In Task 1, we have to design an FSM that generates the output sequence determined by the personal code. If the personal code is 3527461, then the FSM should generate the 0, 3, 5, 2, 7, 4, 6, 1, … repeating output sequence and the output values must be taken directly from the state register (output encoding). The following state table describes the behavior of the FSM. Because output encoding is used, we can get the encoded state table by converting the decimal values to binary.

<table>
<thead>
<tr>
<th>Current state</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current state $S_2S_1S_0$</th>
<th>Next state $N_2N_1N_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>011</td>
</tr>
<tr>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>010</td>
<td>111</td>
</tr>
<tr>
<td>011</td>
<td>101</td>
</tr>
<tr>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>110</td>
<td>001</td>
</tr>
<tr>
<td>111</td>
<td>100</td>
</tr>
</tbody>
</table>

The encoded state table is the truth table of the next state logic. The minimized two-level logic equations can be derived from the Karnaugh maps of the next state bits $N_2$, $N_1$ and $N_0$.

$$N_2 = S_0'S_1'S_2 + S_0S_1 + S_1S_2'$$

$$N_1 = S_0'S_2' + S_1'S_2$$

$$N_0 = S_0'S_2' + S_1S_2' + S_0'S_1$$

The logic equation of the $N_2$ next state bit can be simplified further using the De Morgan’s law if XOR/XNOR gates are allowed. The circuit diagram can be drawn up according to the equations. The Verilog HDL implementation of the FSM can be seen below.

```verilog
module task1(
    input wire clk,
    input wire rst,
    output wire [2:0] y
); //Defining the states.
localparam S0 = 3'd0;
localparam S1 = 3'd1;
localparam S2 = 3'd2;
localparam S3 = 3'd3;
localparam S4 = 3'd4;
localparam S5 = 3'd5;
localparam S6 = 3'd6;
localparam S7 = 3'd7;

//State register.
reg [2:0] state;
//Driving the output.
assign y = state;

//Implementing the state register
//and the next state logic.
always @(posedge clk)
begin
    if (rst)
        state <= S0;
    else
        case (state)
            S0: state <= S3;
            S1: state <= S0;
            S2: state <= S7;
            S3: state <= S5;
            S4: state <= S3;
            S5: state <= S2;
            S6: state <= S1;
            S7: state <= S4;
        default: state <= S0;
        endcase
end
endmodule
```
In **Task 2**, we have to design a 3-bit binary up-counter and a combinational output logic that maps the output of the counter in natural order to the numbers in the order determined by the personal code. If the personal code is 3527461, then the 01234567 → 03527461 mapping should be done (see the table below). We can get the truth table of the mapping logic by converting the decimal numbers to binary. The counter design is not described here because it is very similar to the **Task 1**.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

The minimized two-level logic equations can be derived from the Karnaugh maps of the output bits $Y_2$, $Y_1$ and $Y_0$.

$$Y_2 = X_0'X_1 + X_1'X_2$$
$$Y_1 = X_0X_2' + X_0'X_2$$
$$Y_0 = X_0X_1'X_2' + X_0'X_1X_2' + X_0'X_1'X_2 + X_0X_1X_2 = X_0 \text{ xor } X_1 \text{ xor } X_2$$

The logic equations of the $Y_1$ and $Y_2$ outputs can be simplified further if XOR/XNOR gates are allowed. The circuit diagram can be drawn up according to the equations. The Verilog HDL implementation of the counter and the mapping logic can be seen below.

```verilog
module task2(
    input wire clk,
    input wire rst,
    output reg [2:0] cnt
);
// 3-bit binary up-counter.
reg [2:0] cnt;
always @(posedge clk)
begin
    if (rst)
        cnt <= 3'd0;
    else
        cnt <= cnt + 3'd1;
end
```

```verilog
// Combinational mapping logic.
always @(*)
begin
    case (cnt)
        3'd0: y <= 3'd0;
        3'd1: y <= 3'd3;
        3'd2: y <= 3'd5;
        3'd3: y <= 3'd2;
        3'd4: y <= 3'd7;
        3'd5: y <= 3'd4;
        3'd6: y <= 3'd6;
        3'd7: y <= 3'd1;
    endcase
end
endmodule
```
The following Verilog test fixture can be used for testing the two designs. The timing diagram shows the expected results.

```verilog
module hw1_TF;

// Inputs
reg clk;
reg rst;

// Outputs
wire [2:0] task1_y, task2_y;

// Instantiate the Unit Under Test (UUT)
task1 uut1(.clk(clk), .rst(rst), .y(task1_y));
task2 uut2(.clk(clk), .rst(rst), .y(task2_y));

initial
begin
    // Initialize Inputs
    clk = 0;
    rst = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    rst = 1;
    #20 rst = 0;
end

// Clock generation.
always
    #10 clk <= ~clk;

endmodule
```