BIBLIOGRAFÍA

[1] M. A. N. &. I. L. Chuang, Quantum Computation and Quantum Information, The Edinburgh Building, Cambridge: Cambridge University Press, 2010.

[2] N. D. Mermin, Quantum Computer Science, The Edinburgh Building: Cambridge University Press, 2007.

[3] L. Ruiz Perez, Operaciones Aritmeticas Basicas con Circuitos, Valladolid: Universidad de Valladolid, 2014.

[4] D. J. Benjumea Gayango, Elementos y conceptos de computación, Sevilla: Universidad de Sevilla, 2018.

[5] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin y H. Weinfurter, «Elementary gates for quantum computation,» *Physical Review A,,* vol. 52, pp. pp 3457-3467, 1995.

[6] N. Darwish Miranda, Computación cuántica, San Cristóbal de La Laguna: Universidad de La Laguna.

[7] N. Kumar, B. R. Childers y M. L. Soffa, Proceedings of the sixth international symposium on Automated analysis-driven debugging,, New York USA: ACM Press, 2005.

[8] M. Paredes López, Simulación de Cómputo Cuántico, Mexico: Centro De Investigacion y De Estudio Avanzado Del Instituto Politecnico Nacional Departamento De Computacion , 2007.

[9] B. Ömer, Structured Quantum Programming, Institute for Theoretical Physics, Vienna University of Technology, 2009.

[10] M. I. o. Technology, *Introduction to Quantum Computing,* Copyright, 2018.

[11] Qiskit, «Qiskit Tutorials,» Jupyter, [En línea]. Available: https://nbviewer.jupyter.org/github/Qiskit/qiskit-tutorials/blob/master/qiskit/start_here.ipynb. [Último acceso: 01 06 2019].

[12] Microsoft, «Escribiendo un programa cuántico,» Microsoft Docs, 12 12 2017. [En línea]. Available: https://docs.microsoft.com/es-es/quantum/quickstart?view=qsharp-preview&tabs=tabid-vs2017. [Último acceso: 01 06 2019].