ISSN 1870-4069

Circuito Cuántico Shor Qiskit

Jesús Álvarez Cedillo[1], Raúl Junior Sandoval[1], Teodoro Álvarez Sánchez[2], Adriana Nava Vega [3], Jacobo Sandoval Gutiérrez[4]

[1] Instituto Politécnico Nacional, UPIICSA, Ciudad de México, México
jalvarez@ipn.mx, rsandova@ipn.mx

[2] Instituto Politécnico Nacional, CITEDI, Baja California, México
talvarez@citedi.mx

[3] Universidad Autónoma de Baja California, Campus Tijuana, México
adriana.nava@uabc.edu.mx

[4] Universidad Autónoma Metropolitana, Unidad LERMA, Estado de México, México
jacobosandovalg@gmail.com

Resumen. La computación cuántica es un campo de investigación dinámico y apasionante que reúne el conocimiento de diversas disciplinas, incluida la física cuántica, la informática y la teoría clásica de la complejidad. Durante la década de 1980, el trabajo de David Deutsch ayudó a definir la teoría de la computación cuántica, actualmente IBM cuenta con una computadora cuántica real. En este articulo se utilizó la plataformas de IBM QISK para realizar la implementación desde cero del algoritmo de Shor. Este algoritmo permite la factorización primaria, se tomó como referencia este algoritmo debido a que ha sido muy criticado en la literatura por mostrar la supuesta intratabilidad de la factorización y la aplicabilidad universal de la teoría de la complejidad clásica, además que poner en duda la seguridad no reversible de los métodos de cifrado actuales.

Palabras clave: computación cuántica, algoritmo de Shor, circuito cuántico, lenguaje ensamblador cuántico.

Shor Qiskit Quantum Circuit

Abstract. Quantum computing is a dynamic and exciting field of research that brings together the knowledge of various disciplines, including quantum physics, computer science and the classical theory of complexity. During the 1980s, David Deutsch's work helped define the theory of quantum computing. Currently, IBM has a real quantum computer. In this article, the IBM QISK platforms were used to implement the Shor algorithm from scratch. This algorithm allows primary factorization. This algorithm was taken as a reference, because it has been highly criticized in the literature for showing the supposed intractability of the factorization and the universal applicability of the theory of classical complexity, in addition to questioning the non-reversible safety of current encryption methods.

Keywords: quantum computing, Shor algorithm, quantum circuit, quantum assembly language.

1. Introducción

La unidad mínima de información de las computadoras clásicas son los bits, estos sólo puede tener un estado en un momento determinado en el tiempo. Si hay varios valores diferentes para ser examinados, cada uno debe procesarse por separado. A diferencia de este tipo de computadoras, las computadoras cuánticas codifican la información usando dígitos binarios cuánticos llamados Qubits. La información está representada por su estado cuántico, y su giro o SPIN [1]. Los Qubits pueden tener de acuerdo al principio de superposición el valor de uno o cero, o ambos.

Como resultado del control de todos los estados al mismo tiempo, las computadoras cuánticas rompen con la no computabilidad y reducen la complejidad computacional de algunos algoritmos, el paralelismo inherente a las computadoras cuánticas las hace muy potentes para procesar información. Por desgracia, aún existen varios obstáculos importantes que los investigadores buscan vencer para convertirlas en una una de su homólogos clásicos [2].

Dos principios de la mecánica cuántica desempeñan un papel importante en el funcionamiento de una computadora cuántica, superposición y entrelazamiento [4].

La superposición es un fenómeno mecánico cuántico que permite a los electrones para evitar que se limita a un solo estado. Por lo que pueden tener alguna combinación de los dos, con diferentes probabilidades para cada estado.

Los Qubits son capaces de mantener su condición de superposición hasta que se realiza una medición del sistema, en cuyo punto el sistema se colapsa y el electrón se limita a un estado o el otro y se dice que se pierde la estabilidad.

Este es el principio de superposición lo que hacen tan potentes a esta computadoras y con esto están revolucionando la informática. Debido a que los Qubits presentan una superposición de estados, los cálculos aplicados a cada estado es capaz de explorar todos los valores de forma simultánea. Lo anterior representa un nivel alto de procesamiento paralelo físicamente posible [11].

El entrelazamiento es una condición que puede ser inducida para vincular un par de partículas cuánticas las cuales aunque no tienen estados cuánticos independientes, dependerán de la otra y afectan su estado cuántico. El entrelazamiento se conserva a pesar de la distancia que los separa. Aunque este avance consiste en una arquitectura de magnifico poder y paralelismo existen problemas significativos que deben superarse. Un problema a resolver es determinar el mejor método para ser utilizado en la fabricación del hardware.

En este momento existen dos tipos de computadoras cuánticas la IBM q y D-wave, diferentes por su modo de operación. Los Qubits pueden ser representados por átomos, iones, electrones o fotones [5].

Los mecanismos para congelar al Qubit son diversos y cada uno tiene ventajas y desventajas, los más usados son los siguientes:

1. Trampas de iones: para su captura se utilizan campos ópticos o magnéticos, o alguna combinación de los dos.
2. Ondas de luz para atrapar partículas.
3. Conversión de Qubits a puntos cuánticos: alternativamente, los átomos de impurezas que se encuentran en un semiconductor también se pueden utilizar para la representación de Qubits.

Cualquier método elegido en la construcción de la computadora cuántica, debe proporcionar una solución práctica a cuestiones tales como la de coherencia y corrección de errores. La de-coherencia es la tendencia a que el estado cuántico de una partícula a decaer a un estado arbitrario como un resultado de la interacción con su entorno [6, 12].

El primer algoritmo que se desarrolló para las computadoras cuánticas fue creado en 1994 por Peter Shor, de los laboratorios Bell. Se desarrolló un algoritmo cuántico de complejidad computacional polinomial que se utiliza para factorizar números muy grandes [8, 9].

Una computadora clásica resuelve este algoritmo con una complejidad computacional exponencial, lo que hace que no puedan usarse para resolver el problema. La mayoría de los algoritmos de criptografía principales se basan en la dificultad para resolver este problema. Es importante mencionar que los algoritmos cuánticos no devuelven una respuesta precisa en la forma en que los algoritmos deterministas clásicos lo hacen, su respuesta tiene cierta probabilidad de ser una solución. El algoritmo de Shor es extremadamente potente, pero todavía tiene la posibilidad de fracasar, y siempre hay una posibilidad de que los factores que devuelve simplemente será el número en sí y uno, que no es muy útil, incluso si existen otros factores.

2. Descripción del método

El diseño de nuestra investigación incluye en primer lugar el desarrollo del algoritmo de Shor usando el lenguaje ensamblador cuántico QASM. En este código se plantean las técnicas de recolección de datos y análisis. En segundo lugar, se realiza el circuito y se simula en la supercomputadora IBM q. En tercer lugar, se obtiene la información necesaria par la comparación con otros códigos y la aportación.

Se realizar comprobación matemática y se comprobó su eficacia y exactitud.

2.1. Diseño del algoritmo

En computación clásica no se conoce ningún algoritmo clásico eficiente para realizar la factorización de una serie de datos de grande. Se dice que un algoritmo es eficiente si su tiempo de ejecución es asimétricamente polinomial en la longitud de su entrada usando bits.

