EXPERIMENT 5

MOORE AND MEALY MACHINES

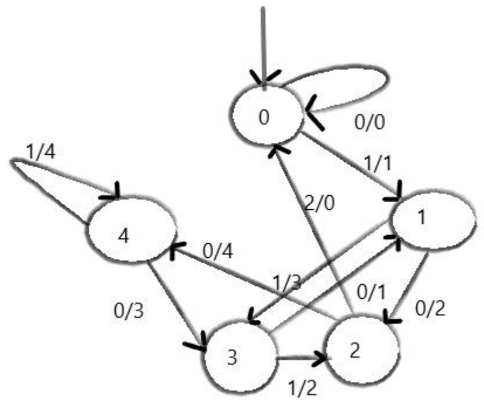
AIM OF THE EXPERIMENT: To determine residues of a positive binary number modulo 5 using mealy and moore machines.

TRUTH TABLE, CIRUIT DIAGRAM and STATE DIAGRAM:

MEALY MACHINE

Part A: Number given serially from MSB

STATE DIAGRAM



Mealy Machine with start state 0

As the input x is received the machine makes corresponding transitions and outputs he corresponding values.

Formally

M= (Q, Σ, Δ, S_0 , δ, λ)

Q={S_0, S_1, S_2, S_3, S_4}

Σ={0, 1}

Δ={0, 1, 2, 3, 4, 5}

δ(q,a)=(2q+a)mod5

λ(q,a)=q

Excitation table

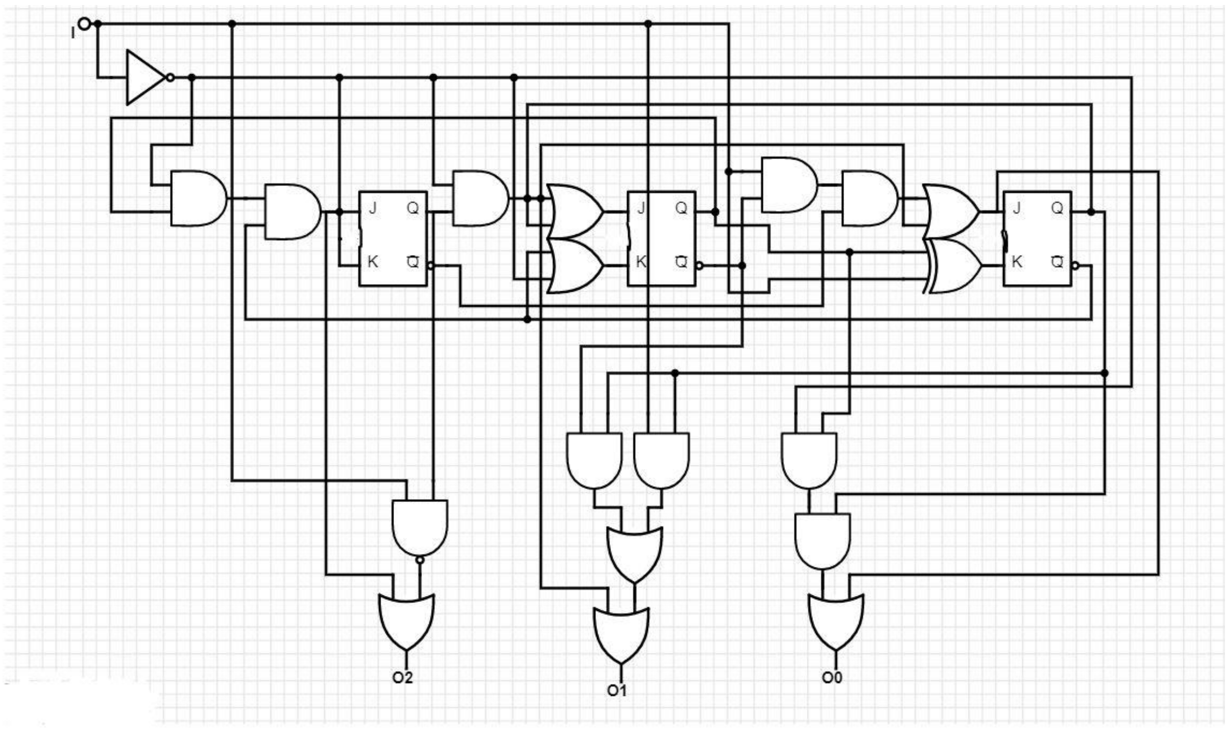
y_2	y_1	y_0	x	Y_2	Y_1	Y_0	z_2	z_1	z_0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	1	0	1	1
0	1	0	0	1	0	0	1	0	0
0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	1
0	1	1	1	0	1	0	0	1	0
1	0	0	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	0	0

