

Efficient Circuit Design of Reversible Square

H.V. Jayashree¹, Himanshu Thapliyal²(✉), and Vinod Kumar Agrawal³

¹ Department of Electronics and Communication Engineering,
PES Institute of Technology, Bengaluru 560085, Karnataka, India
jayashreehv@pes.edu

² Department of Electrical and Computer Engineering,
University of Kentucky, Lexington, KY 40506, USA
hthapliyal@uky.edu

³ Department of Information Science and Engineering, PES Institute of Technology,
Bengaluru 560085, Karnataka, India
vk.agrawal@pes.edu

Abstract. In the midst of emerging technology, reversible computing is promising due to its application in the field of quantum computing. The computing hardware plays a significant role in digital signal processing (DSP) and multimedia application; one such major computing hardware is multiplier. It is a practice to choose multiplier to compute square of an operand. Multiplication hardware requires more elementary computations which leads to performance degradation in terms of reversible performance metrics like quantum cost, garbage outputs, and ancilla inputs. Ancilla inputs and garbage outputs are overhead bits in a reversible circuit. Reversible quantum computers of many qubits are extremely difficult to realize, thus we propose garbageless circuit design for reversible square computation. The proposed design methodology is based on recursion. Recursion technique is adapted from Karatsuba's recursive method to compute square of an operand; we designed inverse computation units to retrieve the inputs and obtain garbageless circuit. On comparing proposed circuit design with existing reversible square designs and Karatsuba multiplier design, we observed that our work improves number of input lines which includes data lines and ancilla lines.

Keywords: Garbageless square · Reversible circuit · Ancilla inputs

1 Introduction

Reversible logic is emerging as a promising computing paradigm with applications in ultra-low power green computing and emerging nano technologies such as quantum computing, quantum dot cellular automata (QCA), optical computing, etc. Reversible circuits are similar to conventional logic circuits except that they are built from reversible gates [2]. In reversible gates, there is a unique, one-to-one mapping between the inputs and outputs, not the case with conventional logic. The most promising applications of reversible logic lies in quantum computing since quantum circuit is a network of quantum gates. Each gate performs

unitary operation on qubits which represents elementary unit of information. Qubits corresponds to conventional binary bits 0 and 1. Qubits are allowed to be in superposition of both the states 0 and 1. These unitary operations are reversible, hence quantum circuits are built using reversible logic gates.

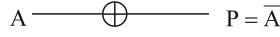
While designing reversible circuit, several performance measuring parameters need to be considered such as quantum cost, garbage outputs, and ancilla input bits. The quantum cost of a reversible circuit is the number of 1×1 and 2×2 reversible gates used in its design; it can be considered equivalent to number of transistors needed in a conventional CMOS design. The garbage output refers to the output which exists in the circuit to maintain one-to-one mapping but is not a primary or a useful output. The constant inputs (0 or 1) are called ancilla bits which are used in reversible circuits for storing intermediate values during computation. Reversible computers including quantum computers of many bits are extremely difficult to realize, so the number of ancilla inputs and the garbage outputs in the reversible circuits need to be minimized. While designing reversible circuits, one needs to optimize these parameters to improve the footprint of the overall design. Arithmetic units are the key components of computing systems. Therefore, researchers have concentrated their efforts towards the design of reversible quantum adders [15, 19, 22], [4, 6, 13], [3, 5, 16, 17], multipliers [9, 23], dividers [7, 21], etc.

Among arithmetic circuits, multiplier circuits play a major role to improve the performance of data processing in a processor. Squaring is the most commonly used function in division (Newton Raphson division and Taylor series expansion), roots, or reciprocals [10, 12]. Squaring also finds its applications in DSP applications such as Euclidean distance computation and exponent calculation in cryptography. For powering functions like squares and cubes, a reversible circuitry of multiplier is not the most efficient solution as it results in redundant partial products and extra addition circuitry that will result in enormous overhead in terms of quantum cost, ancilla bits, and garbage outputs. As ancilla inputs and garbage outputs are overhead bits in a reversible circuit and reversible quantum computers of many qubits are extremely difficult to realize, we propose a garbageless reversible circuit design for square computation based on recursion. Recursion technique is adapted from Karatsuba's recursive method to compute square of an operand. Inverse computation units are designed to retrieve the inputs and obtain garbageless circuit. Further, we compared proposed circuit design with existing reversible square designs and observed that our work improves number of input lines which includes data lines and ancilla lines for data width >8 .

The paper is organized as follows: Sect. 2 presents the background on reversible logic gates; Sects. 3 and 4 elaborate on the design and comparative analysis of proposed reversible circuitry for square respectively; Sect. 5 presents discussion and conclusion.

