ELL201 Experiment 6
Lab Report

Ritvik Gupta
2019MT10512

1 Synchronous 4-bit Gray-Code Counter

1.1 State Table

The State table for the counter is as follows. Q_i^n represents the value of the i^{th} bit of the counter at the n^{th} state.

Q_3^n	Q_2^n	Q_1^n	Q_0^n	Q_3^{n+1}	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	1	1	0
0	0	1	1	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	0	1	1	1	0	0	1
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	0
1	1	1	1	1	1	1	0

Table 1: State Table for Synchronous 4-bit Gray-Code Counter

1.2 Number of Flip-flops required to make the counter

We are implementing a 4-bit Gray-Code Counter. For representing each bit in this counter, we would require an SR Flip-flop. Thus, the total number of SR Flip-flops required to make the 4 bit Gray-Code counter is **4**. We will use S_i, R_i to control the functioning of the i^{th} Flip-flop, which represents the i^{th} binary digit of the counter.

1.3 Assignment of values to S, R for each SR Flip-Flop

We assign the i^{th} SR Flip-flop to the i^{th} binary digit of the counter. Using the State table above, we assign values to the inputs S_i and R_i.
We use the transition table for an SR Flip-flop for the same.

Q_3^n	Q_2^n	Q_1^n	Q_0^n	Q_3^{n+1}	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}	S_3	R_3	S_2	R_2	S_1	R_1	S_0	R_0
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	0	0	1	1	0	0	X	1	0	X	0	0	X
0	0	1	1	0	0	1	0	0	X	0	X	X	0	0	1
0	1	0	0	1	1	0	0	1	0	X	0	0	X	0	X
0	1	0	1	0	1	0	0	0	X	X	0	0	X	0	1
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
1	0	0	0	0	0	0	0	0	1	0	X	0	X	0	X
1	0	0	1	1	0	0	0	X	0	0	X	0	X	0	1
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	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	0	1	0	1	0	X	0	0	1	X	0	0	X
1	1	1	1	1	1	1	0	X	0	X	0	X	0	0	1

Table 2: Assignment of values to S, R for each SR Flip-Flop

1.4 Using Karnaugh Maps to minimise the expressions for the inputs

Using the assigned values of S_i and R_i, we draw the Karnaugh Maps of S_3, R_3, S_2, R_2, S_1, R_1, S_0 and R_0. We use these Karnaugh Maps to achieve the minimised expressions for each of the inputs S_i and R_i.

The obtained minimised expressions are as follows:

$$S_3 = Q_2Q_1'Q_0'$$

$$R_3 = Q_2'Q_1'Q_0'$$

$$S_2 = Q_3'Q_1Q_0'$$

$$R_2 = Q_3Q_1Q_0'$$

$$S_1 = Q_3Q_2Q_0 + Q_3'Q_2'Q_0$$

$$R_1 = Q_3Q_2'Q_0 + Q_3'Q_2Q_0$$

$$S_0 = Q_3'Q_2Q_1 + Q_3'Q_2'Q_1' + Q_3Q_2'Q_1 + Q_3Q_2Q_1'$$

$$R_0 = Q_3'Q_2'Q_1 + Q_3'Q_2Q_1' + Q_3Q_2Q_1 + Q_3Q_2'Q_1'$$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	X	X	X	X
10	0	X	X	X

(a) Karnaugh Map for S_3

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	X	X	X	X
01	0	X	X	X
11	0	0	0	0
10	1	0	0	0

(b) Karnaugh Map for R_3

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	0	0	0	1
01	X	X	X	X
11	X	X	X	0
10	0	0	0	0

(c) Karnaugh Map for S_2

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	X	X	X	0
01	0	0	0	0
11	0	0	0	1
10	X	X	X	X

(d) Karnaugh Map for R_2

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	0	1	X	X
01	0	0	0	X
11	0	1	X	X
10	0	0	0	X

(e) Karnaugh Map for S_1

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	X	0	0	0
01	X	X	1	0
11	X	0	0	0
10	X	X	1	0

(f) K-Map for R_1

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	1	X	0	0
01	0	0	X	1
11	1	X	0	0
10	0	0	X	1

