1 FSM Description

My state machine is a basic implementation of the TCP protocol in a network router. It includes 1) the SYN, SYNACK, ACK three-way handshake to establish the connection, 2) an evaluation of the application layer protocol from the port number in the TCP packet queued in the buffer, and 3) the FIN, FINACK, ACK connection teardown, and 4) a timeout counter in the Read stage that will transition to the Closed state if a valid Port number is not provided within ten clock cycles.

The scalable aspect of the FSM is the ability to add extra application-layer protocols based on the port number input, as well as the potential to include additional timeout counters in states that don't receive their expected input.

2 Testbench Description

One of my testbenches was an example of a typical path through the protocol of a TCP connection – starting from the Closed state and walking all states of the FSM before returning to the Closed state. This was to ensure the outputs were correctly being triggered in their respective states, and to ensure each state could be reached and exited.

Additionally, I checked to make sure the timeout counter would eventually transition us from the Read state to the Closed state if an unrecognized Port number was received. More detail about this testbench is in the comments of the `FSM_TB.sv` file.

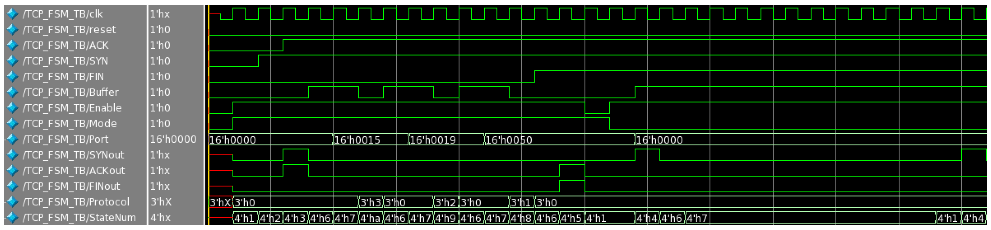


Figure 1: Waveform of Example User Path Simulation

The other testbench was an exhaustive testbench that iterated through all possible combinations of inputs. The only input that wasn't iterated through all combination was the port number which can be any 16-bit number – instead I provided the port number 0 which doesn't correspond to a protocol state and the port number 21 which corresponds to the FTP protocol state.

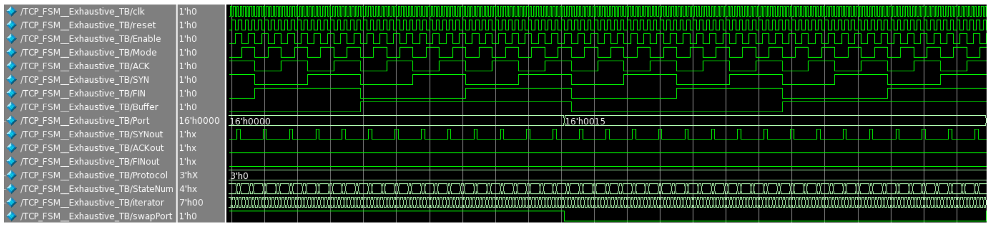


Figure 2: Waveform of Exhaustive Simulation

The problem with this testbench is that the TCP FSM models a protocol that requires inputs in a specific order and the exhaustive testbench did not provide inputs in that order, so the FSM only progressed through two states. However, the point of this testbench was to ensure that the outputs are logically driven for all combinations of inputs.