2 Background on Reversible Logic Gates

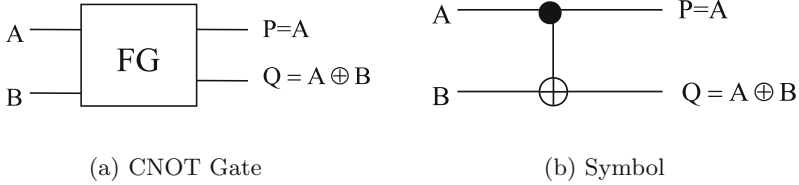
The reversible gates used in this work are discussed in this section. Each reversible gate has a cost associated with it called quantum cost. The quantum cost of a

**Fig. 1.** NOT gate

reversible gate is the number of 1×1 and 2×2 reversible gates or quantum logic gates [14] required in designing it. The quantum cost of all reversible 1×1 and 2×2 gates is taken as unity. NOT gate shown in the Fig. 1 is an example of 1×1 reversible gate.

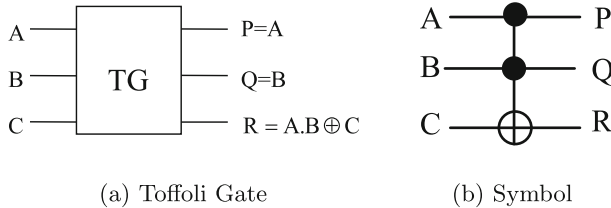
2.1 CNOT or Feynman Gate (FG)

CNOT gate is a 2×2 gate with inputs A and B , where A is the control line and B is the target line. The outputs have two lines; the control line directly passes A to output line while the target line passes transformed B as $B \rightarrow A \oplus B$. Figure 2 shows the block diagram and symbol of CNOT gate.

**Fig. 2.** CNOT gate, and its quantum representation

2.2 Toffoli Gate

Figure 3 shows a Toffoli gate and its symbolic representation. It is a 3×3 gate with inputs (A, B, C) and outputs $P = A$, $Q = B$, $R = AB \oplus C$. Toffoli gate is one of the most popular reversible gates and has quantum cost of 5. The quantum cost of Toffoli gate is 5 as it needs five 2×2 quantum gates for its implementation.

**Fig. 3.** Toffoli gate and its quantum representation

2.3 Peres Gate

Figure 4 shows the Peres gate and its symbolic representation. It is a 3×3 reversible gate having inputs (A, B, C) and outputs $P = A$, $Q = A \oplus B$, $R = AB \oplus C$. The quantum cost of Peres gate is 4 as it requires four 2×2 reversible gates in its design.

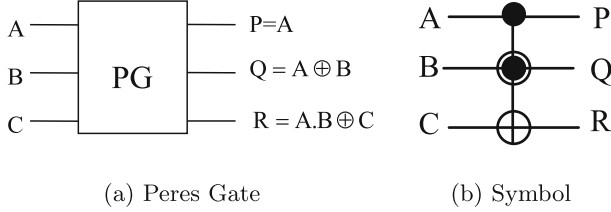


Fig. 4. Peres gate and its quantum representation

2.4 Reversible Full Adder (RFA)

Figure 5 shows the quantum diagram and symbol of reversible full adder [18]. It is a 4×4 reversible block. We alter the inputs given to the RFA block as $(C, B, A, 0)$ to obtain S as Carry out ($Cout$) and R as Sum expression of reversible full adder. The quantum cost of RFA is 6 as it requires six 2×2 quantum gates. In this work, 1 bit reversible full adder is being used in the design of 3 bit reversible square circuitry (Fig. 7(b)).

3 Proposed Dedicated Reversible Square Circuit

In this work, we propose a recursive method based design of n bit square circuit. Dedicated design of square circuit is presented in [1, 8] which proved to be better than the existing efficient multipliers in the literature. By applying equivalence relation, we obtained partial product reduction as shown in [8]. The motivation for the proposed design is derived from Karatsuba's algorithm [11]. We initially illustrate the method with a recursive algorithm and then proceed with the general architecture.