(g) Karnaugh Map for S_0

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
00	0	0	1	X
01	X	1	0	0
11	0	0	1	X
10	X	1	0	0

(h) Karnaugh Map for R_0

Figure 1: Karnaugh Maps for the inputs S_i, R_i for the four SR Flip-flops

1.5 Verilog Code

1.5.1 Verilog Code for 4-bit Gray-Code Counter

```
module SRFlipFlop(S, R, clk, reset, Q);
    input S, R, clk, reset;
    output reg Q;

    always @(posedge clk or posedge reset)
        if(reset) Q <= 0;
        else begin
        case ({S,R})
            2'b00 :  Q <= Q;
            2'b01 :  Q <= 0;
            2'b10 :  Q <= 1;
          endcase
        end
endmodule

module GrayCodeCounter(clk, reset, out);
    input clk, reset;
    output [3:0] out;

    wire [3:0] Q;
    wire [3:0] S;
    wire [3:0] R;

    assign S[3] = (Q[2] & ~Q[1] & ~Q[0]);
    assign R[3] =(~Q[2] & ~Q[1] & ~Q[0]);

    assign S[2] = (~Q[3] & Q[1] & ~Q[0]);
    assign R[2] = (Q[3] & Q[1] & ~Q[0]);

    assign S[1] = (~Q[3] & ~Q[2] & Q[0]) | (Q[3] & Q[2] & Q[0]);
    assign R[1] = (~Q[3] & Q[2] & Q[0]) | (Q[3] & ~Q[2] & Q[0]);

    assign S[0] = (~Q[3] & ~Q[2] & ~Q[1]) | (~Q[3] & Q[2] & Q[1]) | (Q[3] & ~Q[2] & Q[1])
    ↪  | (Q[3] & Q[2]& ~Q[1]);
    assign R[0] = (~Q[3] & ~Q[2] & Q[1]) | (~Q[3] & Q[2] & ~Q[1]) | (Q[3] & ~Q[2] &
    ↪  ~Q[1]) | (Q[3] & Q[2] & Q[1]);

    SRFlipFlop sr0 (S[0], R[0], clk, reset, Q[0]);
    SRFlipFlop sr1 (S[1], R[1], clk, reset, Q[1]);
    SRFlipFlop sr2 (S[2], R[2], clk, reset, Q[2]);
    SRFlipFlop sr3 (S[3], R[3], clk, reset, Q[3]);

    assign out=Q;
endmodule
```

1.5.2 Verilog Code for Testbench

```
module tb_graycode;
    reg clk;
    reg reset;
    wire [3:0] out;

    GrayCodeCounter counter(clk, reset, out);

    always #10 clk = ~clk;
    initial
```

```
        begin
            $dumpfile("graycode.vcd");
                        $dumpvars(0, tb_graycode);
            $monitor($time," %b", out);
            reset <= 1;
            clk <= 0;
            repeat (1) @ (posedge clk);
            reset <= 0;
            repeat (17) @ (posedge clk);
            $finish;
        end
endmodule
```

1.5.3 Waveform for 4-bit Gray-Code Counter

The figure given below shows the waveform obtained after running simulations on Icarus-Verilog.
`clk` and `reset` are the waveforms of the Clock and Reset signal.
`out[3:0]` represents the output of the 4-bit Gray-Code Counter.
Here, the D Flip-flops have an asynchronous active high reset.

