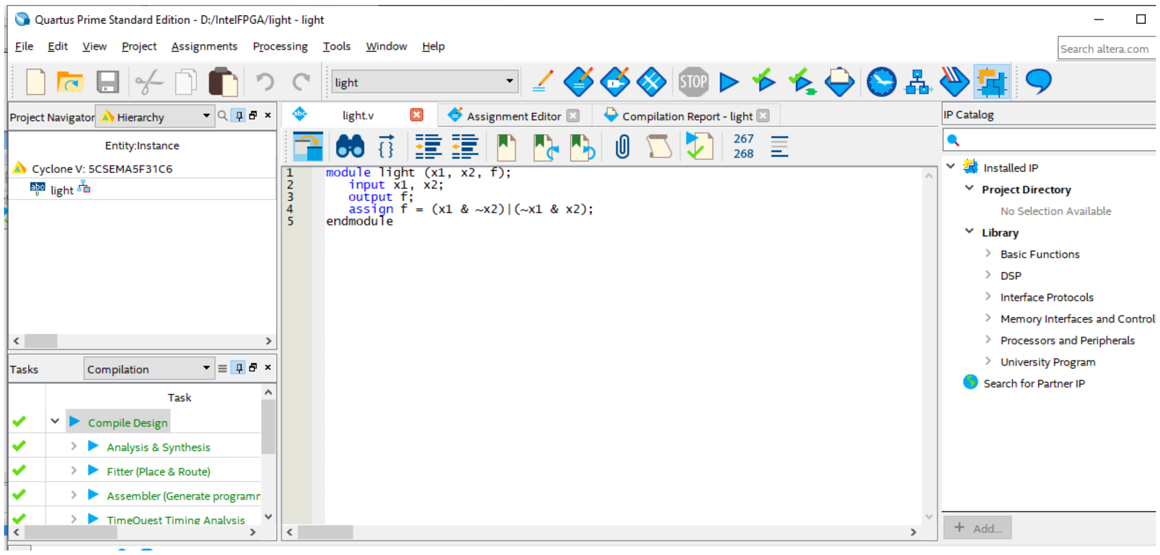
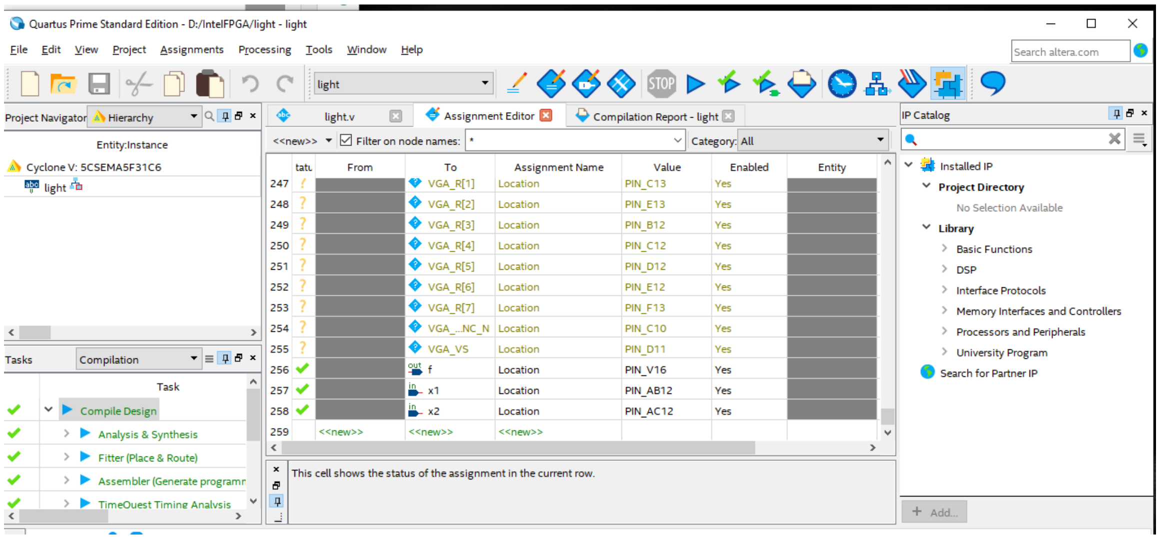
All instructions in the tutorial file are completed and snapshot was taken as followed:

Varilog implimentation:



```
module light (x1, x2, f);
  input x1, x2;
  output f;
  assign f = (x1 & ~x2)|(~x1 & x2);
endmodule
```

Pins assigned:

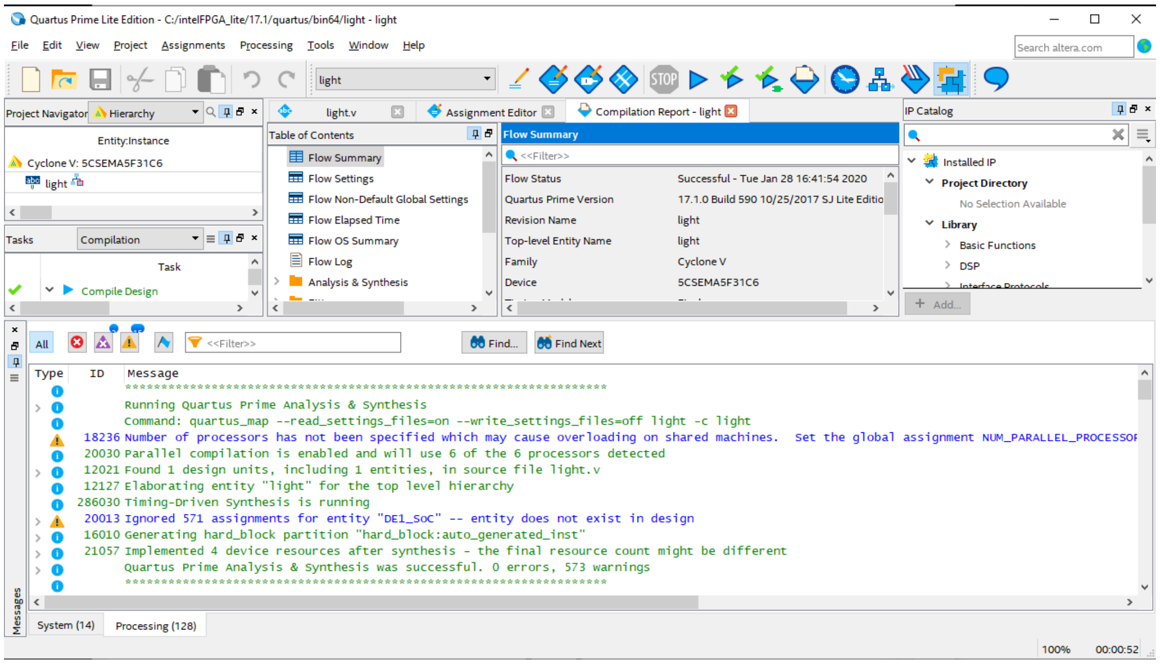


<<new>> Filter on node names: * Category: All

	tatu	From	To	Assignment Name	Value	Enabled	Entity
247	!		VGA_R[1]	Location	PIN_C13	Yes	
248	?		VGA_R[2]	Location	PIN_E13	Yes	
249	?		VGA_R[3]	Location	PIN_B12	Yes	
250	?		VGA_R[4]	Location	PIN_C12	Yes	
251	?		VGA_R[5]	Location	PIN_D12	Yes	
252	?		VGA_R[6]	Location	PIN_E12	Yes	
253	?		VGA_R[7]	Location	PIN_F13	Yes	
254	?		VGA_...NC_N	Location	PIN_C10	Yes	
255	?		VGA_VS	Location	PIN_D11	Yes	
256	✓		out f	Location	PIN_V16	Yes	
257	✓		in x1	Location	PIN_AB12	Yes	
258	✓		in x2	Location	PIN_AC12	Yes	
259		<<new>>	<<new>>	<<new>>			

This cell shows the status of the assignment in the current row.

