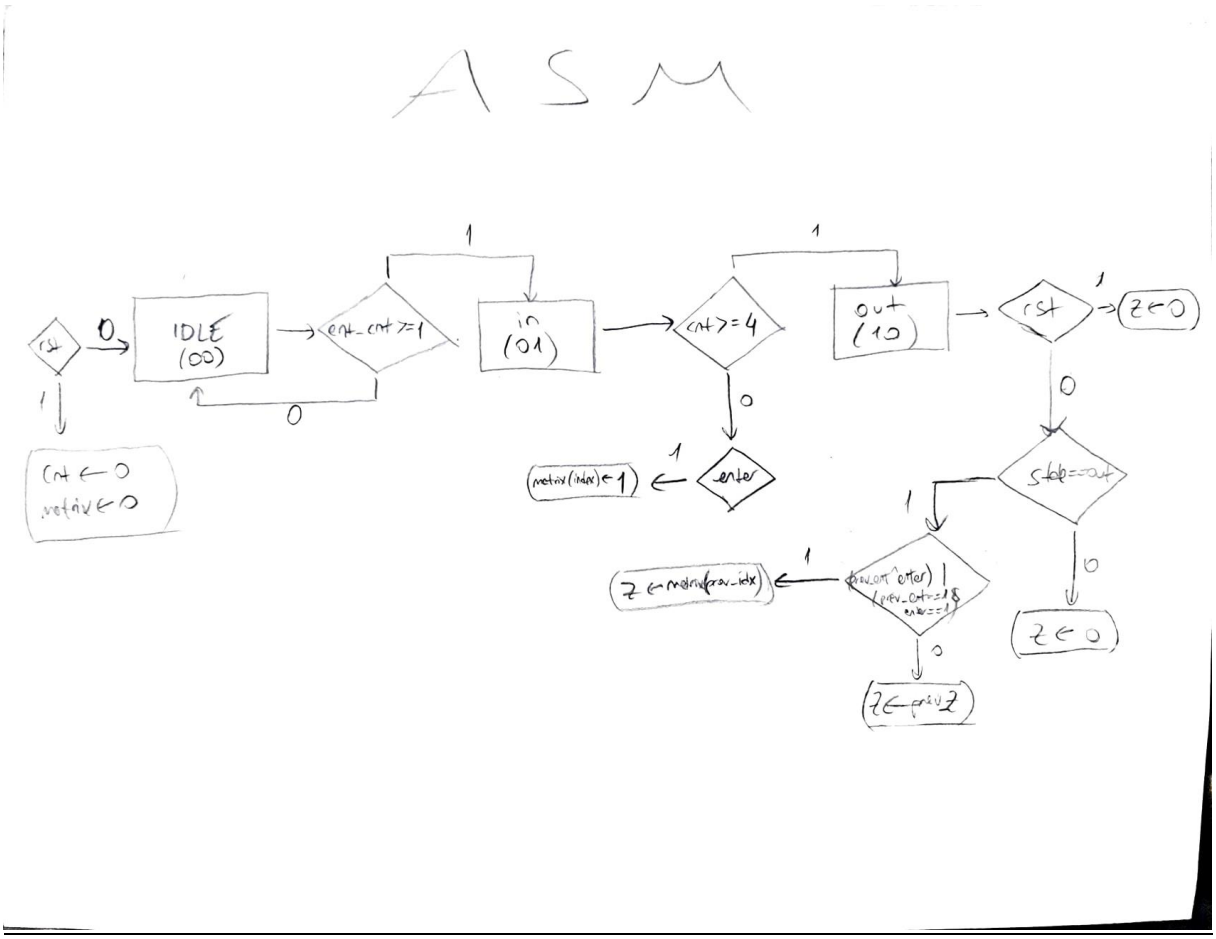
Algorithmic State Machine:




```
// 3 states
parameter IDLE = 2'b00;
parameter in = 2'b01;
parameter out = 2'b10;

//internal variables
reg [1:0] state;
reg [2:0] cnt;
reg [3:0] ent_cnt;
reg [1:0] prev_Y;
reg [1:0] prev_X;
reg  prev_ent;
reg prev_Z;


// one dimensional matrix structure
wire [3:0] index = {Y,X};
reg [15:0] matrix;
```

