Microelectronics Journal 44 (2013) 519–537

Contents lists available at SciVerse ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Design of a compact reversible fault tolerant field programmable gate array: A novel approach in reversible logic synthesis

Md. Shamsujjoha, Hafiz Md. Hasan Babu *, Lafifa Jamal

Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh

ARTICLE INFO

Article history:
Received 28 April 2012
Received in revised form
26 January 2013
Accepted 4 February 2013
Available online 8 April 2013

Keywords:
FPGA
Garbage output
Low power design
Quantum cost
Reversible and fault tolerant computing

ABSTRACT

This paper demonstrates the reversible fault tolerant logic synthesis for the *F*ield *P*rogrammable *G*ate *A*rray (FPGA) and its realization using MOS transistors. Algorithms to design a compact reversible fault tolerant n-to-2^n decoder, $4n$-to-n multiplexers, a random access memory and a Plessey logic block of the FPGA have been presented. In addition, several lower bounds on the numbers of garbage outputs, constant inputs and quantum cost of the FPGA have been proposed. The comparative results show that the proposed design is much better in terms of gate count, garbage outputs, quantum cost, delay, and hardware complexity than the existing approaches.

1. Introduction

The connection between a single bit and corresponding minimum quantity of entropy was ascertained by Maxwell and Szilard in [1,2], which state that, "a minimum amount of energy ($KT\ln(m)$, where K is Boltzmann's constant of 1.38×10^{23} JK^{-1} and T is the absolute temperature of the environment and m is an integer proportional to the number of computed bit) must be dissipated during every elemental enactment of computation". Later Landauer proved that energy dissipation is only avoidable if the system is made reversible [3]. In [4], Bennett showed that a reversible computation, in which no information is destroyed, dissipate an arbitrarily small amount of energy which is equal to $KT\ln(1)$, *i.e.*, logically zero energy dissipation. It has also been pointed out that, an irreversible system has a fundamental lower limit to the energy dissipation during a computation which is equal to $KT\ln(2)$ for each erased bit [3,4]. An irreversible system can also store the information that is produced during a computation rather than erasing it, but does not always provide a unique path from each state to its previous state. Energy used to store these information is unrecoverable unless the computation is performed reversibly. If we are to continue the revolution in performance of computer hardware, we must reduce these energy dissipations. Although power dissipation due to information loss is negligible at present, but it will become a substantial factor by 2020 as a result of increased density in the computer hardware if Moore's Law continues to be in effect [5]. Gordon Moore predicted that, "*The number of transistors and resistors on a chip doubles every* 18 *months*." Which signifies, "*it would continue to improve at an exponential rate with the performance per unit cost increasing by a factor of* 2 *every* 18 *months or so*". This assumption has a great importance lies in the future technological necessity that the power dissipation should be minimized, otherwise internal overheating of a chip will demolish it. In this regard, reversible logic will play an extensively crucial role in the upcoming days. Moreover, a reversible circuit can be viewed as a special case of quantum circuit as quantum evolution must be reversible [6,7]. On the other hand, reversible fault tolerant circuit allows to detect faulty signal in the primary outputs of the circuit through parity checking. Parity checking is one of the oldest and widely used mechanisms for detecting single level fault in communication [8]. Generally it is used to detect errors in the storage or transmission of information, primarily because most of the arithmetic and other processing functions do not tend to preserve the parity of the data [9]. If the parity of the input data is maintained throughout the computation, then the intermediate checking would not be required and the entire circuit can preserve parity if its individual gate is parity preserving [10]. In other words, a reversible fault tolerant circuit can capture any erroneous result that tends to propagate through the downstream of modules without a danger of corrupting additional information. Over the past few years, both reversible and fault tolerant circuitry gained remarkable interest in the field of DNA-technology [11], nano-technology [12], optical computing [13], program debugging and testing [14], database recovery, quantum-dot-cellular automata [15], discrete event

* Corresponding author. Tel.: +880 29571911; fax: +880 28615583.
E-mail addresses: dishacse@yahoo.com (Md. Shamsujjoha), hafizbabu@hotmail.com (H.Md. Hasan Babu), lafifa@yahoo.com (L. Jamal).

0026-2692/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.mejo.2013.02.005

simulation [16], modeling of biochemical systems [17], and in the development of highly efficient algorithms [18,19]. Thus, the reversible fault tolerant and quantum circuits have been implemented and are seen as promising alternatives to conventional technologies.

New products in shortest possible time have become the catchword in today's electronic industry. By being able to test the product even before the fabrication *F*ield *P*rogrammable *G*ate *A*rray (FPGA) has become an extremely useful medium for the digital designs. Usually FPGA consists of an array of programmable logic blocks, interconnects (routing channels) and I/O cells. Logic blocks can be configured to implement sequential and complex combinational functions. Hence, influence both speed and density of an FPGA. As FPGAs are approximately 10 times less dense and three times slower than the mask programmed gate arrays [20], researchers are motivated to explore new logic blocks such that these density and speed gaps become as minimum as possible. Most popular logic blocks of the FPGAs are based on look-up tables (LUTs) and design form Plessey. A look-up table can implement any function of its inputs and accordingly described by their number of inputs. With more inputs, a LUT can implement more logic, and hence fewer logic blocks are needed. This helps in routing by asking for less area, since there are fewer connections to route between the logic blocks. However, as the number of inputs increases, the complexity grows exponentially. Fine-grain Plessey logic block solves this problem through cluster of components [20]. In these consequences, this paper investigates the reversible design methodologies of Plessey logic block and basic input–output cell of the FPGA. To the best of our knowledge, there are three such approaches in the literature [21–23]. However, all these schemes are not generalized and scalable. Thus, the main objective of this research is to develop a generalized structure of the reversible FPGA in addition to the fault detection capabilities.

2. Basic definitions and literature review

Grandness of the reversible and fault tolerant logic synthesis are detailed in the previous section. Initially, this section defines the reversible and fault tolerant gate in addition to their fundamental properties, which will be used in the next sections. Section 2.2 briefly defines the quantum cost, delay, hardware complexity and garbage output. Section 2.3 introduces the implementation techniques of reversible and fault tolerant logic synthesis. Section 2.4 demonstrates the popular reversible and fault tolerant gates along with their input–output specifications and quantum equivalent representations. This section also presents the designs of the transistor representations of the individual gates in order to implement the circuit using MOS transistors. Transistor simulations have been shown to check the correctness of the functions of the individual gates using standard p-MOS 901 and n-MOS 902 model with average delay of 0.030 ns and 0.12 μm channel length. Throughout the paper, it has been considered that the signals less than 0.001 ns stability are the glitches.

2.1. Reversible and fault tolerant gate

Block diagram of an $n \times n$ *reversible gate* is shown in Fig. 1, which is a data stripe block that uniquely maps between input vector $I_v = (I_0, I_1, \ldots, I_{n-1})$ and output vector $O_v = (O_0, O_1, \ldots, O_{n-1})$ denoted as $I_v \leftrightarrow O_v$ [12,24]. This one-to-one mapping among all inputs and outputs for each input–output sequence results in no information loss, *i.e.*, if we design a circuit using only reversible gate entire circuit becomes information lossless.

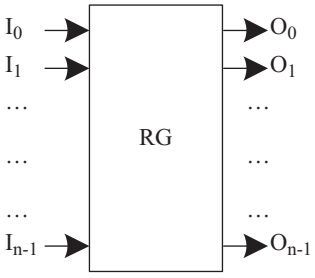

Fig. 1. Block diagram of an $n \times n$ reversible gate.

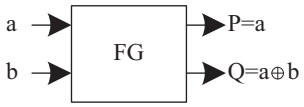

Fig. 2. Block diagram of a 2×2 reversible Feynman gate.

A *Fault tolerant gate* is a reversible gate which constantly preserves same parity between inputs and outputs. More specifically, an $n \times n$ fault tolerant gate clarifies the following property between input and output vectors:

$$I_0 \oplus I_1 \oplus \cdots \oplus I_{n-1} = O_0 \oplus O_1 \oplus \cdots \oplus O_{n-1} \quad (1)$$

The parity preserving property of Eq. (1) allows to detect a faulty signal from the circuit's primary output. Researchers showed that, if a reversible circuit is drawn using only reversible fault tolerant gates, then the entire circuit preserves parity, thus able to detect a faulty signal [7–10].

2.2. Delay, garbage output, hardware complexity and quantum cost

In a logic circuit or reversible logic circuit, the path consists of maximum number of gates from any input to any output is known as critical path. However, it is an NP complete problem to find the critical path specially for large circuits [25]. Also there may be more than one critical path in a circuit. Thus, researchers used to select the path which is the most likely candidates for the critical paths (*delay* of a logic circuit). Generally in reversible logic synthesis, the most probable critical path delay is calculated considering the following two assumptions:

- Each gate performs computation in unit time.
- All inputs to the circuit are known before the computation begins.

Unwanted or unused output of a reversible gate (or circuit) is known as *garbage output*, *i.e.*, the output which are needed only to maintain the reversibility are known as garbage output [7]. In other words, outputs that neither used as primary outputs nor as input to another reversible gate are garbage outputs. For example, to perform the exclusive-OR between two inputs, a 2×2 reversible Feynman gate (*FG*) can be used. This realization produces an extra dummy output along with the desired output signal, which is needed to preserve the reversibility. This extra output is the garbage output, denoted by *P* in Fig. 2. Heavy price is paid off for each garbage output. Thus, the minimization of garbage outputs in the circuit is one of the main challenges for the researchers.

As mentioned earlier, determining the critical path from the large circuit is not easy. Thus, researchers determined the *hardware complexity* of the circuit and is considered as a seemingly equivalent performance evaluation criteria. The number of basic operations (Ex-OR, AND, NOT, etc.) needed to realize the circuit is

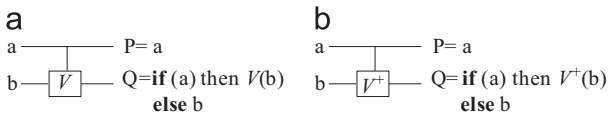

Fig. 3. Quantum realization of controlled (a) V gate (b) and V^+ gate.

referred as the hardware complexity of the circuit. Actually, a constant complexity is assumed for each basic operation of the circuit, such as α for Ex-OR, β for AND, γ for NOT etc. Then, total hardware complexity is calculated with respect to these assumed complexities. For example, the hardware complexity of the circuit shown in Fig. 2 is α, since it can be realized with a single Ex-OR only.

The *quantum cost* for all 1×1 and 2×2 reversible gates are considered as 0 and 1, respectively [7,24]. Hence, the quantum cost of a reversible circuit is the total number of 2×2 quantum gates used in that reversible circuit. In other words, number of 2×2 Ex-OR, controlled-V (square root of NOT, *i.e.,* SRN) or controlled-V^+ (hermitian of SRN) represent the quantum cost of the circuit.[1] A controlled-V gate is shown in Fig. 3(a). In a controlled-V gate, when the control signal $a=0$, the qubit b will pass through the controlled part and keep it unchanged, *i.e.*, $Q=b$. If the value of $a=1$, then $Q=V(b)$, where,

$$V=(i+1)/2\begin{pmatrix}1 & -i\\ -i & -1\end{pmatrix}$$

Here, i is the basic imaginary unit, *i.e.*, $i=\sqrt{-1}$. A controlled-V^+ gate is shown in Fig. 3(b). In a controlled-V^+ gate, when the control signal $a=0$, the qubit b will pass through the controlled part and keep it unchanged, *i.e.*, $Q=b$. If the value of $a=1$, then $Q=V^+(b)$, where V^+ is the hermitian of V. Thus $V\times V$ or $V^+\times V^+$ creates a unitary matrix of NOT gate, whereas, $V\times V^+$ or $V^+\times V$ is an identity matrix (I) describing just a quantum wire.

2.3. Implementation techniques

One of the early implementation techniques for reversible circuits is *C*harge *R*ecovery *L*ogic (CRL) [26]. CRL relies on explicit reversible pipelined logic gates, where the information necessary for energy recovery are used to compute a value which is provided by computing its functional inverse. *S*plit-level *C*harge *R*ecovery *L*ogic (SCRL) was proposed in [27], which used split-level voltages. Lim et al. [28] proposed *R*eversible *E*nergy *R*ecovery *L*ogic (RERL) for ultra-low-energy consumption. Later, they proposed *n* MOS *R*eversible *E*nergy *R*ecovery *L*ogic (nRERL) in [29]. The nano-electronic and opto-electronic implementations of reversible gates can be found in [30]. Transistor level realization of reversible circuits is a relatively new idea and extensively used by the researchers [31,32].

2.4. Popular reversible and fault tolerant gates

There are several reversible and fault tolerant gates in the literature among which Feynman double gate (*F2G*) and Fredkin gate (*FRG*) are the most popular. Following subsections present the above mentioned gates along with their input–output specifications, quantum equivalent realizations, transistor level implementations, working procedures and applications. CAD tools DSCH-2.7 is used as simulator with average delay of 0.030 ns and 0.12 μm transistor channel length. Because it has a built-in extractor which generates a NETLIST, VERILOG descriptions and the timing diagrams [33].

[1] There are some exceptional cases, more details can be found in [7,12,13,24].

2.4.1. Feynman double gate

Input vector (I_v) and output vector (O_v) for 3×3 reversible Feynman double gate (*F2G*) is defined as follows: $I_v=(a,b,c)$ and $O_v=(a,a\oplus b,a\oplus c)$. Block diagram of *F2G* is shown in Fig. 4(a). Fig. 4(b) represent the quantum equivalent realization of *F2G*. From Fig. 4(b), we find that it is realized with two 2×2 Ex-OR gate. Thus, according to our previous discussion in Section 2.3, the quantum cost of Feynman double gate is two. According to our design procedure, 12 transistors are required to realize Feynman double gate reversibly, which is shown in Fig. 4(c). Fig. 4(d) represents the corresponding timing diagram of the transistor realization of the reversible Feynman double gate. Feynman double gate can be used as copying gate. For example, if the input vector (I_v) of a Feynman double gate is set as ($a,b=0,c=0$), then the output vector (O_v) of a Feynman double gate will be (a,a,a).

2.5. Fredkin gate

Over the years, Fredkin gate had been extensively used by the researchers. This gate is one of very few reversible gate which can realize all basic operations while preserving the parity. Input and output vectors for 3×3 Fredkin gate (*FRG*) are defined as follows: $I_v=(a,b,c)$ and $O_v=(a,a'b\oplus ac,a'c\oplus ab)$. Block diagram of *FRG* is shown in Fig. 5(a). Fig. 5(b) represents the quantum equivalent realization of *FRG*. In Fig. 5(b), each rectangle is equivalent to a 2×2 quantum primitive, therefore its quantum cost is considered as one [7,12,13,24]. Thus, the quantum cost of *FRG* is five. To realize the *FRG*, four transistors are needed as shown in Fig. 5(c) and its corresponding timing diagram is shown in Fig. 5(d).