3 Block Diagram of FSM

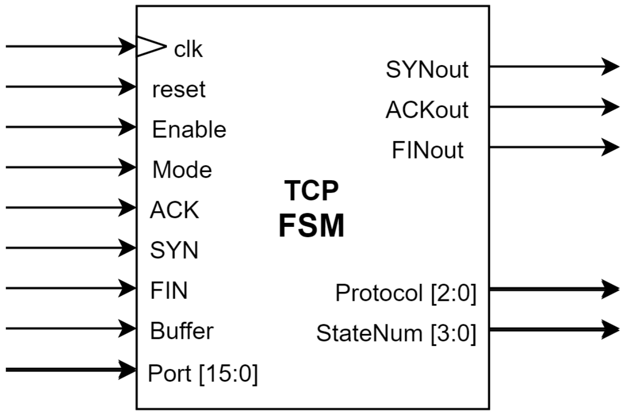


Figure 3: Inputs and Outputs of TCP FSM Module

4 Block Diagram of FSM and Test Bench

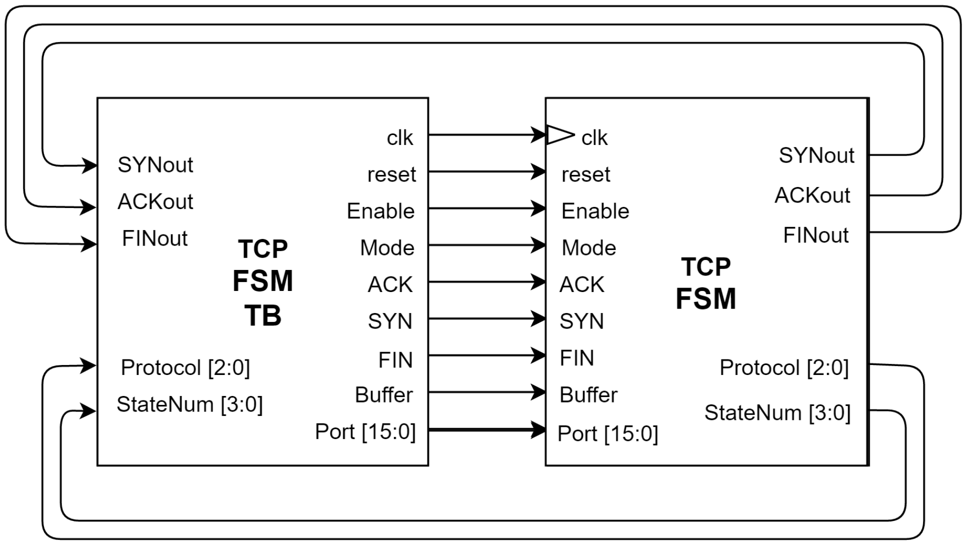


