VERILOG SIMULATIONS IN EXP-6 AND EXP-7
Gray Code Counter
Synchronous Ring Counter
Sequence Generator

NAME: ABHINAV REDDY ORUGANTI

ENTRY NUMBER:2019EE10455

Gray Code Counter

- A Gray code counter counts in such a manner that the difference between any two consecutive states differs only by 1 bit.
- This is the cycle followed by the Gray code counter. X1X2X3X4 represents the Gray code representation and its decimal equivalent is shown in the brackets
- 0000(0) --> 0001(1) --> 0011(3) --> 0010(2) --> 0110(6) --> 0111(7) --> 0101(5) --> 0100(4) --> 1100(12) --> 1101(13) --> 1111(15) --> 1110(14) --> 1010(10) --> 1011(11) --> 1001(9) --> 1000(8)
- We use SR Flip Flops to implement this counter. The number of flip flops required are 4 since there are 4 bits to represent and all 16 states are being used.
- To implement this counter we first draw the state table and assign SR values to each flip flop with the help of Karnaugh Maps to achieve minimized expression for the inputs of flip flops.
- In the next page we have the state table which shows the present state, next state and flip flop inputs, followed by Karnaugh Maps in the next pages and then Verilog code and simulation

PRESENT STATE				NEXT STATE				FLIP FLOP INPUTS							
Q3	**Q2**	**Q1**	**Q0**	**Q3+**	**Q2+**	**Q1+**	**Q0+**	**S3**	**R3**	**S2**	**R2**	**S1**	**R1**	**S0**	**R0**
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	0
0	0	0	1	0	0	1	1	0	x	0	x	1	0	x	0
0	0	1	1	0	0	1	0	0	x	0	x	x	0	0	1
0	0	1	0	0	1	1	0	0	x	1	0	x	0	0	x
0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	0
0	1	1	1	0	1	0	1	0	x	x	0	0	1	x	0
0	1	0	1	0	1	0	0	0	x	x	0	0	x	0	1
0	1	0	0	1	1	0	0	1	0	x	0	0	x	0	x
1	1	0	0	1	1	0	1	x	0	x	0	0	x	1	0
1	1	0	1	1	1	1	1	x	0	x	0	1	0	x	0
1	1	1	1	1	1	1	0	x	0	x	0	x	0	0	1
1	1	1	0	1	0	1	0	x	0	0	1	x	0	0	x
1	0	1	0	1	0	1	1	x	0	0	x	x	0	1	0
1	0	1	1	1	0	0	1	x	0	0	X	0	1	x	0
1	0	0	1	1	0	0	0	x	0	0	x	0	x	0	1
1	0	0	0	0	0	0	0	0	1	0	x	0	x	0	x

Q3Q2/Q1Q0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	x	x	x	x
10	0	x	x	x

K-Map for S3

S3 = Q2Q1’Q0’

Q3Q2/Q1Q0	00	01	11	10
00	x	x	x	x
01	0	x	x	x
11	0	0	0	0
10	1	0	0	0

K- Map for R3

R3 = Q2’Q1’Q0’

Q3Q2/Q1Q0	00	01	11	10
00	0	0	0	1
01	x	x	x	x
11	x	x	x	x
10	0	0	0	0

K-Map for S2

S2 = Q3’Q1Q0’

Q3Q2/Q1Q0	00	01	11	10
00	x	x	x	0
01	0	0	0	0
11	0	0	0	1
10	x	x	x	x

K-Map for R2

R2 = Q3Q1Q0’

Q3Q2/Q1Q0	00	01	11	10
00	0	1	x	x
01	0	0	0	x
11	0	1	x	x
10	0	0	0	x

K-Map for S1

S1 = Q3’Q2’Q0 + Q3Q2Q0

Q3Q2/Q1Q0	00	01	11	10
00	x	0	0	0
01	x	x	1	0
11	x	0	0	0
10	x	x	1	0

K-Map for R1

R1 = Q3’Q2Q0 + Q3Q2’Q0

Q3Q2/Q1Q0	00	01	11	10
00	1	x	0	0
01	0	0	x	1
11	1	x	0	0
10	0	0	x	1

K-Map for S0

S0 = Q3’Q2’Q1’ + Q3’Q2Q1 + Q3Q2’Q1+Q3Q2Q1’