Fredkin gate can be used to swap second and third inputs using first input as a controlled input, *i.e.*, it can work as 2:1 Multiplexer (Mux). Referring to the input–output combinations of Fredkin gate, when $a=1$, the inputs b and c will be swapped. The resulting value of the outputs are $Q=c$ and $R=b$. Whereas, if $a=0$, then outputs P, Q and R are directly connected to inputs $a(=0)$, b, and c, respectively (shown in Fig. 6(a)). Fredkin gate is also an universal gate. The universality of Fredkin gate is shown in Fig. 6(b)–(d).

Among the reversible gates discussed above, Fredkin and Feynman double gates comply with the rule of Eq. (1). Therefore, according to our previous discussion in Section 2.1, if a circuit is designed using only Fredkin and Feynman double gates, then the circuit will inherently become fault tolerant [7–10]. The parity preserving (fault tolerant) property of Fredkin and Feynman double is shown in Table. 1.

3. Existing works

Essential technical background on FPGA is presented in this section which is required to understand the proposed work.

3.1. Basic structures of an FPGA

A crucial part of FPGA creation lies in their architecture as shown in Fig. 7(a). This architecture governs the nature of its programable logic functionality. As shown in the figure, user-programable routing network provides connections between the logic blocks and the logic performs the combinational functions as discussed in Section 1.

Fig. 7(c) and (d) represent the magnified diagrams of the routing channel and basic input-output block of the FPGA, respectively. From Fig. 7(d), we find that, a basic input–output block consists of 2-to-1 Muxs, input buffer, output driver and D-latches. The latches of basic input–output block are identical to the latches of Plessey logic block. From Fig. 7(d), we also find that

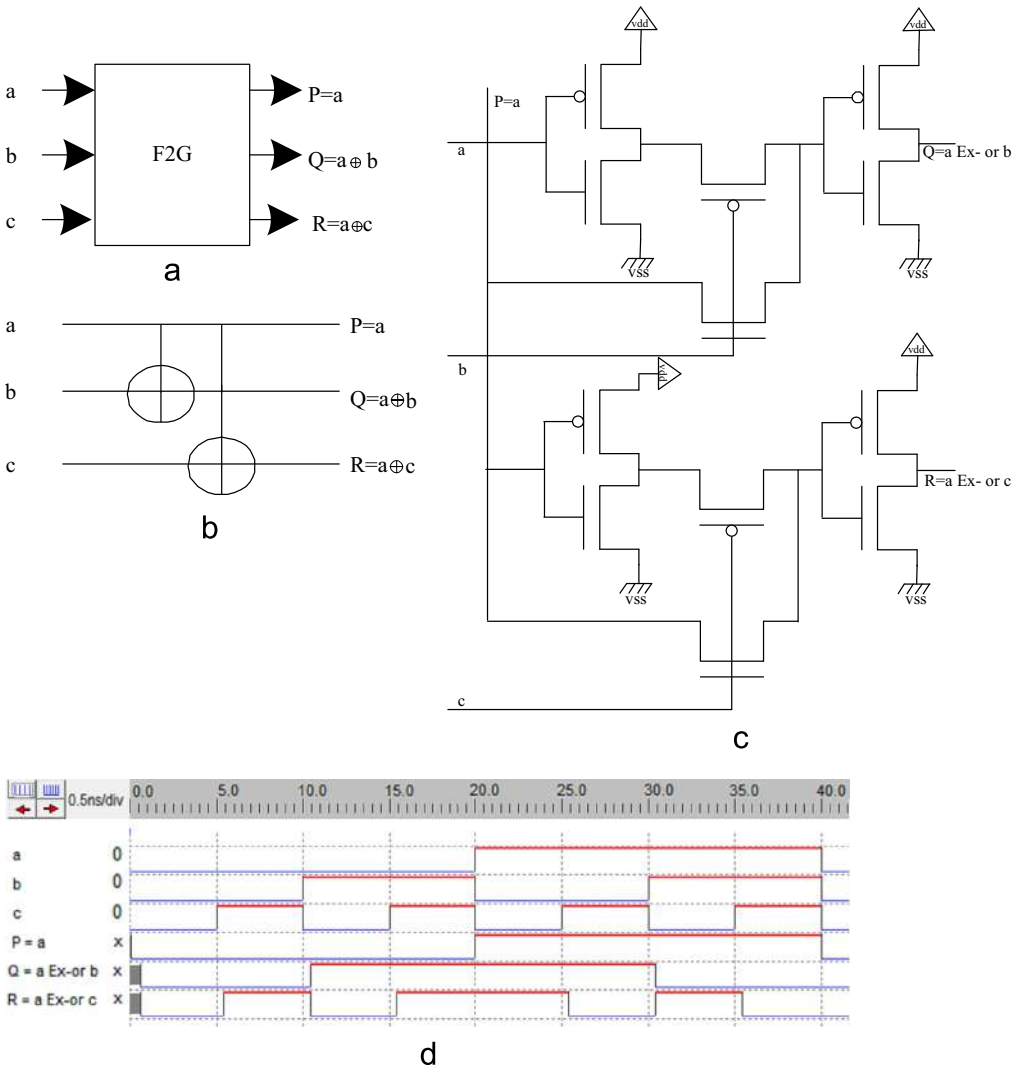

Fig. 4. Reversible 3 × 3 Feynman double gate (a) block diagram, (b) quantum equivalent realization, (c) transistor realization and (d) timing diagram.

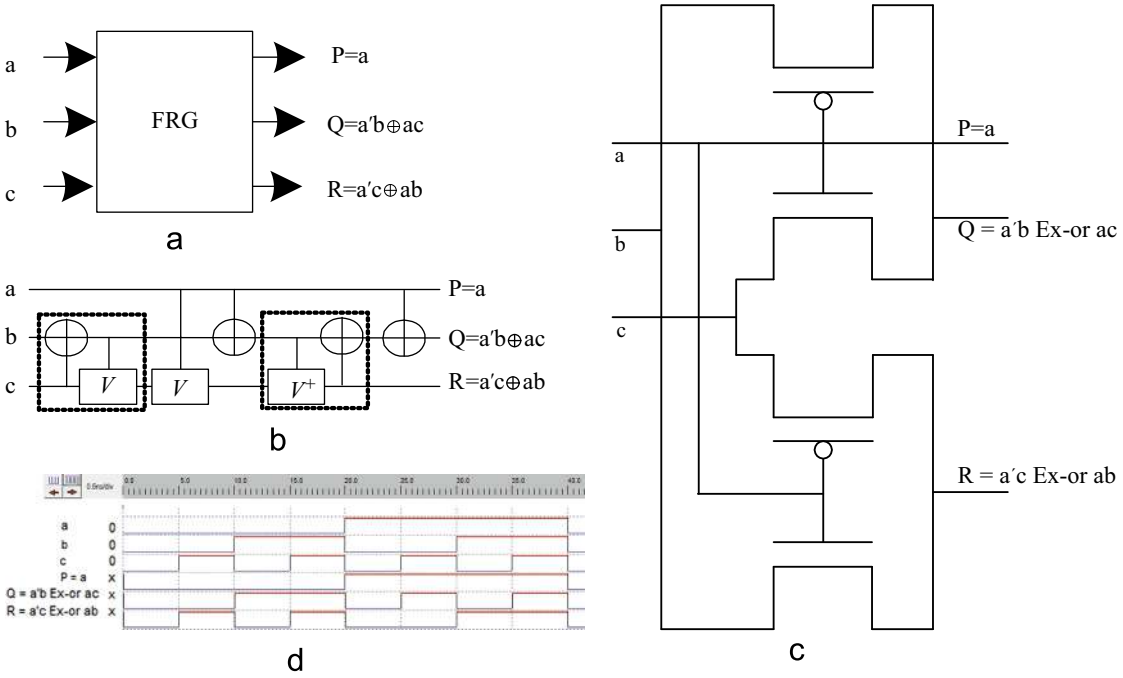

Fig. 5. Reversible 3 × 3 Fredkin gate (a) block diagram, (b) quantum equivalent realization, (c) transistor realization and (d) timing diagram.

the SET/RESET and clock signals of all the latches are common *i.e.*, the shared signals, which generally are adjusted by the designers.

3.2. Reversible FPGAs

To the best of our knowledge, three papers have been published so far on reversible FPGA. First paper on reversible FPGA had been published in 2009 [22]. This paper showed a reversible Plessey logic block of the FPGA. Recently in 2011, an improved design of reversible Plessey logic block had been proposed in [21]. Reversible input–output block had been attempted in [23], which is shown on Fig. 8. To design all the components of reversible Plessey logic block, authors in [22] used reversible *NH*, *BSP*, *FG* and Toffoli gates (*TG*). The quantum cost of *FG*, *NH*, and *TG* are 1, 4, and 5, respectively. However, the quantum cost of *BSP* gate is not shown in [22]. From the architecture of *BSP* gate it can be determined that it is the combinations of 3×3 and 4×4 Toffoli gate, thus its quantum cost is 13 (referring to the Section 2). On the other hand, the reversible Plessey logic block from [21] consists of reversible Mux and Peres gates along with the above mentioned gates. The quantum cost of both Mux and Peres gate are 4. More details on these reversible gates, FPGA and their applications can be found in [7–17,21–24].

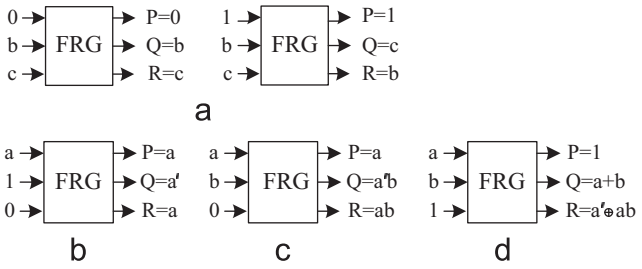

Fig. 6. FRG as (a) 2:1 Mux, (b) NOT gate, (c) AND gate and (d) OR gate.

Table 1
Truth table for *F2G* and *FRG*.

Input			Output of *F2G*			Output of *FRG*			Parity
A	B	C	P	Q	R	P	Q	R	
0	0	0	0	0	0	0	0	0	Even
0	0	1	0	0	1	0	0	1	Odd
0	1	0	0	1	0	0	1	0	Odd
0	1	1	0	1	1	0	1	1	Even
1	0	0	1	1	1	1	0	0	Odd
1	0	1	1	1	0	1	1	0	Even
1	1	0	1	0	1	1	0	1	Even
1	1	1	1	0	0	1	1	1	Odd

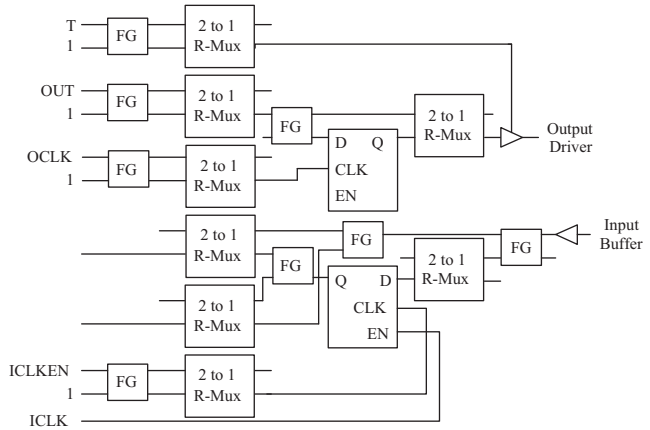

Fig. 8. Block diagram of an existing reversible input–output pad [23].

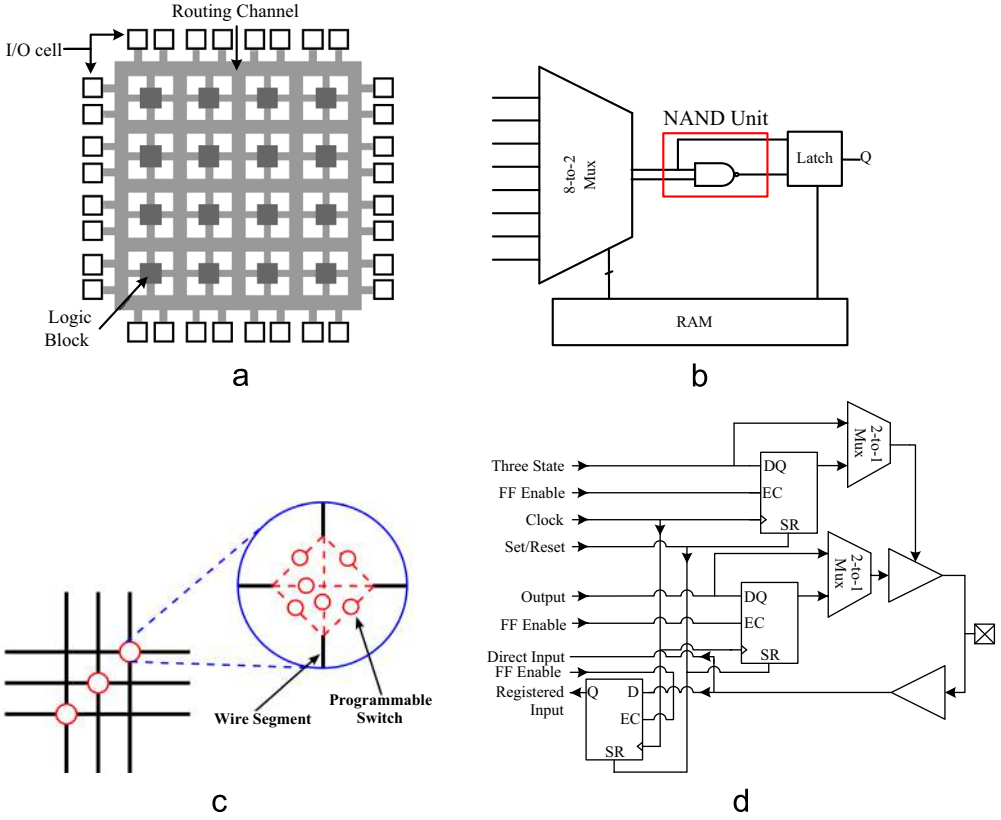

Fig. 7. Block diagram of (a) basic irreversible FPGA, (a) plessey logic block, (c) routing channel and (d) basic I/O block.

4. Proposed reversible fault tolerant FPGA

This section illustrates the design layouts, algorithms, theoretical properties and working procedures of the proposed reversible fault tolerant FPGA. As mentioned in Section 3, Plessey logic block consists of NAND Unit, D-latch, 8-to-2 multiplexer (Mux) and random access memory (RAM). The RAM requires underneath n-to-2^n decoder and write enable master-slave flip-flop circuitry. Therefore, Section 4.1 proposes reversible fault tolerant D-latch of the FPGA. Reversible fault tolerant write enable master-slave flip-flop of the FPGA is proposed in Section 4.2. Section 4.3 demonstrates the proposed fault tolerant n-to-2^n decoder of the FPGA. In Section 4.4, the $4n$-to-n reversible fault tolerant Mux of the FPGA has been proposed. RAM for fault tolerant FPGA is proposed in Section 4.5. To design the RAM for fault tolerant Plessey logic block, we extend our proposed reversible design of the RAM [24] to the reversible fault tolerant RAM with more optimally. Then, combining all these components, the architecture of the fault tolerant Plessey logic block is presented in Section 4.6. The section ends by proposing reversible fault tolerant basic input–output block of the FPGA in Section 4.7.

