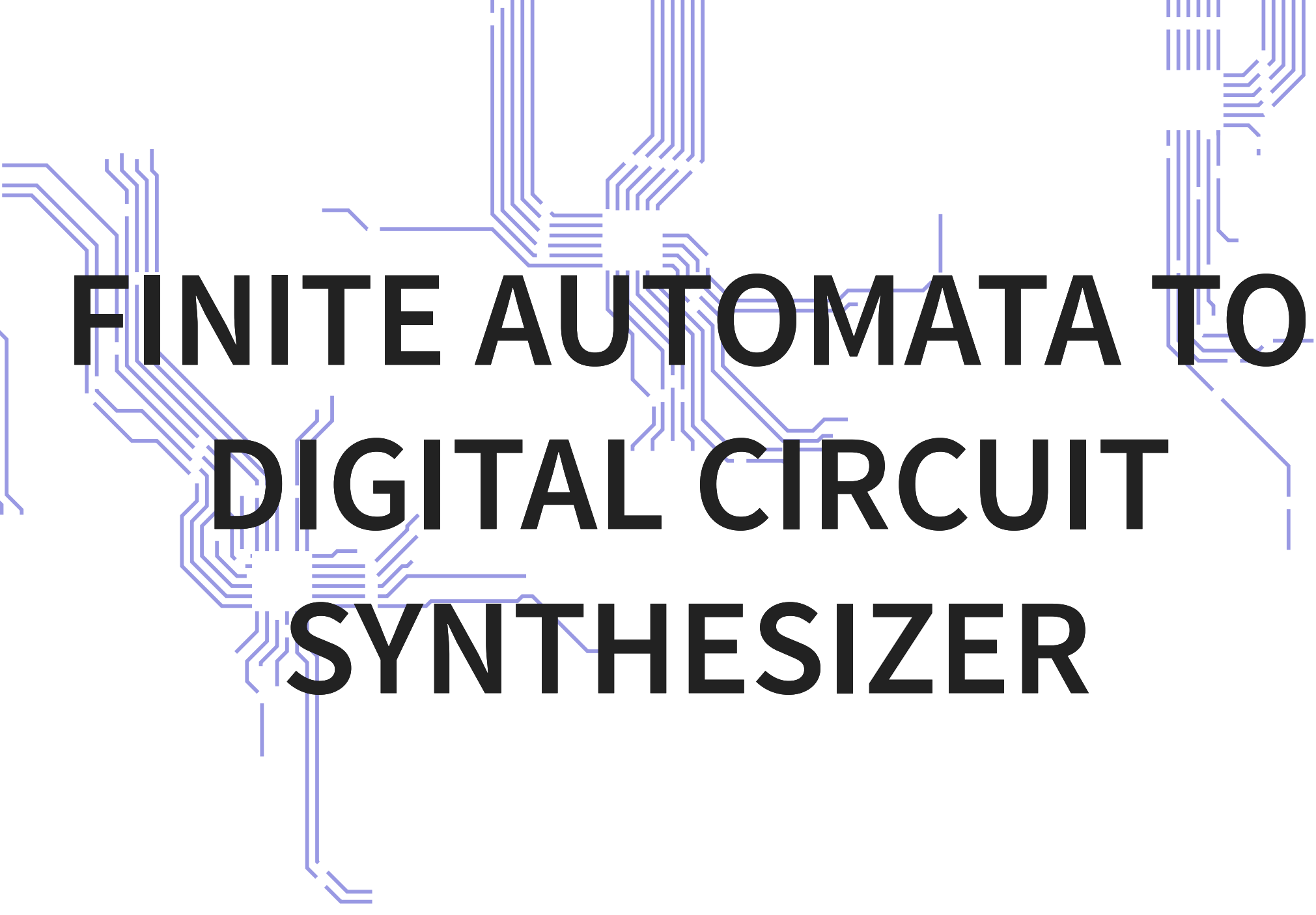


FINITE AUTOMATA TO DIGITAL CIRCUIT SYNTHESIZER

MEMBERS

- 1902085 Dheeraj Lalwani
- 1902086 Chirag Lulla
- 1902168 Kartik Soneji
- 1902184 Aayush Wadhwani