Su algoritmo equivalente es el tamiz cuadrático que tiene una complejidad computacional de

$$O(\exp((\frac{64}{9})^{\frac{1}{3}}N^{\frac{1}{3}}(\ln N)^{\frac{2}{3}}))$$

para factorizar un número binario de N bits.

La multiplicación de números primos grandes es por lo tanto, una función unidireccional, es decir, una función que puede evaluarse fácilmente en una dirección, mientras que su inversión es prácticamente imposible. Las funciones unidireccionales

desempeñan un papel importante en la criptografía y son esenciales para los sistemas criptográficos de clave pública, donde la clave para la codificación es pública y solo la clave para la decodificación permanece en secreto.

En 1978, Rivest, Shamir y Adleman desarrollaron un algoritmo criptográfico basado en el carácter unidireccional de multiplicar dos números primos grandes (100 dígitos), así nació el método RSA y se convirtió en el sistema de clave pública más popular. Se ha demostrado en la literatura que la factorización prima eficiente en una computadora clásica es imposible, Shor propuso un algoritmo eficiente para computadoras cuánticas en 1994.

Se desarrolló un algoritmo secuencial para solucionar este problema en una computadora clásica el algoritmo desarrollado se muestra en el Algoritmo 1. El procedimiento comprueba si el número entero es adecuado para la factorización cuántica, y luego repite el algoritmo de Shor hasta que se haya encontrado un factor.

Algoritmo 1. Algoritmo de Shor clásico.

```
procedimiento shor (número int) {
    int width = ceil (log (número, 2));          // tamaño del número en bits
    qureg reg1 [2 * ancho];                      // primer registro
    qureg reg2 [ancho];              // segundo registro
    int qmax = 2 ^ ancho;
    int factor;                                  // factor encontrado
    int m;  real c;                              // valor medido
    int x;                                       // base de exponenciación
    int p;  int q;                               // aproximación racional p / q
    int a;  int b;                               // posibles factores de número
    int e;                                       // e = x ^ (q / 2) número mod.

    si el número mod 2 == 0 {exit "number debe ser impar";  }
    si testprime (número) {exit "número primo";  }
    si testprimepower (number) {exit "prime power";  };

    {
     {// generar base aleatoria
       x = piso (random () * (number-3)) + 2;
     } hasta gcd (x, number) == 1;
     imprimir "elegido al azar x =", x;
     Mix (reg1);                                 // Transformada Hadamard
     expn (x, número, reg1, reg2);               // exponenciación modular
     medir reg2;                                       // measure 2nd register
     dft (reg1);                                   // Transformada de Fourier
     medir reg1, m;                                    // medida 1er registro
     Reiniciar;                                        // borrar los registros locales

     si m == 0 {                                 // falló si se midió 0
      imprimir "cero medido en el 1er registro. Intentar de nuevo ...";
     } else {
      c = m * 0.5 ^ (2 * ancho);                 // forma de punto fijo de m
      q = denominador (c, qmax);     // encuentra una aproximación racional
      p = piso (q * m * c + 0.5);
```

```
  imprimir "medido", m, ", aproximación para", c, "es", p, "/", q;
  si q mod 2 == 1 y 2 * q <qmax {          // impar q?  intenta expandir p / q
   imprimir "denominador impar, expandiendo por 2";
   p = 2 * p;  q = 2 * q; }
  si q mod 2 == 1 {                        // falló si es impar q
   imprimir "período impar. Intentar de nuevo ...";
  } else {
   imprimir "el posible período es", q;
   e = powmod (x, q / 2, número);          // calcular los candidatos para
   a = (e + 1) número de modulación;       // posibles factores comunes
   b = (e + número-1) número de modulación;          // con número
   imprimir x, "^", q / 2, "+ 1 mod", número, "=", a, ",",
        x, "^", q / 2, "- 1 mod", número, "=", b;
   factor = max (gcd (número, a), gcd (número, b));
  }
 }
} hasta factor> 1 y factor <number;
número de impresión, "=", factor, "*", número / factor;}
```

3. Implementación cuántica

El algoritmo de Shor busca factorizar un valor dado M> 0, que asumimos que es semi-primo $M = pq$ con factores desconocidos. La estrategia es considerar las funciones $fb\,(x) = xb\,mod\,M^2$ potencialmente con varios valores diferentes de $1 < b < M$ y determinar sus períodos en caso de que $gcd\,(b, M) = 1$. Cuando se determina que el período es incluso $b^{2\pi}\,mod\,M = 1$, tenemos $(b^{\pi} - 1)(b^{\pi} + 1) mod\,M = 0$, por lo tanto cualquiera $(b^{\pi} - 1)$ o $(b^{\pi} + 1)$ debe compartir al menos un factor primo con M.

Si $f^{b\pi}\,mod\,M \neq -1$, tal factor se puede encontrar utilizando $gcd\,(b\pi \pm 1, M)$, de lo contrario, conduce a los factores triviales 1 y M. Cuando se determina que el período es diferente, se prueba otro valor b.

El procedimiento de búsqueda de período se basa en un circuito cuántico (Figura 1), instanciado para un valor dado $1 < b < M$ co primo con M.

El circuito opera en dos registros cuánticos inicializados en 0 con:

1. Un bloque de compuertas Hadamard paralelas en el Registro 1,
2. Un circuito para la exponenciación modular (mod-exp) evaluada con $f\,(y) = b^{y} mod\,M$ mediante el mapeo $|\,y\rangle|\,0\rangle \rightarrow |\,y\rangle|\,f\,(y)\rangle$, donde y lee el Registro 1 y $f\,(y)$ escribe para registrar 2; El registro 1 puede modificarse temporalmente, pero debe restaurarse al final,
3. Un circuito para la Transformada Cuántica de Fourier (QFT) en el Registro 1,
4. Un bloque de mediciones paralelas en el Registro 1.

El primer y el último bloque no se pueden optimizar más. Los circuitos QFT se entienden bastante bien y son mucho más pequeños que los circuitos para la exponenciación modular.

Por lo tanto, nuestro enfoque es en los circuitos mod-exp, constan de compuertas reversibles: NOT (N), CNOT (C) y Toffoli (T), que se pueden modelar y optimizar

por completo en términos de lógica booleana. En las implementaciones físicas, las compuertas de Toffoli deben descomponerse en compuertas más pequeñas que son implementables directamente en una tecnología determinada [13].

Los circuitos reversibles para la exponenciación modular comienzan con un inversor en el Registro 2 que cambia el valor de $|000\cdots0\rangle a\ X|000\cdots1\rangle$, y SE muestran la siguiente estructura:

1. Cada bit (i-ésimo) del Registro habilita un bloque de circuito que multiplica el Registro 2 por $Ci = b^2 \, mod \, M$ y reduce el resultado del modulo de M.
2. Cuando se conocen b y M, Ci puede pre calcularse sin cálculo cuántico. Por lo tanto, nos referimos a $Cixmod\ M$ blocks.
3. Normalmente se implementan usando circuitos de desplazamiento y adición, y se conocen varios sumadores cuánticos relevantes.

