CS M51A and EE M16 Summer 2016 Section 1

Logic Design of Digital Systems

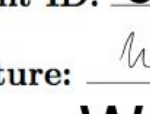
Dr. Yutao He

Verilog Lab #2 - Design of Combinational Systems

Due: July 31st, 2016

(1) **Name:** Zhou Minghong
Last First

Student ID: 004424670

Signature: 

(2) **Name:** Wu Michael
Last First

Student ID: 404751542

Signature: 

Date: 07/23/16

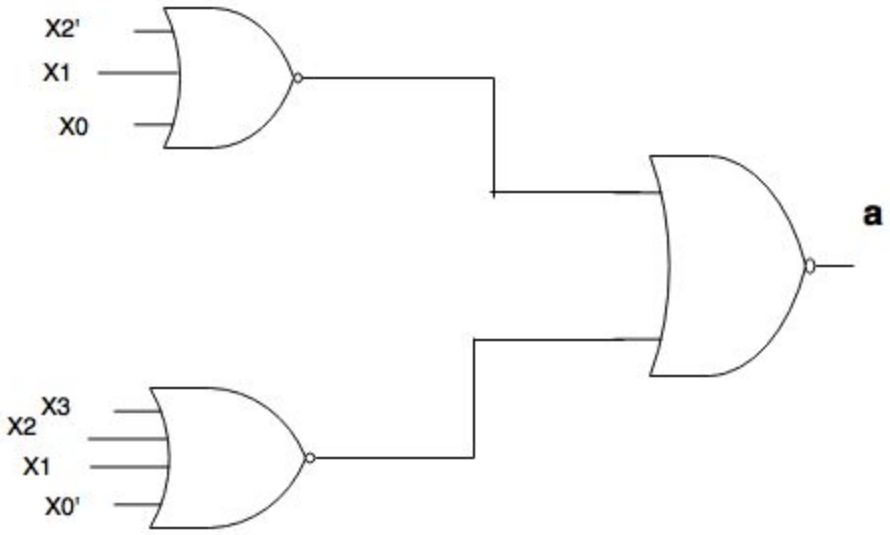
Result	
Correctness	
Creativity	
Report	
Total Score	

Abstract

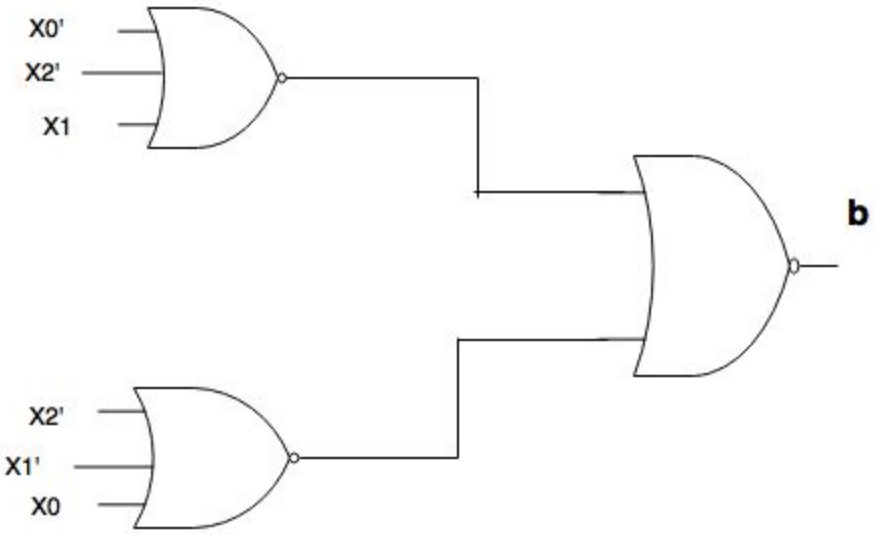
The objective of this project is to build a BCD-to-seven-segment display decoder. This display decoder is normally used in billboards for displaying a decimal digit. It consists of 7 different components whose output are dependent on the input of the decimal digit. The decimal digit is encoded by 4 binary bits. Therefore, for each component, there are 4 inputs denoted by X3,X2,X1 and X0, which dictate whether the component is on (1) or off (0).

The Switching Function of the Circuit

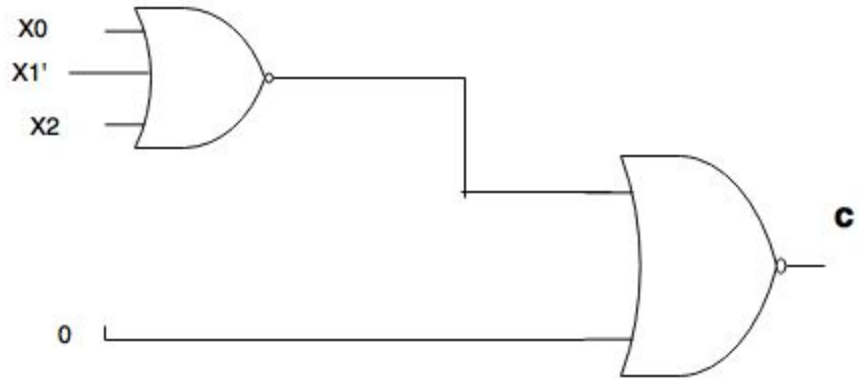
a = ((X2' + X1 + X0)' + (X3 + X2 + X1 + X0')')'



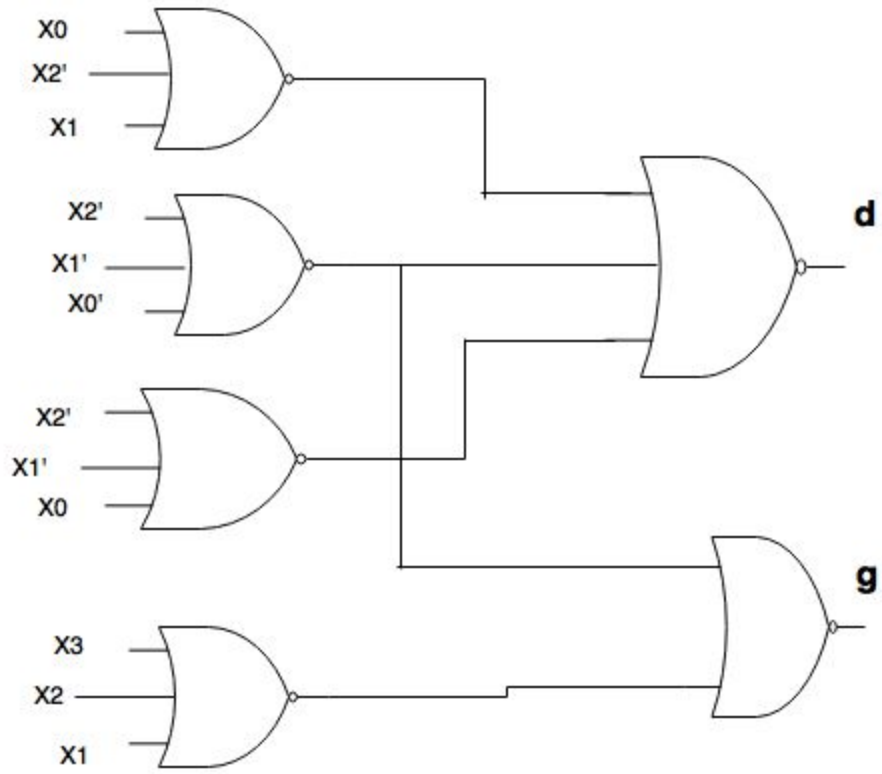
b = ((X2' + X1 + X0')' + (X2' + X1' + X0)')'