3.1 Recursive Square Computation Method

Let a be an n bit number. We split a into aL and aR respectively. aL represents the number coming from first $\lceil n/2 \rceil$ bits and aR represents the second $\lceil n/2 \rceil$ bits; therefore, we have $a = aL * 2^{n/2} + aR$. We compute square of n bit number a as:

$$\begin{aligned}
 a^2 &= (aL * 2^{n/2} + aR) (aL * 2^{n/2} + aR) \\
 &= aL^2 * 2^n + (aL * aR + aR * aL) 2^{n/2} + aR^2
 \end{aligned} \tag{1}$$

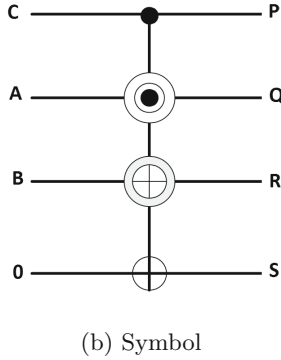
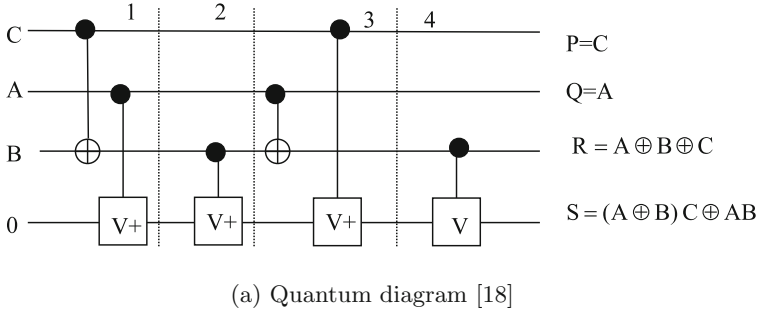


Fig. 5. Reversible full adder and its symbol

The second term is computed as below:

$$(aL * aR + aR * aL) = (aL + aR)(aL + aR) - (aL^2) - (aR^2) \quad (2)$$

The squaring is done recursively using the similar kind of splitting until we arrive at a constant number of bits. We choose this constant to be 2 and 3 so that we are guaranteed that the number of bits decreases in every recursive step. At this point, we directly square the number.

```

1: procedure SQUARE( $a$ )                                ▷ compute square of  $a$ 
2:   input:  $n$  bit unsigned number  $a$ 
3:   output:  $2n$  bit unsigned number                      ▷  $a^2$ 
4:   if  $a \leq 3$  then return  $a^2$ ;
5:   Let  $aL$  and  $aR$  be the leftmost  $\lceil n/2 \rceil$  and rightmost  $\lceil n/2 \rceil$  of input  $a$ 
   respectively
6:    $P1 \leftarrow SQUARE(aL)$ ;
7:    $P2 \leftarrow SQUARE(aR)$ ;
8:    $P'_3 \leftarrow ADD(aL, aR)$ ;
9:    $P_3 \leftarrow SQUARE(P'_3)$ ;
10:  return  $P1 * 2^n + (P3 - (P1 + P2)) * 2^{n/2} + P2$ ;
11: end procedure

