# ELEC 402 – Project 6
# Cell Library Layout

Charles Clayton
#21518139

December 1, 2016

# 1 FSM Description

## 1.1 Function

This FSM is a basic implementation of the TCP protocol in a network router. It includes the following functionality:

1. A 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
3. A FIN, FINACK, ACK connection teardown
4. A timeout counter in the Read stage that will reset the FSM to the Closed state from the Read state if a recognized input is not provided within ten clock cycles

## 1.2 Inputs and Outputs

Table 1: TCP FSM Inputs

| Inputs | Description |
|---|---|
| **Reset** | When high, do a synchronous reset to the Closed state |
| **Enable** | Can only leave the Closed state when the Enable is set |
| **Mode** | Specifies whether the FSM goes to the Listen state (high) or the Send state (low) |
| **SYN** | When the SYN bit is high and ACK is low, an external router is requesting to set up a connection. When SYN and ACK are both high, an external router has agreed to set up the connection we have requested. |
| **FIN** | When the FIN bit is high and ACK is high, an external router is requesting to tear down the connection |
| **ACK** | The ACK bit is high when an external router confirms its request to open/teardown a connection, or when an external router agrees with our request to open a connection. |
| **Buffer** | If Buffer is set, that means a packet is queued and is waiting to be processed, so go to the Read state. When the buffer goes low, it means the packets have been cleared and to go to the Await state. |
| **Port** | The port number is an integer that corresponds to the application-layer protocol. This FSM has states for the protocols FTP (port 21), SMTP (port 25), and HTTP (port 80). |

Table 2: TCP FSM Outputs

| Outputs | Description |
|---|---|
| **SYNout** | High when in the Open state to accept connection requests from external routers (SYNACK) and high when in the Send state to initiate a connection from an external router (SYN) |
| **FINout** | High when in the Finish state to accept the teardown request (FINACK) |
| **ACKout** | High when in the Finish state to accept the teardown request from external routers (FINACK), and high when in the Open state to initiate a connection from an external router (SYNACK) |
| **Protocol** | Outs an integer to indicate which protocol is being used, and 0 if the protocol is unrecognized. |
| **StateNum** | Outputs an integer corresponding to the state number – this is only for debugging purposes |

## 1.3 Testing Procedure

### 1.3.1 Exhaustive Inputs Test

This testbench iterates through all the possible combinations of inputs. This test can show us if there is ever any contention in the output. However, since my machine is purely a Moore machine and since the inputs need to occur in a specific order to walk through all states, the exhaustive input test is not particularly informative. To walk through all states, I use the User Path Test outlined in the next section.

### 1.3.2 User Path Test

The user path testbench is a fixed sequence of inputs. The 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.

A full walkthrough of the process is outlined in Table 3.

Table 3: User Path Test Sequence

| **Description** | **State Transition** | **Input** |
|---|---|---|
| initialize inputs other than reset to zero | stay in Closed state | reset=1; Enable=0; Mode=0; SYN=0; ACK=0; Port=0; Buffer=0; FIN=0; |
| disable the reset | stay in Closed state | reset=0; |
| enable the router in mode 1 (listen mode) | go to Listen state | Enable=1; Mode=1; |
| provide a SYN input to request connection | go to Open state | SYN=1; |
| provide an ACK response to the SYNACK | go to Await state | ACK=1; |
| provide a Buffer signal to indicate queued packet | go to Read state | Buffer=1; |
| provide a Port number 21 to indidate FTP protocol | go to FTP state | Port=21; |
| disable Buffer signal to indicate packet read | go back to Await state | Buffer=0; |
| reenable Buffer to indidate another packet | go to Read state | Buffer=1; |
| provide port Number 25 to indicate SMTP protocol | go to SMTP state | Port=25; |
| clear buffer | go back to Await state | Buffer=0; |
| reenable Buffer to indidate another packet | go to Read state | Buffer=1; |
| provide port Number 80 to indicate HTTP protocol | go to HTTP state | Port=80; |
| clear buffer | back to Await state | Buffer=0; |
| provide FIN request to close connection | go to Finish state | FIN=1; |
| provide ACK to confirm close | go to Closed state | ACK=1; |
| disable Enable to stay in Closed state<br>re-enable in Mode 0 (send mode) | go to Send state | Enable=0;<br>Enable=1; Mode=0; |
| 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; |
| wait for Read state to timeout | go to Closed state | |
| reset the router and turn off enable | stay in Closed state | reset=1; Enable=0; |