Error free compilation:



Quartus Prime Lite Edition - C:/intelFPGA_lite/17.1/quartus/bin64/light - light

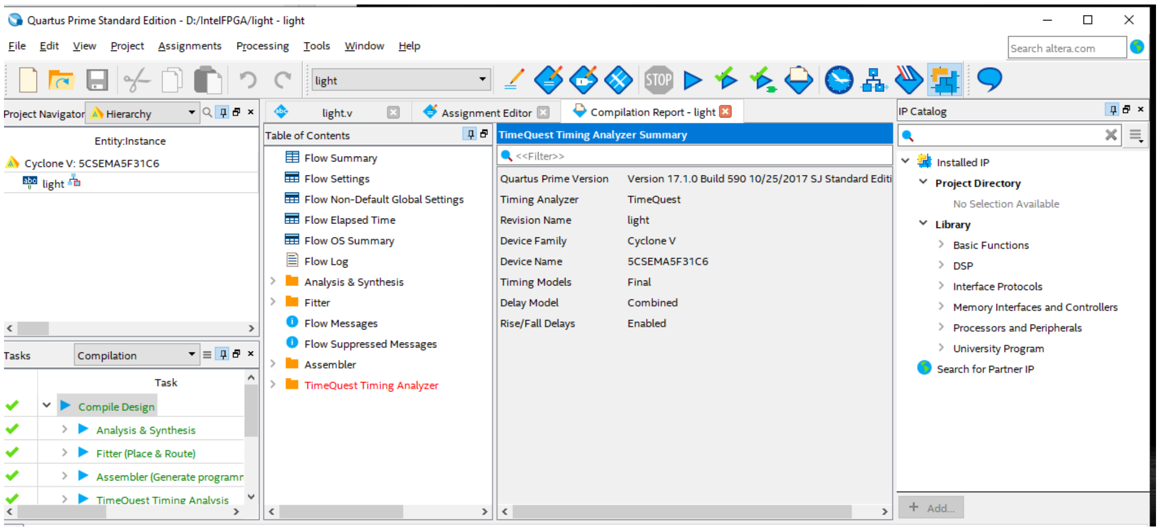
Flow Status	Successful - Tue Jan 28 16:41:54 2020
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Editio
Revision Name	light
Top-level Entity Name	light
Family	Cyclone V
Device	5CSEMA5F31C6

```
Type  ID     Message
             ***********************************************************************
             Running Quartus Prime Analysis & Synthesis
             Command: quartus_map --read_settings_files=on --write_settings_files=off light -c light
       18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSOF
       20030 Parallel compilation is enabled and will use 6 of the 6 processors detected
       12021 Found 1 design units, including 1 entities, in source file light.v
       12127 Elaborating entity "light" for the top level hierarchy
      286030 Timing-Driven Synthesis is running
       20013 Ignored 571 assignments for entity "DE1_SoC" -- entity does not exist in design
       16010 Generating hard_block partition "hard_block:auto_generated_inst"
       21057 Implemented 4 device resources after synthesis - the final resource count might be different
             Quartus Prime Analysis & Synthesis was successful. 0 errors, 573 warnings
             ***********************************************************************
```

System (14) Processing (128)

100% 00:00:52

Summary:



