Project 3

System Synthesis and Modeling

Zack Fravel

010646947

zpfravel@uark.edu

Introduction

The objective of Project 3 was to design and simulate a given combinational circuit in Verilog using ModelSim. Specifically, the circuit we want to design takes in an 8-bit input and counts the number of 1's detected in the word; after counting them up, the circuit then outputs the counter result using a 4 bit output. Similar to our previous projects, we were also tasked with not only implementing the solution in Verilog, but also writing a testbench or .DO file depending on our method of simulation. Finally, we also want to show the the circuit works as designed for all possible 8-bit input patterns.

Design and Simulation

The overarching idea of the assigned combinational circuit is shown below, in figure 1. To reiterate, we want to take in 8 bits at once and set a 4 bit output equal to the number of 1's detected in the 8 bit input.

Bit Counter Circuit Diagram

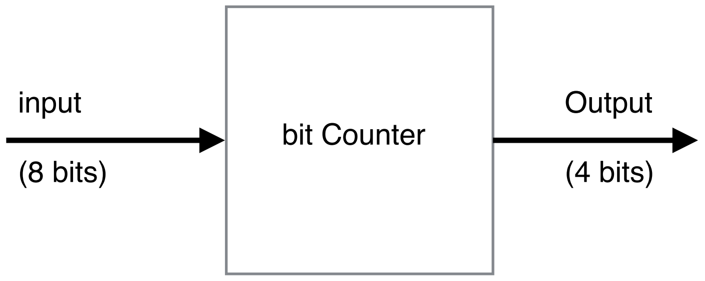
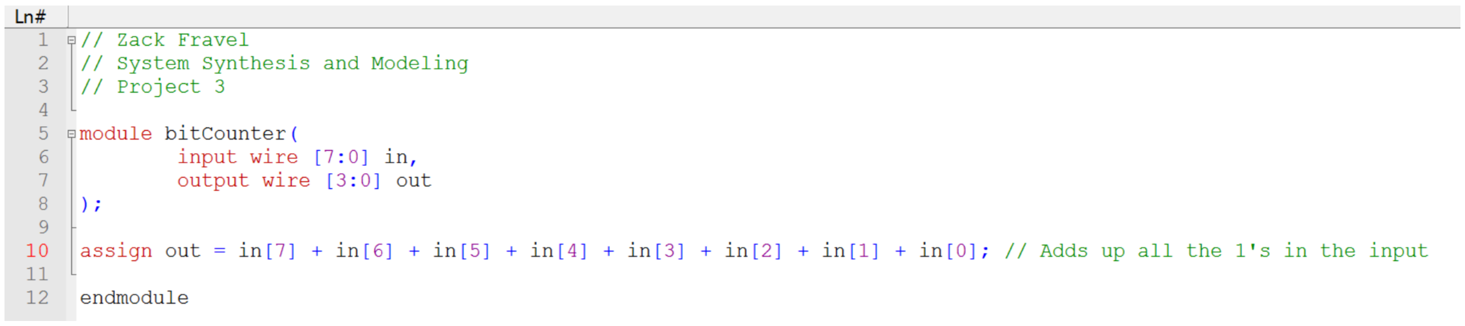


Figure 1

The implementation of this circuit in Verilog is very straight forward, we want to declare out input [7:0] and our output [3:0] (I declare them as wires since we don't need to store any data, just performing an operation). After our ports have been declared all that is needed is one line of code which assigns the output to the sum of each bit in the input stream. Below, in figure 2, is a screenshot of the Verilog design.

