CSARCH1 Design Problem
S14

Submitted by:
Homssi, Yazan

Submitted to:
Best prof, Sir Carlo Adriano

April 5, 2024

Outline of Documentation:

- *State Table*
- *Karnaugh Maps*
- *Logic Diagram*
- *Circuit Outputs*
- *Verilog Code*

==========

1. State table

In this Design Problem, we are supposed to design a circuit that outputs the following numbers: *[1,1,2,2,4,2,6,4,6,4,**10**,4,**12**,6,8,8]* in order and display the results on a 7-segment display, although 10 and 12 will be displayed as A and C respectively. So the display should actually be: *[1,1,2,2,4,2,6,4,6,4, **A**,4, **C**,6,8,8]*. It will then repeat, showing the pattern again. The input *R*, resets the sequence to the first element. This is very similar to Design Exercise 1. Two images for your reference down below:

Much like in our recent lessons & from the exercises in PS10, we can start by creating a `state table` as it will help us progress throughout the Design Problem down below (Let us assume the Clock is always 1):

Inputs	Present State (Qn)				Next State (Qn+1)				D Flip-Flop Inputs			
R	A	B	C	D	A^+	B^+	C^+	D^+	DA	DB	DC	DD
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1	1	0	0	1	1
0	0	0	1	1	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	1	0	1
0	0	1	0	1	0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1	1	0	1	1	1
0	0	1	1	1	1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	1	1	0	0	1
0	1	0	0	1	1	0	1	0	1	0	1	0
0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
0	1	0	1	1	1	1	0	0	1	1	0	0

0	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
0	1	1	0	1	1	1	1	0	1	1	1	0
0	1	1	1	0	1	1	1	1	1	1	1	1
0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0
1	**1**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	**1**	**1**	**0**	**0**	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0

Explanation of the state table:

==========

Inputs: 'R', this is the only external input I need in order to affect the state of the machine, and reset the display to the very beginning.

Present State (Qn): Current state of the machine (A,B,C,D), since it is 4 bits. I have 16 states/rows.

Next State (Qn+1): The state that the machine will transition to in the next clock cycle. If the input 'R' is high (1), the next state is always 0000, regardless of the present state or the clock input. If the input 'R' is low (0), the next state is the next value in the sequence.

Flip Flop Inputs: In this design problem, I am going with the `D Flip Flop` because of its simplicity as the output will just simply follow the input at the next clock edge.
Therefore the FF Inputs are simply the next state values:
$A^+ = DA$, $B^+ = DB$, $C^+ = DC$, $D^+ = DD$

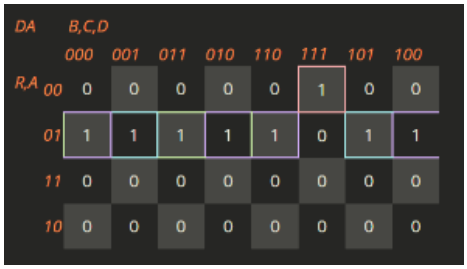
Outputs: The outputs are A, B, C, D, E, F, G. It is currently not in the state table, it will be shown and explained in part *`3: Logic Diagrams`* a few pages down.

==========

2. Karnaugh Maps

What comes next after the State Table? You guessed it! In order to get the equations needed to help me design our logic diagram in CircuitVerse, I made use of Karnaugh Maps (K-maps) in getting the Sum of Product (SOP), which would then lead us to getting the boolean expression needed for our next part. Inputs used in the K-maps: *R,A,B,C,D*

Output DA:



SOP Equation DA(R, A, B, C, D) = ∑m(7,8,9,10,11,12,13,14)
DA(R, A, B, C, D) = R'A'BCD + R'AB' + R'AC' + R'AD'

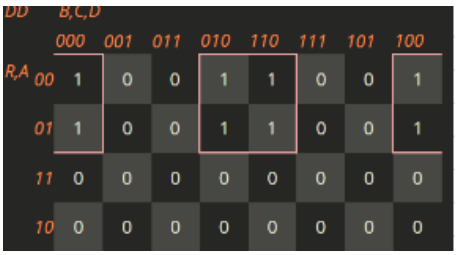
Output DB:

SOP Equation DB(R, A, B, C, D) = ∑m(3,4,5,6,11,12,13,14)
DB(R, A, B, C, D) = R'B'CD + R'BC' + R'BD'

Output DC:

SOP Equation DC(R, A, B, C, D) = ∑m(1,2,5,6,9,10,13,14)
DC(R, A, B, C, D) = R'C'D + R'CD'

Output DD:

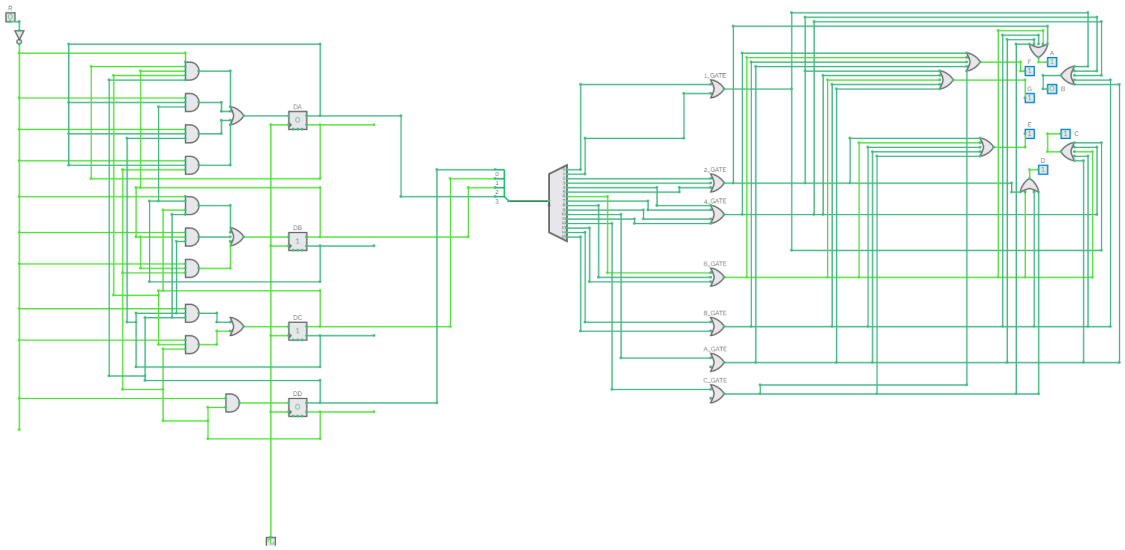


SOP Equation DD(R, A, B, C, D) = ∑m(0,2,4,6,8,10,12,14)
DD(R, A, B, C, D) = R'D'

