1. A serial parity-bit generator is a sequential circuit that does the following: it receives an \( n \)-bit message followed by a 0 (so there are \( n + 1 \) clock bits to send the message). At the output, the circuit sends the original \( n \)-bit message, but replaces the 0 with a parity bit. Design a 4-bit serial parity-bit generator which replaces the zero bit with an odd parity bit.

(a) Draw a state diagram for the circuit.

As you receive the four data bits, you want to send out the same bits, and keep track of whether you received an even or odd number of bits. You need a state which tells you that you have received one bit, and that you have seen an even number of ones (state \( S_{1\_E} \)); and you need a state which tells you that you have received one bit, and that you have seen an odd number of ones (state \( S_{1\_O} \)). You need similar states for receiving two, three, and four bits. After you receive the fourth bit, you will go back to the reset state (no bits received), and and send out a 0 if you have seen an odd number of ones, and put out a 1 if you have seen an even number of ones.
(b) Draw a state transition table for the circuit.

You have 9 states, so you need at least 4 flip-flops. There are 7 unused states. I assign the states so the bits $Q_2, Q_1, Q_0$ tell me how many bits of data we have received, and bit $Q_3$ indicates whether we have received an even or odd number of 1’s. In State $S_0$, we haven’t received any bits, so we have seen an even number of 1’s.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Flip-Flops</th>
<th>Input</th>
<th>Next State</th>
<th>Flip-Flops</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>0</td>
<td>$S_1E$</td>
<td>0 0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>1</td>
<td>$S_1O$</td>
<td>1 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>$S_1E$</td>
<td>0 0 0 1</td>
<td>0</td>
<td>$S_2E$</td>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>$S_1E$</td>
<td>0 0 0 1</td>
<td>1</td>
<td>$S_2O$</td>
<td>1 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_1O$</td>
<td>1 0 0 1</td>
<td>0</td>
<td>$S_2O$</td>
<td>1 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>$S_1O$</td>
<td>1 0 0 1</td>
<td>1</td>
<td>$S_2E$</td>
<td>0 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_2E$</td>
<td>0 0 1 0</td>
<td>0</td>
<td>$S_3E$</td>
<td>0 0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>$S_2E$</td>
<td>0 0 1 0</td>
<td>1</td>
<td>$S_3O$</td>
<td>1 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>$S_2O$</td>
<td>1 0 1 0</td>
<td>0</td>
<td>$S_3O$</td>
<td>1 0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>$S_2O$</td>
<td>1 0 1 0</td>
<td>1</td>
<td>$S_3E$</td>
<td>0 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>$S_3E$</td>
<td>0 0 1 1</td>
<td>0</td>
<td>$S_4E$</td>
<td>0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>$S_3E$</td>
<td>0 0 1 1</td>
<td>1</td>
<td>$S_4O$</td>
<td>1 1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_3O$</td>
<td>1 0 1 1</td>
<td>0</td>
<td>$S_4O$</td>
<td>1 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>$S_3O$</td>
<td>1 0 1 1</td>
<td>1</td>
<td>$S_4E$</td>
<td>0 1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_4E$</td>
<td>0 1 0 0</td>
<td>0</td>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_4E$</td>
<td>0 1 0 0</td>
<td>1</td>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$S_4O$</td>
<td>1 1 0 0</td>
<td>0</td>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>$S_4O$</td>
<td>1 1 0 0</td>
<td>1</td>
<td>$S_0$</td>
<td>0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>
(c) Show how to implement the circuit using D flip-flops.

You have 4 flip-flops and 1 input bit, so you would have to draw five 5-input Karnaugh maps to find the inputs to the flip-flops and the output bit. Rather than doing that, I note that the three least significant bits $Q_2Q_1Q_0$ count 0, 1, 2, 3, 4, 0, 1, 2, 3, 4 ... I can get this behavior by using a modulus five counter.