$Y_0=y_2'y_1x'+y_2x+y_1y_0x'$

$Y_1=y_2x'+y_2x+y_1'y_0$

$Y_2=y_2x+y_1y_0'x'$

$z_i=y_i$



Verilog code

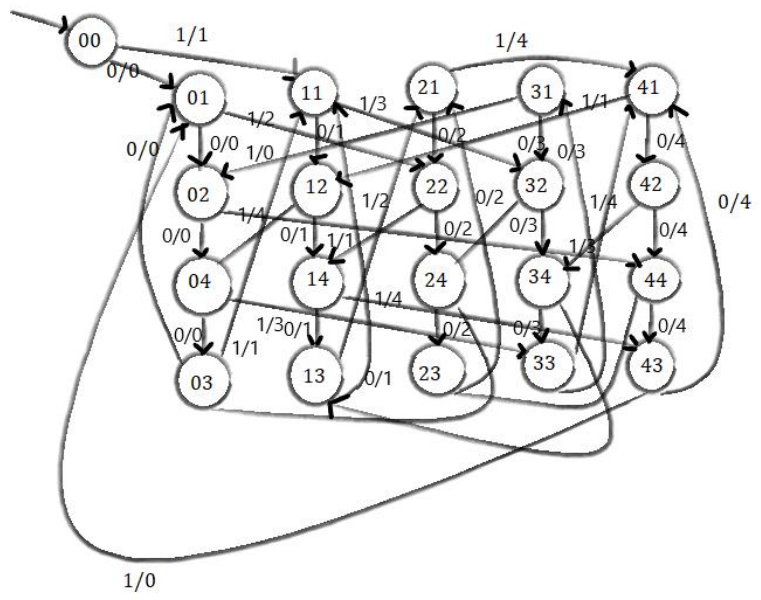
```
module msb_mealy (x, clk, reset, z);
        input x, clk, reset;
        output reg [2:0] z;
        parameter S0=0, S1=1, S2=2, S3=3, S4=4;
        reg [2:0] PS;
        always @(posedge clk or posedge reset)
                if (reset)
                        PS <= S0;
                else
                        case (PS)
                                S0: begin
                                                z = x ? 3'b001 : 3'b000;
                                                PS <= x ? S1 : S0;
                                        end
                                S1: begin
                                                z = x ? 3'b011 : 3'b010;
                                                PS <= x ? S3 : S2;
                                        end
                                S2: begin
                                                z = x ? 3'b000 : 3'b100;
                                                PS <= x ? S0 : S4;
                                        end
                                S3: begin
                                                z = x ? 3'b010 : 3'b001;
                                                PS <= x ? S2 : S1;
                                        end
                                S4: begin
                                                z = x ? 3'b100 : 3'b011;
                                                PS <= x ? S4 : S3;
                                        end
                        endcase
endmodule

module msb_mealy_test;
        reg clk, x, reset;
        wire [2:0] z;
        msb_mealy mm (x, clk, reset, z);
        initial
                begin
                        $dumpfile("msb_mealy.vcd");
                        $dumpvars(0,msb_mealy_test);
                        clk=1'b0; reset=1'b1;
                        #15 reset=1'b0;
                end
        always #5 clk= ~clk;
        initial
```

```
                begin
                        #12 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                        #10 x=0; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                        #10 $finish;
                end
endmodule
```

Part B: Number given serially from LSB

Logic: Say the residue upto current read number is r and no. of digits read is l. Then the next output would be = $(2^l*b_l+r)\bmod 5$
Now, $2^l \bmod 5$ cycles between 4 values 1,2,4 and 3 in this order. So, we maintain 20 states + 1 start state for this mealy machine.

