Exercise 11: How could you modify the code so that **digits** is only updated when an **enable** input is asserted?

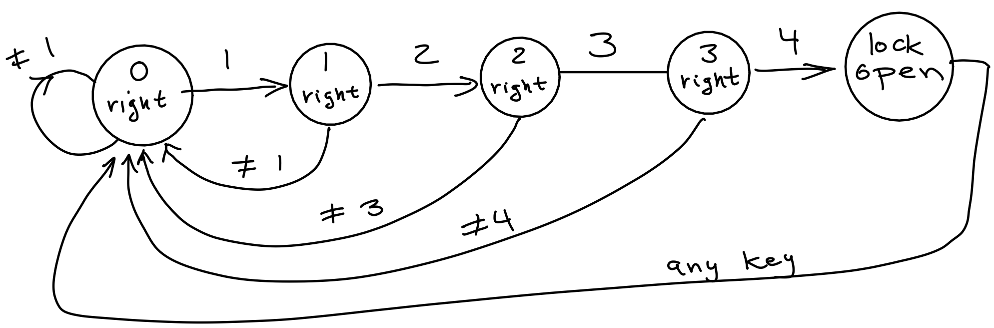
```
always_ff @(posedge clk) digits
   <= { digits[1:3], digit } ;
```

```
always_ff @(posedge clk) digits
 <= enable ? { digits[1:3], digit } : digits;
```

Exercise 12: How many states can this state machine have?

There are 10 possible values for each digit so there are $10 \times 10 \times 10 \times 10 = 10^4 = 10{,}000$ possible states.

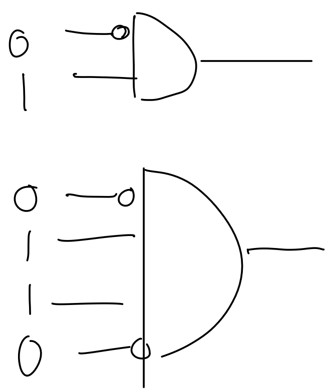
Exercise 13: Draw the state transition diagram for this simpler implementation. How many states are there? Write the Verilog using a 3-bit **count** state variable.



5 states

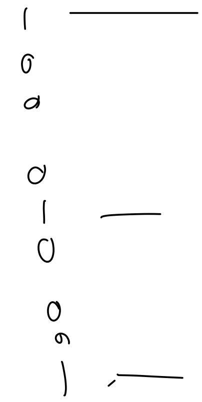
```
logic [2:0] count;
always_ff @(posedge clk)
  count <= count==0 && digit==1 ? 1 :
           count==1 && digit==2 ? 2 :
           count==2 && digit==3 ? 3 :
           count==3 && digit==4 ? 4 : 0;

assign unlock = count==4;
```

Exercise 14: How much logic is required to detect a state when a binary encoding is used? With a one-hot encoding?



need n-bit AND to detect a state for 2^n states

binary encoding



no logic needed.

+ encoding

Exercise 15: If we used 8-bits (w/ binary encoding) of state information, how many states could be represented? What if we used 8 bits of state but used a "one-hot" encoding?

binary: $2^8 = 256$

0000 0000
0000 0001
⋮
1111 1111
} 256

one-hot: 8

1000 0000
⋮
0000 0001