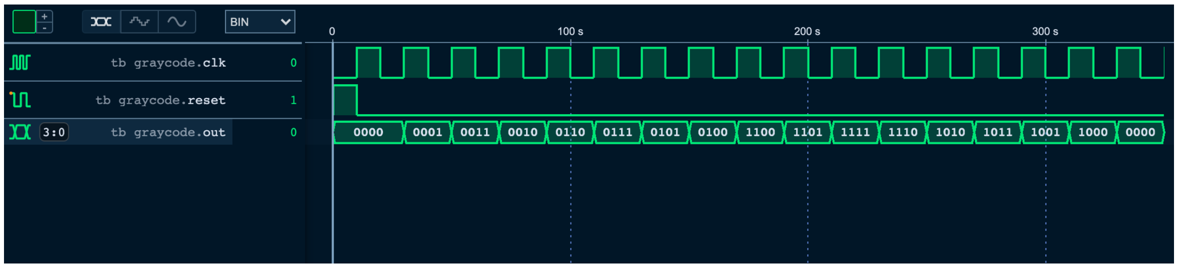
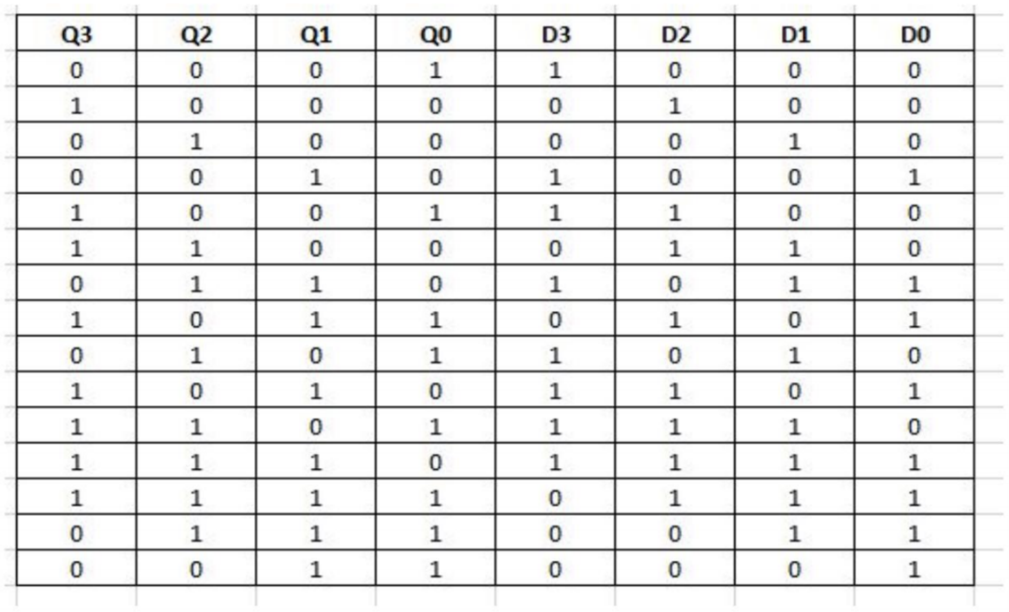


Figure 2: Waveform obtained after running simulation on Icarus-Verilog

2 Synchronous Ring Counter



Q3	Q2	Q1	Q0	D3	D2	D1	D0
0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0
0	0	1	0	1	0	0	1
1	0	0	1	1	1	0	0
1	1	0	0	0	1	1	0
0	1	1	0	1	0	1	1
1	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	1	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	1	1	1
0	1	1	1	0	0	1	1
0	0	1	1	0	0	0	1

Figure 3: State Table for Synchronous Ring Counter

2.1 Number of D Flip-flops required

From the state table of the Ring Counter, we can clearly see that there are 15 states in the counter. Therefore, we will need a minimum of $\lceil \log_2 15 \rceil = \mathbf{4}$ D Flip-flops to implement it.

2.2 Assignment of values to the D inputs and reduction using Karnaugh Maps

The assignment of values to the D inputs is provided in the State table above. Using the state table, we can draw the Karnaugh Maps for the inputs D_i.

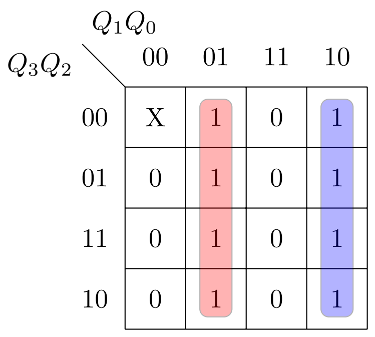
We use these Karnaugh Maps to achieve the minimised expressions for each of the inputs D_i. The obtained minimised expressions are as follows:

$$D_3 = Q_1'Q_0 + Q_1Q_0' = Q_1 \oplus Q_0$$

$$D_2 = Q_3$$

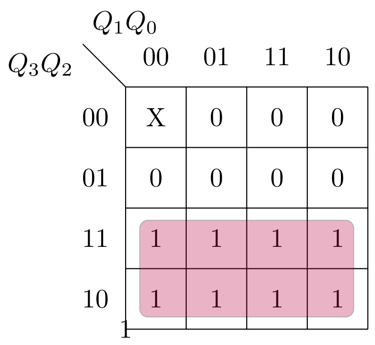
$$D_1 = Q_2$$

$$D_0 = Q_1$$



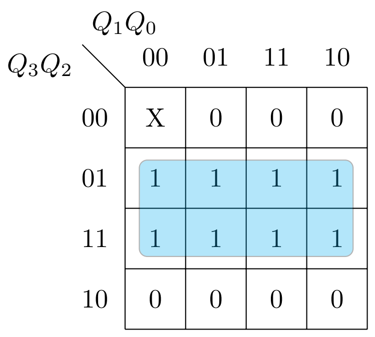
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

(a) Karnaugh Map for D_3



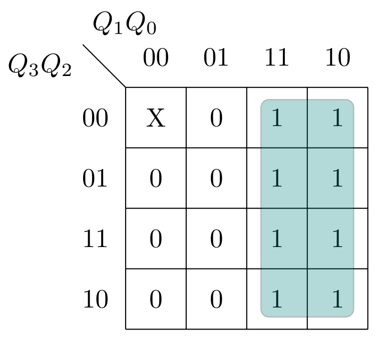
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

(b) Karnaugh Map for D_2



Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

(c) Karnaugh Map for D_1



Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

(d) Karnaugh Map for D_0

Figure 4: Karnaugh Maps for the inputs D_i for the four D Flip-flops

2.3 Verilog Code

2.3.1 Verilog Code for 4-bit Ring Counter

```
module DFlipFlop(D, clk, reset, init, Q);
    input D, clk, reset, init;
    output reg Q;

    always @ (posedge clk or posedge reset)
        if(reset) Q <= init
        else Q <= D;
endmodule

module RingCounter(clk, reset, init, out);
    input clk, reset;
    input [3:0] init;
    output [3:0] out;

    wire [3:0] Q;
    wire [3:0] D;

    assign D[0] = Q[1];
    assign D[1] = Q[2];
    assign D[2] = Q[3];
    assign D[3] = Q[0]^Q[1];

    DFlipFlop d0 (D[0], clk, reset, init[0], Q[0]);
    DFlipFlop d1 (D[1], clk, reset, init[1], Q[1]);
    DFlipFlop d2 (D[2], clk, reset, init[2], Q[2]);
    DFlipFlop d3 (D[3], clk, reset, init[3], Q[3]);

    assign out = Q;
endmodule
```

2.3.2 Verilog Code for Testbench

```
module tb_ringcounter;
    reg clk;
    reg reset;
    wire [3:0] out;
    reg [3:0] init;

    RingCounter counter(clk, reset, init, out);

    always #10 clk = ~clk;
    initial
        begin
            $dumpfile("ringcounter.vcd");
                        $dumpvars(0, tb_ringcounter);
            $monitor($time," %b", out);
            init[3] = 0;
            init[2] = 0;
            init[1] = 1;
            init[0] = 0;
            reset <= 1;
            clk <= 0;
            repeat (1) @ (posedge clk);
            reset <= 0;
            repeat (17) @ (posedge clk);
            $finish;
        end
endmodule
```

2.3.3 Waveform for 4-bit Ring Counter

The figure given below shows the waveform obtained after running simulations on Icarus-Verilog.
`clk` and `reset` are the waveforms of the Clock and Reset signal.
`out[3:0]` represents the output of the 4-bit Ring Counter.
Here, the D Flip-flops have an asynchronous active high reset.