bitCounter.v



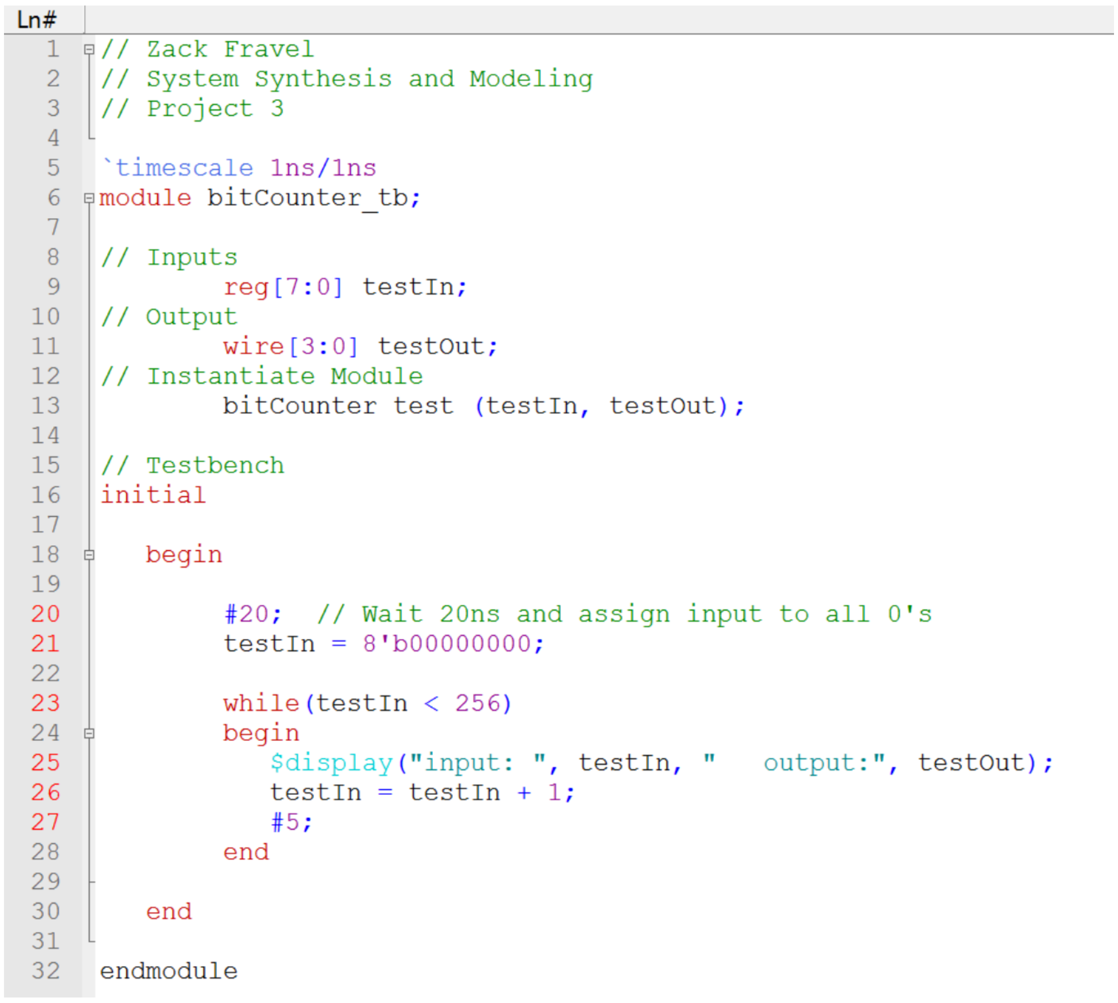
```
// Zack Fravel
// System Synthesis and Modeling
// Project 3

module bitCounter(
        input wire [7:0] in,
        output wire [3:0] out
);

assign out = in[7] + in[6] + in[5] + in[4] + in[3] + in[2] + in[1] + in[0]; // Adds up all the 1's in the input

endmodule
```

Figure 2

Once I was sure it compiled correctly, I moved on to designing a suitable testbench to go through all possible combinations of input.