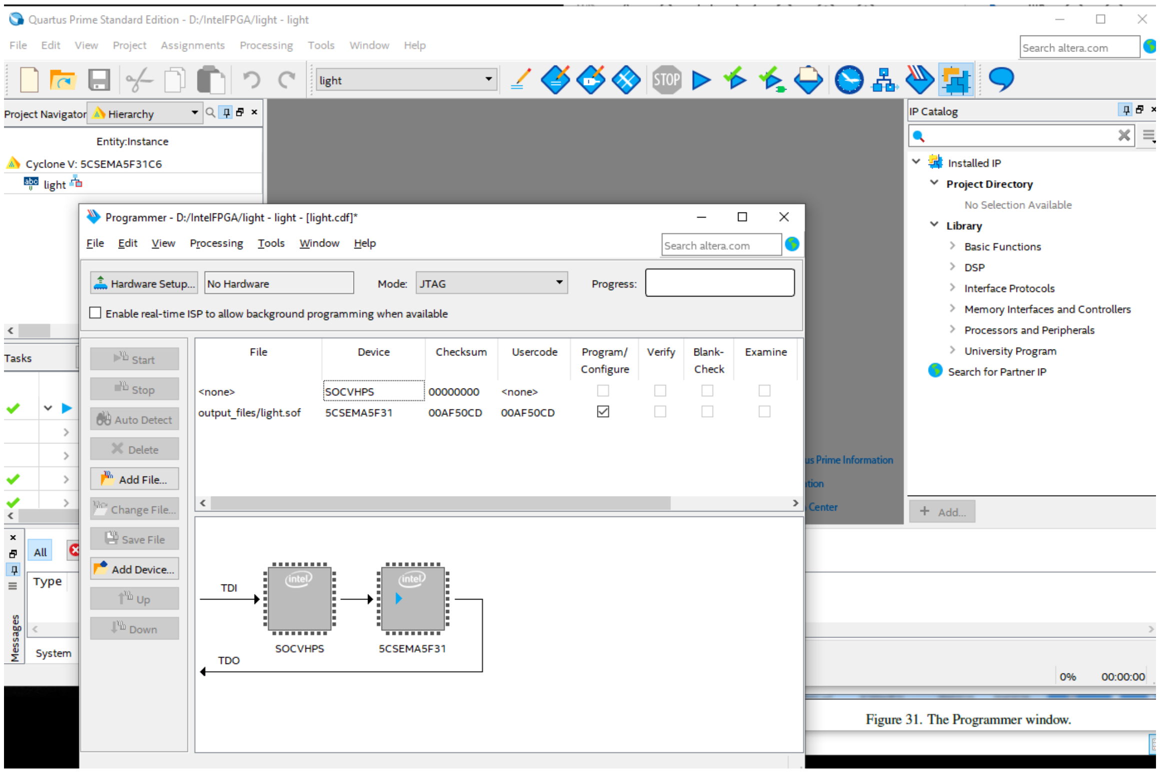
Quartus Prime Standard Edition - D:/IntelFPGA/light - light

TimeQuest Timing Analyzer Summary

Quartus Prime Version	Version 17.1.0 Build 590 10/25/2017 SJ Standard Editi
Timing Analyzer	TimeQuest
Revision Name	light
Device Family	Cyclone V
Device Name	5CSEMA5F31C6
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Enabled

Tasks: Compile Design ✓, Analysis & Synthesis ✓, Fitter (Place & Route) ✓, Assembler (Generate programr ✓, TimeQuest Timing Analysis ✓

Hardware programer:



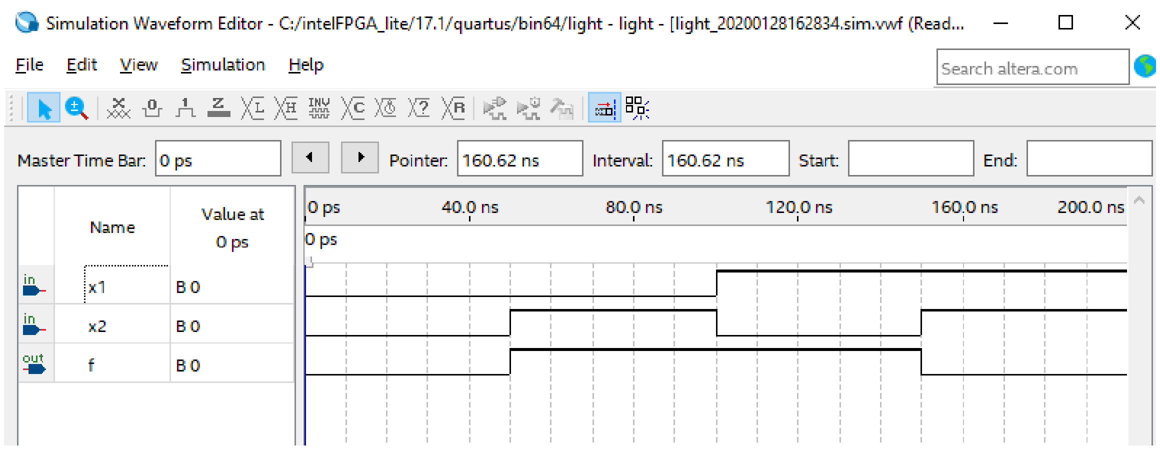
Hardware Setup... No Hardware Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐
output_files/light.sof	5CSEMA5F31	00AF50CD	00AF50CD	☑	☐	☐	☐

Figure 31. The Programmer window.

Function Simulation:


Simulation Waveform Editor - C:/intelFPGA_lite/17.1/quartus/bin64/light - light - [light_20200128162834.sim.vwf (Read...
File Edit View Simulation Help
Search altera.com
0 ps
40.0 ns
80.0 ns
120.0 ns
160.0 ns
200.0 ns
0 ps

Master Time Bar: 0 ps Pointer: 160.62 ns Interval: 160.62 ns Start: End:

	Name	Value at 0 ps
in	x1	B 0
in	x2	B 0
out	f	B 0

1. The **reg** stands for register. It represents data storage elements in Verilog. They retain their value till next value is assigned to them.

   The **wire** is used for connecting different elements. They can be treated as physical wires. They can be treated as physical wires. No values get stored in them.
2. **Wire** can only be used on the left side of a continuous assignment and often declared as an input.
3. Rules for **Inputs**: internally must always be of type net, externally the inputs can be connected to a variable of type reg or net.

   Rules for **Output**: internally can be of type reg or net. Externally can be of datatype or net type net.

   Rules for **Inouts**: internally or externally must always be type net.
4. **Continuous Assignment:** continuous assignment drives a value into a net. It is declared outside of procedural blocks.

   Blocking Assignment: executed before the execution of the statements that follow it in a sequential block. The operator is "="

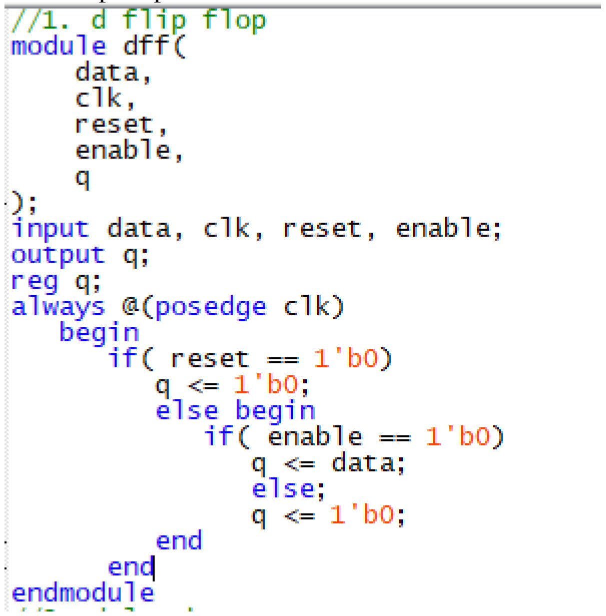
   Nonblocking Assignment: it allows us to assign values without blocking the procedural flow. The operator is "<=".

5. **Combinational logic** is time independent, and logic does not depend on the previous inputs.

   Sequential logic is dependent on the clock cycles and the output depends on present as well as past inputs.

