EECS 151/251A Homework 3 Solution

Problem 1: Simplifying with Karnaugh Maps

Use the following truth table to answer the questions.

A	B	C	D	output
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

(a) Write the sum of products expression for the truth table.

(b) Use a Karnaugh map to write a simplified boolean expression for the truth table.

(c) [251A Only] Using boolean algebra, prove that your Karnaugh map expression is equivalent to your sum of products expression.

Solution:

(a) $a'b'cd + ab'c'd' + ab'c'd + ab'cd + abc'd' + abc'd + abcd$

(b) Karnaugh map:

cd \ ab	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	1	1
10	0	0	0	0

Expression: $ac' + ad + b'cd$

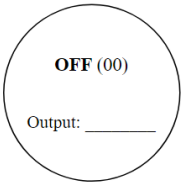
(c) $a'b'cd + ab'c'd' + ab'c'd + ab'cd + abc'd' + abc'd + abcd$
$= (ab'c'd' + ab'c'd + abc'd' + abc'd) + (ab'c'd + ab'cd + abc'd + abcd) + (a'b'cd + ab'cd)$
$= ac'(b'd' + b'd + bd' + bd) + ad(b'c' + b'c + bc' + bc) + b'cd(a' + a)$
$= ac'(b'(d' + d) + b(d' + d)) + ad(b'(c' + c) + b(c' + c)) + b'cd$
$= ac'(b' + b) + ad(b' + b) + b'cd$
$= ac' + ad + b'cd$

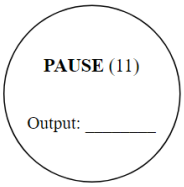
Problem 2: Microwave FSM

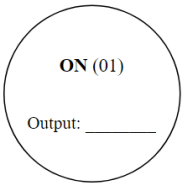
Read the following microwave FSM specification and answer the questions.

> The microwave has two buttons: a stop button and a start button. When the microwave is off, the lights are off and the food is not heated. Pressing the stop button when the microwave is off does nothing. Pressing the start button when the microwave is off turns the microwave on. When the microwave is on, the lights are on and the food is heated. Pressing the start button again when the microwave is already on does nothing. Pressing the stop button when the microwave is on pauses the microwave. The lights stay on when the microwave is paused, but the food is no longer heated. When the microwave is paused, pushing the start button turns the microwave back on, and pushing the stop button turns the microwave off.

(a) Complete the Moore FSM diagram for the microwave FSM. Let bit 1 of the output represent the microwave lights and bit 0 of the output represent the microwave heat, and let pressing the start button give an input of 1 and pressing the stop button give an input of 0.





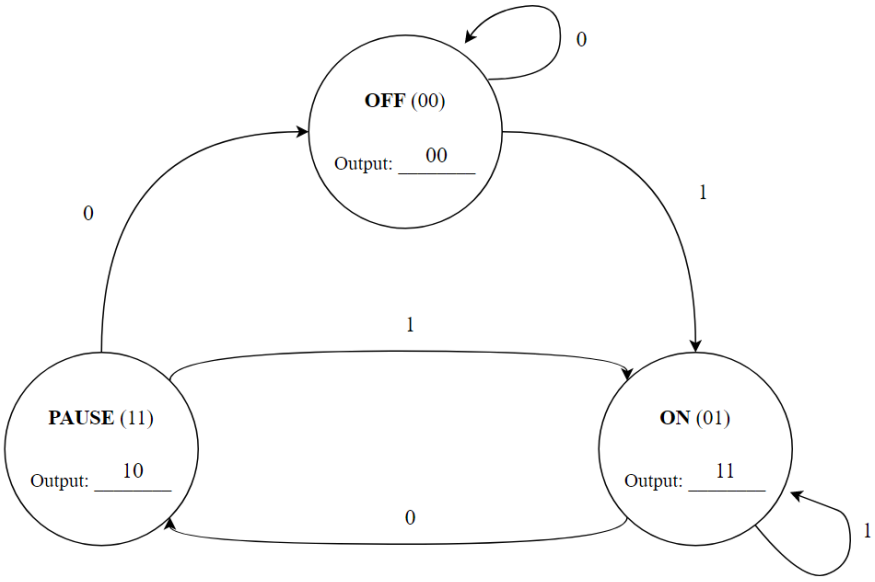


(b) Write the truth table for this FSM.

(c) Using Karnaugh maps, compute a boolean expression for each bit of the next state.

Solution:

(a) FSM Diagram:



(b) Truth Table:

Input	Current State	Next State	Output
0	00	00	00
1	00	01	00
0	01	11	11
1	01	01	11
0	11	00	10
1	11	01	10

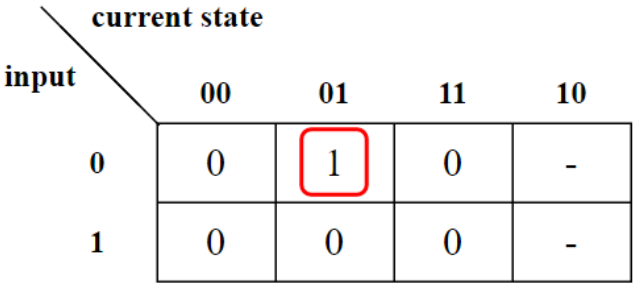
(c) Karnaugh Map for next[0]:

input \ current state	00	01	11	10
0	0	1	0	-
1	1	1	1	-

Expression for next[0]: $\text{input} + (\text{current\_state}[0] * \text{current\_state}[1]')$