I declare the testbench module as bitCounter_tb and below declare reg [7:0] testIn, wire [3:0] testOut, and finally initialize my bitCounter module using the respective ports. At the top of the testbench I declare my timescale as 1ns/1ns to be able and add some delay between each operation. In my initial block I have the simulator wait 20ns and set the input to “00000000.” From here, I create a while loop that checks if the input is less than 256 (255 being the greatest possible integer value that can be stored in 8 bits). Within this while loop I have the compiler display the current state of the input and output; after that is complete the testbench increments the input by 1 and waits 5 ns. This repeats until the input reaches 255, and the cycle continues for as long as is needed. Attached also is figure 3, which shows the Verilog implementation of this testbench module.

bitCounter_tb.v



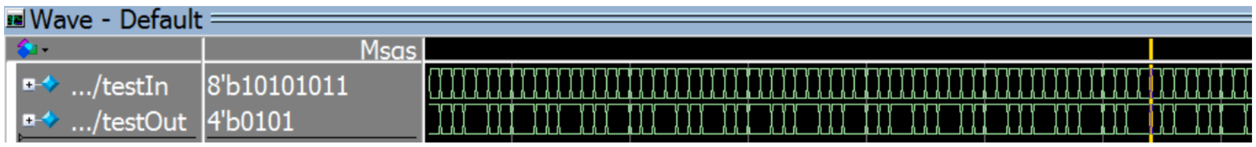
```
// Zack Fravel
// System Synthesis and Modeling
// Project 3

`timescale 1ns/1ns
module bitCounter_tb;

// Inputs
        reg[7:0] testIn;
// Output
        wire[3:0] testOut;
// Instantiate Module
        bitCounter test (testIn, testOut);

// Testbench
initial

   begin

        #20;  // Wait 20ns and assign input to all 0's
        testIn = 8'b00000000;

        while(testIn < 256)
        begin
           $display("input: ", testIn, "   output:", testOut);
           testIn = testIn + 1;
           #5;
        end

   end

endmodule
```

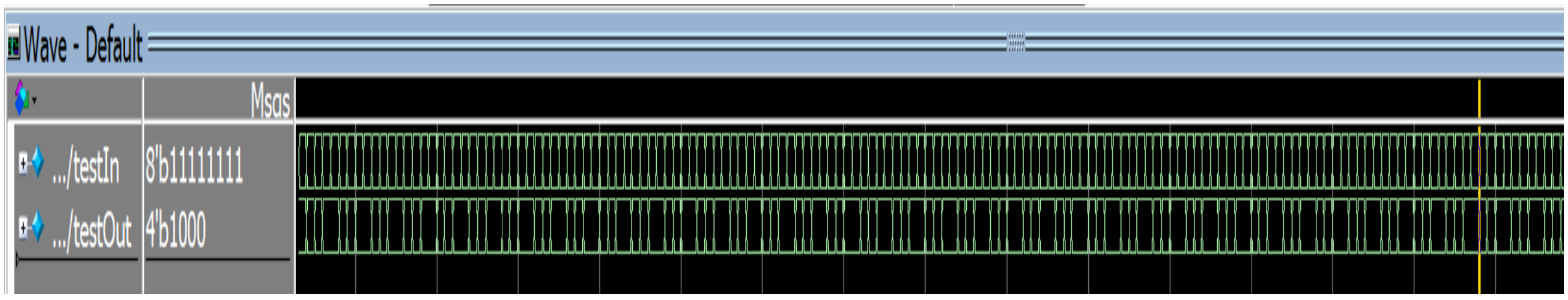
Figure 3

Results, Analysis, and Conclusions

Once I had my testbench compiled I was ready to begin simulating. I loaded the testbench into the simulator and added the testIn and testOut waves and ran for 1500ns, which is more than enough time for the testbench to run through all possible input patterns with a 5ns delay between each. Below are a three waveform screenshots of random points in the simulation run time.