Cada multiplicación modular controlada se implementa tradicionalmente separada. Cuando se trata de circuitos lógicos y cuánticos reversibles, observamos que la co-primidad de C y M hace que $x \rightarrow Cx \, mod \, M$ sea una transformación reversible [14]. El número de valores co-primos de C es $\varphi\,(M) = (p-1)\,(q-1)$, donde $\varphi\,(M)$ es la Función totient de Euler y da el tamaño de $(Z\ /\ MZ)^x$ - el grupo multiplicativo de enteros para mod-M.

Para M = 15, los circuitos modulares de multiplicación para los ocho valores C co primos se ilustran en la Figura 3.

La figura 4 muestra los circuitos para $f\,(x) = b^x \, mod \, 15$, $gcd\,(b, 15) = 1$.

Cuando no se conocen p y q, tampoco se debe asumir ningún conocimiento que facilite encontrarlos.

Cuando se trabaja con un valor $M = pq$ pequeño es difícil evitar el uso de los valores conocidos de p y q, pero los resultados obtenidos de esta manera no se escalan necesariamente a valores grandes. Lo mismo puede decirse sobre los resultados producidos a través de una búsqueda exhaustiva.

El procedimiento ajustado se muestra en el Algoritmo 2. El listado se encuentra en Qiskit de IBM.

Algoritmo 2. Algoritmo de Shor cuántico.

```
#-----------------------------------------------------------------------------------------------
# Importar módulos necesarios
#-----------------------------------------------------------------------------------------------
import sys
from qiskit import QuantumProgram
import Qconfig
import Basic_gates
import math
from random import randint
import control_unitaries
import xlsxwriter
#-----------------------------------------------------------------------------------------------
# variable globales
#-----------------------------------------------------------------------------------------------
```

```
Counts = 0
A = 0
Ran_Quantum_period_finding = 0
global m
m = 0
#-----------------------------------------------------------------------------------------------
# La función para calcular GCD usando el método de Euclides
# Entrada: dos números a X e Y para los cuales se debe calcular un GCD
# Salida: GCD de dos números dados
#-----------------------------------------------------------------------------------------------
def gcd(x,y):
   while y != 0:
      (x, y) = (y, x % y)
   return x
#-----------------------------------------------------------------------------------------------
# La función para construir el unitario
# Entrada: objeto del programa Quantum, el nombre del circuito y el nombre del registro
cuántico
# Salida: ninguna. El circuito relevante está hecho.
#-----------------------------------------------------------------------------------------------
def cmod(Quantum_program_object,Circuit_name,Quantum_register_name,a):
# Get the circuit and the quantum register by name
        qc = Quantum_program_object.get_circuit(Circuit_name)
        qr = Quantum_program_object.get_quantum_register(Quantum_register_name)

# Construct unitary based on a
        if a == 2:
                qc.cswap(qr[4],qr[3],qr[2])
                qc.cswap(qr[4],qr[2],qr[1])
                qc.cswap(qr[4],qr[1],qr[0])
        if a == 4 or a == 11 or a == 14:
                qc.cswap(qr[4],qr[2],qr[0])
                qc.cswap(qr[4],qr[3],qr[1])
                qc.cx(qr[4],qr[3])
                qc.cx(qr[4],qr[2])
                qc.cx(qr[4],qr[1])
                qc.cx(qr[4],qr[0])
        if a == 7:
                qc.cswap(qr[4],qr[1],qr[0])
                qc.cswap(qr[4],qr[2],qr[1])
                qc.cswap(qr[4],qr[3],qr[2])
                qc.cx(qr[4],qr[3])
                qc.cx(qr[4],qr[2])
                qc.cx(qr[4],qr[1])
                qc.cx(qr[4],qr[0])
        if a == 8:
                qc.cswap(qr[4],qr[1],qr[0])
                qc.cswap(qr[4],qr[2],qr[1])
                qc.cswap(qr[4],qr[3],qr[2])
        if a == 13:
                qc.cswap(qr[4],qr[3],qr[2])
                qc.cswap(qr[4],qr[2],qr[1])
```

```
                qc.cswap(qr[4],qr[1],qr[0])
                qc.cx(qr[4],qr[3])
                qc.cx(qr[4],qr[2])
                qc.cx(qr[4],qr[1])
                qc.cx(qr[4],qr[0])

#-------------------------------------------------------------------------------------------
# La función para calcular QFT
# Entrada: circuito, bits cuánticos y cantidad de bits cuánticos
# Salida: ninguna. Circuito creado y guardado
#-------------------------------------------------------------------------------------------
def qft(Quantum_program_object, Circuit_name, Quantum_register_name,
Smallest_Quantum_register_number, Size_of_QFT):
# Get the circuit and the quantum register by name
        qc = Quantum_program_object.get_circuit(Circuit_name)
        qr = Quantum_program_object.get_quantum_register(Quantum_register_name)
        s = Smallest_Quantum_register_number
    for j in range(Size_of_QFT):
                for k in range(j):
                        qc.cu1(math.pi/float(2**(j-k)), qr[s+j], qr[s+k])
                qc.h(qr[s+j])

#-------------------------------------------------------------------------------------------
# La función para encontrar el período usando la computadora Quantum
# Entrada: a y N para la cual se debe calcular el período.
# Salida: período r de la función a ^ x mod N
#-------------------------------------------------------------------------------------------
def period(a,N):
        global Ran_Quantum_period_finding
        Ran_Quantum_period_finding = 1
# Create the first QuantumProgram object instance.
        qp = QuantumProgram()
# qp.set_api(Qconfig.APItoken, Qconfig.config["url"])
# TO DO : generalize the number of qubits and give proper security against rogue input.
# Create the first Quantum Register called "qr" with 12 qubits
        qr = qp.create_quantum_register('qr', 5)
# Create your first Classical Register  called "cr" with 12 bits
        cr = qp.create_classical_register('cr', 3)

# Create the first Quantum Circuit called "qc" involving your Quantum Register "qr"
# and the Classical Register "cr"
        qc = qp.create_circuit('Period_Finding', [qr], [cr])

# Get the circuit and the registers by name
        Shor1 = qp.get_circuit('Period_Finding')
        Q_reg = qp.get_quantum_register('qr')
        C_reg = qp.get_classical_register('cr')

# Create the circuit for period finding
# Initialize qr[0] to |1>
        Shor1.x(Q_reg[0])
        # Step one : apply a**4 mod 15
```

```
        Shor1.h(Q_reg[4])
# Controlled Identity on the remaining 4 qubits. Which is equivalent to doing nothing
        Shor1.h(Q_reg[4])
        Shor1.measure(Q_reg[4],C_reg[0])
# Reinitialize to |0>
        Shor1.reset(Q_reg[4])

# Step two : apply a**2 mod 15
        Shor1.h(Q_reg[4])
# Controlled unitary. Apply a mod 15 twice.
        for k in range(2):
                cmod(qp,'Period_Finding','qr',a)
        if C_reg[0] == 1 :
                Shor1.u1(pi/2.0,Q_reg[4])
        Shor1.h(Q_reg[4])
        Shor1.measure(Q_reg[4],C_reg[1])
# Reinitialize to |0>
        Shor1.reset(Q_reg[4])

# Step three : apply 11 mod 15
        Shor1.h(Q_reg[4])
# Controlled unitary. Apply a mod 15
        cmod(qp,'Period_Finding','qr',a)
# Feed forward and measure
        if C_reg[1] == 1 :
                Shor1.u1(pi/2.0,Q_reg[4])
        if C_reg[0] == 1 :
                Shor1.u1(pi/4.0,Q_reg[4])
        Shor1.h(Q_reg[4])
        Shor1.measure(Q_reg[4],C_reg[2])

        # Run the circuit
        # qp.set_api(Qconfig.APItoken, Qconfig.config['url']) # set the APIToken and API url
        simulate = qp.execute(["Period_Finding"], backend="local_qasm_simulator",
shots=1,timeout=500)
        simulate.get_counts("Period_Finding")
        #print(simulate)
        data = simulate.get_counts("Period_Finding")
        #print(data)
        data = list(data.keys())
        #print(data)
        r = int(data[0])
        #print(r)
        l = gcd(2**3,r)
        #print(l)
        r = int((2**3)/l)
        #print(r)
        return r

#---------------------------------------------------------------------------------------------
# La función principal para calcular los factores
# Entrada: el número a factorizar, N
```

```
# Salida: factores del número
#-------------------------------------------------------------------------------------------
def Factorize_N(N):
        factors = [0,0]
#-------------------------------------------------------------------------------------------
# Paso 1: Determine la cantidad de bits en función de N; n = [log2 (N)]
#-------------------------------------------------------------------------------------------
        n = math.ceil(math.log(N,2))
#-------------------------------------------------------------------------------------------
# Paso 2: comprueba si N es par. En ese caso, devuelve 2 y el número restante como factores
#-------------------------------------------------------------------------------------------
        If  N % 2 == 0:
                factors = [2,N/2]
                return factors
#-------------------------------------------------------------------------------------------
# Paso 3: compruebe si N tiene la forma P ^ (k), donde P es un factor primordial. En ese caso
devuelve P y k.
# ------------------------------------------------ ----------------------------------------
# El paso ha sido eliminado con fines de simulación.
# ------------------------------------------------ ----------------------------------------
# Paso 4: elija un número aleatorio entre 2 ... (N-1).
#-------------------------------------------------------------------------------------------
        while True:
                a = randint(2,N-1)
                global A
                A = a
#-------------------------------------------------------------------------------------------
# Paso 5: Tome GCD de ay N. t = GCD (a, N)
#-------------------------------------------------------------------------------------------
                t = gcd(N,a)
                if t > 1:
                        factors = [t,N/t]
                        return factors
#-------------------------------------------------------------------------------------------
# Paso 6: t = 1. Por lo tanto, no hay un período común. Encontrar período usando el método
de Shor
#-------------------------------------------------------------------------------------------
                r = period(a,N)
                if (r%2 == 0) and (((a**(r/2))+1)%N != 0) and (r != 0) and (r != 8):
                        break
                global Counts
                Counts = Counts + 1
        factor_1 = gcd((a**(r/2))+1,N)
        factor_2 = N/factor_1
        factors = [factor_1,factor_2]
        return factors
#-------------------------------------------------------------------------------------------
# Ejecutando la versión 1 del algoritmo de Shor
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------
# Paso 0: Toma la entrada N
#-------------------------------------------------------------------------------------------
```

```
factors_list = list()
A_used = list()
Ran_QPF = list()
Total_counts = list()
if __name__ == '__main__':
        #global m
        for m in range(100):
                N = 15
                factors_found = Factorize_N(N)
                factors_list.append(factors_found)
                #ws.write(row,col,A)
                #ws.write(row,col+1,Counts)
                #ws.write(row,col+2,factors[0])
                #ws.write(row,col+3,factors[1])
                #ws.write(row,col+4,Ran_Quantum_period_finding)
                #row = row + 1
                #print("The Number being factorized is 15")
                #print("Factors are = ",factors)
                #print("Number of times the quantum circuit did not give correct period =
",Counts)
                #print ("The parameter a used = ", A)
                print("Run ", m)
                A_used.append(A)
                Ran_QPF.append(Ran_Quantum_period_finding)
                Total_counts.append(Counts)
                Counts = 0
                Ran_Quantum_period_finding = 0


if m == 99:
        wb = xlsxwriter.Workbook('log.xlsx')
        ws = wb.add_worksheet('Data')
        row = 0
        col = 0
        ws.write(row,col,'a used for factorizing')
        ws.write(row,col+1,'Number of times the quantum circuit did not give correct
period')
        ws.write(row,col+2,'Factor1')
        ws.write(row,col+3,'Factor2')
        ws.write(row,col+4,'Ran_Quantum_period_finding?')
        row = row + 1

        for k in range(100):
                ws.write(row,col,A_used[k])
                ws.write(row,col+1,Total_counts[k])
                ws.write(row,col+2,factors_list[k][0])
                ws.write(row,col+3,factors_list[k][1])
                ws.write(row,col+4,Ran_QPF[k])
                row = row + 1
        wb.close()
```