Figure 4: Connection of TCP FSM Module and Testbench

5 State Transition Diagram

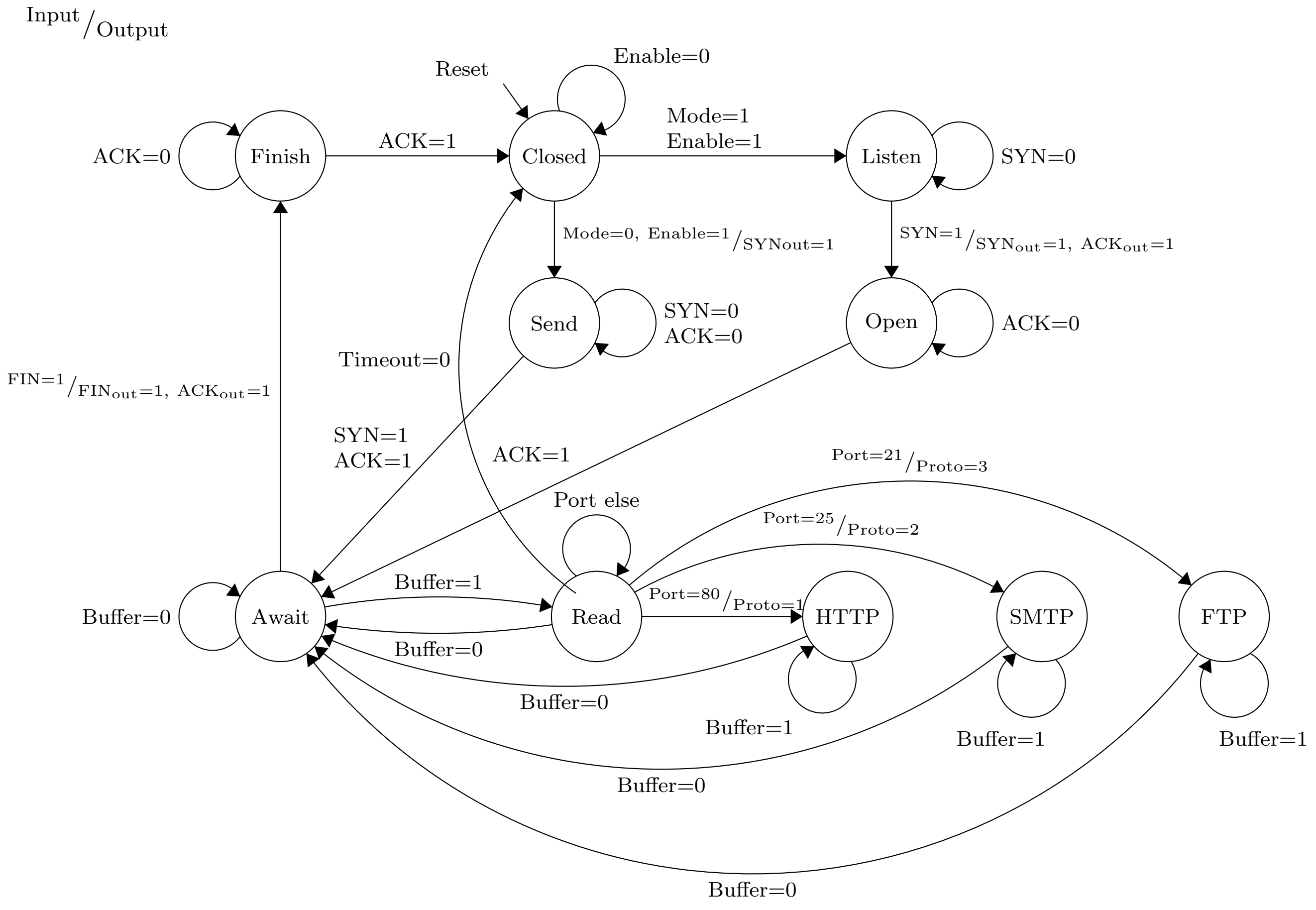


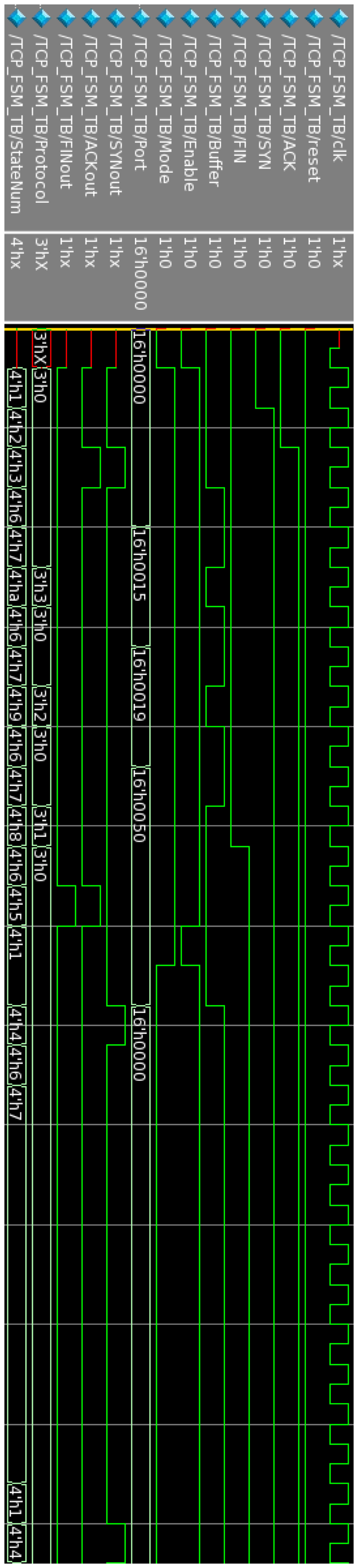
Figure 5: State Transition Diagram of TCP FSM

States	**Inputs**	**Outputs**
Closed	Enable	ACKout
Send	Mode	SYNout
Listen	ACK	FINout
Open	SYN	Protocol
Await	FIN	StateNum
Read	Buffer	
HTTP	Port	
SMTP		
FTP		
Finish		

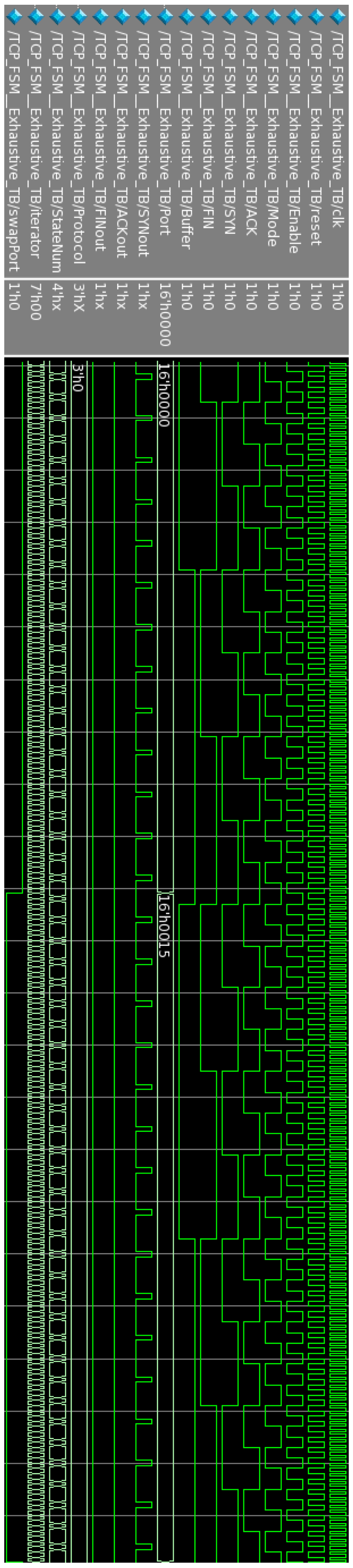
*StateNum is just for debugging, so has not been included in the state transition diagram

