

# ECE M16 Final

Lawrence Liu

August 11, 2022

## Problem 1

	1	1	0	1
1	11	00	00	10
	1			
101	10	00		
	1	01		
1100	11	00		
	0			
	11	00	10	
11001	1	10	01	
	1	10	01	

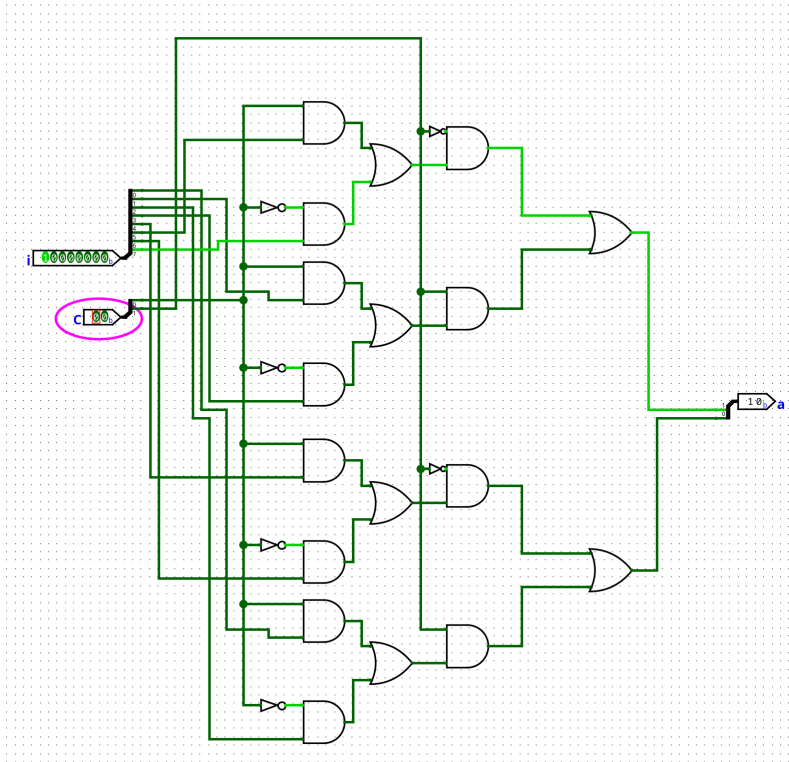
## Problem 2

Basing on the assumption that  $c[1 : 0] = 11$  corresponds with  $o[1 : 0] = i[1 : 0]$  we have that

$$\begin{aligned} a[1] &= \overline{c[1]}. \overline{c[0]}. i[7] + \overline{c[1]}. c[0]. i[5] + c[1]. \overline{c[0]}. i[3] + c[1]. c[0]. i[1] \\ &= \overline{c[1]}. (\overline{c[0]}. i[7] + c[0]. i[5]) + c[1]. (\overline{c[0]}. i[3] + c[0]. i[1]) \end{aligned}$$

$$\begin{aligned} a[0] &= \overline{c[1]}. \overline{c[0]}. i[6] + \overline{c[1]}. c[0]. i[4] + c[1]. \overline{c[0]}. i[2] + c[1]. c[0]. i[0] \\ &= \overline{c[1]}. (\overline{c[0]}. i[6] + c[0]. i[4]) + c[1]. (\overline{c[0]}. i[2] + c[0]. i[0]) \end{aligned}$$

Which results in a circuit like this



## Problem 3

I created the circuit, and it is shown above, and I tested it with the following python checker script

```

1 import numpy as np
2 import pandas as pd
3 import os
4 from calendar import monthrange
5
6 def RunCircuit(logisim_jar : str, circuit : str):
7     """
8     This function runs the logisim simulator and returns the output of
9     the circuit as
10    a pandas dataframe.
11    """
12    output=os.popen(f"java -jar {logisim_jar} {circuit} -tty table").
13    read()
14    output=[o.split() for o in output.split("\n")[:-1]]
15    return pd.DataFrame(output[1:], columns=output[0])
16
17 def checkQ2(truth_table:pd.DataFrame)->bool:
18     """
19     This function checks the output of the circuit for the truth table
20     and returns
21     weather the output is correct or not.