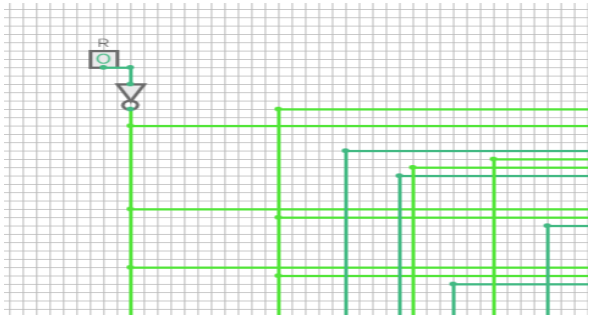
==========

3. Logic Diagram

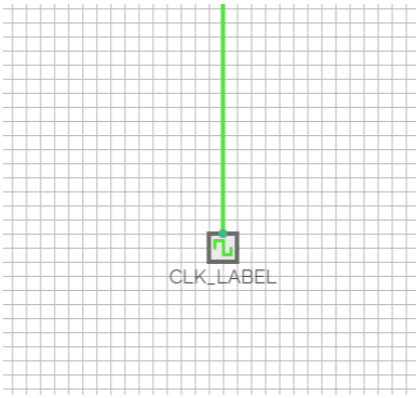
Right after getting the 4 equations from the K-Maps, I then implemented it in Circuit Verse through logic gates/circuits. The images below will show the progress of the diagram from Circuit Verse.

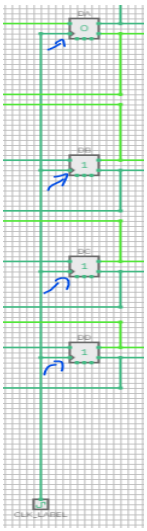


1. Full Image of the logic diagram

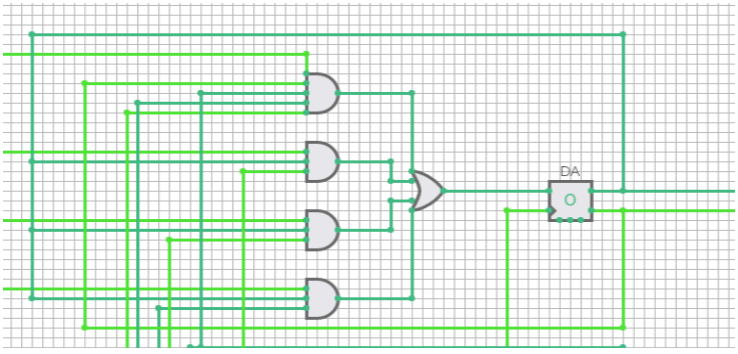


2. The Input ‘R’ that will reset the display back to the state ‘0000’ , which displays ‘1’





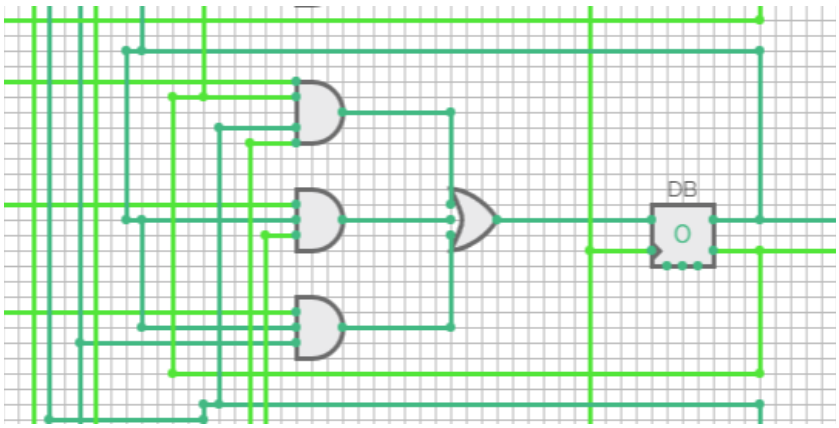
3. Clock Input connected to all 4 flip flops that turns on and off by itself, I cannot really show a GIF here in the PDF sadly



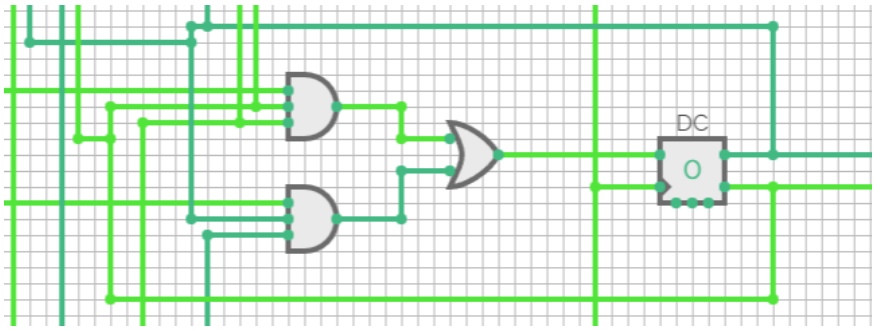
4. DA Flip Flop

Based on our equation: DA(R, A, B, C, D) = R'A'BCD + R'AB' + R'AC' + R'AD'

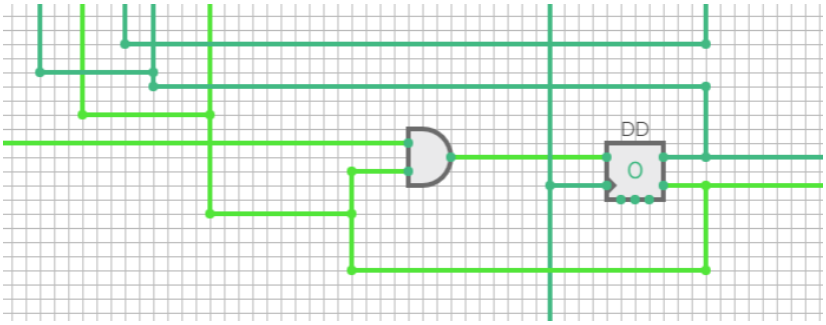
I used 4 AND Gates & 1 OR Gate to connect them all



