Journal of Electrical Systems and Information Technology

RESEARCH Open Access

Rotation of bits: a classical and quantum perspective

Peter Nimbe*, Benjamin Asubam Weyori and Adebayo Felix Adekoya

*Correspondence: peter.nimbe@uenr.edu.gh
University of Energy and Natural Resources, P.O. Box 214, Sunyani, Ghana

Abstract
Bit rotation is an operation similar to shift except that the bits that fall off at one end are put back to the other end. In left rotation, the bits that fall off at left end are put back at right end. In right rotation, the bits that fall off at the right end are put back at the left end. Applications of bit rotation include; registers, cryptography, computing with a single bit string circularly shifted to the right or left based on some position but no work has been done with respect to shifting the bits one position at a time generating emergent bit strings equal to the number of bits-1 from the incident bit string, and then recombining or extracting bit(s) from each of the bit strings or words to form back the incident bit string. In this article, the authors present a new approach of rotating classical bit strings known as CRotate. A quantum approach to bit rotation known as QRotate is presented as well. The quantum perspective uses the concept of bit swapping by avenue of the quantum swap gate in jsqubits. Models and algorithms are duly presented.

Keywords: Classical logic gates, Quantum logic gates, Bitwise operators, Swap gate, Quantum bit rotation, Classical bit rotation

Introduction

Circular bit rotation is an operation where bits are moved in a circular fashion such that bits change their positional values and do not fall off, but rather are placed back to the other end [1, 2]. A left rotation operation is where bits that fall off at the left end are placed back at the right end [1, 2]. In a right rotation operation, bits that fall off at the right end are placed back at the left end [1, 2]. A shift operation that is circular seeks to rearrange the entries present in a data structure by moving the bit to the next position whiles the last bit is moved to the position of the first bit [1, 2]. Elimination of bits is not done by the rotate instructions. For a left rotate (rol), as shown in Fig. 1, bits shifted off the left end of a data word fill the vacated positions on the right. Likewise, for a right rotate (ror), bits "falling off" the right end appear in the vacated positions at left [1, 2].

The bit sequence 00010111 rotated circularly to the left by one bit position produces the bit sequence 00101110 in Fig. 1 above. No bits are lost after the rotational operation or process [1, 2].

With reference to Fig. 2, if the bit sequence 00010111 were subjected to a circular shift of one bit position to the right would yield: 10001011 [1, 2].

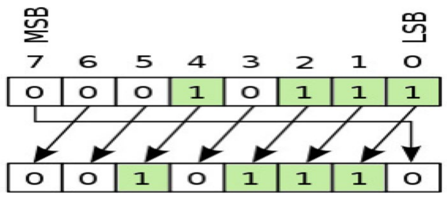


Fig. 1 Rotate left diagram [1, 2]

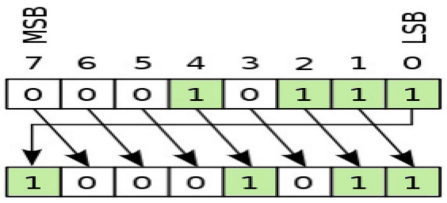


Fig. 2 Rotate right diagram [1, 2]

The concept of circular bit rotation is made clearer by the example below. Assuming we have an 8 bit sequence, say 11100101, moving the bits by 3 places to the left produces 00101111 bit sequence. This implies that the first 3 bits are placed back at the end of the bit sequence. Moving the bits in the original bit sequence by 3 places to the right produces 10111100 bit sequence. This implies that the last 3 bits are placed back at the beginning of the bit sequence. If we have a 16 bit sequence, say 0000000001110010, moving the bits by 3 places to the left produces 0000001110010000 bit sequence. Moving the bits in the original bit sequence by 3 places to the right produces 0100000000001110 bit sequence [1, 2].

A number of cryptographic researchers used the logical XOR gate (digital logical gate which performs a logical operation on one or more logic inputs and produces a single logic output) or operation in their encryption protocols [3, 4].

Rotators and shifter move bits and multiply or divide by powers of 2. A shifter shifts a number in base 2 left or right by some specified number of positions. Some commonly used shifter kinds include; Logical shifter, Arithmetic shifter and Rotator [5, 6]. Logical shifter shifts the number to the left (LSL) or right (LSR) and fills empty spots or positions with zeros. Ex: 11001 LSL 2 = 00100, 11001 LSR 2 = 00110. The arithmetic shifter operates just like the logical shifter but with some slight operation difference [5, 6]. On the right shift, it fills the most significant bit (msb) with a copy of the old significant bit, which is vital or key for dividing and multiplying signed numbers. Arithmetic shift left works the same as the logical shift left. Ex: 11001 ROR 2 = 01110; 11001 ROL 2 = 00111. Rotators rotate the bit string in a circle such that empty spots are filled with bits shifted off the other end. Ex: 11001 ROR 2 = 01110; 11001 ROL 2 = 00111 [5, 6].

A shifter that perform left shifts, logical and arithmetic right shifts, or no shift is shown in Fig. 3 below from a classical perspective [7].

In 2008, Renesas Electronics Corporation demonstrated a method for decreasing the time taken by a H8 CPU in performing an 8 bit rotate from LSB first to

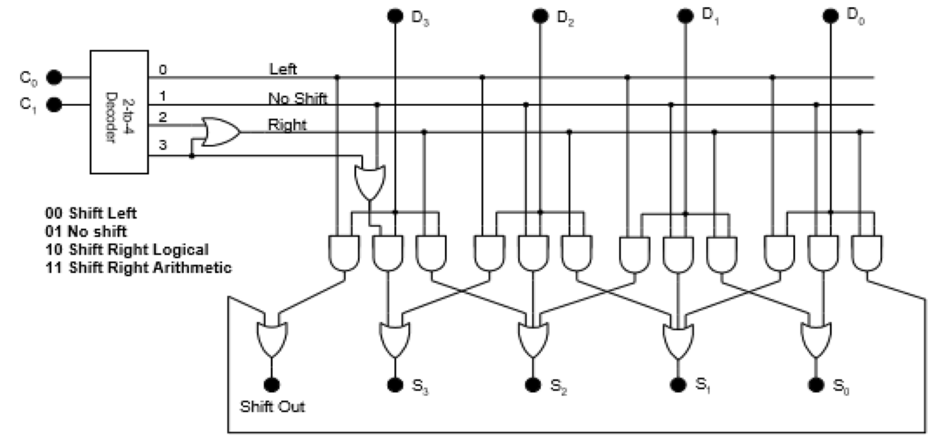


Fig. 3 A shifter that performs left shifts, logical and arithmetic right shifts, or no shift [7]

MSB first. This approach has two methods for completing an 8 bit rotate. The first method employs a look up table; the second rotates physically the data from MSB first to LSB first [8].

The OR, AND, NOT and XOR are simple bit-parallel operations supported in word-oriented processors. They are logical operations that are typically implemented with integer arithmetic operations in computer system, specifically the arithmetic and logic unit, which is the most basic functional unit in a processor. In bit rotations, shifter and mix operations can be used [9].

The standard set of bitwise operations, including OR, AND, XOR, LEFT/RIGHT SHIFT, NOT, is incorporated or available in the C and C++ programming languages. However, circular shift is excluded in the language [10]. When a computer integer is rotated, any bit that falls off one end of the register is moved to the other end as if they are connected end-to-end in a conceptual manner. Circular rotation has some applications in cryptography, for the purposes of encryption and decryption [10].

Bennett disclosed the secret of saving energy by maintaining a unique mapping between input and output vectors called logical reversibility of computation [11]. A reversible circuit is composed of reversible gates only whereas a reversible gate has the property of maintaining one-to-one mapping between input and output vectors. Among the conventional logic gates, NOT operation is the only one which itself is reversible. However, the other conventional logic operations have their reversible counterpart, which is known as $n \times n$ dimensional reversible gate. In designing reversible circuits, there exist 2×2 and several 3×3 gates [12–14]. A typical example of the 2×2 gate is the Feynman gate [12]. Fredkin gate [14] and Feynman double gate [13] are examples of 3×3 gates.

Muwafi et al. also proposed a circuit for rotating, left shifting, or right shifting bit where a circuit for rotating bits of an input word during a single cycle, by duplicating the input word to form an extended word, shifting bits of the extended word, and selecting a subset of the shifted bits of the extended word [15].

Application

Bit rotation is employed in barrel shifters (a logic circuits extensively used in embedded digital signal processors as well as in general purpose processors to manipulate the data as rotating and shifting information is required in few fields including bit-indexing, arithmetic tasks and variable-length coding) [16, 17]. Bharathesh et al. proposed a low power mux based on dynamic barrel shifter using footed diode domino logic [16, 17]. There are bidirectional barrel shifters that can perform six unique tasks: shift right arithmetic (SRA), shift right logical (SRL), rotate left (RL), shift left logical (SLL), shift left arithmetic (SLA), and rotate right (RR) [16, 17].

Shah et al. designed a fully custom 8 bit barrel shifter using 8×1 multiplexer with the help of GDI technique. The barrel shifter is simply a bit-rotating shift register [18]. A robust architecture of logarithmic barrel shifter that performs bidirectional arithmetic and logical shifting, including rotate operation [19]. Aarthi et al. conducted an image encryption using binary bit plane and rotation method for an image security. Yeng et al. also used the concept of bit rotation to design an encryption algorithm [2, 20]. A new universal hash function, circulant hash based on bit rotation was proposed and is a variant of the classical random matrix-based hash of Carter and Wegman, called H_3, and Toeplitz hash by Krawczyk [21]. An encryption approach for Images using Bits Rotation Reversal and Extended Hill Cipher Technique was also devised [22] as well as an Adaptive Bit Rotation and Inversion Scoring, a novel approach to LSB Image Steganography [23].

Quantum perspective

In quantum computing, unit of data is called qubit and the value of qubit is the superposition of $|0>$ and $|1>$. Every quantum operation is reversible if it is represented by a unitary matrix which is used to multiply the state of qubit(s) that produces output [24]. Quantum computers process data by applying a universal set of quantum gates that can emulate any rotation of the quantum state vector [25]. The comparative quantum realization of any reversible circuit is used to verify the operability of that circuit [19].