4.1. Proposed reversible fault tolerant D-latch for FPGA

Fig. 9(a) represents the architecture of proposed reversible fault tolerant D-latch for the FPGA, from where we find that, a Fredkin and a Feynman double gates are used in the circuit. The Fredkin gate is needed to produce Q, whereas Feynman double gate produces Q^+ and Q. The equations for the proposed fault tolerant latch are $Q_n = D.Clk + Clk'.Q_{n-1}$ and $Q^+ = Q'$ (n is time varying), which are used to capture the logic level when the data is present and the clock is SET.

Corresponding timing diagram of the proposed latch is shown in Fig. 9(b). The circuit is simulated with the average current of 0.001 mA, average power of 0.001 mW and average delay of 0.235 ns. From Fig. 9(b), we find that when clock is low the value of D has no effect on Q (0–2.5 ns). However, when clock is high Q follows D, *i.e.*, are in transparent mode (2.5–7.5 ns). If the clock is low again, then Q retains the previous value of D, *i.e.*, the value of D when clock went from high to low ($1 \rightarrow 0$) last time (7.5 to 10 ns, 15 to 17.5 ns and so on).

Table 2 shows the comparison of proposed latch with the existing latches.[2] From Table 2, we find that the proposed design performs better than the existing designs. As shown earlier in Section 2 and in Table 1, both the reversible gates that has been used to design the proposed latch preserves the parity. Hence, the proposed latch inherently preserves parity, and thus it able to detect a faulty signal at its primary output [7–10].

Lemma 1. *A reversible fault tolerant D-latch for FPGA can be realized with*2 *reversible fault tolerant gates*, 2 *garbage outputs*, 7 *quantum cost*, $4\alpha+4\beta+2\gamma$ *hardware complexity and*16 *transistors*, *where*, α, β *and* γ *are the hardware complexity of two-input Ex-OR*, *AND and NOT calculations*, *respectively*.

4.2. Proposed reversible fault tolerant write enable master-slave flip-flop for FPGA

Write enable master-slave flip-flop is the most significant element for a RAM of FPGA. Fig. 10(a) shows the architecture of the proposed *R*eversible *F*ault tolerant write enable *M*aster-*S*lave *F*lip-*F*lop (RFMSFF). From now on, we denote a reversible fault tolerant write enable master-slave flip-flop as RFMSFF which works on two modes (read and write), since data are both read from and written into RAM of FPGA. In Fig. 10(a), W works as the write enable bit. From block diagram, we find that the proposed RFMSFF consists of an *FRG*, two *F2G* and two reversible *F*ault

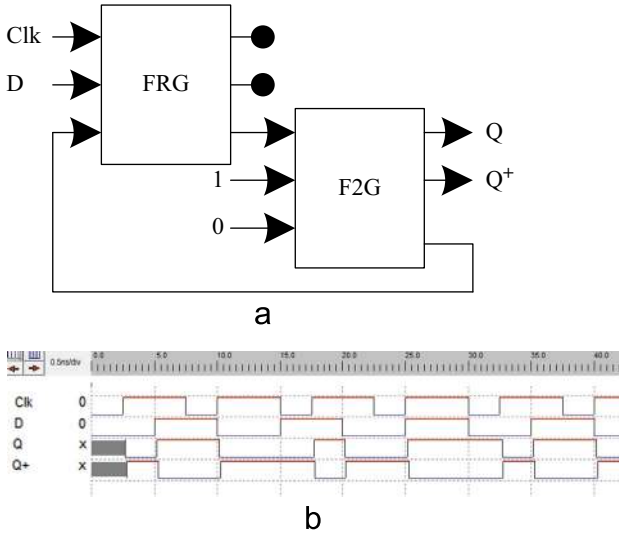

Fig. 9. (a) Block diagram of the proposed reversible fault tolerant D-Latch and (b) timing diagram of the proposed reversible fault tolerant D-Latch.

Table 2
Comparison of reversible D-Latch.

	GT	GO	QC	HC	UD	TR
Existing circuit [22]	2	2	9	$6\alpha+6\beta+3\gamma$	2	18
Existing circuit [23,34][a]	2	2	8	$5\alpha+4\beta+2\gamma$	2	22
Existing circuit [35]	2	2	11	[b]	2	[b]
Proposed circuit	2	2	7	$4\alpha+4\beta+2\gamma$	2	16

[a] The D-latch designs from [23,34] are identical.
[b] As the circuit in [35] is a quantum circuit, we do not compare the proposed work with [35] in-terms of HC and TR.

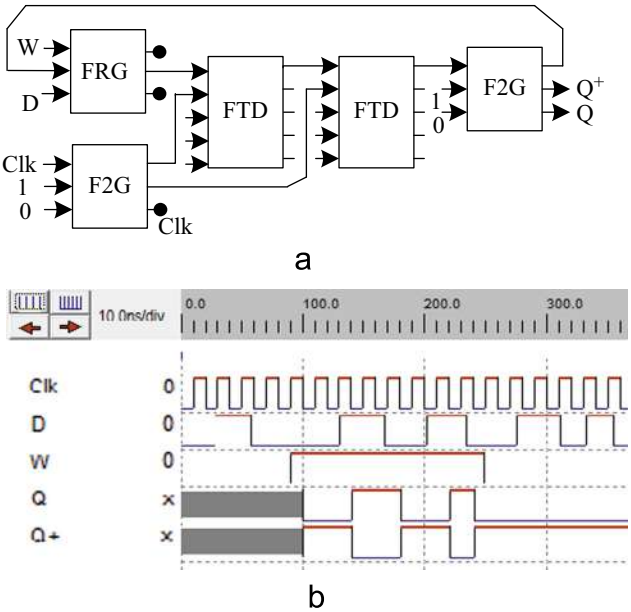

Fig. 10. Proposed reversible fault tolerant write enable master-slave flip-flop. (a) Architectural block diagram and (b) timing diagram.

[2] In this table and all the following tables, we consider, GT=no of gates, GO=garbage outputs, QC=quantum cost, HC=hardware complexity, UD=unit delay of the critical path, and TR=no of transistors. Throughout the paper, it is also considered that α, β, and γ are the hardware complexity of two-input Ex-OR, AND and NOT calculations, respectively.

Tolerant *D*-latch (FTD) (proposed in Section 4.1). The corresponding Timing diagram of Fig. 10(a) is shown in Fig. 10(b). From Fig. 10(b), we find that the value of *D* has no effect on *Q* or Q^+ (0–100 ns) until the write enable bit (*W*) is SET. However, when *W* is in high state, *Q* follows *D* (100–250 ns). Also, *Q* keeps the previous value of *D* when *W* gets low again (250 to 365 ns). This previous value is the value of *D*, when *W* is changed into low form high last time (250 ns), and of-course, the values are changed according to the clock pulses.

An alternate design of the proposed RFMSFF without FTD is shown in Fig. 11, which is our proposed reversible design which has already been published in [24]. From Fig. 11, we find that this alternative design consists of three *FRG*s, four *F2G*s and produces seven garbage outputs. From above two figures, we also find that both designs are identical in all aspects of evaluation parameters of reversible logic synthesis, thus, either of them can be used for the proposed RAM of the FPGA. Table 3 shows the comparison of proposed reversible fault tolerant write enable master-slave flip-flop with the existing reversible write enable master-slave flip-flop. In [22], unused clock pulse(s) (*Clk*) was(were) not considered as garbage output. However, according to our previous discussion in Section 2, the unused clock pulse(s) should be considered as garbage output(s). Considering the unused clock pulse as garbage outputs, the design described in [22] has 7 garbage outputs, whereas, the proposed design has 6 garbage outputs. Most interestingly, proposed write enable master-slave flip-flops can detect the faulty signal at its primary outputs, since the proposed circuits are designed using only reversible fault tolerant gates or circuit [7–10], which is missing in the existing circuit.

Lemma 2. *A reversible fault tolerant write enable master-slave flip-flop for FPGA can be realized with*7 *gates and*7 *garbage outputs*, 23 *quantum cost and* $14\alpha+12\beta+6\gamma$ *hardware complexity*.

4.3. Proposed reversible fault tolerant *n*-to-2^n decoder for FPGA

Decoder is the collection of logic gates fixed up in a specific way such that, for an input combination, all output *terms* are low except one. These *terms* are the *minterms* which use a variable once, and once only. Thus, when an input combination changes, two outputs will change. Let there be *n* inputs. So, number of outputs will be 2^n. In other words, there will be one line at the output for each possible input. There are several designs of reversible decoders in the literature, but the design [14] is the only reversible decoder which can preserve parity too. However, the design [14] is not generalized and compact. Therefore, this section propose a reversible fault tolerant decoder, which is compact and generalized.

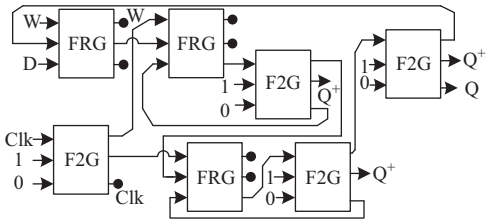

Fig. 11. Alternative architectural block diagram of the proposed reversible fault tolerant write enable master-slave flip-flop.

Table 3
Comparison of reversible write enable master-slave flip-flop.

	GT	GO	QC	HC	UD
Existing circuit [22]	7	9	25	$14\alpha+20\beta+8\gamma$	7
Proposed circuit	7	7	23	$14\alpha+12\beta+6\gamma$	6

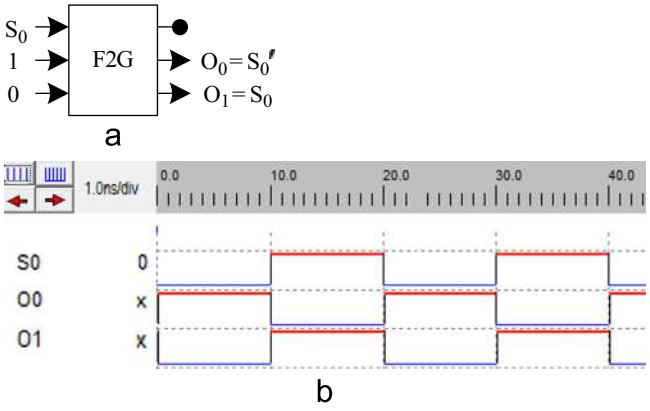

Fig. 12. Proposed 1-to-2 RFD (a) block diagram and (b) timing diagram.

Considering the simplest case, $n=1$, we have a 1-to-2 decoder. Only an *F2G* can work as 1-to-2 *R*eversible *F*ault tolerant *D*ecoder (RFD) as shown in Fig. 12(a). From now on, we denote a reversible fault tolerant decoder as RFD. The corresponding timing diagram of Fig. 12(a) is shown in Fig. 12(b). Increasing *n* from 1 to 2 first, and then 2 to 3, we have 2-to-4 and 3-to-8 decoders. Fig. 13(a) and (d) represent the architectures of 2-to-4 and 3-to-8 RFD. The corresponding timing diagrams of Fig. 13(a) and (d) are shown in Fig. 13(c) and (e), respectively. From Fig. 13(d), we find that 3-to-8 RFD is designed using 2-to-4 RFD. Thus, a schema of Fig. 13(a) is created as shown in Fig. 13(b).

Algorithm 1 presents the design procedure of the proposed *n*-to-2^n RFD. Line 5 of the algorithm assigns the input to the *F2G* for the first control bit (S_0), whereas line 8 assigns first two inputs to the *FRG* for all the remaining control bits. Lines 9–11 assign third input to the *FRG* for $n=2$, while lines 12–14 assign third input to the *FRG* through a recursive call to previous RFD for $n>2$. Lines 17–18 returned the outputs. The complexity of this algorithm is $O(n)$, where *n* is the number of data bits. To the best of our knowledge, no other polynomial time algorithm exists for an *n*-to-2^n reversible fault tolerant decoder.

Algorithm 1. Algorithm for the proposed reversible fault tolerant *n*-to-2^n decoder for FPGA, ***RFD***(***S***,***F2G***,***FRG***).

Input: Data input set $S(S_0,S_1,\ldots,S_{n-1})$
Feynman double gate (*F2G*) and Fredkin gate (*FRG*)
Output: *n*-to-2 ^{n} reversible fault tolerant decoder circuit

```
1  begin
2    i=input, o=output
3    for j←0 to n−1 do
4      if j=0 then
5        S_j→first.i.F2G, 1→second.i.F2G, 0→third.i.F2G
6      end if
7      else
8        S_j→first.i.FRG, 0→second.i.FRG
9        if j=2 then
10         third.o.F2G→third.i.FRG
11       end if
12       else
13         call RFD(j-1), RFD.o.j→third.i.FRG_j
14       end if
15     end if
16   end for
17   return FRG.o.3 and FRG.o.2→desired output
18          remaining F2G.o and FRG.o→garbage output.
19 end
```

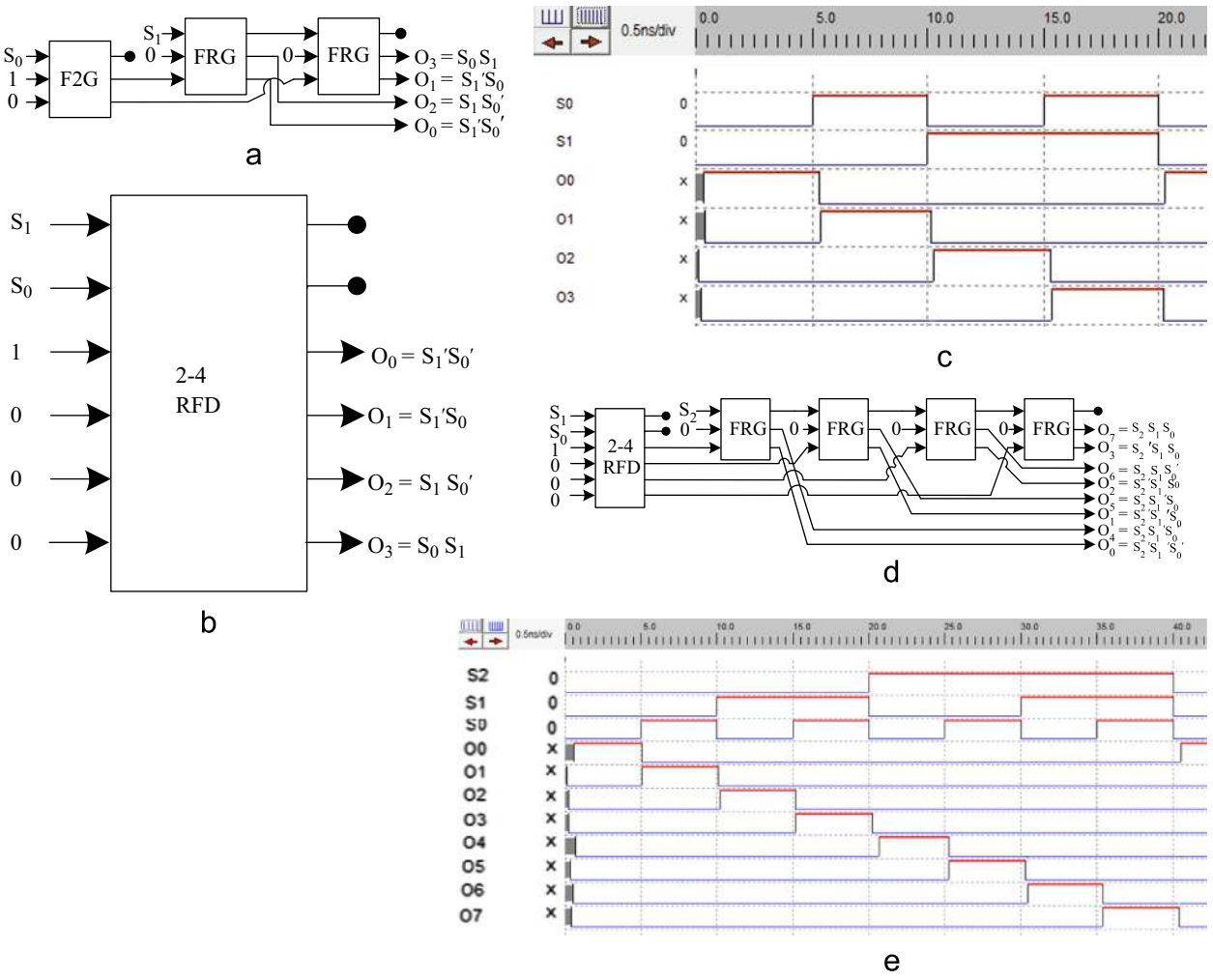

Fig. 13. (a) Block diagram of the proposed 2-to-4 RFD. (b) Schematic diagram of 2-to-4 RFD. (c) Timing diagram of 2-to-4 RFD. (d) Block diagram of the proposed 3-to-8 RFD. (e) Timing diagram of 2-to-4 RFD.