## 2 Cadence Layout

The area of the Encounter layout was $26.22 \times 41.1 = 1077.642$

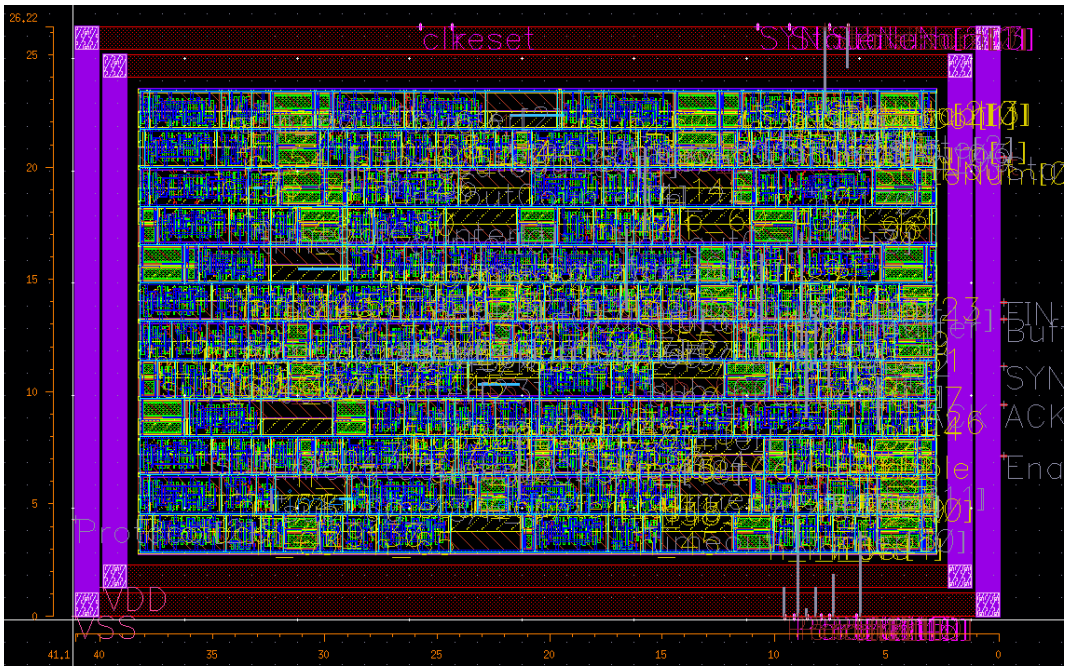


Figure 1: Virtuoso Layout

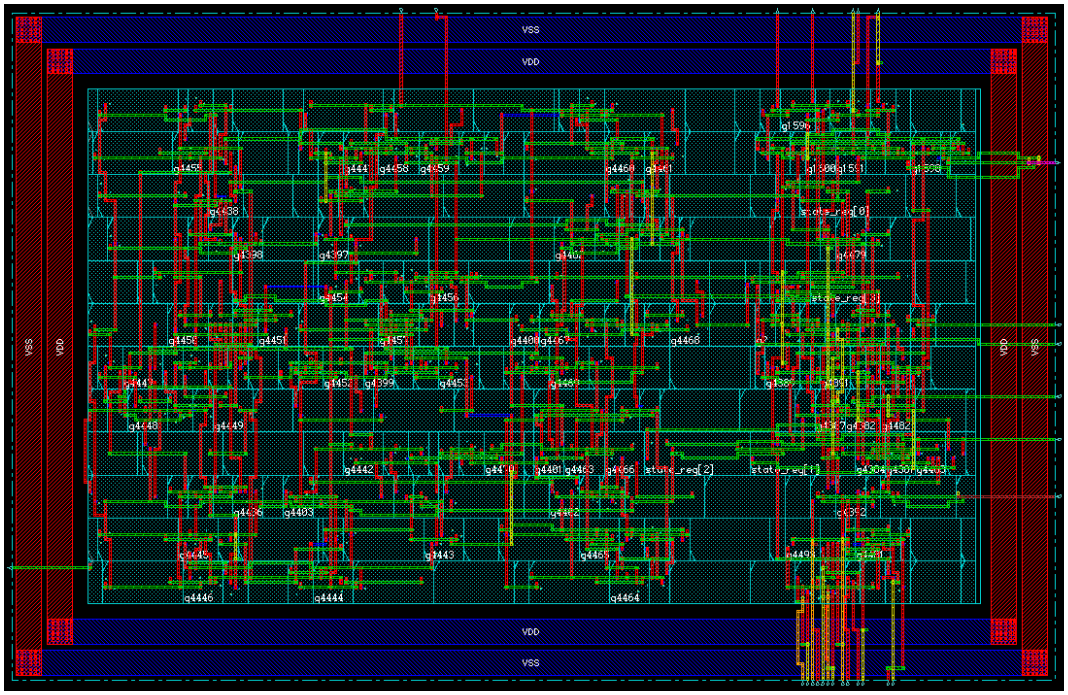


Figure 2: Encounter Layout

# 3 Testbench Waveforms

When running my testbench files with the SDF generated by Encounter, the inputs signals were not included in the waveform, however the testbench supplied the same inputs.

## 3.1 User Path

The only difference between the output waveforms of the User Path testbench when run with the original SDF and the SDF generated by encouter is a slight glitch in the ACKout output.