Después de comprobar su programación se procedió a crear el circuito en IBM *experience* [10][1]. Valores obtenidos de esta manera no se escalan necesariamente a valores grandes. Lo mismo puede decirse sobre los resultados producidos a través de una búsqueda exhaustiva.

4. Resultados

La complejidades de los algoritmos se muestra en la tabla 1. La Figura 1 muestra su curva típica.

Tabla 1. Complejidades de los algoritmos de Shor

Shor clásico	Shor cuántico
t~O(EX(ln n)^1/3(ln lnn)^2/3)	t ~ O(n3)

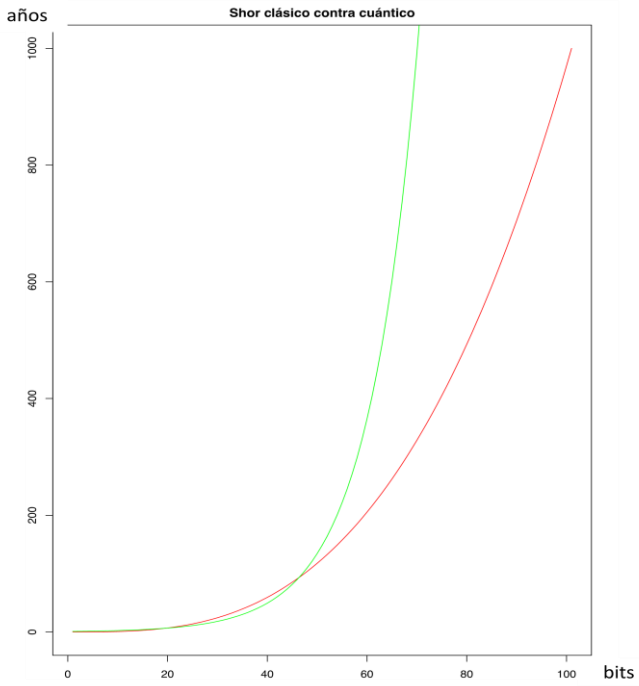


Fig. 1. Comparación de las complejidades computacionales para ambos (años contra bits procesados).Las líneas en color verde representan el algoritmo clásico, las líneas rojas el algoritmo cuántico

La interpretación de los resultados se pueden establecer desde el modelo cuántico a utilizar, los modelos cuánticos son:

[1] https://quantumexperience.ng.bluemix.net/qx/editor

1. Modelo basado en compuertas
2. Modelo basado en la maquina de Turín Cuántica [7]
3. Modelo basado en el calculo de Lambda
4. Topología adiabática
5. Modelo basado en medición

Dado que el desarrollo de la investigación esta basada en la creación de los algoritmos de programación establecemos como importante el desarrollo de los procedimientos basándonos en un modelo universal como lo es el modelo basado en compuertas cuánticas (MBCC) [3].

