Habib University
Digital System Design - CE325

Final Exam - Report

Instructor: Dr. Arsalan Javed

Sadiqah Mushtaq - 07152

1 Architecture

The architecture consists of two FSMs. One for Client side and the other for Server side.

1.1 Moore Finite State Machine for Client

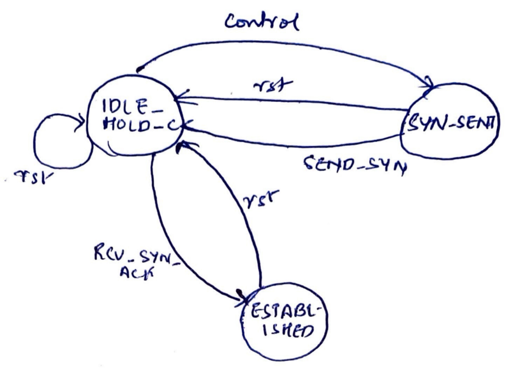

Figure 1: Client FSM

1.2 Moore Finite State Machine for Server

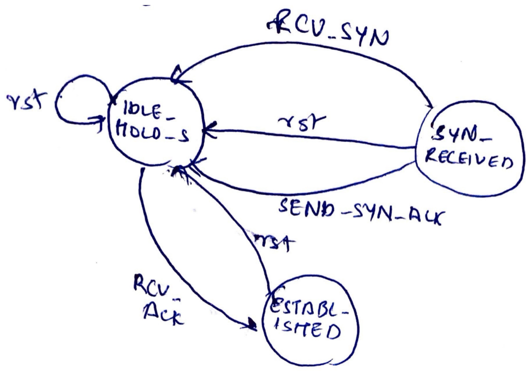

Figure 2: Server FSM

2 Verilog Code

2.1 Client Side

This module consists of inputs for reset (`rst`), control signal (`Control`), system clock (`clock`), and a signal indicating the reception of SYN-ACK packets (`RCV_SYN_ACK`). The outputs include signals for sending SYN (`SEND_SYN`) and ACK (`SEND_ACK`) packets.

The module maintains three states encoded as follows:

```
IDLE_HOLD_C = 2'b00;
SYN_SENT = 2'b01;
ESTABLISHED_C = 2'b10;
```

The state transitions are controlled by the current state (`state_reg_c`), control signals, and the reception of SYN-ACK packets. The only purpose of the control signal is to trigger the start of the handshake and is just an arbitrary signal. Upon reset, the module initializes to the `IDLE_HOLD_C` state and sets the output signals `SEND_SYN` and `SEND_ACK` to 0. Depending on the current state and control signals, the module transitions between states and sets the output signals accordingly. For instance, in the `SYN_SENT` state, it sets `SEND_SYN` to 1 when receiving the control signal and transitions back to `IDLE_HOLD_C` otherwise.

Overall, this module encapsulates the behavior of a client-side TCP connection initiation process, adhering to the TCP three-way handshake protocol.

```

[language=Veilog,
                      frame=single,
                      numbers=left,
                      numberstyle=\tiny,
                      breaklines=true,
                      basicstyle=\ttfamily\footnotesize,
                      commentstyle=\color{mydarkgreen},
                      keywordstyle=\color{blue}][language=Verilog]
`timescale 1ns / 1ps

module client_side(
    rst,
    clock,
    Control,
    RCV_SYN_ACK,
    SEND_SYN,
    SEND_ACK
);

input rst,
      Control,
      clock,
      RCV_SYN_ACK;

output reg SEND_SYN, SEND_ACK;

// States
parameter IDLE_HOLD_C = 2'b00;
parameter SYN_SENT = 2'b01;
parameter ESTABLISHED_C = 2'b10;

// State registers
reg [1:0] state_reg_c, state_next_c;

always @(posedge clock) begin
    if (rst) begin
        state_reg_c <= IDLE_HOLD_C;
        SEND_SYN <= 0;
        SEND_ACK <= 0;
    end
    else
        state_reg_c <= state_next_c;
end


always @(state_reg_c or Control or RCV_SYN_ACK or posedge clock) begin
    case(state_reg_c)
        IDLE_HOLD_C: begin
            if (rst)
                state_next_c <= IDLE_HOLD_C;
            else if (Control & RCV_SYN_ACK)
                state_next_c <= ESTABLISHED_C;
            else if (Control)
                state_next_c <= SYN_SENT;
            else
                state_next_c <= IDLE_HOLD_C;
        end

        SYN_SENT: begin
            if (rst)
                state_next_c <= IDLE_HOLD_C;
            else if (Control) begin
                state_next_c <= IDLE_HOLD_C;
                SEND_SYN <= 1;
            end
            else
                state_next_c <= IDLE_HOLD_C;
        end

        ESTABLISHED_C: begin
            if (rst)
                state_next_c <= IDLE_HOLD_C;
            else if (Control & RCV_SYN_ACK) begin
                SEND_ACK <= 1;
                state_next_c <= ESTABLISHED_C;
            end
            else
                state_next_c <= IDLE_HOLD_C;
        end

        default:
            state_next_c <= IDLE_HOLD_C;
    endcase
end

endmodule
```

