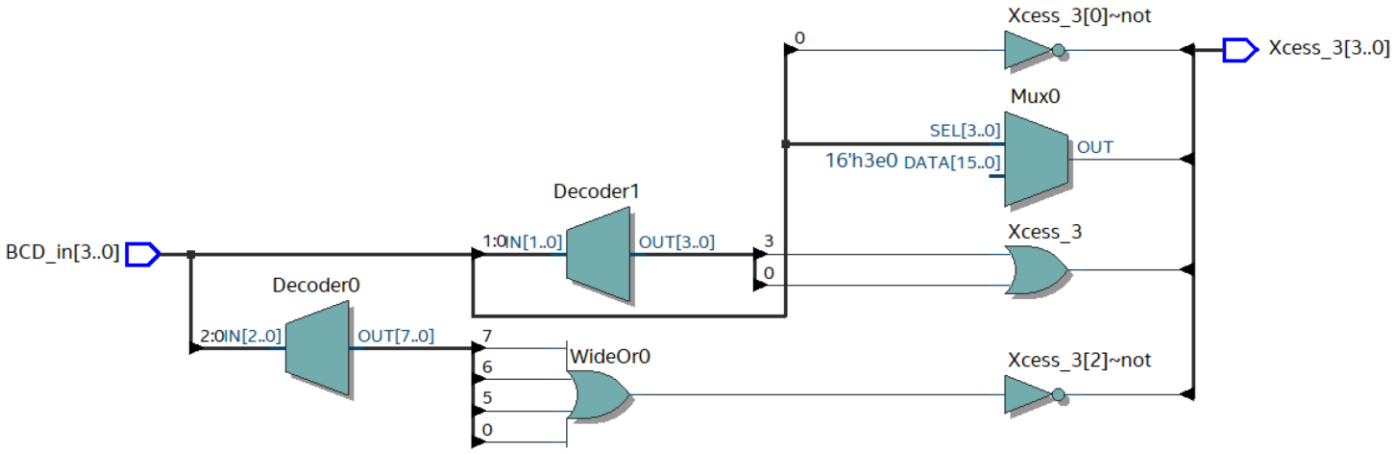
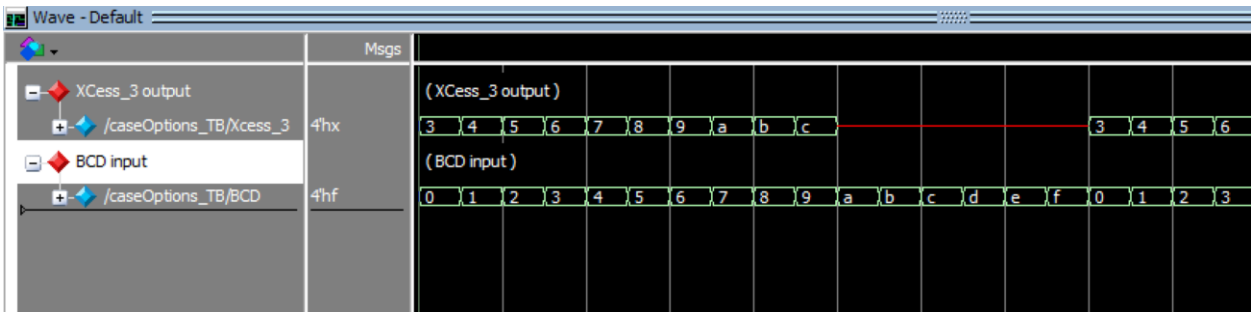
Case Structure options and how they are synthesized in Hardware

```
module caseOptions (input [3:0] BCD_in, output reg [3:0] Xcess_3);

always @ (BCD_in)
    case(BCD_in)
    0: Xcess_3 = 3;
    1: Xcess_3 = 4;
    2: Xcess_3 = 5;
    3: Xcess_3 = 6;
    4: Xcess_3 = 7;
    5: Xcess_3 = 8;
    6: Xcess_3 = 9;
    7: Xcess_3 = 10;
    8: Xcess_3 = 11;
    9: Xcess_3 = 12;
    default: Xcess_3 = 4'bxxxx;
    endcase

endmodule
```