Se realizó la adecuación MBCC del listado 2. Para construir el listado en Qiskit de IBM mostrado en el Algoritmo 3.

Algoritmo 3. Algoritmo de Shor en Qiskit de IBM.

```
//OPENQASM 2.0
IBMQASM 2.0;
include "qelib1.inc";
qreg q[5];
creg c[5];
x q[2];
x q[1];
x q[2];
x q[3];
x q[4];
cx q[3],q[2];
cx q[2],q[3];
cx q[3],q[2];
cx q[2],q[1];
cx q[1],q[2];
cx q[2],q[1];
cx q[4],q[1];
cx q[1],q[4];
cx q[4],q[1];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4]; }
```

Se realizó la simulación del circuito usando el simulador QX-Studio para definir con más precisión las compuertas a utilizar. Al realizar la simulación se obtuvieron los siguientes valores:

```
----------[Estado cuántico Inicial]----------
(+1.000000,+0.000000) |1101> +
-------------------------------------------
[>>] prediccion en la medida: | 1 | 1 | 0 | 1 |
```

[>>] Medida en el registro inicial : | 0 | 0 | 0 | 0 |

----------[Estado cuántico final]----------

(+1.000000,+0.000000) |1011> +

[>>prediccion en la medida: | 1 | 0 | 1 | 1 |

[>>]Medida en el registro final: | 1 | 0 | 1 | 1 |

Tiempo de ejecución del circuito: +0.000045 sec.

La figura 1 Muestra el circuito construido por el simulador QX Studio. Con este circuito se realizó el análisis de el comportamiento sobre un registro binario cuántico para determinar y comprobar matemáticamente que los resultados sean coherentes, en la Tabla 3 se muestra el proceso de comprobación por registro de cada valor obtenido en los registros en los 14 pasos observados.

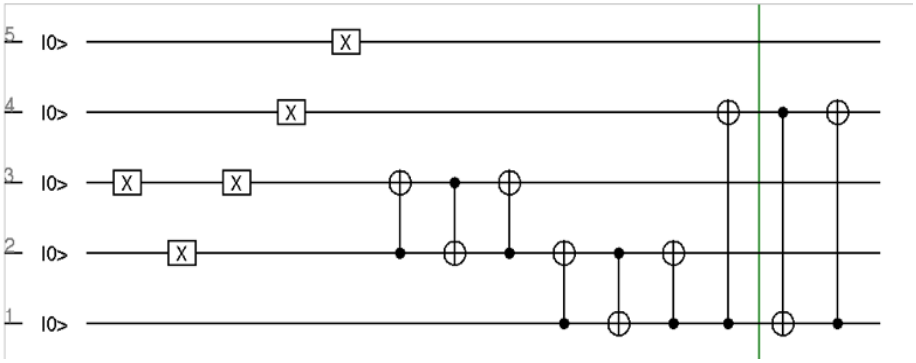
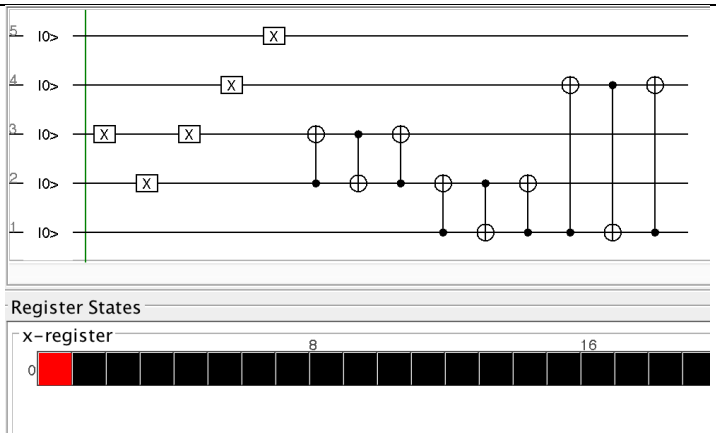


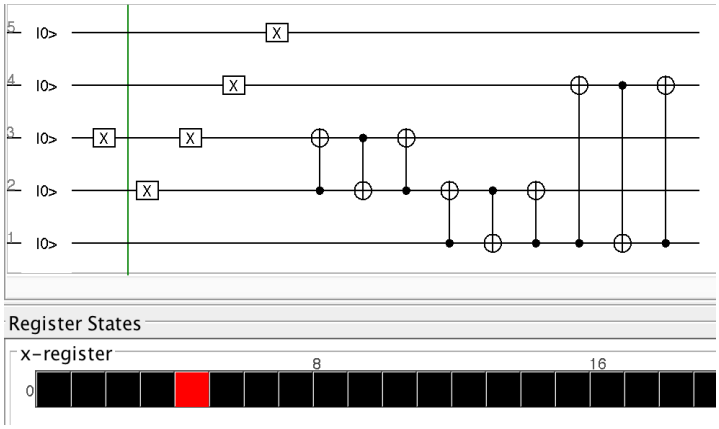
Fig. 2. Circuito cuántico resultante del análisis del código del Algoritmo 3.

Tabla 3 muestra los 14 pasos para comprobar cada compuerta en el circuito cuántico propuesto.

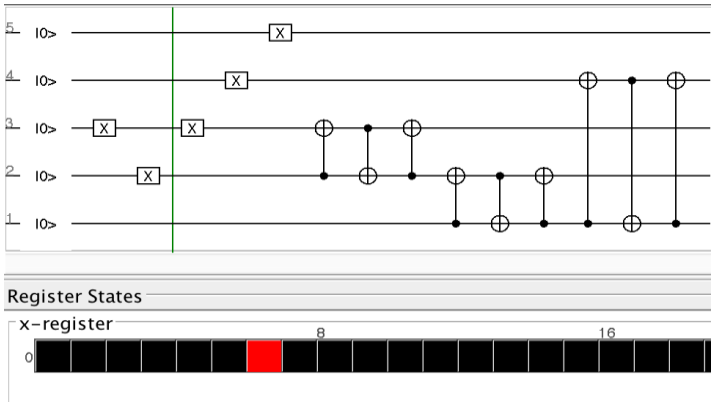
Como último paso, se procedió a implementar las compuertas de acuerdo al listado 3 en la computadora cuántica IBM Q de manera real. La Figura 3 muestra el circuito básico construido en IBM *experience*.



