Clocked Sequence Generator

Anshuman Banik
Department of Instrumentation and Electronics
Jadavpur University

October 2024

1 Problem Statement

Design and implement a sequence generator that continuously produces the following repeating binary sequence: $\ldots, 1, 0, 1, 1, 0, 0, \ldots$. The sequence should repeat indefinitely and be generated at successive rising edges of a clock signal. The generator should have a clock input that triggers the output on the rising edge, producing a single bit representing the current value in the sequence.

The output sequence should follow this pattern: - At t_0: Output = 1 - At t_1: Output = 0 - At t_2: Output = 1 - At t_3: Output = 1 - At t_4: Output = 0 - At t_5: Output = 0 - At t_6: Output = 1 (and then repeats)

1.1 Solution

The following Verilog code implements a sequence generator using JK Flip-Flops. The generator produces the specified repeating binary sequence $\ldots, 1, 0, 1, 1, 0, 0, \ldots$ based on the rising edge of the clock signal.

```
// JK Flip-Flop Module Definition with Reset
module jk_ff (
    input J, K, clk, rst_n, // JK inputs, clock, and active-low reset
    output reg Q            // Flip-flop output
);

    // Initial state of the flip-flop
    initial begin
        Q = 1'b0;          // Initialize Q to 0
    end

    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            Q <= 1'b0;      // Reset the output to 0 when rst_n is low
        end else if (J & ~K) begin
```

```
            Q <= 1;         // Set the output to 1 if J is high and K is low
        end else if (~J & K) begin
            Q <= 0;         // Reset the output to 0 if J is low and K is high
        end else if (J & K) begin
            Q <= ~Q;        // Toggle the output if both J and K are high
        end
    end

endmodule


module seq_generator (
    input clk,          // Clock signal
    output wire q       // Output signal for the generated sequence
);

    wire [2:0] q_1;     // 3-bit state register as wires

    // Logic for the next state of din
    wire din;           // Next state input for the JK flip-flops
    assign din = ((~q_1[2]) & (~q_1[0])) | (q_1[2] & (~q_1[1]));

    // JK Flip-Flops instantiated
    wire q_ff0, q_ff1, q_ff2; // Intermediate flip-flop outputs
    jk_ff J2 (.J(din), .K(~din), .clk(clk), .Q(q_ff0));  // Flip-flop for q_1[0]
    jk_ff J1 (.J(q_ff0), .K(~q_ff0), .clk(clk), .Q(q_ff1)); // Flip-flop for q_1[1]
    jk_ff J0 (.J(q_ff1), .K(~q_ff1), .clk(clk), .Q(q_ff2)); // Flip-flop for q_1[2]

    // Assign flip-flop outputs to q_1
    assign q_1 = {q_ff2, q_ff1, q_ff0}; // Concatenate outputs to form 3-bit state

    // Output logic based on the state of q_1[2]
    assign q = q_1[2];  // Output follows the state of q_1[2]

endmodule
```

1.2 Code Explanation

The Verilog code implements a sequence generator using JK Flip-Flops to produce a repeating binary sequence of $\ldots, 1, 0, 1, 1, 0, 0, \ldots$. The code consists of two modules: the JK Flip-Flop module (‘jk$_ff$‘$) and the sequence generator module ($‘$seq_generator$‘).

1.2.1 JK Flip-Flop Module

The JK Flip-Flop module is defined as follows:

```
// JK Flip-Flop Module Definition with Reset
```

```
module jk_ff (
    input J, K, clk, rst_n, // JK inputs, clock, and active-low reset
    output reg Q            // Flip-flop output
);
```

- **Inputs:**
  - `J, K`: Control inputs that determine the operation of the flip-flop.
  - `clk`: The clock signal that triggers state changes on the rising edge.
  - `rst_n`: Active-low reset signal. When low, it resets the flip-flop.
- **Output:**
  - `Q`: The current state of the flip-flop output.

The initial state of the flip-flop is set to zero:

```
// Initial state of the flip-flop
initial begin
    Q = 1'b0;          // Initialize Q to 0
end
```

The main functionality is defined in the always block, which is triggered on the rising edge of the clock or when the reset signal is activated:

```
always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
        Q <= 1'b0;      // Reset the output to 0 when rst_n is low
    end else if (J & ~K) begin
        Q <= 1;         // Set the output to 1 if J is high and K is low
    end else if (~J & K) begin
        Q <= 0;         // Reset the output to 0 if J is low and K is high
    end else if (J & K) begin
        Q <= ~Q;        // Toggle the output if both J and K are high
    end
end
```

- **Reset Condition:** If the reset signal `rst_n` is low, the output `Q` is reset to 0.
- **Set Condition:** If `J` is high and `K` is low, the output is set to 1.
- **Reset Condition:** If `J` is low and `K` is high, the output is reset to 0.
- **Toggle Condition:** If both `J` and `K` are high, the output toggles its current state.