STATE DIAGRAM



Formally
$M=(Q, \Sigma, \Delta, S_{00}, \delta, \lambda)$

$Q=\{S_{00}, S_{01},\ldots, S_{03}, \ldots\ldots ,S_{43}\}$

$\Sigma=\{0,1\}$

$\Delta=\{0,1,2,3,4\}$

$\delta(q,a)=(q(\text{left digit})+2^{q(\text{right digit})}*b)\bmod 5$

$\Delta(q,a)=q(\text{left digit})$

Verilog code:

```
module lsb_mealy (x, clk, reset, z);
        input x, clk, reset;
        output reg [2:0] z;
        parameter start=0;
        parameter zero1=1, zero2=2, zero4=3, zero3=4;
        parameter one1=5, one2=6, one4=7, one3=8;
        parameter two1=9, two2=10, two4=11, two3=12;
        parameter three1=13, three2=14, three4=15, three3=16;
        parameter four1=17, four2=18, four4=19, four3=20;
        reg [4:0] state;
        always @(posedge clk or posedge reset)
                if (reset)
                        state <= start;
                else
                        case (state)
                                start: begin
                                                        state <= x ? one1 : zero1;
                                                        z = x ? 1 : 0;
                                         end
                                zero1: begin
                                                        state <= x ? two2 : zero2;
                                                        z= x ? 2 : 0;
                                         end
                                zero2: begin
                                                        state <= x ? four4 : zero4;
                                                        z= x ? 4 : 2;
                                         end
                                zero4: begin
                                                        state <= x ? three3 : zero3;
                                                        z= x ? 3 : 0;
                                         end
                                zero3: begin
                                                        state <= x ? one1 : zero1;
                                                        z= x ? 1 : 0;
                                         end

                                one1: begin
                                                        state <= x ? three2 : one2;
                                                        z= x ? 3 : 1;
                                        end
                                one2: begin
                                                        state <= x ? zero4 : one4;
                                                        z= x ? 0 : 1;
                                        end
                                one4: begin
                                                        state <= x ? four3 : one3;
```

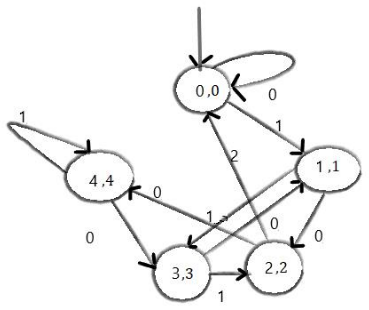
```
                              z= x ? 4 : 1;
            end
one3: begin
                              state <= x ? two1 : one1;
                              z= x ? 2 : 1;
            end

two1: begin
                              state <= x ? four2 : two2;
                              z= x ? 4 : 2;
            end
two2: begin
                              state <= x ? one4 : two4;
                              z= x ? 1 : 2;
            end
two4: begin
                              state <= x ? zero3 : two3;
                              z= x ? 0 : 2;
            end
two3: begin
                              state <= x ? three1 : two1;
                              z= x ? 3 : 2;
            end

three1: begin
                              state <= x ? zero2 : three2;
                              z= x ? 1 : 3;
                    end
three2: begin
                              state <= x ? two4 : three4;
                              z= x ? 2 : 3;
                    end
three4: begin
                              state <= x ? one3 : three3;
                              z= x ? 1 : 3;
                    end
three3: begin
                              state <= x ? four1 : three1;
                              z= x ? 4 : 3;
                    end

four1: begin
                              state <= x ? one2 : four2;
                              z= x ? 1 : 4;
             end
four2: begin
                              state <= x ? three4 : four4;
                              z= x ? 3 : 4;
```

```
                                        end
                                four4: begin
                                                        state <= x ? two3 : four3;
                                                        z= x ? 2 : 4;
                                        end
                                four3: begin
                                                        state <= x ? zero1 : four1;
                                                        z= x ? 0 : 4;
                                        end
                        endcase
endmodule


module lsb_mealy_test;
        reg clk, x, reset;
        wire [2:0] z;
        lsb_mealy lm (x, clk, reset, z);
        initial
                begin
                        $dumpfile("lsb_mealy.vcd");
                        $dumpvars(0,lsb_mealy_test);
                        clk=1'b0; reset=1'b1;
                        #15 reset=1'b0;
                end
        always #5 clk= ~clk;
        initial
                begin
                        #12 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=0;
                        #10 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                        #10 $finish;
                end
endmodule
```