2.4 Observations

The counter cycles in the order 0010→1001→1100→0110→1011→0101→1010→1101→1110→1111→0111→0011 →0001→1000→0100→0010. The counter never reaches the state 0000. Besides this state, the counter covers

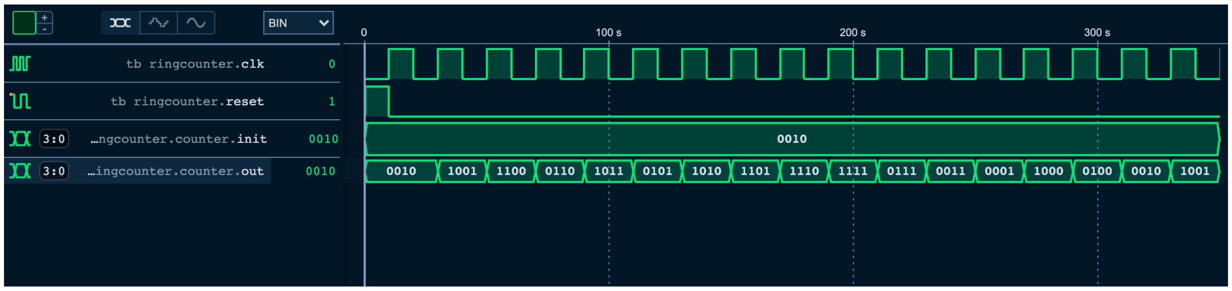


Figure 5: Waveform obtained after running simulation on Icarus-Verilog

all other 15 states. This counter is a mod-15 counter, but it does not follow the normal binary order.

ELL201 Experiment 7
Lab Report

Ritvik Gupta
2019MT10512

1 Exercise with Verilog Simulations - FSM

Entry Number: 2019MT10512
$X_3 = 1$, $X_4 = 2$
$X_3\%8 = 1 = (001)_2$
$X_4\%8 = 2 = (010)_2$
$\implies$ The sequence to be generated is $\{0,0,1,0,1,0\}$

1.1 State Diagram for the Finite State Machine

The state diagram for the FSM is shown below.

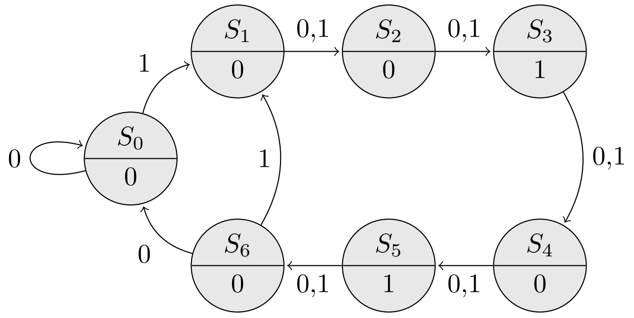


Figure 1: State diagram for the Sequence Generator

	Initial State	Final State		Output
		$X_{in} = 0$	$X_{in} = 1$	
$\rightarrow$	S_0	S_0	S_1	0
	S_1	S_2	S_2	0
	S_2	S_3	S_3	0
	S_3	S_4	S_4	1
	S_4	S_5	S_5	0
	S_5	S_6	S_6	1
$\rightarrow$	S_6	S_0	S_1	0

Table 1: State Transition table for the above diagram

From Table 1, we can see that the states S_0 and S_6 are identical. Therefore, we can reduce the FSM to **5 states**. The reduced state diagram for the FSM is shown below.

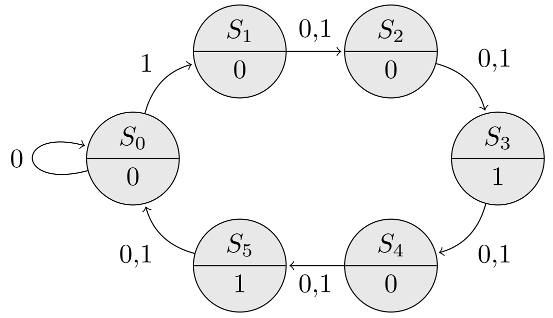


Figure 2: Reduced state diagram for the FSM

1.2 Number of Flip-flops required

In the FSM designed by us, we have **6 states** in the most reduced form.
Hence, the minimum number of D Flip-flops required to implement the FSM will be $\lceil \log_2 6 \rceil = \mathbf{3}$.