To get bit $Q_3$, I note that if bit $Q_3$ was a 0 and the input is 0, $Q_3$ stays a 0; if bit $Q_3$ was a 0 and the input is 1, $Q_3$ goes to a 1; if bit $Q_3$ was a 1 and the input is 0, $Q_3$ goes to a 1; if bit $Q_3$ was a 1 and the input is 1, $Q_3$ stays a 1 (except that in state $S_4$, $Q_3$ always goes to 0). If you are in states $S_0$ through $S_3$, bit $Q_3$ will become $Q_3 \oplus D_{in}$; if you are in state $S_4$, bit $Q_3$ will become 0. If you are in states $S_0$ through $S_3$, $Q_2$ will be 0, and if you are in state $S_4$, $Q_2$ will be 1. This gives the equation:

$$Q_3(t + 1) = Q'_2(Q_3 \oplus D_{in})$$

The output is always equal to the input in states $S_0$ through $S_3$; the output in $S_4_E$ is 1 and the output in $S_4_O$ is 0. This gives the equation:

$$D_{out} = Q'_2D_{in} + Q_2Q'_3$$

(d) Write a Verilog program to implement the circuit.

```verilog
module serial_parity_generator(input clk, reset, Din, output reg Dout);
parameter S0 = 4'b0000, /* List the names of the states */
    S1_E = 4'b0001,
    S1_O = 4'b1001,
    S2_E = 4'b0010,
    S2_O = 4'b1010,
    S3_E = 4'b0011,
    S3_O = 4'b1011,
    S4_E = 4'b0100,
    S4_O = 4'b1100;
```
/* Need 4 ff's for 9 states */
reg [3:0] state;

always @(posedge clk, negedge reset)
if (~reset) state <= S0;
else case (state)
  S0: if (Dout) state <= S1_O;
       else state <= S1_E;
  S1_E: if (Dout) state <= S2_0;
         else state <= S2_E;
  S1_O: if (Dout) state <= S2_E;
         else state <= S2_O;
  S2_E: if (Dout) state <= S3_0;
         else state <= S3_E;
  S2_0: if (Dout) state <= S3_E;
         else state <= S3_0;
  S3_E: if (Dout) state <= S4_0;
         else state <= S4_E;
  S3_0: if (Dout) state <= S4_E;
         else state <= S4_0;
  S4_E: state <= S0;
  S4_0: state <= S0;
endcase

always @(state, Din)
  case (state)
    S0 : Dout = Din;
    S1_E: Dout = Din;
    S1_O: Dout = Din;
    S2_E: Dout = Din;
    S2_0: Dout = Din;
    S3_E: Dout = Din;
    S3_0: Dout = Din;
    S4_E: Dout = 1'b1;
    S4_0: Dout = 1'b0;
  endcase

endmodule

(e) Is this a Mealy machine or a Moore machine? Why?
The output depends on the present state and the current input, so it is a Mealy machine.
2. Design a synchronous sequential circuit that will count through the sequence 0, 2, 4, 6 when its control input \( x \) is 0, and through the sequence 6, 4, 2, 0 when \( x = 1 \). The circuit should return to the 0 state if it finds itself in an invalid state.

(a) Draw a state diagram for the circuit.

(b) Draw a state transition table for the circuit.

You have eight states, so you need three flip-flops:

\[
\begin{array}{c|ccc|c|ccc}
\text{State} & \text{Flip-Flops} & \text{Next State} & \text{Flip-Flops} \\
 & Q_2 & Q_1 & Q_0 & & Q_2 & Q_1 & Q_0 \\
\hline
S_0 & 0 & 0 & 0 & S_2 & 0 & 1 & 0 \\
S_1 & 0 & 0 & 1 & S_0 & 0 & 0 & 0 \\
S_2 & 0 & 1 & 0 & S_4 & 1 & 0 & 0 \\
S_3 & 0 & 1 & 1 & S_0 & 0 & 0 & 0 \\
S_4 & 1 & 0 & 0 & S_6 & 1 & 1 & 0 \\
S_5 & 1 & 0 & 1 & S_0 & 0 & 0 & 0 \\
S_6 & 1 & 1 & 0 & S_0 & 0 & 0 & 0 \\
S_7 & 1 & 1 & 1 & S_0 & 0 & 0 & 0 \\
\end{array}
\]
(c) Write a Verilog module to implement the system.

```verilog
module hw9_p2(input reset, clk, output reg [3:0] Q);

parameter S0 = 3'b000,
    S1 = 3'b001,
    S2 = 3'b010,
    S3 = 3'b011,
    S4 = 3'b100,
    S5 = 3'b101,
    S6 = 3'b110,
    S7 = 3'b111;

always @(posedge clk, negedge reset)
    if (~reset) Q <= S0;
else case (Q)
    S0: Q <= S2;
    S2: Q <= S4;
    S4: Q <= S6;
    S6: Q <= S0;
    default: Q <= S0;
endcase

endmodule
```
3. The serial adder of Fig. 6.6 uses two four-bit shift registers. Register \( A \) holds the binary number 1101 and register \( B \) holds 0110. The carry flip-flop is initially reset to 0. List the binary values in register \( A \) and the carry flip-flop after each shift.