MOORE MACHINE

Part A: Number serially from MSB

STATE DIAGRAM



The logic is same as that of the Mealy machine, except that the output is corresponding to the state the machine is in.

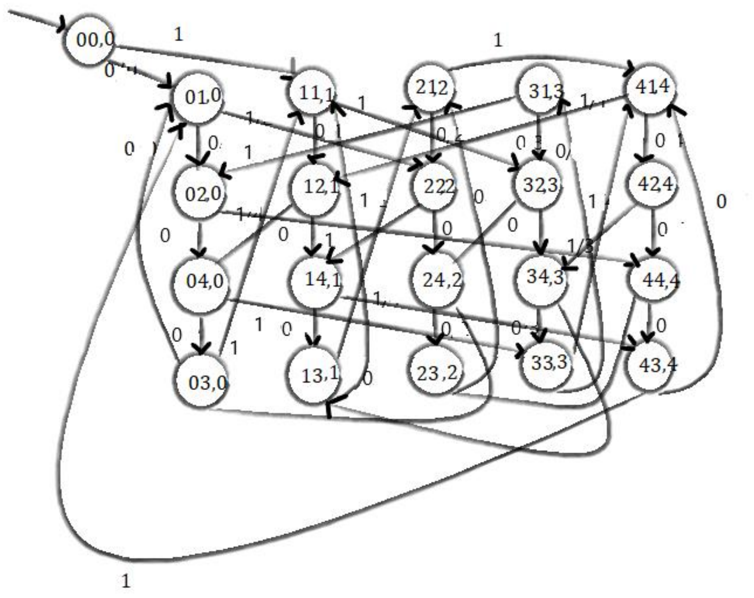
Verilog code

```
module msb_moore (x, clk, reset, z);

        input x, clk, reset;
        output reg [2:0] z;
        parameter S0=0, S1=1, S2=2, S3=3, S4=4;
        reg [2:0] state;
        always @(posedge clk or posedge reset)
                if (reset)
                        state <= S0;
                else
                        case (state)
                                S0: state <= x ? S1 : S0;
                                S1: state <= x ? S3 : S2;
                                S2: state <= x ? S0 : S4;
                                S3: state <= x ? S2 : S1;
                                S4: state <= x ? S4 : S3;
                        endcase
        always @(state)
                case (state)
                        S0: z=3'b000;
                        S1: z=3'b001;
                        S2: z=3'b010;
                        S3: z=3'b011;
                        S4: z=3'b100;
                endcase
endmodule
```

```
module msb_moore_test;
        reg clk, x, reset;
        wire [2:0] z;
        msb_moore mm (x, clk, reset, z);
        initial
                begin
                        $dumpfile("msb_moore.vcd");
                        $dumpvars(0,msb_moore_test);
                        clk=1'b0; reset=1'b1;
                        #15 reset=1'b0;
                end
        always #5 clk= ~clk;
        initial
                begin
                        #12 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                        #10 x=0; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                        #10 $finish;
                end
endmodule
```

Part B: Number serially from LSB

STATE DIAGRAM



The state machine is the same as the mealy machine, except that the output corresponging to each state is fixed.