6 Simulation Waveforms

6.1 Example User Path Waveform



6.2 Exhaustive Waveform



7 SystemVerilog Code

7.1 TPC FSM

```
// Student Name:   Charles Clayton
// Student Number: 21518139

/* Function: A basic implementation of the TCP protocol in a network router. It includes
    1)  the SYN, SYNACK, ACK three-way handshake to establish the connection,
    2)  an evaluation of the application layer protocol from
        the port number in the TCP packet queued in the buffer, and
    3)  the FIN, FINACK, ACK connection teardown. The scalable aspect of the FSM is the
    ability to add extra application-layer protocols based on the port number input,
    as well as the potential to include timeout counters in states that don't receive the expected input.
*/

module TCP_FSM( // Module inputs
                input logic clk,
                input logic reset,
                input logic ACK,
                input logic SYN,
                input logic FIN,
                input logic Buffer,
                input logic Enable,
                input logic Mode,
                input logic [15:0] Port,

                // Module outputs
                output logic SYNout,
                output logic ACKout,
                output logic FINout,
                output logic [2:0] Protocol,
                output logic [3:0] StateNum);


// Define the state types to be used
typedef enum logic [3:0] {
    State_Closed,
    State_Listen,
    State_Open,
    State_Send,
    State_Finish,
    State_Await,
    State_Read,
    State_HTTP,
    State_SMTP,
    State_FTP
} statetype;

statetype state, nextstate;

integer timeoutCounter;

always_ff @(posedge clk)
    // When reset is high, do a synchronous reset to the Closed state
    if(reset) state <= State_Closed;

    // if in the Read state, decrement the timeout counter
    // if the counter reaches zero, go to the Closed state
        // otherwise go to the next state and keep the timer high
      else if(state == State_Read) begin
         timeoutCounter <= timeoutCounter - 1;

         if(timeoutCounter == 0) state <= State_Closed;
         else                    state <= nextstate;
    end

    // if not in the Await state reset the timeout counter to max
    // and update our state
    else begin
         state <= nextstate;
         timeoutCounter <= 10;
    end

// Combinational logic to update our state based on inputs
always_comb
    case(state)

               // You can only leave the Closed state when the "Enable" is set.
               // If "Mode" is set, you go to the Listen state.
               // If not, you go to the Send state.
        State_Closed:
            if     (Enable == 1 & Mode == 1) nextstate = State_Listen;
            else if(Enable == 1 & Mode == 0) nextstate = State_Send;
            else                             nextstate = State_Closed;

               // This is the first stage of the TCP 3-way handshake
               // The router stays in the Listen state until it gets
               // a SYN signal from an external router wishing to set up a connection
        State_Listen:
            if(SYN == 1) nextstate = State_Open;
            else         nextstate = State_Listen;

               // This is the second stage of the TCP 3-way handshake
               // The router responds to the SYN signal with a SYNACK
               // meaning both SYNout and ACKout are set.
               // The router waits in the Open state until it recieves an ACK response
        State_Open:
            if(ACK == 1) nextstate = State_Await;
            else         nextstate = State_Open;

               // This is the third stage of the TPC 3-way handshake
               // Once the Router recieves the ACK response, it waits for
               // requests sent by the external router.

               // If the Buffer is set, that means a packet is queued and
               // is waiting to be procossed, so go to the Read state

               // If a FIN signal is set, that means the external router
               // wants to tear down the connection, so go to the Finish state
        State_Await:
            if(Buffer == 1)   nextstate = State_Read;
            else if(FIN == 1) nextstate = State_Finish;
            else              nextstate = State_Await;

        // In the Read state, if the router's Buffer
        // goes low, then it returns to the Await state for new data.
        // Otherwise it will go to one of the FTP, SMTP, and HTTP states
        // depending on the port number it recieves.

        // If the Port number corresponds to an unknown protocol, it will
        // wait here until it recieves a valid protocol.
        // Note:     In the future to scale up this project, a Timer will be
        //                placed in this state to Timeout if a valid protocol is not
        //               recieved in a certain amount of time.
        State_Read:
            if(Buffer == 0)     nextstate = State_Await;
            else if(Port == 21) nextstate = State_FTP;
            else if(Port == 25) nextstate = State_SMTP;
            else if(Port == 80) nextstate = State_HTTP;
            else                nextstate = State_Read;

        // Stay in your respective protocol state until the buffer is cleared
        // Note:     Again, Timeouts can be placed here if the packet cannot be processed
        State_HTTP:
            if(Buffer == 0) nextstate = State_Await;
            else            nextstate = State_HTTP;

        State_SMTP:
            if(Buffer == 0) nextstate = State_Await;
            else            nextstate = State_SMTP;

        State_FTP:
            if(Buffer == 0) nextstate = State_Await;
            else            nextstate = State_FTP;

        // When a FIN signal is recieved, you will come to the Finish state
        // This state triggers an FINout and ACKout signal to be sent.
        // Stay in this state until the external router responds with an ACK
        // then close the connection and go to the Closed state
        State_Finish:
            if(ACK == 1) nextstate = State_Closed;
            else         nextstate = State_Finish;

        // This is an alternative route to the Await state that bypasses the
        // TCP 3-way handshake if we are in sender mode (Mode 0).

        // In sender mode, we are the router that establishes a connection,
        // so we send a SYNout signal and wait until we recieve a SYNACK
        // from the external router. Then we go to the Await state.
        State_Send:
            if(SYN == 1 & ACK == 1) nextstate = State_Await;
            else                    nextstate = State_Send;

        default:
            nextstate = State_Closed;
    endcase


// Output signals
// Note: This is purely a Moore machine

// output SYNACK when in State_Open to accept the connection request
// output FINACK when in State_Finish to accept the teardown request
// output SYN when in State_Send to request a connection
assign SYNout = (state == State_Open) | (state == State_Send);
assign ACKout = (state == State_Open) | (state == State_Finish);
assign FINout =                                                (state == State_Finish);

// output a value to signal which protocol is being used
assign Protocol = ( state == State_HTTP ) ? 1 :
                  ( state == State_SMTP ) ? 2 :
                  ( state == State_FTP )  ? 3 : 0;

// output state number (only for debugging/waveform purposes)
assign StateNum =   ( State_Closed  == state ) ? 1  :
                    ( State_Listen  == state ) ? 2  :
                    ( State_Open    == state ) ? 3  :
                    ( State_Send    == state ) ? 4  :
                    ( State_Finish  == state ) ? 5  :
                    ( State_Await   == state ) ? 6  :
                    ( State_Read    == state ) ? 7  :
                    ( State_HTTP    == state ) ? 8  :
                    ( State_SMTP    == state ) ? 9  :
                    ( State_FTP     == state ) ? 10 : 0;

endmodule
```

