|  |
| --- |
| **External Project Report on**  **Digital Logic Design(EET1211)** |

**4-Bit Incrementer**

**& 4-Bit Decrementer**



**Submitted by**

**Name: BRAVISH GHOSH Regd. No: 1941012333**

**Group Members**

**Name: SATYAN PRADHAN Regd. No: 1941012311**

**Name: ASHUTOSH DASH Regd. No: 1941012274**

**Name: SAURANSU KHATUA Regd. No: 1941012394**

**B. Tech. CSE 3rd Semester (Section - Q)**

****

# Declaration

We, the undersigned students of B. Tech. of **Computer Science Engineering** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**4-BIT INCREMENTER & 4-BIT DECREMENTER**” submitted to **Siksha ‘O’ Anusandhan Deemed to be University, Bhubaneswar** for the partial fulfillment of the subject **DigitalLogic Design (EET 1211)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

|  |  |
| --- | --- |
| **ASHUTOSH DASH**  **Registration No.: 1941012274** | **BRAVISH GHOSH**  **Registration No.: 1941012333** |
| **SATYAN PRADHAN**  **Registration No.: 1941012311** | **SAURANSU KHATUA**  **Registration No.: 1941012394** |

**DATE: 15.01.2021**

**PLACE: BHUBANESWAR**

# Abstract

For the External Project Report on Digital Logic Design (EE1211) we designed a 4-bit incrementer and a 4-bit decrementer. We used four half adder circuits for the incrementer, and one half adder circuit and three full adder circuits for the decrementer. The incrementer and decrementer circuits are used to add (increment) or subtract (decrement) a four-bit binary data by unity, based on combinational logic blocks.

We generated the solution using Truth table, K-map and (or) boolean algebra. This project was programmed in EDA playground using SystemVerilog HDL, with the aid of Icarus Verilog 0.9.7 simulation tool. The project involved the design of the combinational circuits and simulations of the same. We also made the block diagram and, logic diagram using logic gates.

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial No.** | **Chapter No.** | **Title of the Chapter** | **Page No.** |
|  | 1 | Introduction | 1 |
|  | 2 | Problem Statement | 2 to 3 |
|  | 3 | Methodology | 4 to 8 |
|  | 4 | Implementation | 9 to 24 |
|  | 5 | Results and interpretation | 25 to 26 |
|  | 6 | Conclusion | 27 |
|  |  | References | 28 |
|  |  |  |  |

# **1. Introduction**

**A)**For the four-bit incrementer, we designed a combinational circuit that adds 1 to

given four-bit binary number i.e. if the user inputs a four-bit binary number to it,

then it returns the output as 1 bit incremented number. It is made by cascading ‘n’

half adders for ‘n’ number of bits.

Here we used four half adder circuits connected to each other. The least significant

bit must have one input connected to logic1. The other input receives the number to

be incremented or the carry from the previous stage. At last the sum output is the

required incremented number and the carry output is discarded.

**B)**For the four-bit decrementer, we designed a combinational circuit that subtracts 1

from a given four-bit binary number i.e. if the user inputs a four-bit binary number to

it,then it returns the output as 1 bit decremented number. It is made by cascading

’n-1’full adders and 1 half adder for ‘n’ number of bits.

Here we used 1 half adder circuit and 3 full adder circuits connected to each other.

The least significant bit must have one input connected to logic1. The other input

receives the number to be decremented or the carry from the previous stage. At last

the output is the required decremented number and the carry output is discarded. In

each of the full adder used it must have one input connected to logic 1.

**2. Problem Statement**

# **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# **A) Design a four bit incrementer (a circuit that adds one to a four bitbinary number)**

**B) Design a four bit decrementer (a circuit that subtracts one from a four bit binary number)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# I. Explanation of problem and identification of input and output variables.

**A)** In this, we have to design a combinational circuit that adds 1 to a given four-bit

binary number. It means that if the user inputs a four-bit binary number to it, then it

returns the output as 1 bit incremented number i.e. it adds 1 to the given four bit

number.

For example: If the user inputs a four bit binary number as 1010. Then the output will

be 1010 + 1 = 1011

**1010**

**+ 1**

**\_\_\_\_\_\_\_\_\_**

**1011**

Input: Four-Bit Binary Number A3 A2 A1 A0.

Output: Four-Bit Binary Number, i.e, Sum S3 S2 S1 S0.

**B)** In this, we have to design a combinational circuit that subtracts 1 from a given

four-bit binary number. It means if the user inputs a four-bit binary number to it,

then it returns the output as 1 bit decremented number i.e. it subtracts 1 from the

given four bit number.

For example: If the user inputs a four bit binary number as 1011. Then the output will be 1011 - 1 = 1010

**1 0 1 1**

**- 1**

**\_\_\_\_\_\_\_\_\_**

**1 0 1 0**

Input: Four-Bit Binary Number A3 A2 A1 A0.

Output: Four-Bit Binary Number, i.e, Difference D3 D2 D1 D0.

II. Highlighting the constraints.

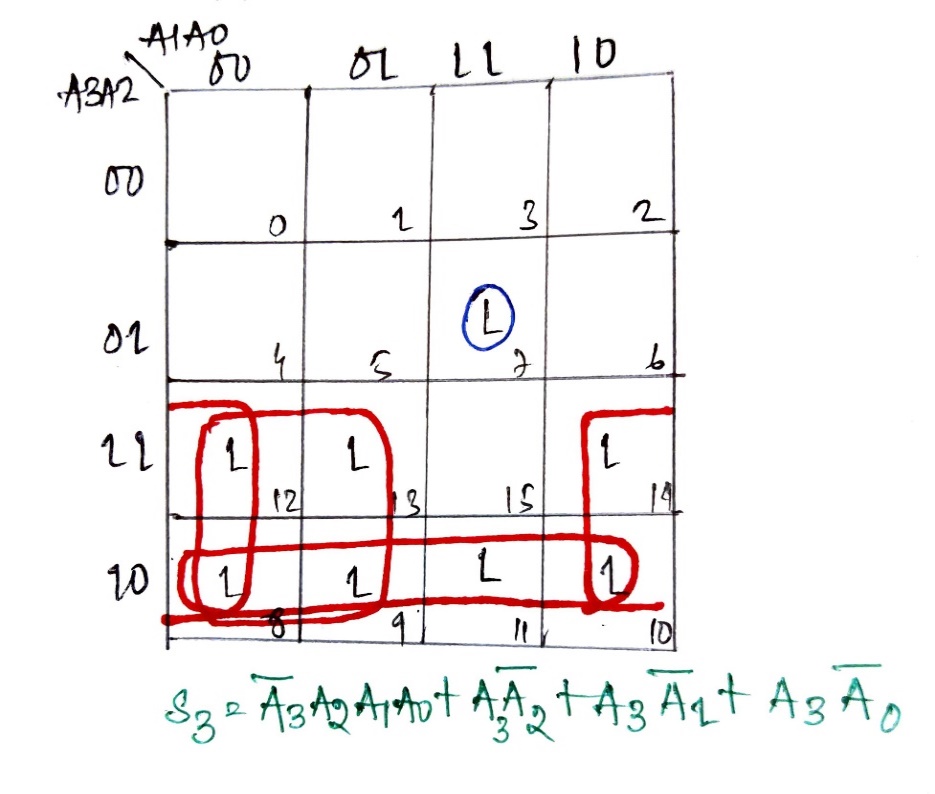
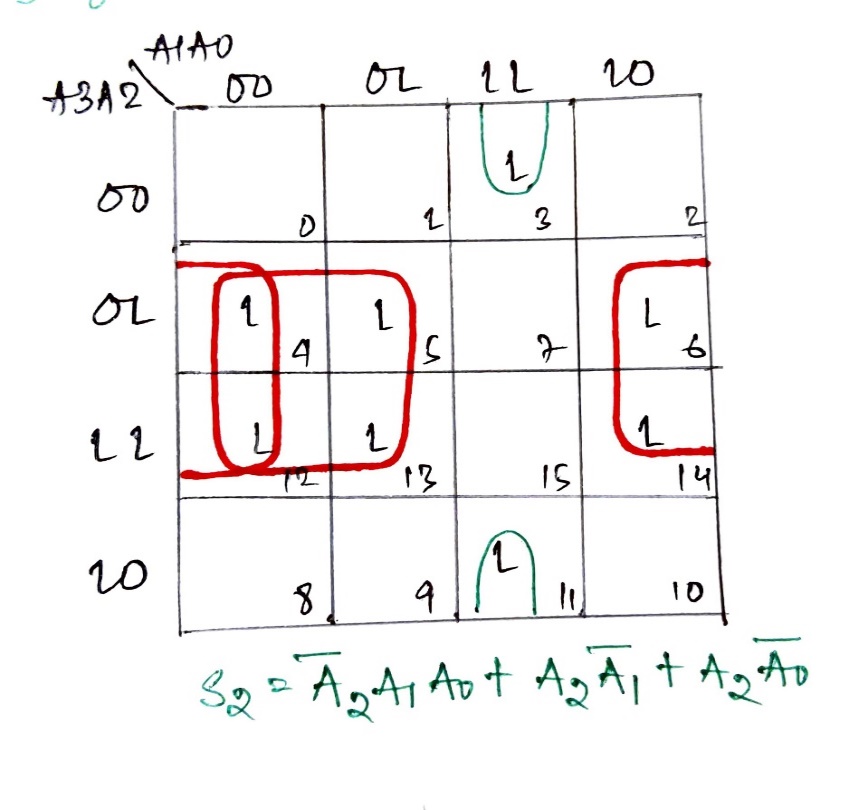
* It is less efficient in terms of speed and hardware complexity compared to mux.
* It consumes 35% more energy than multiplexer based design.
* The delay is 40% more than multiplexer design.