Verilog code:

```
module lsb_moore (x, clk, reset, z);
        input x, clk, reset;
        output reg [2:0] z;
        parameter start=0;
        parameter zero1=1, zero2=2, zero4=3, zero3=4;
        parameter one1=5, one2=6, one4=7, one3=8;
        parameter two1=9, two2=10, two4=11, two3=12;
        parameter three1=13, three2=14, three4=15, three3=16;
        parameter four1=17, four2=18, four4=19, four3=20;
        reg [4:0] state;
        always @(posedge clk or posedge reset)
                if (reset)
                        state <= start;
                else
                        case (state)
                                start: state <= x ? one1 : zero1;
                                zero1: state <= x ? two2 : zero2;
                                zero2: state <= x ? four4 : zero4;
                                zero4: state <= x ? three3 : zero3;
                                zero3: state <= x ? one1 : zero1;

                                one1: state <= x ? three2 : one2;
                                one2: state <= x ? zero4 : one4;
                                one4: state <= x ? four3 : one3;
                                one3: state <= x ? two1 : one1;

                                two1: state <= x ? four2 : two2;
                                two2: state <= x ? one4 : two4;
                                two4: state <= x ? zero3 : two3;
                                two3: state <= x ? three1 : two1;

                                three1: state <= x ? zero2 : three2;
                                three2: state <= x ? two4 : three4;
                                three4: state <= x ? one3 : three3;
                                three3: state <= x ? four1 : three1;

                                four1: state <= x ? one2 : four2;
                                four2: state <= x ? three4 : four4;
                                four4: state <= x ? two3 : four3;
                                four3: state <= x ? zero1 : four1;
                        endcase
                always @(state)
                        case(state)
                                start: z=3'b000;
                                zero1: z=3'b000;
                                zero2: z=3'b000;
```

```
                                        zero4: z=3'b000;
                                        zero3: z=3'b000;

                                        one1: z=3'b001;
                                        one2: z=3'b001;
                                        one4: z=3'b001;
                                        one3: z=3'b001;

                                        two1: z=3'b010;
                                        two2: z=3'b010;
                                        two4: z=3'b010;
                                        two3: z=3'b010;

                                        three1: z=3'b011;
                                        three2: z=3'b011;
                                        three4: z=3'b011;
                                        three3: z=3'b011;

                                        four1: z=3'b100;
                                        four2: z=3'b100;
                                        four4: z=3'b100;
                                        four3: z=3'b100;
                              endcase
endmodule

module lsb_moore_test;
          reg clk, x, reset;
          wire [2:0] z;
          lsb_moore lm (x, clk, reset, z);
          initial
                    begin
                              $dumpfile("lsb_moore.vcd");
                              $dumpvars(0,lsb_moore_test);
                              clk=1'b0; reset=1'b1;
                              #15 reset=1'b0;
                    end
          always #5 clk= ~clk;
          initial
                    begin
                              #12 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=0;
                              #10 x=1; #10 x=0; #10 x=1; #10 x=0; #10 x=1;
                              #10 $finish;
                    end
endmodule
```

RESULTS

The same input in binary is given in all 4 test cases, i.e. $(1010100101)_2$ and the corresponding remainders at each stage of corresponding mealy and moore machines are consistent.

DISCUSSION

- ➢ To implement each machine, we have mentioned a procedural code. It automatically creates memory elements (D-flip-flops) for each state and each output.
- ➢ We can visualize each state for MSB machine as $y_2y_1y_0$ binary number and we can consider their excitation tables too for their corresponding state transitions consistent with input and their previous values.
- ➢ We have intentionally given in input after 2 secs of the positive edge of the clock, so that there is no ambiguity in case of the machine to recognize the input or the state.
- ➢ The output is obtained on implementing the Verilog code in gtk wave.

Prepared by-Group2
Somnath Jena
Soumitra Das