**Annexure – A**

**KMD Gates**

The functional diagrams of the reversible KMD gates are shown in Figure A.1;

|  |  |
| --- | --- |
|  |  |
|  |  |

**Figure A.1: Functional Diagrams of the Reversible KMD gates**

**Annexure - B**

**Derivative functions of KMD Gates**

ALU, Floating Point Division and Vedic Multiplier is designed using the proposed reversible KMD Gates. The functional units required for the construction of the above systems / circuits are as shown in Table B.1.

**Table B.1: Functional units required for Proposed Designs**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Proposed Design** | **Functional Units** |
|  | Arithmetic and Logic Unit | Inverter, AND logic, OR logic, XOR logic, Adder and Multiplexer |
|  | Floating Point Division | Multiplexer, Adder, D Latch and Multi-functional Register |
|  | Vedic Multiplier | AND logic, Half adder and Full adder |

The functional units needed to construct the Arithmetic and Logic Unit, Floating-Point Division and Vedic Multiplier can be derived from the KMD Gates are as follows:

* Inverter function obtained from KMD Gate1 for the inputs   
  A & B.
* 2×1 multiplexer & XOR gate is derived from KMD Gate 3
* AND logic is derived from KMD Gate2
* OR logic is derived from KMD Gate1
* XOR logic from KMD Gate3
* Adder function from KMD Gate4
* D latch function from KMD Gate3
* The multi-functional register is derived from D-latch

The inverter is derived from KMD Gate1 by keeping B=C=1 and the inverted output is available at both Q and R. The AND gate operation is derived from KMD Gate2 by keeping C = 0 and the output is available at R. The OR gate operation is derived from KMD Gate1 by keeping C = 0 and the output is available at R. XOR logic is derived from the KMD Gate3 by keeping Constant inputs B = C = 0 and D = B. The output is observed at P.

The MUX operation is derived from KMD Gate3 by keeping A = sel (selection line), B = A, C = B and D = 0 and the output is available at R and the selection line traverses through P. Half adder is obtained from KMD Gate3. The Constant input is C = 0, the inputs are A, B and D input connected with B. Sum and Carry outputs are available at P, R respectively. The full adder function is derived from the KMD Gate4. The constant inputs are D = E = 0 and outputs are R, S for Sum and Carry respectively. The D Latch function is obtained from KMD Gate3 by keeping D = 0, A = CLK and B = Data. The output is observed at R and the Q output is applied to C input as feedback. The multi-functional register is constructed from the combination of KMD Gate3 and D Latch. The Quantum Equivalent circuit of the derivative functions is shown in Table B.2.

**Table B.2: Derivative functions of KMD Gates and its Quantum Realization**

| **Function** | **KMD Structure** | **Quantum realization** |
| --- | --- | --- |
| **Inverter** | **Q, R = NOT (A)** |  |
| **AND Logic** | **R = AND (A,B)** |  |
| **OR Logic** | **R = OR (A, B)** |  |
| **XOR** | **P = XOR (A,B)** |  |
| **Multiplexer** | **R = MUX (A,B)** |  |
| **Half Adder** | **P = SUM (A, B)**  **R = CARRY (A,B)** |  |
| **Adder** | **R = SUM (A, B, C)**  **S = CARRY (A,B,C,D)** |  |
| **D Latch** | **R = CLK’.Q ⊕ CLK.D** |  |
| **Multi -functional register** |  |  |
|  | |

**Annexure – C**

**The performance characteristics of the KMD Gates are as follows;**

**Table C.1a: Truth Table of KMD Gate 1**

| **Inputs** | | | **Outputs** | | | **Permutation** | **Parity Preservation** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | P | Q | R |
| 0 | 0 | 0 | 0 | 0 | 0 | 1-cycle | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 2-cycle | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 2-cycle | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1-cycle | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 4-cycle | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 3-cycle | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 3-cycle | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 4-cycle | 1 |

**Table C.1b: Truth Table of KMD Gate 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Inputs** | | | **Outputs** | | | **Permutation** | **Parity Preservation** |
| A | B | C | P | Q | R |
| 0 | 0 | 0 | 0 | 0 | 0 | 1- cycle | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 5- cycle | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 2 -cycle | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 4- cycle | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1- cycle | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 5 - cycle | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 2 - cycle | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 2 - cycle | 1 |

**Table C.1c: Truth Table of KMD Gate 3**

| **Inputs** | | | | **Outputs** | | | | **Permutation** | **Parity Preservation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | P | Q | R | S |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1-cycle | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 13-cycle | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 6-cycle | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 8-cycle | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1-cycle | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 8-cycle | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 4-cycle | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 8-cycle | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1-cycle | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 5-cycle | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 4-cycle | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 11-cycle | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4-cycle | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 12-cycle | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 4-cycle | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 10-cycle | 0 |

**Table C.1d: Truth Table of KMD Gate 4**

| **Inputs** | | | | | **Outputs** | | | | | **Permutation** | **Parity Preservation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | P | Q | R | S | T |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1-Cycle | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1-Cycle | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1-Cycle | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1-Cycle | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1-Cycle | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1-Cycle | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1-Cycle | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1-Cycle | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 6-Cycle | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 4-Cycle | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 6-Cycle | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 4-Cycle | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 3-Cycle | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 3-Cycle | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 7-Cycle | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 7-Cycle | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 13-Cycle | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 13-Cycle | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 13-Cycle | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 13-Cycle | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 3-Cycle | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 3-Cycle | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 7-Cycle | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 7-Cycle | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 4-Cycle | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 6-Cycle | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2-Cycle | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 4-Cycle | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 7-Cycle | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 7-Cycle | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 11-Cycle | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 11-Cycle | 1 |

The universality property of the KMD Gates is tabulated in Table C.2. The constant input is set at any one of the inputs of the KMD Gates, which made the KMD Gate to be function as NOT, AND / NAND, OR / NOR gate. For example, for the KMD Gate1 input ‘C’ is set as constant input. For C = 1, the KMD Gate1 produces NAND function of A, B at output ‘Q’; and for C = 0, the KMD Gate1 produces OR function of A, B at output ‘Q’. Also, if A = C = 1, then the output ‘Q’ produces the NOT function of B.