Q3Q2/Q1Q0	00	01	11	10
00	0	0	1	x
01	x	1	0	0
11	0	0	0	x
10	x	1	0	0

K-Map for R0

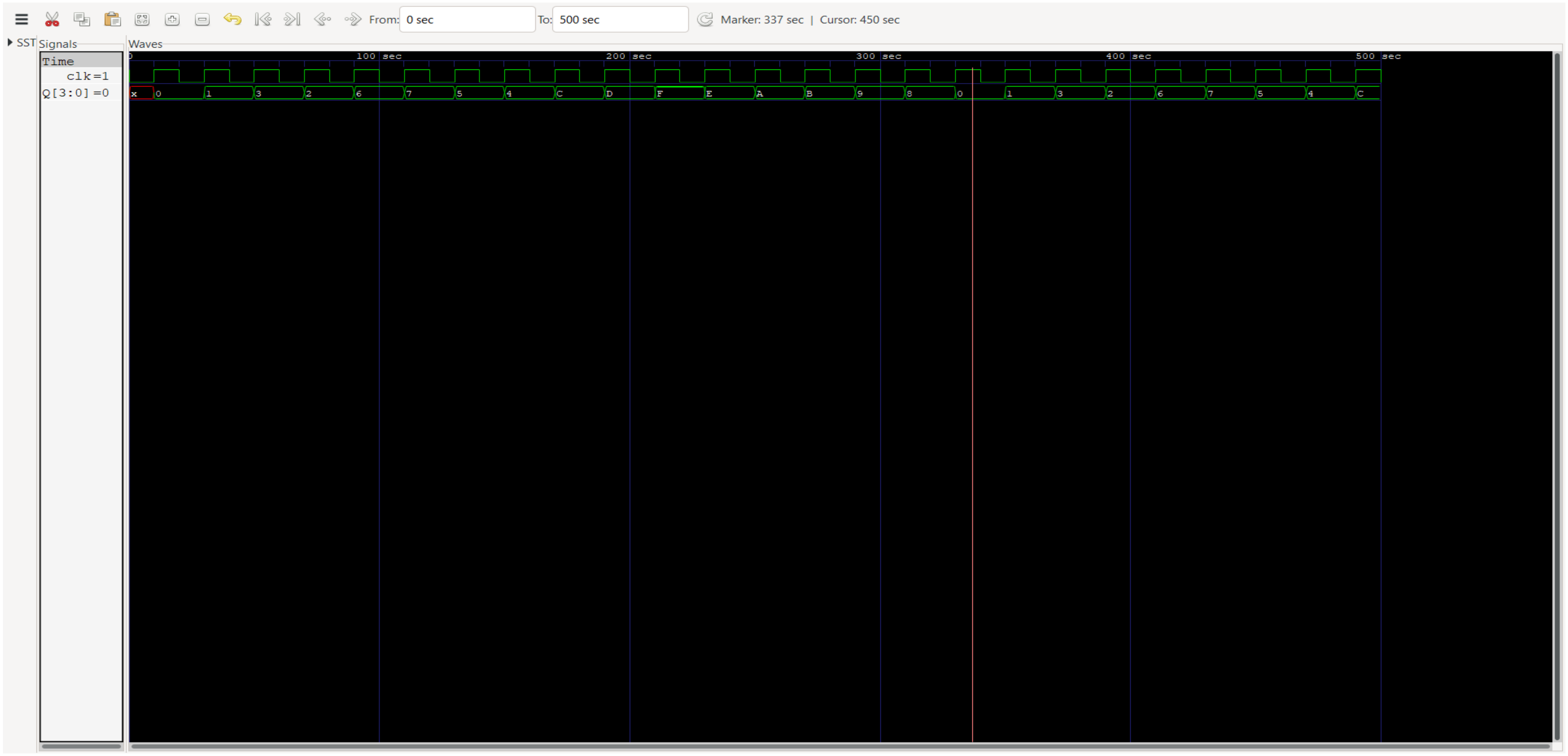
R0 = Q3’Q2’Q1 + Q3’Q2Q1’ + Q3Q2’Q1’ + Q3Q2Q1