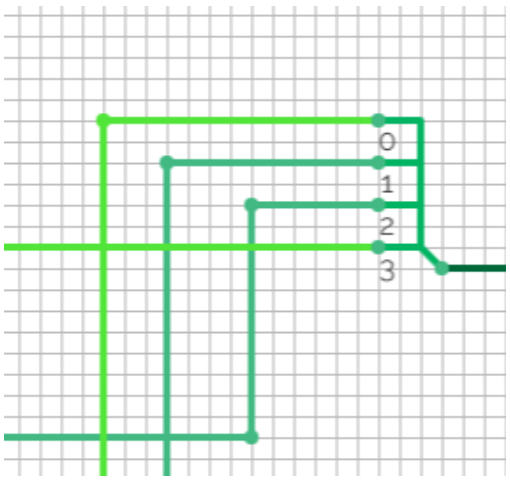
5. DB Flip Flop
Based on our equation: DB(R, A, B, C, D) = R'B'CD + R'BC' + R'BD'
I used 3 AND Gates & 1 OR Gate to connect them all



6. DC Flip Flop
Based on our equation: DC(R, A, B, C, D) = R'C'D + R'CD'
I used 2 AND Gates and 1 OR Gate to connect them both

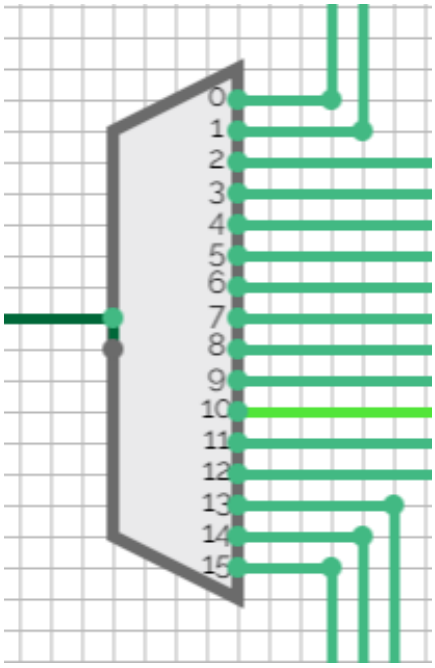


7. DD Flip Flop
Based on our equation: DD(R, A, B, C, D) = R'D'
I only made use of 1 AND Gate going straight to the flip flop

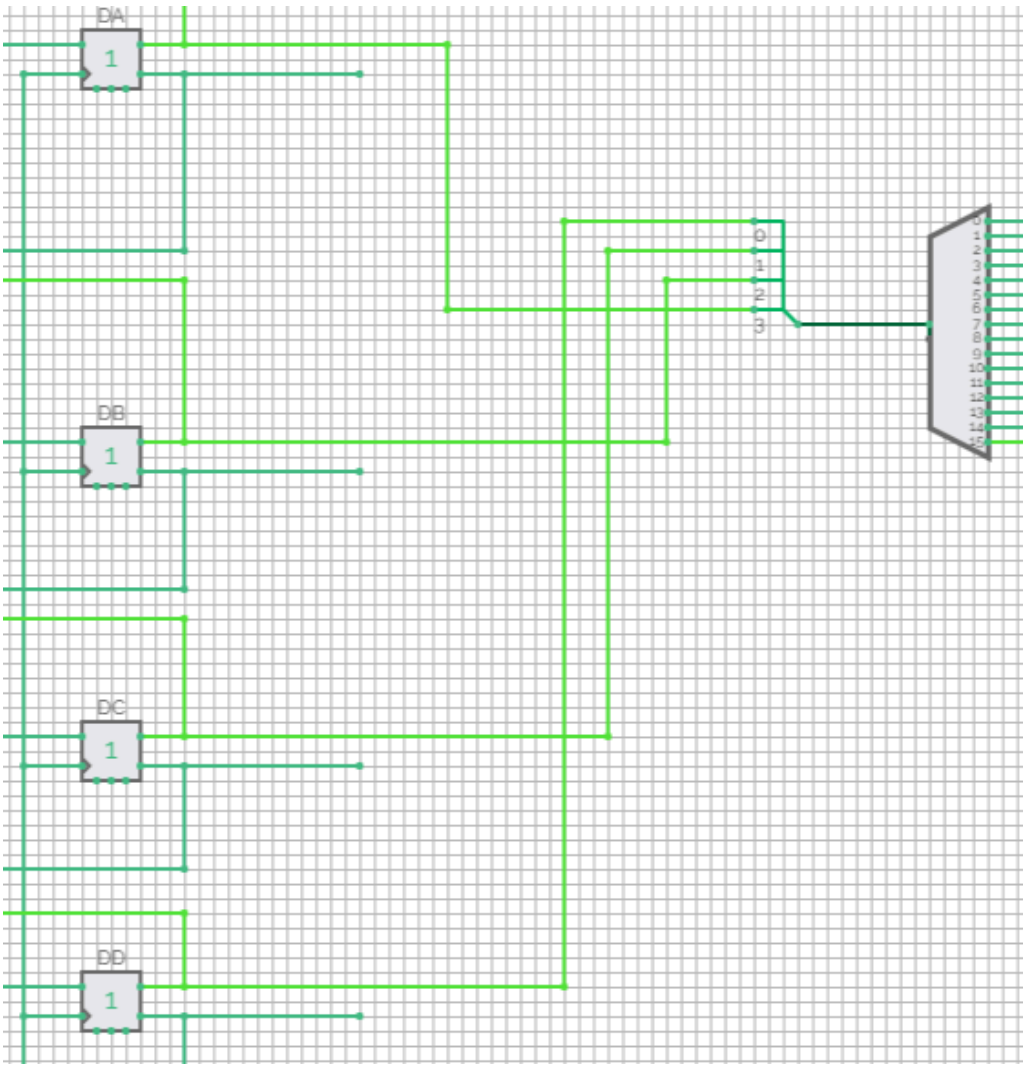


8. Splitter

Normally it would be facing the opposite direction, hence the name `Splitter`.
But in this case, I am combining the 4 different wires from each FF into the Decoder.