A quantum circuit called quantum shift register in which shift and rotation operations on qubits are performed by swap gates and controlled swap gates. For quantum computers to perform arithmetic operations that are elementary (bitwise comparison of qubits and multiplication), these operations are of essence [26].

Rotation operators are defined as R_x, R_y and R_z and are shown in Fig. 4 below.

When the Pauli matrices are exponentiated, rotation operators are generated according to $exp\ (i\mathrm{A}x) = \cos(x)\ I + I\ \sin(x)\ A$, where A is one of the three Pauli Matrices [27, 28].

Problem, objective and contribution

The current or existing circular bit rotation techniques never ends because it is in a cycle and is continuous. Unlike a logical shift, the vacant bit positions are not filled in with zeros but are filled in with the bits that are shifted out of the sequence [29]. Repetition of bit strings tend to show up at some point when the bit string is rotated

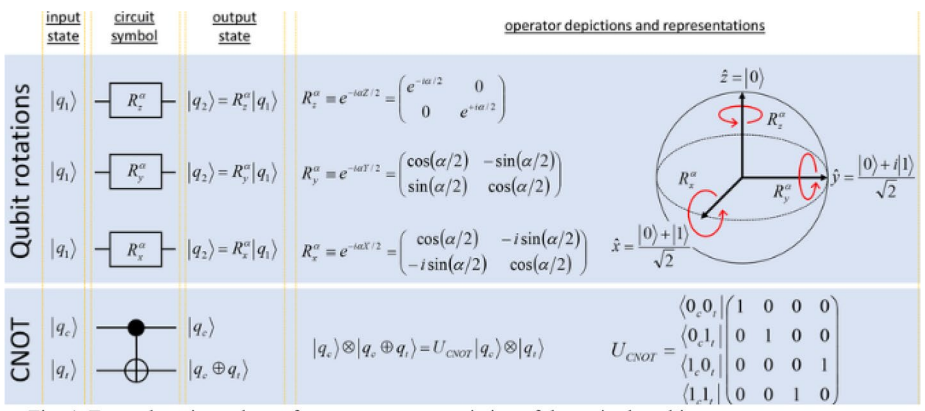

input state
circuit symbol
output state
operator depictions and representations
Qubit rotations
$|q_1\rangle$
R_z^α
$|q_2\rangle = R_z^\alpha|q_1\rangle$
$R_z^\alpha = e^{-i\alpha Z/2} = \begin{pmatrix} e^{-i\alpha/2} & 0 \\ 0 & e^{+i\alpha/2} \end{pmatrix}$
R_y^α
$|q_2\rangle = R_y^\alpha|q_1\rangle$
$R_y^\alpha = e^{-i\alpha Y/2} = \begin{pmatrix} \cos(\alpha/2) & -\sin(\alpha/2) \\ \sin(\alpha/2) & \cos(\alpha/2) \end{pmatrix}$
R_x^α
$|q_2\rangle = R_x^\alpha|q_1\rangle$
$R_x^\alpha = e^{-i\alpha X/2} = \begin{pmatrix} \cos(\alpha/2) & -i\sin(\alpha/2) \\ -i\sin(\alpha/2) & \cos(\alpha/2) \end{pmatrix}$
$\hat{z} = |0\rangle$
$\hat{y} = \frac{|0\rangle + i|1\rangle}{\sqrt{2}}$
$\hat{x} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$
CNOT
$|q_c\rangle$
$|q_t\rangle$
$|q_c\rangle$
$|q_c \oplus q_t\rangle$
$|q_c\rangle \otimes |q_c \oplus q_t\rangle = U_{CNOT}|q_c\rangle \otimes |q_t\rangle$
$U_{CNOT} = \begin{array}{c|cccc} \langle 0_c 0_t| & 1 & 0 & 0 & 0 \\ \langle 0_c 1_t| & 0 & 1 & 0 & 0 \\ \langle 1_c 0_t| & 0 & 0 & 0 & 1 \\ \langle 1_c 1_t| & 0 & 0 & 1 & 0 \end{array}$

Fig. 4 Example universal set of quantum gates consisting of three single qubit rotation gates and one two-qubit CNOT gate, with circuit symbols, depictions, and representations [25]

circularly using shifts progressively and this poses as a weakness and danger, even to computing and cryptography [2]. For the quantum–classical perspective also, no work has been done with circular bit rotation. The objectives of this paper include;

1. Address the problem of repetitions in emergent/resulting bit strings when the incident/original bit string is rotated circularly.
2. Add or contribute to existing knowledge using a novel concept.
3. Provide a quantum perspective to bit rotation using the quantum swap gate.
4. Provide a performance analysis based on execution times of classical implementation and quantum implementation codes on the binary bit strings.

Method

In the development of the algorithm, the traditional system development life cycle (SDLC) was adopted. Concepts and models which are proven were used in the design process, hence can guarantee an effective and efficient algorithm. In this paper, secondary data were used basically from journals, literature and websites. Primarily, the concepts of circular bit rotation, bit dispersal, bit recombination/extraction and bit prism were used in the design phase of the classical algorithm. There was a strict adherence to the code of ethics for writing manuscripts by ensuring that no plagiarism was made. This work poses no ethical issues or challenges and follows high compliance to ethical standards.

The concept of bit prism is derived from the principle of passing light through a glass prism where the incident light ray comes out as emergent light rays, more like a color spectrum [30, 31] A bit prism is an abstract object or concept which entails the principles of circular bit rotation, dispersal and recombination. An operation where bits that fall off at the left end is put back at the right end and bits that fall off at the right end is put back at the left end.

The concept of bit swapping was used to rotate quantum bit strings made possible by a quantum swap gate using jsqubits runner (an online quantum computer simulator).

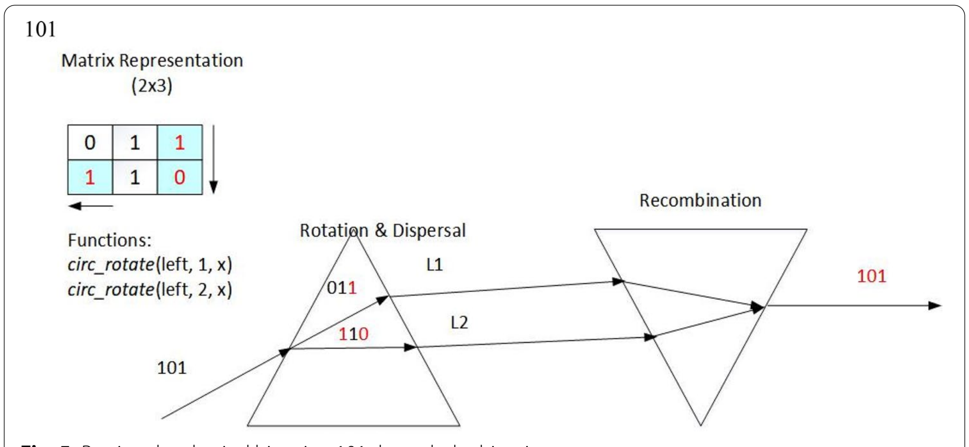


Fig. 5 Passing the classical bit string 101 through the bit prism

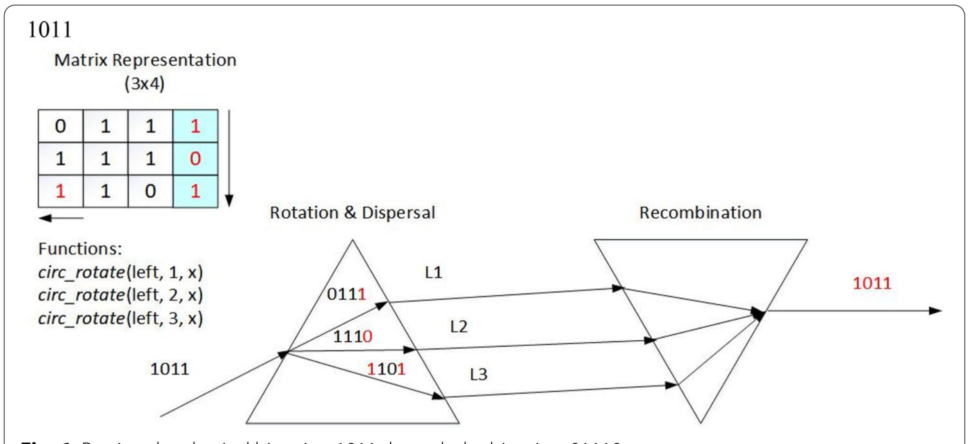


Fig. 6 Passing the classical bit string 1011 through the bit prism 01110

Classical perspective (CRotate)

In a bit prism, the incident word or classical string is not parallel to emergent word. When a word is passed through the bit prism it gets deviated a number of times ($n - 1$), where n is the number of bits in a word or classical string. An incident word can be split into multiple emergent words by means of circular bit rotation and dispersal. These multiple words are referred to as the spectrum of incident word. The emergent multiple words each have different deviation or rotational values, starting with lower deviations, then to higher ones. A spectrum of incident word consisting of emergent word(s) can be recombined to form back the incident word. This can be done by picking of the emergent words and rotating it back by the same deviation value used to rotate it. Another approach is to pick the last bit of each emergent word and the first bit of the last emergent word in a systematic fashion, starting from the word with the least deviation and progressing downwards to the word with the most deviation and concatenating them. At this point, a word or classical string is produced and this corresponds to the incident word. For example using the classical incident words 101, 1011, 01110 and 1101010, we have what is shown in Figs. 5, 6, 7, 8, 9, 10, 11 and 12.