```

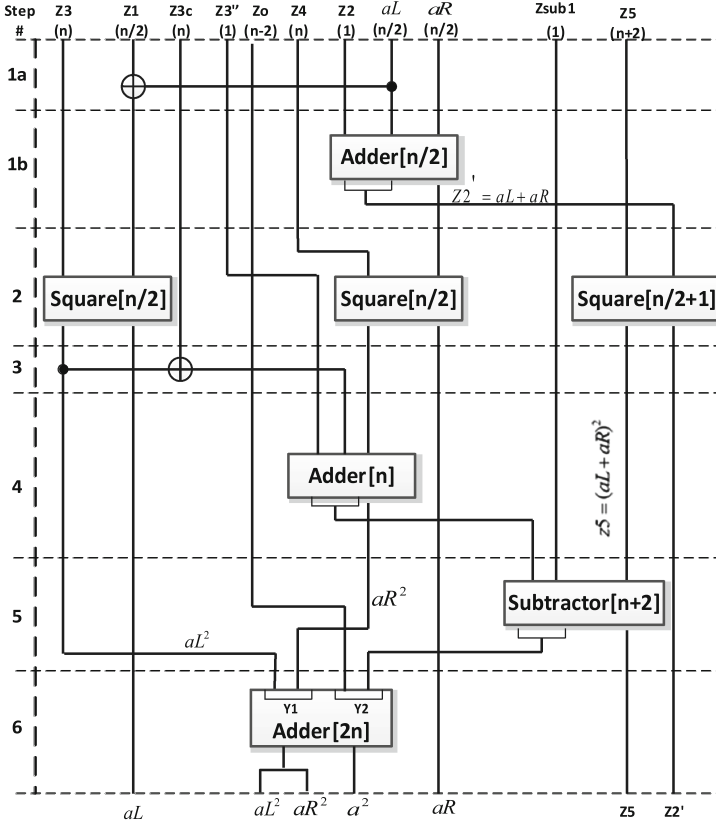


Fig. 6. Recursive square design:computation steps

3.2 Proposed Design Methodology

We present the design of dedicated square computation unit without garbage outputs. The proposed design comprises of few computational blocks for addition and subtraction which also produces zero garbage outputs [19,20]. Consider an n bit number $a[n-1:0] \in \{0,1\}^n$ represents a binary number. If the number of bits in a is even, then we split a into aL coming from leftmost $[n/2]$ bits and aR from rightmost $[n/2]$ bits. If the number of bits is not even, we split aL and aR into two parts each with $[n+1]/2$ and $[n-1]/2$ bits respectively and rest of the operations are modified appropriately. For the design methodology illustration, we consider n as even and proceed. Consider the n bit number a_i split into aR_i and aL_i , stored at locations AR_i where $0 \leq i \leq n/2 - 1$ and AL_i where $n/2 \leq i \leq n-1$ respectively. Further, consider that memory location z is initialized with ancilla 0 bits. Ancilla locations used in each design step vary in size and name, hence it is defined in the corresponding design step. At the end of the computation, the ancilla locations will hold a^2 , while the values aL and aR are restored.

We present the design steps in two phases. Phase 1 comprises of computation steps as shown in Fig. 6 to compute square of an n bit operand. Phase 2 comprises of decomputation steps and are required to remove garbage bits.

Phase 1: Computation Steps

1. Step 1: For $0 \leq i \leq n/2 - 1$

- a: At pair of locations AL and $z1$ (here $z1$ represents ancilla 0 bits, where $z1[n/2-1 : 0] \in \{0\}^n$) apply $n/2$ CNOT gate array such that location AL will maintain the same value whereas $z1$ will hold the copy of aL value. The transformation of $z1$ and AL location values are shown below. Here, $*z1$ represents the value present in the location $z1$.

$$\bigotimes_{i=0}^{n/2-1} |*z1_i \oplus aL_i\rangle |aL_i\rangle \quad (3)$$

This step needs $n/2$ ancilla input bits.

- b: At locations AL , AR , and $z2$, apply $n/2$ bit adder. $z2$ initially will hold ancilla 0 bit. AR will maintain the same value whereas AL transforms to s_i ; $z2$ transforms to c_i state. s_i and c_i indicate the sum and carry bits generated during addition operation irrespective of the design steps. The locations $z2$ and AL are referred as $z2'$ which holds c_i bit as MSB and s_i bits as LSB bits.

$$z2' = \begin{cases} s_i & \text{for } 0 \leq i \leq n/2 - 1 \\ c_{n/2} & \text{for } i = n/2 \end{cases} \quad (4)$$

$$s_i = \begin{cases} aL_i \oplus aR_i \oplus c_i & \text{for } 0 \leq i \leq n/2 - 1 \\ c_{n/2} & \text{for } i = n/2 \end{cases} \quad (5)$$

where c_i is the carry bit and is defined as:

$$c_i = \begin{cases} 0 & i = 0 \\ (aL_{i-1} \oplus aR_{i-1})c_{i-1} \oplus aL_{i-1}aR_{i-1} & 1 \leq i \leq n/2 \end{cases} \quad (6)$$

This step needs 1 ancilla input bit.

Steps 1a and 1b are executed sequentially.

2. Step 2: Apply values present at locations $z1$, AR , and $z2'$ to square computation units which operates on the operand width $n/2$, $n/2$, and $n/2 + 1$ respectively. Each square computation block executes recursively until operand width boils down to 3 (go to step 2 until $n = 2$ or 3). Each square computation block takes second operand as ancilla inputs stored at locations $z3$, $z4$, and $z5$ respectively, where width of $z3$, $z4$, and $z5$ are n , n , and $n + 2$ bits respectively. At the end of recursive computation $z1$, AR and $z2'$ will maintain its value as shown in Step 1a and Step 1b respectively. The ancilla

locations $z3$, $z4$, and $z5$ will hold the computation result as shown below. $*z3$, $*z4$, and $*z5$ represent the values at the locations $z3$, $z4$, and $z5$ respectively. Here, k and l indicate the bit position and n is the number of bits used to represent the operand value. The below shown equations need to be computed with $n = 2$ (for $*z3$, $*z4$), $n = 3$ (for $*z5$); these are computed when recursive call reaches $n = 2$ or 3 .