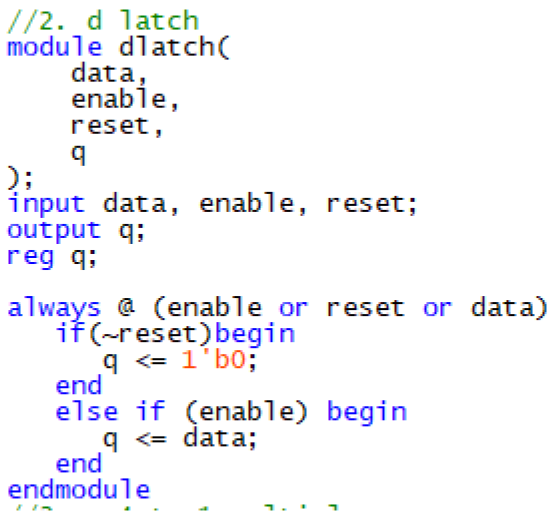
6.
   1. Include all the branches of an if or case statement
   2. Assign a value to every output signal in every branch.
   3. Use default assignment at the start of the procedure, so every signal will be assigned.
7. << is binary logical shift, and <<< is arithmetic left shift.
8. wire[6:0] x[5:0];

//1. d flip flop



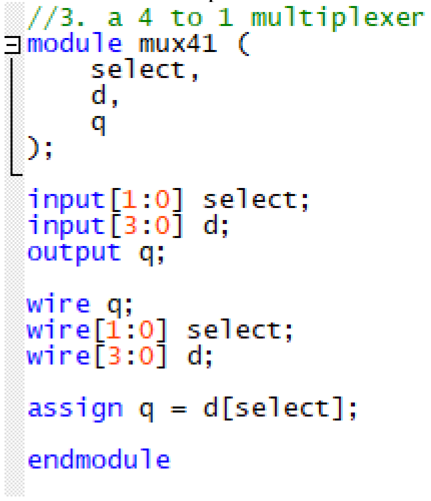
```
//1. d flip flop
module dff(
    data,
    clk,
    reset,
    enable,
    q
);
input data, clk, reset, enable;
output q;
reg q;
always @(posedge clk)
   begin
      if( reset == 1'b0)
         q <= 1'b0;
         else begin
            if( enable == 1'b0)
               q <= data;
               else;
               q <= 1'b0;
         end
      end
endmodule
```

//2. d latch



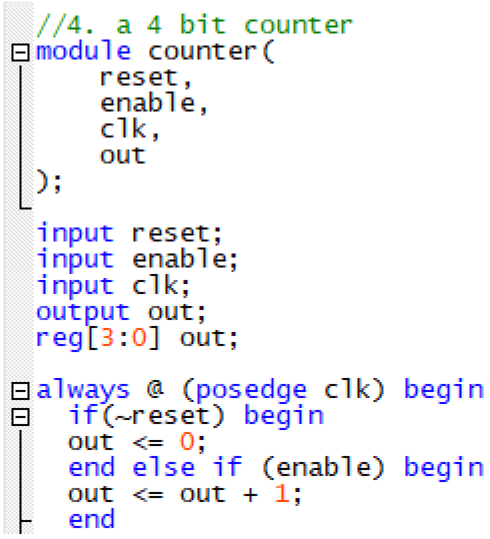
```
//2. d latch
module dlatch(
    data,
    enable,
    reset,
    q
);
input data, enable, reset;
output q;
reg q;

always @ (enable or reset or data)
   if(~reset)begin
      q <= 1'b0;
   end
   else if (enable) begin
      q <= data;
   end
endmodule
```