# **3. Methodology**

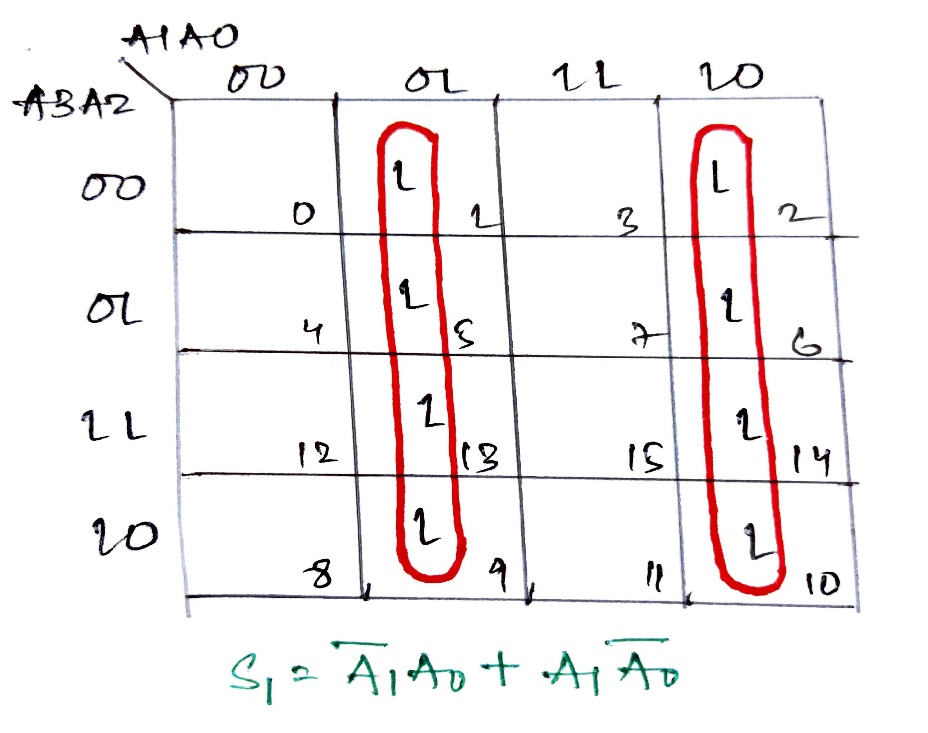
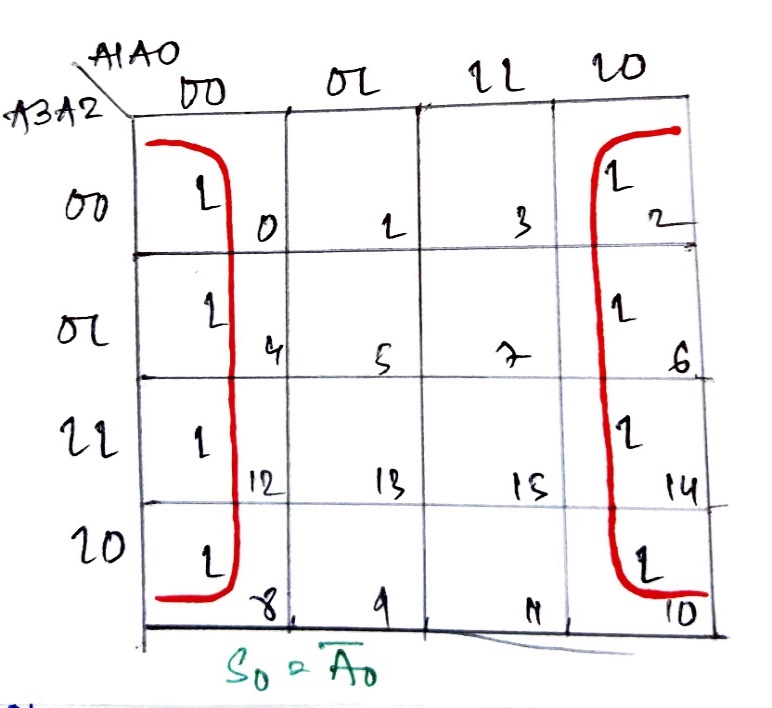
I. Generating the solution to the problem by the use of Truth table, K-map and (or) Boolean algebra.

**A) 4-Bit Incrementer**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **A3** | **A2** | **A1** | **A0** | **S3** | **S2** | **S1** | **S0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |



­



Boolean algebraic expressions for:

S3 = (A3’.A2.A1.A0) + (A3.A2’) + (A3.A1’) + (A3.A0’)