$$*z3 = \bigotimes_{k=0}^{n-1} |aL_k\rangle 2^{2k} \oplus \bigotimes_{k=0}^{n-2} \bigotimes_{l=k+1}^{n-1} |aL_k \cdot aL_l\rangle 2^{k+l+1} \quad (7)$$

$$*z4 = \bigotimes_{k=0}^{n-1} |aR_k\rangle 2^{2k} \oplus \bigotimes_{k=0}^{n-2} \bigotimes_{l=k+1}^{n-1} |aR_k \cdot aR_l\rangle 2^{k+l+1} \quad (8)$$

$$*z5 = \bigotimes_{k=0}^{n-1} |(aL + aR)_k\rangle 2^{2k} \oplus \bigotimes_{k=0}^{n-2} \bigotimes_{l=k+1}^{n-1} |(aL + aR)_k \cdot (aL + aR)_l\rangle 2^{k+l+1} \quad (9)$$

The square computation circuit designs for 2 bit and 3 bit square units are given in Fig. 7(a), (b). *This step needs $3n + 2$ ancilla input bits.*

3. Step 3: Apply an array of n CNOT gates at locations $(z3, z3c)$. At the end of computation, $z3$ will maintain the same value whereas $z3c$ will have the copy of $*z3$ as shown below.

$$|*z3c\rangle = \bigotimes_{i=0}^{n-1} |*z3_i\rangle \oplus |*z3c_i\rangle \quad (10)$$

$$\text{where } z3c[n-1:0] \in \{0\}^n$$

This step needs n ancilla input bits.

4. Step 4: Apply values present at $z3c$ and $z4$ locations to adder which adds two operands of n bit width each. To store the carry out bit, an ancilla bit location $z3''$ is given as third operand. After the computation, $z4$ will maintain the same value whereas the value at location $z3c$ will be transformed as shown below. Since output of adder is concatenated output of sum and carry bit, we need an additional ancilla zero bit, so we append one ancilla zero location ($z3''$) to $z3c$ to hold the carry out bit. We refer transformed $z3c$ location as $z3''$ here onwards.

$$z3'' = \begin{cases} s_i & \text{for } 0 \leq i \leq n-1 \\ c_n & \text{for } i = n \end{cases} \quad (11)$$

$$s_i = \begin{cases} *z3c_i \oplus *z4_i \oplus c_i & 0 \leq i \leq n-1 \\ c_n & i = n \end{cases} \quad (12)$$

where c_i is the carry bit and is defined as:

$$c_i = \begin{cases} 0 & i = 0 \\ (*z3c_{i-1} \oplus *z4_{i-1})c_{i-1} \oplus *z3c_{i-1} * z4_{i-1} & 1 \leq i \leq n \end{cases} \quad (13)$$

This step needs 1 ancilla bit.

5. Step 5: Apply values present at locations $z5$ and $z3''$ as inputs at the subtractor unit. After the computation, $z5$ location retains its value whereas values at $z3''$ will be transformed as below. D_i indicates difference and B_i indicates borrow bits. We append an ancilla bit to location $z3''$ to adjust the width mismatch between the operands. We continue to refer $z3''$ with the same name.

$$z3'' = \begin{cases} D_i & \text{for } 0 \leq i \leq n \\ B_{n+1} & \text{for } i = n + 1 \end{cases} \quad (14)$$

$$D_i = \begin{cases} *z3_i'' \oplus *z5_i \oplus B_i & \text{for } 0 \leq i \leq n \\ B_{n+1} & \text{for } i = n + 1 \end{cases} \quad (15)$$

where B_i is the borrow bit and is defined as:

$$B_i = \begin{cases} 0 & i = 0 \\ (*z3_{i-1}'' \oplus *z5_{i-1})B_{i-1} \oplus *z3_{i-1}'' \cdot *z5_{i-1} & 1 \leq i \leq n + 1 \end{cases} \quad (16)$$

This step needs 1 ancilla bit.

6. Step 6: Apply values present at locations ($z3$, $z4$) and $z3''$ to $2n$ bit wide binary adder as first and second operand respectively. Here, $*z3$ and $*z4$ are concatenated and considered as single operand. After the computation, $*z3$ and $*z4$ will maintain the same value. $z3''$ will transform to the state as shown below. Here, both the operand width are not same, so $z3''$ is appended with $n/2$ ancilla bits on its LSB side and $n/2 - 2$ ancilla bits on its MSB side. This extended location of $z3''$ is referred as $Y2$ and concatenated locations ($z3$, $z4$) are referred as $Y1$. Now, resultant at location $Y2$ is computed as below.