- IDLE, in and out are declared as parameters which represent the three states of ASM. prev_X, prev_Y, prev_Z and prev_ent are declared in order to keep track of the previous values for each. To implement the matrix, one dimensional matrix structure is used, and the indices are decided by concatenating Y and X.

```
//state transitions
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= IDLE;
        cnt <= 3'd0;
        matrix <= 16'b0;
    end else begin
        case (state)
            IDLE: begin
                if (ent_cnt>=1) begin
                  state <= in;
                  if (enter) begin
                    matrix[index] <= 1'b1; //inserting the first element into matrix
                  end

                end
            end

            in: begin
                if (cnt < 3'd4 & enter) begin
                    matrix[index] <= 1'b1;

                end else if (cnt >= 3'd4)begin
                    state <= out;
                end
            end

            out: begin end
        endcase
    end
end
```

- In this piece of code, the state transitions are established. If the enter counter exceeds one, transition must occur from IDLE to “in” state. Similarly, if 5 inputs are taken, we must move on with the “out” state. Additionally, the module uses the positive edge of the clock to sample inputs like enter, X, and Y by using the always block with “posedge clk”.

```
   // Counter implementation which counts the number of enters
  always @(posedge clk or posedge rst) begin
   if(rst) begin
     ent_cnt <=3'b0;
   end else if (enter) begin
     ent_cnt<= ent_cnt+1;
   end
 end

 //Counter implementation which counts the number of inputs
 always @(posedge clk or posedge rst) begin
   if(rst) begin
       cnt <= 3'd0;
   end
   else if (enter) begin
       if (ent_cnt>= 4'd2) begin
           cnt <= cnt + 3'd1;
       end
   end
 end
```

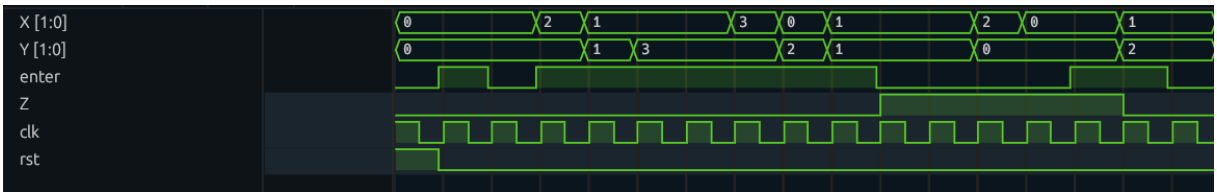
- Here are the implementations of enter counter and counter. Enter counter is kept for the transition from IDLE to “in”, and counter is kept for transition from “in” to “out”.

```
// Updating the previous values
always @(negedge clk) begin
  prev_Y<=Y;
  prev_X<=X;
  prev_ent<=enter;
end

wire [3:0] prev_idx = {prev_Y,prev_X};

always @(negedge clk) begin
  if (state == out) begin
    prev_Z<=Z;
  end
end
```

- To update the prev versions of X, Y, Z and enter, the above code piece is used.

```
// Deciding on output (Z value) by covering all transitions of enter from one phase to another
  always @(posedge clk or posedge rst) begin
      if (rst) begin
          Z <= 1'b0;
      end else if ((((!prev_ent & !enter)|(enter==1&prev_ent==0))&(state == out)) begin
          Z <= prev_Z;
      end else if (state == out & ((prev_ent ^ enter)|(prev_ent==1&enter==1))) begin

          Z <= matrix[prev_idx];
      end else begin
          Z <= 1'b0;
  end
end
```

- Here, this always block decides on the Z value by taking care of all combinations of enter and prev_ent. By updating Z only in the out state, Z can stay 0 consistently until to the “out” state. As an additional caution, the Z value is decided as 0 in all other possibilities than “out” state.



- The code is tested under various conditions and gave the correct waveform. The above simulation used as an example demonstration.