1.3 Assignment of values to each state

Let Q_2, Q_1 and Q_0 denote the state of the i^{th} D Flip-flop. We assign the values to the states S_i as follows

State	$Q_2Q_1Q_0$
S_0	0 0 0
S_1	0 0 1
S_2	0 1 0
S_3	0 1 1
S_4	1 0 0
S_5	1 0 1

Table 2: Assignment of values to the states of FSM

1.4 State Table for the FSM

Using the State diagrams drawn, we can construct the state table for the Sequence Generator.

In the table,
Q_i^n denotes the state of the i^{th} D Flip-flop at the n^{th} state.
Q_i^{n+1} denotes the state of the i^{th} D Flip-flop at the $n+1^{th}$ state.
X_{in} denotes the input received by the Sequence Generator.
Y_{out} denoted the output of the Sequence Generator.

Here, X denotes the "don't care" values.

Current State	Q_2^n	Q_1^n	Q_0^n	X_{in}	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}	Y_{out}
S_0	0	0	0	0	0	0	0	0
S_0	0	0	0	1	0	0	1	0
S_1	0	0	1	0	0	1	0	0
S_1	0	0	1	1	0	1	0	0
S_2	0	1	0	0	0	1	1	0
S_2	0	1	0	1	0	1	1	0
S_3	0	1	1	0	1	0	0	1
S_3	0	1	1	1	1	0	0	1
S_4	1	0	0	0	1	0	1	0
S_4	1	0	0	1	1	0	1	0
S_5	1	0	1	0	0	0	0	0
S_5	1	0	1	1	0	0	0	0
–	1	1	0	0	X	X	X	X
–	1	1	0	1	X	X	X	X
–	1	1	1	0	X	X	X	X
–	1	1	1	1	X	X	X	X

Table 3: State Table for the Sequence Generator FSM

1.5 Using Karnaugh Maps to minimise the expressions for D_i and Y_{out}

For D Flip-flops, we know that $Q^{n+1} = D$. Therefore, to make the Karnaugh maps for the inputs D_i, we can simply use the values of Q_i^{n+1}.
Using the state table constructed, we draw the Karnaugh Maps for the inputs D_i, and the FSM output Y_{out}. We use these Karnaugh Maps to achieve the minimised expressions for each of the inputs D_i and the output Y_{out}, in terms of Q_2, Q_1, Q_0 and X_{in}.

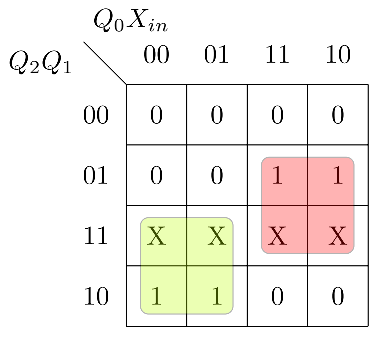
The obtained minimised expressions are as follows:

$$D_2 = Q_2Q_0' + Q_1Q_0$$

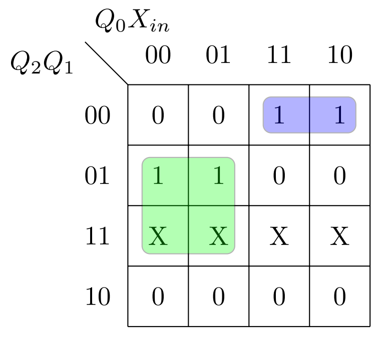
$$D_1 = Q_1Q_0' + Q_2'Q_1'Q_0$$

$$D_0 = Q_1Q_0' + Q_2Q_0' + Q_0'X_{in}$$

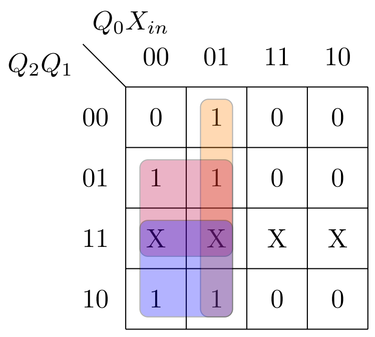
$$Y_{out} = Q_2Q_0 + Q_1Q_0$$



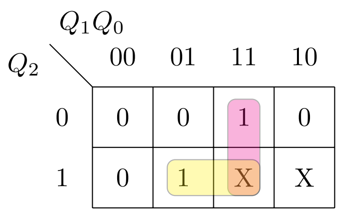
(a) Karnaugh Map for D_2



(b) Karnaugh Map for D_1



(c) Karnaugh Map for D_0



(d) Karnaugh Map for Y_{out}

Figure 3: Karnaugh Maps for the inputs D_i and the FSM output Y_{out}

1.6 Verilog Code

1.6.1 Verilog Code for the Sequence Generator FSM

```verilog
module DFlipFlop(D, clk, reset, Q);
    input D, clk, reset;
    output reg Q;

    always @ (posedge clk or posedge reset)
        if(reset) Q <= 0;
        else Q <= D;
endmodule

module SequenceGenerator(Xin, clk, reset, Yout);
    input Xin, clk, reset;
    output Yout;

    wire [2:0] Q;
    wire [2:0] D;

    assign D[2] = (Q[2] & ~Q[0]) | (Q[1] & Q[0]);
    assign D[1] = (Q[1] & ~Q[0]) | (~Q[2] & ~Q[1] & Q[0]);
    assign D[0] = (Q[1] & ~Q[0]) | (Q[2] & ~Q[0]) | (~Q[0] & Xin);

    DFlipFlop d0 (D[0], clk, reset, Q[0]);
    DFlipFlop d1 (D[1], clk, reset, Q[1]);
    DFlipFlop d2 (D[2], clk, reset, Q[2]);

    assign Yout = (Q[2] & Q[0]) | (Q[1] & Q[0]);
endmodule
```

1.6.2 Verilog Code for Testbench

```verilog
module tb_sequence;
    reg Xin;
    reg clk;
    reg reset;
    wire Yout;

    SequenceGenerator generator(Xin, clk, reset, Yout);

    always #10 clk = ~clk;
    always #40 Xin = ~Xin;
    initial
        begin
            $dumpfile("sequence.vcd");
                        $dumpvars(0, tb_sequence);
            $monitor($time," %b", Yout);
            reset <= 1;
            clk <= 0;
            Xin <= 1;
            repeat (1) @ (posedge clk);
            reset <= 0;
            repeat (17) @ (posedge clk);
            $finish;
        end
endmodule
```

1.6.3 Waveform for the Sequence Generator FSM

The figure given below shows the waveform obtained after running simulations on Icarus-Verilog.
`clk` and `reset` are the waveforms of the Clock and Reset signal. `Xin`, `Yout` and `out[3:0]` represent the input, output and the states of the Sequence Generator FSM respectively.
Here, the D Flip-flops have an asynchronous active high reset.

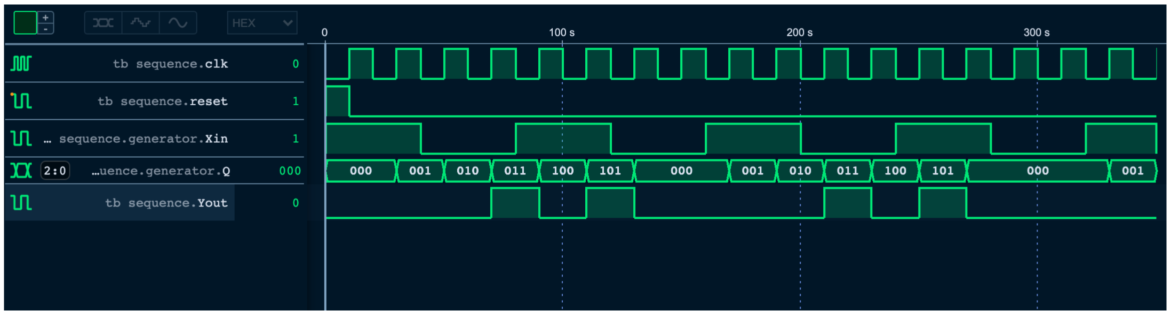


Figure 4: Waveform obtained after running simulation on Icarus-Verilog