```

```

19     """
20     #convert hex to binary
21     truth_table['i']=truth_table['i'].apply(lambda x: f'{int(x,16):0>8b}
22     ')
23     for i,row in truth_table.iterrows():
24         c=int(row.C,2)
25         i=row.i
26         a=row.a
27         #calculate a expected
28         a_expected=i[c*2:c*2+2]
29         #check if a is equal to a_expected
30         if a!=a_expected:
31             print("error!")
32             print(f"at c={row.C}")
33             print(f"expected a={a_expected}")
34             print(f"got a={a}")
35             print(f"i={row.i}")
36             return False
37         return True
38
39 if __name__=="__main__":
40     truth_table=RunCircuit("../logisim-evolution.jar","Logisim/FinalQ3.
41     circ")
42     if checkQ2(truth_table):
43         print("Q2 passed!")

```

This script utilizes Logisim's command line ability. I had the files in the following format

ECEM16

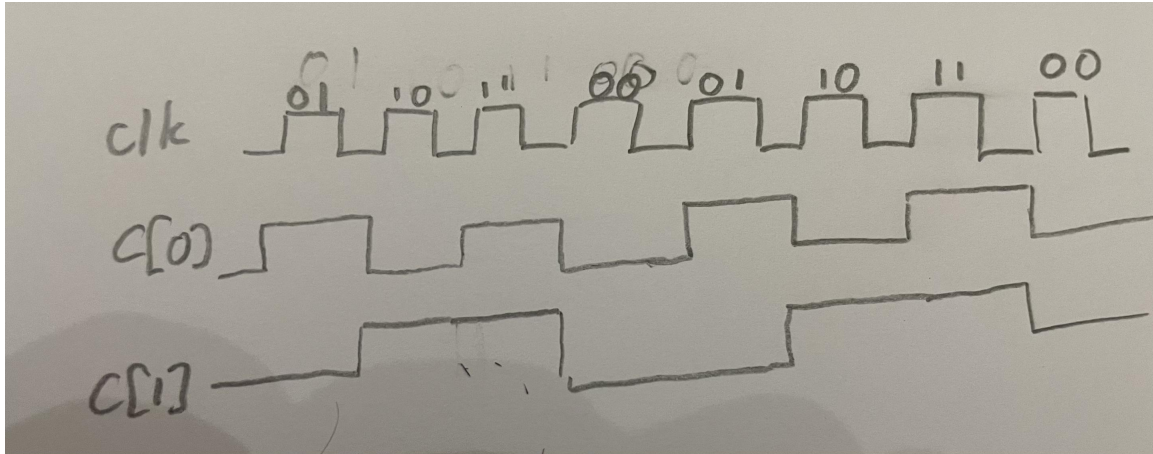
```

|- .gitignore
|- Final
| |- Logisim
| | |- FinalQ3.circ
| :
| :
| |- checker.py
|- .gitignore
|- logisim-evolution.jar

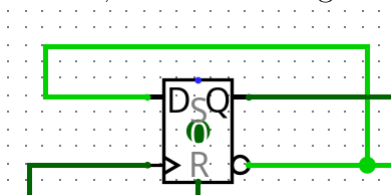
```

## Problem 4

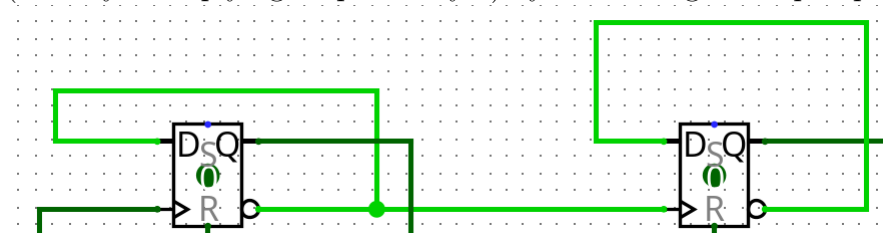
Let  $c[1 : 0]$  be the desired output from our counter: we want the following timing diagram:



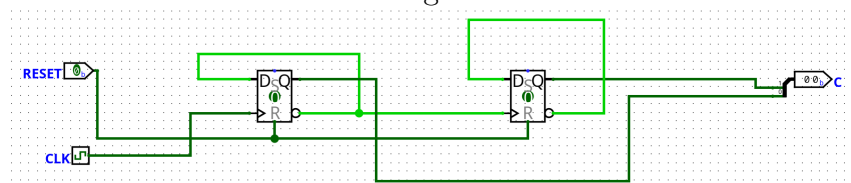
The first thing we notice is that  $c[0]$  is just the clock input but with a period twice of clock, and  $c[1]$  is just the clock input with a period of 4 times clock. We can make the output from a circuit toggle with each clock pulse (ie doubling its frequency), by connecting  $\overline{Q}$  with  $D$ , in the following manner:



Likewise we can make the output from a circuit toggle with every other clock pulse (thereby multiplying its period by 4) by connecting two flip flops in the following manner:

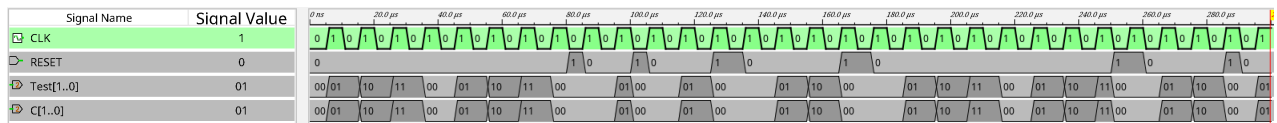


Therefore we have the following circuit:



## Problem 5

I created the circuit, and it is shown above, and I tested by comparing its output vs that of a logisim counter, its output is denoted as test in the chronograph below.



As one can see, the chronograph outputs of my counter vs logisim's counter is the same.

## Problem 6

We can break it up into two circuits, one for the GO signal with inputs of REQ and DONE, and one for the ACK signal with an input of DONE. For the ACK circuit we have the following transition table:

Current State DONE,ACK	input (DONE)	Next State
0,0	0	0
0,0	1	1
0,1	0	1
0,1	1	0
1,1	0	0
1,1	1	1
1,0	0	1
1,0	1	0

This can be implemented with a flip flop with the clock input as DONE and the output  $Q$  as ACK, and  $\overline{Q}$  being fed into  $D$ . The transition table for the GO circuit is the following:

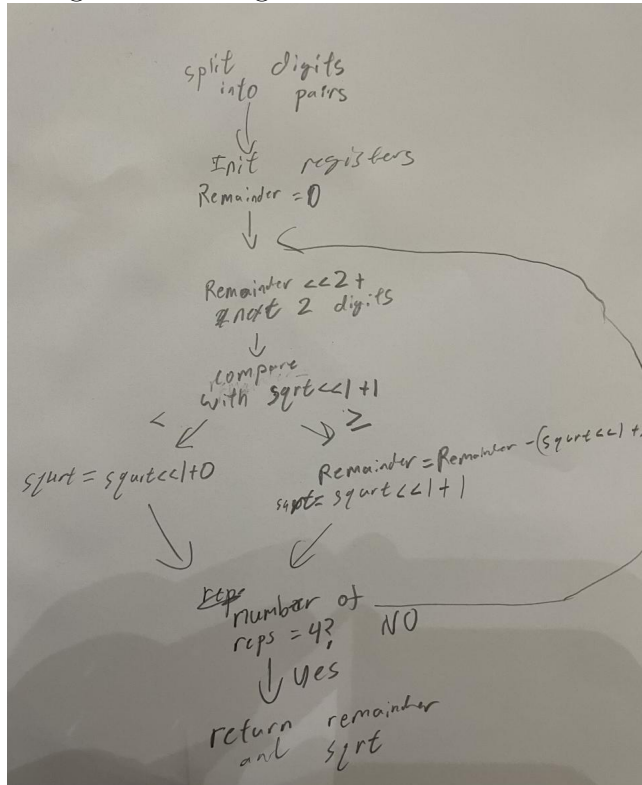
Current State REQ,DONE,GO	input (REQ,DONE)	Next State (GO)
0,0,x	0,x	0
0,0,x	1,x	1
0,1,x	0,x	0
0,1,x	1,x	1
1,0,x	0,x	1
1,0,x	1,x	1
x,0,1	x,0	1
x,0,1	x,1	0
0,1	0	1
0,1	1	0
1,1	0	0
1,1	1	1
1,0	0	1
1,0	1	0

When constructing this transition tables, I made the following assumptions, that immediately after DONE goes from 0 to 1, the GO signal must be dropped, as it appears on the example timing diagram.

## Problem 7

## Problem 8

Using the following flow chart



I created the circuit