Simulation





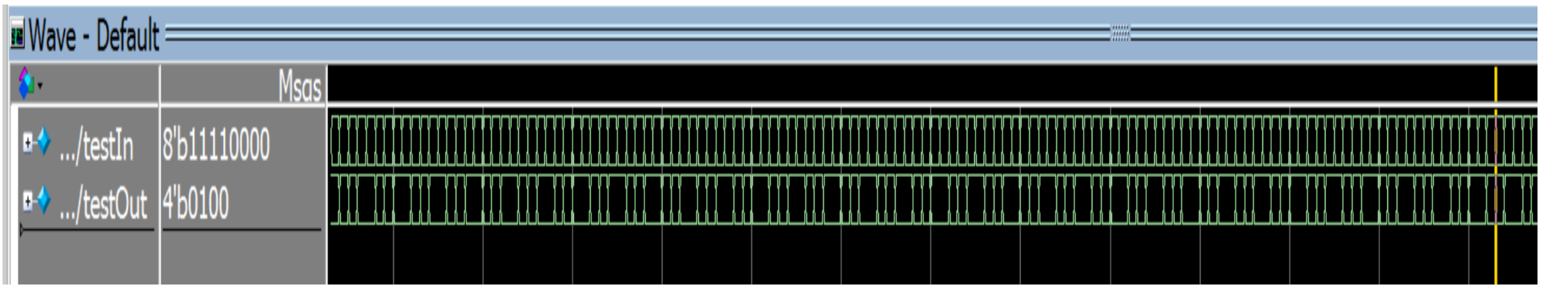
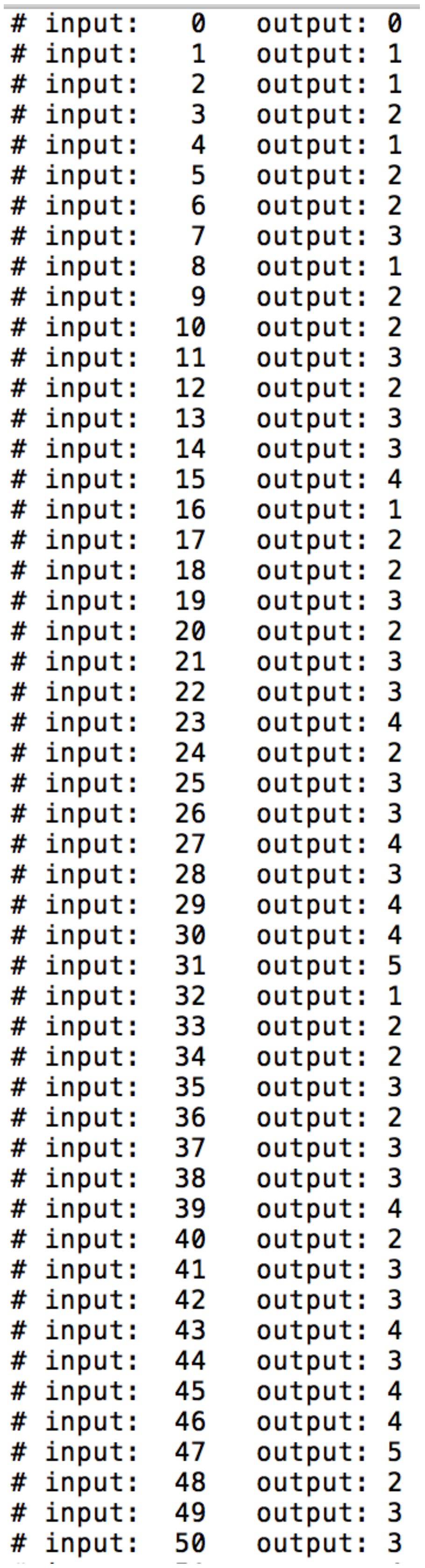