With the Encounter-generated SDF, ACKout momentarily goes high in the state transition between the Closed state (4) and the Send state (1), as indicated by the red circle in Figure 3 and 4. Other than that, both waveforms were the same.

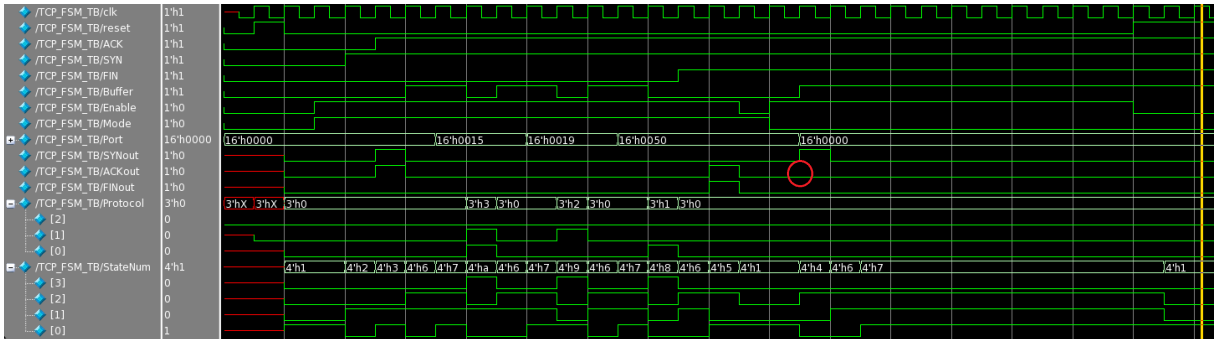


Figure 3: Mapped SystemVerilog User Path TB with Original SDF

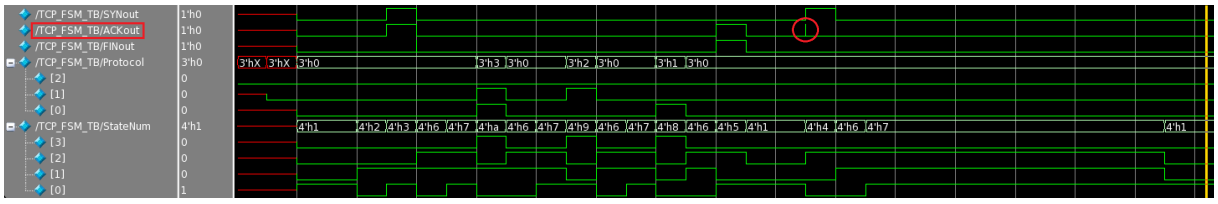


Figure 4: Mapped SystemVerilog User Path TB with Encounter-Generated SDF

## 3.2 Exhaustive Inputs

Similarly with the User Path testbench, the only difference between the waveforms is a momentary high signal of the ACKout output during the state transition from the Closed state to the Send state.

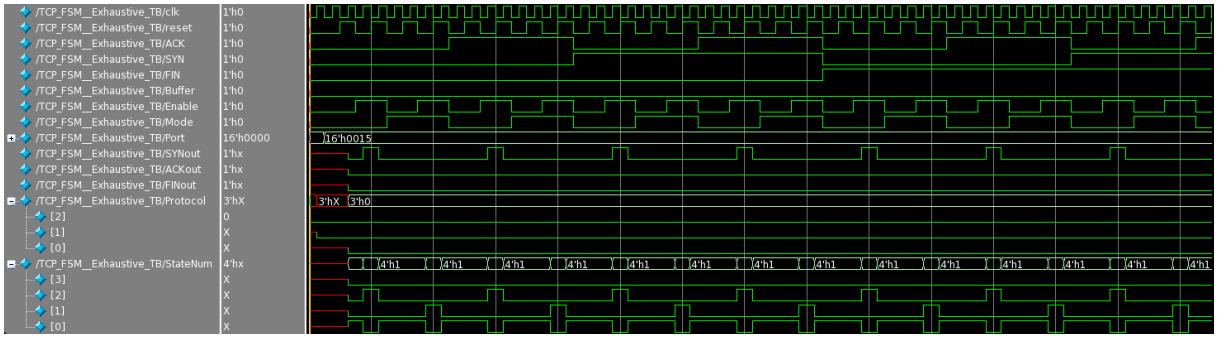


Figure 5: Mapped SystemVerilog Exhaustive TB with Original SDF

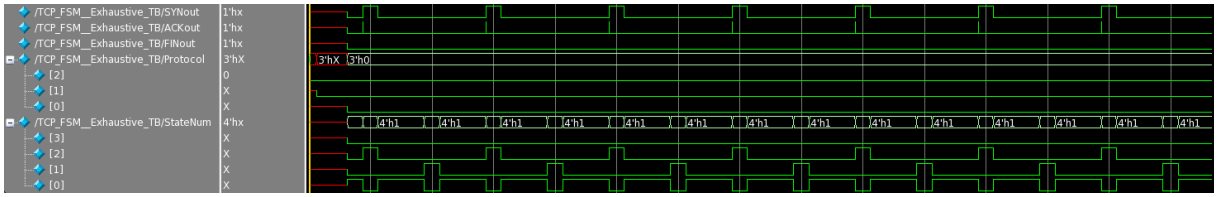


Figure 6: Mapped SystemVerilog Exhaustive TB with Encounter-Generated SDF

# 4 Test Files

## 4.1 FSM User Path TB.sv

```
// 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 1ps/1ps

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 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)
);

// 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;

    // toggle the reset
    reset=0;          #500;
    reset=1;          #500; //
    reset=0;          #500; //

    // enable the router in mode 1 (listen mode) -- go to Listen state
    Enable=1; Mode=1; #500;

    // provide a SYN input to request connection -- go to Open state
    SYN=1;            #500;

    // provide an ACK response to the SYNACK -- go to Await state
    ACK=1;            #500;

    // provide a Buffer signal to indicate queued packet -- go to Read state
    Buffer=1;         #500;

    // provide a Port number 21 to indidate FTP protocol -- go to FTP state
    Port=21;          #500;

    // disable Buffer signal to indicate packet read -- go back to Await state
    Buffer=0;         #500;

    // reenable Buffer to indidate another packet -- go to Read state
    Buffer=1;         #500;

     // provide port Number 25 to indicate SMTP protocol -- go to SMTP state
    Port=25;          #500;

    // clear buffer -- go back to Await state
    Buffer=0;         #500;

    // reenable Buffer to indidate another packet -- go to Read state
    Buffer=1;         #500;

     // provide port Number 80 to indicate HTTP protocol -- go to HTTP state
    Port=80;          #500;

    // clear buffer -- back to Await state
    Buffer=0;         #500;
```

```
        // provide FIN request to close connection -- go to Finish state
        FIN=1;            #500;

        // provide ACK to confirm close -- go to Closed state
        ACK=1;            #500;

        // disable Enable to stay in Closed state
        Enable=0;         #500;

        // re-enable in Mode 0 (send mode) -- go to Send state
        Enable=1; Mode=0; #500;

        // 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; #500;

        // 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; #500;
    end


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

endmodule
```

## 4.2 FSM Exhaustive TB.sv

```
// 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 1ps/1ps

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; #250;
         clk = 1; #250;

         // 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
```