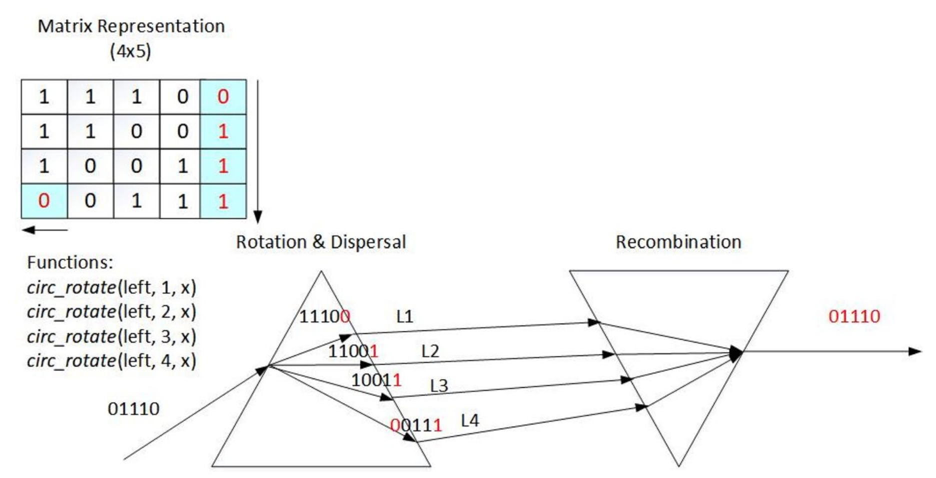


Fig. 7 Passing the classical bit string 01110 through the bit prism

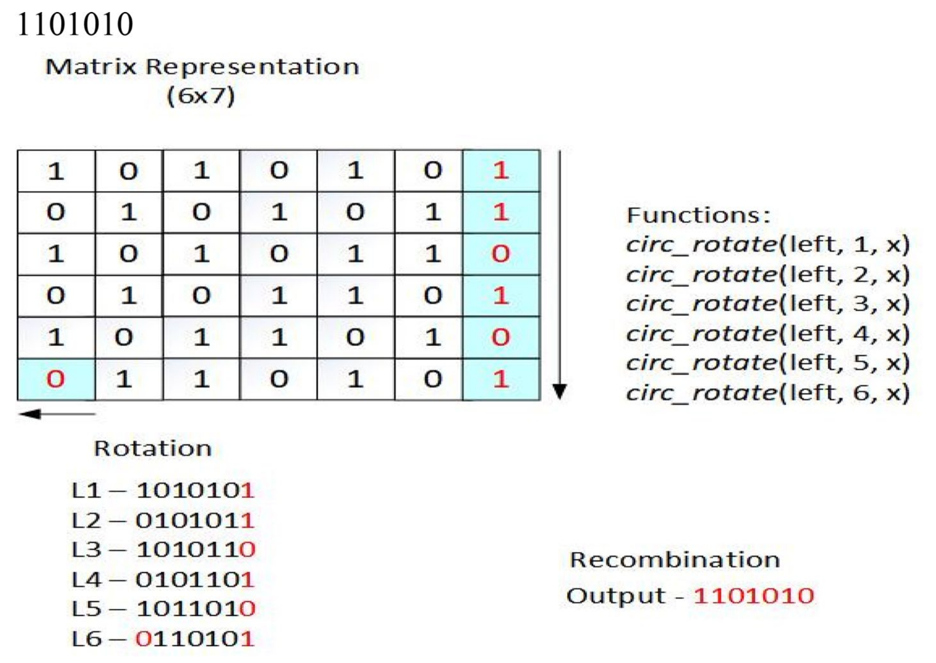


Fig. 8 Result after passing the classical bit string 1101010 through the bit prism

Left (or anticlockwise) rotation

In Fig. 5 above as an example, the incident bit string 101, splits into 2 emergent bit strings 011 and 110 by avenue of rotation. These 2 emergent bit strings recombine again to form back 101 using the principle stated in the "classical perspective section." The recombination is achieved by concatenating the last bit of each emergent bit strings plus the first bit of the last emergent bit string. This approach is for left or anticlockwise rotation.

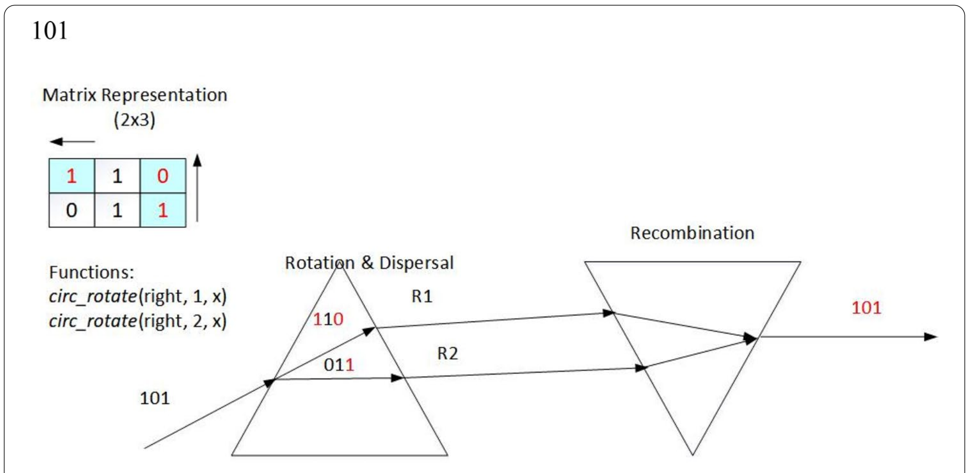

101
Matrix Representation (2x3)
1 1 0
0 1 1
Functions:
circ_rotate(right, 1, x)
circ_rotate(right, 2, x)
Rotation & Dispersal
Recombination
101
110
011
R1
R2
101

Fig. 9 Passing the classical bit string 101 through the bit prism

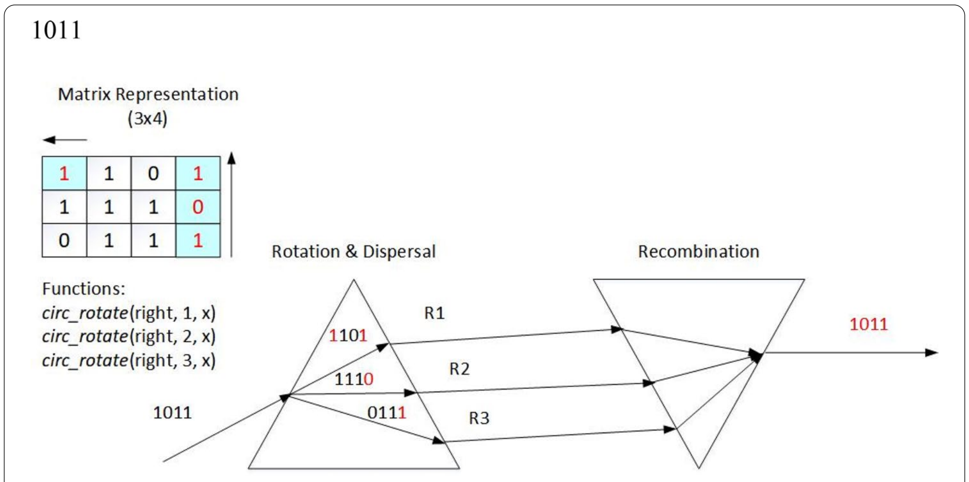

1011
Matrix Representation (3x4)
1 1 0 1
1 1 1 0
0 1 1 1
Functions:
circ_rotate(right, 1, x)
circ_rotate(right, 2, x)
circ_rotate(right, 3, x)
Rotation & Dispersal
Recombination
1011
1101
1110
0111
R1
R2
R3
1011

Fig. 10 Passing the classical bit string 1011 through the bit prism

Right (or clockwise) rotation

In Fig. 9 above as an example, the incident bit string 101, splits into 2 emergent bit strings 110 and 011 by avenue of rotation. These 2 emergent bit strings recombines again to form back 101 using the principle stated in the "classical perspective section." The recombination is achieved by concatenating the first bit of the first emergent bit string plus the last bit of each emergent bit string. This approach is for right or clockwise rotation.

Quantum–classical perspective

In the quantum–classical perspective, *qbits* (quantum bits) must be represented in multiple *cbits* (classical bits) by means of applying tensor product to produce the *product state*. The product state can be factored back into the individual state representation. The product state of *n* bits is a vector of size 2^n. For example, the quantum bit strings |101>, |1011>, and |01110> in Dirac notation give corresponding classical bits (in the form of a vector or product state) after the tensor product has been applied to all the quantum bits in vector form.