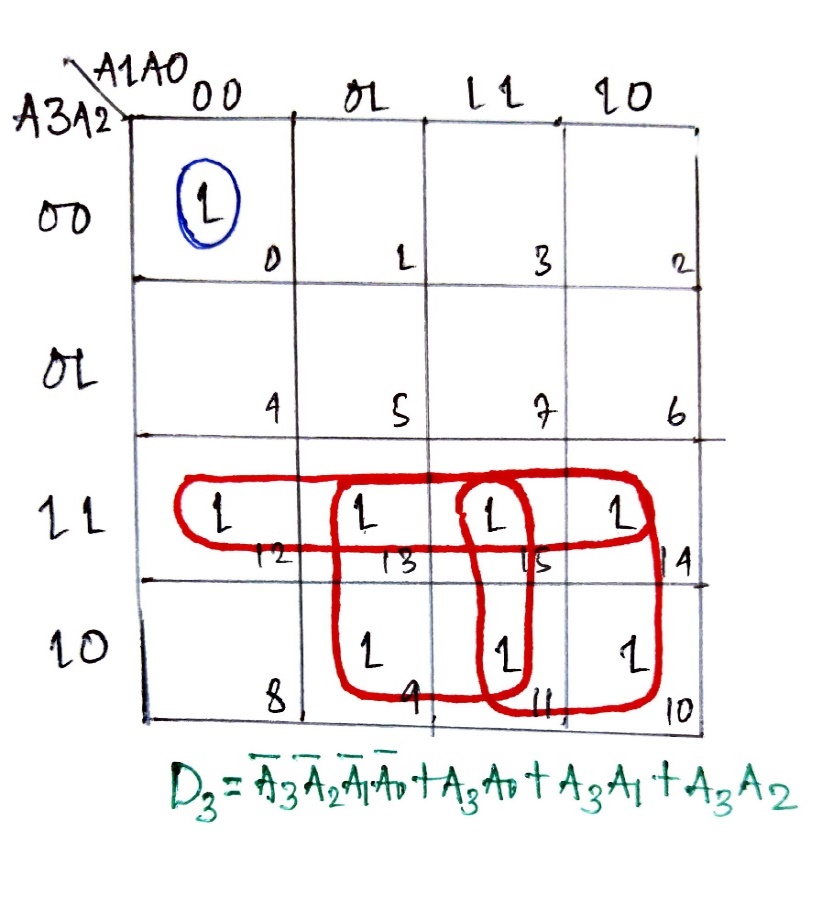
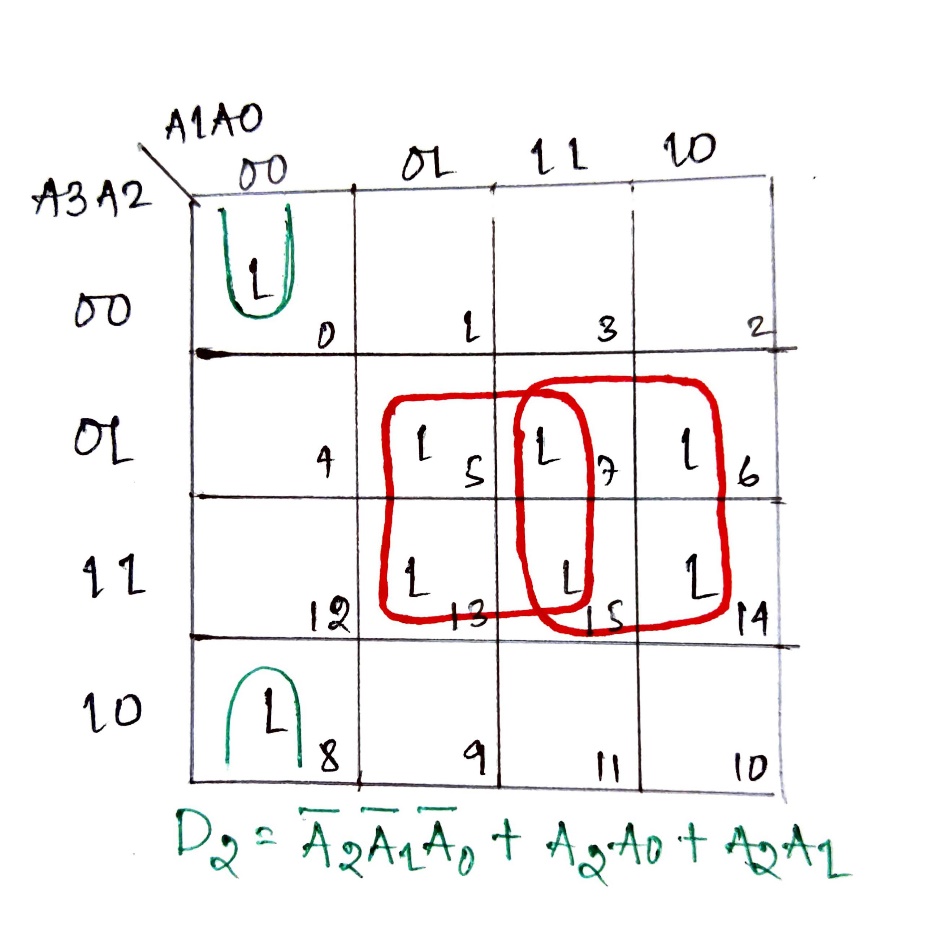
S2 = (A2’.A1.A0) + (A2.A1’) + (A2.A0’)

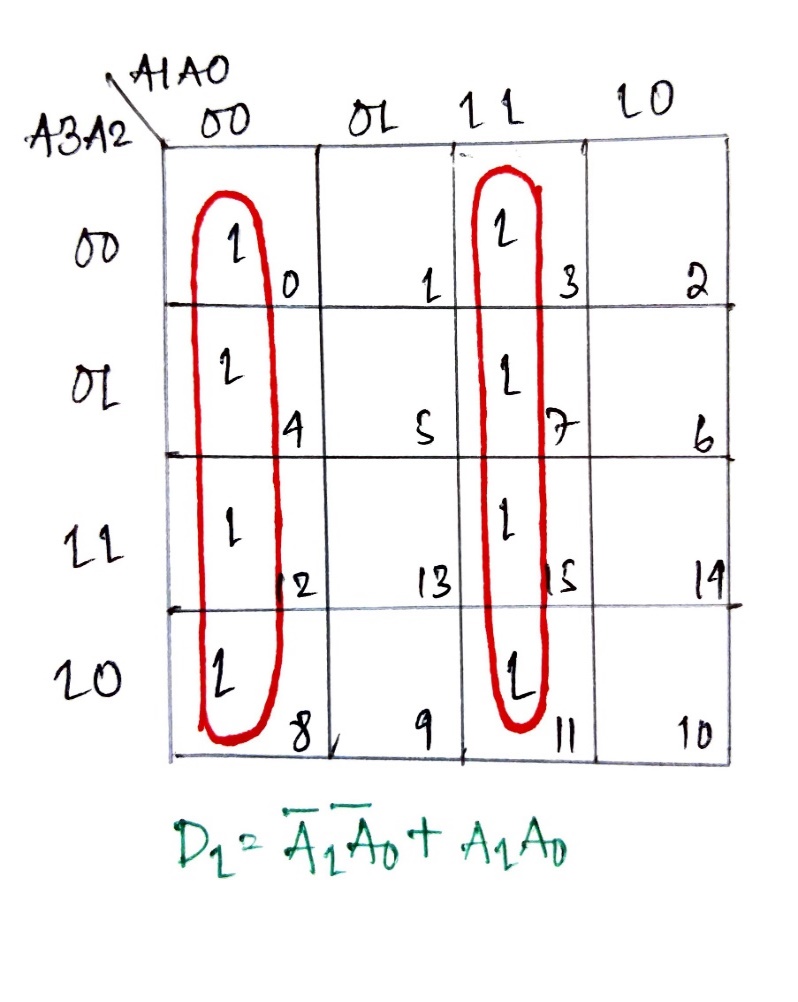
S1 = (A1’.A0) + (A1.A0’)

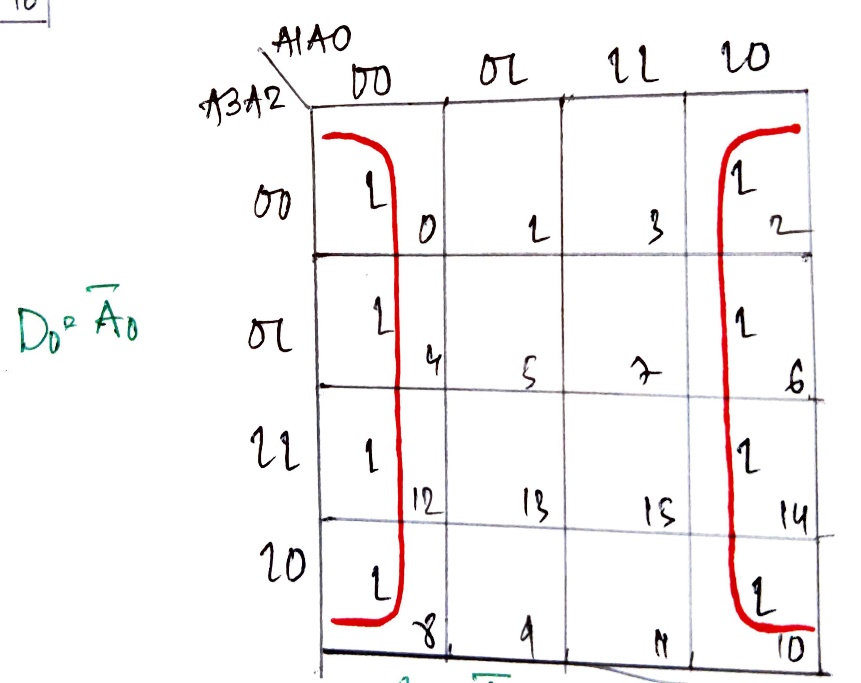
S0 = A0’