//3. a 4 to 1 multiplexer

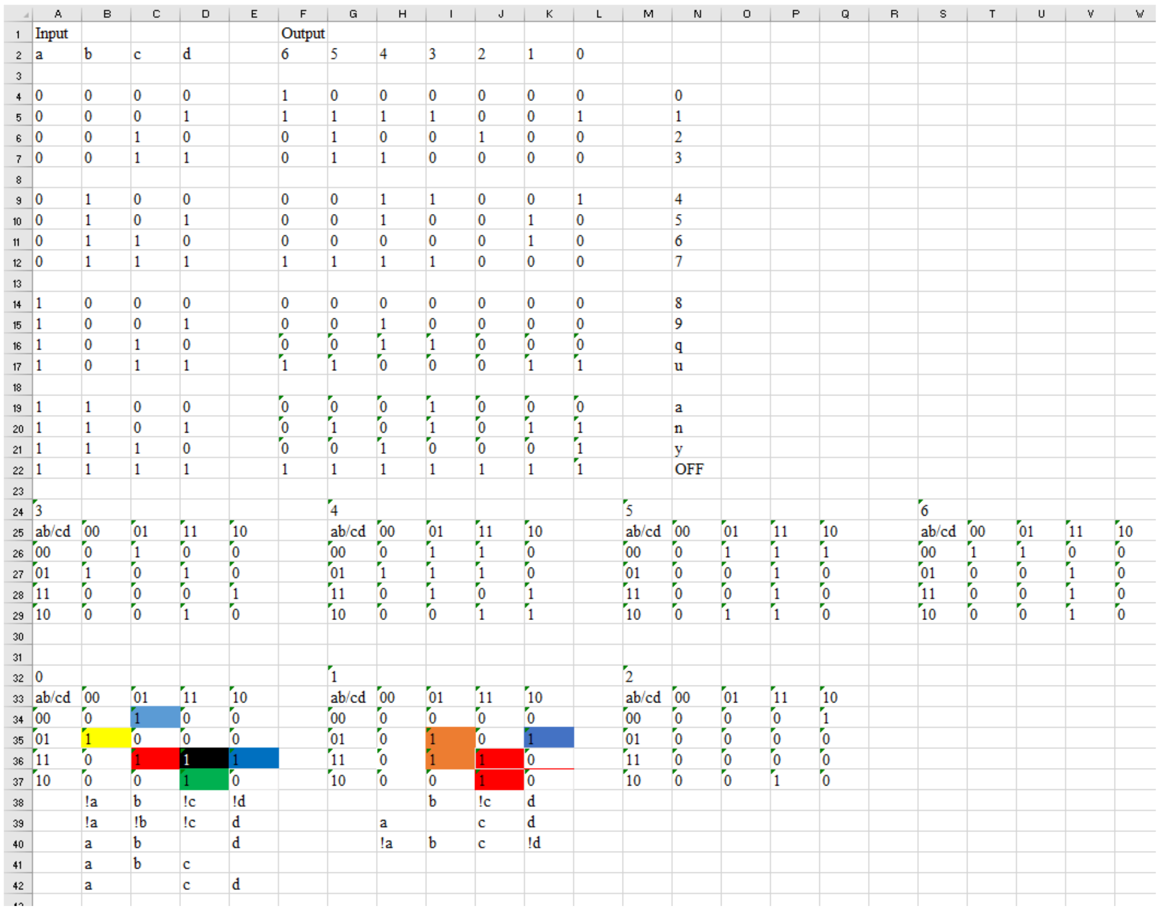


```
//3. a 4 to 1 multiplexer
module mux41 (
    select,
    d,
    q
);

input[1:0] select;
input[3:0] d;
output q;

wire q;
wire[1:0] select;
wire[3:0] d;

assign q = d[select];

endmodule
```

//4. a 4 bit counter



```
//4. a 4 bit counter
module counter(
    reset,
    enable,
    clk,
    out
);

input reset;
input enable;
input clk;
output out;
reg[3:0] out;

always @ (posedge clk) begin
  if(~reset) begin
  out <= 0;
  end else if (enable) begin
  out <= out + 1;
  end
```