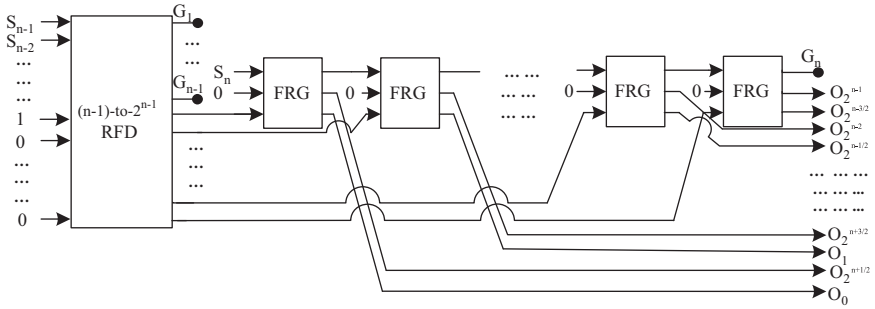

Fig. 14. Block diagram of the proposed n-to-2^n RFD.

According to Algorithm 1, the architecture of the proposed n-to-2^n RFD is shown in Fig. 14. Section 2 presents the designs of the transistor representations of *FRG* and *F2G*. These are used to get the MOS circuit of the proposed decoder. The timing diagrams of individual circuits also show the functional correctness of the proposed decoders. Table 4 shows a comparative study of proposed decoders with existing one.

Theorem 1. *An n-to-2^n reversible fault tolerant decoder can be realized with at least n garbage outputs and 2^n constant inputs, where n is the number of data bits.*

Table 4
Comparison of reversible fault tolerant decoders.

	GT	GO	QC	HC	UD
2-to-4 Existing circuit [14]	3	2	15	$6\alpha+12\beta+6\gamma$	3
2-to-4 Proposed circuit	3	2	12	$6\alpha+8\beta+4\gamma$	2
3-to-8 Existing circuit [14][a]	≥ 7	≥ 3	≥ 35	$\geq 14\alpha+28\beta+14\gamma$	≥ 7
3-to-8 Proposed Circuit	7	3	32	$14\alpha+24\beta+12\gamma$	4

[a] The design is not generalized one, *i.e.*, it is not an n-to-2^n decoder.

Proof. An n-to-2^n decoder has n inputs and 2^n outputs. Thus, to maintain the property of reversibility, there should be at least (2^n-n)

Table 5
1-to-2 decoder with one constant input.

Inputs		Outputs		Parity
CI	S_0	O_0	O_1	
0	0	1	0	I_p=E, O_p=O
0	1	0	1	$I_p=O_p$
1	0	1	0	$I_p=O_p$
1	1	0	1	I_p=E, O_p=O

constant inputs. However, this combinations does not preserve parity. To preserve the parity, at least n more constant inputs are needed. So, there should be at least n garbage outputs. □

Example 1. Let the value of n be 1. Then, we have the 1-to-2 reversible fault tolerant decoder. As shown in Section 2, for a reversible circuit, it is necessary to maintain the one-to-one correspondence between input and output vectors. Thus, any reversible circuit should have the equal number of inputs and outputs. In the 1-to-2 decoder, there are 2 primary outputs (O_0, O_1) but 1 input (S_0). So, 1-to-2 reversible decoder should have at least 1 constant input. The value of this constant input can be either 0 or 1. Table 5 shows that whatever the value of this constant input, it will never be able to preserve the parity between input and output vectors, which is the prime requirement of the reversible fault tolerant logic circuit.[3] Therefore, to preserve the parity for the 1-to-2 reversible fault tolerant decoder, we need at least one more constant input, *i.e.*, the total constant inputs are 2.

Next, we must prove the existence of combinational circuit which can realize the reversible fault tolerant 1-to-2 decoder by 2 constant inputs. This can easily be accomplished by the circuit as shown in Fig. 12(a). It can be verified that the circuit in Fig. 12(a) is reversible and fault tolerant with the help of its corresponding truth table.

Now, in 1-to-2 reversible fault tolerant decoder, there are at least 2 constant inputs and 1 primary input, *i.e.*, the total inputs are 3. Thus, in the 1-to-2 reversible fault tolerant decoder, there should be at least 3 outputs, otherwise it will never comply with the properties of reversible parity preserving circuit. Among these 3 outputs, only 2 are primary outputs. So, the remaining 1 output is the garbage output, which holds Theorem 1 for $n=1$.

Theorem 2. *A 2-to-4 reversible fault tolerant decoder can be realized with at least*12 *quantum cost.*

Proof. A 2-to-4 decoder has four different 2×2 logical AND operations. A reversible fault tolerant AND[4] operation requires at least 3 quantum cost. Therefore, 2-to-4 reversible fault tolerant decoder can not be realized with less than 12 quantum cost. □

Example 2. Fig. 13(a) and the discussions in Section 2 are the proof for the existence of 2-to-4 RFD 12 quantum cost. Next, we want to prove that it is not possible to realize a reversible fault tolerant 2-to-4 decoder with fewer than 12 quantum cost. In the 2-to-4 decoder, there are four different AND operations ($S_1'S_0'$, $S_1'S_0$, S_1S_0', S_1S_0). It will be enough if we are able to prove that it is not possible to realize a reversible fault tolerant AND with fewer than three quantum cost. Consider

(i) If we make use of one quantum cost to design the AND, that of course is not possible according to our discussion in Section 2.

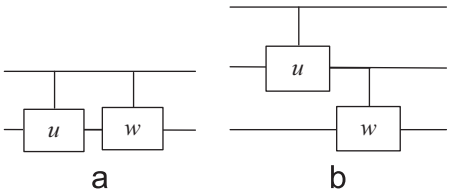

Fig. 15. Combinations of the two 2×2 quantum primitive gates.

Table 6
Truth table of Fig. 15(a).

a	*b*	*a*	*ab*
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

Table 7
Truth table of Fig. 15(b).

a	*b*	*c*	*a*	*ab*
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

(ii) If we make use of two quantum cost to design AND, then we must make use of two 1×1 or 2×2 gates. Apparently two 1×1 gates cannot generate the AND. Aiming at two 2×2 gates, we have two combinations, which are shown in Fig. 15(a) and (b). In Fig. 15(a), the output must be (a, ab) if the inputs are (a, b). The corresponding truth table is shown in Table 6.

From Table 6, we find that, first and second row have the same output combinations for different input combinations, *i.e.*, outputs are not at all unique to its corresponding input combinations. So it can not achieve the reversible AND. For Fig. 15(b) if inputs are (a, b, c) then, the outputs of the lower level will be offered to the next level as a controlled input, this means that second output of Fig. 15(b) have to be ab, otherwise it will never be able to get output ab, since, third output of Fig. 15(b) is controlled by the second output, thereby according to Table 7, we can assert that the second combination is impossible to realize the AND no matter how we set the third output of Fig. 15(b), the input vectors will never be one-to-one correspondent with the output vectors. Therefore, we can conclude that, a combinational circuit for reversible fault tolerant 2×2 logical AND operation cannot be realized with less than three quantum cost.

The above example clarifies the lower bound in terms of quantum cost of 2-to-4 RFD. Similarly, it can be proved that the n-to-2^n RFD can be realized with $5(2^n-\frac{8}{5})$ quantum cost, when $n\geq1$, and by assigning different values to n, the validity of this equation can be proved.

4.4. Proposed reversible fault tolerant 4n-to-n multiplexer for FPGA

Section 3 shows that the Plessey logic block of FPGA requires an 8-to-2 Mux. A Mux is a device that sends multiple signals on a carrier channel at the same time in the form of a single complex signal to another device that recovers the separate signals at the receiving end. In other words, Mux selects one of many analog or digital data sources and outputs that source into a single channel.

[3] In this table and all the following tables, it is considered that, E = even, O = odd, I_p = input parity, O_p = output parity, and CI = constant input.
[4] AND means a two-input logical AND.

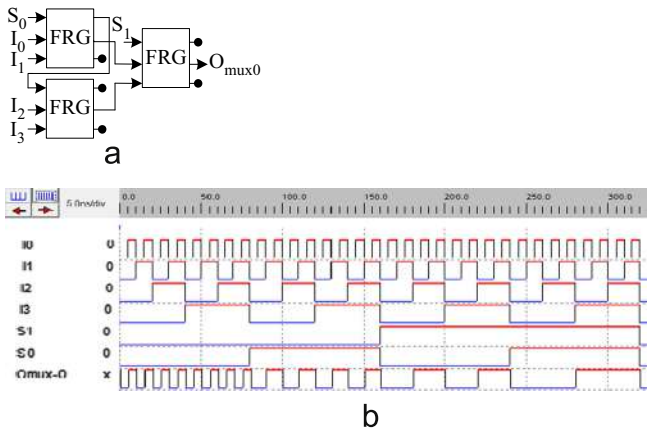

Fig. 16. Proposed 4-to-1 RFMux (a) architecture and (b) timing diagram.

Thus, Mux is also known as a data selector. This subsection elaborately proposes algorithm, design and working procedure of 4*n*-to-*n* *R*eversible *F*ault tolerant *Mux* (RFMux) for the FPGA. From now on, we denote a reversible fault tolerant Mux as RFMux.

Considering the simplest case, $n=1$, we have a 4-to-1 RFMux. Fig. 16(a) shows the architecture of the proposed 4-to-1 RFMux for the FPGA. The corresponding timing diagram of the proposed 4-to-1 RFMux is shown in Fig. 16(b). From Fig. 16(b), we find that, when both control signals are low, the final output O_{mux0} follows I_0 (from the start to 80 ns). However, when control signal $S_0=1$ and $S_1=0$, the final output O_{mux0} follows I_1 (80–160 ns), when control signal $S_0=0$ and $S_1=1$, the final output O_{mux0} follows I_2 (160–240 ns) and so on.

Algorithm 2. Algorithm for the proposed reversible fault tolerant 4*n*-to-*n* multiplexer for FPGA, ***RFMux***(***I***,***S***,***F2G***,***FRG***).

Input: Data input $I(I_0,I_1,\ldots,I_{4n-1})$, Control input $S(S_0,S_1,\ldots,S_n)$
n and Fredkin gate (*FRG*)
Output: 4*n*-to-*n* RFMux circuit
1 **begin**
2 *i*=*input*, *o*=*output*, *j*=0
3 **begin procedure** (4-to-1 RFMux)
4 $S_j \rightarrow$*first.i.FRG*.1&2, $I_j \rightarrow$*second.i.FRG*.1
5 $I_{j+2} \rightarrow$*second.i.FRG*.2$I_{j+1} \rightarrow$*third.i.FRG*.1
6 $I_{j+3} \rightarrow$*third.i.FRG*.2, *j*++, $S_j \rightarrow$*first.i.FRG*.3,
7 *second.o.FRG*.1 → *second.i.FRG*.3,
8 *second.o.FRG*.2 → *third.i.FRG*.3,
9 **return** *second.o.FRG*.3 → *desired output*
10 *remaining.FRG.o* → *garbage output*.
11 **end procedure**
12 **for** *i*←0 **to** *n*−1 **do**
13 **goto** line 3
14 **end for**
14 **return** 4*n*-to-*n* *RFMux circuit*
15 **end**

From Fig. 16(a), we find that the 4-to-1 RFMux produces 5 garbage outputs, whereas no constant input is needed. These are the minimal garbage outputs and constant input for the 4-to-1 RFMux which is proved below using Theorem 3 and Example 3. Table 8 shows the comparison of the 4-to-1 proposed Mux and the existing Muxs [22,23]. The bottom row of the table shows the improvements in the proposed design in percentage with respect to existing designs. In this table and all the following tables, the improvement ratio $(IR)=LVE-VPD/LVE \times 100\%$, where,

Table 8
Comparison of 4-to-1 reversible Mux.

	GT	GO	QC	UD	TR
Existing design [22]	9	11	57	6	110
Existing design [23]	7	11	57	7	110
Proposed design	3	5	15	3	12
Improvement ratio (*IR*) (%)	> 57	> 54	> 73	≥ 50	> 89

LVE=*L*owest *V*alue between the *E*xisting designs and *VPD*=*V*alue of the *P*roposed *D*esign. Algorithm 2 represents the design procedure of the proposed 4*n*-to-*n* RFMux. Initially, the algorithm builds a 4-to-1 RFMux circuit, then recursively builds a 4*n*-to-*n* RFMux using the procedure of 4-to-1 RFMux.