Verilog Code for Gray Code Counter

```
module gray_code_counter (
   input clk,
   output [3:0] Q
);
   wire S3,S2,S1,S0,R3,R2,R1,R0,Qb3,Qb2,Qb1,Qb0;
   assign S3 = (Q[2] && Qb1 && Qb0);
   assign R3 = (Qb2 && Qb1 && Qb0);
   assign S2 = (Qb3 && Q[1] && Qb0);
   assign R2 = (Q[3] && Q[1] && Qb0);
   assign S1 = (Qb3 && Qb2 && Q[0]) || (Q[3] && Q[2] && Q[0]);
   assign R1 = (Qb3 && Q[2] && Q[0]) || (Q[3] && Qb2 && Q[0]);
   assign S0 = (Qb3 && Qb2 && Qb1) || (Qb3 && Q[2] && Q[1]) || (Q[3] &&
Qb2 && Q[1]) || (Q[3] && Q[2] && Qb1);
   assign R0 = (Qb3 && Qb2 && Q[1]) || (Qb3 && Q[2] && Qb1) || (Q[3] &&
Qb2 && Qb1) || (Q[3] && Q[2] && Q[1]);
   SR_FF ff3(.clk(clk), .sr({S3,R3}), .Q(Q[3]), .Qb(Qb3));
   SR_FF ff2(.clk(clk), .sr({S2,R2}), .Q(Q[2]), .Qb(Qb2));
   SR_FF ff1(.clk(clk), .sr({S1,R1}), .Q(Q[1]), .Qb(Qb1));
   SR_FF ff0(.clk(clk), .sr({S0,R0}), .Q(Q[0]), .Qb(Qb0));
endmodule
```

```
module gray_code_counter_tb;
   wire [3:0] Q; reg clk;
   gray_code_counter M_UUT(.clk(clk), .Q(Q));
   initial #500 $finish;
   initial begin
      $dumpfile("grayCode.vcd"); $dumpvars(0,gray_code_counter_tb);
      clk = 0;
     forever begin
         #10 clk = ~clk;
      end
   end
Endmodule

module SR_FF (
   input [1:0] sr,
   input clk,  output Q,Qb);
   reg Q,Qb;
   always @(posedge clk) begin
      case (sr)
         2'b00: Q=Q; 2'b01: Q=1’b0;  2'b10: Q=1'b1; 2'b11: Q=1'bz; default: Q=1'b0;       endcase
    Qb=~Q;
   end
endmodule
```

Verilog Simulation



Synchronous Ring Counter using D Flip Flops

- My entry number is 2019EE10545. Hence the ring counter should start from 0101 (last digit of entry number is 5)
- The number of flip flops required are 4 since there are 4 bits to represent.
- The counter covers 15 states irrespective of the state it starts from. The one state which is not being used is 0.
- This counter is a pseudo random sequence generator. It generates random numbers. Its not truly random because the sequence depends on the initial value.
- To design this we first draw the state table from the given table and then we draw Karnaugh Maps to determine the flip flop input equations
- We then write the Verilog code and show its simulation.

PRESENT STATE				FLIP FLOP INPUTS				NEXT STATE			
Q3	**Q2**	**Q1**	**Q0**	**D3**	**D2**	**D1**	**D0**	**Q3+**	**Q2+**	**Q1+**	**Q0+**
0	1	0	1	1	0	1	0	1	0	1	0
1	0	1	0	1	1	0	1	1	1	0	1
1	1	0	1	1	1	1	0	1	1	1	0
1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	0	1	0	0	0	1
0	0	0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0
0	0	1	0	1	0	0	1	1	0	0	1
1	0	0	1	1	1	0	0	1	1	0	0
1	1	0	0	0	1	1	0	0	1	1	0
0	1	1	0	1	0	1	1	1	0	1	1
1	0	1	1	0	1	0	1	0	1	0	1

Q3Q2/Q1Q0	00	01	11	10
00	x	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

K-Map for D3

D3 = Q1’Q0 + Q1Q0’