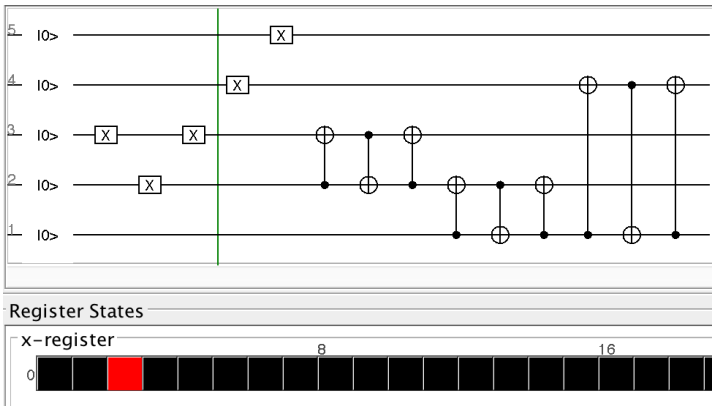
Paso 1



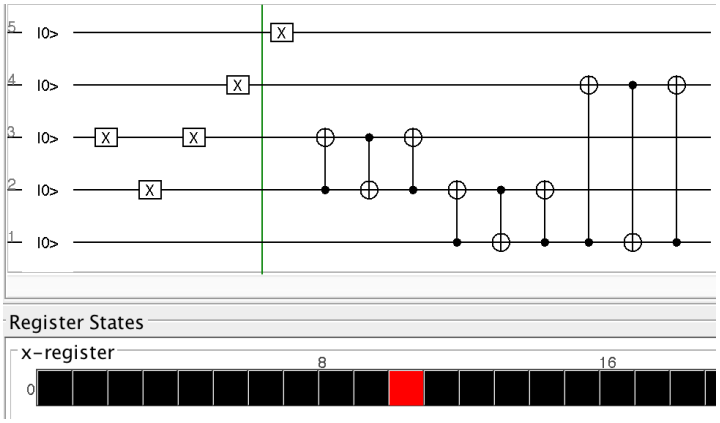
Paso 2



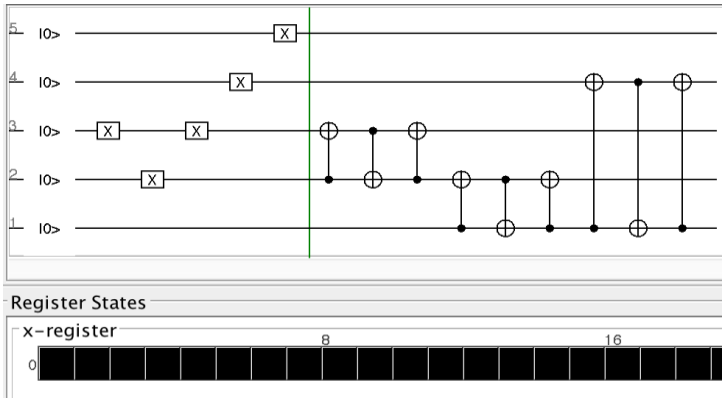
Paso 3



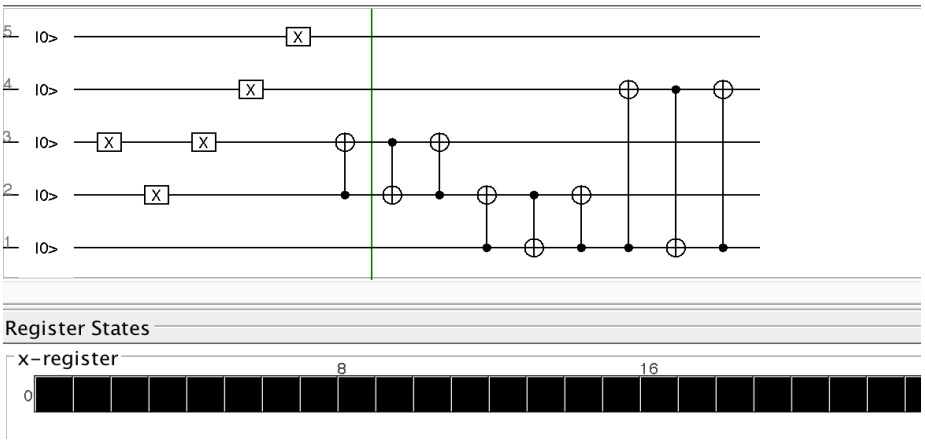
Paso 4



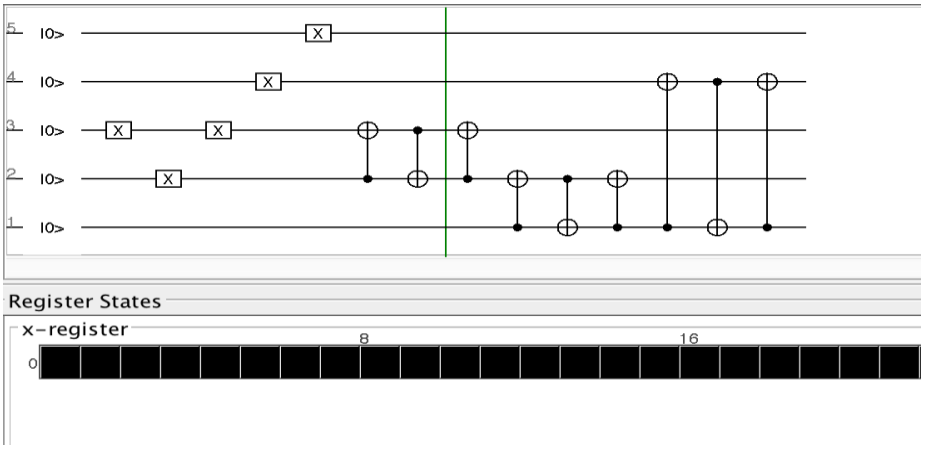
Paso 5



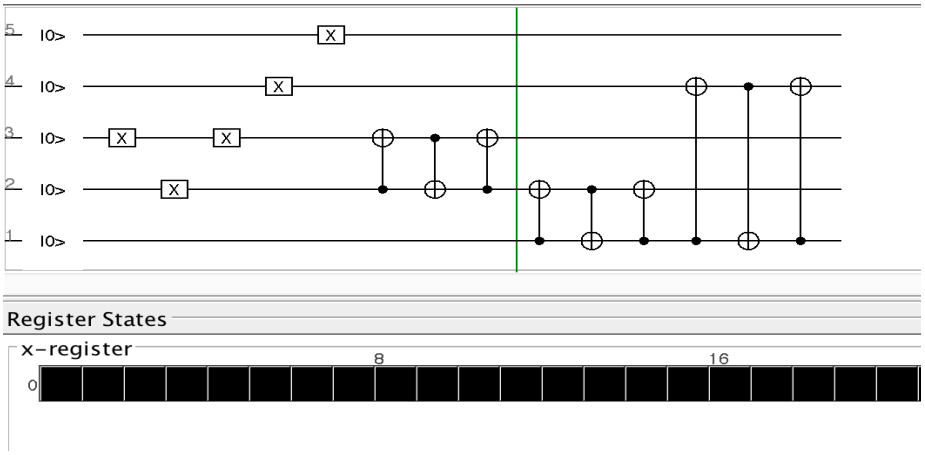
Paso 6



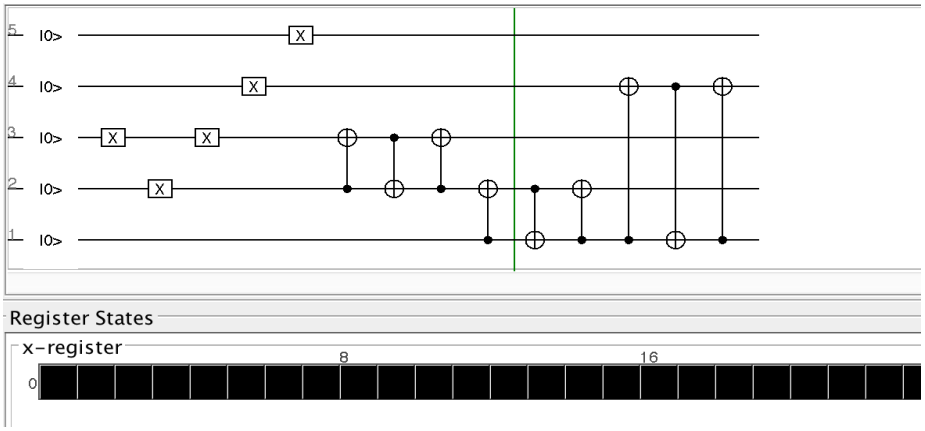
Paso 7



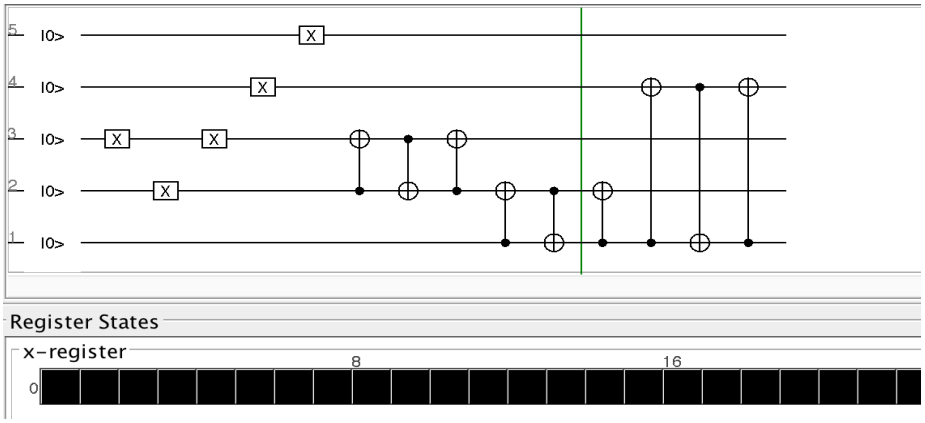
Paso 8



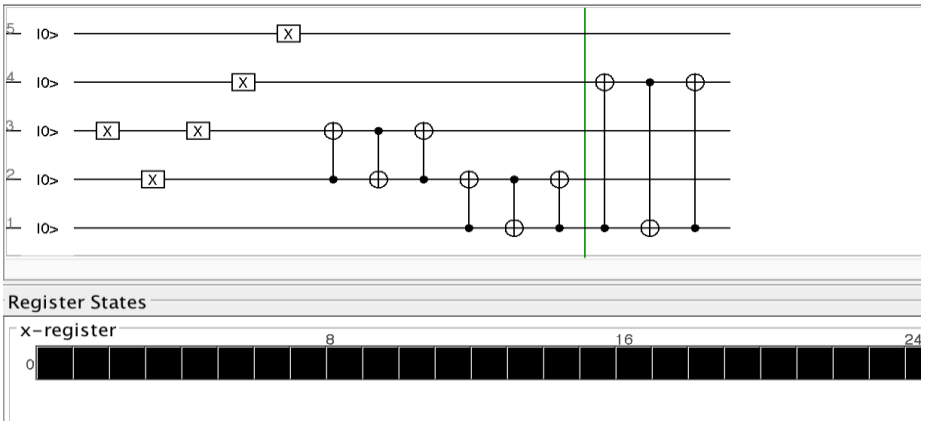
Paso 9



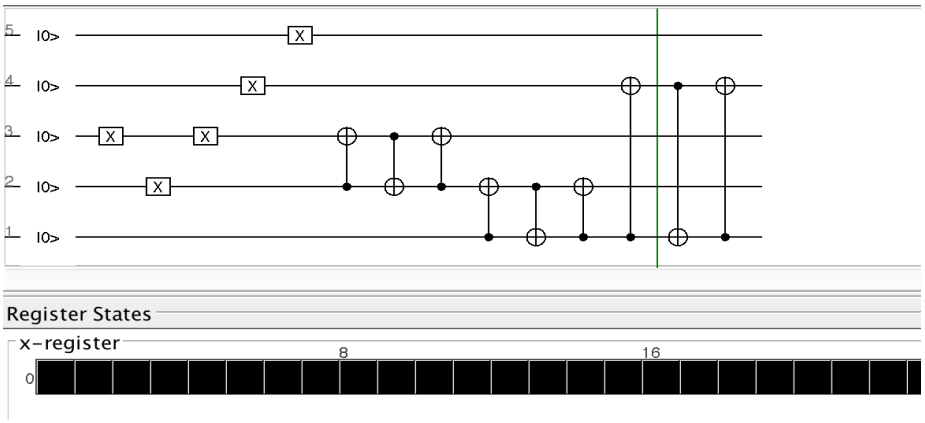
Paso 10



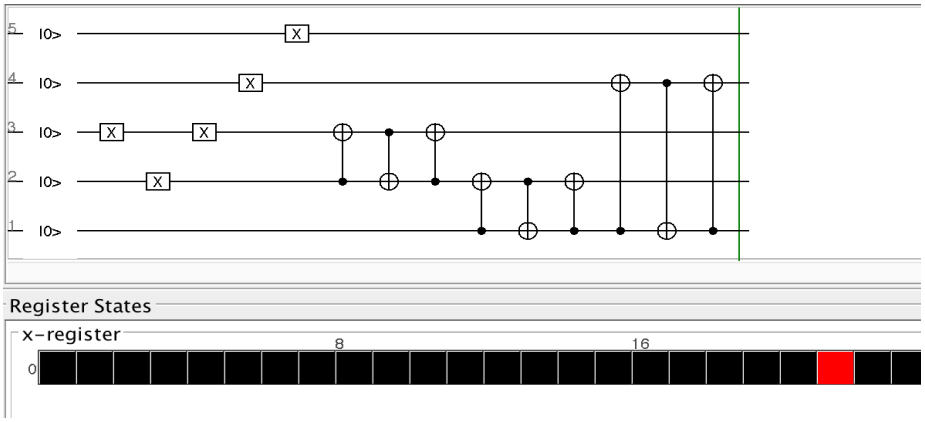
Paso 11



Paso 12



Paso 13



Paso 14

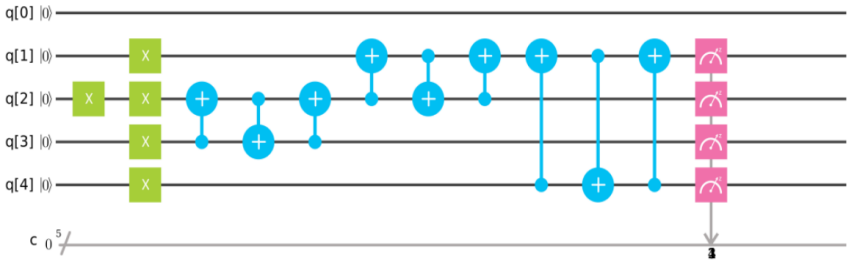
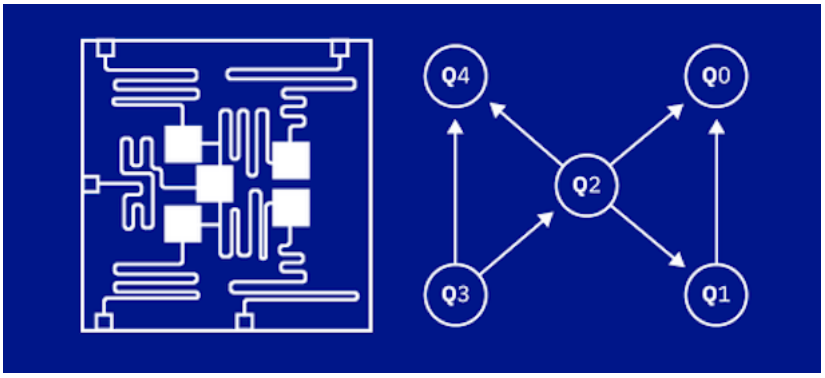


Fig. 3. Circuito cuántico propuesto en IBM Experience.

Este circuito fue compilado en la IBM q de IBM y corrió de acuerdo a los parámetros mostrados en la Tabla 2.

Tabla 2. Parámetros de funcionamiento.

Semilla	2007893536					
Numero de muestras	100					
Temperatura	0.021 K					
Backend	Ibmqx4(5 qubits)	Q0	Q1	Q2	Q3	Q4
Error por compuerta	10e-3	0.86	0.69	1.97	1.97	1.8
Error de lectura	10e-2	4.60	5.40	12.80	8.70	4.5
Error de compuerta multi-Qubit	10 e -2	5.85	1.99	2.48	2.68	.94



Ibmqx4(5 qubits)

La Figura 4 muestra el cálculo final obtenido.

Fig. 4. Cálculo final para 5 Qubits.

Resumen de resultados. Se pudo comprobar que nuestro algoritmo desarrollado trabaja perfectamente al realizar su comparación con otros

algoritmos desarrollados en la literatura, sin embargo la ventaja que presentamos es su adaptación para su comprobación en la computadora cuántica de IBM (IBM Q).

5. Conclusiones

La invención de una computadora cuántica práctica sea el comienzo de una nueva era en la computación y la tecnología en general. Dicha tecnología nos permitirá romper muchas de las barreras físicas que actualmente limitan las posibilidades de la tecnología informática clásica, mientras que la introducción de las nuevas posibilidades.