Theorem 3. *A 4n-to-n reversible Mux requires at least*$(4n+1)$ *garbage outputs and no constant inputs, where n is number of data bits.*

Proof. Let *n* be the number of data bits. Then, for a 4*n*-to-*n* Mux, there are 4*n* data inputs and $(n+1)$ select inputs. So, to preserve the one-to-one mapping of reversibility, there should be at least $(5n+1)$ outputs, among which there are *n* primary outputs. Hence, there should be at least $(4n+1)$ garbage outputs for 4*n*-to-*n* reversible Mux, which causes no constant input. □

Example 3. Let the value of *n* be 1. Then, we have 4-to-1 Mux. The 4-to-1 Mux has four data inputs (I_0, I_1, I_2 and I_3) and two control inputs (S_0 and S_1). So, the total number of inputs is six. To maintain one-to-one correspondence, the reversible 4-to-1 Mux must have at least six outputs. Among these six outputs, only one is the primary output. Thus, there are at least five outputs that remain unused, *i.e.*, garbage outputs. If we replace *n* with 1 in Theorem 3, we get five garbage outputs as well. From Fig. 16(a), we also find that the proposed 4-to-1 RFMux has five garbage outputs. Next, since it is possible to realize the 4-to-1 Mux reversibly with six outputs and thus it will not require any constant input. Circuitry of Fig. 16(a) also has no constant input. So, the proposed 4-to-1 RFMux (Fig. 16(a)) is the optimum design of reversible Mux in terms of garbage outputs and constant input as it adapts Theorem 3. Because the 4-to-1 RFMux is used in 4*n*-to-*n* RFMux as a main a procedure which is recursively called *n* times, thus the above optimality statement is relevant to any 4*n*-to-*n* RFMux.

Lemma 3. *Let n be the number of data bits. Then, a 4n-to-n RFMux for FPGA can be realized with* 3*n Fredkin gates*, 15*n quantum cost and* $(6n\alpha+12n\beta+6n\gamma)$ *hardware complexity.*

4.5. Proposed reversible fault tolerant random access memory for FPGA

The *R*andom *A*ccess *M*emory (RAM) for the FPGA can be considered as an array of individual memory cells. More specifically, there are 2^n rows where each row contains *m* master-slave flip-flops. To address these rows and columns a decoder is needed. In the decoder, only one out of 2^n outputs is SET. The write bit *W* determines whether a read or a write operation is to be performed. The SET output of the decoder selects one row of the flip-flops. If *W* is SET, then *m* flip-flops of the selected row are written with the inputs D_0 to D_{m-1}. On the other hand, if *W* is LOW, Q_0 to Q_{m-1} holds the previously stored bits in the flip-flops of the selected row and are refreshed with the stored bits.

We have proposed a reversible RAM in [24]. This design consists of an *n*-to-2^n reversible decoder, Toffoli gates, write enable master-slave flip-flops, 3×3 and $2^n\times2^n$ (2^n input bits)

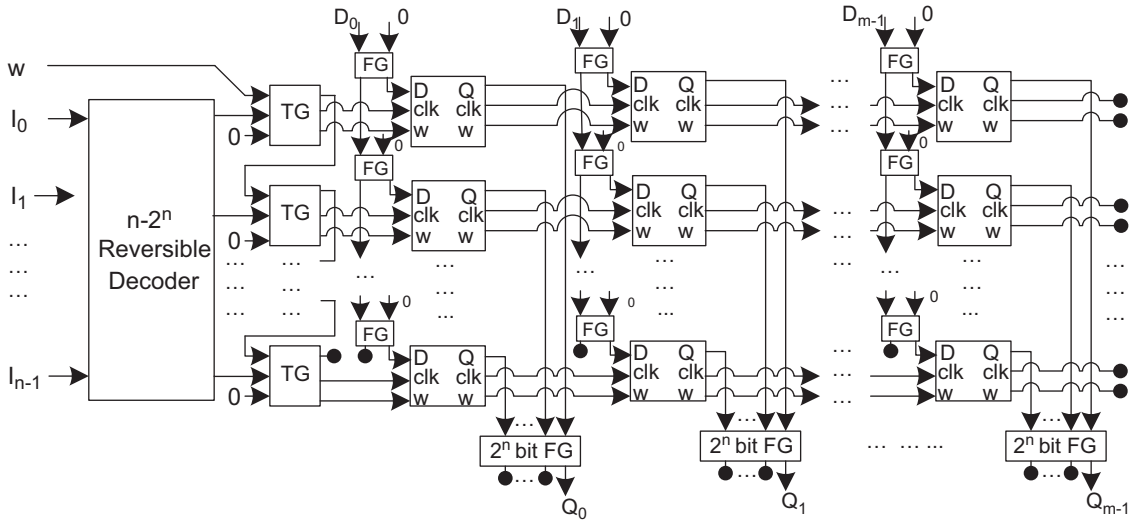

Fig. 17. Architecture of our proposed reversible RAM [24].

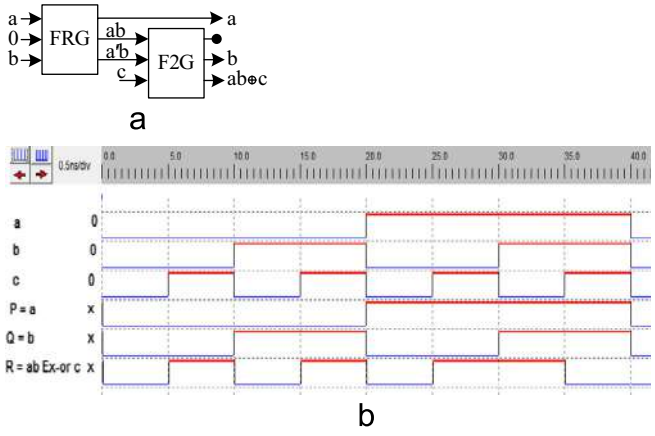

Fig. 18. Proposed fault tolerant realization of Toffoli gate (a) architectural block diagram and (b) timing diagram.

Feynman gates. To design the *R*eversible *F*ault tolerant *RAM* (RFRAM) for the FPGA, we extend this reversible design [24] to reversible fault tolerant design. The extension is straight forward, we transformed each of the reversible component [24] to reversible fault tolerant components with more optimality. From now on, we denote a reversible fault tolerant RAM of the FPGA as RFRAM. We have already projected a compact design of reversible fault tolerant write enable master-slave flip-flop in Section 4.2. In addition to that, an optimal design of reversible fault tolerant decoder is presented in Section 4.3 (Algorithm 1). Now, replacing Feynman gate (*FG*) of Fig. 17 with Feynman double gate (*F2G*), the task remains how to realize the reversible Toffoli gate[5] so that it can preserve parity. Fig. 18(a) represents the architecture of a *F*ault *T*olerant realization of reversible *T*offoli *G*ate, hence named as FTTG. The corresponding timing diagram of FTTG is shown in Fig. 18(b).

From Fig. 18(a), we find that the fault tolerant realization of a Toffoli gate requires an *FRG* and an *F2G*. We also find that, this circuit requires one constant input, produces one garbage output, and its quantum cost is 7. In the following theorems, we prove that, if we want to realize a compact reversible Toffoli gate with parity preserving mode, these numbers are minimal.

[5] Toffoli gate means 3 ×3 reversible Toffoli gate unless mentioned explicitly.

Table 9
Truth table of a reversible Toffoli gate.

Inputs			Outputs			Parity
a	*b*	*c*	*P*	*Q*	*R*	
0	0	0	0	0	0	$I_p=O_p$
0	0	1	0	0	1	$I_p=O_p$
0	1	0	0	1	0	$I_p=O_p$
0	1	1	0	1	1	$I_p=O_p$
1	0	0	1	0	0	$I_p=O_p$
1	0	1	1	0	1	$I_p=O_p$
1	1	0	1	1	1	I_p=E, O_p=O
1	1	1	1	1	0	I_p=O, O_p=E

Theorem 4. *The fault tolerant realization of reversible Toffoli gate requires at least one garbage output and one constant input.*

Proof. Input–output combinations of reversible Toffoli gate do not preserve the parity. To preserve the parity at the output level, at-least one more output is needed, which demands at-least one constant input at the input level. Thus, fault tolerant realization of reversible Toffoli gate requires at least one garbage output and one constant input. □

Example 4. Fig. 18(a) is a straight forward proof for the existence of fault tolerant circuit which can realize the reversible Toffoli gate with one garbage output and one constant input. Next, we must demonstrate the nonexistence of the reversible fault tolerant circuit which can realize the reversible Toffoli gate without the garbage output and without the constant input. Input and output vectors for reversible Toffoli gate (TG) are $I_v=(a,b,c)$ and $O_v=(P=a, Q=b, R=ab\oplus c)$. The corresponding truth table for these input–output combinations is shown in Table 9.

From Table 9, we find that the input combinations 1-1-0 and 1-1-1 (last two rows) have different parity at the input and output levels. This result establishes the nonexistence of the reversible fault tolerant circuit which can realize the Toffoli gate without the garbage output and without the constant input.

Theorem 5. *A fault tolerant Toffoli gate can be realized with at least seven quantum cost.*

Proof. Theorem 4 and Example 4 are the proof for the nonexistence of reversible Toffoli gate with 5 quantum cost in parity

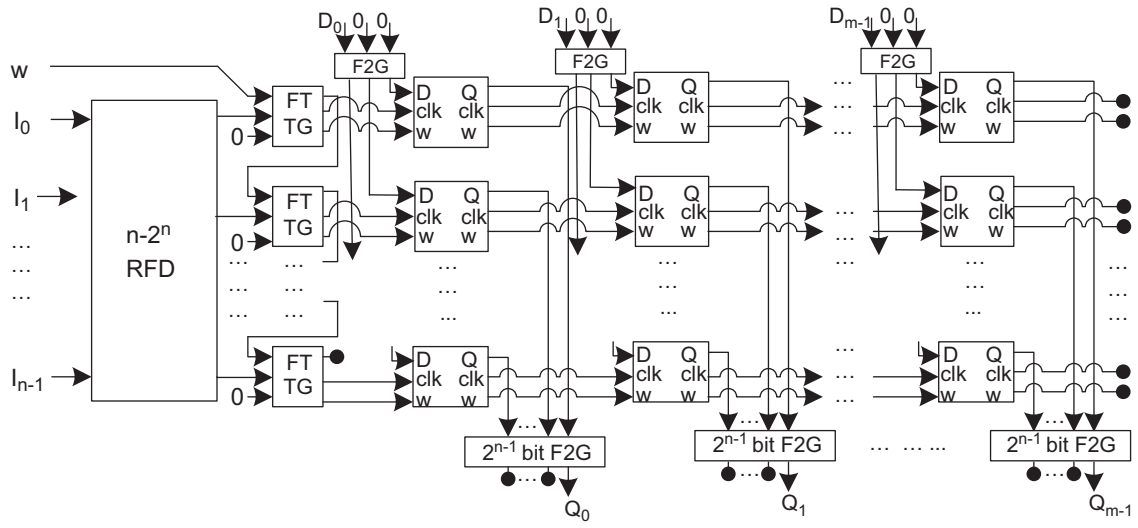

Fig. 19. Architecture of the proposed RFRAM for FPGA.

preserving fashion. Here, it is also not possible to preserve the parity at the output level with six quantum cost. However, seven 2×2 primitive gates can realize reversible Toffoli gate while preserving parity. Thus, at least seven quantum cost are required to realize a reversible fault tolerant Toffoli gate. □

Example 5. Fig. 18(a) is a straight forward proof for the existence of fault tolerant circuit which can realize the reversible Toffoli gate by seven quantum cost. Next, we must demonstrate the nonexistence of the reversible fault tolerant circuit which can realize the reversible Toffoli gate with less than seven quantum cost. The quantum equivalent realization of a reversible Tofoli gate consists of five 2×2 primitive gates as shown in [7,12,22,24]. Example 8 shows that this realization does not preserve the parity. By using Theorem 2 and Example 2 (Section 4.3), it can be shown that no matter whether we use one more primitive gate or six primitives gates to realize a Toffoli gate, the corresponding circuit will not be able to maintain the input–output combinations or violates the parity preserving property. More specifically, either it will violates the one-to-one correspondence of reversibility or fail to adapts with Eq. (1), *i.e.*, unable to preserve the parity. This shows the nonexistence of the claim assumed above.

At this point, we have designed reversible fault tolerant circuitry of each component of the proposed RFRAM for the FPGA. Combining all these components the architectural block diagram of the proposed RFRAM is shown in Fig. 19. Algorithm 3 demonstrates the design procedures of the proposed RFRAM. Primary inputs to the algorithm are data input sets $I(I_0,I_1,\ldots,I_{n-1})$, $D(D_0,D_1,\ldots,D_{m-1})$ and write enable bit W.

Algorithm 3. Algorithm for the proposed reversible fault tolerant random access memory for FPGA, ***RFRAM*(*I*,*D*,*W*,*F2G*,*FRG*)**.

Input: Data $I(I_0,I_1,\ldots,I_{n-1})$, $D(D_0,D_1,\ldots,D_{m-1})$, W, FRG, $F2G$
Output: reversible fault tolerant RAM circuit for FPGA

```
1  begin
2    i=input, o=output
3    call RFD(I, F2G, FRG)
4    for k←0 to n−1 do
5      if k=0 then W→first.i_k.FTTG end if
6      else first.o_{k−1}.FTTG→first.i_k.FTTG end if
7      RFD.o_k→second.i_k.FTTG, 0→third.i_k.FTTG
8    end for
9    for k←0 to m−1 do
10     D_k→first.i_k.F2G, 0→second & third.i_k.F2G
11     for l←1 to n/2 do
12       F2G.o_k→first.i_l.F2G, 0→second & third.i_l.F2G
13     end for
14     for l←0 to n−1 do
15       for D_k(F2G.o_k l)→first.RFMSFF.i
16       x=second & third
17         if l=0 then x.FTTG.o_l→x.RFMSFF.i_l end if
18         else x.RFMSFF.o_{l−1}→x.RFMSFF.i_l end if
19       first.RFMSFF.o.1→first.2^{n−1} bitF2G.i
20     end for
21   end for
22   return
23   for j←0 to m−1 do
24     2^{n−1} bitF2G.o.2^{n−1}→desired output
25   end for
26   all remaining F2G.o,FRG.o &
     2^{n−1} bitF2G.o→garbage output.
27 end
```

From the architectural block diagram of Fig. 19, we find that the proposed RFRAM for FPGA demands an underneath n-to-2^n reversible fault tolerant decoder (RFD) circuitry. Thus, a call to RFD[6] is specified by line 3 in this proposed algorithm. Lines 4–13 assign inputs to fault tolerant Toffoli gates according to our design procedure of reversible fault tolerant random access memory for the FPGA. The inputs to *F2G*s and *RFMSFF*s are assigned by the lines 14–21. From Fig. 19, we also find that there are m number of primary outputs which are returned by 2^{n-1} bit Feynman double gates. In Algorithm 3, these outputs are returned by the line 24 whereas line 26 finds all garbage outputs of RFRAM. Time complexity of Algorithm 3 depends on the For loops and there are at-most two levels of dependency among them, n then m or vice versa. Thus, its complexity is $O(mn)$.

Lemma 4. *An RFRAM for the FPGA is realized with* $(2^n+2n+15\,\mathrm{nm}/2+m-1)$ *reversible fault tolerant gates,* $(2n+(m(2^{n-2}$

[6] Algorithm 1 (*RFD*(*I*,*F2G*,*FRG*)) presents detailed design procedure of n-to 2^n RFD.