The system will behave according to the state table of Figure 6.2:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs x y</th>
<th>Next State Q</th>
<th>Output S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( x \) and \( y \) are the LSB of \( A \) and \( B \). After each clock cycle, we will have:

<table>
<thead>
<tr>
<th>Before Clock</th>
<th>After Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1101 B 0110 x 0 y 0 Q S 0</td>
<td>A 1110 B x011</td>
</tr>
<tr>
<td>1110 x011 0 1 0 1 0 1</td>
<td>1111 xx01</td>
</tr>
<tr>
<td>1111 xxx0 1 1 0 0 1 1 0 1</td>
<td>0111 xxx0</td>
</tr>
<tr>
<td>0111 xxx0 1 0 1 0 1 0 0 1</td>
<td>0011 xxx0</td>
</tr>
</tbody>
</table>

Before the first clock cycle, \( x \) will be 1 and \( y \) will be 0 (the LSB of \( A \) and \( B \)), and \( Q \) will be 0 (reset condition). This will make \( S \) equal to 1, and the \( J - K \) inputs will be 00, so the \( J - K \) flip-flop will hold \( Q \) at 0. After the clock, \( A \) will shift one to the right, and \( S \) will shift in from the right, so \( A \) becomes 1 \( \Rightarrow \) 1101 \( \Rightarrow \) 1110, and \( Q \) will become 0. We don’t know what is on the Serial Input to \( B \), so \( B \) becomes \( x \rightarrow 0110 \Rightarrow x011 \). The second clock cycle will start with \( A = 1110 \), \( B = x011 \), and \( Q = 0 \). After four clock cycles, \( A \) will become 0011, and \( Q \) will be 1. This is correct: 1101 + 0110 = 1\_0011.
4. Consider the following Verilog statements

(a) \texttt{RegA <= 32;}  \\
\texttt{RegB <= RegA;}

Assume that \texttt{RegA} contains the value of 45 and \texttt{RegB} contains the value of 32 initially. What are the values of \texttt{RegA} and \texttt{RegB} after execution?

The two statements are executed at the same time, on the rising edge of the clock. Before the clock edge, \texttt{RegA} has a 45, so after the clock edge, \texttt{RegA} will have a 32 and \texttt{RegB} will have a 45.

(b) \texttt{RegA = 32;}  \\
\texttt{RegB = RegA;}

Assume that \texttt{RegA} contains the value of 45 and \texttt{RegB} contains the value of 32 initially. What are the values of \texttt{RegA} and \texttt{RegB} after execution?

The two statements are executed sequentially. A 32 goes into \texttt{RegA}, then that 32 goes into \texttt{RegB}. After the clock edge, \texttt{RegA} and \texttt{RegB} will both be 32.
5. Consider the following Verilog code fragment:

```verilog
reg [3:0] A, B;

always @( posedge clock )
begin
  A = 5;
  B = A + 2;
end
```

Before the clock edge, A has the value of 3, and B has the value of 6. What will be the values of A and B after the clock edge?

The statements are executed sequentially. A will go to 5, then B will go to 7 (A + 2).

6. Consider the following Verilog code fragment:

```verilog
reg [3:0] A, B;

always @( posedge clock )
begin
  A <= 5;
  B <= A + 2;
end
```

Before the clock edge, A has the value of 3, and B has the value of 6. What will be the values of A and B after the clock edge?

The statements are executed simultaneously. A will go to 5, then B will go to 7 (3 + 2).