Figure 4

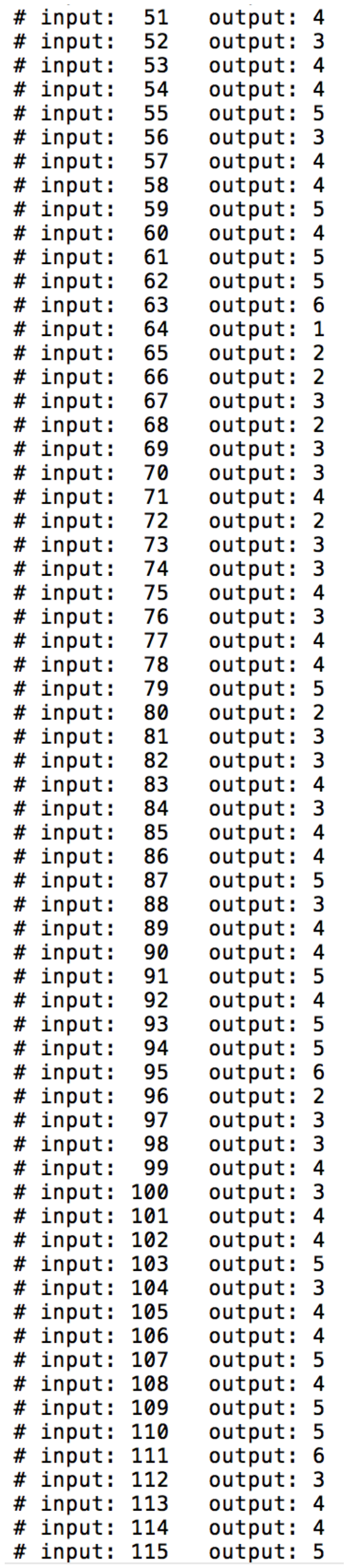
It can be seen above that the combinational circuit designed does exactly what we set out to do. In the first screenshot, we have an input "10101011," which contains 5 "1" bits; on our output, we have 0101, which is the 4 bit representation for the number 5. On the second screenshot we show 8 bits that assert "1", and on the third shows 4 bits with each having their correct output. The compiler also outputs a string each time it loops displaying the input and output, which is attached to the end of the report.

My simulation ran for 1500 ns, which was enough time for the circuit to go through all possible combinations, plus about 45 after being set back to "00000000." I included some of the repeating in the output attached at the end, just to show it functions and loops correctly. In conclusion, I successfully designed a simple combinational circuit in Verilog that takes in 8-bits, counts the number of 1's present, and outputs a 4-bit representation of that counter. All source files are included in the report package.