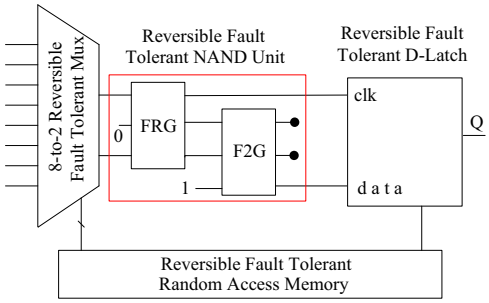

Fig. 20. Proposed reversible fault tolerant plessey logic block for FPGA.

$+1+7n)$) *garbage outputs*, $(2^n+9\,\mathrm{nm}+n)$ *constant inputs*, $(2^n(m/4+5)+24\,\mathrm{mn}+7n-8)$ *quantum cost and* $(2^{n-2}(m+8)+15\,\mathrm{mn}+4n-2)\alpha+(2^{n+1}+6\,\mathrm{mn}+2n-4)(2\beta+\gamma)$ *hardware complexity, where,* α, β *and* γ *are the hardware complexity of two-input Ex-OR, AND and NOT calculations, respectively, and* $n,m \in N$, *the set of positive integers and* $nm \geq 2$.

4.6. Proposed reversible fault tolerant plessey logic block for FPGA

The logic blocks of FPGA are connected through programmable interconnects or wire segments. Wire segment consists of programmable switches. Generally, one or two switches are attached in a segment and each end of a segment has a switch attached to it. A routing channel is a group of parallel tracks. The track consists of one or more segment(s) arranged in a sequence which is shown earlier in Figs. 7(a)–(c) (Section 3.1). Fig. 20 shows the block diagram of the proposed reversible fault tolerant Plessey logic block of static RAM based FPGA. This architecture consists of a reversible fault tolerant D-latch (FTD), 8-to-2 multiplexer (RFMux), random access memory (RFRAM) and an extra Fredkin and Feynman double gates. The FTD, RFMux and RFRAM have already been proposed in Sections 4.1, 4.4 and 4.5, respectively. Here, the extra Fredkin and Feynman double gates are needed to implement the NAND unit of Plessey logic block. First input to the Fredkin gate of this NAND unit comes from first 4-to-1 RFMux where third input to the Fredkin gate of this NAND unit comes from second 4-to-1 RFMux, *i.e.*, inputs to this NAND unit are simply the outputs of the 8-to-2 RFMux. Inputs of each RFMux are either connected to the output of the previous NAND unit in the row or the output of the NAND unit below or above this block (to the closer one). From Fig. 20, we find that the proposed NAND unit of Plessey logic block produces two garbage outputs. This is the optimum number of garbage outputs for the two-input NAND unit as proved in the following theorem.

Theorem 6. *At least two garbage outputs are required to realize a two-input NAND operation in reversible and fault tolerant mode.*

Proof. In a two-input (a,b) NAND operation, there is one primary output $(ab)'$, thus it has at least one garbage output. These output combinations do not preserve the one-to-one mapping of reversibility with respect to the input combinations, hence an additional garbage output is required, which also preserves parity. Thus, a reversible fault tolerant two-input NAND operations is realized with at least two garbage outputs. □

Example 6. Let the inputs are (a,b) for the two-input NAND operation. Then, the output of two-input NAND operation is $(ab)'$. According to the property of reversibility, there should be equal number of inputs and outputs, otherwise it will never be able to maintain the one-to-one correspondence. Therefore, there should be at least one more output. Let this output be q. This $(a,b)\leftrightarrow((ab)',q)$ combinations have one garbage output namely q. Truth table of this input–output combinations is shown in Table 10.

Table 10
Two input NAND operation with one garbages.

a	b	$(ab)'$	q
0	0	1	
0	1	1	
1	0	1	
1	1	0	

Table 11
Two input NAND operation with two garbages.

Inputs			Outputs			Parity
a	b	CI	$(ab)'$	q	r	
0	0	1	1	0	0	$I_p=O_p=O$
0	1	1	1	1	0	$I_p=O_p=E$
1	0	1	1	0	1	$I_p=O_p=E$
1	1	1	0	1	0	$I_p=O_p=O$

According to the Table 10, it is clear that there are three identical outputs at the output level at $(ab)'$ (first three cells) for three different input combinations, which result's whatever the value of q, the output combinations of Table 10 do not able to maintain the one-to-one mapping of reversibility with respect to the input combinations. Hence, an additional garbage output is required. Thus, a reversible two input NAND operation requires at least two garbage outputs. Next, we must prove that this two input NAND operation with two garbage outputs maintain the fault tolerant property of Eq. (1), *i.e.*, it can preserve the parity for all input–output combinations. Now, we have one primary output and two garbage outputs, *i.e.*, total of three outputs but two inputs. So, there should be at least one constant input. Let the later garbage output be r and constant input (CI) be 1. Table 11 shows how this input–output combinations preserve the parity.

Algorithm 4 presents a formal representation of the proposed reversible fault tolerant Plessey logic block for FPGA. From Fig. 20, we find that, fault tolerant Plessey logic block considerably depends on RFRAM, thus a call to RFRAM is specified in line 3. The outputs of RFRAM are the selection bits for 8-to-2 RFMux. Thus, it calls $4n$-to $-n$ RFMux with $n=2$,[7] which is defined in line 4 of Algorithm 4. On the other hand, outputs of RFMux are the inputs to the NAND unit (shown in Fig. 20). Line 5 of the algorithm assigns input to the reversible fault tolerant NAND unit and D-latch of Plessey logic block for FPGA. Line 6 returns the primary output Q which actually is the input to the next logic block. Finally, line 7 returns all the garbage outputs of a reversible fault tolerant Plessey logic block for FPGA. As mentioned in Sections 1 and 3, there are clusters of logic blocks in the FPGA, but the above procedure is capable to build a single reversible fault tolerant Plessey logic, thus, to iterate the entire procedure till the last logic block, a recursive call has been specified in line 8 of this algorithm.

Algorithm 4. Algorithm for the proposed reversible fault tolerant Plessey logic block of the FPGA.

Input: Data input sets I,S,D, Write enable bit (W)
Fredkin gate (FRG) and Feynman double gate $(F2G)$
Output: Reversible fault tolerant Plessey logic block's circuit

[7] Algorithm 2 presents a detailed design procedure of $4n$-to-n RFMux.

1 begin
2 *i=input, o=output*
3 **call** *RFRAM(I,D,W,F2G,FRG)*
4 **call** *RFMux(I,S,2,FRG)*; where *S←RFRAM.o*
5 *RFMux.o.1→FRG.i.1, 0→FRG.i.2,*
RFMux.o.2→FRG.i.3; FRG.o.2→F2G.i.1,
FRG.o.3→F2G.i.2, 1→F2G.i.3; FRG.o.1→D−Latch.i.1,
F2G.o.3→D−Latch.i.2
6 **return** *D−Latch.o →Q (input for next logic block)*
7 *remaining F2G.o & FRG.o → garbage output.*
8 **goto** *line 2* **if** *this is not the last logic block*
9 **end for**

Lemma 5. *Let GT, GO, CI, QC, and HC be the required numbers of gates, garbage outputs, constant inputs, quantum cost and hardware complexity for a reversible fault tolerant Plessey logic block of FPGA. Also let* gt_{RAM}, go_{RAM}, ci_{RAM}, qc_{RAM} *and* hc_{RAM} *be the required numbers of gates, garbage outputs, constant inputs, quantum cost and hardware complexity for an RFRAM of the FPGA. Then*

$GT \geq 10+gt_{RAM}$
$GO \geq 13+go_{RAM}$
$CI \geq 4+ci_{RAM}$
$QC \geq 44+qc_{RAM}$
$HC \geq 20\alpha+32\beta+16\gamma+hc_{RAM}$

4.7. Proposed reversible fault tolerant input–output block for FPGA

Basic input–output block of Xilinx is one of the most popular as it maintains a hierarchical structures while providing the interface between the package pins and control blocks [23]. Moreover, it can work in uni- or bi-directional modes. Fig. 21 shows the block diagram of the proposed reversible fault tolerant input–output block of Xilinx FPGA without the SET/RESET and flip-flop enable options. The explanations to exclude these options from the proposed circuit have already been presented in Section 3.1.

From Fig. 21, we find that in the proposed input–output block, there are two *FRG*s and each one of them produces two garbage outputs. Here, each *FRG* works as 2-to-1 Mux. The following theorem proves that, these are the minimal number of garbage outputs of a 2-to-1 reversible fault tolerant Mux.

Theorem 7. *A 2-to-1 reversible fault tolerant Mux requires at least two garbage outputs.*

Proof. In a 2-to-1, Mux there are two primary inputs and one selector input. Thus, according to the property of reversibility, there should be at least three outputs. Among these three outputs, only one is the primary output. This input–output combination preserves parity too. Thus, a reversible fault tolerant 2-to-1 Mux is realized with at least two garbage outputs. □

From Fig. 21, we find that there are three *F2G*s and three *D* flip-flops. The D flip-flop functions quite differently than the D-latch. However, both are identical without the options of SET/RESET and enables of flip-flop. Thus, in Fig. 21, proposed reversible fault tolerant D-latch (Section 4.1) has been used. In the proposed architecture of input–output block, a *R*eversible *F*ault *T*olerant *I*nput *B*uffer (RFTIB) and a *R*eversible *F*ault *T*olerant *O*utput *D*river (RFTOD) have also been used. Table 12 presents the performance of the proposed input–output block for the FPGA with the existing one [23]. In Section 3, it is evidenced that the attempts toward the reversible input–output block for the FPGA have not been completed [23]. However, Table 12 evidenced that the proposed design is not only complete and but also performs much better than its existing counterpart. Moreover, the proposed input–output block can detect the fault signal in its primary outputs, which is not at-all present in the existing design.

Lemma 6. *A reversible fault tolerant input–output block of the FPGA can be realized with* 12 *gates*, 13 *garbage outputs*, 14 *constant inputs*, 39 *quantum cost and* $24\alpha+20\beta+10\gamma$ *hardware complexity.*

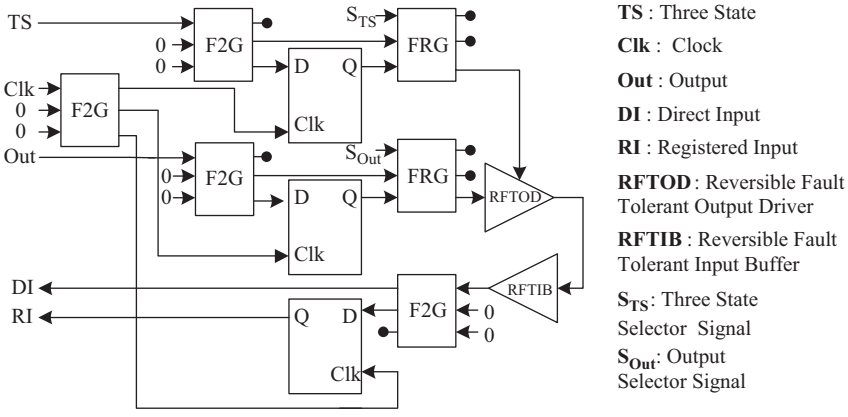

Fig. 21. Proposed reversible fault tolerant basic input–output block for FPGA.

Table 12
Comparison of the reversible input–output block.

	GT	GO	QC	CI	HC	UD	TR
Existing design [23]	20	20	64	17	$34\alpha+40\beta+20\gamma$	6	124
Proposed design	12	13	39	14	$24\alpha+20\beta+10\gamma$	4	104
Improvement ratio (*IR*) (%)	40	35	≥39	>17	>29	>33	>16

5. Performance evaluation of the proposed method

There are two variables n and m that fix the structure of the FPGA. Thus, the overall comparative results are shown in contrast with three possible design approaches: (i) varying m while keeping n constant; (ii) varying n while keeping m constant; and (iii) varying both m and n.

5.1. Gate comparison

Fig. 22 presents the performance of the proposed reversible fault tolerant FPGA with respect to the existing reversible FPGAs in terms of number of gates. Fig. 22(a)–(c) represents the above mentioned (i), (ii) and (iii) design approaches, respectively. The plotted graphs of Fig. 22 show that the proposed scheme outperforms the existing works.

5.2. Garbage outputs comparison

Fig. 23 presents the performance of the proposed FPGA with respect to the existing FPGAs in terms of garbage outputs. With respect to the garbage outputs produced, the above mentioned (i), (ii) and (iii) design approaches are shown in Fig. 23(a)–(c), respectively. Fig. 23 shows that the proposed design performs much better than existing works for all possible circumstances.

Most interestingly, above two figures show that, for the last two cases, existing designs increase exponentially, whereas the proposed design increases linearly with respect to the number of inputs.

5.3. Constant inputs comparison

The quantum circuits with many constant bits which have the reversible gates as a building block are unmanageable. Thus, constant input is considered as a major overhead and is needed to be minimized. Fig. 24(a)–(c) presents the performance of the proposed FPGA with respect to the existing FPGAs in terms of required number of constant inputs for the above mentioned (i), (ii) and (iii) design approaches, respectively. The graphs of Fig. 24 show that the slope (growth rate) of all the edges are almost equal, but the performance of the proposed design is the best among them.

5.4. Delay comparison

The delay of a logic circuit is the delay of the critical path. However, it is an NP complete problem to find all the critical path specially for large circuits, for large n or m in case of FPGA. Thus, researchers used to select the path which is the most likely candidates for the critical paths. The delay of the proposed circuit is evaluated considering the most probable critical paths. Fig. 25 presents the performance of the proposed work with respect to the existing work [21,22] in terms of the delay of the circuit. From Fig. 25, we find that the proposed design performs better than the existing designs for all possible circumstances.

5.5. Quantum cost comparison

Fig. 26 compares the performance of the proposed FPGA with respect to the existing works in terms of total quantum cost of the circuits. Fig. 26(a)–(c) represents the performance for above mentioned cases (i)–(iii), respectively. The graphs of Fig. 26 show that, when the number of inputs increases, the design [21] performs better than the design [22] for case (i), but not for cases (ii) and (iii). However, the proposed scheme outperforms the existing schemes [21,22] for all possible circumstances by a huge margin. Moreover, for the last two cases, the rate of changes with respect to the quantum cost in existing designs are exponential, whereas the proposed design changes polynomially.