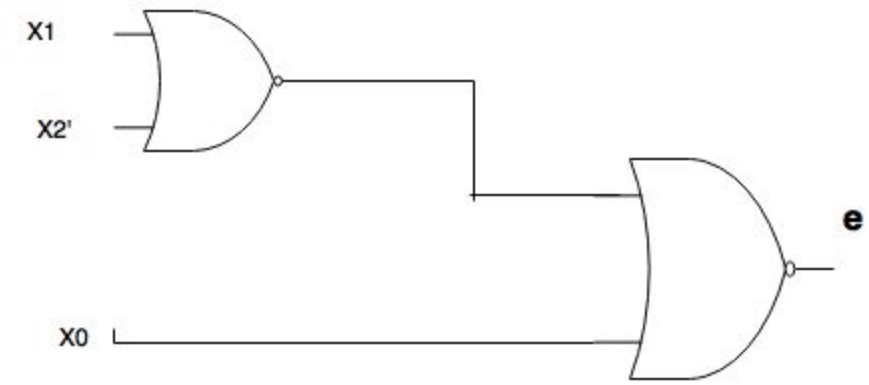
c = ((X2 + X1’ + X0)’ + 0)’



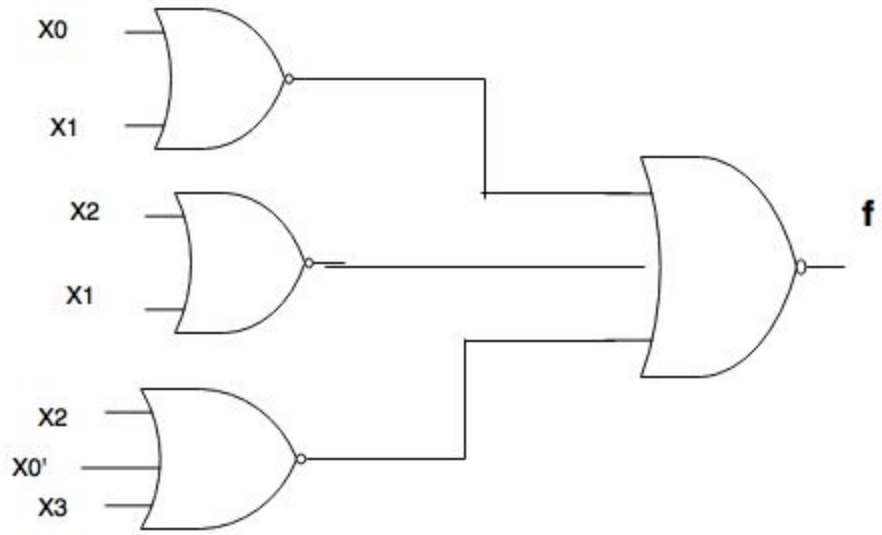
d = ((X2’ + X1 + X0)’ + (X2’ + X1’ + X0’)’ + (X3 + X2 + X1 + X0’)’)’
g = ((X3 + X2 + X1)’ + (X2’ + X1’ + X0’)’)’



e = (X0 + (X2’ + X1)’)’



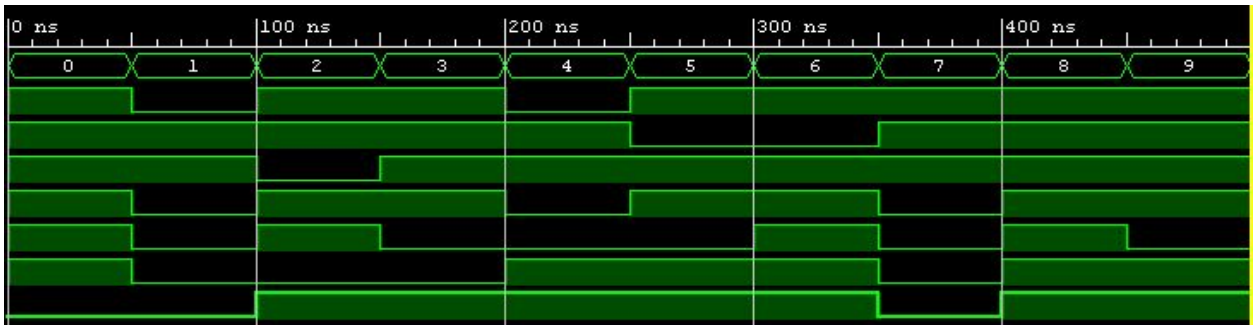
f = ((X1’ + X0’)’ + (X2 + X1’)’ + (X3 + X2 + X0’)’)



The Verilog Code of the Circuit

```
//csm51a_proj2.v
//Michael Wu, ID: 404751542
//Minghong Zhou, ID: 004424670
`timescale 1ns / 1ps
module csm51a_proj2(
     input [3:0]x,
     output a,
     output b,
     output c,
     output d,
     output e,
     output f,
     output g
     );
wire term0=~(~x[2]|~x[1]|~x[0]);
assign a=~(~(~x[2]|x[1]|x[0])|~(x[3]|x[2]|x[1]|~x[0]));
assign b=~(~(~x[2]|x[1]|~x[0])|~(~x[2]|~x[1]|x[0]));
assign c=~(~(x[2]|~x[1]|x[0])|0);
assign d=~(~(~x[2]|x[1]|x[0])|term0|~(x[3]|x[2]|x[1]|~x[0]));
assign e=~(x[0]|~(~x[2]|x[1]));
assign f=~(~(~x[1]|~x[0])|~(x[2]|~x[1])|~(x[3]|x[2]|~x[0]));
assign g=~(~(x[3]|x[2]|x[1])|term0);
endmodule
```

The Simulation Result



This test bench tested the module for the decimal digits 0 to 9. It converts the decimal values to binary, then plugs it into the module. The top row shows the values of the input, and the waveforms below show the output of a,b,c,d,e,f, and g. The first row is a, the second row is b, the third row is c, etc. When the waveform is high it corresponds to an output of 1, and when it is low it corresponds to an output of 0. The waveforms correctly represent the associated truth table that we were trying to implement.