Q3Q2/Q1Q0	00	01	11	10
00	x	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

K-Map for D2

D2 = Q3

Q3Q2/Q1Q0	00	01	11	10
00	x	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

K-Map for D1

D1 = Q2

Q3Q2/Q1Q0	00	01	11	10
00	x	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

K-Map for D0

D0 = Q1

Verilog Code for Synchronous Ring Counter

```
module sync_ring_counter (
   input clk,
   output [3:0] Q
);
   wire D3,D2,D1,D0,Qb3,Qb2,Qb1,Qb0;
   assign D3 = (Qb1 && Q[0]) || (Q[1] && Qb0);
   assign D2 = Q[3];
   assign D1 = Q[2];
   assign D0 = Q[1];

   DFF2 ff3(.clk(clk), .D(D3), .Q(Q[3]), .Qb(Qb3));
   DFF1 ff2(.clk(clk), .D(D2), .Q(Q[2]), .Qb(Qb2));
   DFF2 ff1(.clk(clk), .D(D1), .Q(Q[1]), .Qb(Qb1));
   DFF1 ff0(.clk(clk), .D(D0), .Q(Q[0]), .Qb(Qb0));

endmodule
```

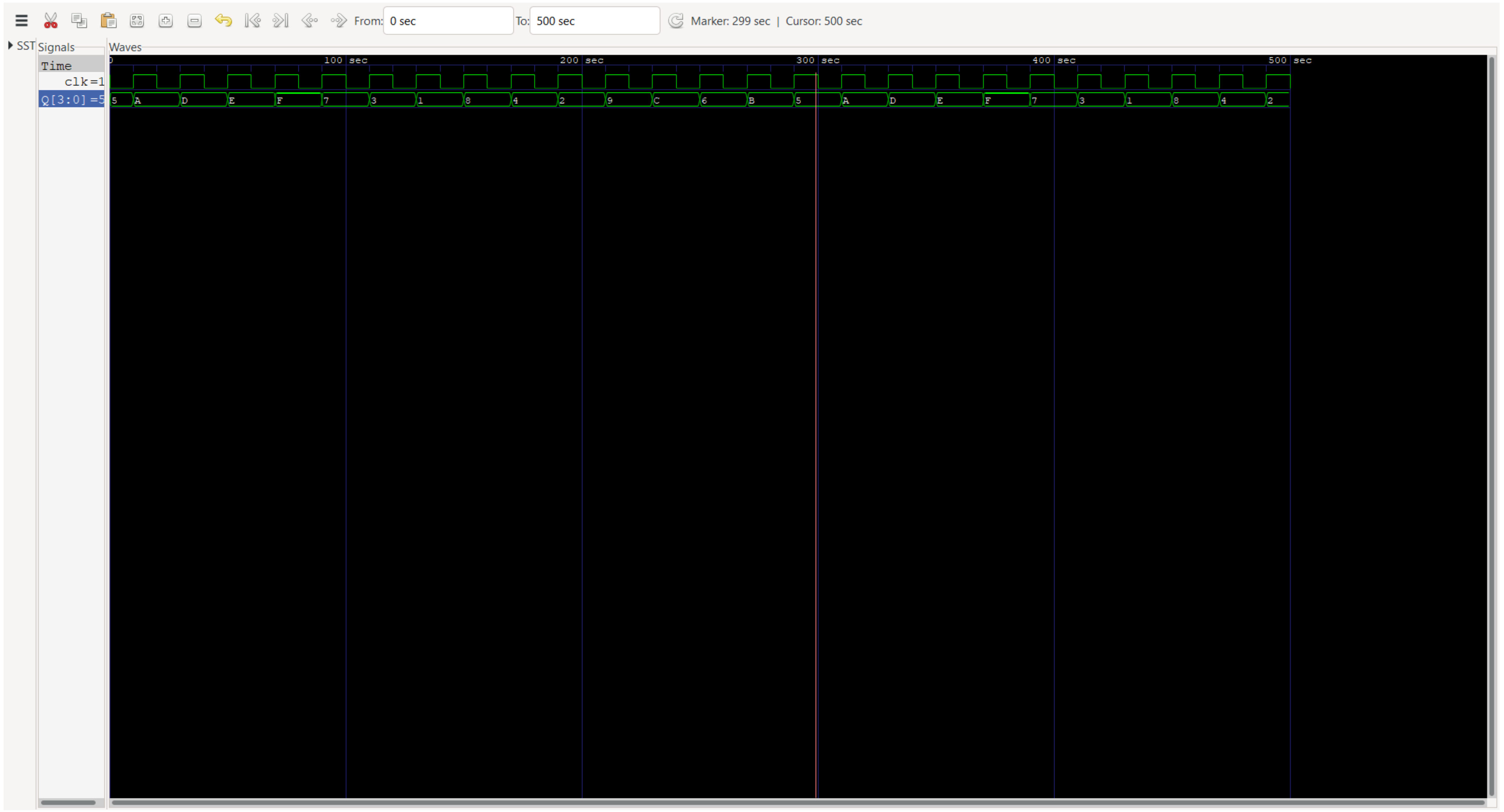
```
module DFF2 (
   input D,clk,
   output Q, Qb
);
   initial begin
      Q = 1'b0;
      Qb = 1'b1;
   end
   reg Q, Qb;
   always @(posedge clk) begin
      Q=D;
      Qb=~Q;
   end
endmodule
```