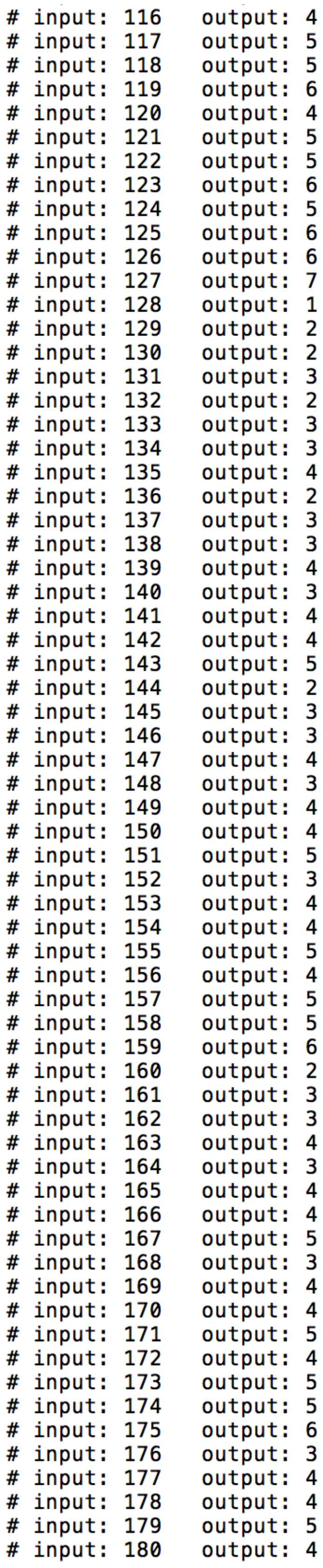
Compiler Output



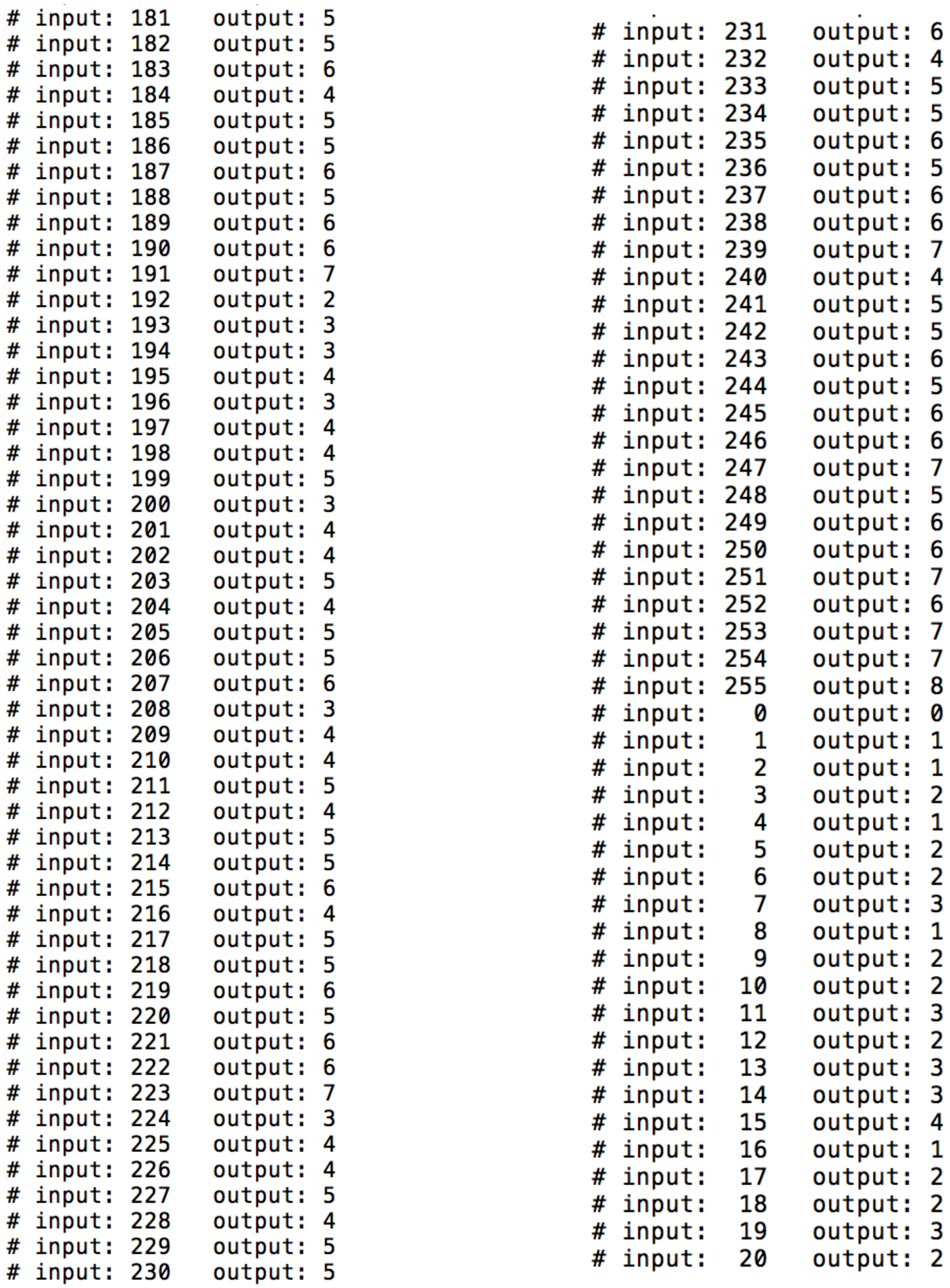
```
# input:   0   output: 0
# input:   1   output: 1
# input:   2   output: 1
# input:   3   output: 2
# input:   4   output: 1
# input:   5   output: 2
# input:   6   output: 2
# input:   7   output: 3
# input:   8   output: 1
# input:   9   output: 2
# input:  10   output: 2
# input:  11   output: 3
# input:  12   output: 2
# input:  13   output: 3
# input:  14   output: 3
# input:  15   output: 4
# input:  16   output: 1
# input:  17   output: 2
# input:  18   output: 2
# input:  19   output: 3
# input:  20   output: 2
# input:  21   output: 3
# input:  22   output: 3
# input:  23   output: 4
# input:  24   output: 2
# input:  25   output: 3
# input:  26   output: 3
# input:  27   output: 4
# input:  28   output: 3
# input:  29   output: 4
# input:  30   output: 4
# input:  31   output: 5
# input:  32   output: 1
# input:  33   output: 2
# input:  34   output: 2
# input:  35   output: 3
# input:  36   output: 2
# input:  37   output: 3
# input:  38   output: 3
# input:  39   output: 4
# input:  40   output: 2
# input:  41   output: 3
# input:  42   output: 3
# input:  43   output: 4
# input:  44   output: 3
# input:  45   output: 4
# input:  46   output: 4
# input:  47   output: 5
# input:  48   output: 2
# input:  49   output: 3
# input:  50   output: 3
```



```
# input:  51   output: 4
# input:  52   output: 3
# input:  53   output: 4
# input:  54   output: 4
# input:  55   output: 5
# input:  56   output: 3
# input:  57   output: 4
# input:  58   output: 4
# input:  59   output: 5
# input:  60   output: 4
# input:  61   output: 5
# input:  62   output: 5
# input:  63   output: 6
# input:  64   output: 1
# input:  65   output: 2
# input:  66   output: 2
# input:  67   output: 3
# input:  68   output: 2
# input:  69   output: 3
# input:  70   output: 3
# input:  71   output: 4
# input:  72   output: 2
# input:  73   output: 3
# input:  74   output: 3
# input:  75   output: 4
# input:  76   output: 3
# input:  77   output: 4
# input:  78   output: 4
# input:  79   output: 5
# input:  80   output: 2
# input:  81   output: 3
# input:  82   output: 3
# input:  83   output: 4
# input:  84   output: 3
# input:  85   output: 4
# input:  86   output: 4
# input:  87   output: 5
# input:  88   output: 3
# input:  89   output: 