7.2 TPC FSM Userpath Testbench

```
// Student Name   : Charles Clayton
// Student Number : 21518139

/* Function: This testbench is an example of a typical path through the
    protocol of a TCP router -- starting from the Closed state and walking
    through almost all states of the FSM before returning to the Closed state.
    This is to ensure the outputs were correctly being triggered in
    their respective states, and to ensure each state could be reached and exited.
*/

‘timescale 1ns/1ps  // unit step = 1ns and 1ps is the resolution

module TCP_FSM_TB;
    // inputs of FSM
    logic clk;
    logic reset;
    logic ACK;
    logic SYN;
    logic FIN;
    logic Buffer;
    logic Enable;
    logic Mode;
    logic [15:0] Port;

    // outputs of FSM
    logic SYNout;
    logic ACKout;
    logic FINout;
    logic [2:0] Protocol;
    logic [3:0] StateNum;

    // connecting logit to FSM module
    TCP_FSM U0(
        .clk(clk),
        .reset(reset),
        .ACK(ACK),
        .SYN(SYN),
        .FIN(FIN),
        .Buffer(Buffer),
        .Enable(Enable),
        .Mode(Mode),
        .Port(Port),
        .SYNout(SYNout),
        .ACKout(ACKout),
        .FINout(FINout),
        .Protocol(Protocol),
        .StateNum(StateNum)
    );

    // This is a fixed sequence of inputs.
    // This process is an example of a typical user path
    // that a TPC router may walk through, this particular
    // example will send us through every single state in the FSM
     initial begin
        // initialize inputs other than reset to zero
        reset=1; Enable=0; Mode=0; SYN=0; ACK=0; Port=0; Buffer=0; FIN=0;

        // disable the reset
        reset=0;          #200;

                // enable the router in mode 1 (listen mode) -- go to Listen state
        Enable=1; Mode=1; #200;

        // provide a SYN input to request connection -- go to Open state
        SYN=1;            #200;
        // provide an ACK response to the SYNACK -- go to Await state
        ACK=1;            #200;

        // provide a Buffer signal to indicate queued packet -- go to Read state
        Buffer=1;         #200;

        // provide a Port number 21 to indidate FTP protocol -- go to FTP state
        Port=21;          #200;

        // disable Buffer signal to indicate packet read -- go back to Await state
        Buffer=0;         #200;

        // reenable Buffer to indidate another packet -- go to Read state
        Buffer=1;         #200;

         // provide port Number 25 to indicate SMTP protocol -- go to SMTP state
        Port=25;          #200;

        // clear buffer -- go back to Await state
        Buffer=0;         #200;

        // reenable Buffer to indidate another packet -- go to Read state
        Buffer=1;         #200;

         // provide port Number 80 to indicate HTTP protocol -- go to HTTP state
        Port=80;          #200;

        // clear buffer -- back to Await state
        Buffer=0;         #200;

        // provide FIN request to close connection -- go to Finish state
        FIN=1;            #200;

        // provide ACK to confirm close -- go to Closed state
        ACK=1;            #200;

        // disable Enable to stay in Closed state
        Enable=0;         #200;

        // re-enable in Mode 0 (send mode) -- go to Send state
        Enable=1; Mode=0; #200;

        // provide SYNACK to accept connection, this will send us to Await
        // but a high buffer will send us straight through to the Read state
        // however, Port=0 is an unrecognized port number -- stay in Read state
        SYN=1; ACK=1; Buffer=1; Port=0; #200;

        // wait for Read state to timeout and send us back to Closed state
        #5000;

        // reset the router and turn off enable
        reset=1; Enable=0; #200;
    end


    // Defining clock to cycle
    always
        begin
            #100 clk = 0;
            #100 clk = 1;
        end

endmodule
```