Group Number: 24

Guide: Prof. Sakshi Surve

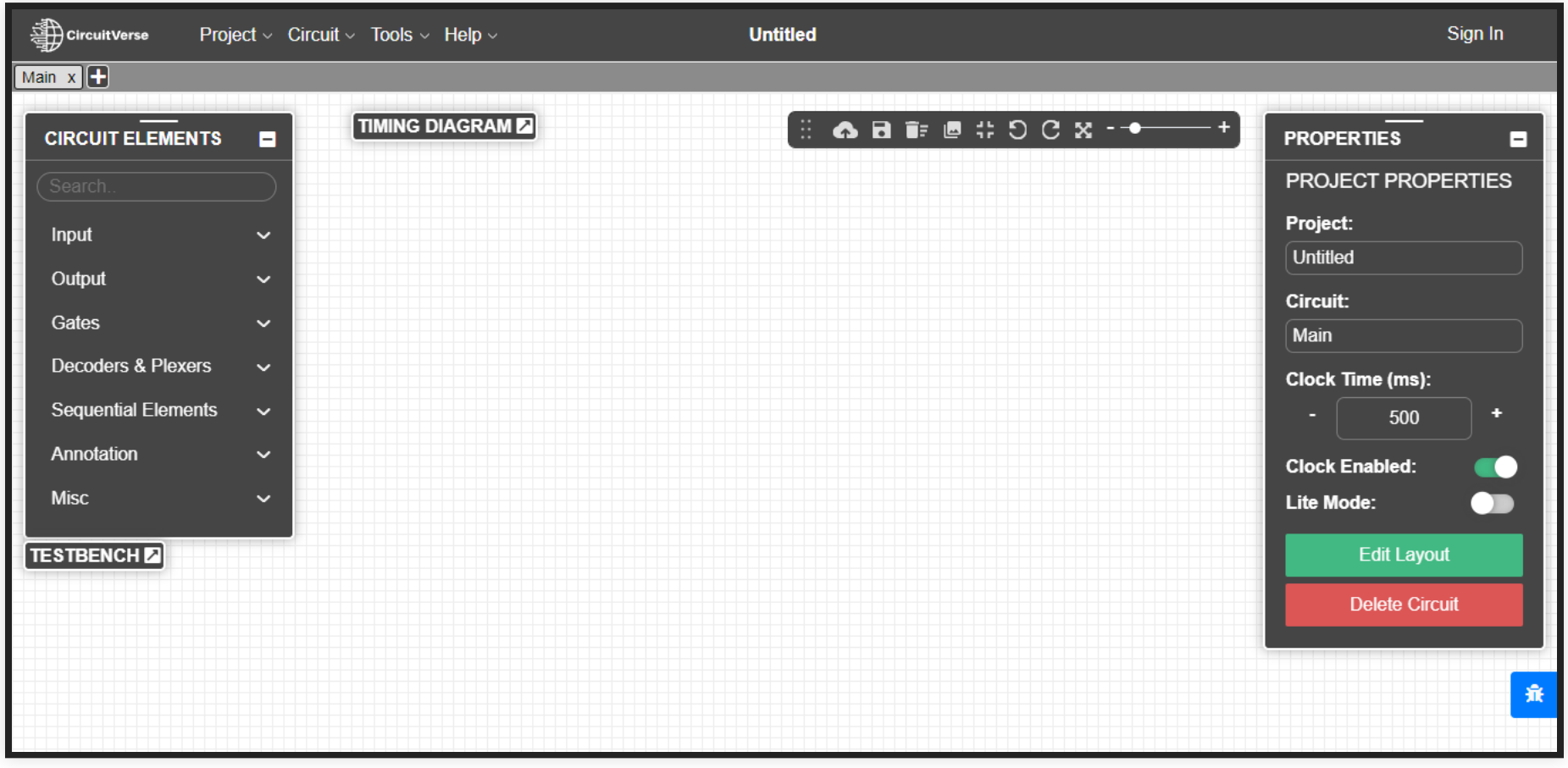
INTRODUCTION

- Finite Automata are fundamental concepts in Automata Theory and Digital Logic Design.
- These FA can be converted into a circuit.
- Given an FA transition table as input, this project aims to generate the equivalent transition logic with primitive/basic gates/components.