Input				Output							
a	b	c	d	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	1	1
0	0	1	0	0	1	0	0	1	0	0	2
0	0	1	1	0	1	1	0	0	0	0	3
0	1	0	0	0	0	1	1	0	0	1	4
0	1	0	1	0	0	1	0	0	1	0	5
0	1	1	0	0	0	0	0	0	1	0	6
0	1	1	1	1	1	1	1	0	0	0	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	1	0	0	0	0	9
1	0	1	0	0	0	1	1	0	0	0	q
1	0	1	1	1	1	0	0	0	1	1	u
1	1	0	0	0	0	0	1	0	0	0	a
1	1	0	1	0	1	0	1	0	1	1	n
1	1	1	0	0	0	1	0	0	0	1	y
1	1	1	1	1	1	1	1	1	1	1	OFF

3				
ab/cd	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	1	0

4				
ab/cd	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	0	1	0	1
10	0	0	1	1

5				
ab/cd	00	01	11	10
00	0	1	1	1
01	0	0	1	0
11	0	0	1	0
10	0	1	1	0

6				
ab/cd	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

0				
ab/cd	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	1	1	1
10	0	0	1	0
	!a	b	!c	!d
	!a	!b	!c	d
	a	b		d
	a	b	c	
	a		c	d

1				
ab/cd	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	0	1	1	0
10	0	0	1	0
		b	!c	d
	a		c	d
	!a	b	c	!d

2				
ab/cd	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	1	0

0: !a b !c !d + !a !b !c d + a b d + a b c + a c d

1: b !c d + a c d + !a b c !d

```
module seven seg_decoder(input [3:0] x, output [6:0] hex_LEDs);
        reg [6:0] reg_LEDs;
        assign hex_LEDs[0]=(~input[a]&input[b]&~input[c]&~input[d])|
        (~input[a]&~input[b]&~input[c]&input[d])|(input[a]&input[b]&input[d])|
        (input[a]&input[b]&input[c]) |(input[a]&input[c]&input[d]);
        assign hex_LEDs[1]=(input[b]&~input[c]&input[d]) |(input[a]&input[c]&input[d])
        |(~input[a]&input[b]&input[c]&~input[d]);
        assign hex_LEDs[6:2]=reg_LEDs[6:2];
        always @(*)
        begin
                case (x)
                        4'b0000: reg_LEDs[6:2]=7'b10000; //7'b1000000 decimal 0
                        4'b0001: reg_LEDs[6:2]=7'b11110; //7'b1111001 decimal 1
                        4'b0010: reg_LEDs[6:2]=7'b01001; //7'b0100100 decimal 2
                        4'b0011: reg_LEDs[6:2]=7'b01001; //7'b0110000 decimal 3
                        4'b0100: reg_LEDs[6:2]=7'b01001; //7'b0011001 decimal 4
                        4'b0101: reg_LEDs[6:2]=7'b01001; //7'b0010010 decimal 5
                        4'b0110: reg_LEDs[6:2]=7'b01001; //7'b0000010 decimal 6
                        4'b0111: reg_LEDs[6:2]=7'b01001; //7'b1111000 decimal 7
                        4'b1000: reg_LEDs[6:2]=7'b01001; //7'b0000000 decimal 8
                        4'b1001: reg_LEDs[6:2]=7'b01001; //7'b0010000 decimal 9
                        4'b1010: reg_LEDs[6:2]=7'b01001; //7'b0011000 decimal q
                        4'b1011: reg_LEDs[6:2]=7'b01001; //7'b1100011 decimal u
                        4'b1100: reg_LEDs[6:2]=7'b01001; //7'b0001000 decimal a
                        4'b1101: reg_LEDs[6:2]=7'b01001; //7'b0101011 decimal n
                        4'b1110: reg_LEDs[6:2]=7'b01001; //7'b0010001 decimal y
                        4'b1111: reg_LEDs[6:2]=7'b01001; //7'b1111111 decimal OFF
                endcase
        end
endmodule
```