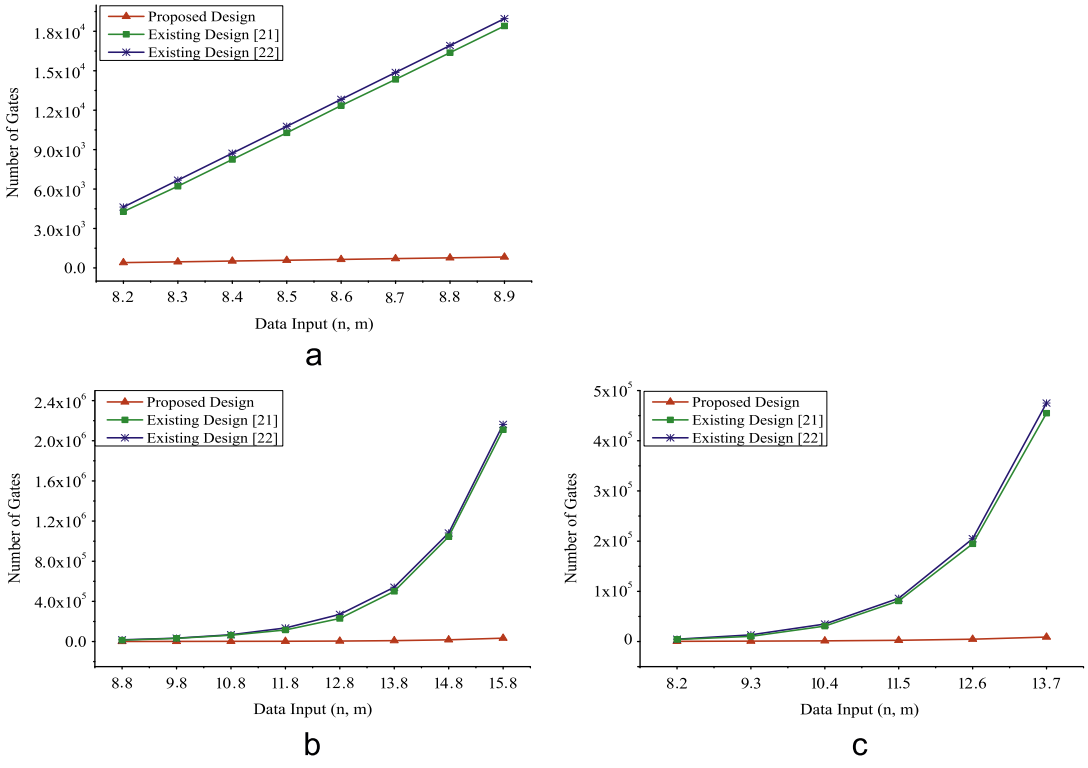

Fig. 22. Performance of the proposed work and existing works [21,22] with respect to the number of gates. (a) Varying m while keeping the constant n. (b) Varying n while keeping the constant m. (c) Varying both m and n.

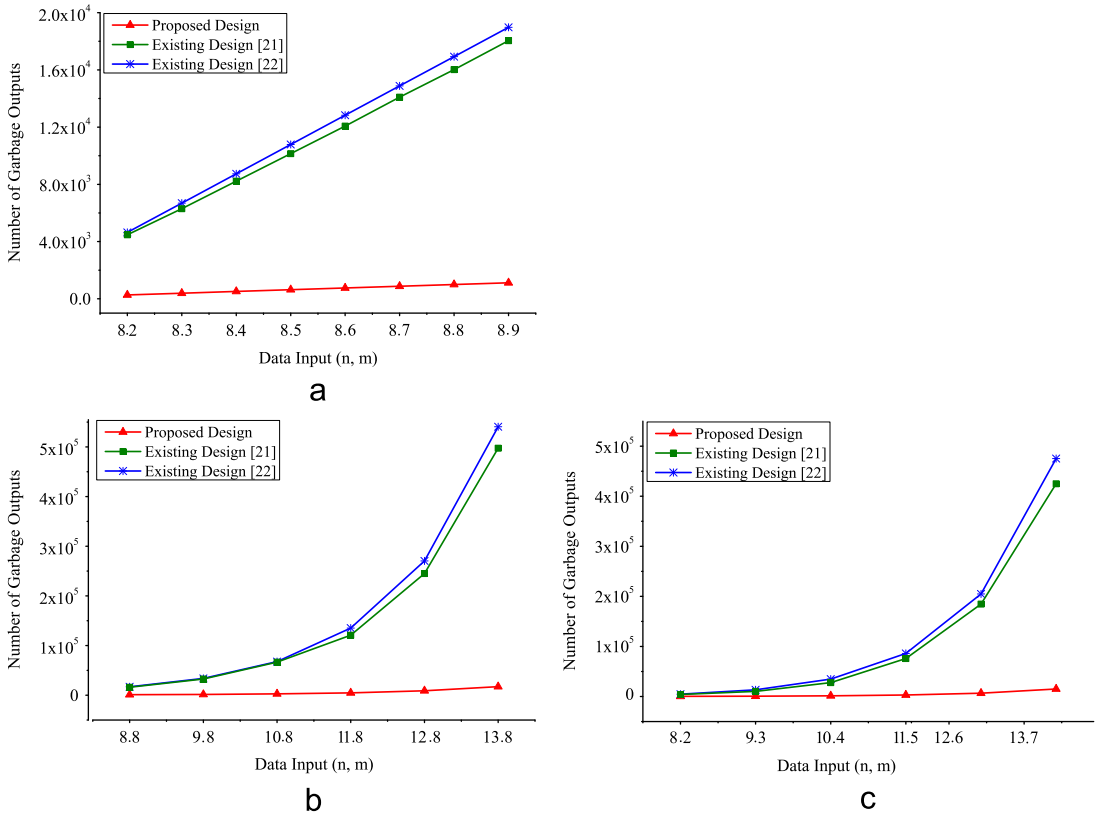

Fig. 23. Performance of the proposed work and existing works [21,22] with respect to the garbage outputs. (a) Varying m while keeping the constant n. (b) Varying n while keeping the constant m. (c) Varying both m and n.

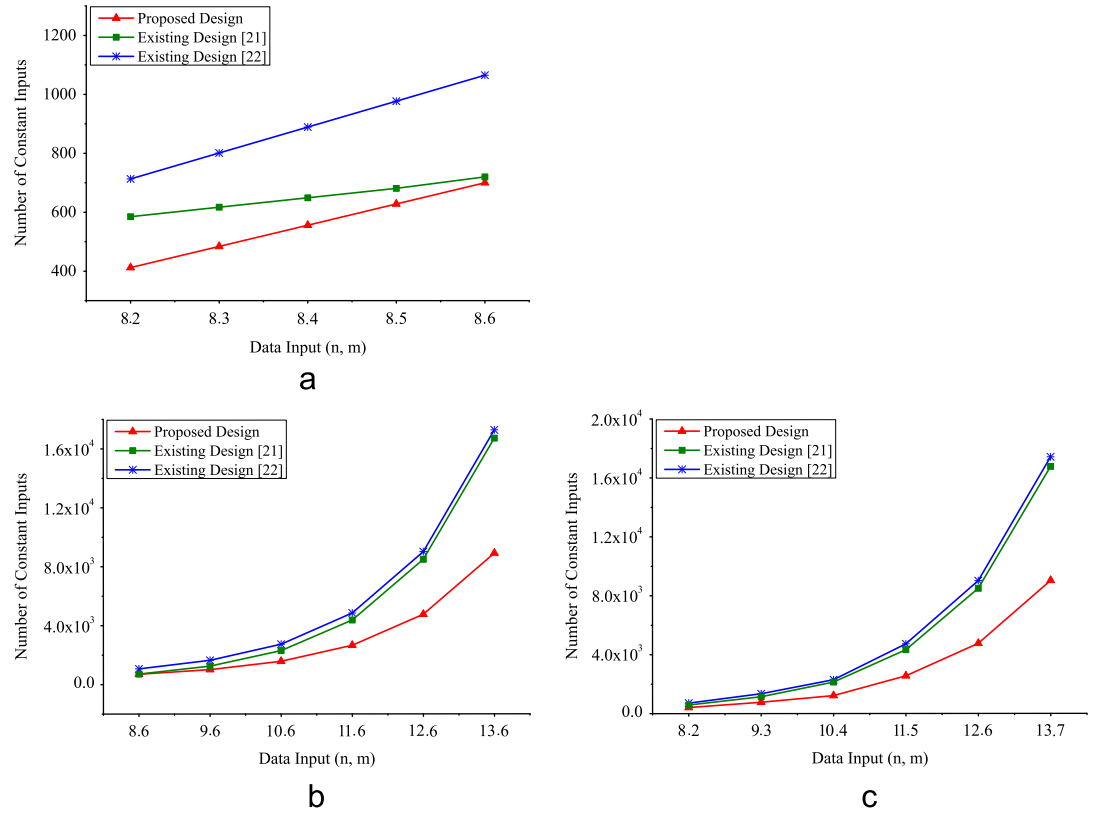

Fig. 24. Performance of the proposed work and existing works [21,22] with respect to the constant inputs. (a) Varying m while keeping the constant n. (b) Varying n while keeping the constant m. (c) Varying both m and n.

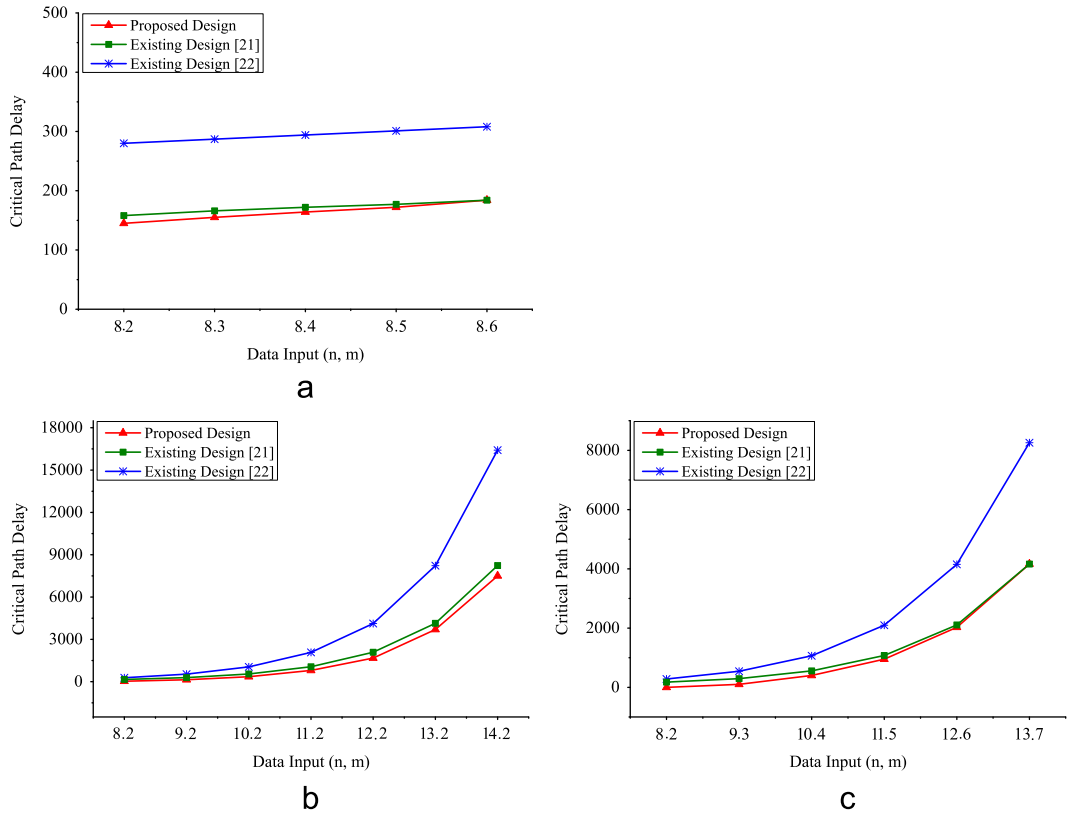

Fig. 25. Performance of the proposed work and existing works [21,22] with respect to the critical path delay. (a) Varying *m* while keeping the constant *n*. (b) Varying *n* while keeping the constant *m*. (c) Varying both *m* and *n*.

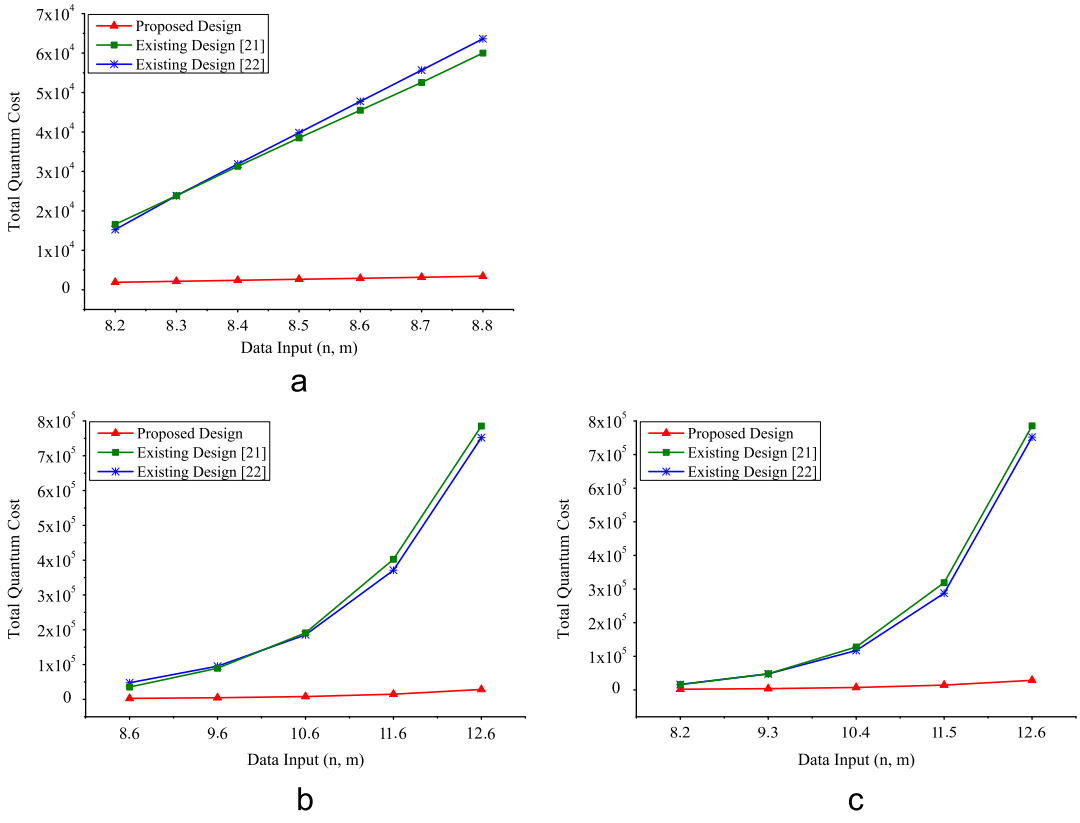

Fig. 26. Performance of the proposed work and existing works [21,22] with respect to the quantum cost. (a) Varying *m* while keeping the constant *n*. (b) Varying *n* while keeping the constant *m*. (c) Varying both *m* and *n*.

Table 13
Comparative study with respect to hardware complexity.