LITERATURE REVIEW

- CircuitVerse
- JFLAP

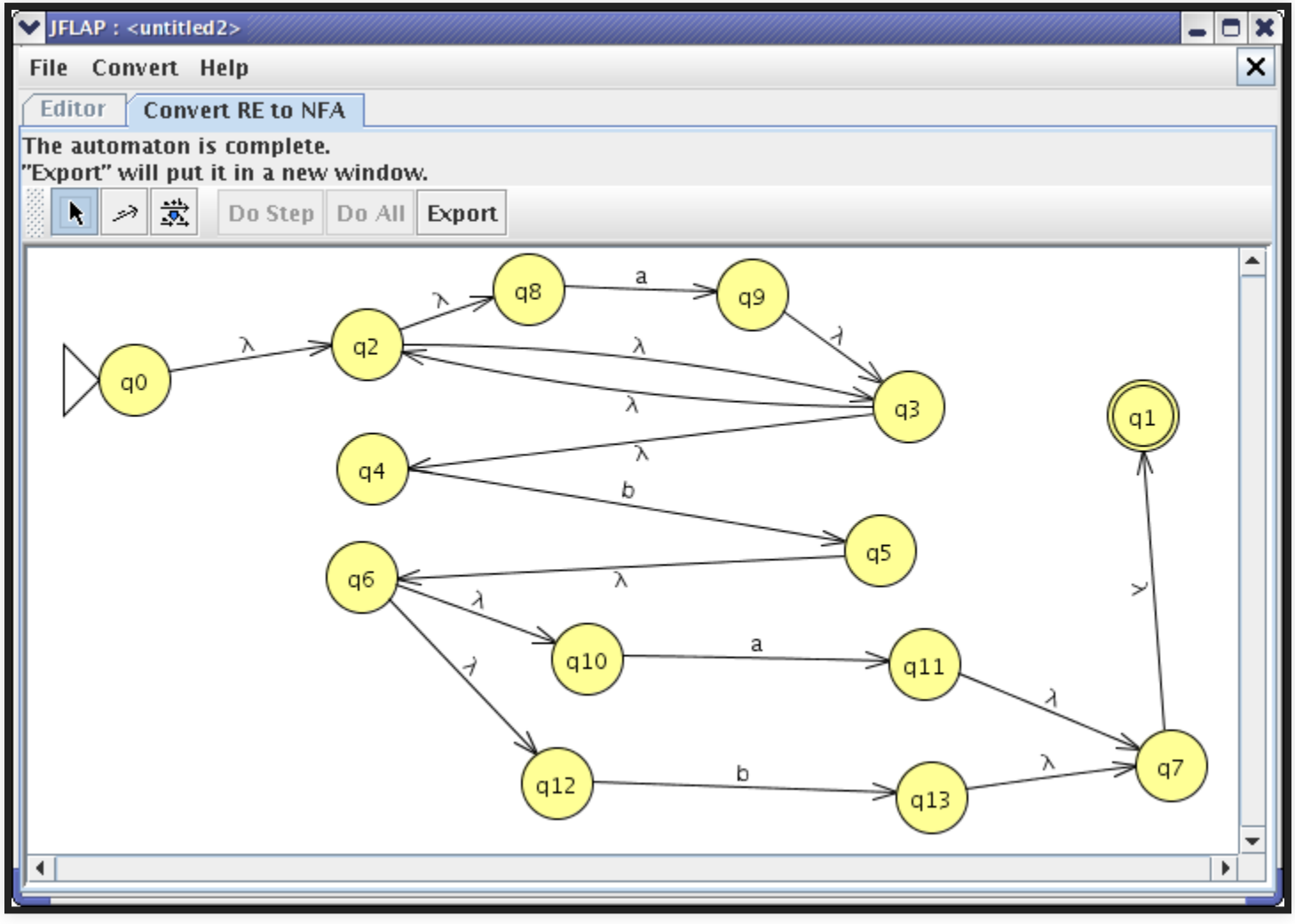
CIRCUITVERSE



LIMITATIONS

- Does not support simulation of Automata
- Supports circuit simulation but not conversion from FA

JFLAP


JFLAP : <untitled2>
File Convert Help
Editor Convert RE to NFA
The automaton is complete.
"Export" will put it in a new window.
Do Step Do All Export
q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 q12 q13
λ a b

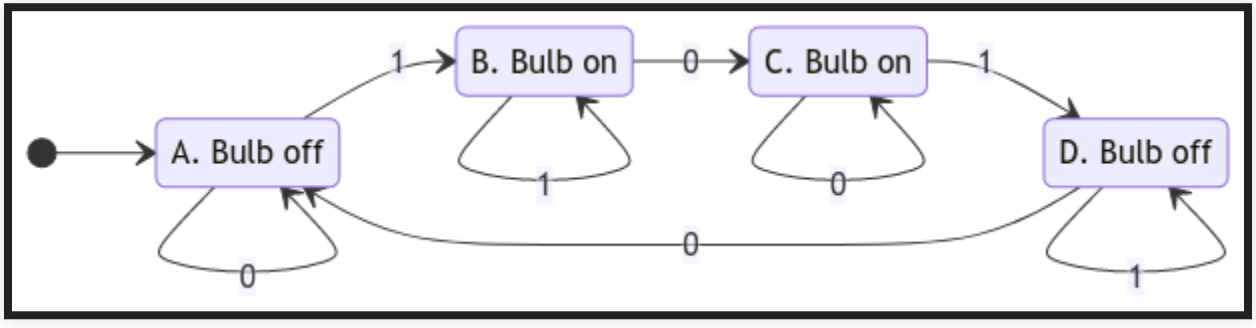
LIMITATIONS

- Has to be run locally with Java
- Users cannot share designed automata easily

METHODOLOGY

- Determine what states / transitions are needed in order to solve the problem.
- Draw the state diagram, labelling the states and the edges.