The Design Review

This project was fairly straightforward. We broke the project into two separate steps. First we determined the minimum two level OR-AND switching function for each component from a to g using k-maps. Then we converted these switching functions to NOR-NOR circuit networks. This divide-and-conquer approach simplified the project and minimized the number of mistakes we made in the implementation. In the implementation, we attempted to use the least amount of gates as possible. Therefore, we examined the switching function for each component to see if there were any repeating terms that we could implement with a single NOR gate. We realized that component d and g share the same term (X2' + X1' + X0'). This observation enabled us to reduce the number of gates used by 1.

We learned how to design a module that has practical use in real life. This project taught us to be very careful when implementing a digital network, and helped us practice minimization. The most important aspects of the project for us was the minimization. That was the part we spent the most time on. The actual code did not take very long, because all we had to do was copy the switching functions we wrote with pencil and paper.

Appendix - The detailed design worksheet

inputs and outputs

4 bit input
7 bit output

encoding scheme

input

decimal digit	binary bits
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

output

LED state	binary bit
on	1
off	0

~~truth table~~

~~input~~

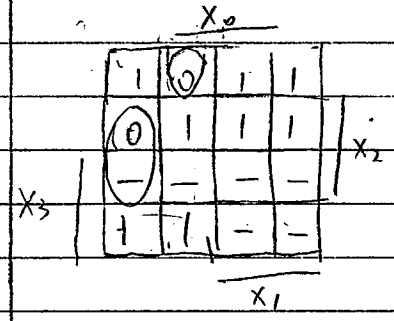
~~output~~

truth table

input	output
$X_3 X_2 X_1 X_0$	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	- - - - - - -
1 0 1 1	- - - - - - -
1 1 0 0	- - - - - - -
1 1 0 1	- - - - - - -
1 1 1 0	- - - - - - -
1 1 1 1	- - - - - - -

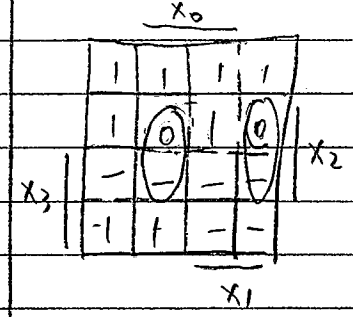
Minimization

a = M(1,4)



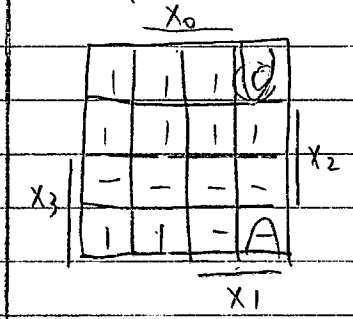
$a = (X_2' + X_1 + X_0)(X_3 + X_2 + X_1 + X_0')$

b = M(5,6)



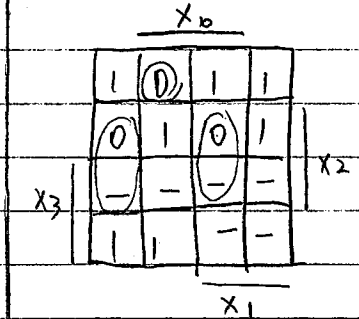
$b = (X_2' + X_1 + X_0')(X_2' + X_1' + X_0)$

c = M(2)



$c = X_3 + X_2 + X_1' + X_0$

d = M(1,4,7)



$d = (X_2' + X_1 + X_0)(X_2' + X_1' + X_0')(X_3 + X_2 + X_1 + X_0')$

Minimization of output g.

Using K-maps

X_3X_2 \ X_1X_0	00	01	11	10
00	①	②	1	1
01	1	1	①	1
11	–	–	②	–
10	1	1	–	–

Prime implicates
$(X_3+X_2+X_1)(X_3'+X_1'+X_0')$

essential prime implicates
$(X_3+X_2+X_1)(X_2'+X_1'+X_0')$

$$g = (X_3+X_2+X_1)(X_2'+X_1'+X_0')$$

Minimization of output f

Using K-maps

X_3X_2 \ X_1X_0	00	01	11	10
00	1	①	②	③
01	1	1	0	1
11	–	–	–	–
10	1	1	–	–

Prime implicates
$(X_2'+X_0')$
(X_3+X_1')
$(X_3+X_2+X_0')$

essential prime implicates

$$f = (X_2'+X_0')(X_3+X_1')(X_3+X_2+X_0')$$

Minimization of output e

Using K-maps

X_3X_2 \ X_1X_0	00	01	11	10
00	1	①	0	1
01	①	0	0	1
11	–	–	–	–
10	1	0	–	–

prime implicates
$(X_0')(X_3'+X_1)$

essential prime implicates
$(X_0')(X_3'+X_1)$

$$e = (X_0')(X_3'+X_1)$$

X_1

transformation procedure

by demorgan's law

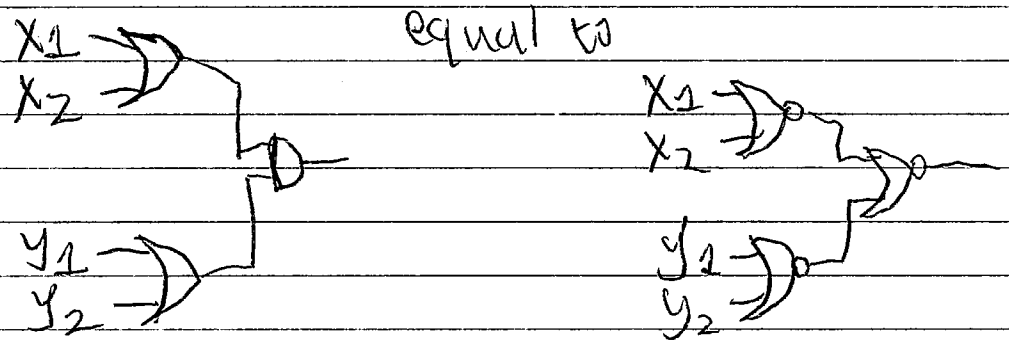
$$X \cdot Y = (X' + Y')'$$

OR-AND to NOR ~ NOR

$X = X_1 + X_2$

$Y = Y_1 + Y_2$

$$(X_1 + X_2)(Y_1 + Y_2) = ((X_1 + X_2)' + (Y_1 + Y_2)')'$$

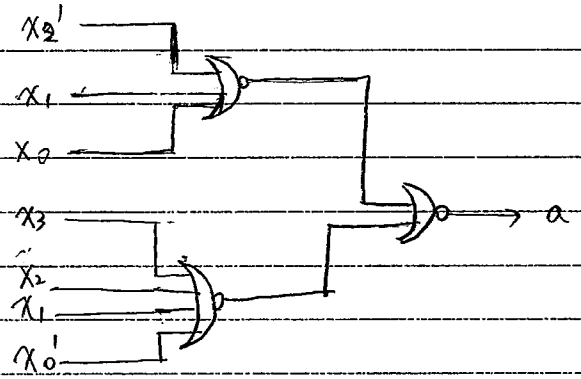


final minimal expressions

a. original

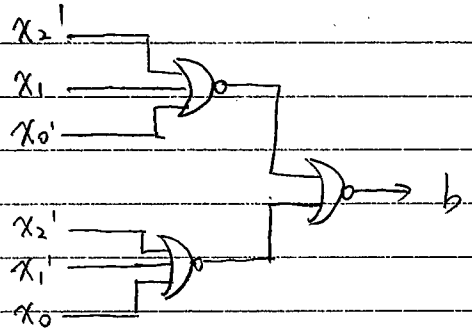
$a = (x_2' + x_1 + x_0)(x_3 + x_2 + x_1 + x_0')$

