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

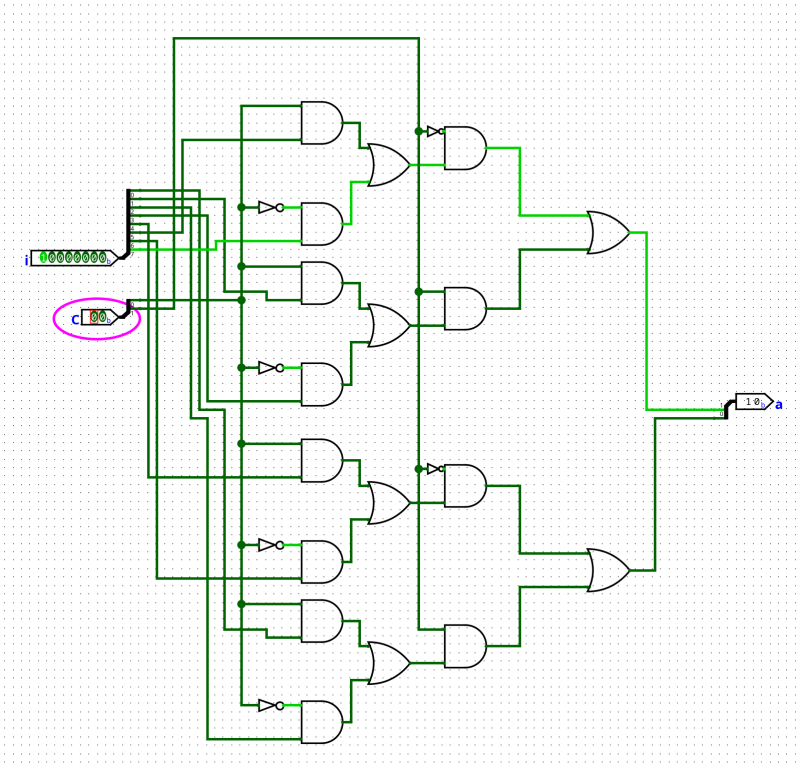
Problem 2

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 srcipt

```python
import numpy as np
import pandas as pd
import os
from calendar import monthrange

def RunCircuit(logisim_jar  : str, circuit : str):
    """
    This function runs the logisim simulator and returns the output of the circuit as
    a  pandas dataframe.
    """
    output=os.popen(f"java -jar {logisim_jar} {circuit} -tty table").read()
    output=[o.split() for o in output.split("\n")[:-1]]
    return pd.DataFrame(output[1:],columns=output[0])

def checkQ2(truth_table:pd.DataFrame)->bool:
    """
    This function checks the output of the circuit for the truth table and returns
    weather the output is correct or not.
```

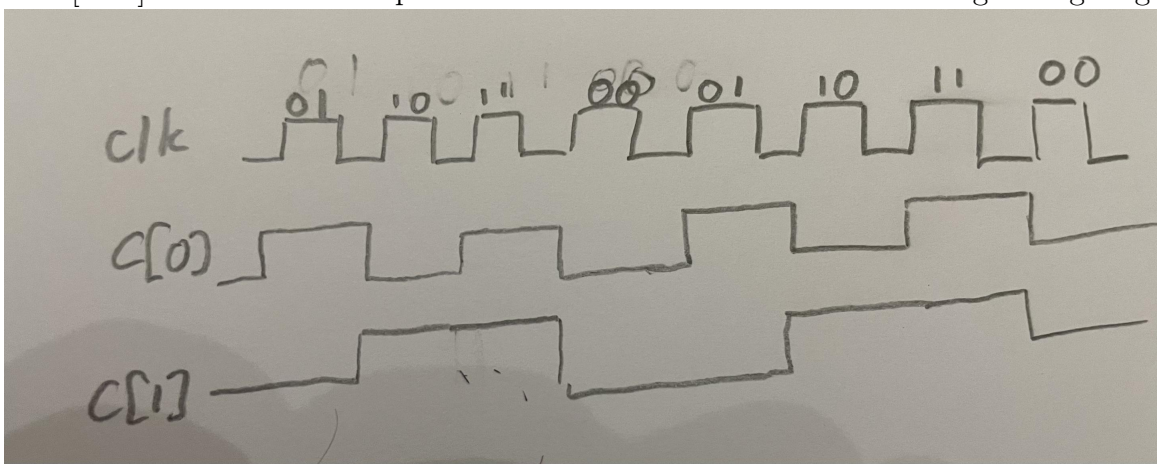
```
    """
    #convert hex to binary
    truth_table['i']=truth_table['i'].apply(lambda x: f'{int(x,16):0>8b}')
    for i,row in truth_table.iterrows():
        c=int(row.C,2)
        i=row.i
        a=row.a
        #calculate a expected
        a_expected=i[c*2:c*2+2]
        #check if a is equal to a_expected
        if a!=a_expected:
            print("error!")
            print(f"at c={row.C}")
            print(f"expected a={a_expected}")
            print(f"got a={a}")
            print(f"i={row.i}")
            return False
    return True

if __name__=="__main__":
    truth_table=RunCircuit("../logisim-evolution.jar","Logisim/FinalQ3.circ")
    if checkQ2(truth_table):
        print("Q2 passed!")
```

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

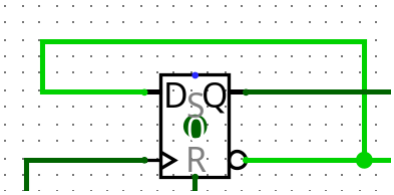
```
ECEM16
 |- .gitignore
 |- Final
 | |- Logisim
 | | |- FinalQ3.circ
 | :
 | :
 | |- checker.py
 |- .gitignore
 |- logisim-evolution.jar
```

Problem 4

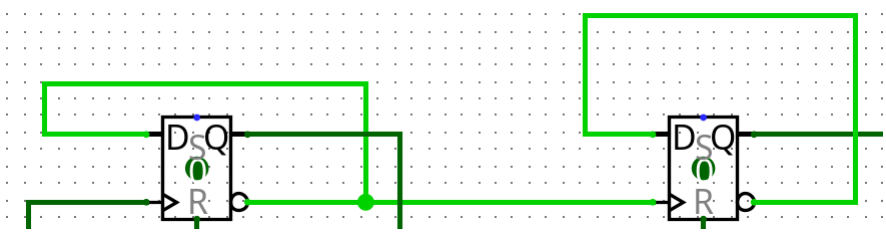
Let $c[1:0]$ be the desired output from our counter: we want the following timing diagram:



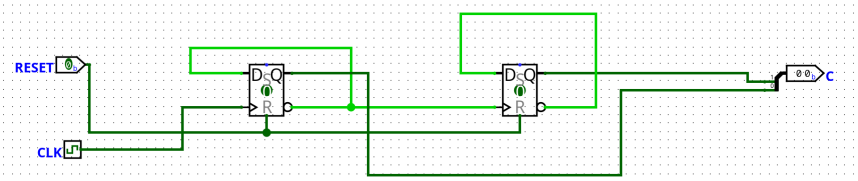
The first thing we notice is that $c[0]$ is just the clock input but with a period twice of clock, and $c[1]$ is just the clock input with a period of 4 times clock. We can make the output from a circuit toggle with each clock pulse (ie doubling its frequency), by connecting $\overline{Q}$ with D, in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

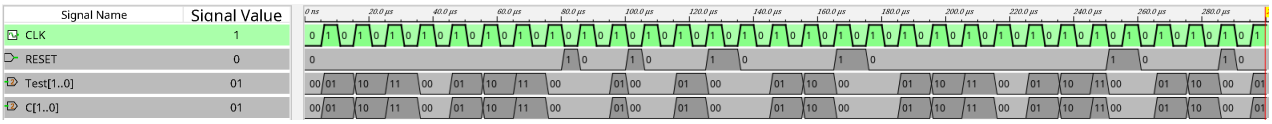


Therefore we have the following circuit:



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.



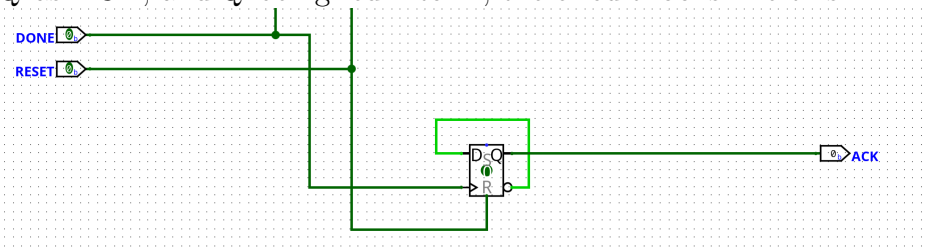
As one can see, the chronograph outputs of my counter vs logisim's counter is the same.

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

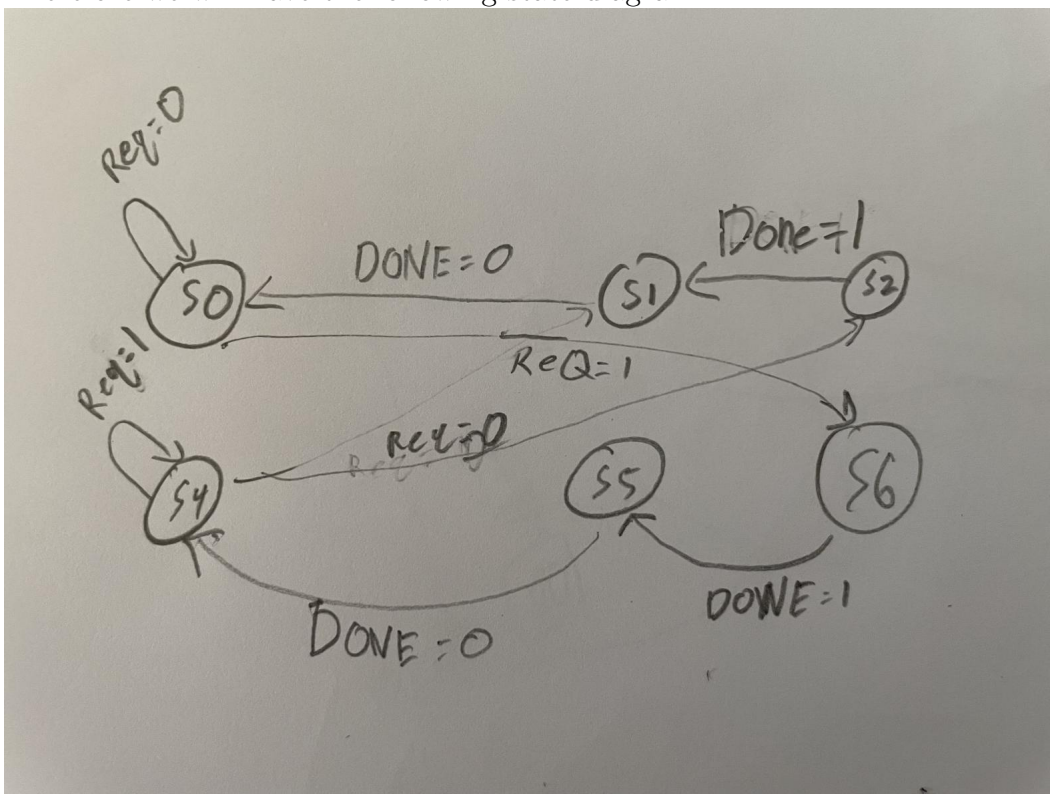
This can be implemented with a flip flop with the clock input as DONE and the output Q as ACK, and $\overline{Q}$ being fed into D, the circuit looks like this



When constructing the state diagram, 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. We have that the following states corresponds with the following combinations of REQ GO and DONE

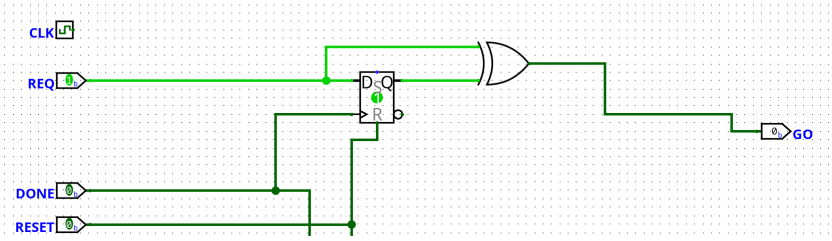
State	REQ GO DONE
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

S3 and S7 cannot happen since we assumed that when DONE goes to 1 GO drops to 0. Therefore we will have the following state diagram.



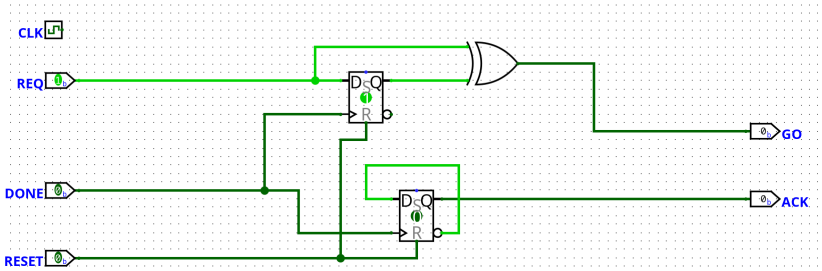
A thing to notice, is that effectively we want GO's output to go to 1 every time REQ signal switches. This can be done by connecting the REQ signal with the input to a D-flip flo, and XORing the output of the flip flop with REQ.

We also want GO to drop to zero every time we receive a DONE signal. This can be easily done by having the DONE input being connected to the clock input of the aforementioned D-flip flop, resulting in a circuit like this.

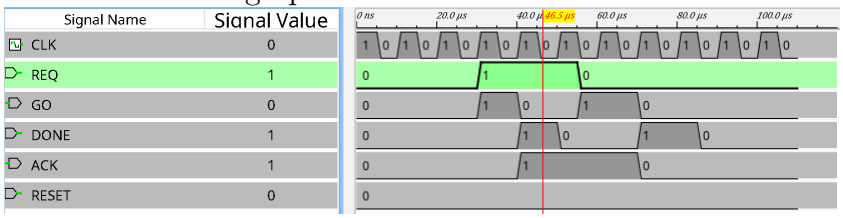


Problem 7

The whole circuit looks like this

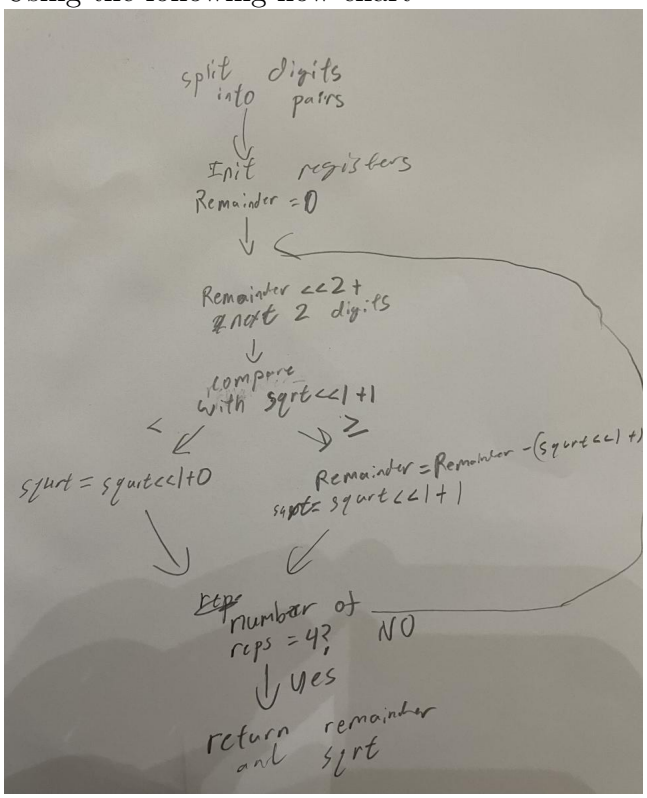


And has a chronograph of:

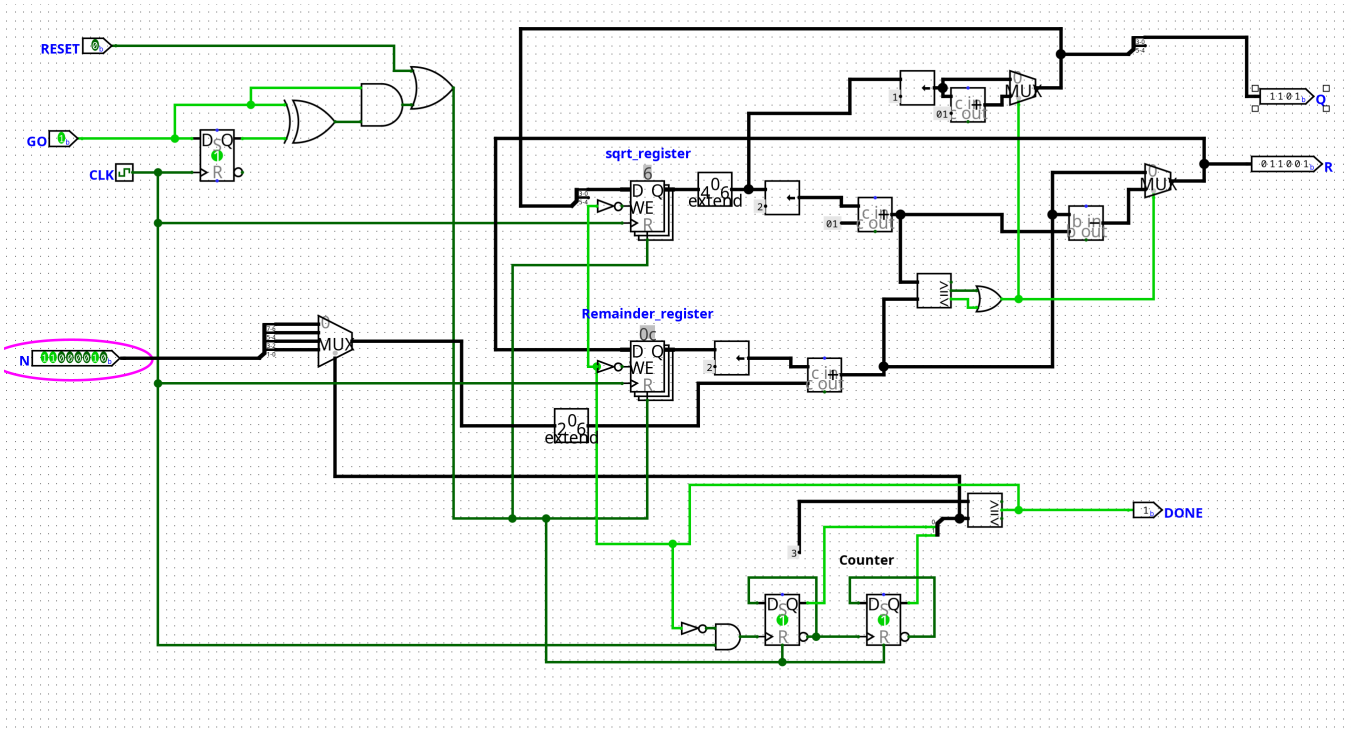


Problem 8

Using the following flow chart



I created the circuit


RESET
GO
CLK
N
sqrt_register
Remainder_register
extend
MUX
DONE
Counter
Q
R