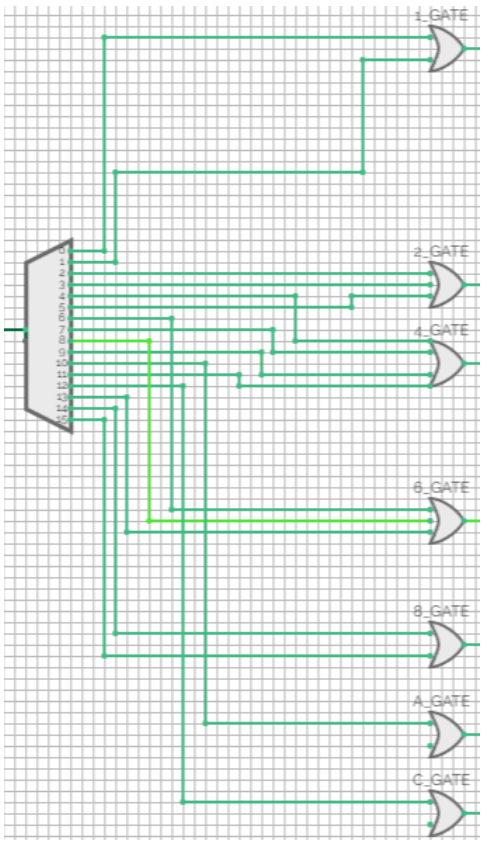


9. Decoder accepts the specific state from the Splitter, and actually acts as some sort of ‘switch case’ to light up the wire one-by-one from 0 to 16, and will help display the specific output needed towards the 7 segment display.



10. FF → Splitter → Decoder

After storing each state (Current number) in the system through the use of `Flip Flops`, they will go through the `Splitter` which combines the 4-bit into a multi-bit out, then decodes into a specific output through the `Decoder`.

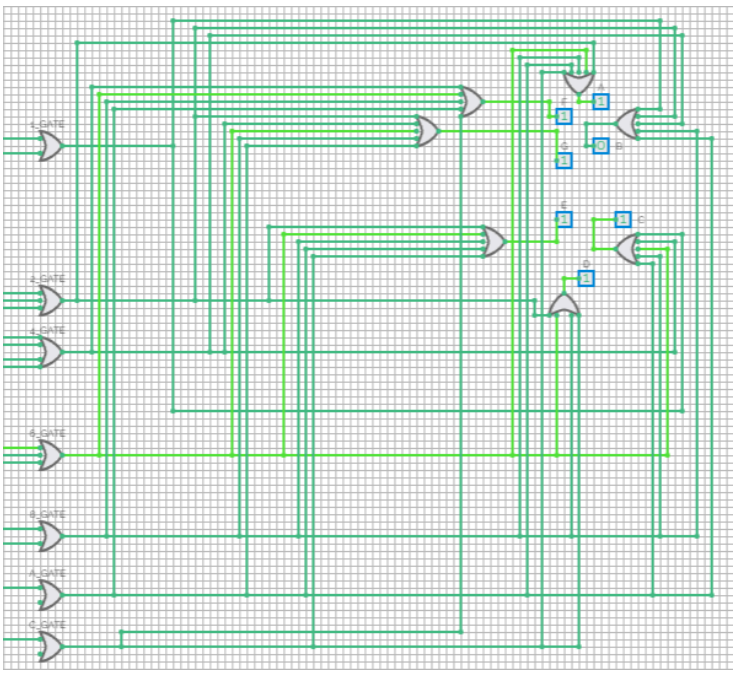


11. Decoder to 7 OR Gates

Following *image 10*, the inputs to the OR gate would be the outputs from the decoder that should light up that particular segment. The output of the OR gate would then be used to drive the corresponding segment of the 7-segment display.

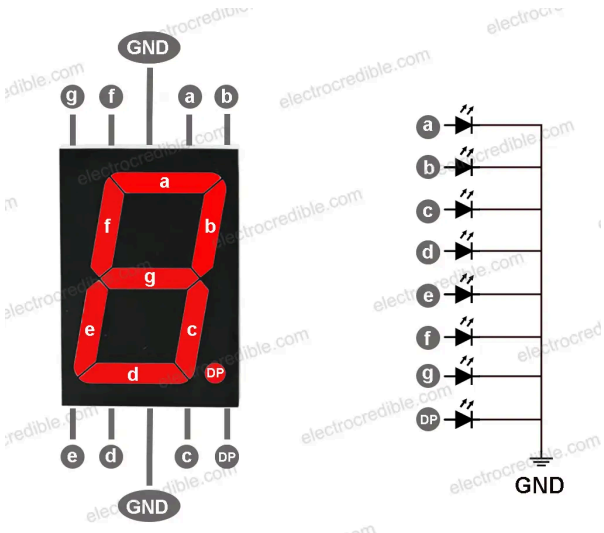
I'd love to point out that the order of the lines from the Decoder matters with the displayed outputs: [1,1,2,2,4,2,6,4,6,4, A,4, C,6,8,8].

Take a look and trace the lines for each.

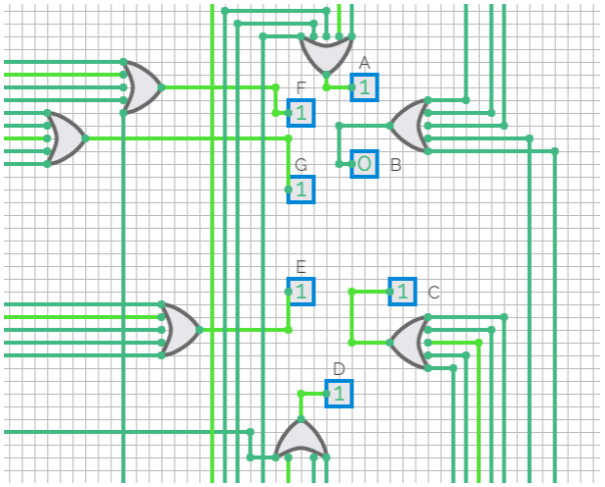


12. The Outputs

There used to be a 7 Segment Display to verify if the logic diagram is correct..
Which it is, I then used 7 Output gates for each letter, and yes the order matter