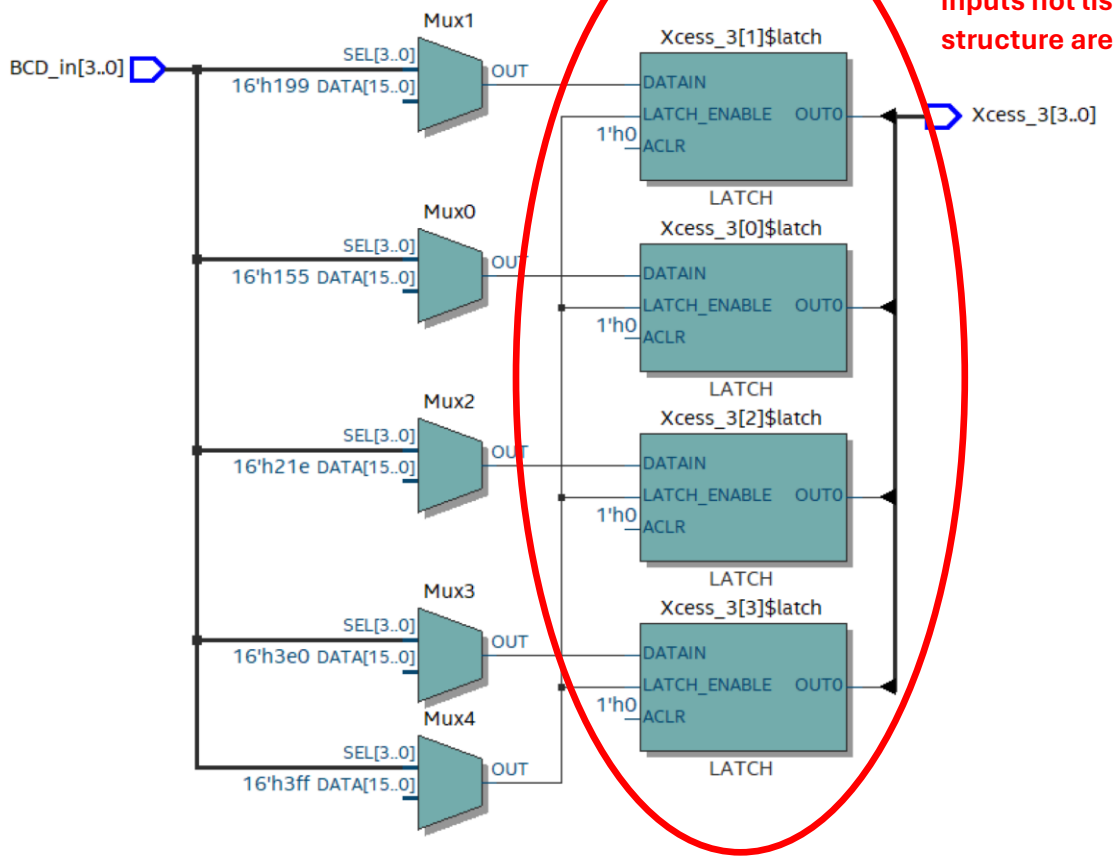


```
module caseOptions (input [3:0] BCD_in, output reg [3:0] Xcess_3);

always @ (BCD_in)
    case(BCD_in)
    0: Xcess_3 = 3;
    1: Xcess_3 = 4;
    2: Xcess_3 = 5;
    3: Xcess_3 = 6;
    4: Xcess_3 = 7;
    5: Xcess_3 = 8;
    6: Xcess_3 = 9;
    7: Xcess_3 = 10;
    8: Xcess_3 = 11;
    9: Xcess_3 = 12;
    //default: Xcess_3 = 4'bxxxx;
    endcase

endmodule
```

Quartus has inferred latches to keep, save and remember the output of the last valid input it was assigned. It did not get instructions in the code for what to do when inputs not listed in the case structure are assigned.