FINITE AUTOMATA



- Develop a mapping between state and representation in FFs.
- Write out the state transition table.

TRANSITION TABLE

State	button not pushed (0)	button pushed (1)
A	A	B
B	C	B
C	C	D
D	A	D

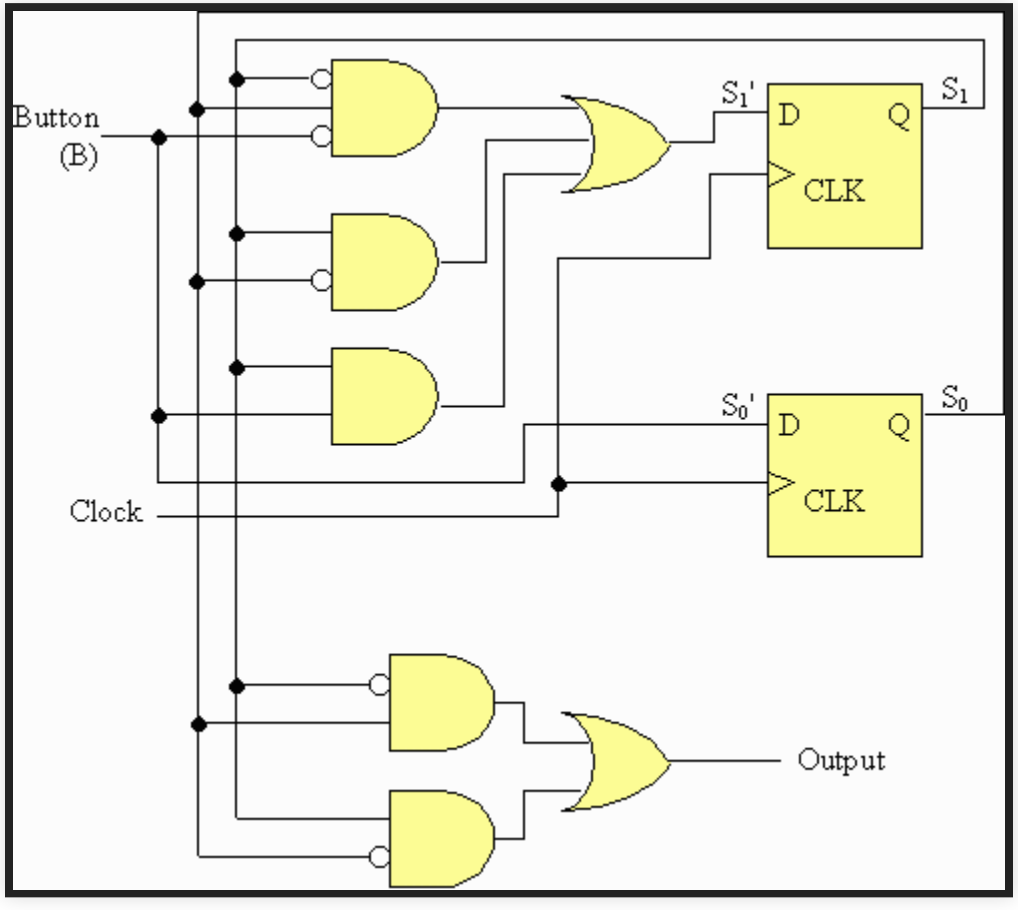
- Based on the Activation table of the flip-flops used, construct the truth table from the transition table.
- Based on the truth table, construct the equation for each output variable.

TRUTH TABLE

Present State		Button Input	Next State		Output
b0	b1		b0	b1	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

- Reduce the expressions using the suitable algorithm (K-Maps / Quine-McClusky algorithm)
- Convert the expressions to a circuit diagram using logic gates.

CIRCUIT



APPLICATION OF OUR PROJECT

Finite Automata can be used to break down a circuit's function into a collection of states and rules which determine when the system moves from one state to another state. The state diagram consists of nodes which represent the states and arrows (sometimes called edges) which give the possible transitions between states.