^ Based on this order



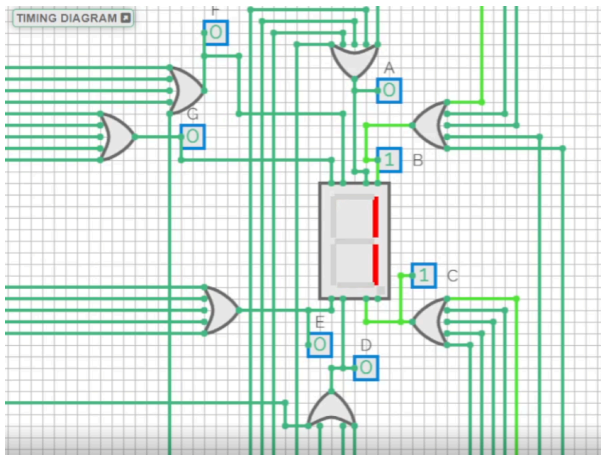
^ Zoomed in photo of the 7 Outputs, each with labels from A-G

==========

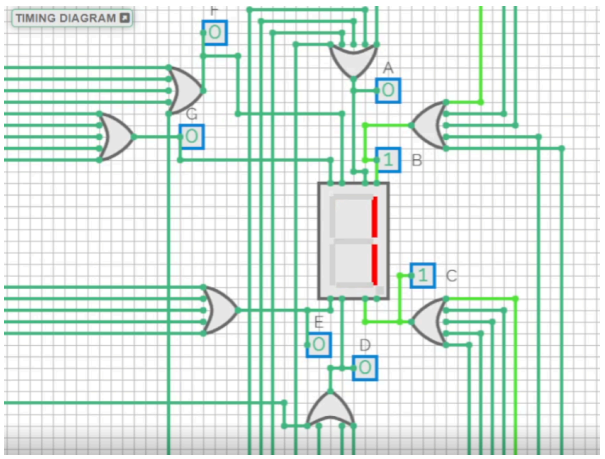
4. Circuit Outputs

This part of the documentation will serve as ‘proof’ that the logic diagram works following this order: [1,1,2,2,4,2,6,4,6,4, A,4, C,6,8,8]