**Table C.2: Universality Property of KMD Gates**

| **S. No.** | **Reversible Gate** | **Constant Input** | **Logic Function** | **Expression** |
| --- | --- | --- | --- | --- |
|  | KMD Gate1 | A=1; C=0 / 1 | Q=NOT (B) | Q = R’ |
| C=1 | Q = NAND (A,B) | Q = A’ + AB’ |
| C=0 | R = OR (A, B) | Q = A + A’B |
|  | KMD Gate2 | B=C=0 | R = NOT (A) | R = A’ |
| C=0 | Q = NOR (A,B) | Q = A’B’ = (A+B)’ |
| C=1 | R = AND (A,B) | R = AB |
|  | KMD Gate3 | B=D=0; C=1 | S = NOT (A) | S = A’ |
| B=1 | R = OR (A, C) | R = A + A’C |
| C=0 | R = AND (A, B) | R = AB |
|  | KMD Gate4 | B=C=1;D=0 | Q = NOT(A) | Q = A’ |
| B=1 | Q = NAND (A,C) | Q = A’ + AC’ |
| B=1; D=0  C=0 | T = OR (A,C)  Q = OR(A,C) | T = AC’ + C  Q = AC’ + C |

The performance comparison of the available reversible gates and KMD Gates are shown in Table C.3 and Table C.4. It has been observed that the proposed KMD Gates are able to generate all basic logical, sum and carry functions. Also, it is having reversibility, universality and parity preservation with the optimum quantum cost.

**Table C.3a: Comparison of Reversible gates with the proposed KMD Gates (3×3)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Characteristics** | **TR** | **PPRG** | **TSG** | **DKG** | **KMD Gate1** |
| Functions | NOT, XOR, XNOR | NOT, AND, XOR | NOT, XOR, XNOR | NOT, XOR, AND | NOT, OR, NAND |
| Cell Count | 113 | 171 | - | 752 | 169 |

**Table C.3b: Comparison of Reversible gates with the proposed KMD Gates (4×4)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Characteristics** | **Peres** | **NFT** | **RM** | **RUG** | **KMD Gate2** | **KMD Gate3** |
| Functions | NOT, AND, NAND, XOR | XOR, AND, NOT, OR | NOT, XOR, AND, OR | AND, XOR, OR, NOT | AND, OR, NOT, NOR | XOR, AND, NOT, OR, SUM, CARRY |
| Cell Count | 273 | 128 | 178 | 196 | 121 | 116 |

**Table C.3c: Comparison of Reversible gates with the proposed KMD Gates (5×5)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Characteristics** | **R Gate** | **R1** | **OTG** | **SMS** | **KMD Gate4** |
| Functions | NOT, AND, OR, NAND, XOR | NOT, OR, NAND, XOR, XNOR | NOT, NAND, XOR, XNOR, Majority | NOT, AND, NAND, XOR, XNOR | XOR, AND, OR, NOT, NAND, SUM, CARRY |
| Cell Count | 105 | - | - |  | 244 |

**Table C.4: Characteristics and Performance of KMD Gates**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Name of the Reversible Gate** | **Reversibility** | **Universality** | **Parity Preservation** | **Gate Count** | **Quantum Cost** | **Number of Functions (NoF)** | **Logical Calculations** |
|  | KMD Gate1 | Yes | Yes | Yes | 5 | 10 | 3 | 2α+4β+3γ |
|  | KMD Gate2 | Yes | Yes | Yes | 6 | 10 | 4 | 2α+4β+3γ |
|  | KMD Gate3 | Yes | Yes | Yes | 5 | 9 | 6 | 5α+3β+1γ |
|  | KMD Gate4 | Yes | Yes | Yes | 8 | 12 | 7 | 6α+4β+2γ |

The proposed KMD Gates structure and their Reversibility & Parity Preservation, Universality, Characteristics and Performances are listed in Table 1 – 4 respectively. From the above, it is observed that the proposed KMD Gates are having Reversibility, Universality and Parity preservation. Also, they are able to realize many arithmetic and logical functions with lower performance cost.

Thirteen standard functions are implemented using KMD Gates and their performances are evaluated and tabulated in Table C.5 for comparison. The Toffoli, Peres and TR gates perform well for the simple functions compared to Fredkin, RM and RUG. KMD Gates perform well in the case of complex functions. It is observed from the table that the average number of gates that are required to implement the standard functions is 1.76 and the average quantum cost is 18.23. Therefore, the proposed Reversible KMD Gates will be useful to realize the complex computational circuits with the minimum average number of gates and with a minimum average cost.