$$*Y2 = \begin{cases} s_i & \text{for } 0 \leq i \leq 2n - 1 \\ c_n & \text{for } i = 2n \end{cases} \quad (17)$$

$$s_i = \begin{cases} *Y1_i \oplus *Y2_i \oplus c_i & \text{for } 0 \leq i \leq 2n - 1 \\ c_{2n} & \text{for } i = 2n \end{cases} \quad (18)$$

where c_i is the carry bit and is defined as:

$$c_i = \begin{cases} 0 & \text{for } i = 0 \\ (*Y1_{i-1} \oplus *Y2_{i-1})c_{i-1} \oplus *Y1_{i-1} * Y2_{i-1} & \text{for } 1 \leq i \leq 2n \end{cases} \quad (19)$$

This step needs $n-2$ ancilla input bits.

Phase 2: Decomputation Steps

The design steps shown here comprise of decomputation steps, which is necessary to remove the garbage bits generated in Phase 1. Design steps illustrated here are shown in Fig. 8.

7. Step 7: Apply values of locations ($z3$, $z1$), ($z4$, AR), and ($z5$, $z2'$) to inverse square computation blocks. The design of Inverse square blocks are same as square blocks. At the end of computation, $z1$, AR , and $z2'$ will retain the same values; $z3$, $z4$, and $z5$ will hold ancilla 0 bits.

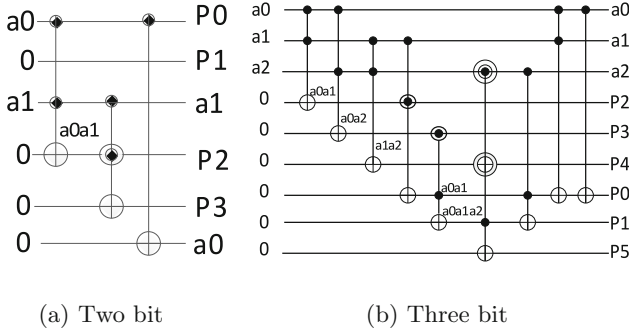


Fig. 7. Square design of two and three bits

8. Step 8: Apply values at locations AR and $z2'$ to subtractor of width $n/2 + 1$. At the end of computation, AR will retain the same value; $z2'$ is transformed to hold MSB $n/2$ bits of input a , so $*z2' = aL$. To adjust the width of the operands, an ancilla 0 bit is appended. This block releases two ancilla bits after the computation. *This Step requires an ancilla bit.*
9. Step 9: There are two copies of aL bits generated from Step 7 and Step 8. Apply values at locations $z1$ and $z2'$ to $n/2$ array of CNOT gates. After the computation, $z2'$ will retain aL , whereas $z1$ will hold $n/2$ ancilla 0 bits. After this step, input aL is regenerated. Thus, no garbage is produced in the computation of square. Total number of ancilla inputs required for each recursive call in this methodology is $5n + n/2 + 4$.

The complete architecture comprising of computation and de computation steps is shown in Fig. 9.