Solution:

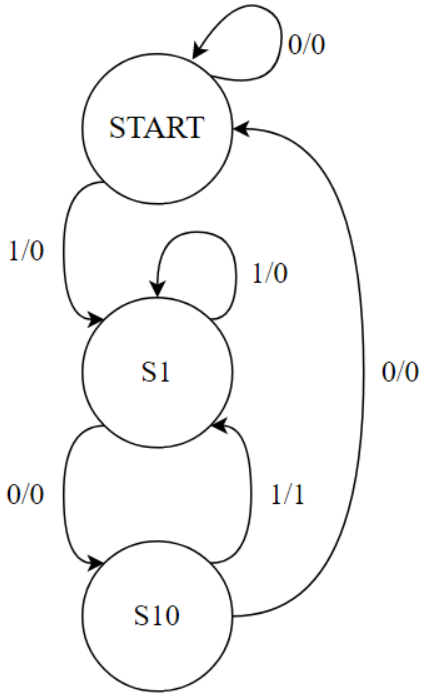
(c) (cont.)
Karnaugh Map for next[1]:



Expression for next[1]: $\text{input}' * \text{current\_state}[0] * \text{current\_state}[1]'$

Problem 3: FSMs in Verilog

Use the following FSM diagram for this question.



(a) The verilog starter code for this FSM is shown below:

```
module fsm(
        input clk,
        input rst,
        input in,
        output out
);
        parameter START = 2'b00,
                  S1    = 2'b01,
                  S10   = 2'b10;

        reg [1:0] current_state;
        reg [1:0] next_state;
        reg out_state;

        assign out = out_state;

        always @(posedge clk) begin
            //Synchronous state update logic
            //TODO: Part (i) code goes here
        end
```

```
    always @(*) begin
        //Combinational state update logic
        //TODO: Part (ii) code goes here
    end

    always @(*) begin
        //Output computation
        //TODO: Part (iii) code goes here
    end
endmodule
```

i. Write the verilog code that goes in the synchronous state update always block.

ii. Write the verilog code that goes in the combinational state update always block. Use a case statement.

iii. Write the verilog code that goes in the output always block. Use a case statement.

(b) Describe the function of this FSM.

(c) Is this a Moore state machine or a Mealy state machine? Explain your reasoning.

(d) Redraw this FSM as the other type of state machine (ex. Moore to Mealy, Mealy to Moore).

(e) [251A Only] Write a verilog module that implements the FSM in your diagram from part (d).

Solution:

(a) i.
```
if (rst) current_state <= START;
else current_state <= next_state;
```

ii.
```
case (current_state)
    START: begin
        if (in) next_state = S1;
        else next_state = START;
    end
    S1: begin
        if (in) next_state = S1;
        else next_state = S10;
    end
    S10: begin
        if (in) next_state = S1;
        else next_state = START;
    end
    default: begin
        next_state = START;
    end
endcase
```

Solution:

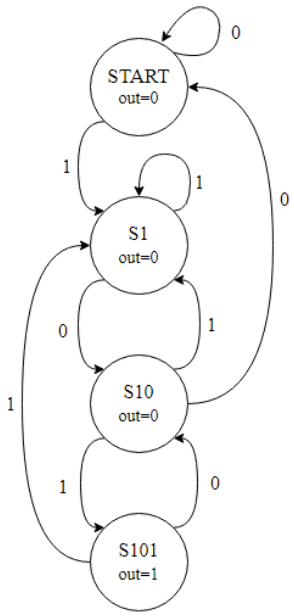
(a) (cont.)

iii.
```
case (current_state)
        START: begin
            out = 1'b0;
        end
        S1: begin
            out = 1'b0;
        end
        S10: begin
            if (in) out = 1'b1;
            else out = 1'b0;
        end
        default: begin
            out = 1'b0;
        end
    endcase
```

(b) This FSM detects overlapping patterns of 101 from a bitstream.

(c) This is a Mealy state machine because the output is dependent on the value of the input.

(d) Moore state machine version:



Solution:

(c)
```
module fsm(
            input clk,
            input rst,
            input in,
            output out
    );
            parameter START = 2'b00,
                      S1    = 2'b01,
                      S10   = 2'b10,
                      S101  = 2'b11;

            reg [1:0] current_state;
            reg [1:0] next_state;
            reg out_state;

            assign out = out_state;

            always @(posedge clk) begin
                if (rst) current_state <= START;
                else current_state <= next_state;
            end

            always @(*) begin
                case (current_state)
                    START: begin
                        out_state = 1'b0;
                        if (in) next_state = S1;
                        else next_state = START;
                    end
                    S1: begin
                        out_state = 1'b0;
                        if (in) next_state = S1;
                        else next_state = S10;
                    end
                    S10: begin
                        out_state = 1'b0;
                        if (in) next_state = S101;
                        else next_state = START;
                    end
                    S101: begin
                        out_state = 1'b1;
                        if (in) next_state = S1;
                        else next_state = S10;
                    end
                    default: begin
```

```
                    out_state = 1'b0;
                    next_state = START;
                end
            endcase
        end
endmodule
```