**Table C.5: Performance evaluation of KMD Gates**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Standard Functions** | **Toffoli** | | **Peres** | | **TR** | | **Fredkin** | | **RM** | | **RUG** | | **KMD** | |
| **NoG** | **QC** | **NoG** | **QC** | **NoG** | **QC** | **NoG** | **QC** | **NoG** | **QC** | **NoG** | **QC** | **NoG** | **QC** |
|  | F=ABC | 2 | 10 | 2 | 8 | 2 | 12 | 4 | 20 | 2 | 70 | 2 | 62 | **2** | **18** |
|  | F=AB | 1 | 5 | 1 | 4 | 1 | 6 | 2 | 10 | 1 | 35 | 1 | 31 | **1** | **9** |
|  | F=ABC+AB’C’ | 3 | 15 | 3 | 12 | 2 | 12 | 3 | 15 | 2 | 70 | 2 | 62 | **4** | **36** |
|  | F=ABC+A’B’C’ | 10 | 50 | 10 | 40 | 9 | 54 | 4 | 20 | 3 | 105 | 3 | 93 | **3** | **27** |
|  | F=AB+BC | 5 | 25 | 4 | 16 | 4 | 24 | 2 | 10 | 2 | 70 | 2 | 62 | **2** | **18** |
|  | F=AB+A’B’C | 8 | 40 | 6 | 24 | 7 | 42 | 5 | 25 | 2 | 70 | 3 | 93 | **3** | **27** |
|  | F=ABC+A’BC’+AB’C’ | 8 | 40 | 7 | 28 | 8 | 48 | 6 | 30 | 3 | 105 | 3 | 93 | **5** | **45** |
|  | F=A | 1 | 5 | 1 | 4 | 1 | 6 | 1 | 5 | 1 | 35 | 1 | 31 | **1** | **9** |
|  | F=AB+BC+AC | 9 | 45 | 9 | 36 | 6 | 36 | 5 | 25 | 5 | 175 | 1 | 31 | **1** | **24** |
|  | F=AB+B’C | 6 | 30 | 6 | 24 | 5 | 30 | 1 | 5 | 1 | 35 | 1 | 31 | **1** | **9** |
|  | F=AB+BC+A’B’C’ | 4 | 20 | 4 | 16 | 3 | 18 | 6 | 30 | 2 | 70 | 2 | 62 | **2** | **18** |
|  | F=AB+A’B’ | 2 | 10 | 2 | 8 | 2 | 12 | 2 | 10 | 2 | 70 | 1 | 31 | **1** | **9** |
|  | F=ABC+A’B’C+AB’C’+A’BC’ | 2 | 10 | 2 | 8 | 2 | 12 | 3 | 15 | 2 | 70 | 2 | 62 | **1** | **24** |
|  | **AVERAGE** | **4.69** | **23.4** | **4.38** | **17.5** | **4.23** | **24** | **3.15** | **16.9** | **2.15** | **75.3** | **1.84** | **57.2** | **1.76** | **18.23** |

\* NoG – Number of Gates; QC – Quantum Cost

**Annexure D**

The detailed explanation and hardware architecture of the ALU, Floating point division and Vedic multiplier has been explained in this Annexure.

**ARITHMETIC AND LOGIC UNIT ARCHITECTURE**

The ALU structure has 3 major modules;

* **Input module**: The 2×1 multiplexer selects either True or Complement form of the input and forwards it to the next computing module.
* **Data Processing module**: It does the arithmetic and logical operations on the input data fed from the input module. The possible list of arithmetic, logical operations it could perform is shown in Table 4.2 & 4.3.
* **Output module**: The data received from various functional modules are available as inputs to the output module; the select line of the 4×1 multiplexer controls which output has to be forwarded to the Final Result.

The proposed ALU structure is having following advantages over the existing ALU structures,

* A Single functional module performs both Arithmetic and Logical operations with the help of Select Signals.
* The input signals (A, B & Cin) also act as Constant inputs. This reduces the requirement of the number of inputs and constant inputs compared to the existing ALU structures.

The proposed ALU architecture has been designed in two different ways and their performances were observed.

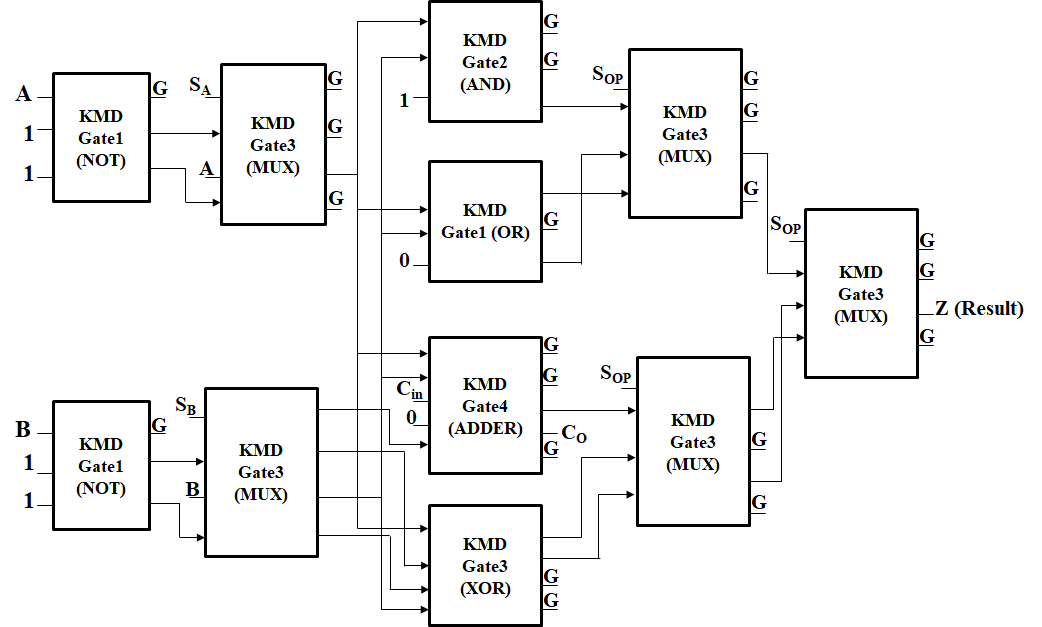
* Approach-1: All the functional modules (Input, Data processing & Output) are constructed only with KMD Gates.
* Approach-2: The functional modules are constructed using the combination of KMD Gates, Fredkin and Toffoli gates.