7.3 TPC FSM Exhaustive Testbench

```
// Student Name   : Charles Clayton
// Student Number : 21518139
/* Function: This testbench is an exhaustive testbench that iterates through
    all possible combinations of inputs. The only input that isn't iterated
    through all combination is the port number which can be any 16-bit number
    -- instead it alternates between the port number 0 which doesn't correspond to a protocol
    state and the port number 21 which corresponds to the FTP protocol state.
    All other protocol states are identical. */

`timescale 1ns/1ps  // unit step = 1ns and 1ps is the resolution

module TCP_FSM__Exhaustive_TB;
    // inputs of FSM
    logic clk;
    logic reset;
    logic ACK;
    logic SYN;
    logic FIN;
    logic Buffer;
    logic Enable;
    logic Mode;
    logic [15:0] Port;

    // outputs of FSM
    logic SYNout;
    logic ACKout;
    logic FINout;
    logic [2:0] Protocol;
    logic [3:0] StateNum;

    // iterator to provide inputs with all input values
    logic [6:0] iterator = 0;

    // Connecting logic to FSM module
    TCP_FSM U0(
        .clk(clk),
        .reset(reset),
        .ACK(ACK),
        .SYN(SYN),
        .FIN(FIN),
        .Buffer(Buffer),
        .Enable(Enable),
        .Mode(Mode),
        .Port(Port),
        .SYNout(SYNout),
        .ACKout(ACKout),
        .FINout(FINout),
        .Protocol(Protocol),
        .StateNum(StateNum)
    );

    // Set all initial values to zero
    initial begin
       reset  = 0;
       Enable = 0;
       Mode   = 0;
       ACK    = 0;
       SYN    = 0;
       FIN    = 0;
       Buffer = 0;
       Port   = 0;
    end

    logic swapPort = 0;
    always
        begin
         // alternate clock with low and high signals
         clk = 0; #100;
         clk = 1; #100;

         // iterate the iterator bit through all possible values
         // to give inputs all possible combinations of inputs
         reset  = iterator[0];
         Enable = iterator[1];
         Mode   = iterator[2];
         ACK    = iterator[3];
         SYN    = iterator[4];
         FIN    = iterator[5];
         Buffer = iterator[6];

         iterator += 1;

         // when iterator overflows try a different port number for next round
         if(iterator == 7'b111_1111) swapPort = ~swapPort;

         if(swapPort) Port=0;
         else         Port=21;

       end

endmodule
```