Sin embargo, aún existen varios obstáculos que deben superarse. Computadoras cuánticas actuales tienden a ser extremadamente caros, y requieren entornos operativos específicos, tales como ser capaz de refrigerar los componentes de la computadora cuántica cerca del cero absoluto. Muchos de los métodos que se han utilizado con éxito hoy en día no se escala fácilmente para permitir el uso de un mayor número de Qubits.

Sin embargo, hay una tremenda presión en esta zona, y es sólo cuestión de tiempo antes de que se superen estos obstáculos. Se requerirá más años de investigación y desarrollo antes de la edad de la computación cuántica llega, pero los bloques de construcción básicos que ya se han realizado. Las computadoras cuánticas se han construido ya que pueden realizar funciones básicas, y ya hay empresas que tratan de comercializar la tecnología.

Por ejemplo, D-Wave Systems es una empresa canadiense que afirma haber desarrollado una computadora cuántica de 28 Qubits, aunque ha habido críticas a sus demandas. En 2007, demostraron el uso de un equipo de 16 Qubits para resolver problemas tales como el reconocimiento de patrones, disposición de los asientos, y un Sudoku.

Incluso después de que las computadoras cuánticos prácticos se realizan, es poco probable que van a sustituir por completo a las computadoras clásicas. Su ventaja sobre las computadoras clásicas es significativa sólo en áreas específicas de aplicación. Lo más probable es que las computadoras del futuro tendrán lugar una especie de híbrido, que contiene los componentes de ambos tipos de computadoras.

La verificación de artículos que cumplen con este tipo de tipo de clasificación propuesto fue solo el 40 % debido a algunos pierden el enfoque de su investigación hablando de otros elementos que para nuestra propuesta no tienen relación.

Agradecimientos. Agradecemos las facilidades otorgadas para la realización de este trabajo al Instituto Politécnico Nacional a través de la Secretaría de Investigación y Posgrado con el proyecto SIP 20180023. A la Unidad Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas y Centro de Investigación y Desarrollo de Tecnología Digital. Asimismo, al Programa de Estímulo al Desempeño de los Investigadores (EDI) y al Programa de Estímulo al Desempeño Docente (EDD).