**Approach-1**

The functional modules of the proposed ALU are constructed using the following in Approach-1:

* The inverter function obtained from KMD Gate1 for the inputs A & B.
* 2 × 1 multiplexer & XOR gate is derived from KMD Gate 3
* AND logic is derived from KMD Gate2
* Adder is derived from KMD Gate4 and
* OR logic is derived from KMD Gate1

In the approach-1, the construction of Reversible Parity Preserving ALU architecture using KMD Gates alone is shown in Figure D.1. The variables represented in this Figure D.1 are; A, B, Cin - Inputs; 1 or 0 – Constant Inputs; G – Garbage Outputs and Z – Final Result.

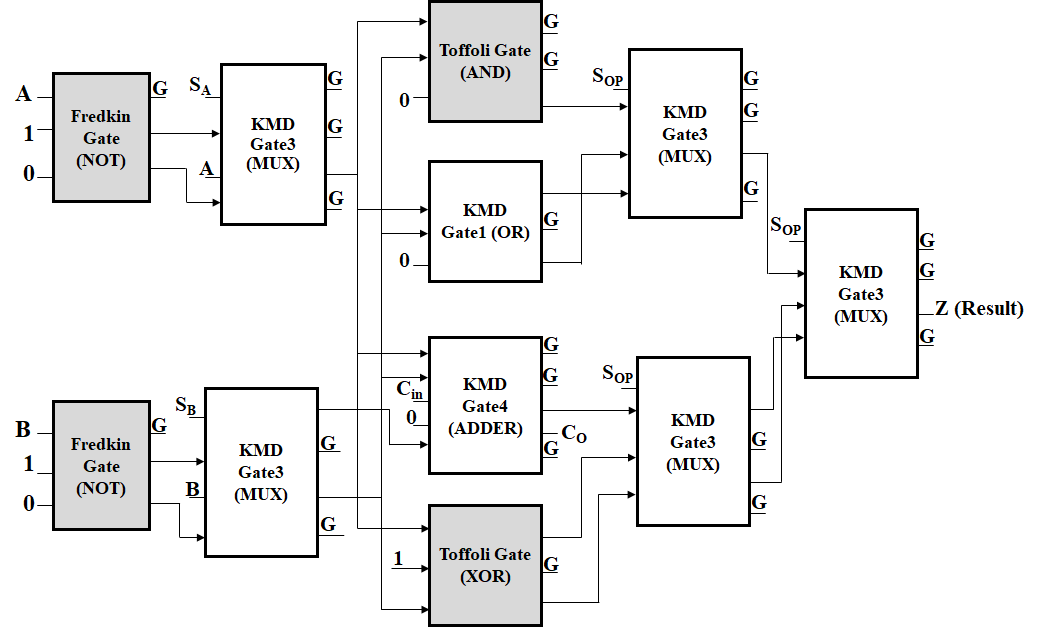


**Figure D.1: Proposed Reversible Parity preserving ALU Architecture (Approach-1)**

**Approach-2**

The functional modules of the proposed ALU are constructed using the following in Approach-2:

* The inverter function obtained from the Fredkin gate for inputs A & B.
* 2×1 multiplexer using KMD Gate3
* XOR and AND gate is derived from Toffoli gate
* Adder is derived from KMD Gate4 and
* OR logic is derived from KMD Gate1