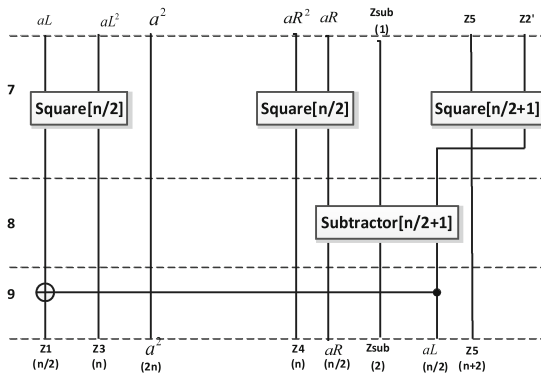


Fig. 8. Recursive square design: de computation steps

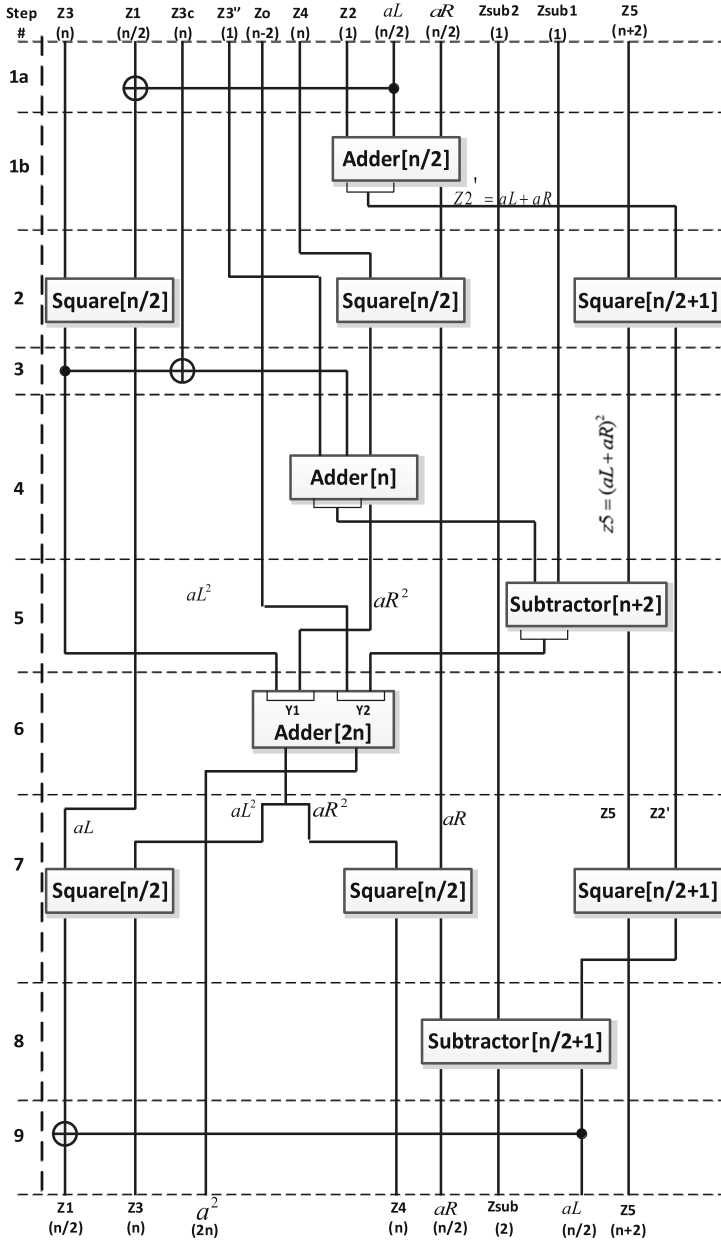


Fig. 9. Recursive square design

Table 1. Input lines comparison of reversible square designs

Data width	Proposed design	Design in-[1] Method 1	Design in-[1] Method 2	Design in-[8]
4	30	11	11	17
8	78	55	52	65
16	218	239	228	257
32	644	991	964	1025
64	1944	4031	3972	4097
128	5896	16255	16132	16385

Table 2. Input lines comparison with Karatsuba multiplier [11]

Data width	Parallel execution		Sequential execution	
n	Design in-[11]	Proposed square	Design in-[11]	Proposed square
4	44	30	44	30
8	174	114	118	78
16	596	382	308	218
32	1926	1218	830	644
64	6044	3790	2308	1944
128	18654	11634	6598	5896

4 Comparative Analysis

In reversible logic systems, it is not desirable to have garbage, so we have presented garbageless square design. In the literature, there exist dedicated square designs proposed by authors in [1, 8]. Our work utilizes $(5n + n/2 + 4)$ of ancilla bits(per recursive call) and produces zero garbage outputs. These values are obtained by taking case study of adder and subtractor designs proposed in [18–20]. The proposed work has zero garbage outputs. The existing works in [8] has $n^2 - 2n + 1$ number of garbage outputs. The two methodologies in [1] have $(n^2 - 3n - 2)$ and $(n^2 - 4n - 7)$ number of garbage outputs in the first and second approach respectively. This shows that the proposed work is best optimized in terms of garbage outputs as it has zero garbage bits overhead. Further, we compared proposed circuit design with existing reversible square designs. The improvements obtained are significant compared to existing works. We observed that our work improves number of input lines which includes data lines and ancilla lines for data width >8 compared to [1, 8], except the case when data width is ≤ 8 . The comparison result is captured in Table 1.

In Table 2, we present the results of comparison with Karatsuba multiplier design [11]. The parallel execution of recursive calls make use of separate ancilla work bits, whereas sequential recursive calls reuse the ancilla bits released by previous recursive call. In both the execution methods, the improvement is seen for proposed reversible square circuit. Since the objective of this work is to

optimize ancilla bits, in Table 1, we have shown ancilla bits calculation with respect to sequential recursive execution.

5 Discussion and Conclusion

In reversible computing, garbage bits and ancilla lines are considered as the design overhead. We have presented a new design of garbageless (zero garbage bits) reversible square computation circuit using recursive computation method. Further, the proposed design has improvement in terms of input lines compared to existing works in [1, 8]. The algorithm is functionally verified. Virtual verification is carried out by functional simulation using Xilinx ISE simulator.