2.2 Server Side

It includes inputs for reset (`rst`), clock (`clock`), and signals indicating whether the server has received a SYN packet (`RCV_SYN`) or an ACK packet (`RCV_ACK`). The output `SEND_SYN_ACK` signifies the transmission of a SYN-ACK packet from the server to the client.

The FSM states are encoded as follows:

```
IDLE_HOLD_S = 2'b00;
SYN_RECEIVED = 2'b01;
ESTABLISHED_S = 2'b10;
```

In the `IDLE_HOLD_S` state, the server waits for incoming SYN packets. Upon receiving a SYN packet (`RCV_SYN`) without an ACK, it transitions to the `SYN_RECEIVED` state, indicating acknowledgment of the SYN. If the server receives both SYN and ACK signals (`RCV_SYN & RCV_ACK`), it transitions to the `ESTABLISHED_S` state, signifying the establishment of a connection.

The module continuously updates its state based on inputs and clock edges. When reset (`rst`) is activated, the state returns to `IDLE_HOLD_S`, ensuring proper initialization.

Overall, this module manages the server-side behavior of acknowledging SYN packets, sending SYN-ACK packets, and transitioning to the established state upon receiving appropriate signals, adhering to the TCP three-way handshake protocol.

```verilog
`timescale 1ns / 1ps

module server_side (
    rst,
    clock,
    RCV_SYN,
    RCV_ACK,
    SEND_SYN_ACK
);

input rst, clock, RCV_SYN, RCV_ACK;

output SEND_SYN_ACK;

reg SEND_SYN_ACK;

// FSM States
parameter IDLE_HOLD_S = 2'b00;
parameter SYN_RECEIVED = 2'b01;
parameter ESTABLISHED_S = 2'b10;

// State registers
reg [1:0] state_reg_s, state_next_s;

always @(posedge clock) begin
    if (rst)
        state_reg_s <= IDLE_HOLD_S;
    else
        state_reg_s <= state_next_s;
end

always @(state_reg_s or RCV_SYN or RCV_ACK or posedge clock) begin
    case(state_reg_s)
        IDLE_HOLD_S: begin
            if (rst)
                state_next_s <= IDLE_HOLD_S;
            else if (RCV_SYN & ~RCV_ACK)
                state_next_s <= SYN_RECEIVED;
            else if (RCV_SYN & RCV_ACK)
                state_next_s <= ESTABLISHED_S;
            else
                state_next_s <= IDLE_HOLD_S;
        end

        SYN_RECEIVED: begin
            if (rst)
                state_next_s <= IDLE_HOLD_S;
            else if (RCV_SYN & ~RCV_ACK) begin
                state_next_s <= IDLE_HOLD_S;
                SEND_SYN_ACK = 1;
            end
            else
                state_next_s <= IDLE_HOLD_S;
        end

        ESTABLISHED_S: begin
            if (rst)
                state_next_s <= IDLE_HOLD_S;
            else if (RCV_SYN & RCV_ACK)
                state_next_s <= ESTABLISHED_S;
            else
                state_next_s <= ESTABLISHED_S;
        end

        default:
            state_next_s <= IDLE_HOLD_S;
    endcase
end

endmodule
```

2.3 TCP Module

The Verilog module named TCP integrates both client and server sides of the TCP three-way handshake protocol. It includes inputs for reset (rst), clock (clock), and control signals (Control).