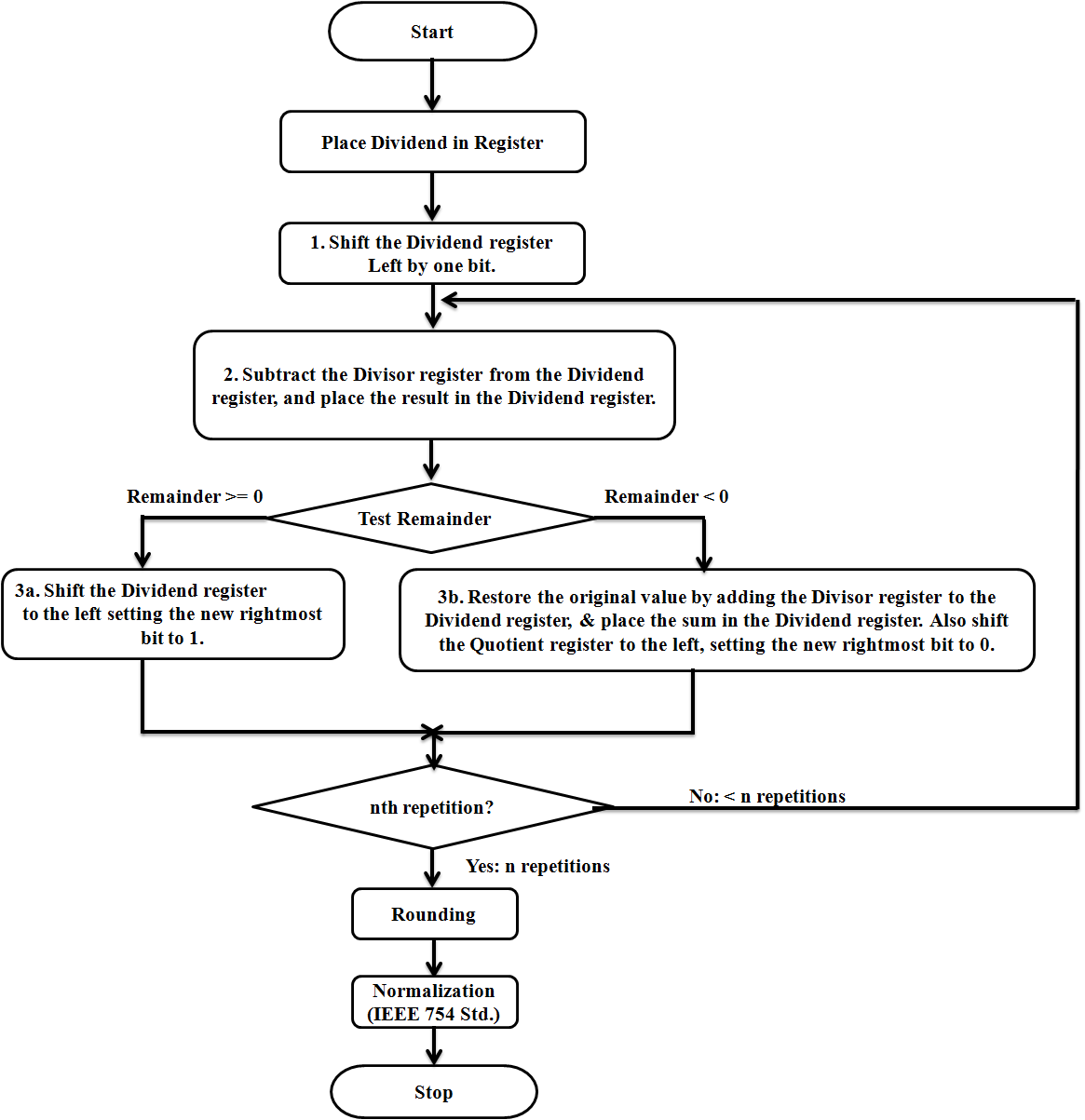


**Figure D. 2: Proposed Reversible ALU Architecture (Approach-2)**

In the Approach-2, the Fredkin, Toffoli and KMD Gates are used to construct the reversible ALU as shown in Figure D.2. The Fredkin gate replaces the KMD Gate1 in the input module to invert the inputs A, B; and Toffoli gate replaces the KMD Gate2 & KMD Gate3 to generate AND & XOR logic function. Here, the positive side of the design is the reduction of quantum cost and the number of garbage outputs.

**FLOATING POINT DIVISION**

The floating point division algorithm that has been discussed in the manuscript, whose flow diagram of has been shown in Figure D.3.



**Figure D.3: Flow Diagram of Floating-Point Division Operation**

The significant functional modules are multiplexers, multi-functional register, registers, parallel adder, rounding and normalization units. The control signals for sequencing the operations are Clk, load, sel, SP, set Q0, shift and hold. These signals are released from the control unit at the proper time to sequence the FP division operation in the functional modules.

When Clk input is available, the load & SP signals are high, the multi-functional register ‘A’ is loaded with ‘0’, Q register loaded with dividend and V register loaded with divisor. During the next Clk, the dividend and divisor are forwarded to (n+1) parallel adder. The adder performs the 2’s complement addition via F2G register bank and FRG register during every cycle.

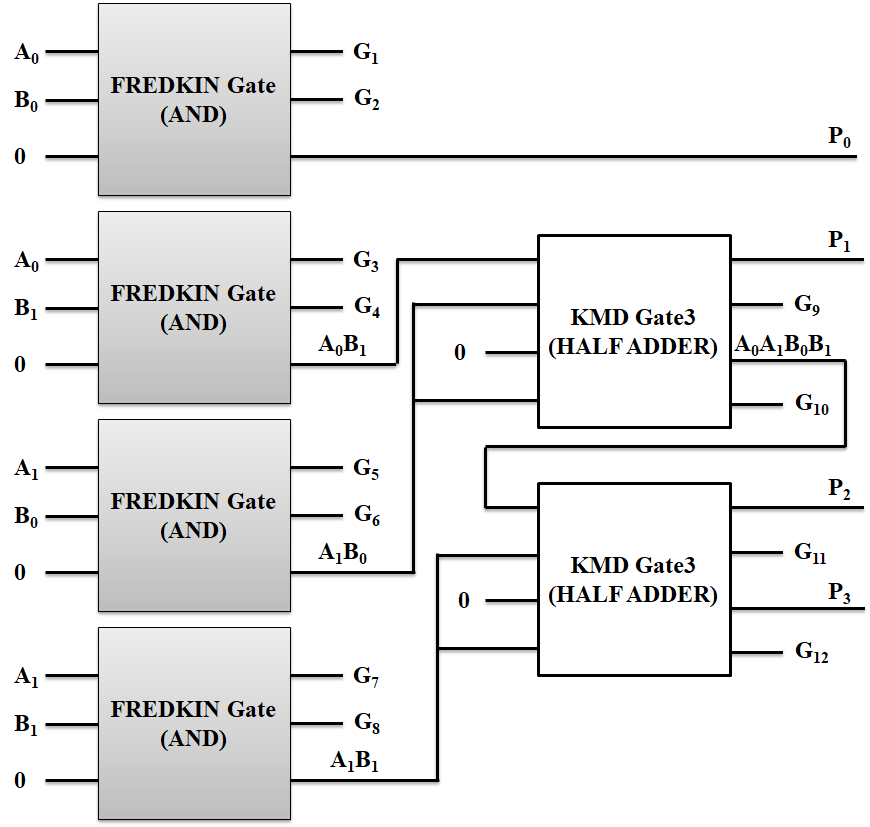
The partial results thus obtained are stored back to the   
multi-functional register ‘A’. In the meantime, left shifted (via F2G register) dividend is loaded back through the multiplexer and the MSB bit is serially shifted to the ‘SO’ of ‘A’ register.

The sign bit of the partial result decides the Q0 value (ie. Q0=0, if sign =1; Q0=1, otherwise). The same procedure is repeated for ‘n’ Clk cycle. After n-cycle, if the result is negative, restoration takes place. Register Q contains the quotient and ‘A’ register contains the remainder after the successful division.

vedic multiplier

The 2×2 Vedic multiplier has been designed in two different approaches.