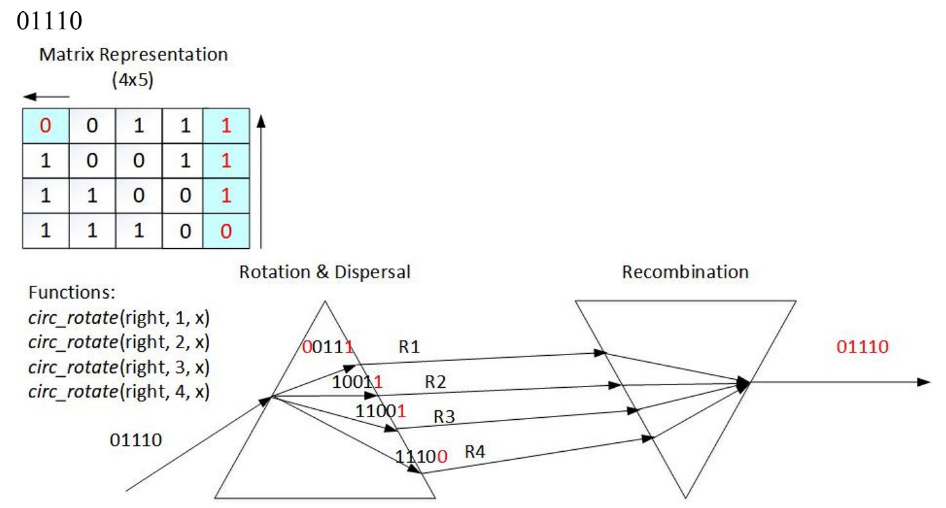


Fig. 11 Passing the classical bit string 01110 through the bit prism

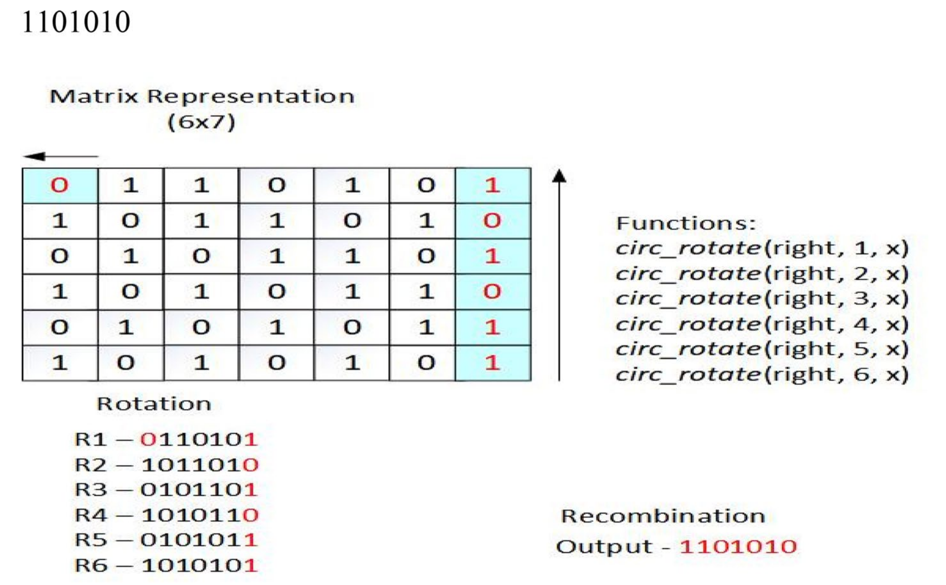


Fig. 12 Passing the classical bit string 1101010 through the bit prism

$$|101\rangle = \begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix}$$

The quantum string |101> results in a classical string 00000010.

$$|1011\rangle = \begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix}$$

The quantum string |1011> results in a classical string 0000000000000010

$$|01110\rangle = \begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}$$

$$= \begin{pmatrix}0\\1\\0\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

The quantum string |01110> results in a classical string 0000000000000100000000000 0000000.

The classical bit strings obtained from the quantum bit strings become the incident binary strings which is passed through the conceptual bit prism, just like what was done for the examples in the classical perspective. This produces a number of emergent classical strings ($n - 1$) by means of circular rotation, where n is the number of bits in the vector/classical bit string (product state). Bit(s) are subsequently extracted from the emergent classical strings to form back the incident classical bit string or vector. This resulting incident string or vector can be factored back into the individual state representation, and then subsequently back to the quantum bit representation.

Quantum perspective (QRotate)

From a quantum perspective, bit rotation is made possible by avenue of a swap gate, control bit, and data. The algorithm that is proposed can perform shift left, shift right, rotation left and rotation right on an n-qubit string. This algorithm is demonstrated at the Results section of this paper, however, the method or approach is explained below. The following five principles are employed in the quantum bit rotation process; control bit selection, bit swapping, and bit truncation. Below is the steps for the quantum bit string rotation.

1. A quantum bit string is chosen.
2. A control bit is selected and could be either |0> or |1>.
3. The control bit is placed or concatenated to the beginning or the end of the quantum bit string resulting in a new quantum bit string.
4. The swapping process begins from the position where the control bit is placed and successive bits in a progressive order is swapped till ($n - 1$) position is reached.
5. At this stage $n - 1$, quantum bit strings is obtained.
6. The last quantum bit string is the string obtained after the rotational swap is done. However, any repeated quantum string during the rotational swap is discarded or excluded otherwise it is included in the set of quantum bit stings.
7. To decode and get back the original quantum bit string, the last but one and the last bit are swapped if the control bit was placed at the beginning of the quantum bit string. If the control bit was placed at the end of the quantum bit string, then the first and second bits are swapped. Swapping of bits is done using the swap gate.
8. If the control bit was placed at the beginning of the quantum bit string, the last bit is truncated otherwise the first bit is truncated. Alternatively, the rotated bit string obtained in step 6 can be rotated by means of swapping in a reverse order till the position where the control bit was concatenated to get back the original quantum bit string.
9. At this stage, the original quantum bit string is obtained.

Alternative approach

From a quantum perspective, bit rotation is made possible by avenue of use of a swap gate only. Here, bits are swapped from position 0 of the quantum string to position n. To get back the original quantum string, bits are swapped in the reverse order from position *n* to position 0. The steps include;

1. A quantum bit is chosen
2. Bits swapping is done using swap gate from position 0 of the quantum string to position *n*.
3. At this point, a new quantum string is obtained.
4. To get back the original quantum string, bit swapping is done using swap gate from position *n* of the quantum string to position 0 (this is the reverse of step 2).

Let's consider the quantum bit strings |101>, |1011>, |01110> and |1101010> using the alternative approach. The following results is obtained.

|101>
Original quantum string: |101>
Bit swapping using swap gate from position 0 to *n*.
Swap position 0 and 1: |110>
Swap position 1 and 2: |110>
New quantum string: |110>
To get back the original quantum string.
Bit swapping using swap gate from position *n* to 0.
Swap position *n* and $n - 1$: |110>
Swap position $n - 1$ and $n - 2$: |101>
Original quantum string: |110>
|1011>
Original quantum string: |1011>
Bit swapping using swap gate from position 0 to *n*.
Swap position 0 and 1: |1011>
Swap position 1 and 2: |1101>
Swap position 2 and 3: |1101>
New quantum string: |1101>
To get back the original quantum string.
Bit swapping using swap gate from position *n* to 0.
Swap position *n* and $n - 1$: |1101>
Swap position $n - 1$ and $n - 2$: |1011>
Swap position $n - 2$ and $n - 3$: |1011>
Original quantum string: |1011>
|01110>
Original quantum string: |01110>
Bit swapping using swap gate from position 0 to *n*.
Swap position 0 and 1: |01101>
Swap position 1 and 2: |01011>
Swap position 2 and 3: |00111>
Swap position 3 and 4: |00111>
New quantum string: |00111>

To get back the original quantum string.
Bit swapping using swap gate from position n to 0.
Swap position n and $n - 1$: |00111>
Swap position $n - 1$ and $n - 2$: |01011>
Swap position $n - 2$ and $n - 3$: |01101>
Swap position $n - 3$ and $n - 4$: |01110>
Original quantum string: |01110>
|1101010>
Original quantum string: |1101010>
Bit swapping using swap gate from position 0 to n.
Swap position 0 and 1: |1101001>
Swap position 1 and 2: |1101001>
Swap position 2 and 3: |1100101>
Swap position 3 and 4: |1100101>
Swap position 4 and 5: |1010101>
Swap position 5 and 6: |0110101>
New quantum string: |0110101>
To get back the original quantum string.
Bit swapping using swap gate from position n to 0.
Swap position n and $n - 1$: |1010101>
Swap position $n - 1$ and $n - 2$: |1100101>
Swap position $n - 2$ and $n - 3$: |1100101>
Swap position $n - 3$ and $n - 4$: |1101001>
Swap position $n - 4$ and $n - 5$: |1101001>
Swap position $n - 5$ and $n - 6$: |1101010>
Original quantum string: |1101010>

The results above is shown from Figs. 21, 22, 23 and 24 in the "Quantum Implementation using jsqubits" section.

Results

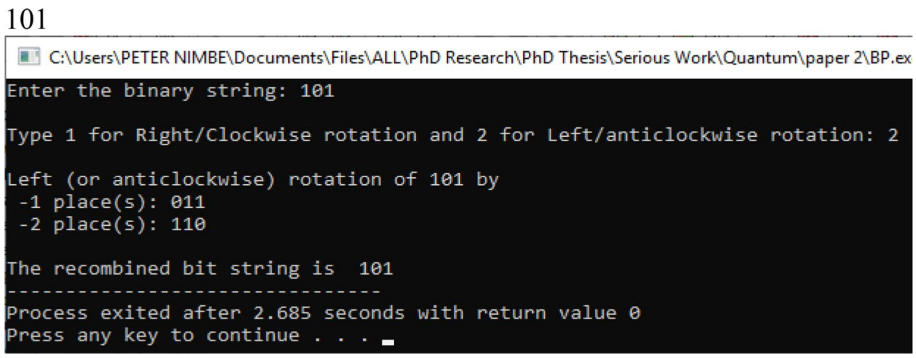
Classical implementation using C++

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

int opt;
string convertedNumber, arr[1000], s, sm;

// In-place rotates s towards left by d
void leftrotate(string &s, int d)
{
   reverse(s.begin(), s.begin()+d);
   reverse(s.begin()+d, s.end());
   reverse(s.begin(), s.end());
}

// In-place rotates s towards right by d
void rightrotate(string &s, int d)
{
  leftrotate(s, s.length()-d);
}

// This function generates all n classical bit strings
void Rotation_Dispersal(string ss)
{

    cout<<"\nType 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation:
";
    cin>>opt;

        if(opt==1){
            cout<<"\nRight (or clockwise) rotation of "<<ss <<" by ";
            for(int i=1;i<ss.size();i++){
                convertedNumber = ss;
                rightrotate(convertedNumber,i);
                arr[i] = convertedNumber;
                cout<<"\n -"<<i<<" place(s): "<<convertedNumber;
            }
        }else if (opt==2){
            cout<<"\nLeft (or anticlockwise) rotation of "<<ss <<" by ";
            for(int i=1;i<ss.size();i++){
                convertedNumber = ss;
                leftrotate(convertedNumber,i);
                arr[i] = convertedNumber;
                cout<<"\n -"<<i<<" place(s): "<<convertedNumber;
            }
        }
}

// This function generates all n classical bit strings
void Recombination_Extraction_L(string arr[], string ss)
{
cout<<endl;
    for (int i=0;i<ss.size();i++)
    {
        s = arr[i];
        sm += s[ss.size()-1];
        if (i==ss.size()-1){
            sm += s[0];
        }
    }
    cout<<"\nThe recombined bit string is "<<sm;
}

// This function generates all n classical bit strings
void Recombination_Extraction_R(string arr[], string ss)
{
cout<<endl;
int i;
    for (i=ss.size()-1;i>0;i--)
    {
        s = arr[i];
        sm += s[ss.size()-1];
    }
    if (i==0){
            sm += s[0];
    }
    cout<<"\nThe recombined bit string is "<<sm;
}