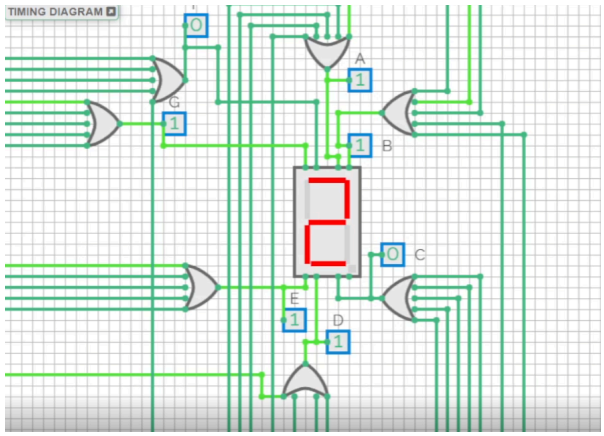
Goes from Left to Right



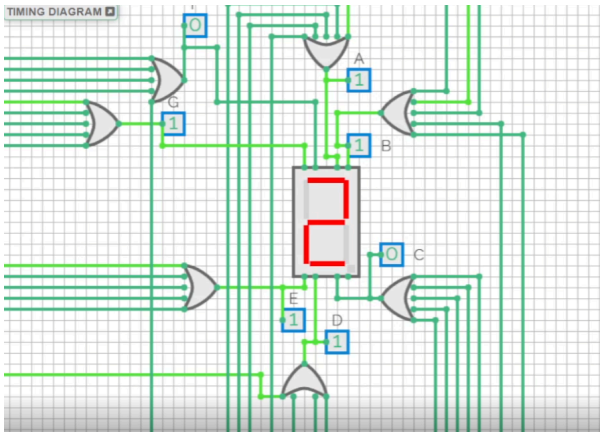
1



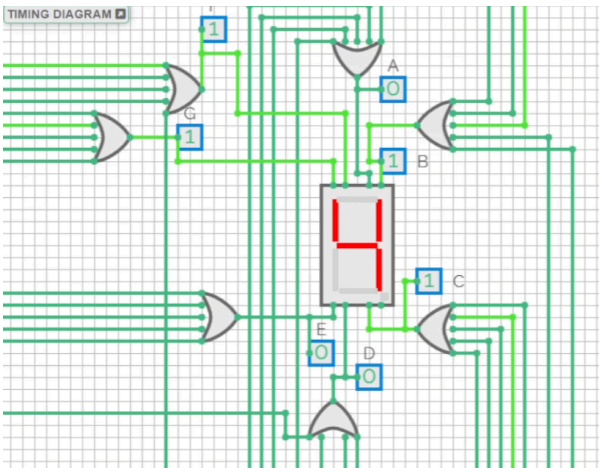
1



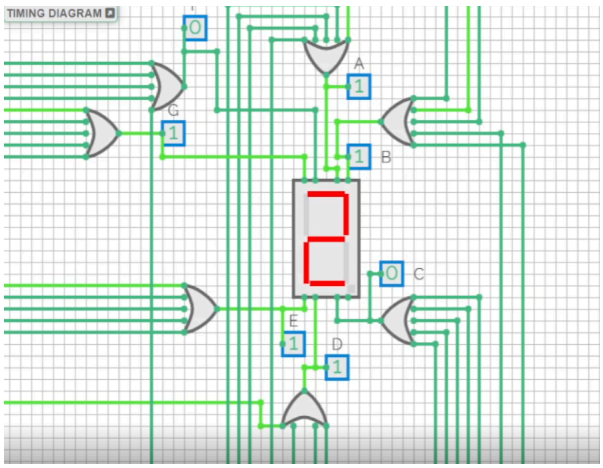
2



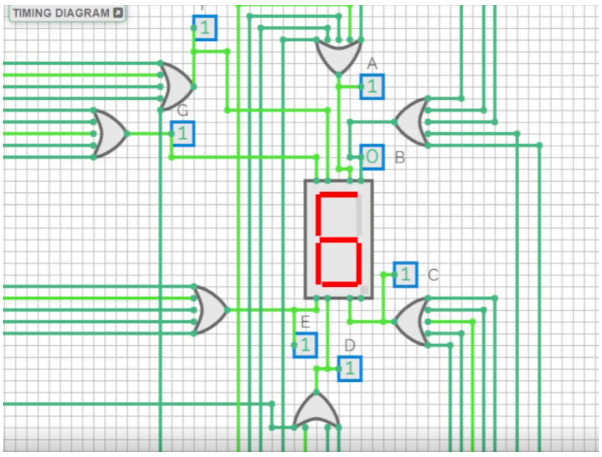
2


TIMING DIAGRAM
G
A
B
C
E
D

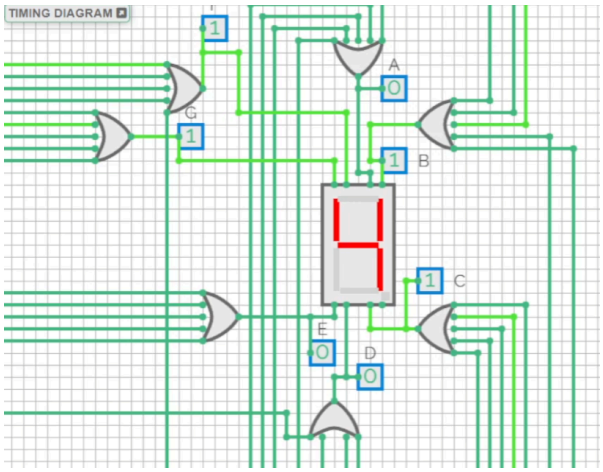
4


TIMING DIAGRAM
G
A
B
C
E
D

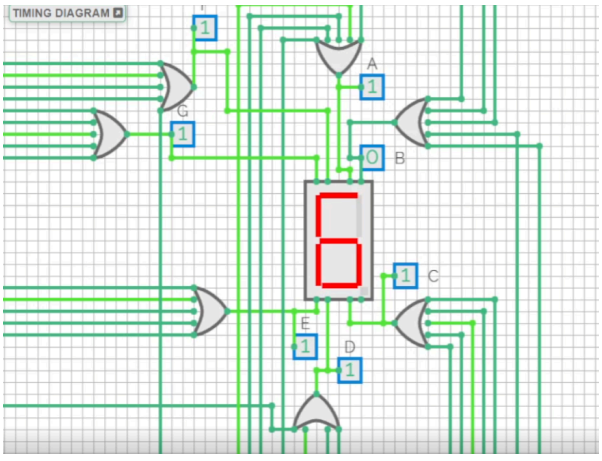
2


TIMING DIAGRAM
G
A
B
C
E
D

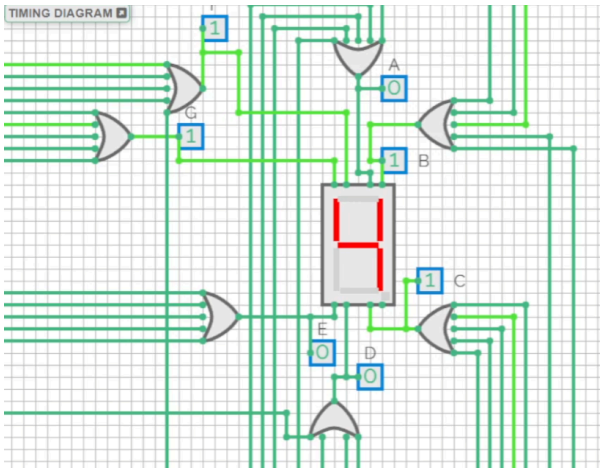
6


TIMING DIAGRAM
G
A
B
C
E
D

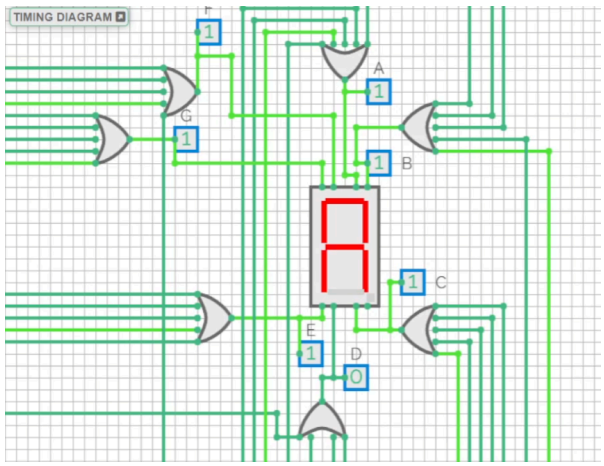
4


TIMING DIAGRAM
G
A
B
C
E
D

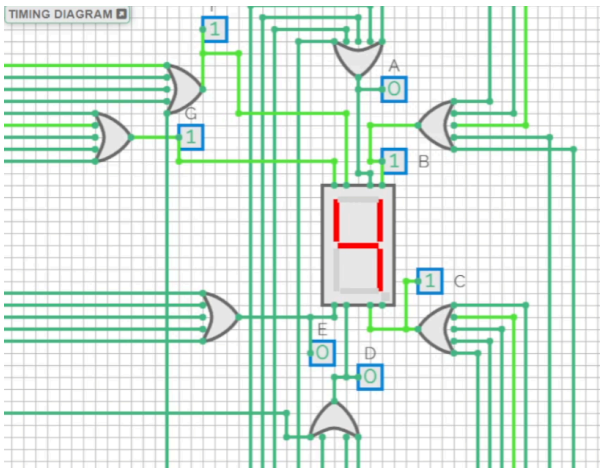
6


TIMING DIAGRAM
G
A
B
C
E
D

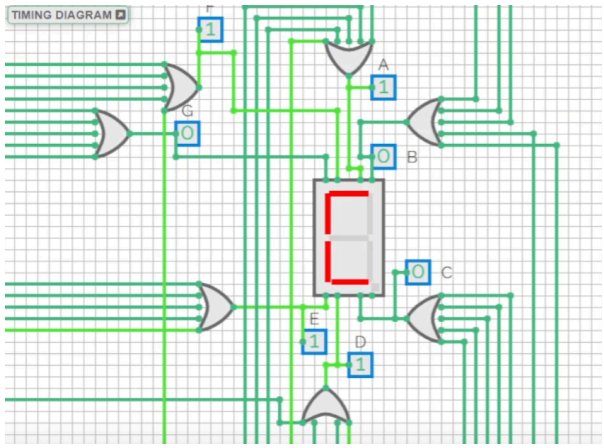
4


TIMING DIAGRAM
F
1
A
1
G
1
B
1
C
1
E
1
D
0

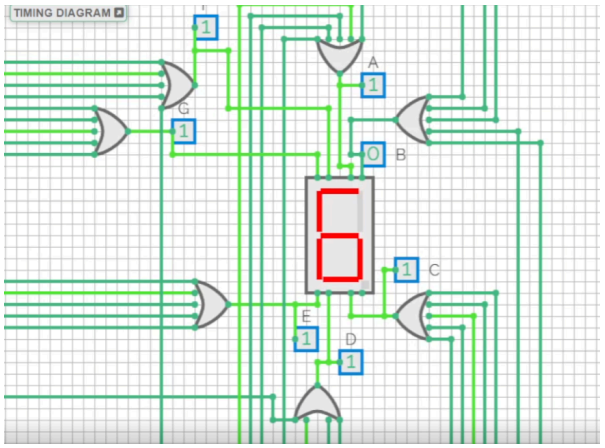
A (10)