transformation: $a = ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')')'$



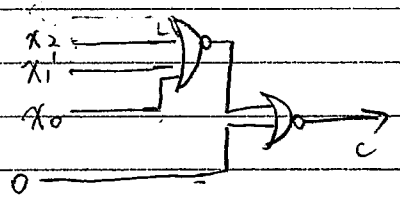
b original $(x_2' + x_1 + x_0')(x_2' + x_1' + x_0)$

transformation $((x_2' + x_1 + x_0')' + (x_2' + x_1' + x_0)')'$



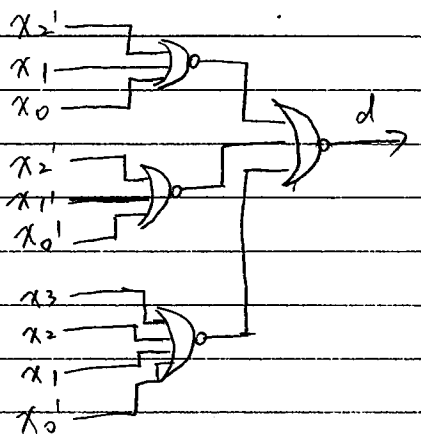
c original $(x_2 + x_1' + x_0)'$

transformation $((x_2 + x_1' + x_0)' + 0)'$



d origional $(x_2' + x_1 + x_0)(x_2' + x_1' + x_0')(x_3 + x_2 + x_1 + x_0')$

transformation $((x_2' + x_1 + x_0)' + (x_2' + x_1' + x_0')' + (x_3 + x_2 + x_1 + x_0')')'$

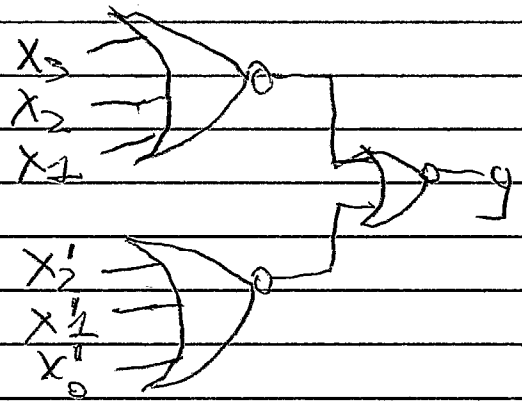


find minimal expressions

Output g

OR-AND $g = (x_3 + x_2 + x_1)(x_2' + x_1' + x_0')$

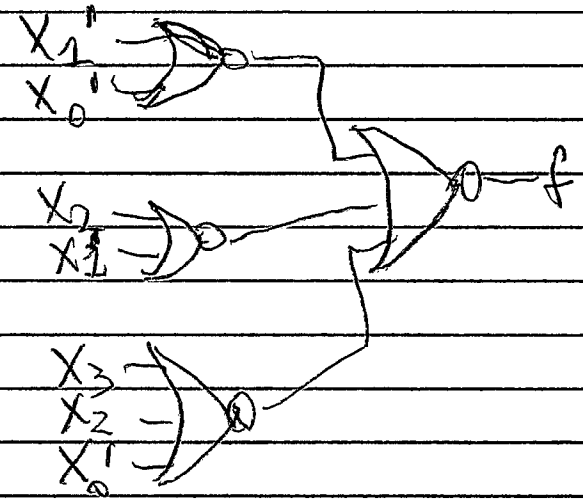
NOR-NOR $g = \left((x_3 + x_2 + x_1)' + (x_2' + x_1' + x_0')'' \right)'$



Output f

OR-AND $f = (x_1' + x_0')(x_2 + x_1')(x_3 + x_2 + x_0')$

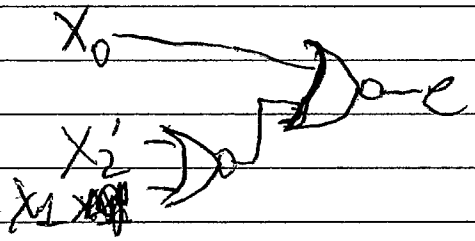
NOR-NOR $f = \left((x_1' + x_0')' + (x_2 + x_1')' + (x_3 + x_2 + x_0')'' \right)'$