// Rotation and Recombination functions call
void Bit_Prism(string ss, string arr[])
{
    Rotation_Dispersal(ss);
    if(opt==1)
        Recombination_Extraction_R(arr, ss);
    else if (opt==2)
        Recombination_Extraction_L(arr, ss);
    else
        cout<<"Invalid Option";
}

int main()
{
    string ss;
    cout<<"Enter the binary string: ";
    cin>>ss;

    Bit_Prism(ss, arr);

    return 0;
}
```

Time Complexity: O(n)

Using the Dev-C++ compiler, the following results in Figs. 13, 14, 15, 16, 17, 18, 19 and 20 are obtained below.

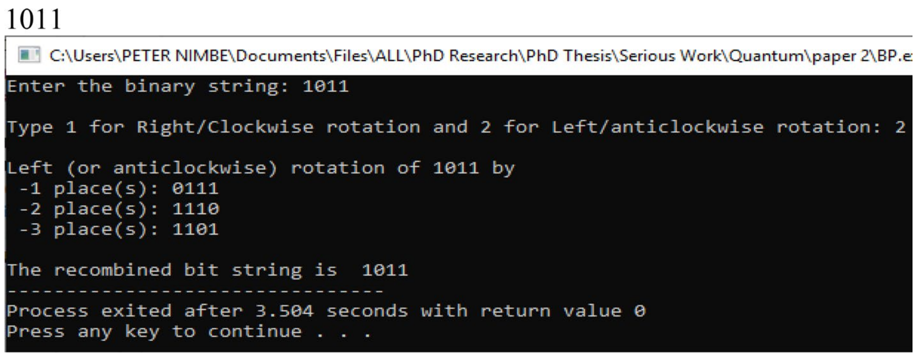
Left (or anticlockwise) rotation


101

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.ex

```
Enter the binary string: 101

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 101 by
 -1 place(s): 011
 -2 place(s): 110

The recombined bit string is  101
--------------------------------
Process exited after 2.685 seconds with return value 0
Press any key to continue . . .
```

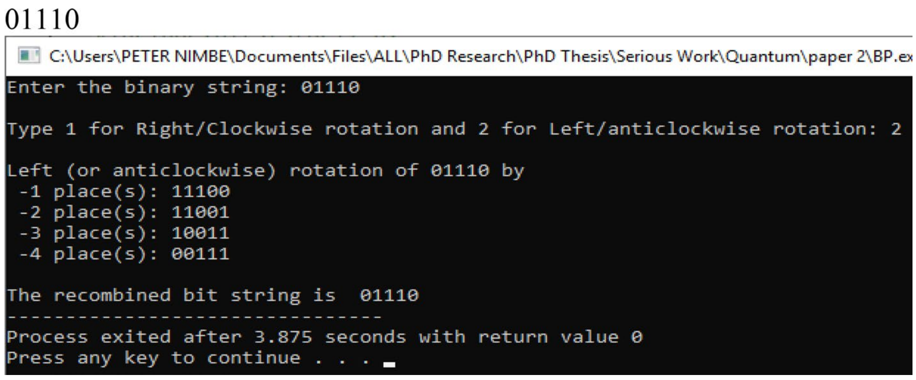
Fig. 13 Left or anticlockwise rotation of 101


1011

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.e

```
Enter the binary string: 1011

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 1011 by
 -1 place(s): 0111
 -2 place(s): 1110
 -3 place(s): 1101

The recombined bit string is  1011
--------------------------------
Process exited after 3.504 seconds with return value 0
Press any key to continue . . .
```

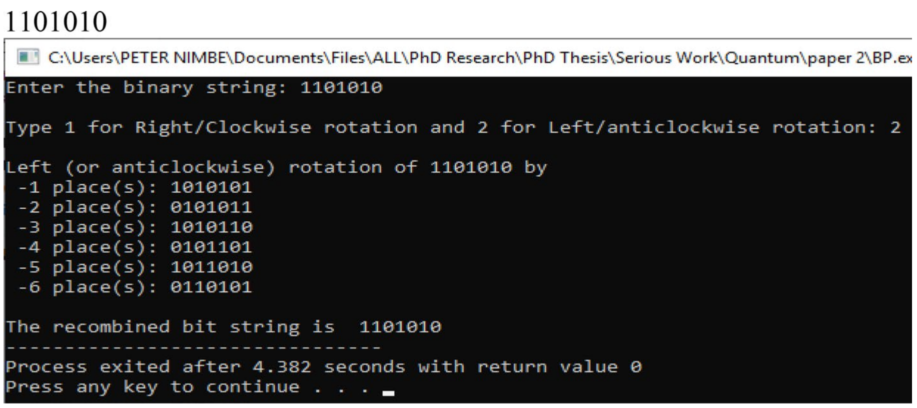
Fig. 14 Left or anticlockwise rotation of 1011


01110

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.ex

```
Enter the binary string: 01110

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 01110 by
 -1 place(s): 11100
 -2 place(s): 11001
 -3 place(s): 10011
 -4 place(s): 00111

The recombined bit string is  01110
--------------------------------
Process exited after 3.875 seconds with return value 0
Press any key to continue . . .
```

Fig. 15 Left or anticlockwise rotation of 01110

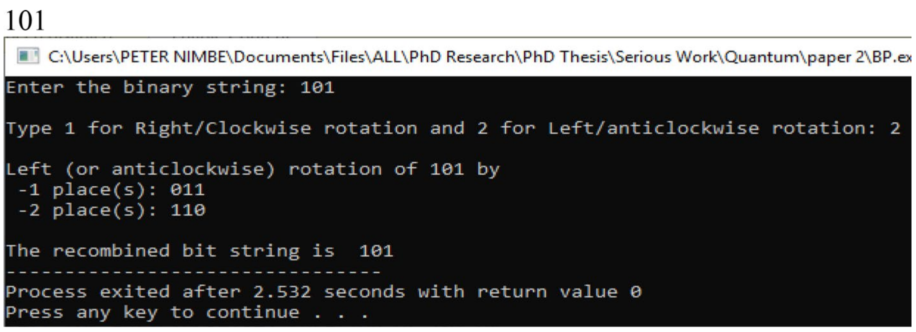

1101010

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.ex

```
Enter the binary string: 1101010

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 1101010 by
 -1 place(s): 1010101
 -2 place(s): 0101011
 -3 place(s): 1010110
 -4 place(s): 0101101
 -5 place(s): 1011010
 -6 place(s): 0110101

The recombined bit string is  1101010
--------------------------------
Process exited after 4.382 seconds with return value 0
Press any key to continue . . .
```

Fig. 16 Left or anticlockwise rotation of 1101010

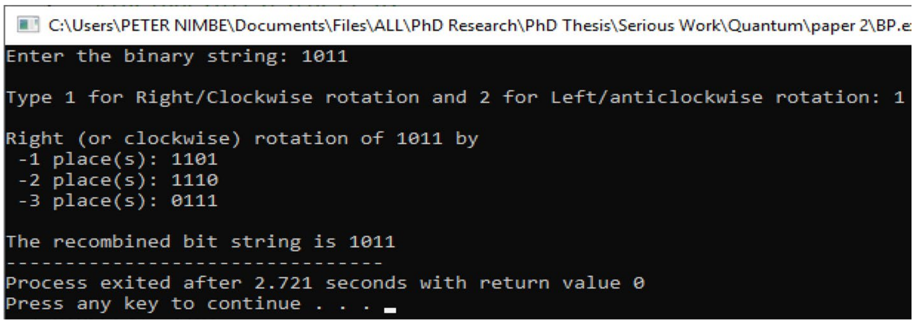
Right (or clockwise) rotation


101

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.ex

```
Enter the binary string: 101

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 101 by
 -1 place(s): 011
 -2 place(s): 110

The recombined bit string is  101
--------------------------------
Process exited after 2.532 seconds with return value 0
Press any key to continue . . .
```

Fig. 17 Right or clockwise rotation of 101

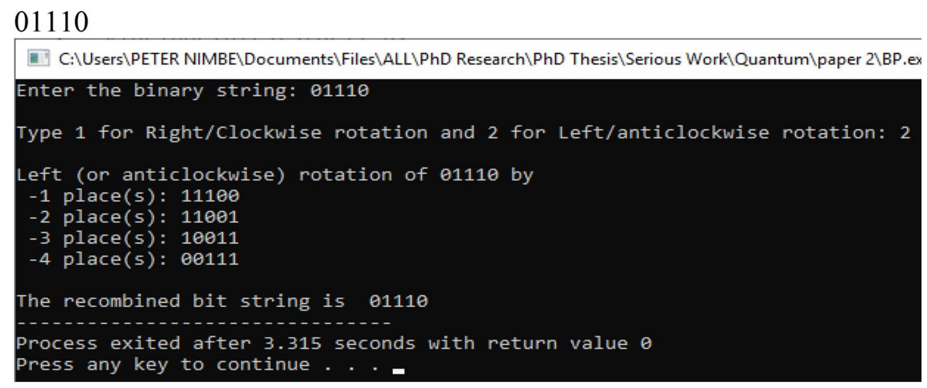

C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.e

```
Enter the binary string: 1011

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 1

Right (or clockwise) rotation of 1011 by
 -1 place(s): 1101
 -2 place(s): 1110
 -3 place(s): 0111