The reversible square computation unit is a key component in digital signal processor and finds application in Newton Raphson divider, Euclidean distance computation, etc. The proposed work of designing the garbageless reversible circuitry will have a tremendous contribution in designing an efficient reversible hardware.

References

1. Banerjee, A., Das, D.K.: Squaring in reversible logic using iterative structure. In: 2014 East-West Design & Test Symposium (EWDTS), pp. 1–4. IEEE (2014)
2. Bennett, C.H.: Logical reversibility of computation. *IBM J. Res. Dev.* **17**(6), 525–532 (1973)
3. Choi, B.S., Van Meter, R.: A $\theta(\sqrt{n})$ -depth quantum adder on the 2D NTC quantum computer architecture. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **8**(3), 24 (2012)
4. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit. arXiv preprint quant-ph/0410184 (2004)
5. Draper, T.G., Kutin, S.A., Rains, E.M., Svore, K.M.: A logarithmic-depth quantum carry-lookahead adder. arXiv preprint quant-ph/0406142 (2004)
6. Haghparsat, M., Bolhassani, A.: Optimized parity preserving quantum reversible full adder/subtractor. *Int. J. Quantum Inf.* **14**(03), 1650019 (2016)
7. Jamal, L., Babu, H.M., et al.: Efficient approaches to design a reversible floating point divider. In: 2013 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3004–3007. IEEE (2013)
8. Jayashree, H.V., Thapliyal, H., Agrawal, V.K.: Design of dedicated reversible quantum circuitry for square computation. In: 27th International Conference on VLSI Design and 13th International Conference on Embedded Systems, pp. 551–556. IEEE (2014)
9. Jayashree, H.V., Thapliyal, H., Arabnia, H.R., Agrawal, V.K.: Ancilla-input and garbage-output optimized design of a reversible quantum integer multiplier. *J. Supercomputing* **72**(4), 1477–1493 (2016)
10. Oberman, S.F., Flynn, M.: Division algorithms and implementations. *IEEE Trans. Comput.* **46**(8), 833–854 (1997)
11. Portugal, R., Figueiredo, C.: Reversible Karatsubas algorithm. *J. Universal Comput. Sci.* **12**(5), 499–511 (2006)
12. Rabinowitz, P.: Multiple-precision division. *Commun. ACM* **4**(2), 98 (1961)

13. Shoaee, S., Haghparast, M.: Novel designs of nanometric parity preserving reversible compressor. *Quantum Inf. Process.* **13**(8), 1701–1714 (2014)
14. Smolin, J.A., Di Vincenzo, D.P.: Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate. *Phys. Rev. A* **53**(4), 2855–2856 (1996)
15. Takahashi, Y.: Quantum arithmetic circuits: a survey. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **92**(5), 1276–1283 (2009)
16. Takahashi, Y., Kunihiro, N.: A linear-size quantum circuit for addition with no ancillary qubits. *Quantum Inf. Comput.* **5**(6), 440–448 (2005)
17. Takahashi, Y., Kunihiro, N.: A fast quantum circuit for addition with few qubits. *Quantum Inf. Comput.* **8**(6), 636–649 (2008)
18. Thapliyal, H.: Mapping of subtractor and adder-subtractor circuits on reversible quantum gates. In: Gavrilova, M.L., Tan, C.J.K. (eds.) *Transactions on Computational Science XXVII. LNCS*, vol. 9570, pp. 10–34. Springer, Heidelberg (2016)
19. Thapliyal, H., Ranganathan, N.: Design of efficient reversible binary subtractors based on a new reversible gate. In: 2009 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2009, pp. 229–234. IEEE (2009)
20. Thapliyal, H., Ranganathan, N.: Design of efficient reversible logic-based binary and BCD adder circuits. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **9**(3), 17 (2013)
21. Thapliyal, H., Varun, T., Munoz-Coreas, E.: Quantum circuit design of integer division optimizing ancillary qubits and T-count. *arXiv preprint [arxiv:1609.01241](https://arxiv.org/abs/1609.01241)* (2016)
22. Vedral, V., Barenco, A., Ekert, A.: Quantum networks for elementary arithmetic operations. *Phys. Rev. A* **54**(1), 147 (1996)
23. Moghadam, M.Z., Navi, K.: Ultra-area-efficient reversible multiplier. *Microelectronics J.* **43**(6), 377–385 (2012)

Transactions on Computational Science XXIX

Gavrilova, M.; Tan, C.J.K. (Eds.)

2017, XI, 139 p. 66 illus., Softcover

ISBN: 978-3-662-54562-1