(m, n)	Proposed design	Existing design [21]	Existing design [22]
(8, 3)	1114α	2876α	2224α
	1368β	1159β	931β
	684γ	563γ	404γ
	3166	4598	3559
(8, 4)	1298α	2972α	2615α
	1464β	1199β	1059β
	732γ	579γ	452γ
	3494	4750	4126
(10, 4)	3730α	12796α	8897α
	4640β	4313β	2725β
	2320γ	2131γ	1268γ
	10,690	19,240	12,890
(10, 5)	4136α	12916α	10090α
	4760β	4363γ	2885β
	2380β	2151γ	1328γ
	11,276	19,430	14,303

5.6. Hardware complexity comparison

Finding a critical path from the large circuit is not always easy. Thus, the researchers calculate the hardware complexity from the circuit. Generally, a constant complexity is assumed for each basic operation and the number of operations required to realize the circuit are calculated first. For example, let α, β and γ be the hardware complexity of the two-input Ex-OR, AND and NOT calculations, respectively. Let the hardware complexity of gates A and B of our running example be $(E\alpha+F\beta+G\gamma)$ and $(H\alpha+I\beta+J\gamma)$, respectively. Then, the hardware complexity of the circuit is $(NE\alpha+NF\beta+NG\gamma)+(MH\alpha+MI\beta+MJ\gamma)= (NE+MH)\alpha+(NF+ MI)\beta +(NG+MJ)\gamma$. Table 13 compares the performance of the proposed design with respect to existing work in terms of hardware complexity according to these above equation. Here, the summations of total number of operations are shown at the bottom of each row. Performance of the proposed method with respect to hardware complexity for case (i) is shown in row (1,2) or (3,4). Row (2,3) shows the performance for case (ii) and the row (1,3) or (1,4) or (2,4) of this table show the performance for case (iii).

6. Conclusions

This paper presented the design methodologies of reversible fault tolerant FPGA. In the customized VLSI, FPGA provides low time-to-market and low costs compared to application specific integrated circuits and mask programmable gate arrays. The FPGA consists of an array of programmable logic blocks, input–output cells and interconnections. The programmability of logic blocks enables multiple processes to run it without time sharing. Plessey and look-up table (LUT) are the most popular logic blocks of an FPGA. With more inputs, an LUT can implement more logic, hence fewer logic blocks are needed. This saves routing area, however the complexity grows exponentially with the number of inputs. Plessey logic block overcomes this difficulty through cluster of components such as NAND unit, RAM, Mux and latches. The RAM requires an n-to-2^n decoder and a master-slave flip-flops.

6.1. Summary and contributions

This paper proposed the algorithms to design a compact reversible fault tolerant n-to-2^n decoder, a $4n$-to-n Mux, a RAM and a Plessey logic block of the FPGA. In addition, reversible fault tolerant designs of D-latch, write enable master-slave flip-flop and input-output block have been presented. Several lower bounds on the numbers of garbage outputs, constant inputs and quantum cost of the reversible fault tolerant FPGA have also been proposed in Theorems 1–7. The examples followed these theorems clarifies the representative analysis in details. It has been evidenced that the proposed circuits are constructed with the optimum garbage outputs, constant inputs and quantum cost. The efficiency and supremacy of the proposed designs have been proved through several Lemmas. In order to implement the circuits of the fault tolerant FPGA using MOS transistors, the designs of the transistor representations of the individual gates of the proposed FPGA have been presented using standard p-MOS 901 and n-MOS 902 model with delay of 0.030 ns and 0.12 μm channel length. Then, considering the transistor level designs of the reversible fault tolerant gates as a schematic, all the proposed circuits are simulated. The simulations evidenced that the proposed fault tolerant circuits work correctly. The comparative results show that the proposed design is much better in terms of numbers of gates, garbage outputs, quantum cost, delay, hardware complexity and are more scalable than its counterparts [21–23]. Moreover, all the proposed designs have the capability of detecting errors at circuit's primary outputs which is absent in the existing circuits [21–23]. The proposed circuit provides a basis for quantum computation with its applications and solves the power dissipation problem of conventional irreversible circuits. On the other hand, the fault tolerant property of the proposed circuit solve the bit error problem. Moreover, proposed fault tolerant circuit can also be used in on-site hardware reconfiguration [36], logic emulation [37], network components designing [38], aerospace and defense systems designing [39], computer vision specially in medical imaging [40], speech recognition [41], inexpensive prototype development [42], and digital signal processing [43].

6.2. Future work

The proposed FPGA is primarily based on the Plessey logic block. An interesting future enhancement can be considered to design a reversible fault tolerant FPGA with multi-logic block capability. For example, Plessey and LUT based logic block in a single circuit that preserves bi-directionality and fault detection capability. In that case, more signals will be needed to choose among the options. The proposed Theorems 1, 3 and 7, proved the lower bounds on the numbers of garbage outputs and constant inputs for the reversible fault tolerant decoders and Muxs. A possible future work is the realization all the basic components of the arithmetic logic unit and the central processing unit with minimal garbage outputs and the constant inputs. Theorem 2 and Example 2 proved that the two-input logical AND operation requires at-least three quantum cost. Here, another interesting future work can be realizing all the basic and universal logical operations with minimal quantum cost. In that case, the similar theorems and examples are needed only for the logical OR, NOR and NAND operations, as the minimal quantum cost for the NOT and two-input Ex-OR are zero and one, respectively. Theorems 4 and 5 proved that the fault tolerant realization of a reversible Toffoli gate requires at least one garbage output, one constant input and seven quantum cost. Providing the similar property of other non-fault tolerant reversible gate is an interesting work. From Theorem 6 and corresponding Example, it can be found that at least two garbage outputs are required to realize two-input logical NAND operation in a reversible and fault tolerant mode. A similar proof is given in Example 2 for the two-input logical AND operation. Here, the possible future enhancement is to realize all the basic operations in fault tolerant modes with minimal garbage outputs and constant inputs.

References

[1] J.C. Maxwell, Theory of Heat, fourth ed., Green and Co, Longmans, 1875.
[2] L. Szilard, On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings maxwell's demon 2 entropy, Class. Quant. Inf. Comput. (1929) 840–856.
[3] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 183–191, http://dx.doi.org/10.1147/rd.53.0183.
[4] C.H. Bennett, Logical reversibility of computation, IBM J. Res. Dev. 17 (6) (1973) 525–532, http://dx.doi.org/10.1147/rd.176.0525.
[5] H.F. Chau, F. Wilczek, Simple realization of the Fredkin gate using a series of two-body operators, Phys. Rev. Lett. 75 (1995) 748–750, http://dx.doi.org/10.1103/PhysRevLett.75.748.
[6] M. Nielsen, I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, New York, USA, 2000.
[7] L. Jamal, M. Shamsujjoha, H.M. Hasan Babu, Design of optimal reversible carry look-ahead adder with optimal garbage and quantum cost, Int. J. Eng. Technol. 2 (2012) 44–50. URL 〈http://iet-journals.org/archive/2012/jan_vol_2_no_1/349421324456832.pdf〉.
[8] M.S. Islam, M.M. Rahman, Z. Begum, M.Z. Hafiz, A.A. Mahmud, Synthesis of fault tolerant reversible logic circuits, CoRR abs/1008.3340 (2010) 1–4. URL 〈http://arxiv.org/abs/1008.3340〉.
[9] B. Parhami, Fault-tolerant reversible circuits, in: Fortieth Asilomar Conference on Signals, Systems and Computers, 2006, pp. 1726 –1729. http://dx.doi.org/10.1109/ACSSC.2006.355056.
[10] R.K. James, T.K. Shahana, K.P. Jacob, S. Sasi, Fault tolerant error coding and detection using reversible gates, in: IEEE TENCON, 2007, pp. 1–4.
[11] M.P. Frank, The physical limits of computing, Comput. Sci. Eng. 4 (3) (2002) 16–26, http://dx.doi.org/10.1109/5992.998637.
[12] A.K. Biswas, M.M. Hasan, A.R. Chowdhury, H.M. Hasan Babu, Efficient approaches for designing reversible binary coded decimal adders, Microelectron. J. 39 (12) (2008) 1693–1703, http://dx.doi.org/10.1016/j.mejo.2008.04.003.
[13] M. Lukac, M. Perkowski, P. Kerntopf, M. Pivtoraiko, M. Folgheraiter, D. Lee, H. Kim, W. Hwuangbo, J. Wook Kim, Y.W. Choi, A hierarchical approach to computer-aided design of quantum circuits, in: Sixth International Symposium on Representations and Methodology of Future Computing Technology, 2003, pp. 201–209.
[14] S.N. Mahammad, K. Veezhinathan, Constructing online testable circuits using reversible logic, IEEE Trans. Instrum. Meas. 59 (2010) 101–109, http://dx.doi.org/10.1109/TIM.2009.2022103.
[15] W.N.N. Hung, X. Song, G. Yang, J. Yang, M.A. Perkowski, Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis, IEEE Trans. CAD Integr. Circuits Syst. 25 (9) (2006) 1652–1663.
[16] D. Maslov, G. W. Dueck, N. Scott, Reversible logic synthesis benchmarks page, 2005. 〈http://webhome.cs.uvic.ca/~dmaslov〉.
[17] E.P.A. Akbar, M. Haghparast, K. Navi, Novel design of a fast reversible wallace sign multiplier circuit in nano-technology, Microelectron. J. 42 (8) (2011) 973–981. URL 〈http://www.sciencedirect.com/science/article/pii/S0026269211001194〉.
[18] D.M. Miller, D. Maslov, G.W. Dueck, A transformation based algorithm for reversible logic synthesis, in: Proceedings of the 40th annual Design Automation Conference, Anaheim, CA, USA, 2003, pp. 318–323. http://dx.doi.org/10.1145/775832.775915.
[19] A. Gepp, P. Stocks, A review of procedures to evolve quantum algorithms, Genet. Program. Evol. Mach. 10 (2) (2009) 181–228.
[20] V. Betz, J. Rose, How much logic should go in an FPGA logic block? in: IEEE Design and Test Magazine, 1998, pp. 10–15.
[21] A.S.M. Sayem, S.K. Mitra, Efficient approach to design low power reversible logic blocks for field programmable gate arrays, in: IEEE International Conference on Computer Science and Automation Engineering, Shanghai, China, 2011, pp. 251–255. http://dx.doi.org/10.1109/CSAE.2011.5952845.
[22] A.S.M. Sayem, M.M.A. Polash, H.M.H. Babu, Design of a reversible logic block of field programmable gate array, in: Silver Jubilee Conference on Communication Technologies and VLSI Design, VIT University, Vellore, India, 2009, pp. 500–501.
[23] M.M.A. Polash, S. Sultana, Design of a LUT-based reversible field programmable gate array, J. Comput. 2 (2010) 103–108.
[24] F. Sharmin, M.M.A. Polash, M. Shamsujjoha, L. Jamal, H.M. Hasan Babu, Design of a compact reversible random access memory, in: Fourth IEEE International Conference on Computer Science and Information Technology, vol. 10, Chengdu, China, 2011, pp. 103–107.
[25] L. Singhal, E. Bozorgzadeh, Fast timing closure by interconnect criticality driven delay relaxation, in: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design, IEEE Computer Society, Washington, DC, USA, 2005, pp. 792–797. 〈http://dl.acm.org/citation.cfm?id=1129601.1129713〉.
[26] S.G. Younis, T.F. Knight, Practical implementation of charge recovering asymptotically zero power CMOS, in: Proceedings of the 1993 Symposium on Research on Integrated Systems, MIT Press, Cambridge, MA, USA, 1993, pp. 234–250. 〈http://dl.acm.org/citation.cfm?id=163429.163468〉.
[27] S.G. Younis, Asymptotically Zero Energy Computing Using Split-level Charge Recovery Logic, Technical Report, Cambridge, MA, USA, 1994.
[28] J. Lim, K. Kwon, S.I. Chae, Reversible energy recovery logic circuits and its 8-phase clocked power generator for ultra-low-power applications, IEICE Trans. Electron E82-C (1999) 646–653.
[29] J. Lim, S.-I. Chae, D.-G. Kim, nMOS reversible energy recovery logic for ultra-low-energy applications, IEEE J. Solid-State Circuits 35 (2000) 865–875.
[30] P.D. Picton, Optoelectronic multi-valued conservative logic, Int. J. Opt. Comput. 2 (1991) 19–29.
[31] D.P. Vasudevan, P.K. Lala, J.P. Parkerson, Online testable reversible logic circuit design using nand blocks, defect and fault-tolerance in VLSI systems, in: IEEE International Symposium on 0, 2004, pp. 324–331. 〈http://doi.ieeecomputersociety.org/10.1109/DFT.2004.47〉.
[32] H. Thapliyal, A.P. Vinod, Design of reversible sequential elements with feasibility of transistor implementation, in: International Symposium on Circuits and Systems (ISCAS 2007), IEEE, 2007, pp. 625–628. 〈http://doi.ieeecomputersociety.org/10.1109/ISCAS.2007.378815〉.
[33] DSCH, Microwind and DSCH information page. URL 〈http://www.microwind.org/〉.
[34] A.S.M. Sayem, M. Ueda, Optimization of reversible sequential circuits, J. Comput. 2 (6) (2010) 208–214. URL 〈http://arxiv.org/abs/1006.4570〉.
[35] M.-L. Chuang, C.-Y. Wang, Synthesis of reversible sequential elements, J. Emerg. Technol. Comput. Syst. 3 (2008) 1–19, http://dx.doi.org/10.1145/1324177.1324181.
[36] R. Wisniewski, Synthesis of compositional microprogram control units for programmable devices, in: Lecture Notes in Control and Computer Science, vol. 14, University of Zielona Góra Press, Zielona Góra, 2009. ISBN: 978-83-7481-293-1.
[37] W.-K. Mak, D.F. Wong, Board-level multiterminal net routing for FPGA-based logic emulation, ACM Trans. Des. Autom. Electron. Syst. 2 (2) (1997) 151–167 http://doi.acm.org/10.1145/253052.253136.
[38] D. Yin, D. Unnikrishnan, Y. Liao, L. Gao, R. Tessier, Customizing virtual networks with partial FPGA reconfiguration, SIGCOMM Comput. Commun. Rev. 41 (1) (2011) 125–132, http://dx.doi.org/10.1145/1925861.1925882.
[39] M. French, L. Wang, M. Wirthlin, P. Graham, Reducing power consumption of radiation mitigated designs for FPGAs, in: 9th Annual International Conference on Military and Aerospace Programmable Logic Devices, 2006.
[40] S. Coric, M. Leeser, E. Miller, M. Trepanier, Parallel-beam backprojection: an FPGA implementation optimized for medical imaging, in: Proceedings of the ACM/SIGDA 10th International Symposium on Field-programmable Gate Arrays, ACM, New York, USA, 2002, pp. 217–226. http://dx.doi.org/10.1145/503048.503080.
[41] E.C. Lin, K. Yu, R.A. Rutenbar, T. Chen, A 1000-word vocabulary, speaker-independent, continuous live-mode speech recognizer implemented in a single FPGA, in: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays, ACM, New York, NY, USA, 2007, pp. 60–68. http://dx.doi.org/10.1145/1216919.1216928.
[42] S. Brown, R. Francis, J. Rose, Z. Vranesic, Field-Programmable Gate Arrays, Springer/Kluwer Academic Publishers. ISBN: 978-0-7923-9248-4.
[43] Voyiatzis, D. Gizopoulos, A. Paschalis, Accumulator-based test generation for robust sequential fault testing in DSP cores in near-optimal time, IEEE Trans. VLSI Syst. 14 (2005) 1079–1086.