**B) 4-Bit Decrementer**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **A3** | **A2** | **A1** | **A0** | **D3** | **D2** | **D1** | **D0** |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |







Boolean algebraic expressions for:

D3 = (A3’.A2’.A1’.A0’) + (A3.A2) + (A3.A1) + (A3.A0)

D2 = (A2’.A1’.A0’) + (A2.A1) + (A2.A0)

D1 = (A1’.A0’) + (A1.A0)

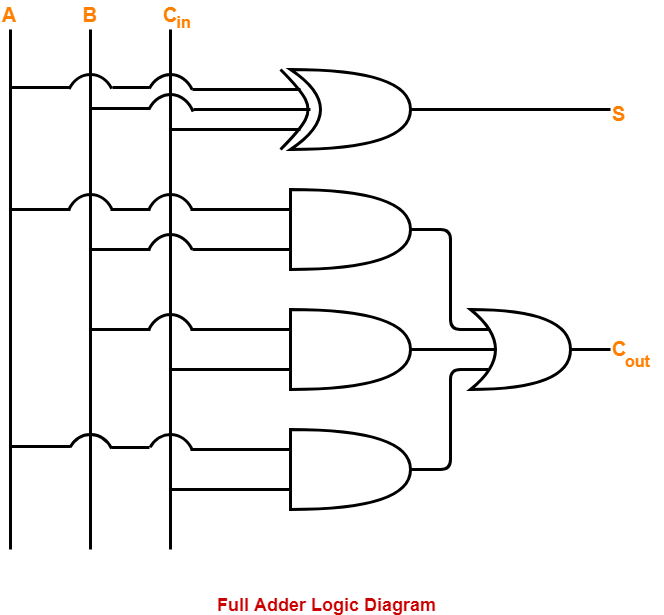
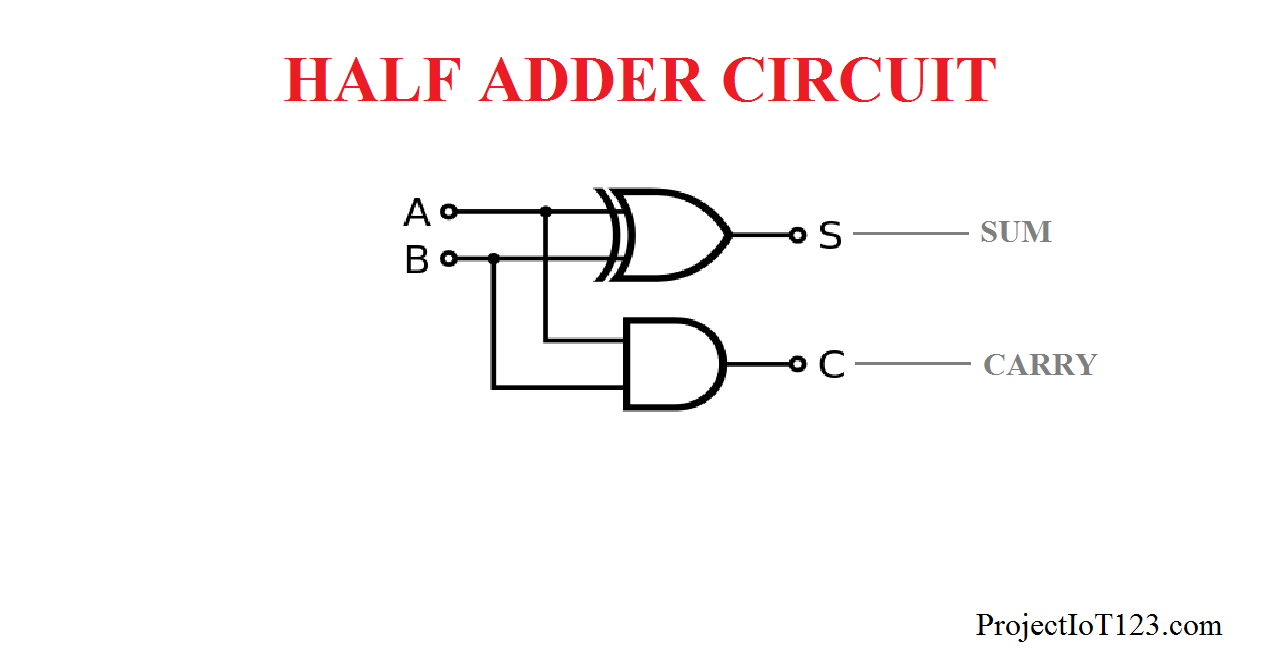
D0 = A0’

II. Finding out the different digital ICs to be used in the optimized design.

**A)** For the 4-bit Incrementer, we used four half adders. Each half adder consists of a

XOR gate IC 7486 and an AND gate IC 7408.

So to design the entire combinational circuit, four XOR gates IC 7486 and four AND gates IC 7408 are required.



**B)** For the 4-bit Decrementer, we used threefull adders and one half adder. Each full adder consists of a XOR gate IC 7486 , three AND gates IC 7408 and an OR gate IC 7432.

So to design the entire combinational circuit, four XOR gates IC 7486,ten AND gates IC 7408 and three OR gates IC 7432 are required.