The module instantiates separate modules for the client side (client_side) and server side (server_side). The client side communicates with the server through signals for sending SYN and ACK packets (SYN and ACK), while the server side communicates with the client through a signal indicating the reception of SYN-ACK packets (SYN_ACK).

```
`timescale 1ns / 1ps


module TCP (
    rst,
    clock,
    Control
);

input rst,
      clock,
      Control;

wire SYN, ACK, SYN_ACK;

client_side client_inst1 (
    .rst(rst),
    .clock(clock),
    .Control(Control),
    .RCV_SYN_ACK(SYN_ACK),
    .SEND_SYN(SYN),
    .SEND_ACK(ACK)
);

server_side server_inst1 (
    .rst(rst),
    .clock(clock),
    .RCV_SYN(SYN),
    .RCV_ACK(ACK),
    .SEND_SYN_ACK(SYN_ACK)
);

endmodule
```

2.4 Testbench

```
[language=Veilog,
                    frame=single,
                    numbers=left,
                    numberstyle=\tiny,
                    breaklines=true,
                    basicstyle=\ttfamily\footnotesize,
                    commentstyle=\color{mydarkgreen},
                    keywordstyle=\color{blue}][language=Verilog]
`timescale 1ns / 1ps

module TCP_tb;

// Parameters
parameter PERIOD = 10; // Clock period in ns

// Signals
reg rst = 0;
reg clock = 0;
reg Control = 0;

// Instantiate the module under test
TCP dut (
    .rst(rst),
    .clock(clock),
    .Control(Control)
);

// Clock generation
always #((PERIOD/2)) clock = ~clock;

// Stimulus
initial begin
    // Reset sequence
    #20 rst = 1;
    #20 rst = 0;

    // Control sequence
    #10 Control = 1; // Example: Setting control signal to 1

    // Simulation end
    #100 $finish;
    end

endmodule
```

3 Simulation

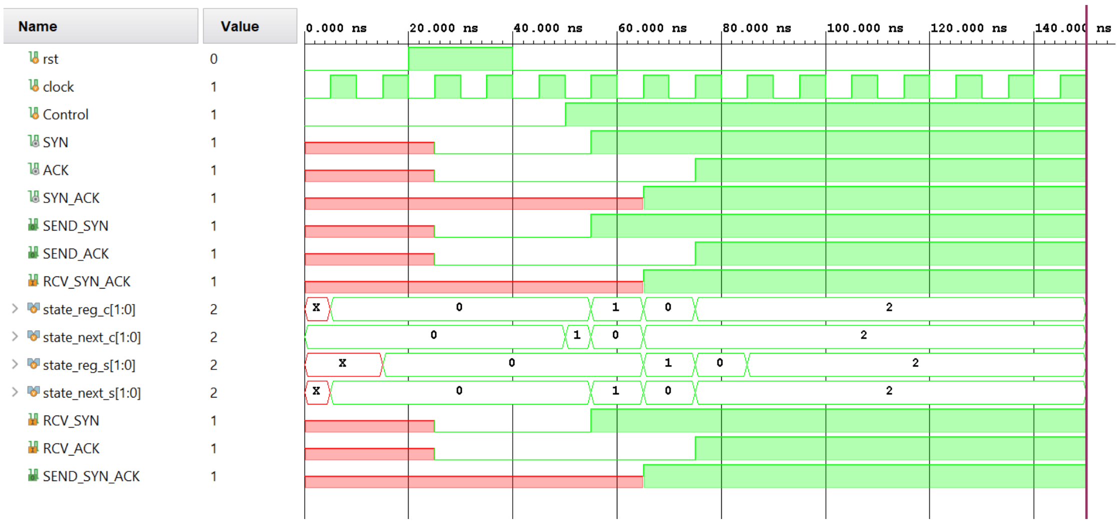

Figure 3: Simulation of TCP Three Way Handshake using FSMs

In the above simulation, the progression of the TCP three-way handshake protocol is evident. Initially, when the client enters the SYN_SENT state, it triggers the transmission of the SEND_SYN signal. This action signifies the initiation of the handshake process. Upon receiving this SYN signal, the server transitions to the SYN_RECEIVED state. In response, the server send the SYN_ACK signal to client, indicating its acknowledgment of the client's SYN packet.

The client advances to the ESTABLISHED_C state, signifying the successful establishment of the connection from its perspective. As part of the final step of the handshake, the client sends an ACK signal back to the server. Upon receiving this acknowledgment, the server progresses to the ESTABLISHED_S state, indicating the completion of the handshake process and the establishment of a reliable communication channel between the client and server.

This sequence illustrates the coordinated exchange of SYN, SYN-ACK, and ACK packets between the client and server, culminating in the successful establishment of the TCP connection.