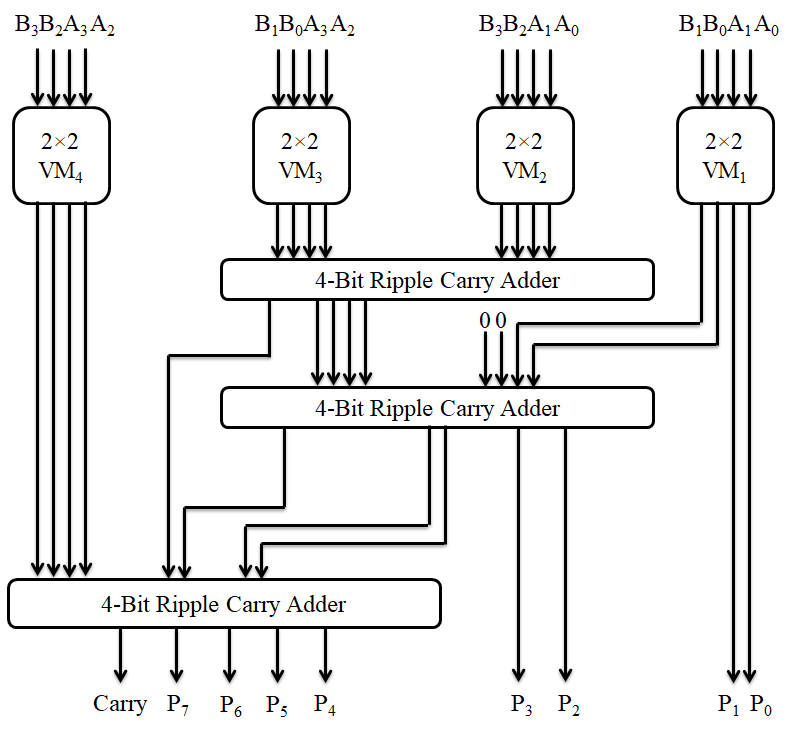
* Approach-1: All the functional units (AND and Adder) are constructed using only KMD Gates
* Approach-2: Combination of KMD Gates (for Adder) and Fredkin (for AND logic) are used for construction (Figure D.4).



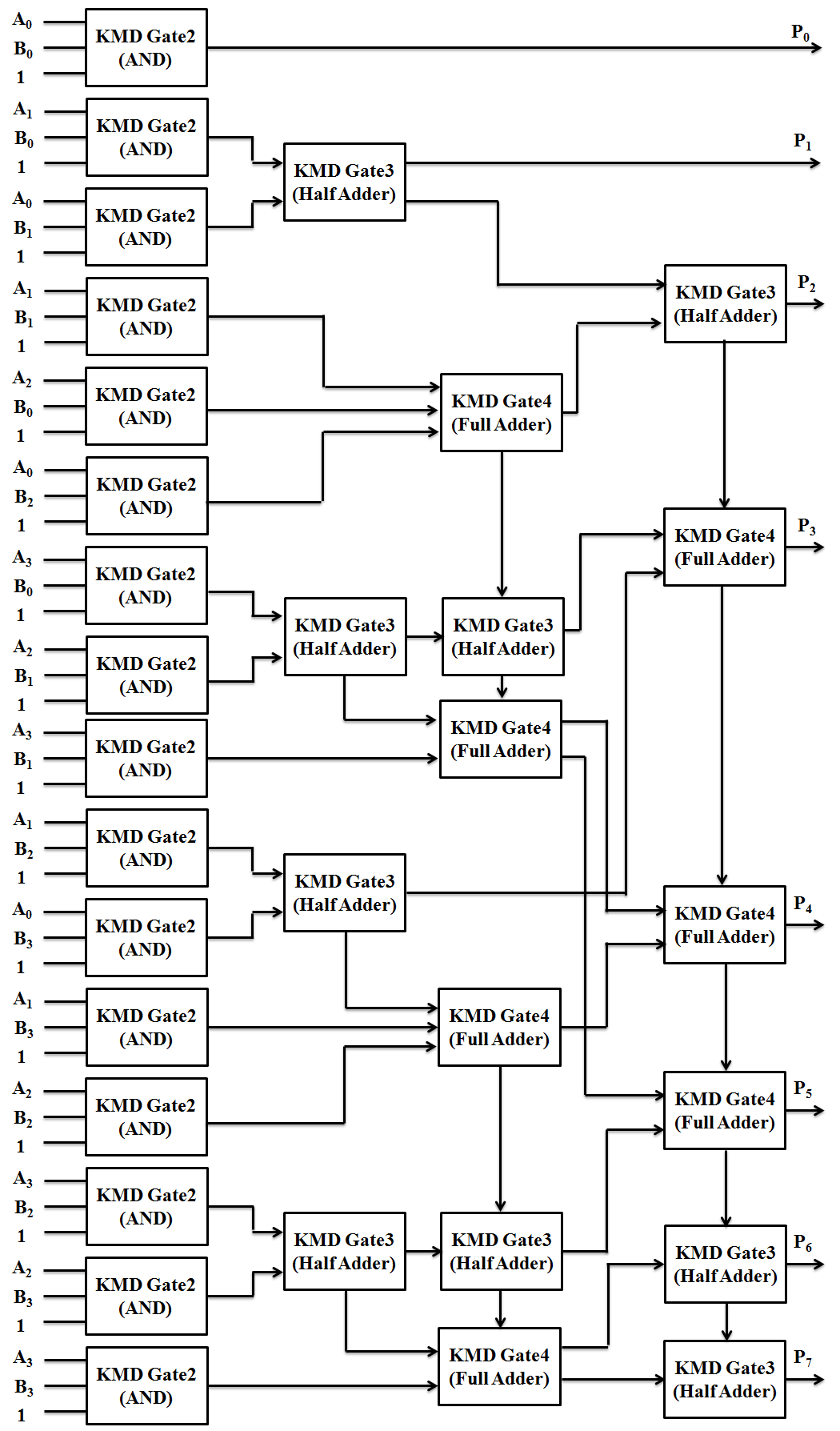
**Figure D.4: KMD and Fredkin Gates architecture of 2×2 Vedic multiplier (Approach-2)**

**4**×**4 Vedic Multiplier**

The 4-bit Vedic Multiplier (VM) module is constructed using a 2-bit Vedic multiplier unit (VMi) four times. The other functional units are three 4-bit ripple carry adders. The complete architecture of 4×4 Vedic multiplier is shown in Figure D.5 and its KMD Gate realization shown in Figure D.6. The inputs Ai (A3A2A1A0) and Bi (B3B2B1B0) are given to 2-bit Vedic multiplier bitwise. Its output is forwarded to the 4-bit RCA adder for summand addition. The output of the 4-bit multiplier is an 8-bit product term Pi (P7…P0).