WORK DONE SO FAR

1) Implement KMaps for expression reduction

```
Enter the minterm expression
Separate the terms with a space( ) or plus(+)
Use ' for inverse
Eg: ab'c + ab'c'
    ab ac'

a + b + ac'

Canonical Expression:
a'bc' + a'bc + ab'c' + ab'c + abc' + abc

KMap:
+----+------+------+------+------+
|  \ | b'c' | b'c  | b c  | b c' |
+----+------+------+------+------+
| a' |      |      |    1 |    1 |
| a  |    1 |    1 |    1 |    1 |
+----+------+------+------+------+
```

```
Truth Table:
+-----+---+
| abc |   |
+-----+---+
| 000 | 0 |
| 001 | 0 |
| 010 | 1 |
| 011 | 1 |
| 100 | 1 |
| 101 | 1 |
| 110 | 1 |
| 111 | 1 |
+-----+---+

Minimized expression:
a + b
```

2) Verification Of Finite Automata

```
npx ts-node test.js
>> DFA error: state A defines more transition for input 1: B,

>> DFA error: state B does not define a transition for input 1

>> null (No error in FA)
```

3) Generation of State Transition Table

```
npx ts-node test.js
>>       ---------------------------------------
         |  State/Input  |    0    |    1    |
         ---------------------------------------
         |      A/0      |    A    |    B    |
         ---------------------------------------
         |      B/1      |    C    |    B    |
         ---------------------------------------
         |      C/1      |    C    |    D    |
         ---------------------------------------
         |      D/0      |    A    |    D    |
         ---------------------------------------
```

4) Generation of Truth Table

```
npx ts-node test.js
>>      ---------------------------------------
        |b1      b0      i       b1      b0      O|
        ---------------------------------------
        |0       0       0       0       0       0|
        ---------------------------------------
        |0       0       1       0       1       0|
        ---------------------------------------
        |0       1       0       1       0       1|
        ---------------------------------------
        |0       1       1       0       1       1|
        ---------------------------------------
        |1       0       0       1       0       1|
        ---------------------------------------
        |1       0       1       1       1       1|
        ---------------------------------------
        |1       1       0       0       0       0|
        ---------------------------------------
        |1       1       1       1       1       0|
        ---------------------------------------
```

5) Generation of minterm expressions

```
npx ts-node test.js
>>      b1 = b1'.b0.i' + b1.b0'.i' + b1.b0'.i + b1.b0.i
>>      b0 = b1'.b0'.i + b1'.b0.i + b1.b0'.i + b1.b0.i
>>      i = b1'.b0.i' + b1'.b0.i + b1.b0'.i' + b1.b0'.i
```

We first used KMaps to reduce our expression, which uses a k-map table to create the minimized expression, it’s a tabular method to get the expression and hard to implement it in code.

THE KMAP ALGORITHM HAS SOME SEVERE DISADVANTAGES:

- KMap is a NP Hard problem i.e. the time complexity to get the min expression increases exponentially with increase in number of inputs.
- There is no guarantee that the expression obtained after the process will be the minimum one.

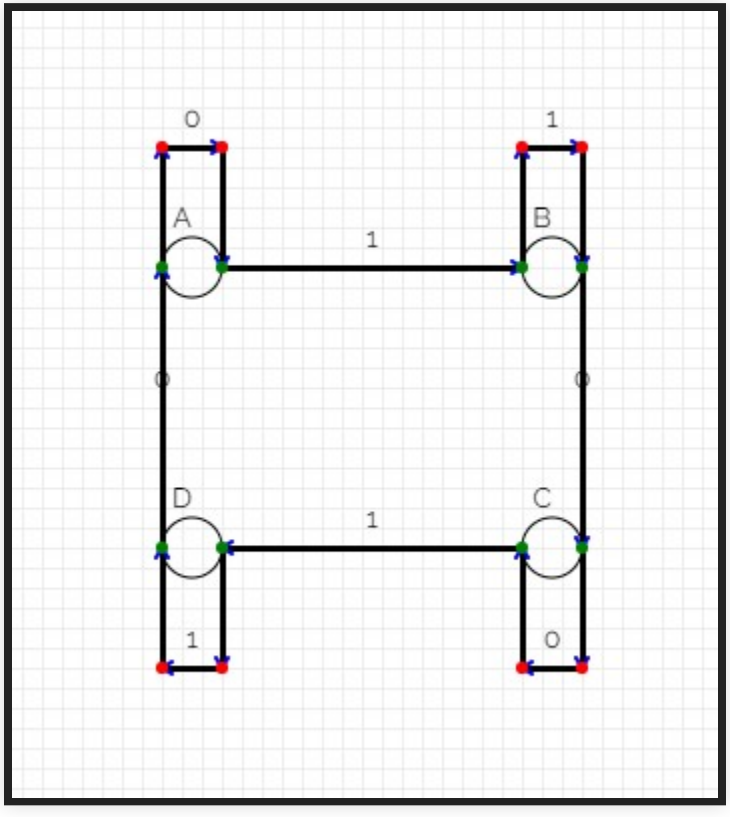
SO WE SHIFTED TO THE QUINE-MCCLUSKEY ALGORITHM.