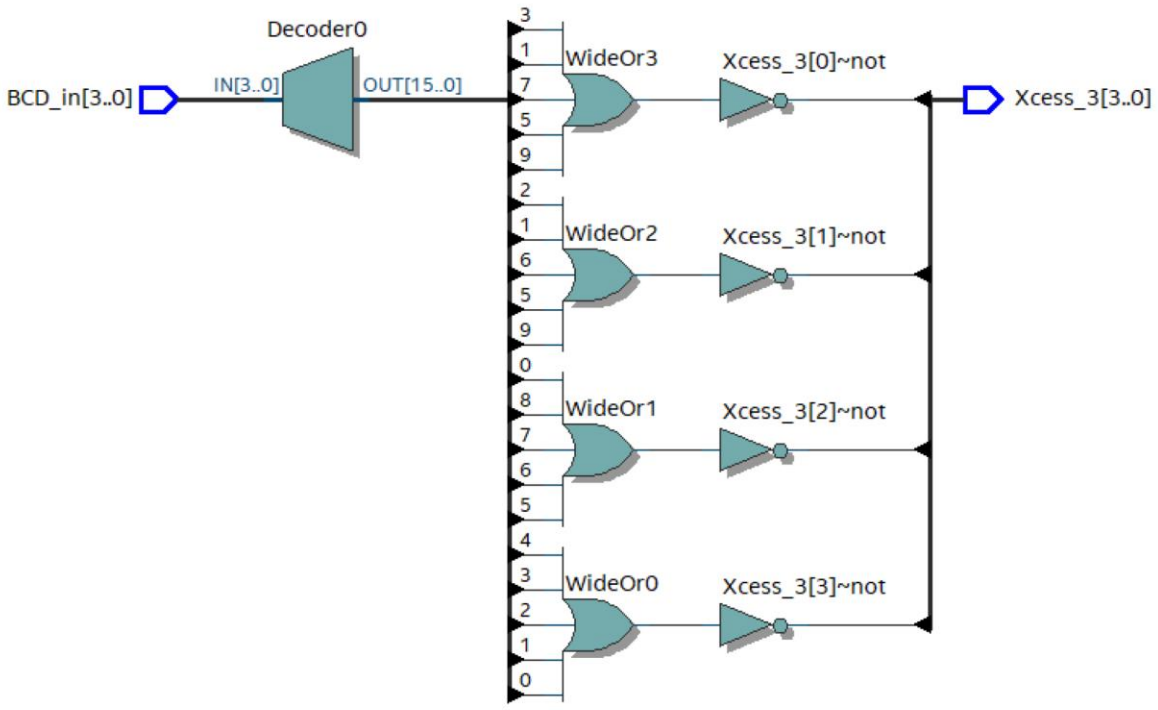


We see here that whenever the input is greater than 9, the design holds the old Xcess_3 value from the previous input. The latches were inferred because the code was not fully covering all possible input options.

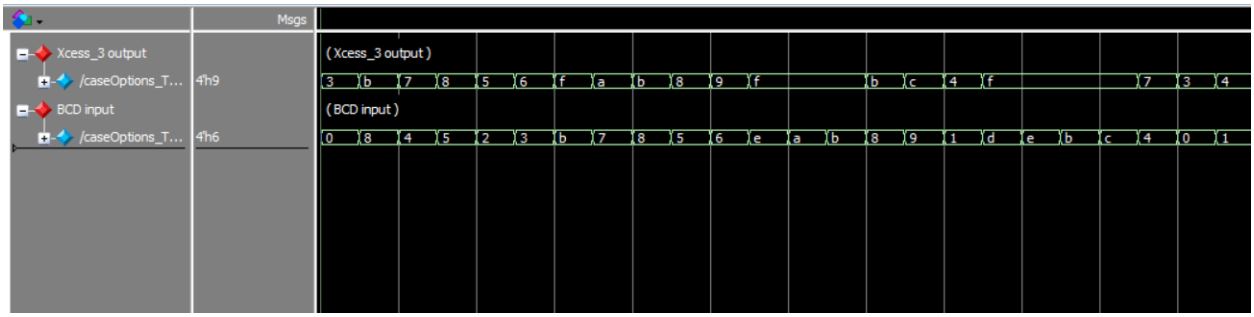
```
module caseOptions (input [3:0] BCD_in, output reg [3:0] Xcess_3);

always @ (BCD_in)
    case(BCD_in)
    0: Xcess_3 = 3;
    1: Xcess_3 = 4;
    2: Xcess_3 = 5;
    3: Xcess_3 = 6;
    4: Xcess_3 = 7;
    5: Xcess_3 = 8;
    6: Xcess_3 = 9;
    7: Xcess_3 = 10;
    8: Xcess_3 = 11;
    9: Xcess_3 = 12;
default: Xcess_3 = 4'b1111;    //An invalid output
    endcase

endmodule
```



The code is mostly instantiating primitive logic gates (or and not gates).



Whenever the BCD is greater than 9 the output shows 1111, which is not an Xcess_3 valid code. I can use this as a flag that the input is not within the BCD range.