The recombined bit string is 1011
--------------------------------
Process exited after 2.721 seconds with return value 0
Press any key to continue . . .
```

Fig. 18 Right or clockwise rotation of 1011

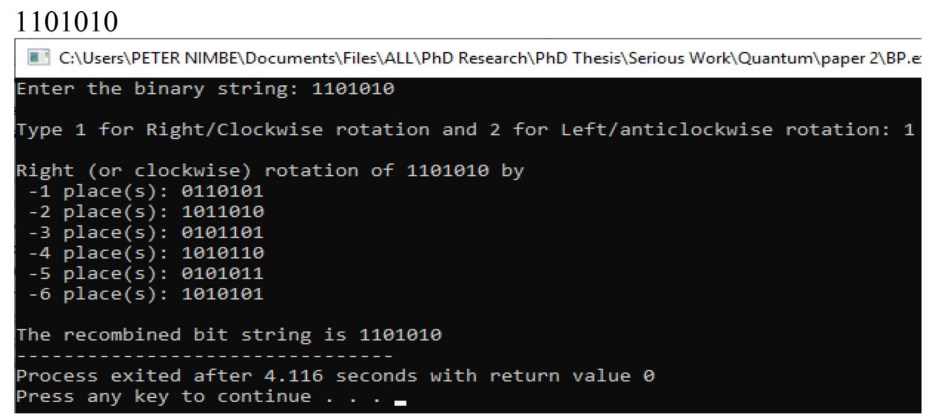
Quantum implementation using jsqubits

Using the jsqubits runner, an online quantum computer simulator, the following results are obtained below in Figs. 21, 22, 23 and 24. The time complexity depends on the size of the quantum bit string-1. The quantum swap gate **is** employed here.


01110

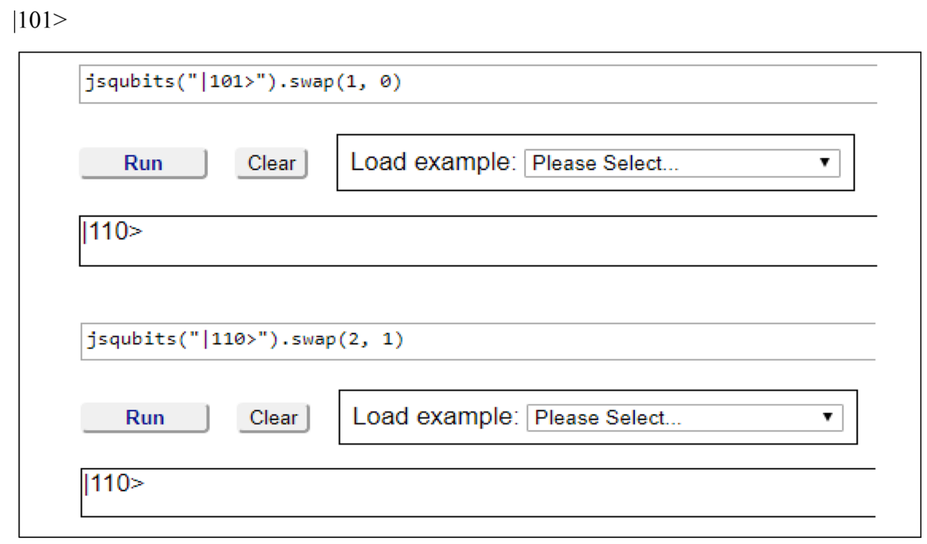
```
C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.ex
Enter the binary string: 01110

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 2

Left (or anticlockwise) rotation of 01110 by
 -1 place(s): 11100
 -2 place(s): 11001
 -3 place(s): 10011
 -4 place(s): 00111

The recombined bit string is  01110
--------------------------------
Process exited after 3.315 seconds with return value 0
Press any key to continue . . .
```

Fig. 19 Right or clockwise rotation of 01110


1101010

```
C:\Users\PETER NIMBE\Documents\Files\ALL\PhD Research\PhD Thesis\Serious Work\Quantum\paper 2\BP.e
Enter the binary string: 1101010

Type 1 for Right/Clockwise rotation and 2 for Left/anticlockwise rotation: 1

Right (or clockwise) rotation of 1101010 by
 -1 place(s): 0110101
 -2 place(s): 1011010
 -3 place(s): 0101101
 -4 place(s): 1010110
 -5 place(s): 0101011
 -6 place(s): 1010101

The recombined bit string is 1101010
--------------------------------
Process exited after 4.116 seconds with return value 0
Press any key to continue . . .
```

Fig. 20 Right or clockwise rotation of 1101010


|101>

```
jsqubits("|101>").swap(1, 0)
```

Run
Clear
Load example: Please Select...

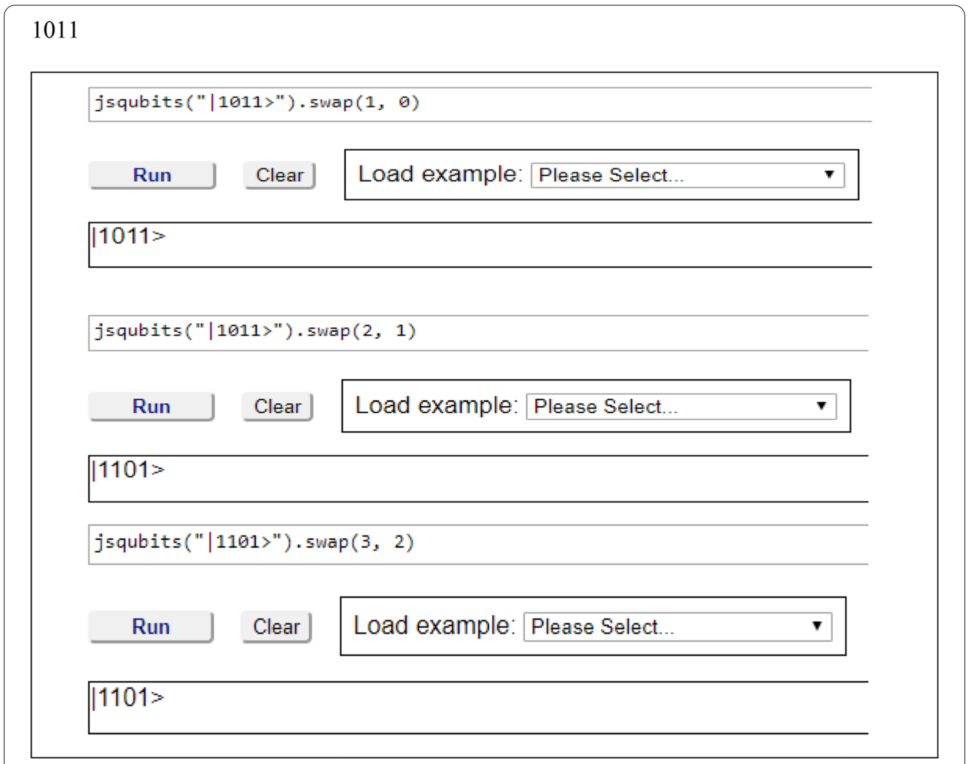
```
|110>
```

```
jsqubits("|110>").swap(2, 1)
```

Run
Clear
Load example: Please Select...

```
|110>
```

Fig. 21 Rotating quantum bit string |101 > by using a swap gate



1011

```
jsqubits("|1011>").swap(1, 0)
```

Run
Clear
Load example: Please Select...

```
|1011>
```

```
jsqubits("|1011>").swap(2, 1)
```

Run
Clear
Load example: Please Select...

```
|1101>
```

```
jsqubits("|1101>").swap(3, 2)
```

Run
Clear
Load example: Please Select...

```
|1101>
```

Fig. 22 Rotating quantum bit string |1011 > by using a swap gate

Various models for circular bit rotation using bits and qubits have been presented using bit strings or words. These models have been implemented in the C++ programming language, that is for the classical, and jsqubits for the quantum. They are both functional and effective. The execution time (seconds) of 101, 1011, 01110, 1101010 for both left and right circular rotation using the classical and quantum algorithms increases as the bit string becomes longer. However, from Figs. 25, 26 and 27 below, it is realized that the execution time for the quantum code is far smaller than the classical code for all the bit strings. This seems to suggest that the quantum algorithm is faster in terms of execution time as compared to the classical algorithm.

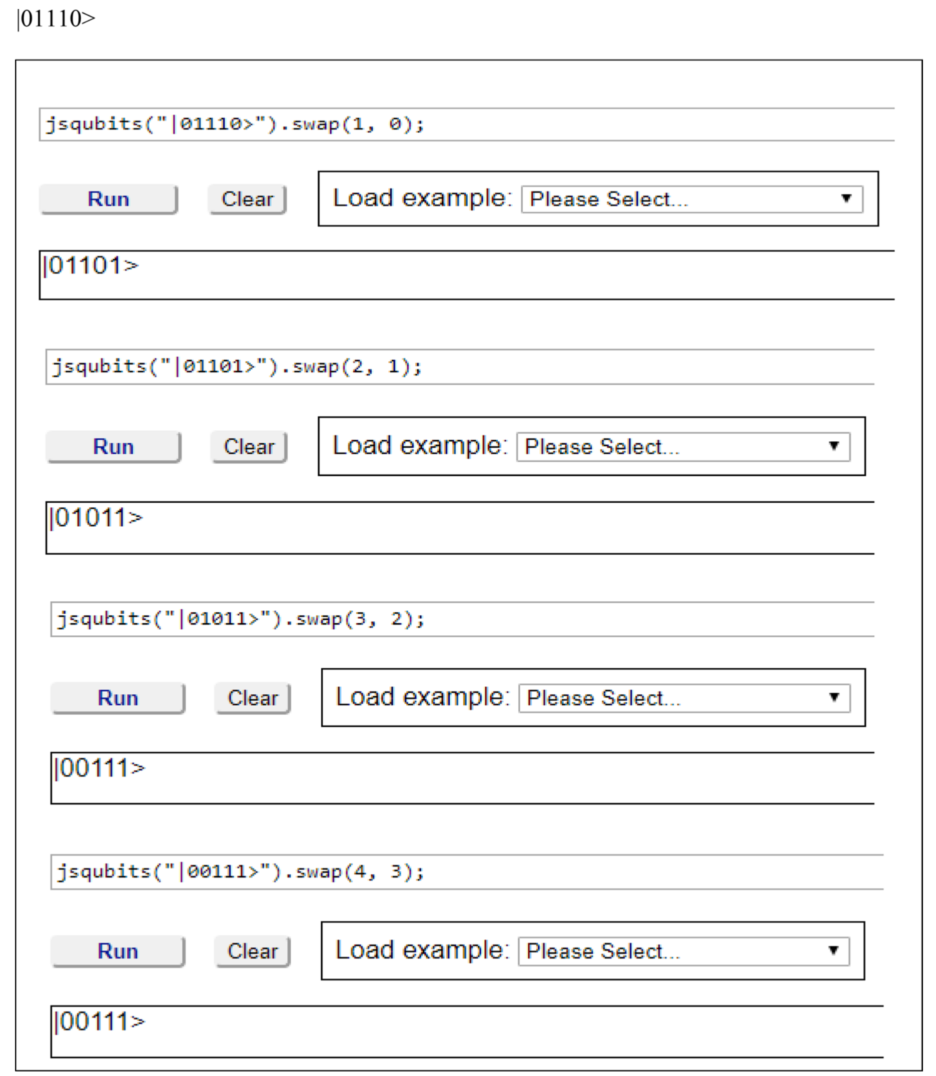
Below is the execution times shown in Figs. 25, 26 and 27.

Performance analysis of bit rotation algorithms and techniques

The running times, operation and support of some bit rotation algorithms or techniques including **CRotate** and **QRotate** (classical and quantum approach proposed in this paper) are summarized in Table 1 below.

Discussion

The classical implementation or algorithm in the results section, accepts a binary input or string, rotates the bit string circularly using bit rotation and the conceptual bit prism and emergent strings come out as the output. These emergent strings are recombined (by means of extracting bit(s) from each of the strings) to form back the original or incident string. The quantum snippets of code accepts a quantum bit string, swaps the bits


|01110>

```
jsqubits("|01110>").swap(1, 0);
```
Run
Clear
Load example: Please Select...