Verilog Code(Continued…..)

```
module DFF1 (
   input D,clk,
   output Q, Qb
);
   initial begin
      Q = 1'b1;
      Qb = 1'b0;
   end
   reg Q, Qb;
   always @(posedge clk) begin
      Q=D;
      Qb=~Q;
   end
endmodule
```

```
module sync_ring_counter_tb;
   wire [3:0] Q;
   reg clk;
   sync_ring_counter M_UUT(.clk(clk), .Q(Q));

   initial #500 $finish;
   initial begin
      $dumpfile("syncRing.vcd");
      $dumpvars(0,sync_ring_counter_tb);
      clk = 0;
      forever begin
         #10 clk = ~clk;
      end
   end

endmodule
```

Verilog Simulation


From: 0 sec
To: 500 sec
Marker: 299 sec | Cursor: 500 sec

Sequence Generator

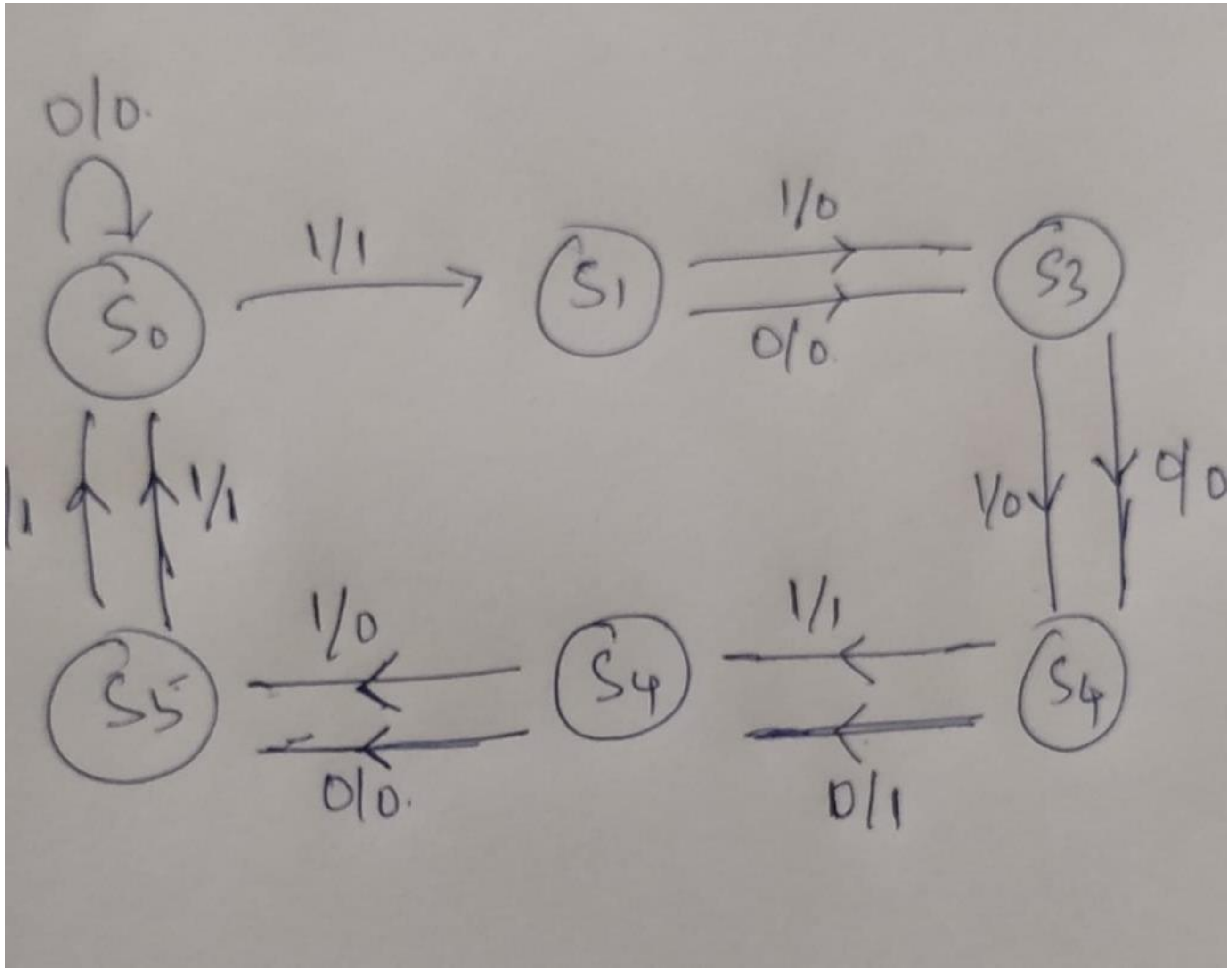
- My entry number is 2019EE10545. X3%8 = 100, X4%8 = 101.Hence the sequence to be generated from the finite state machine made of D Flip Flops is 100101.
- To do this, we first draw the state diagram from the given specifications and then obtain state table. No. of Flip flops required are 3, since there are 6 states in the state diagram.
- Let the 6 states be 000,001,010,011,100,101.
- Then we draw Karnaugh maps to find the inputs of flip flop.
- Then we simulate the FSM using Verilog.