4
# input:  90   output: 4
# input:  91   output: 5
# input:  92   output: 4
# input:  93   output: 5
# input:  94   output: 5
# input:  95   output: 6
# input:  96   output: 2
# input:  97   output: 3
# input:  98   output: 3
# input:  99   output: 4
# input: 100   output: 3
# input: 101   output: 4
# input: 102   output: 4
# input: 103   output: 5
# input: 104   output: 3
# input: 105   output: 4
# input: 106   output: 4
# input: 107   output: 5
# input: 108   output: 4
# input: 109   output: 5
# input: 110   output: 5
# input: 111   output: 6
# input: 112   output: 3
# input: 113   output: 4
# input: 114   output: 4
# input: 115   output: 5
```



```
# input: 116   output: 4
# input: 117   output: 5
# input: 118   output: 5
# input: 119   output: 6
# input: 120   output: 4
# input: 121   output: 5
# input: 122   output: 5
# input: 123   output: 6
# input: 124   output: 5
# input: 125   output: 6
# input: 126   output: 6
# input: 127   output: 7
# input: 128   output: 1
# input: 129   output: 2
# input: 130   output: 2
# input: 131   output: 3
# input: 132   output: 2
# input: 133   output: 3
# input: 134   output: 3
# input: 135   output: 4
# input: 136   output: 2
# input: 137   output: 3
# input: 138   output: 3
# input: 139   output: 4
# input: 140   output: 3
# input: 141   output: 4
# input: 142   output: 4
# input: 143   output: 5
# input: 144   output: 2
# input: 145   output: 3
# input: 146   output: 3
# input: 147   output: 4
# input: 148   output: 3
# input: 149   output: 4
# input: 150   output: 4
# input: 151   output: 5
# input: 152   output: 3
# input: 153   output: 4
# input: 154   output: 4
# input: 155   output: 5
# input: 156   output: 4
# input: 157   output: 5
# input: 158   output: 5
# input: 159   output: 6
# input: 160   output: 2
# input: 161   output: 3
# input: 162   output: 3
# input: 163   output: 4
# input: 164   output: 3
# input: 165   output: 4
# input: 166   output: 4
# input: 167   output: 5
# input: 168   output: 3
# input: 169   output: 4
# input: 170   output: 4
# input: 171   output: 5
# input: 172   output: 4
# input: 173   output: 5
# input: 174   output: 5
# input: 175   output: 6
# input: 176   output: 3
# input: 177   output: 4
# input: 178   output: 4
# input: 179   output: 5
# input: 180   output: 4
```



```