TIMING DIAGRAM
F
1
A
0
G
1
B
1
C
1
E
0
D
0

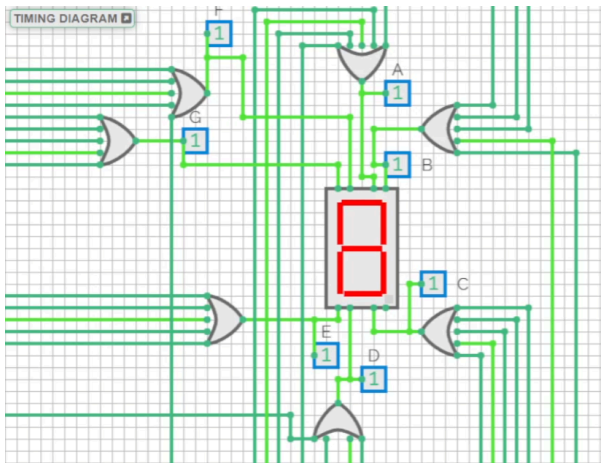
4


TIMING DIAGRAM
F
1
A
1
G
0
B
0
C
0
E
1
D
1

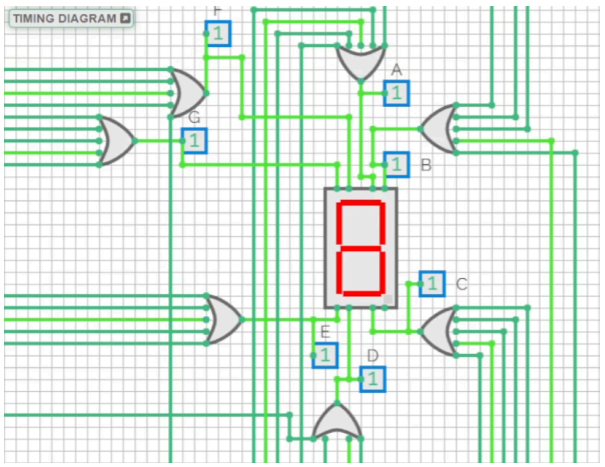
C (12)


TIMING DIAGRAM
F
1
A
1
G
1
B
0
C
1
E
1
D
1

6


TIMING DIAGRAM
F
1
A
1
G
1
B
1
C
1
E
1
D
1

8


TIMING DIAGRAM
F
1
A
1
G
1
B
1
C
1
E
1
D
1

8

==========

5. **Verilog Code**

   This last part of the documentation will just provide code for the followings files:

   a. Homssi_S14_DP.v (Verilog Code from Circuit Verse)

   b. Homssi_S14_DP_tb.v (Testbench File)

   Will also provide a guide on how to run code.