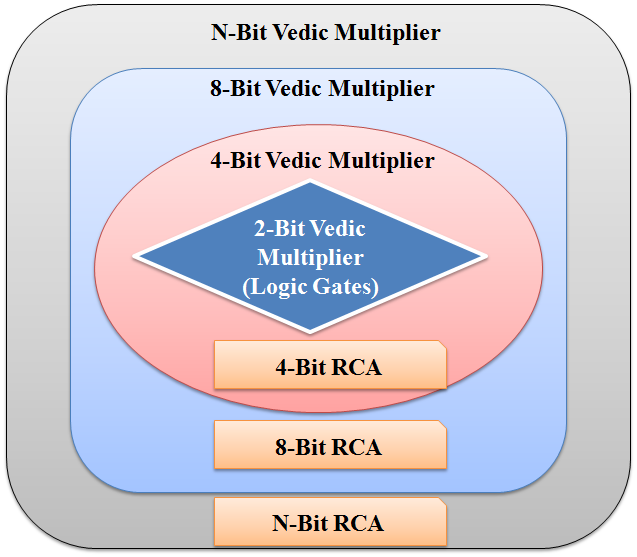


**Figure D.5: The hardware structure of 4-bit Vedic Multiplier**

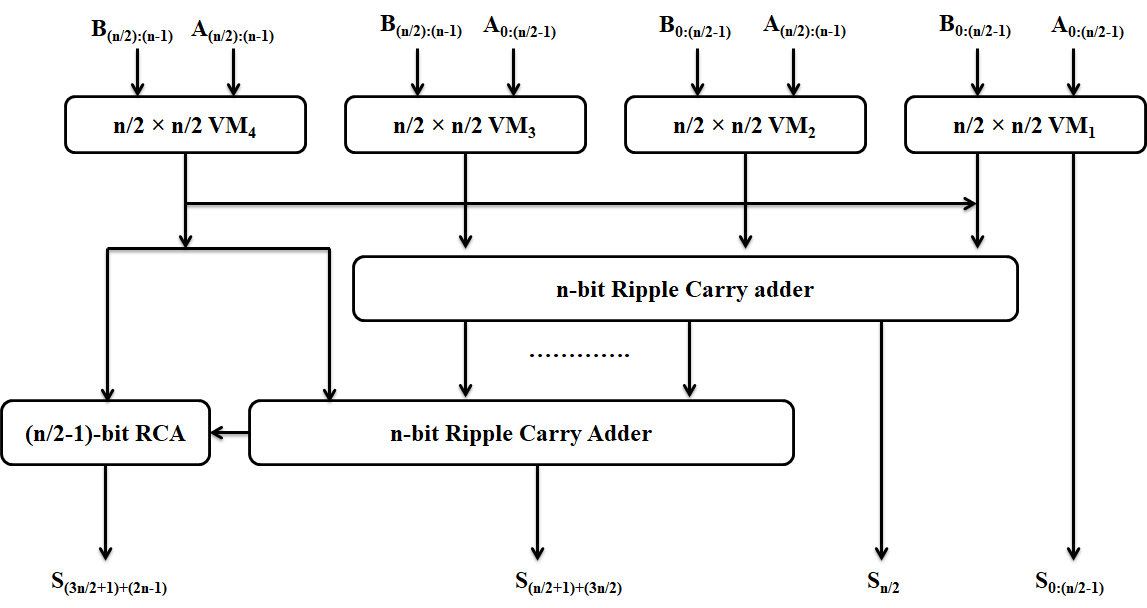


**Figure D.6: 4×4 reversible Vedic multiplier structure using KMD Gates**

Higher-order Vedic multiplier is constructed using the lower order Vedic Multipliers (VMi) as the base module and RCA adders for summation of the partial products as shown in Figure D.7. The n-bit Vedic multiplier uses four number of n/2-bit multipliers, two n-bit RCA adders, and one ((n/2)−1) bit RCA adder (using only Half adder). For example, construction of 16-bit Vedic multipliers by this comprehensive structure requires four 8-bit multiplier units, two 16-bit RCA adders, and one 7-bit RCA adder (using only half adder). So, using this generalization, n-bit Vedic multiplication unit can be constructed as shown in Figure D.8.