# input: 181   output: 5
# input: 182   output: 5
# input: 183   output: 6
# input: 184   output: 4
# input: 185   output: 5
# input: 186   output: 5
# input: 187   output: 6
# input: 188   output: 5
# input: 189   output: 6
# input: 190   output: 6
# input: 191   output: 7
# input: 192   output: 2
# input: 193   output: 3
# input: 194   output: 3
# input: 195   output: 4
# input: 196   output: 3
# input: 197   output: 4
# input: 198   output: 4
# input: 199   output: 5
# input: 200   output: 3
# input: 201   output: 4
# input: 202   output: 4
# input: 203   output: 5
# input: 204   output: 4
# input: 205   output: 5
# input: 206   output: 5
# input: 207   output: 6
# input: 208   output: 3
# input: 209   output: 4
# input: 210   output: 4
# input: 211   output: 5
# input: 212   output: 4
# input: 213   output: 5
# input: 214   output: 5
# input: 215   output: 6
# input: 216   output: 4
# input: 217   output: 5
# input: 218   output: 5
# input: 219   output: 6
# input: 220   output: 5
# input: 221   output: 6
# input: 222   output: 6
# input: 223   output: 7
# input: 224   output: 3
# input: 225   output: 4
# input: 226   output: 4
# input: 227   output: 5
# input: 228   output: 4
# input: 229   output: 5
# input: 230   output: 5
```

```
# input: 231   output: 6
# input: 232   output: 4
# input: 233   output: 5
# input: 234   output: 5
# input: 235   output: 6
# input: 236   output: 5
# input: 237   output: 6
# input: 238   output: 6
# input: 239   output: 7
# input: 240   output: 4
# input: 241   output: 5
# input: 242   output: 5
# input: 243   output: 6
# input: 244   output: 5
# input: 245   output: 6
# input: 246   output: 6
# input: 247   output: 7
# input: 248   output: 5
# input: 249   output: 6
# input: 250   output: 6
# input: 251   output: 7
# input: 252   output: 6
# input: 253   output: 7
# input: 254   output: 7
# input: 255   output: 8
# input:   0   output: 0
# input:   1   output: 1
# input:   2   output: 1
# input:   3   output: 2
# input:   4   output: 1
# input:   5   output: 2
# input:   6   output: 2
# input:   7   output: 3
# input:   8   output: 1
# input:   9   output: 2
# input:  10   output: 2
# input:  11   output: 3
# input:  12   output: 2
# input:  13   output: 3
# input:  14   output: 3
# input:  15   output: 4
# input:  16   output: 1
# input:  17   output: 2
# input:  18   output: 2
# input:  19   output: 3
# input:  20   output: 2
```

Figure 6