HOMSSI_S14_DP.v:

```
/*
  Element Usage Report
    Clock - 1 times
    DflipFlop - 4 times
    AndGate - 10 times
    OrGate - 17 times
    Splitter - 1 times
    Decoder - 1 times
    Output - 7 times
    Input - 1 times
    NotGate - 1 times
*/

/*
  Usage Instructions and Tips
    Labels - Ensure unique label names and avoid using verilog keywords
    Warnings - Connect all optional inputs to remove warnings
    Clock - Use a single global clock

    Instructions on how to run code:
    - Open:   Command Prompt
    - Type:   iverilog -o HOMSSI_S14_DP.vvp HOMSSI_S14_DP.v HOMSSI_S14_DP_tb.v
    - Type:   vvp HOMSSI_S14_DP.vvp

      You should see results now, thank you sir, pls give me full grades
*/


module EulerTotient(R,CLK_LABEL,A,B,C,D,E,F,G);
  output A,  B,  C,  D,  E,  F,  G;
  input R, CLK_LABEL;
  wire DA_Q, DA_Q_inv, and_0_out, or_0_out, and_3_out, and_2_out, and_1_out,
Decoder_0_out_0, Decoder_0_out_1, Decoder_0_out_2, Decoder_0_out_3, Decoder_0_out_4,
Decoder_0_out_5, Decoder_0_out_6, Decoder_0_out_7, Decoder_0_out_8, Decoder_0_out_9,
Decoder_0_out_10, Decoder_0_out_11, Decoder_0_out_12, Decoder_0_out_13, Decoder_0_out_14,
Decoder_0_out_15, \8_GATE_out , or_15_out, or_14_out, or_16_out, or_12_out, or_6_out,
or_11_out, or_13_out, \6_GATE_out , C_GATE_out, \4_GATE_out , A_GATE_out, \2_GATE_out ,
\1_GATE_out , DB_Q, DB_Q_inv, and_6_out, or_1_out, and_5_out, and_4_out, DC_Q, DC_Q_inv,
and_7_out, or_2_out, and_8_out, DD_Q, DD_Q_inv, and_9_out, not_0_out;
  wire [3:0] Splitter_0_cmb;
  DflipFlop DA(DA_Q, DA_Q_inv, CLK_LABEL, or_0_out, , , );
  assign and_0_out = DA_Q_inv & not_0_out & DB_Q & DC_Q & DD_Q;
  assign or_0_out = and_0_out | and_1_out | and_2_out | and_3_out;
```

```
  assign and_3_out = not_0_out & DA_Q & DD_Q_inv;
  assign and_2_out = not_0_out & DA_Q & DC_Q_inv;
  assign and_1_out = not_0_out & DA_Q & DB_Q_inv;
  assign Splitter_0_cmb = {DD_Q,DC_Q,DB_Q,DA_Q};
  Decoder16 #(4) Decoder_0(Decoder_0_out_0, Decoder_0_out_1, Decoder_0_out_2,
Decoder_0_out_3, Decoder_0_out_4, Decoder_0_out_5, Decoder_0_out_6, Decoder_0_out_7,
Decoder_0_out_8, Decoder_0_out_9, Decoder_0_out_10, Decoder_0_out_11, Decoder_0_out_12,
Decoder_0_out_13, Decoder_0_out_14, Decoder_0_out_15, Splitter_0_cmb);
  assign \8_GATE_out  = Decoder_0_out_14 | Decoder_0_out_15;
  assign or_15_out = \4_GATE_out  | \6_GATE_out  | \8_GATE_out  | A_GATE_out | C_GATE_out;
  assign F = or_15_out;
  assign or_14_out = \2_GATE_out  | \4_GATE_out  | \6_GATE_out  | \8_GATE_out  | A_GATE_out;
  assign G = or_14_out;
  assign or_16_out = C_GATE_out | A_GATE_out | \8_GATE_out  | \6_GATE_out  | \2_GATE_out ;
  assign A = or_16_out;
  assign or_12_out = \2_GATE_out  | \6_GATE_out  | \8_GATE_out  | C_GATE_out;
  assign D = or_12_out;
  assign or_6_out = \1_GATE_out  | \2_GATE_out  | \4_GATE_out  | \8_GATE_out  | A_GATE_out;
  assign B = or_6_out;
  assign or_11_out = \1_GATE_out  | \4_GATE_out  | \6_GATE_out  | \8_GATE_out  | A_GATE_out;
  assign C = or_11_out;
  assign or_13_out = \2_GATE_out  | \6_GATE_out  | \8_GATE_out  | A_GATE_out | C_GATE_out;
  assign E = or_13_out;
  assign \6_GATE_out  = Decoder_0_out_6 | Decoder_0_out_8 | Decoder_0_out_13;
  assign C_GATE_out = Decoder_0_out_12;
  assign \4_GATE_out  = Decoder_0_out_4 | Decoder_0_out_7 | Decoder_0_out_9 |
Decoder_0_out_11;
  assign A_GATE_out = Decoder_0_out_10;
  assign \2_GATE_out  = Decoder_0_out_2 | Decoder_0_out_3 | Decoder_0_out_5;
  assign \1_GATE_out  = Decoder_0_out_0 | Decoder_0_out_1;
  DflipFlop DB(DB_Q, DB_Q_inv, CLK_LABEL, or_1_out, , , );
  assign and_6_out = DB_Q_inv & not_0_out & DC_Q & DD_Q;
  assign or_1_out = and_6_out | and_4_out | and_5_out;
  assign and_5_out = not_0_out & DB_Q & DD_Q_inv;
  assign and_4_out = not_0_out & DB_Q & DC_Q_inv;
  DflipFlop DC(DC_Q, DC_Q_inv, CLK_LABEL, or_2_out, , , );
  assign and_7_out = not_0_out & DC_Q_inv & DD_Q;
  assign or_2_out = and_7_out | and_8_out;
  assign and_8_out = not_0_out & DC_Q & DD_Q_inv;
  DflipFlop DD(DD_Q, DD_Q_inv, CLK_LABEL, and_9_out, , , );
  assign and_9_out = not_0_out & DD_Q_inv;
  assign not_0_out = ~R;
endmodule

module DflipFlop(q, q_inv, clk, d, a_rst, pre, en);
    parameter WIDTH = 1;
    output reg [WIDTH-1:0] q, q_inv;
    input clk, a_rst, pre, en;
    input [WIDTH-1:0] d;

    always @ (posedge clk or posedge a_rst)
    if (a_rst) begin
```

```
        q <= 'b0;
        q_inv <= 'b1;
    end else if (en == 0) ;
    else begin
        q <= d;
        q_inv <= ~d;
    end
endmodule

module Decoder16(out0, out1, out2, out3, out4, out5, out6, out7, out8, out9, out10, out11,
out12, out13, out14, out15, sel);
  output reg out0, out1, out2, out3, out4, out5, out6, out7, out8, out9, out10, out11,
out12, out13, out14, out15;
  input [3:0] sel;

  always @ (*) begin
    out0 = 0;
    out1 = 0;
    out2 = 0;
    out3 = 0;
    out4 = 0;
    out5 = 0;
    out6 = 0;
    out7 = 0;
    out8 = 0;
    out9 = 0;
    out10 = 0;
    out11 = 0;
    out12 = 0;
    out13 = 0;
    out14 = 0;
    out15 = 0;
    case (sel)
      0 : out0 = 1;
      1 : out1 = 1;
      2 : out2 = 1;
      3 : out3 = 1;
      4 : out4 = 1;
      5 : out5 = 1;
      6 : out6 = 1;
      7 : out7 = 1;
      8 : out8 = 1;
      9 : out9 = 1;
      10 : out10 = 1;
      11 : out11 = 1;
      12 : out12 = 1;
      13 : out13 = 1;
      14 : out14 = 1;
      15 : out15 = 1;
    endcase
  end
endmodule
```

HOMSSI_S14_DP_tb.v:

```
`timescale 1ns / 1ps

module TestBench();
  reg R;
  reg clk_0;
  wire A, B, C, D, E, F, G;

EulerTotient DUT0(R, clk_0, A, B, C, D, E, F, G);

  always begin
    clk_0 = 0;
    forever #100 clk_0 = ~clk_0;
  end

  always begin
    clk_0 = 0;
    forever #1000 clk_0 = ~clk_0;
  end

  always @(posedge clk_0) begin
         R=0;
         #2000
         R=1;
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 1
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 1
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 1
         R=0;
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 1
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 2
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 2
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 4
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 2
         #2000
         $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
```

```
// 6
          #2000
          $display("R=%b clk_0=%b ABCDEFG = %b%b%b%b%b%b%b", R, clk_0, A, B, C, D, E ,F, G);
// 4
     #2000
          /*add more tests as needed*/

       $finish;
  end
endmodule
```

Tutorial on how to run code:

1. Open Command Prompt, then type the steps 2 and 3:
2. iverilog -o Homssi_S14_DP.vvp Homssi_S14_DP_tb.v
3. vvp Homssi_S14_DP.vvp

output: