Experiment 2 – 7-Segemnt Decoder in Verilog

```
module hex_to_7seg   (out,in);

   output   [6:0] out;     // low-active out
   input [3:0] in;       // 4-bit binary inp

   reg      [6:0] out;   // make out a varia

   always @ (*)
     case (in)
       4'h0: out = 7'b1000000;
       4'h1: out = 7'b1111001;     // --0 --
       4'h2: out = 7'b0100100;     // |     |
       4'h3: out = 7'b0110000;     // 5     1
       4'h4: out = 7'b0011001;     // |     |
       4'h5: out = 7'b0010010;     // --6 --
       4'h6: out = 7'b0000010;     // |     |
       4'h7: out = 7'b1111000;     // 4     2
       4'h8: out = 7'b0000000;     // |     |
       4'h9: out = 7'b0011000;     // --3 --
       4'ha: out = 7'b0001000;
       4'hb: out = 7'b0000011;
       4'hc: out = 7'b1000110;
       4'hd: out = 7'b0100001;
       4'he: out = 7'b0000110;
       4'hf: out = 7'b0001110;
     endcase
endmodule
```

Instead of using schematic capture, I now use Verilog code to define modules and wire them up using a top – level module. On the left is the code for 7-segment decoder with 4-bit binary input which we read as hexadecimal. Output register must also be defined as we expect them to hold values. Verilog assignment inside an always block must always be declared as reg, since its value can change, this is a rule in Verilog. In this case, all the states are declared, however if they weren't then we would have to write in default to catch all undefined states, otherwise output would be unpredictable if inside an undefined state. The * means at any change in input.

```
module  ex2_top (
      SW,                  // input switches
      HEX0                 // Hex output on 7 segment display
);
   input    [3:0] SW;    // declare input/output ports
   output   [6:0] HEX0;

   hex_to_7seg    SEG0 (HEX0, SW);

endmodule
```

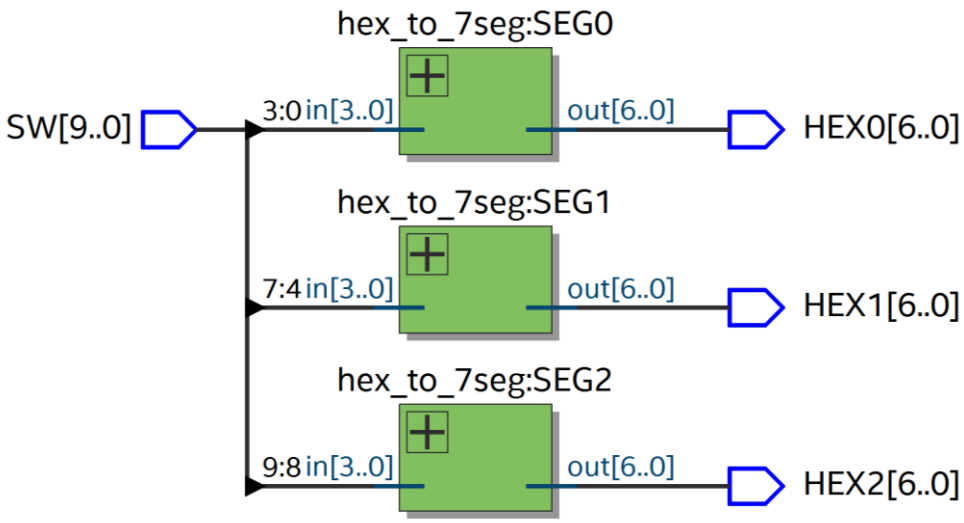
The top module defines all inputs and outputs and relates them to actual hardware, in this case the switches and a hex display. SEG0 is an instantiation of hex_to_7seg, we do this because we may need multiple instances and so we must declare them all.

Experiment 3 – 10-bit Binary switch

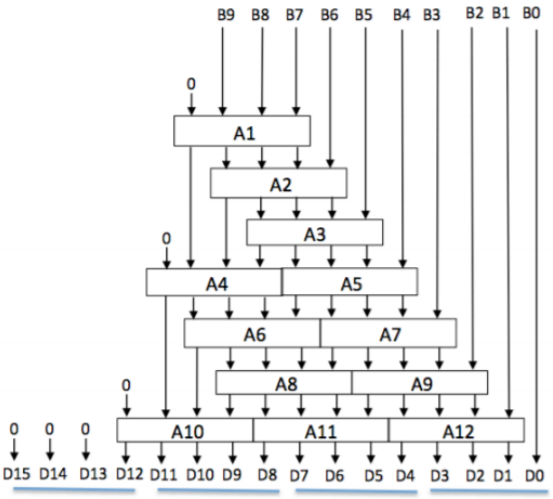
```
1   module ex3_top (
2                   SW,
3                   HEX0,
4                   HEX1,
5                   HEX2
6   );
7                   input   [9:0] SW;       //define inputs and outputs
8                   output  [6:0] HEX0;
9                   output  [6:0] HEX1;
10                  output  [6:0] HEX2;
11
12                  hex_to_7seg             SEG0 (HEX0, SW[3:0]);   //define  module behaviour
13                  hex_to_7seg             SEG1 (HEX1, SW[7:4]);
14                  hex_to_7seg             SEG2 (HEX2, SW[9:8]);
15
16  endmodule
```

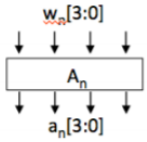
Above is the top module for the 10-bit binary switch. As can be seen from above, each seg display is configured to 4 switches apart from the last one, as only two remain. Each switch represents a binary bit in this case, which are read as hexadecimal when coupled into two 4s and one 2, allowing a simple conversion that can display a 10-bit binary number as 3 hexadecimal digits.

Below is the RTL top-level block diagram to show a clearer overview of the code.

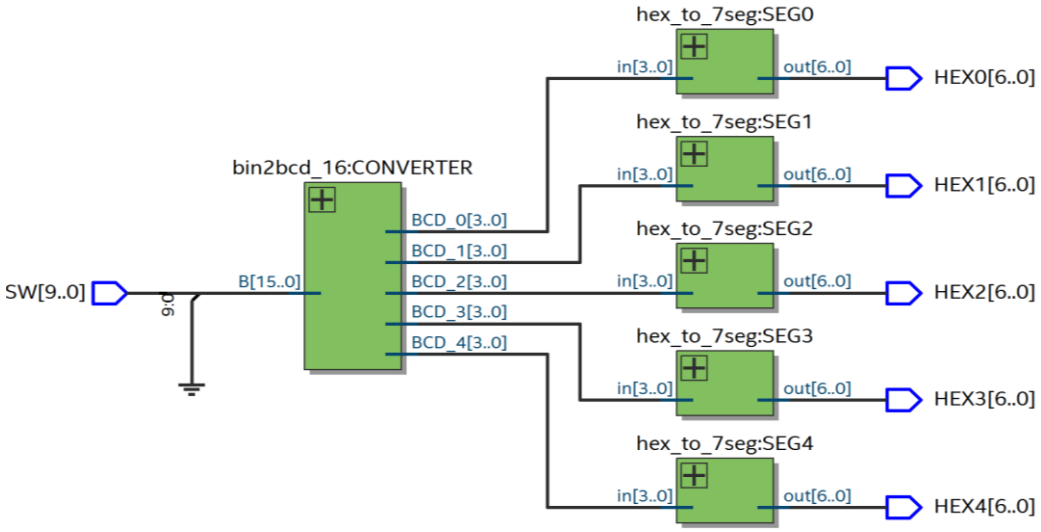


Experiment 4 (Optional) – Displaying 10-bit Binary as BCD digits on the 7-Segment Displays





In order to make this design work, I had to create a 10-bit binary to BCD converter, which I had already completed in problem sheet one, using a tree of shift and add 3 modules as shown on the left. The shift and add 3 algorithm shifts bits until the value reaches 5 or greater and then it shifts and adds 3. This allows the value in each BCD bin to be between 0-9 instead of 0-15 allowing the correct conversion.



Logic utilization (in ALMs)	37 / 32,070 (< 1 %)
Total registers	0
Total pins	45 / 457 (10 %)

Logic utilization (in ALMs)	37 / 32,070 (< 1 %)
Total registers	0
Total pins	38 / 457 (8 %)

In order to carry out the design I could use either a 10 or 16-bit converter since the software does the optimisation for us. As can be seen from above the top-level RTL block diagram would be the same for either converter and they use the same number of ALMs. However, when using the 16-bit converter I had to use concatenation to set the first 6 MSB digits of the input to 0 since we only have 10 switches.