Output e

OR-AND $e = (x_0')(x_2' + x_1)$

NOR-NOR $e = (x_0 + (x_2' + x_1)')'$



Extra Credit

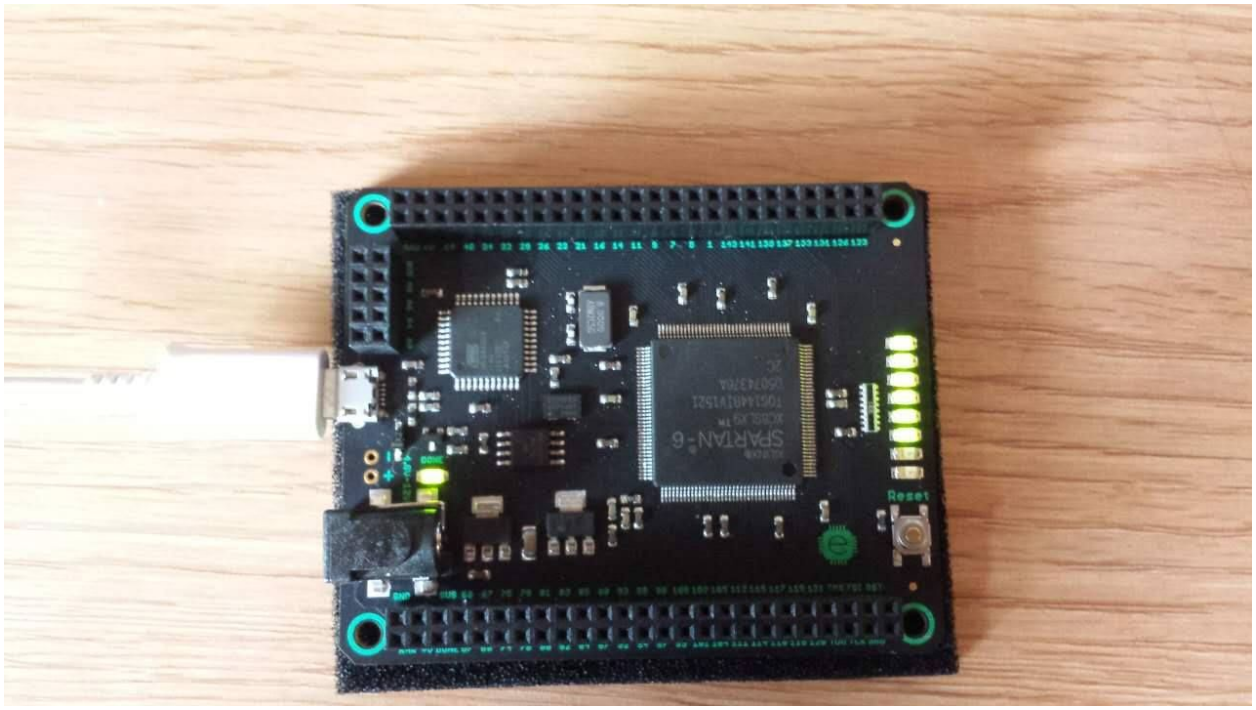
The code for the extra credit is below:

```
module mojo_top(
      // 50MHz clock input
      input clk,
      // Input from reset button (active low)
      input rst_n,
      // clock is high when AVR is ready
      input cclk,
      // We send the outputs to the 8 LEDs
      output[7:0]led,
      // AVR SPI connections
      output spi_miso,
      input spi_ss,
      input spi_mosi,
      input spi_sck,
      // AVR ADC channel select
      output [3:0] spi_channel,
      // Serial connections
      input avr_tx,
      output avr_rx,
      input avr_rx_busy
      );

wire rst = ~rst_n; // make reset active high
reg clk2 = 1'b0;
reg[3:0] x = 4'b000;
reg[23:0] ii = 24'b0;
//every press +1

always@(posedge clk)
begin
      if(ii<12500000)
            begin ii <= ii + 1; end
      else if(ii == 12500000)
            begin ii <= 24'b0 ; clk2 <= ~clk2; end
      end
      //4Hz signal
      always@(posedge clk2)
      begin
            if(rst)
                  begin x <= x + 1; end
```

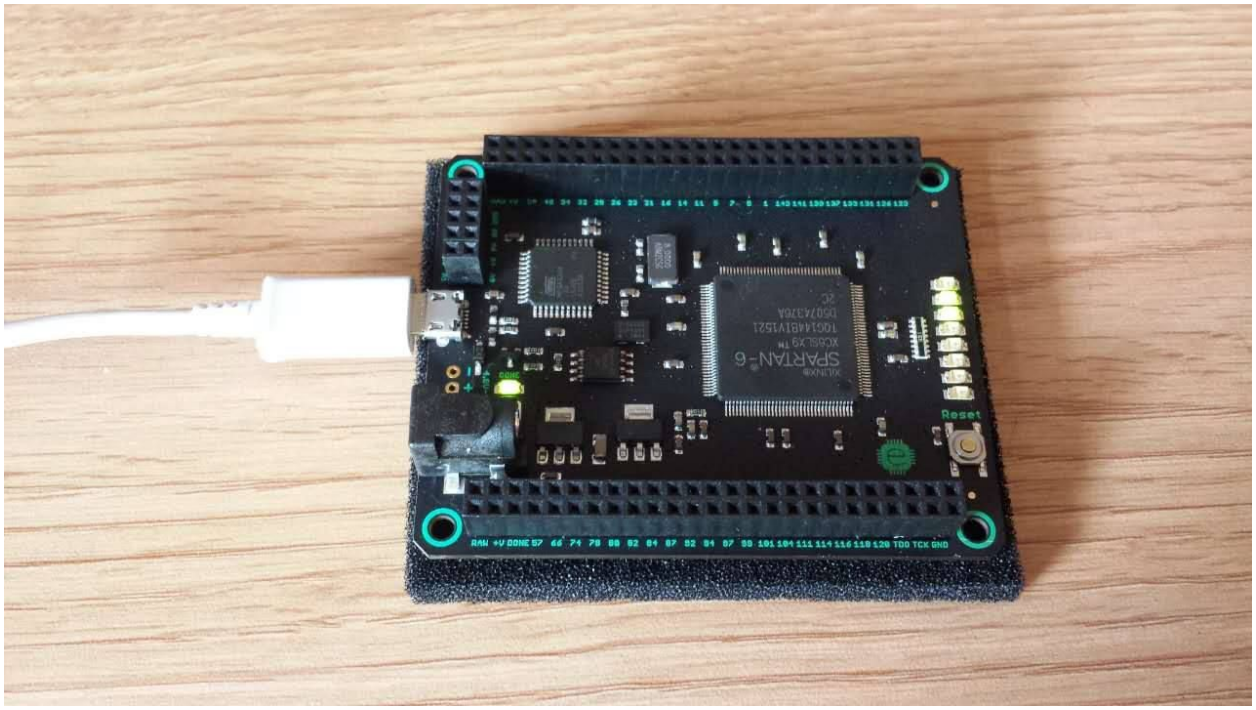
```
      end
      // these signals should be high-z when not used
      assign spi_miso = 1'bz;
      assign avr_rx = 1'bz;
      assign spi_channel = 4'bzzzz;
      assign led[7]=1'b0;
      assign led[6] =~(~(x[3]|x[2]|x[1])|~(~x[2]|~x[1]|~x[0]));
      assign led[5]
=~(~(~x[1]|~x[0])|~(x[2]|~x[1])|~(x[3]|x[2]|~x[0]));
      assign led[4] =~(~(x[3]|~x[0])|~(x[2]|~x[0])|~(~x[2]|x[1]));
      assign led[3]
    =~(~(x[3]|x[2]|x[1]|~x[0])|~(x[3]|~x[2]|x[1]|x[0])|~(x[3]|~x[2]|~
    x[1]|~x[0]));
      assign led[2] =~(~(x[3]|x[2]|~x[1]|x[0])|0);
      assign led[1]
    =~(~(x[3]|~x[2]|x[1]|~x[0])|~(x[3]|~x[2]|~x[1]|x[0]));
      assign led[0] =~(
    ~(x[3]|x[2]|x[1]|~x[0])|~(x[3]|~x[2]|x[1]|x[0]));
endmodule
```

The outputs of the board are the leds on the right side, and the first 7 correspond to a,b,c,d,e,f, and g from top to bottom. The last led on the bottom is not used and is always off. Below are the pictures of the mojo board:

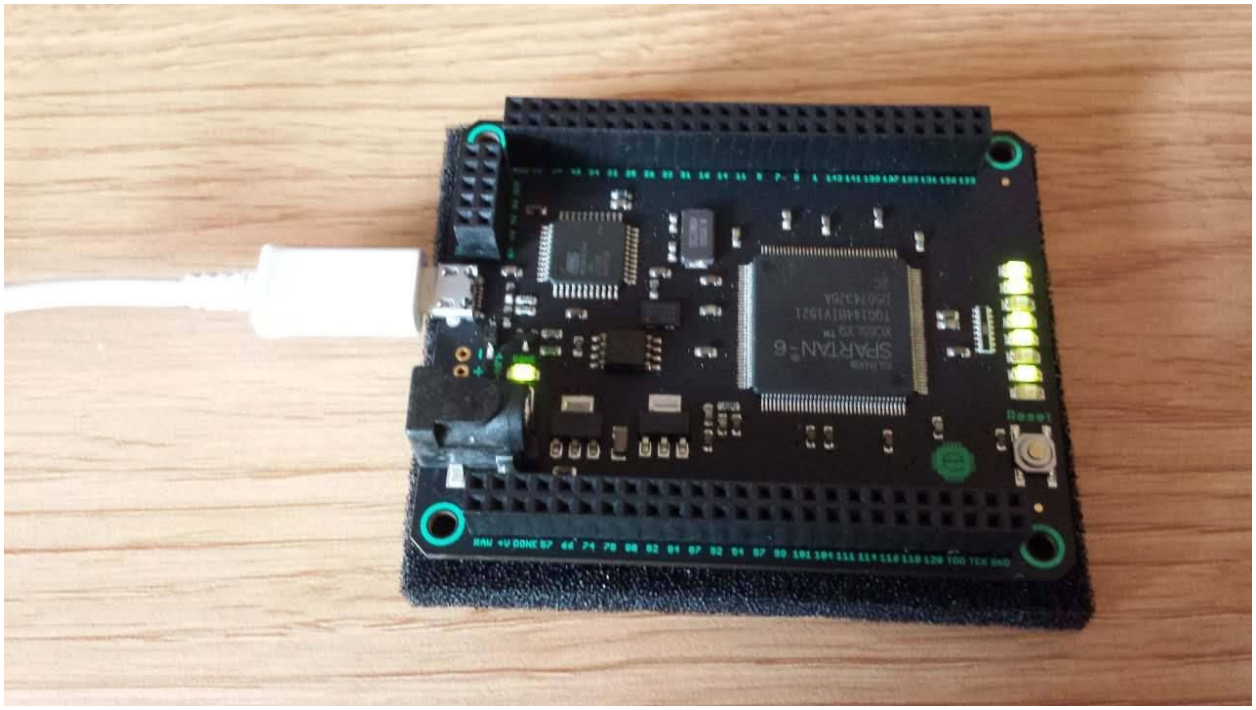
Input: 0



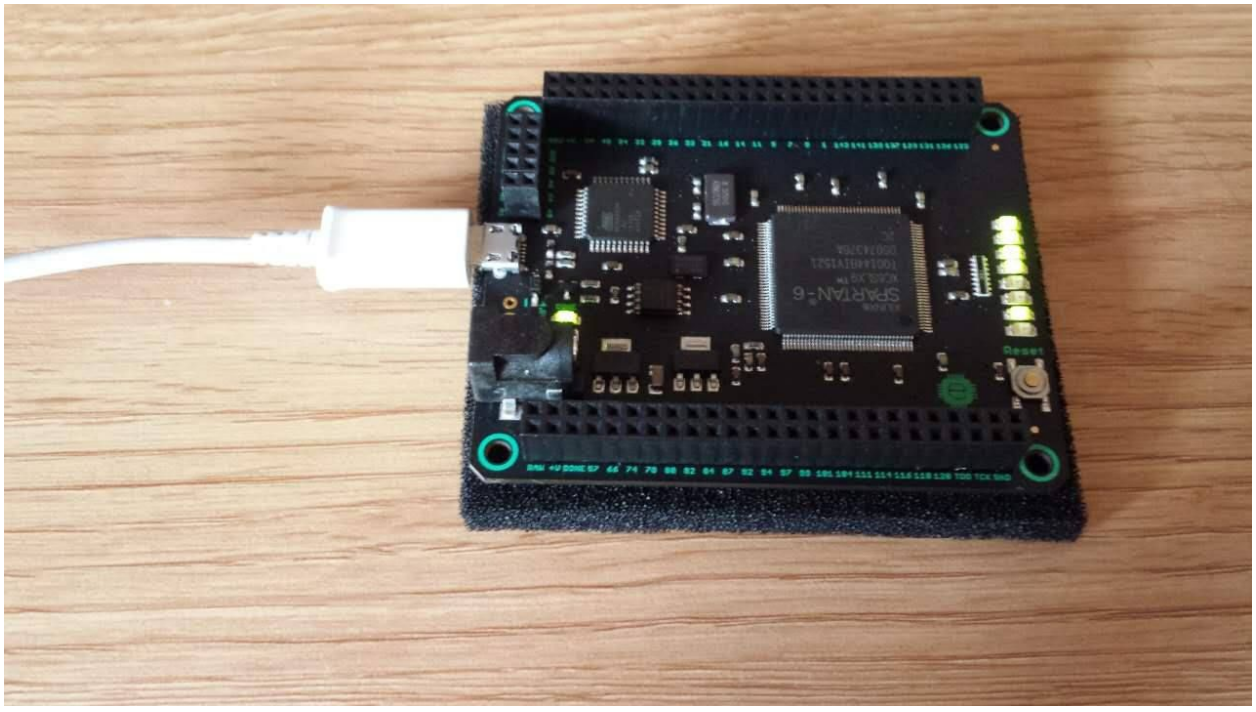
Input: 1



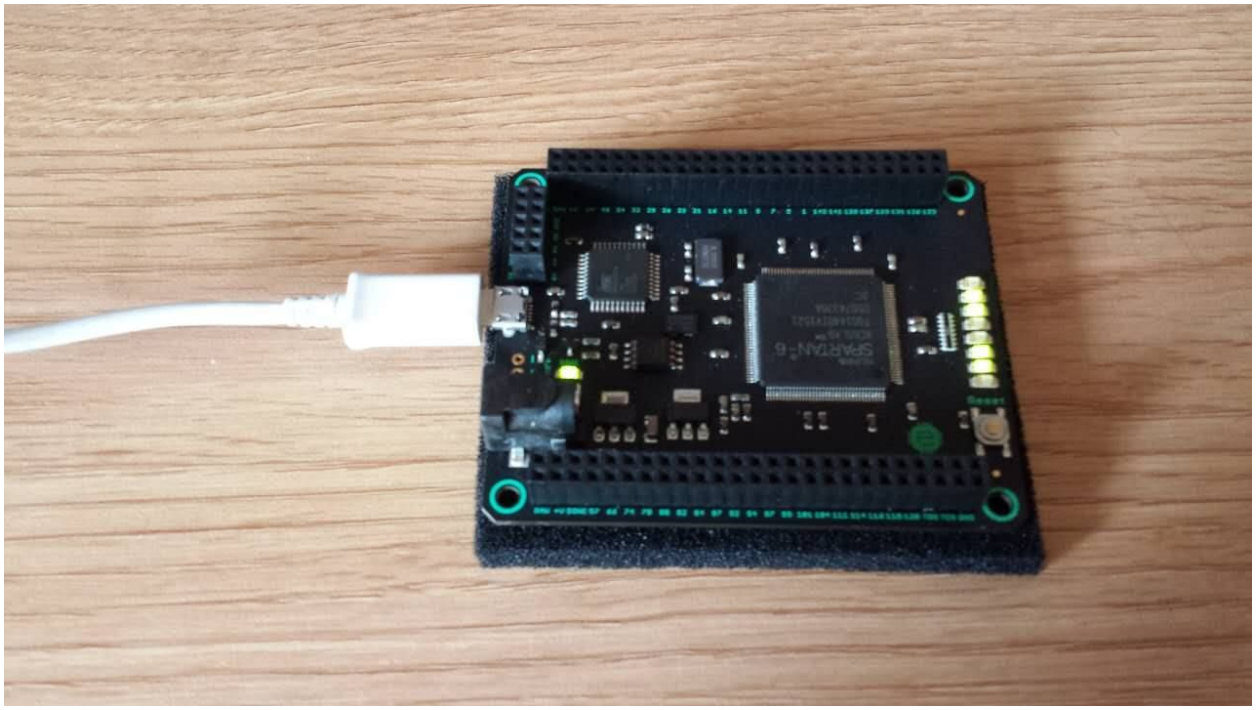