```
|01101>
```

```
jsqubits("|01101>").swap(2, 1);
```
Run
Clear
Load example: Please Select...

```
|01011>
```

```
jsqubits("|01011>").swap(3, 2);
```
Run
Clear
Load example: Please Select...

```
|00111>
```

```
jsqubits("|00111>").swap(4, 3);
```
Run
Clear
Load example: Please Select...

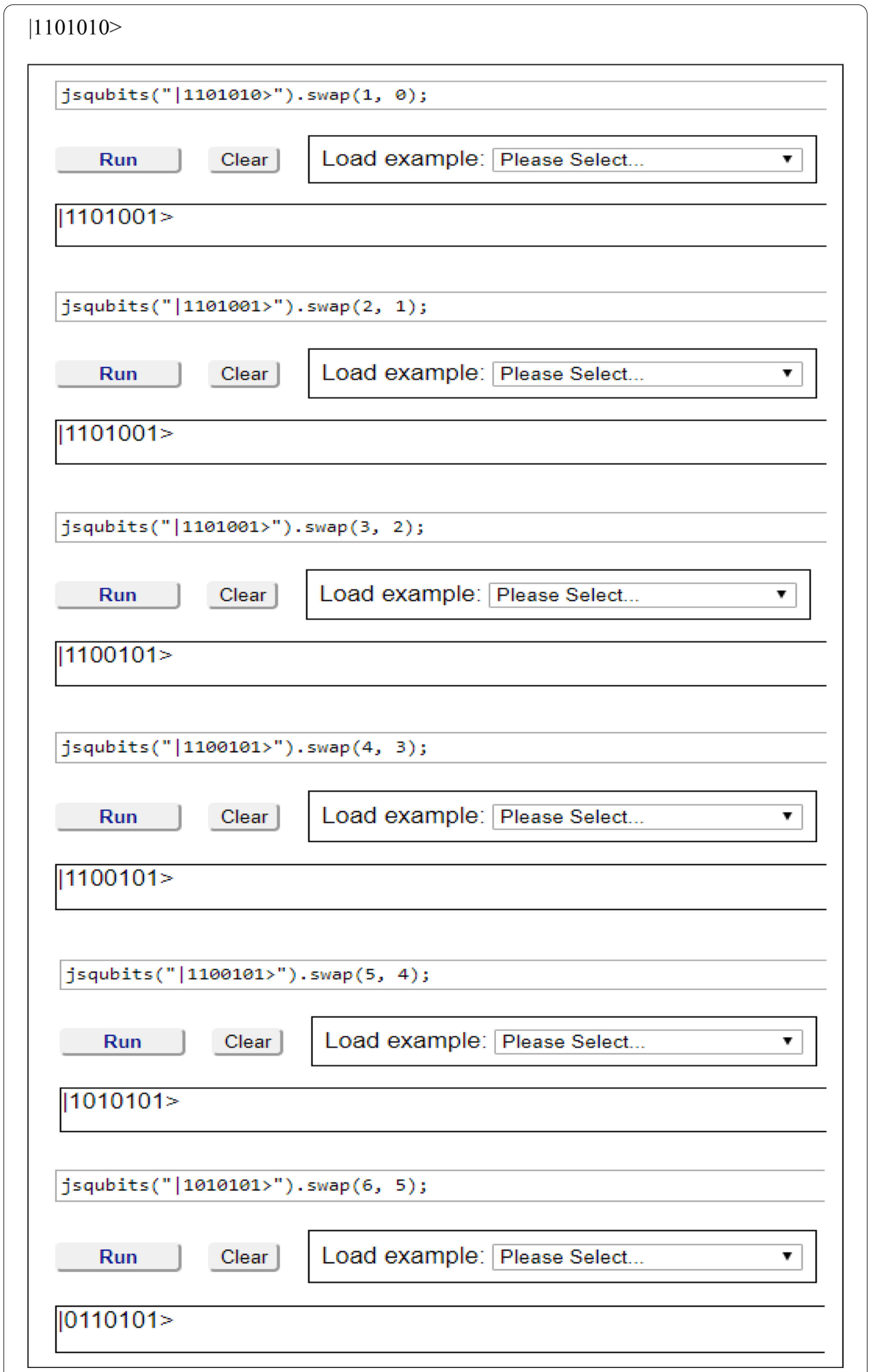
```
|00111>
```

Fig. 23 Rotating quantum bit string |01110 > by using a swap gate

in an orderly fashion from position 0 to *n*. The original quantum bit string is gotten back by swapping the bits in an orderly fashion from position *n* to 0. The algorithms are effective and able to perform the tasks above. The time complexity for the classical and quantum implementation is *O*(*n*). Some deductions were made and include:

Deduction

(a) There is an increase in the execution time as the size of the bit string increases.
(b) For the classical perspective, number of emergent bit strings equals size of incident bit string – 1, where emergent bit strings are strings obtained after the rotation and the incident bit string is the original bit string before the rotational process.
(c) For the quantum perspective, the number of bit swapping equals the size of the quantum bit string – 1.



|1101010>