Referencias

1. Cooper, K.B., Steffen, Matthias, McDermott, R. et al.: Observation of Quantum Oscillations between a Josephson Phase Qubit and a Microscopic Resonator Using Fast Readout. Physical Review Letters, 29 October (2004)
2. Devoret, Wallraff, Martinis: Superconducting Qubits: A Short Review. Department of Applied Physics, Yale University (2004)
3. Devoret: Manipulating the Quantum State of an Electrical Circuit. Science: 886–889, 3rd May (2002)
4. Han, J., Jonker, P.: On Quantum Computing with macroscopic Josephson Qubits. Pattern Recognition Group, Faculty of Applied Sciences, Delft University of Technology (2002)
5. Gruska, J.: Fundamentos de la informática, cap. 12.: Frontiers - Quantum Computing (1998)
6. McDermott, R., Simmonds, R.W., Steffen, Matthias, et al.: Simultaneous State Measurement of Coupled Josephson Phase Qubits. Science: 1299–1302, 25 February (2005)
7. Han, M.: Superconducting quantum qubits. Physics World, December (2004)
8. Benioff, P.: Modelos de máquinas Quantum Turing. Archivo LANL quant-ph / 9708054 (1997)
9. Shor, P. W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20-22 (1994)
10. Shor, P.W.: Quantum, Information Theory: Results and Open Problems. Geom. Funct. Anal. (GAFA), Special Volume - GAF A2000:816–838 (2000)
11. Keyes, RW: IBM experience. IBM J. Res. Desarrollar. 32, 24 (1988)
12. Simmonds, R.W., Hite, D. A., McDermott, R., Steffen, M., Cooper, K. B., Lang, K.M., Martinis, J. M., Pappas, D. R.: Josephson Junctions Materials Research Using Phase Qubits. University of California, Santa Barbara (2005)
13. Simmonds, R.W., Lang, K.M., Hite, S. et al.: Decoherence in Josephson Phase Qubits from Junction Resonators. Physical Review Letters 13, August (2004)
14. Strauch, F.W., Johnson, P. R., Dragt, A. J., Lobb, C. J., Anderson, J. R., Wellstood, F. C.: Quantum Logic Gates for Coupled Superconducting Phase Qubits. Phys. Rev. Lett. 91, 167005 (2003)