State Diagram

The sequence I have to generate is 100101. There are total 6 states .We assign states as follows:

- S0 --> 000
- S1 --> 001
- S2 --> 010
- S3 --> 011
- S4 --> 100
- S5 --> 101

Here S0 is the idle state.



PRESENT STATE			i/p	NEXT STATE			FLIP FLOP INPUTS			o/p
Q2	**Q1**	**Q0**	**x**	**Q2+**	**Q1+**	**Q0+**	**D2**	**D1**	**D0**	**y**
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1	1
0	0	1	0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	0	1	0	0
0	1	0	0	0	1	1	0	1	1	0
0	1	0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	0	1	0	0	1
0	1	1	1	1	0	0	1	0	0	1
1	0	0	0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	1	0	1	0
1	0	1	0	0	0	0	0	0	0	1
1	0	1	1	0	0	0	0	0	0	1

STATE TABLE

Q3Q2/Q1Q0	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	x	x	x	x
10	1	1	0	0

K-Map for D2

D2 = Q1Q0+Q2Q0’

Q3Q2/Q1Q0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	x	x	x	x
10	0	0	0	0

K-Map for D1

D2 = Q1Q0’+Q2’Q1’Q0

Q3Q2/Q1Q0	00	01	11	10
00	0	1	0	0
01	1	1	0	0
11	x	x	x	x
10	1	1	0	0

K-Map for D1

D1 = Q0’(Q1+Q2+x)

Q3Q2/Q1Q0	00	01	11	10
00	0	1	0	0
01	0	0	1	1
11	x	x	x	x
10	0	0	1	1

K-Map for y

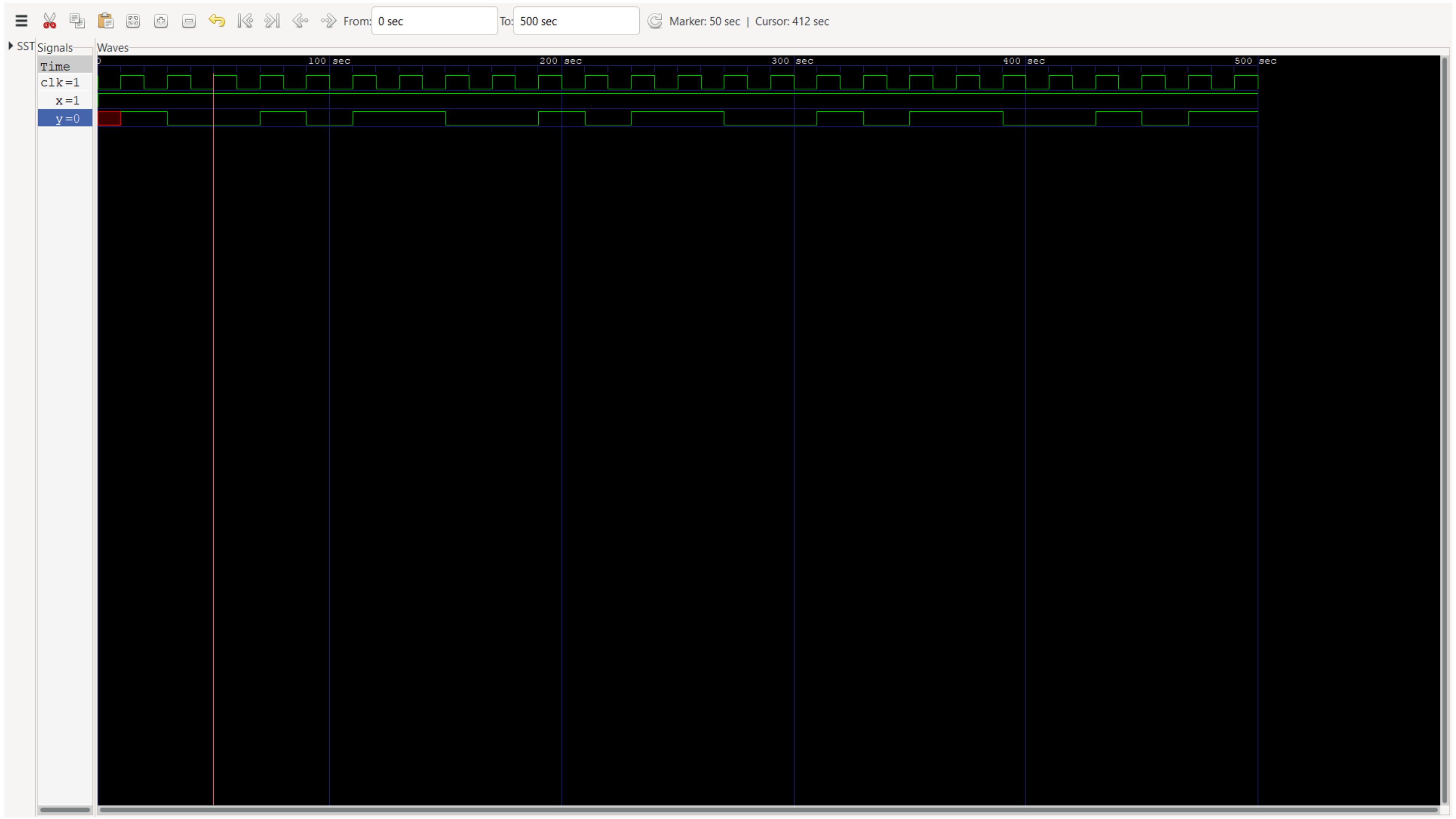
y = Q1Q0+Q2Q0+Q2’Q1’Q0’x

Verilog Code for Sequence Generator

```
module FSM (
   input x,clk,
   output [3:0] Q,
   output y

);
wire D2,D1,D0,Qb2,Qb1,Qb0,y;
assign D2 = (Q[1] && Q[0]) || (Q[2] && Qb0);
assign D1 = (Q[1] && Qb0) || (Qb2 && Qb1 && Q[0]);
assign D0 = (Qb0 && (Q[2] || Q[1] || x));
assign y = (Q[1] && Q[0]) || (Q[2] && Q[0])  || (Qb2 && Qb1 &&
Qb0 && x);

DFF ff2(.D(D2), .clk(clk), .Q(Q[2]), .Qb(Qb2));
DFF ff1(.D(D1), .clk(clk), .Q(Q[1]), .Qb(Qb1));
DFF ff0(.D(D0), .clk(clk), .Q(Q[0]), .Qb(Qb0));

endmodule
```

```
module FSM_tb;
   reg clk, x;
   wire y;
   initial #500 $finish;
   FSM M_UUT(.clk(clk), .x(x), .y(y));
   initial begin
      $dumpfile("FSM.vcd");
      $dumpvars(0,FSM_tb);
      clk = 0;
      forever begin
         #10 clk = ~clk;
      end
   end
   initial begin
      x=1;
   end
endmodule
```

Verilog Simulation


From: 0 sec
To: 500 sec
Marker: 50 sec | Cursor: 412 sec
SST
Signals
Waves
Time
clk=1
x=1
y=0
0
100 sec
200 sec
300 sec
400 sec
500 sec