# **4. Implementation**

I. Drawing the logic diagram using different logic gates.

**A) 4-Bit Incrementer**

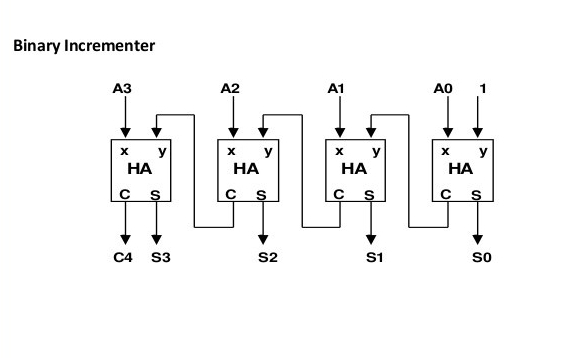


Fig. Block Diagram

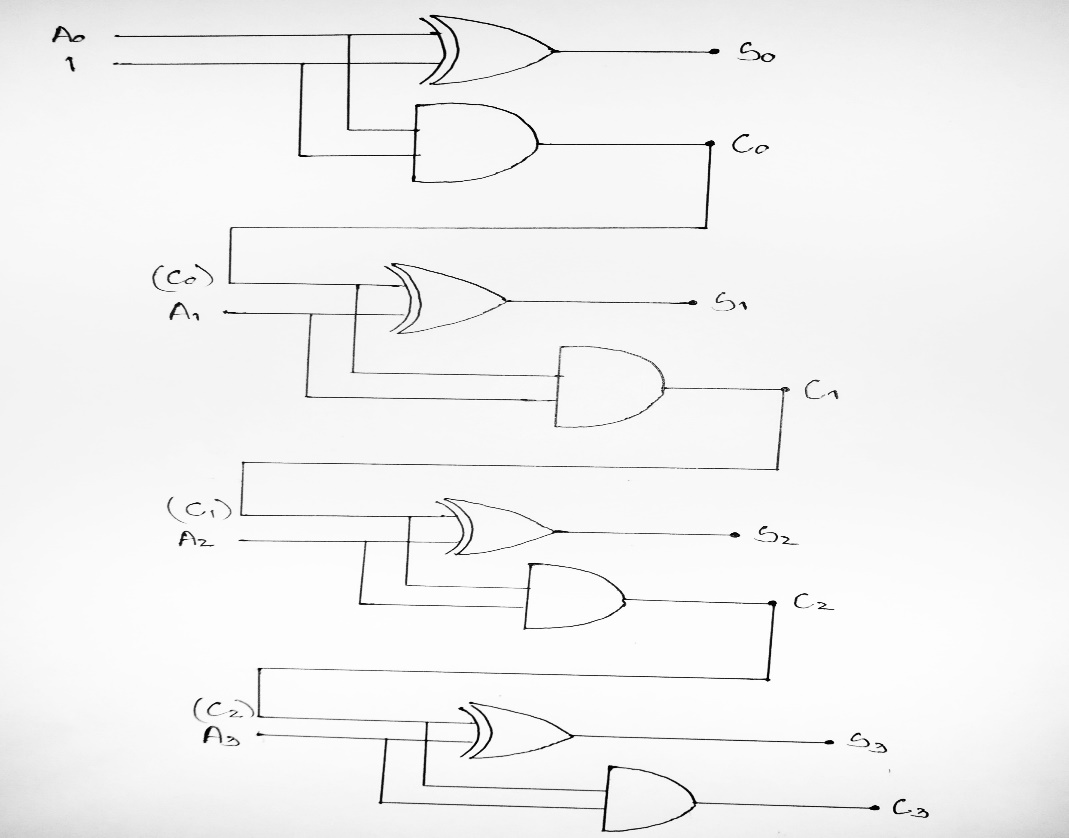


Fig. Logic Diagram

**B) 4-Bit Decrementer**

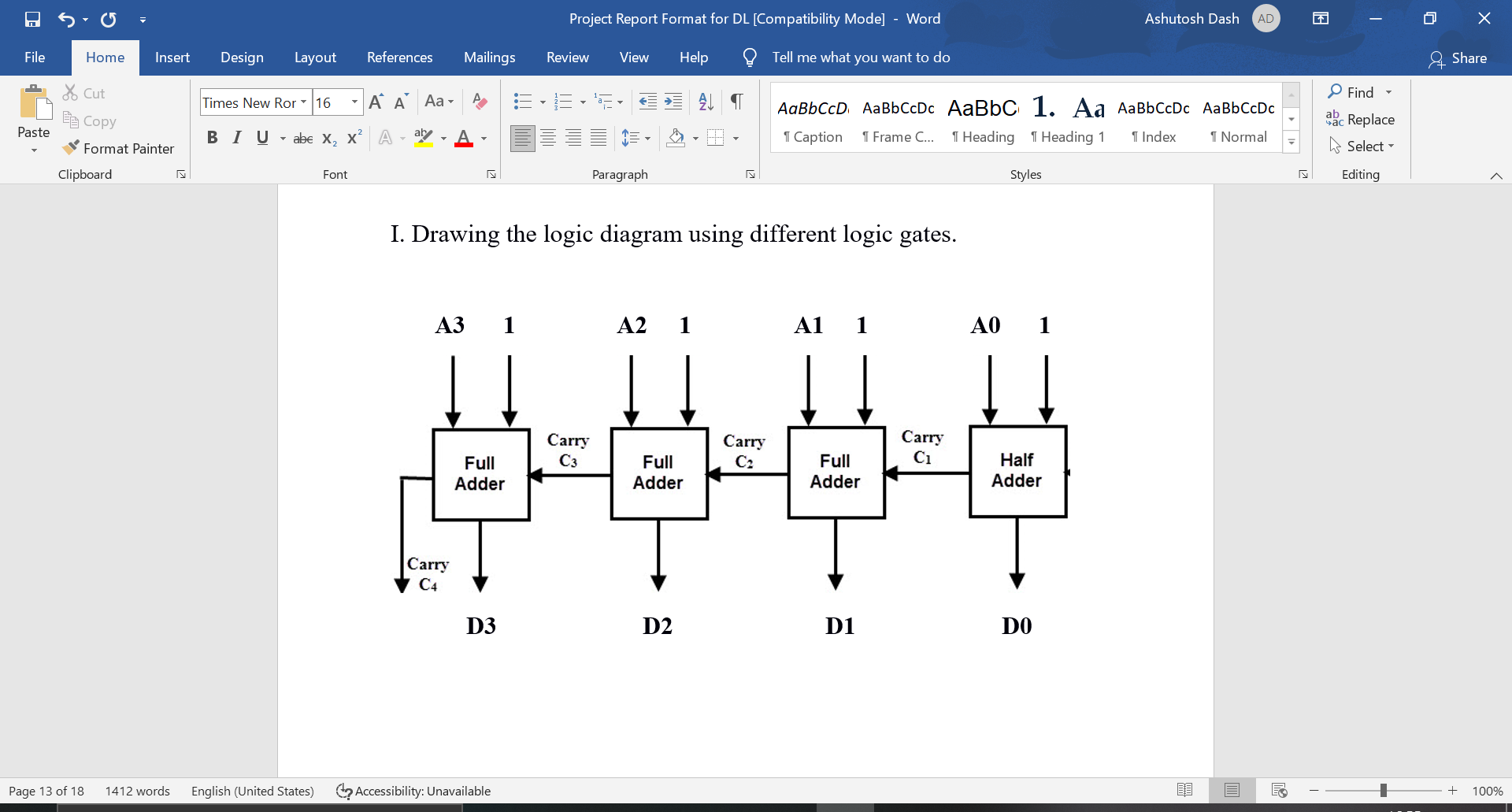


Fig. Block Diagram

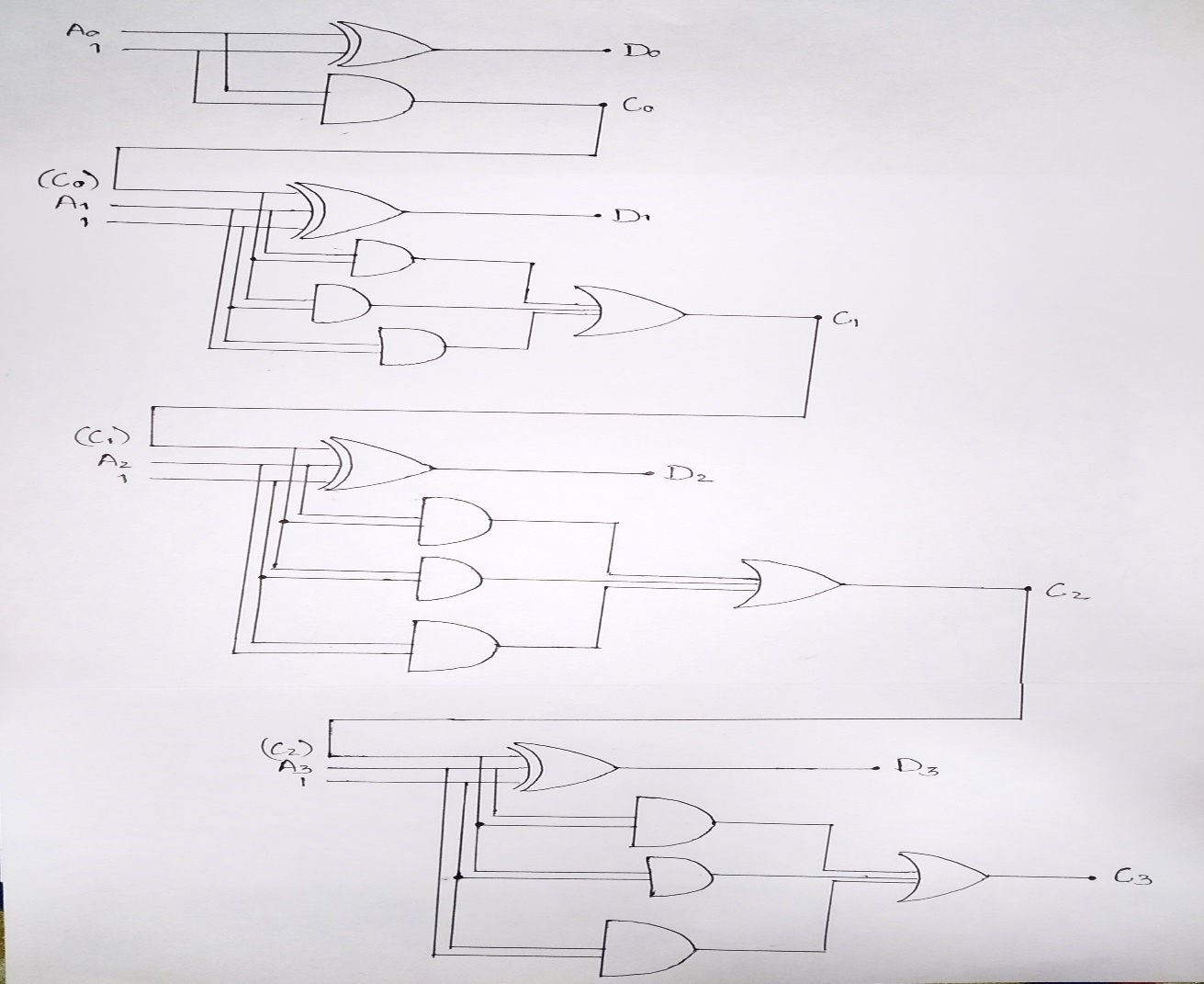


Fig. Logic Diagram

II. HDL Program

**A) 4-Bit Incrementer**

**Dataflow Model :** <https://www.edaplayground.com/x/r9Eb>

|  |  |
| --- | --- |
| **Testbench** | **Design** |
| module bit4Incr\_sp;  reg a3,a2,a1,a0;  wire s3,s2,s1,s0, c3,c2,c1,c0;  bit4Incr bit4Incr\_sp(a3,a2,a1,a0, s3,s2,s1,s0, c3,c2,c1,c0);  initial  begin  $dumpfile("dump.vcd");  $dumpvars(1, bit4Incr\_sp);  $display("A3 A2 A1 A0 | S3 S2 S1 S0");  $monitor(a3,,,,a2,,,,a1,,,,a0,,,,"|",,,,s3,,,,s2,,,,s1,,,,s0);  a3<=0;  a2<=0;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=1;  a0<=0;  #1a3<=0;  a2<=1;  a1<=1;  a0<=1;  #1  a3<=1;  a2<=0;  a1<=0;  a0<=0;  #1a3<=1;  a2<=0;  a1<=0;  a0<=1;  #1a3<=1;  a2<=0;  a1<=1;  a0<=0;  #1a3<=1;  a2<=0;  a1<=1;  a0<=1;  #1a3<=1;  a2<=1;  a1<=0;  a0<=0;  #1a3<=1;  a2<=1;  a1<=0;  a0<=1;  #1a3<=1;  a2<=1;  a1<=1;  a0<=0;  #1a3<=1;  a2<=1;  a1<=1;  a0<=1;  $finish();  end  endmodule | module bit4Incr(A3,A2,A1,A0, S3,S2,S1,S0, C3,C2,C1,C0);  input A3,A2,A1,A0;  output S3,S2,S1,S0, C3,C2,C1,C0;  assign S0 = A0^1;  assign C0 = A0;  assign S1 = A1^C0;  assign C1 = A1&&C0;  assign S2 = A2^C1;  assign C2 = A2&&C1;  assign S3 = A3^C2;  assign C3 = A3&&C2;  endmodule |

**Gatelevel Model:**  <https://www.edaplayground.com/x/HLNG>

|  |  |
| --- | --- |
| **Testbench** | **Design** |
| module bit4Incr\_sp;  reg a3,a2,a1,a0;  wire s3,s2,s1,s0, c3,c2,c1,c0;  bit4Incr bit4Incr\_sp(a3,a2,a1,a0, s3,s2,s1,s0, c3,c2,c1,c0);  initial  begin  $dumpfile("dump.vcd");  $dumpvars(1, bit4Incr\_sp);  $display("A3 A2 A1 A0 | S3 S2 S1 S0");  $monitor(a3,,,,a2,,,,a1,,,,a0,,,,"|",,,,s3,,,,s2,,,,s1,,,,s0);  a3<=0;  a2<=0;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=1;  a0<=0;  #1a3<=0;  a2<=1;  a1<=1;  a0<=1;  #1  a3<=1;  a2<=0;  a1<=0;  a0<=0;  #1a3<=1;  a2<=0;  a1<=0;  a0<=1;  #1a3<=1;  a2<=0;  a1<=1;  a0<=0;  #1a3<=1;  a2<=0;  a1<=1;  a0<=1;  #1a3<=1;  a2<=1;  a1<=0;  a0<=0;  #1a3<=1;  a2<=1;  a1<=0;  a0<=1;  #1a3<=1;  a2<=1;  a1<=1;  a0<=0;  #1a3<=1;  a2<=1;  a1<=1;  a0<=1;  $finish();  end  endmodule | module bit4Incr(A3,A2,A1,A0, S3,S2,S1,S0, C3,C2,C1,C0);  input A3,A2,A1,A0;  output S3,S2,S1,S0, C3,C2,C1,C0;  xor(S0,A0,1);  buf(C0,A0);  xor(S1,A1,C0);  and(C1,A1,C0);  xor(S2,A2,C1);  and(C2,A2,C1);  xor(S3,A3,C2);  and(C3,A3,C2);  endmodule |

**A) 4-Bit Decrementer**

**Dataflow Model:**  <https://www.edaplayground.com/x/rj6R>

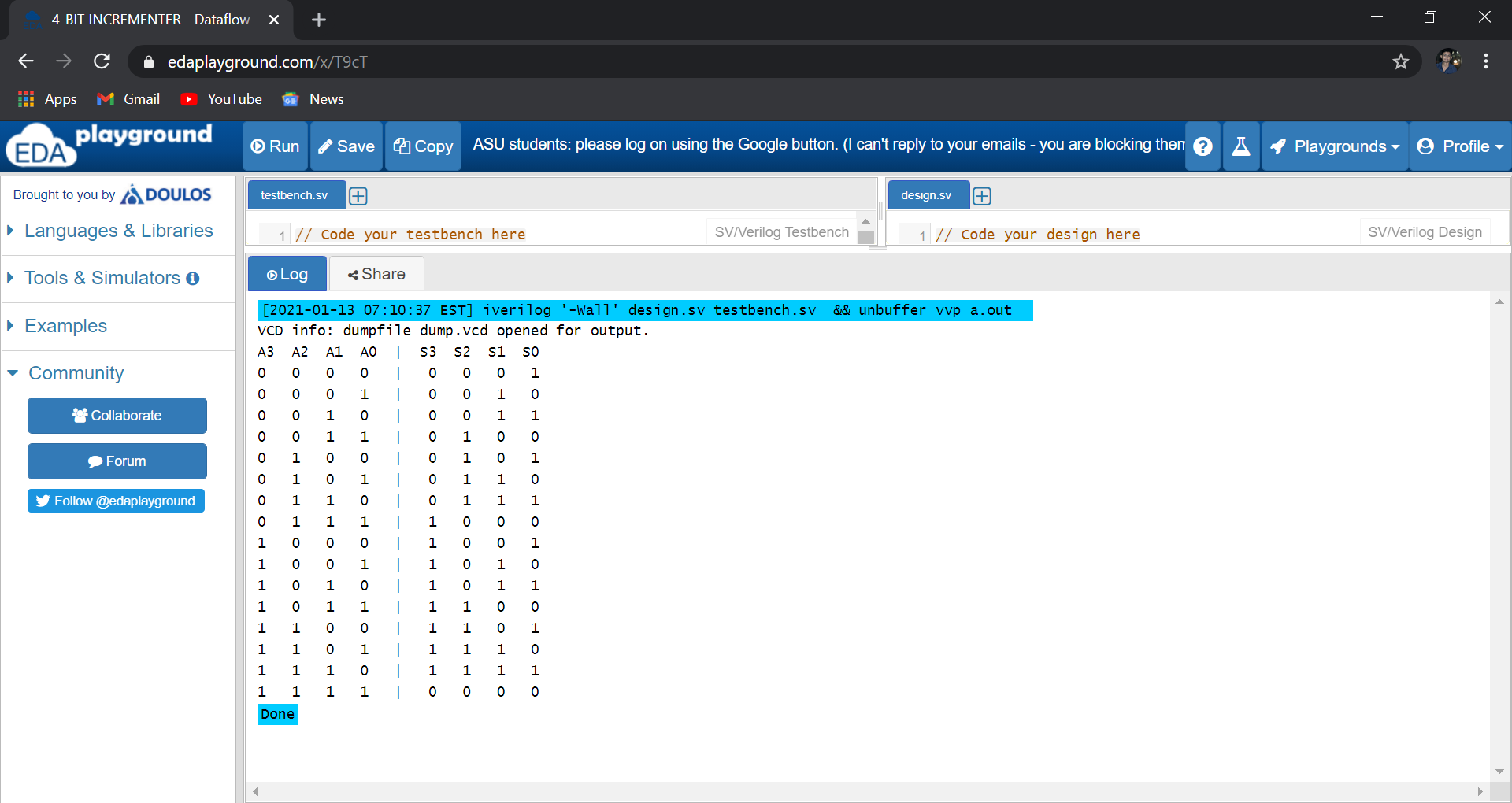
|  |  |
| --- | --- |
| **Testbench** | **Design** |
| module bit4Decr\_sp;  reg a3,a2,a1,a0;  wire d3,d2,d1,d0, c3,c2,c1,c0;  bit4Decr bit4Decr\_sp(a3,a2,a1,a0, d3,d2,d1,d0, c3,c2,c1,c0);  initial  begin  $dumpfile("dump.vcd");  $dumpvars(1, bit4Decr\_sp);  $display("A3 A2 A1 A0 | D3 D2 D1 D0");  $monitor(a3,,,,a2,,,,a1,,,,a0,,,,"|",,,,d3,,,,d2,,,,d1,,,,d0);  a3<=0;  a2<=0;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=1;  a0<=0;  #1a3<=0;  a2<=1;  a1<=1;  a0<=1;  #1  a3<=1;  a2<=0;  a1<=0;  a0<=0;  #1a3<=1;  a2<=0;  a1<=0;  a0<=1;  #1a3<=1;  a2<=0;  a1<=1;  a0<=0;  #1a3<=1;  a2<=0;  a1<=1;  a0<=1;  #1a3<=1;  a2<=1;  a1<=0;  a0<=0;  #1a3<=1;  a2<=1;  a1<=0;  a0<=1;  #1a3<=1;  a2<=1;  a1<=1;  a0<=0;  #1a3<=1;  a2<=1;  a1<=1;  a0<=1;  $finish();  end  endmodule | module bit4Decr(A3,A2,A1,A0, D3,D2,D1,D0, C3,C2,C1,C0);  input A3,A2,A1,A0;  output D3,D2,D1,D0, C3,C2,C1,C0;  assign D0 = A0^1;  assign C0 = A0;  assign D1 = A1^C0^1;  assign C1 = C0|A1|(A1&&C0);  assign D2 = A2^C1^1;  assign C2 = C1|A2|(A2&&C1);  assign D3 = A3^C2^1;  assign C3 = C2|A3|(A3&&C2);  endmodule |

**Gatelevel Model:**  <https://www.edaplayground.com/x/CUMn>

|  |  |
| --- | --- |
| **Testbench** | **Design** |
| module bit4Decr\_sp;  reg a3,a2,a1,a0;  wire d3,d2,d1,d0, c3,c2,c1,c0;  bit4Decr bit4Decr\_sp(a3,a2,a1,a0, d3,d2,d1,d0, c3,c2,c1,c0);  initial  begin  $dumpfile("dump.vcd");  $dumpvars(1, bit4Decr\_sp);  $display("A3 A2 A1 A0 | D3 D2 D1 D0");  $monitor(a3,,,,a2,,,,a1,,,,a0,,,,"|",,,,d3,,,,d2,,,,d1,,,,d0);  a3<=0;  a2<=0;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=0;  #1  a3<=0;  a2<=0;  a1<=1;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=0;  #1  a3<=0;  a2<=1;  a1<=0;  a0<=1;  #1  a3<=0;  a2<=1;  a1<=1;  a0<=0;  #1a3<=0;  a2<=1;  a1<=1;  a0<=1;  #1  a3<=1;  a2<=0;  a1<=0;  a0<=0;  #1a3<=1;  a2<=0;  a1<=0;  a0<=1;  #1a3<=1;  a2<=0;  a1<=1;  a0<=0;  #1a3<=1;  a2<=0;  a1<=1;  a0<=1;  #1a3<=1;  a2<=1;  a1<=0;  a0<=0;  #1a3<=1;  a2<=1;  a1<=0;  a0<=1;  #1a3<=1;  a2<=1;  a1<=1;  a0<=0;  #1a3<=1;  a2<=1;  a1<=1;  a0<=1;  $finish();  end  endmodule | module bit4Decr(A3,A2,A1,A0, D3,D2,D1,D0, C3,C2,C1,C0);  input A3,A2,A1,A0;  output D3,D2,D1,D0, C3,C2,C1,C0;  xor(D0,A0,1);  buf(C0,A0);  xor(D1,A1,C0,1);  and(w1,A1,C0);  or(C1,C0,A1,w1);  xor(D2,A2,C1,1);  and(w2,A2,C1);  or(C2,A2,C1,w2);  xor(D3,A3,C2,1);  and(w3,A3,C2);  or(C3,A3,C2,w3);  endmodule |