****

**Figure D.7: Generalized construction of n-bit Vedic Multiplier**

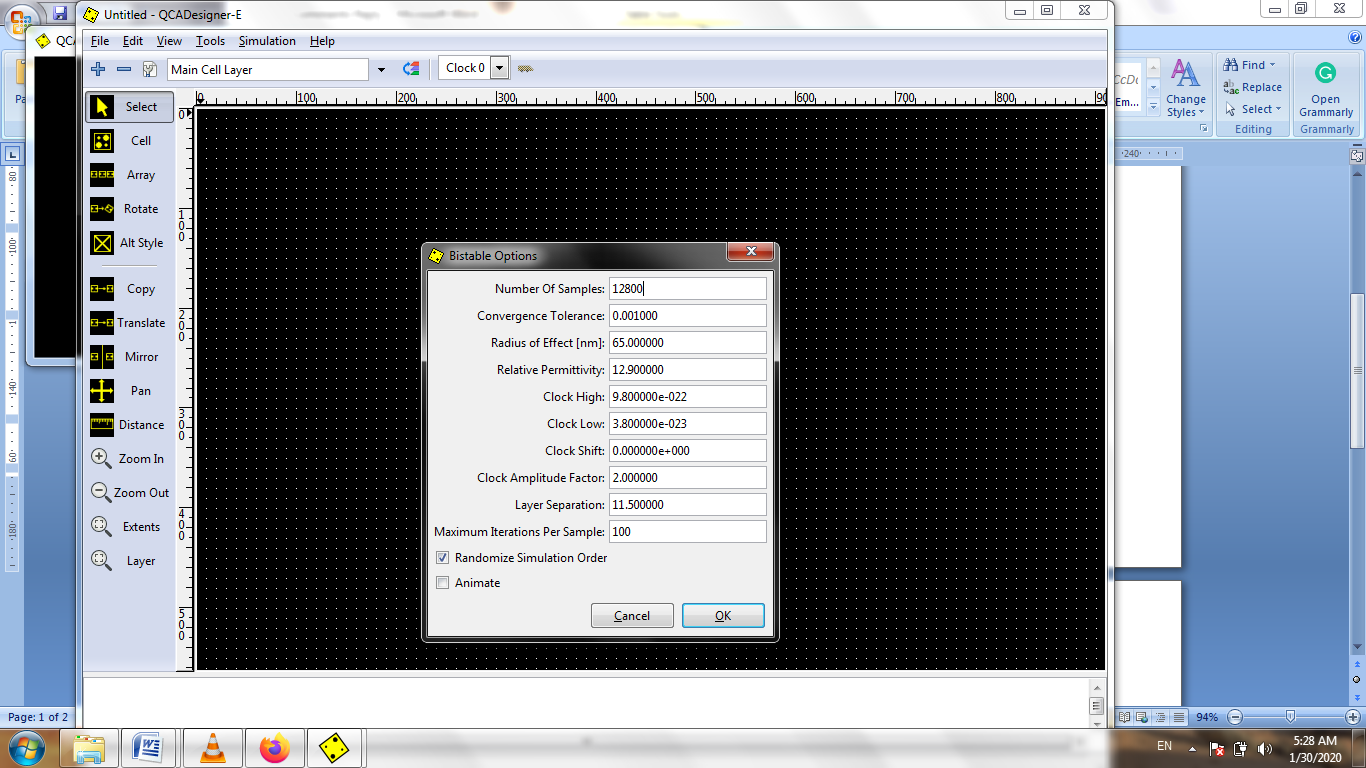


**Figure D.8: Construction of n-bit Vedic Multiplier**

**Annexure E**

**Simulation Environment**

The simulation environment chosen for the proposed design in QCA Tool is Bistable Simulation engine with exhaustive simulation and the complete the setting has been shown in Figure E1.



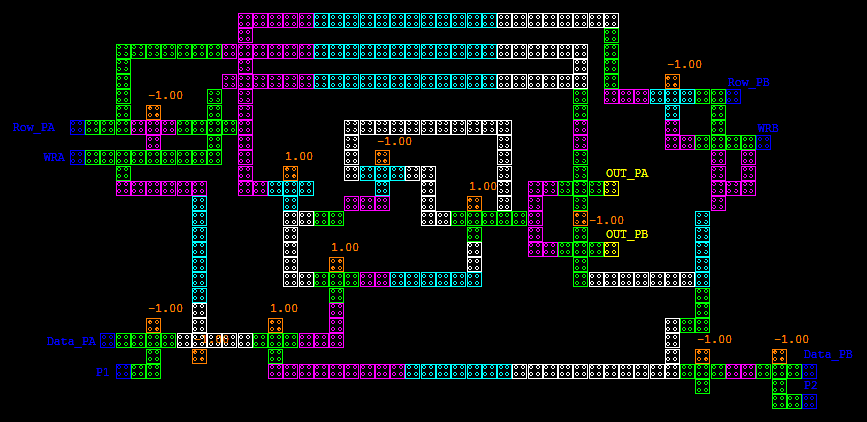
**Figure E.1: Simulation Settings**

The Figure E.2 shows the simulation waveform of the Vedic multiplier in QCA.

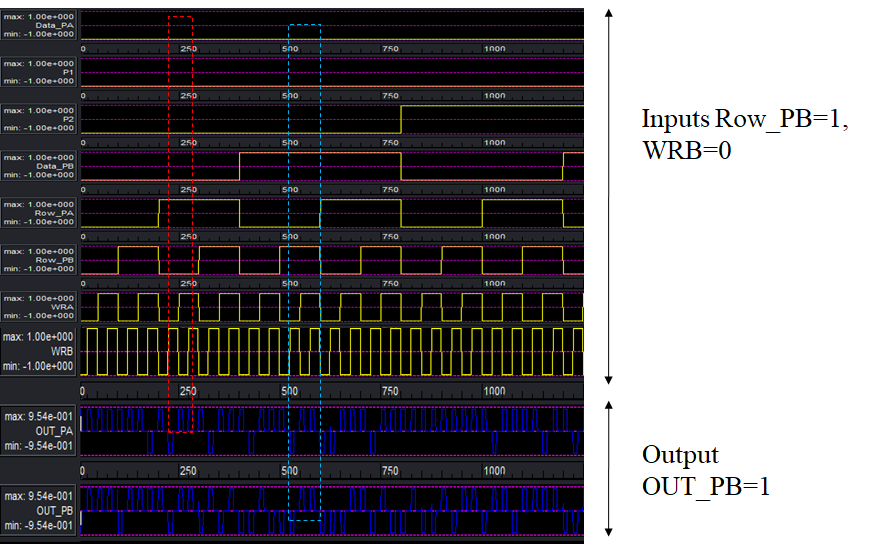


**Figure E.2. Simulation of Vedic Multiplier**

The QCA realization of Memory Cell and its simulation are shown in Figure E.3 and Figure E.4.

****

**Figure E.3. QCA realization of Macro Memory Cell**

**Figure E.4. Simulation of Macro Memory cell**