Input: 2



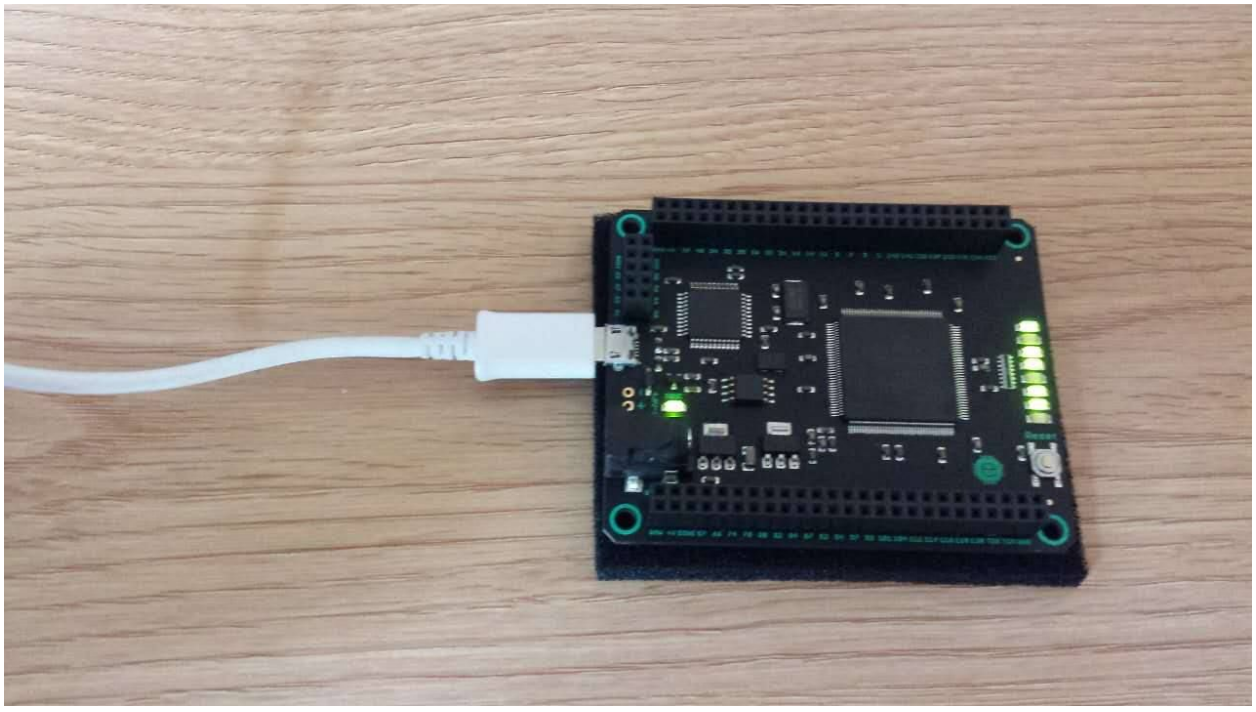
Input: 3



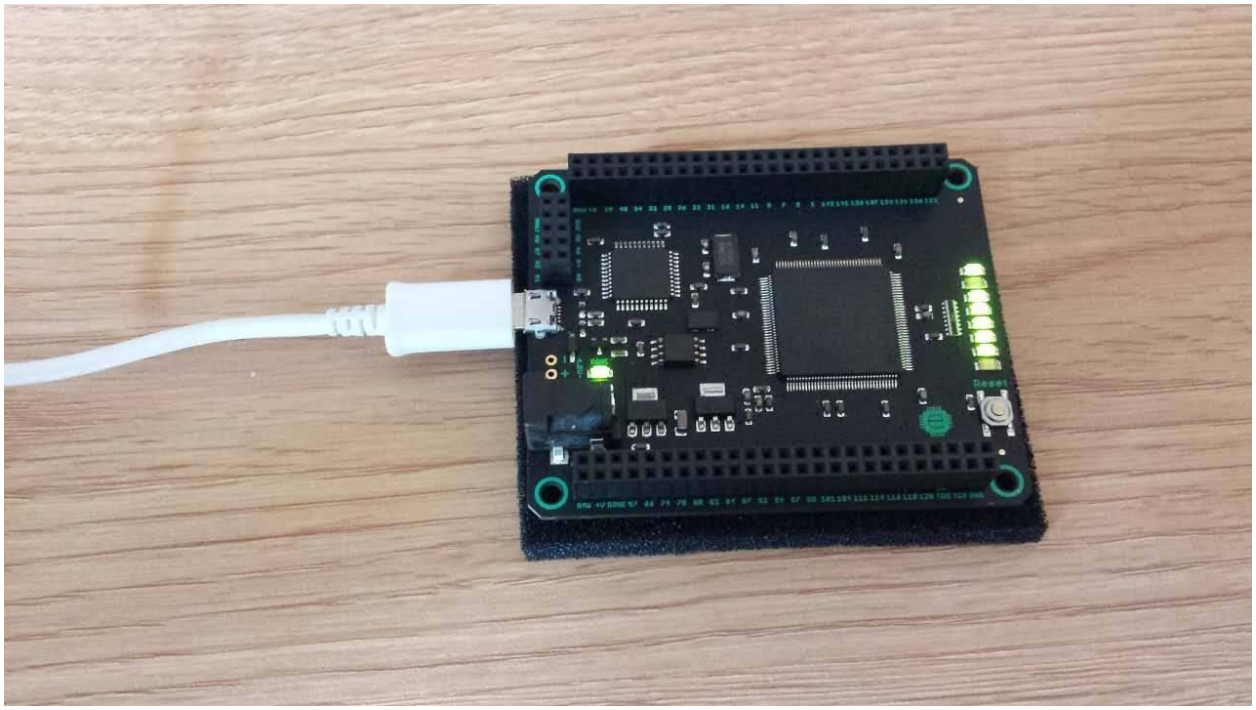
Input: 4