1.2.2 Sequence Generator Module

The sequence generator module is defined as follows:

```
module seq_generator (
    input clk,          // Clock signal
    output wire q       // Output signal for the generated sequence
);
```

- **Input:**
  - `clk`: The clock signal that triggers the sequence generation.
- **Output:**
  - `q`: The output signal that follows the specified sequence.

A 3-bit state register is declared as wires to keep track of the states of the JK Flip-Flops:

```
    wire [2:0] q_1;      // 3-bit state register as wires
```

The next state input for the JK Flip-Flops is determined by the following logic:

```
    wire din;            // Next state input for the JK flip-flops
    assign din = ((~q_1[2]) & (~q_1[0])) | (q_1[2] & (~q_1[1]));
```

Here, `din` is computed based on the current state of the flip-flops to ensure the desired sequence is generated.

The JK Flip-Flops are instantiated as follows:

```
    // JK Flip-Flops instantiated
    wire q_ff0, q_ff1, q_ff2; // Intermediate flip-flop outputs
    jk_ff J2 (.J(din), .K(~din), .clk(clk), .Q(q_ff0));  // Flip-flop for q_1[0]
    jk_ff J1 (.J(q_ff0), .K(~q_ff0), .clk(clk), .Q(q_ff1)); // Flip-flop for q_1[1]
    jk_ff J0 (.J(q_ff1), .K(~q_ff1), .clk(clk), .Q(q_ff2)); // Flip-flop for q_1[2]
```

Each JK Flip-Flop's input is connected according to the specified logic. The outputs of the flip-flops are concatenated to form the 3-bit state:

```
    // Assign flip-flop outputs to q_1
    assign q_1 = {q_ff2, q_ff1, q_ff0}; // Concatenate outputs to form 3-bit state
```

Finally, the output logic is determined based on the state of the most significant bit of the state register:

```
    // Output logic based on the state of q_1[2]
    assign q = q_1[2];  // Output follows the state of q_1[2]
```

In summary, the sequence generator utilizes JK Flip-Flops to produce the desired output sequence on each rising edge of the clock signal, effectively implementing the specified logic.

1.3 Testbench Stimulus

The following Verilog code implements a testbench for the sequence generator. This testbench instantiates the sequence generator module, generates the clock signal, and monitors the output.

```
module seq_generator_tb;

    reg clk;          // Clock signal
    wire q;           // Output of the sequence generator

    // Instantiate the sequence generator
    seq_generator dut (
        .clk(clk),
        .q(q)
    );

    // Clock generation (50 MHz clock, toggling every 10 ns)
    initial begin
        clk = 0;
        forever #10 clk = ~clk;
    end

    // Stimulus for the sequence generator
    initial begin
        // Allow the sequence generator to operate normally for a while
        #500; // Let it run for a bit

        // End simulation
        $stop;
    end

    // Monitor output signals during simulation
    initial begin
        $monitor("Time: %0t | clk: %b | q: %b",
                 $time, clk, q);
    end

endmodule
```

1.4 Schematic Diagram

The schematic diagram of the sequence generator circuit is shown in Figure 1.

1.5 Simulation Results

The simulation results for the sequence generator are depicted in Figure 2.

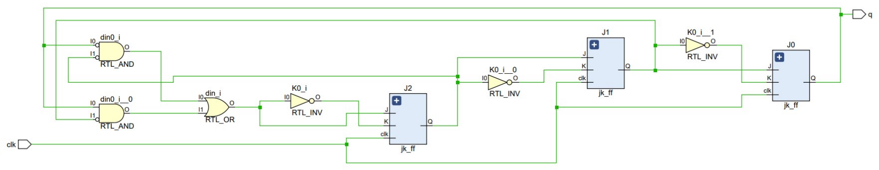

Figure 1: Schematic Diagram of the Sequence Generator Circuit

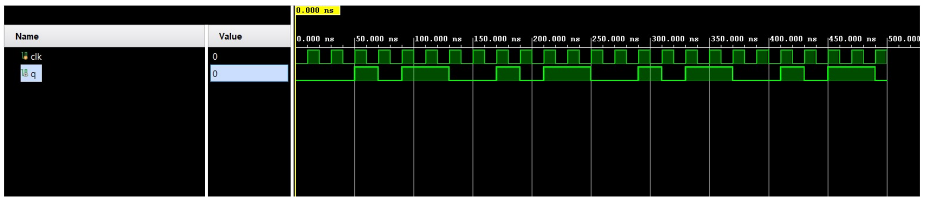

Figure 2: Simulation Results of the Sequence Generator