```
jsqubits("|1101010>").swap(1, 0);
```

```
|1101001>
```

```
jsqubits("|1101001>").swap(2, 1);
```

```
|1101001>
```

```
jsqubits("|1101001>").swap(3, 2);
```

```
|1100101>
```

```
jsqubits("|1100101>").swap(4, 3);
```

```
|1100101>
```

```
jsqubits("|1100101>").swap(5, 4);
```

```
|1010101>
```

```
jsqubits("|1010101>").swap(6, 5);
```

```
|0110101>
```

Fig. 24 Rotating quantum bit string |1101010 > by using a swap gate

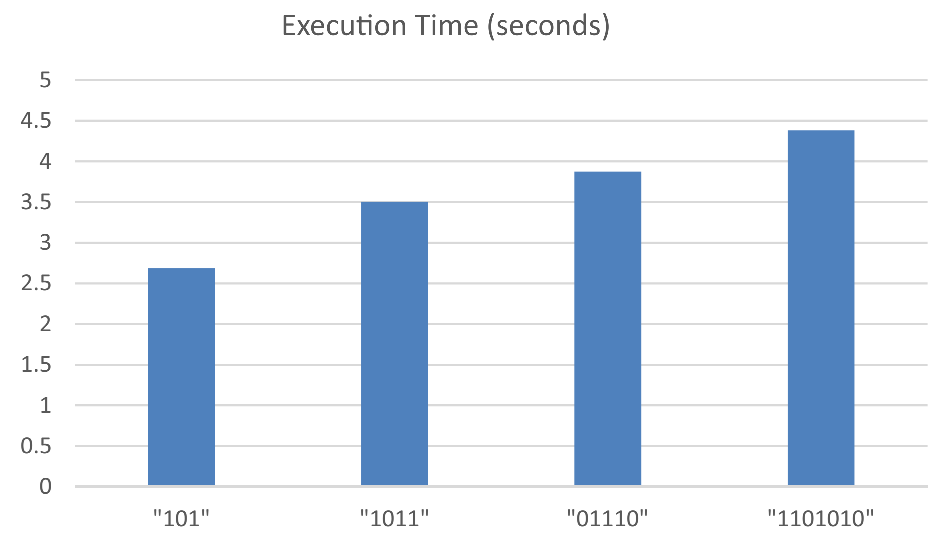


Fig. 25 Execution times of classical bit strings—left or clockwise rotation

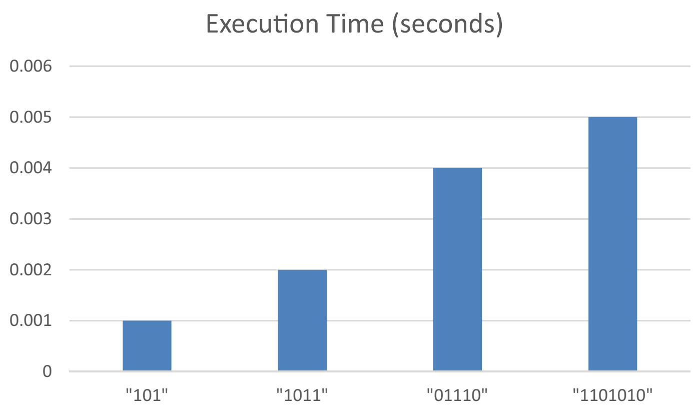


Fig. 26 Execution times of classical bit strings—right or anticlockwise rotation

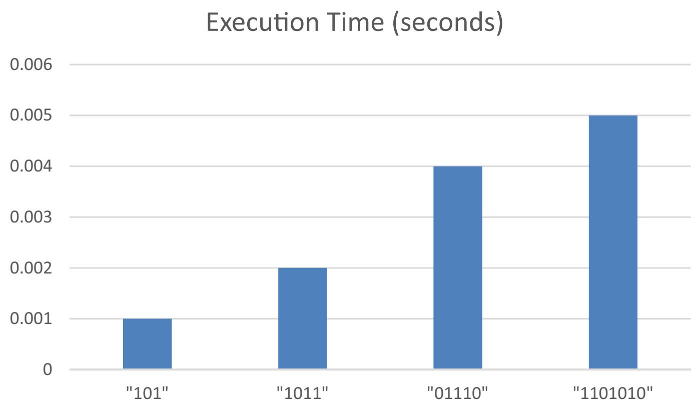


Fig. 27 Execution times of quantum bit strings—rotation by bit swapping

Table 1 Running time of bit rotation algorithms, techniques and circuits

	Algorithm/technique	Running time	Operation and support
1	Circular shift	$O(n)$ upwards	Permutation σ of the n entries in a tuple Involves arithmetic operations
2	Arithmetic shift	$O(1)$–$O(n)$	–
3	Logical shift	$O(1)$–$O(n)$	Does not preserve a number's sign bit Every bit in the operand is simply moved a given number of bit positions Vacant bit positions are filled, usually with zeros or one's
4	Faster bit rotation (Improved version of the circular shift)	$O(n)$	Permutation σ of the n entries in a tuple No arithmetic operations, only bit manipulations
5	Bit shift and bit rotation algorithm with Scilab implementation	$O(\text{NB-}n)$	–
6	A highly efficient reconfigurable rotation unit based on an inverse butterfly network	Not applicable because it is a circuit	64-bit Single Instruction Single Data (SISD) MultiMedia eXtensions/Streaming SIMD Extensions (MMX/SSE) instructions
7	sb-rotate-byte		
8	Fast MSB and LSB Rotate method	–	8 bit data
9	CORDIC algorithm	–	Operand word-length of 54 bits
10	CORDIC II	–	–
11	VHDL 16-bit shifter	Circuit	16 bit data
12	Power mux based on dynamic barrel shifter using footed diode domino logic	Circuit	8 bit
13	8 bit barrel shifter using 8×1 multiplexer	Circuit	8 bit
14	Quantum shift register	Circuit	Qubit data
15	CRotate and QRotate	$O(n)$	*CRotate*: 2^{32} bytes or 2^{64} bytes determined by the amount of memory that the program can access. The size of the array can be increased from 1000 as is in the case of the CRotate to up to 2^{32} or 2^{64} *QRotate*: n qubit

Conclusion

Based on the galloping rate of the development and implementation of classical and quantum models, this paper also seeks to propose a model for circular bit rotation using a conceptual bit prism drawn from the inspiration of white light going through a glass prism. Existing techniques in circular bit rotation have some challenge with respect to the fact that it does not have an end and keeps going on in cycles or in a circle and also produces some form of repeating bit strings at some point in the rotational process. This necessitated the need to use the bit prism concept in this work to help address this challenge and from the method and results above, it has been addressed incorporating a high level of strictness in the rotational process especially for the classical aspect. The quantum bit rotation in this paper, however, uses a bit swapping technique by avenue of a quantum swap gate made available in jsqubits.

Future works will be to use this concept for cryptographic purposes.

Abbreviations

CRotate: circular rotate; QRotate: quantum rotate; XOR: exclusive OR; LSL: logical shift left; LSR: logical shift right; LSB: least significant bit; MSB: most significant bit; ROR: rotate right; ROL: rotate left; CPU: central processing unit; SLA: shift left arithmetic; SRA: shift right arithmetic; SRL: shift left logical; SRL: shift right logical; RL: rotate left; RR: rotate right; GDI: gate diffusion input; qubit: quantum Bit; CNOT: controlled not; SDLC: system development life cycle; jsqubits: JavaScript quantum computer simulator; circ_rotate: circular rotate.

Acknowledgements

First and foremost, we would like to thank our Father for the inspiration to write this manuscript. This manuscript or article would never have been possible without the support and guidance of various people at the University of Energy and Natural Resources, Sunyani, Ghana. We also wish to express our sincere thanks to Prosper Kandabongee Yeng at Norwegian University of Science and Technology for his vital inputs and proofreading.

Authors' contributions

Study concept and design, and acquisition of literature were done by PN. Analysis and some portions of the method were done by BAW. Finally, AFA helped in the interpretation of the results as well as conducted a critical revision of the paper. All authors have read and approved the manuscript.

Funding

Not applicable.

Availability of data and materials

Not applicable.

Competing interests

On behalf of all authors, the corresponding author states that there is no competing interest.

Received: 10 September 2020 Accepted: 15 February 2021
Published online: 27 February 2021

References

1. Dodge NB (2012) Shift and rotate instruction. http://www.utdallas.edu/~dodge/EE2310/lec14.pdf. Accessed 25 Jan 2020
2. Yeng PK, Panford JK, Hayfron-Acquah JB, Twum F (2016) An efficient symmetric cipher algorithm for data encryption. Int Res J Eng Technol 03(05):8–9
3. Damgard IB, Kundsen LR (1998) Two-key triple encryption. J Cryptol 11(3):209–218
4. Black J, Rogaway P (2005) CBC MACs for arbitrary-length messages: the three-key constructions. J Cryptol 18:111–131
5. Harris SL (2013) Digital building blocks. Digit Des Comput Arch . https://doi.org/10.1016/B978-0-12-394424-5.00005-7
6. Harris SL, Harris DM (2016) Shifters and rotators, digital design and computer architecture, ARM edition. Morgan Kaufmann, pp 251–253. ISBN: 978-0-12-800056-4
7. Brown B (2013) Shifting and shifters. Computer Science Department Southern Polytechnic State University. http://ksuweb.kennesaw.edu/faculty/rbrow211/papers/shifter.pdf. Accessed 25 Jan 2020
8. Renesas Electronics Corporation (2008) Fast MSB and LSB rotate method for 8 bit data. https://www.renesas.com/us/en/doc/products/mpumcu/apn/001/reg05b0008_h8slpap.pdf. Accessed 25 Jan 2020
9. Hilewitz Y, Lee RB (2007) Performing advanced bit manipulations efficiently in general-purpose processors. In: 18th IEEE symposium on computer arithmetic (ARITH '07), Montepellier, pp 251–260
10. Guston D (2014) Adding standard circular shift operators for computer integers. Programming language C++, evolution working group. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3990.pdf. Accessed 25 Jan 2020
11. Bennett CH (1973) Logical reversibility of computation. IBM J Res Dev 17(6):525–532
12. Feynman RP (1986) Quantum mechanical computers. Found Phys 16(6):507–531
13. Parhami B (2006) Fault tolerant reversible circuits. Asilomar Conf Signals Syst Comput 2006:1726–1729
14. Fredkin E, Toffoli T (1982) Conservative logic. Int J Theor Phys 21:219–253
15. Muwafi JA, Fettweis G, Neff HW (1999) Circuit for rotating, left shifting, or right shifting bits. United States Patent. Patent No: 5,978,822
16. Bharathesh PN, Manju D (2019) A novel low power MUX based dynamic barrel shifter using footed diode domino logic. Int J Innov Technol Exp Eng 8(6S3):307–311
17. Priyanka AP, Mehra R (2016) Design and performance analysis of barrel shifter using 45nm technology. IOSR J VLSI Signal Process 6(3):38–44
18. Mitra SK, Chowdhury AR (2015) Optimized logarithmic barrel shifter in reversible logic synthesis. In: 28th International conference on VLSI design, Bangalore, India, pp 441–446
19. Aarthi R, Kavitha S (2017) Image encryption using binary bit plane and rotation method for an image security. Int J Eng Dev Res 5(2):1–6
20. Minematsu K. (2013) A short universal hash function from bit rotation, and applications to blockcipher modes. In: Susilo W, Reyhanitabar R (eds) Provable security. ProvSec 2013. Lecture notes in computer science, vol 8209. Springer, Berlin. https://doi.org/https://doi.org/10.1007/978-3-642-41227-1_13
21. Shah S, Khan L, Maurya VK (2018) Designing of low power GDI based 8-bit barrel shifter. J Emerg Technol Innov Res 5(9):397–400

22. Naveen Kumar SK, Sharath Kumar HS, Panduranga HT (2012) Encryption approach for images using bits rotation reversal and extended hill cipher techniques. Int J Comput Appl (0975-8887) 59(16):10–14
23. Subong RA, Fajardo AC, Kim YJ (2018) Adaptive bit rotation and inversion scoring: a novel approach to lsb image steganography. In: IEEE 10th international conference on humanoid. nanotechnology. information technology. Communication and control. Environment and management (HNICEM), Baguio City, Philippines, pp 1–6
24. Perkowski M, Lukac M, Kerntopf P, Pivtoraiko M, Folgheraiter M, Choi YW, Kim J, Lee D, Hwangbo W, Kim H (2003) A hierarchical approach to computer-aided design of quantum circuits. Electr Comput Eng Fac Publ Present 228:201–209
25. Microsoft (2017) The Qubit. https://docs.microsoft.com/en-us/quantum/concepts/the-qubit Accessed 25 Jan 2020
26. Lee J, Lee EK (2001) Quantum shift register. Department of Chemistry, School of Molecular Science. Korea Advanced Institute of Science and Technology
27. Quantum Inspire (2018) Rotation operators. https://www.quantum-inspire.com/kbase/rotation-operators/. Accessed 25 Jan 2020
28. Foell https://www.researchgate.net/profile/Charles_Foell_IIIC.A III, (2016). Luminescent properties of Pb-based (PbX) colloidal quantum dots (CQDs) in vacuum, on silicon and integrated with a silicon-on-insulator (SOI) photonic integrated circuit (PIC). PhD Dissertation, University of British Columbia Library
29. Weik MH (2000) circular shift. In: Computer science and communications dictionary. Springer, Boston. https://doi.org/10.1007/1-4020-0613-6_2654
30. Veerendra (2018) CBSE sample papers. CBSE Class 10 Science practical skills-refraction through prism. https://www.aplustopper.com/cbse-class-10-science-lab-manual-refraction-prism/. Accessed 25 Jan 2020
31. Lowe C (2017) Sciencing, science projects with a prism. https://sciencing.com/science-projects-prism-7976707.html. Accessed 25 Jan 2020