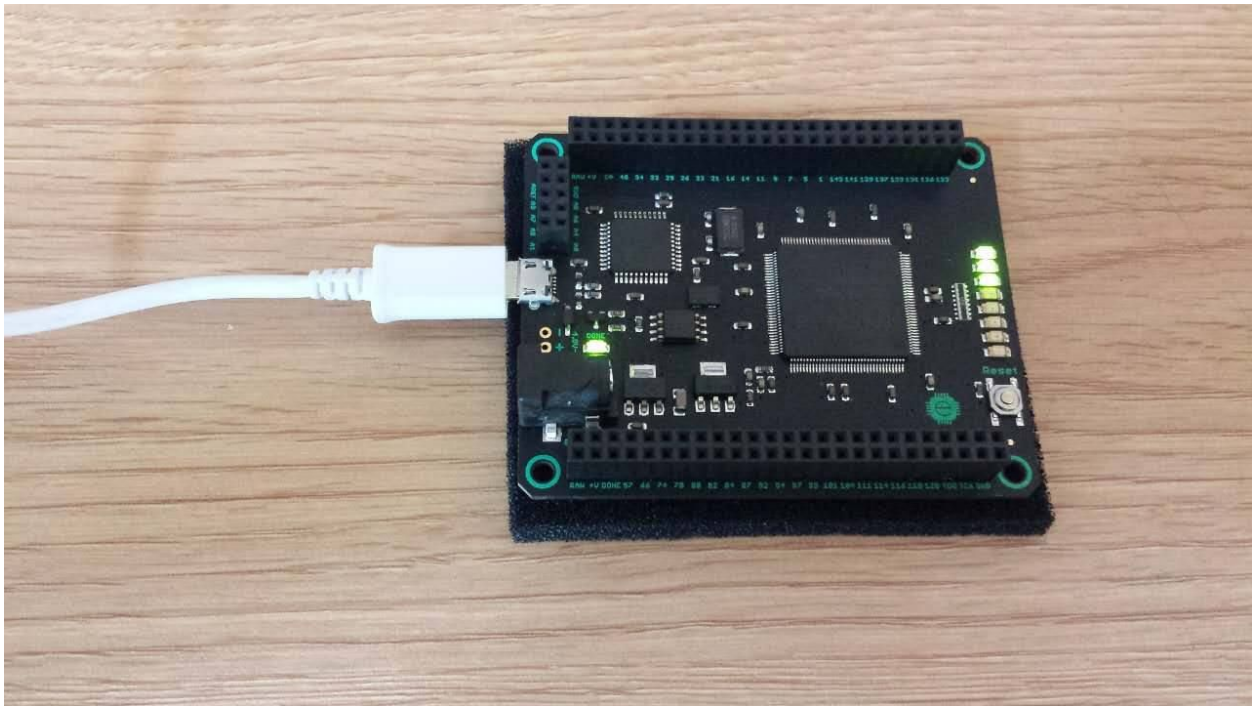
Input: 5



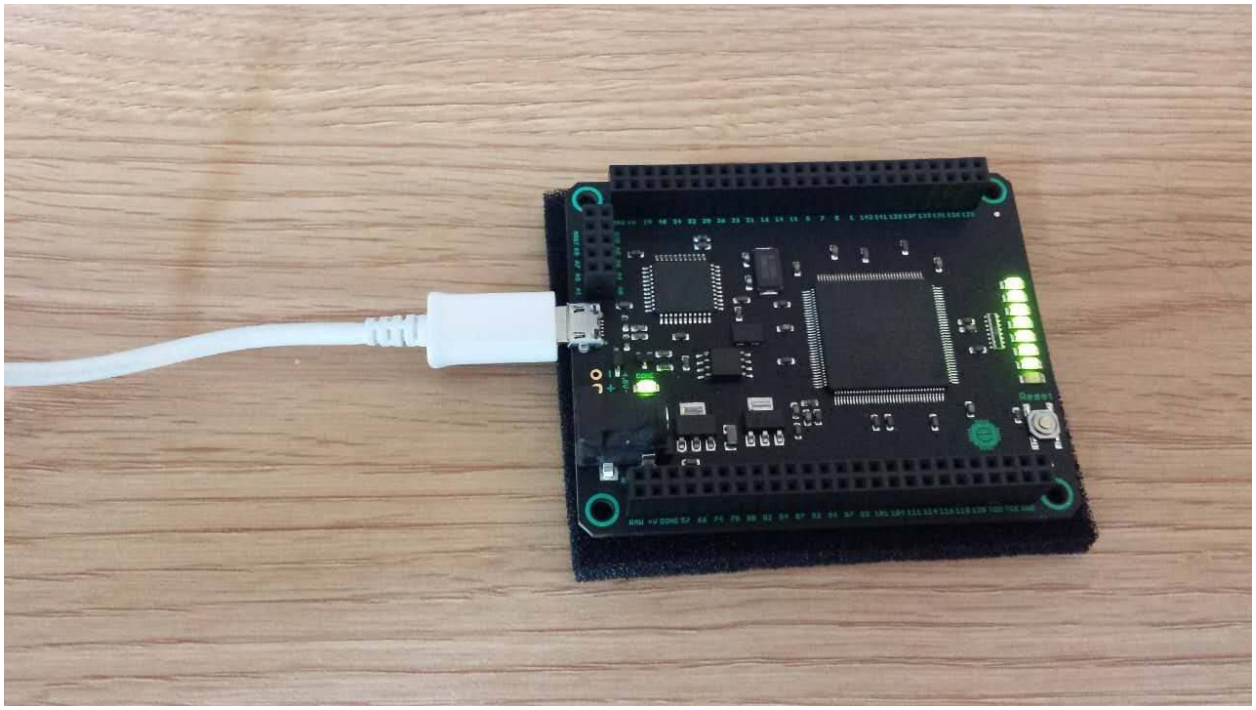
Input: 6



Input: 7



Input: 8



Input: 9