6) Reduction of minterm expressions using Quine-McCluskey reduction technique

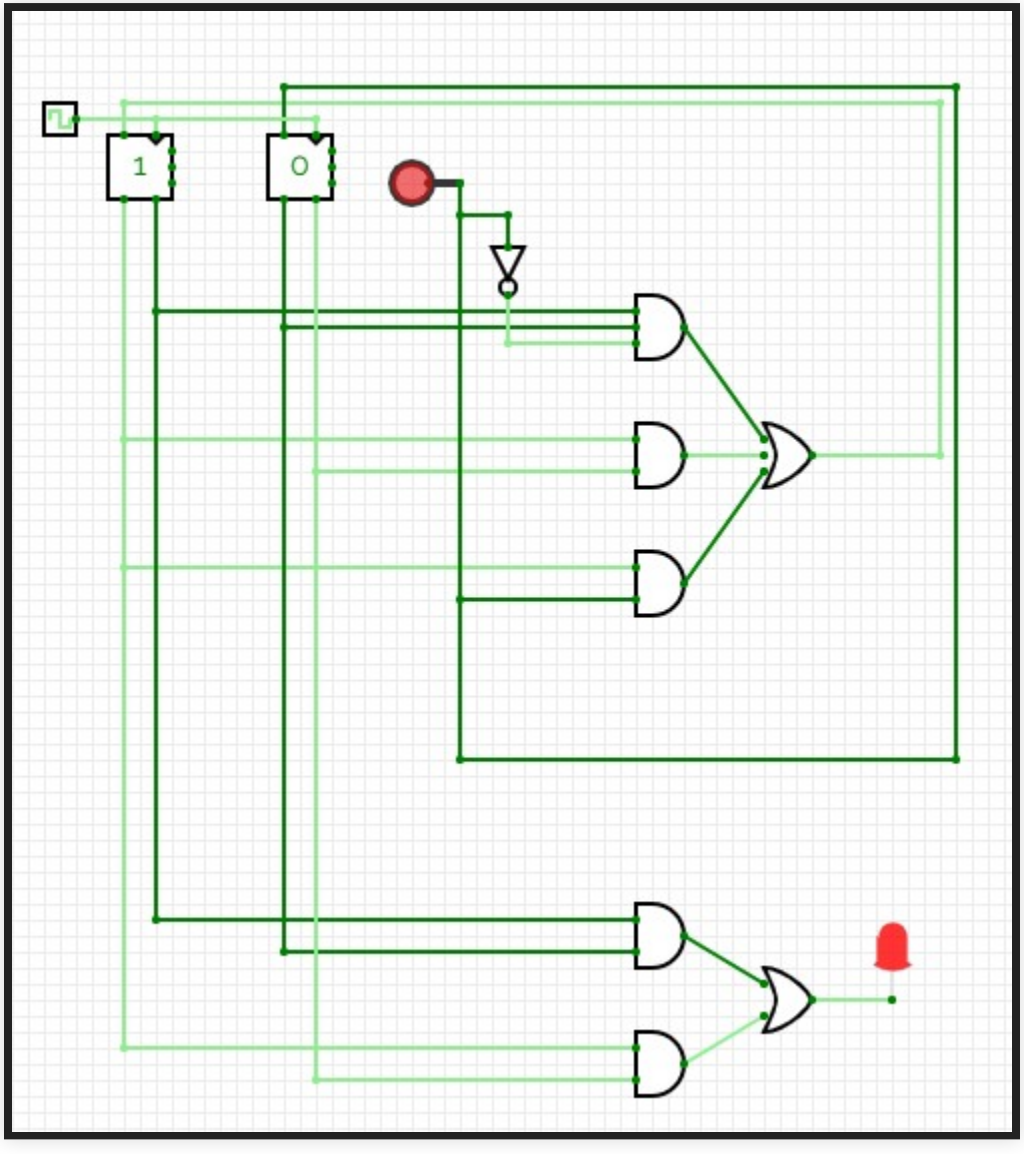
```
npx ts-node quineMcClusky.js
>>      b1 = b1'.b0.i' + b1.b0' + b1.i
>>      b0 = o
>>      i = b1'.b0 + b1.b0'
```

RESULTS

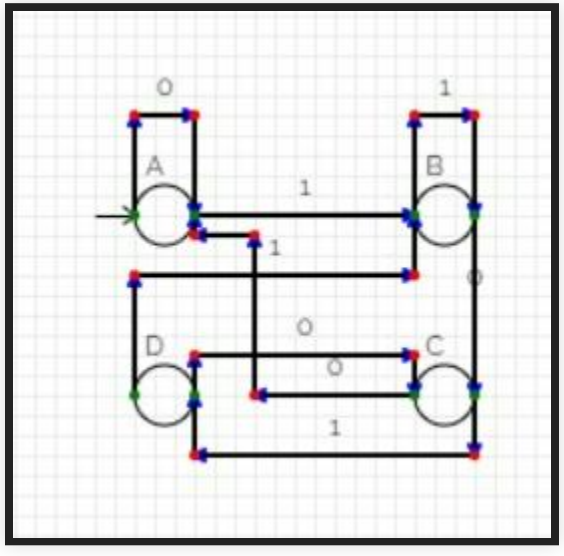
THE FINITE AUTOMATA



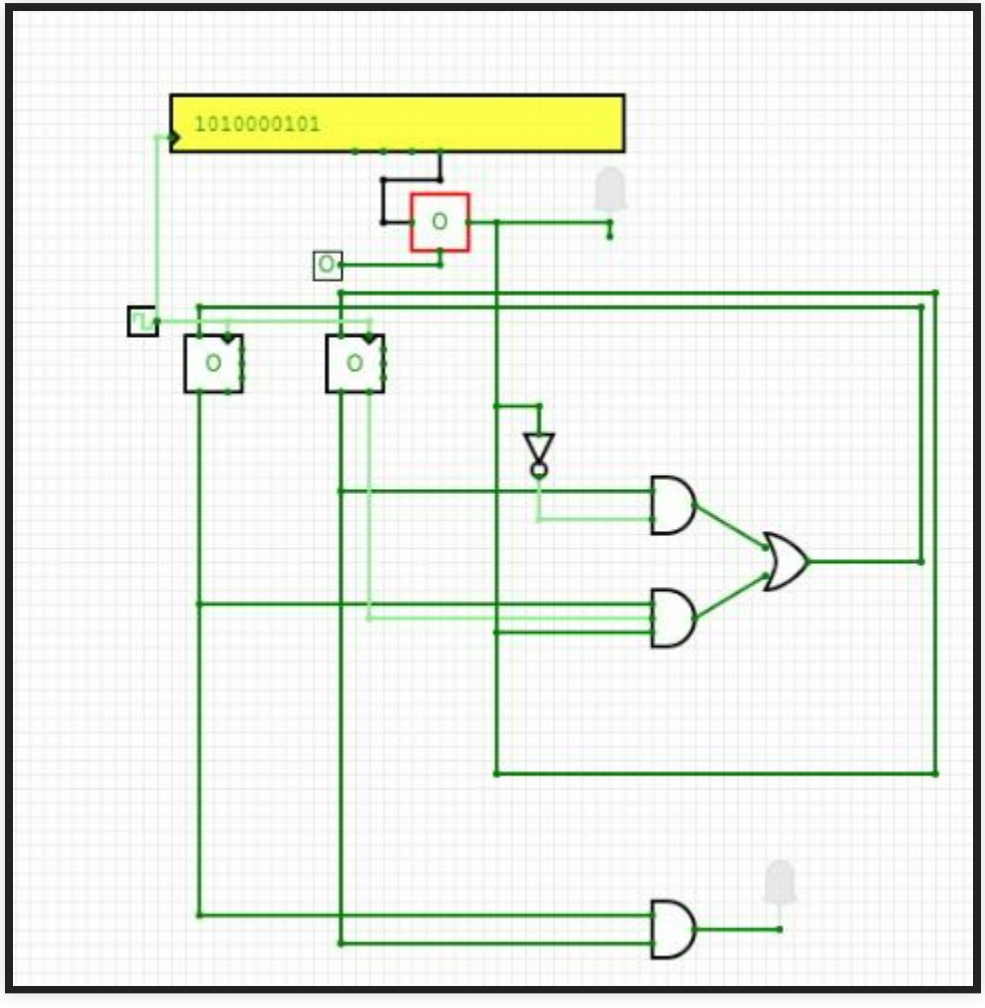
THE CIRCUIT



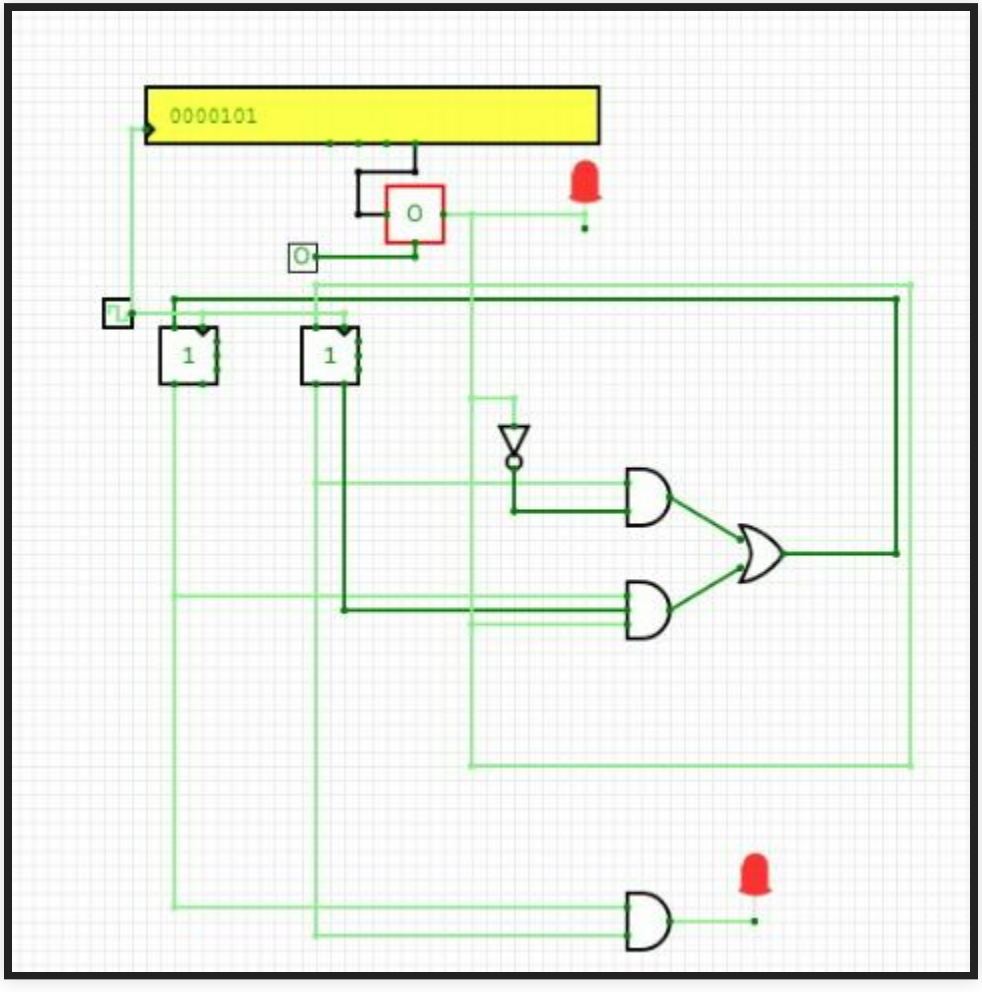
THE FINITE AUTOMATA



THE CIRCUIT



THE CIRCUIT



STEPS COMPLETED

- ~~Determine what states / transitions are needed in order to solve the problem.~~
- ~~Draw the state diagram, labelling the states and the edges.~~
- ~~Develop a mapping between state and representation in FFs.~~
- ~~Write out the state transition table.~~

- ~~Based on the Activation table of the flip-flops used, construct the truth table from the transition table.~~
- ~~Based on the truth table, construct the equation for each output variable.~~
- ~~Reduce the expressions using the suitable algorithm (K-Maps / Quine-McClusky algorithm)~~
- ~~Convert the expressions to a circuit diagram using logic gates.~~

FUTURE SCOPE

- Add support for NFA
- Add support for other types of Automata like Pushdown Automata and Turing Machine.
- Extend the algorithm to choose from the variety of flip flops

TECHNOLOGIES

- 

JavaScript
- 

TypeScript

REFERENCES

1. JFlap www.jflap.org
2. CircuitVerse circuitverse.org
3. McCluskey(1956). Minimization of Boolean Functions. 1956
4. Kodwani, Rajurkar, Mundada(2017). Realization of Sequential Circuit using Finite state Machine