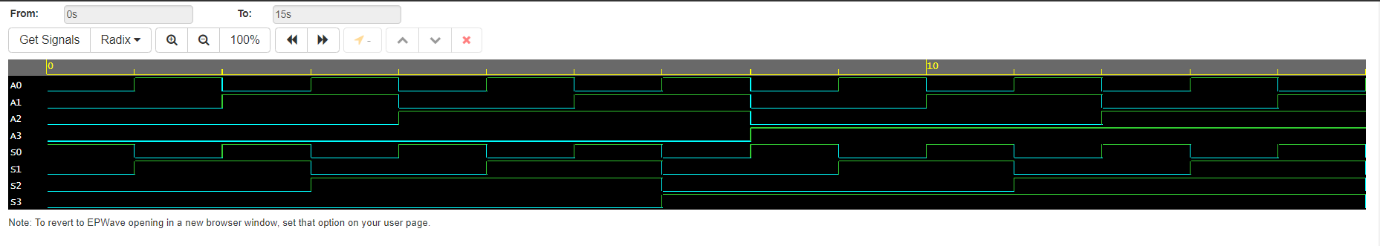
# **5. Results & Interpretation**

**A) 4-Bit Incrementer**

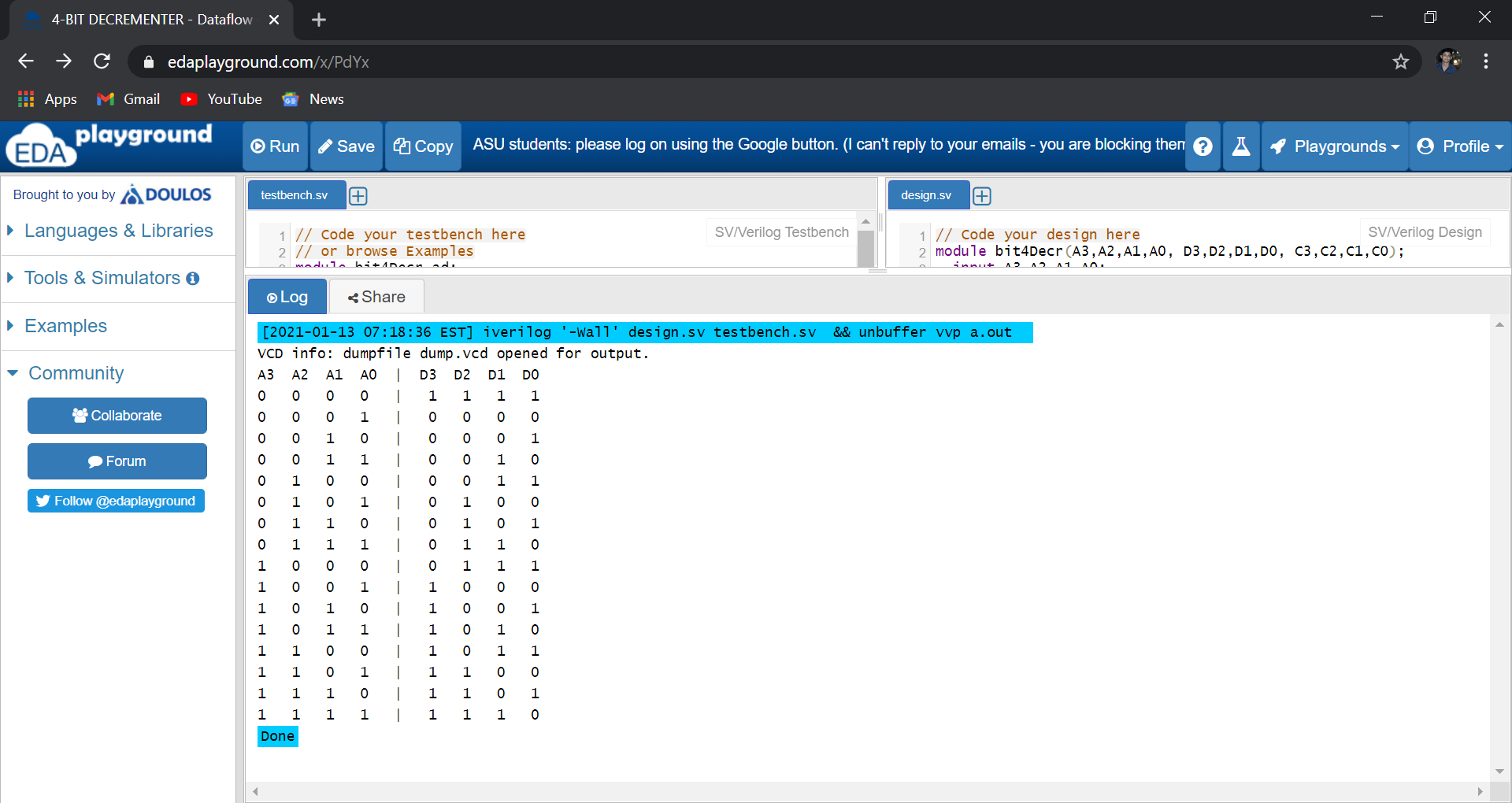
Truth Table

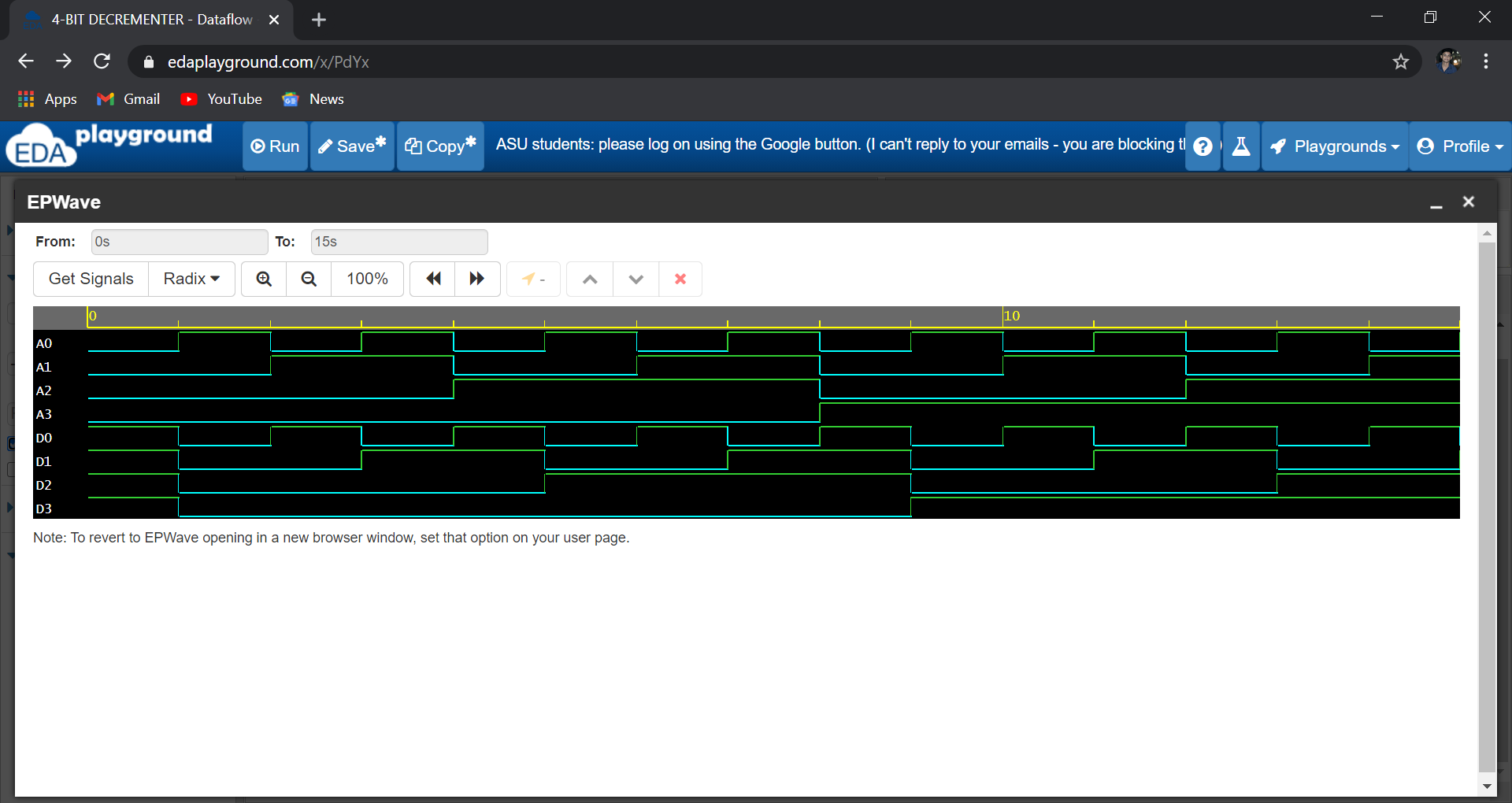
P

**EP WAVE**



**B) 4-Bit Decrementer**

Truth Table

EP Wave

# **6. Conclusion**

After performing this project on the four bit incrementer and decrementer, we got to know many things about various logic gates ICs and designing combinational circuits and writing the HDL program for the problem. In this project we have mainly used full adders and half adders for our objectives. Both half adder and full adder comes under the category of combinational logic circuits that are used for arithmetic operations. However, the major difference between half adder and the full adder is that the half adder operates on 2 inputs. On the other hand, full adder operates on 3 inputs. We have used 3 full adders and 1 half adder to design the four bit decrementer but we can also use four full adders for the same.

With the verilog HDL program for the incrementer and decrementer circuits we also concluded and verified that the results we get after running the code in form of EP Waves are same as that of the truth table.

# **7. References**

1. Digital Logic Design, 6th Edition, M. Morris Mano, Michael D Ciletti,California State University, Los Angeles

2. Introduction to Digital Logic Design, Hayes, 1993, Wesley.

3. Bhasker, 1997, A Verilog HDL Primer, Allentown

4. GeeksForGeeks, tutorialspoint, NESO Academy