Flip-Flop Conversion using Clustering

Yamir Rashid
Student Id: 011 142 147

Prince Mohammad
Student Id: 011 141 124

A project/thesis in the Department of Computer Science and Engineering presented in partial fulfillment of the requirements for the Degree of Bachelor/Master of Science in Computer Science and Engineering

United International University

Dhaka, Bangladesh

December 2018

Declaration

We, Yamir Rashid and Prince Mohammad, declare that this thesis titled, Flip-Flop Conversion using Clustering" and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a BSc degree at United International University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

Yamir Rashid, 011142147, CSE

Prince Mohammad, 011142147, CSE

Certificate

I do hereby declare that the research works embodied in this thesis entitled "**Flip-Flop Conversion using Clustering**" is the outcome of an original work carried out by Yamir Rashid and Prince Mohammad under my supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of BSc in Computer Science and Engineering.

____

Dr. Mohammad Nurul Huda

Professor and Director, MSCSE Department of CSE,UIU

Abstract

This thesis work describes the conversion of flip-flops (FF) using Clustering method to generate the Sum of Products (SOP) from the K-Maps. The approach to this work is divided into two major steps: generation of the conversion table and extraction of SOP from the K-Map using clustering. The conversion table was constructed using the characteristic table of the required FF and output obtained from the excitation table of the available FF. In the second part, a row was randomly selected with output 1 as a cluster and compared the minterms with the cluster to create groups of 4. If the matching pattern bit is 1, this step was repeated until all the output 1's are covered in the group. The process was repeated for groups of 2 but the pattern matching bit is 2. The quality among the groups was then checked and the SOP published by adding all the maximum quality patterns. From the benchmark analysis it was found that, the approach converts the flip-flops with the minimum time.

Acknowledgement

We would first like to thank the almighty Allah for bestowing us with the greatest gift, that is life and giving us the resilience to complete this thesis. We would also like to thank our parents for giving us their undying support and fortitude when we needed it. Last, but certainly not the least, we would like to offer our heartiest gratitude and thanks to our thesis supervisor Dr. Mohammad Nurul Huda for his inspiring guidance and evoking our creativity from within us.

Yamir Rashid, 011 142 147

Prince Mohammad, 011 141 124

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

A flip-Flop (FF) [1]is an electronic circuit that consists of two stable states that can be used to store binary data. The circuit can be made to change their state by applying signals to one or more control inputs which will then result in one or two outputs. It is part of the sequential circuit that helps store binary data. A Flip-flop works in a way where the output is not only dependent on the present inputs, but also dependent on the former inputs and outputs. Flip-flops have many benefits in the digital electronics world. Some of their applications are: Data storage, Data transfer, Counter, Frequency Division. There are four kinds of flip-flops: SR, JK, D and T. Each of them displays unique properties and hence it is crucial to be able to convert from one flip-flop to another.

This study can be broken down into two parts: (1) The generation of the conversion table and (2) the minimization of the Karnaugh Map (K-Map) using clustering to obtain the Sum of Products (SOP). In the first part, the conversion table we constructed using the characteristic table of the required FF and output obtained from the excitation table of the available FF. In the second part, we removed the rows with output 0 from the conversion table and selected a row with output 1 as a cluster randomly. We then compared the minterms and the cluster to create a group of 4. If the matching pattern is 1, we repeated this step until all the output 1's are covered in the group. If more 1's are remaining, then compare the minterms and cluster to create group of 2. If the matching pattern is 2, we repeated this step until all the output 1's are covered in the group. We then checked the quality among the groups and published the SOP by adding all the maximum quality patterns. After performing the benchmark analysis for the proposed algorithm we found out that this approach converts the flip flops in minimum time.

Chapter 2

Background and Literature Review

2.1 Sequential circuits

Digital electronics [2] can be essentially classified into combinational logic and sequential logic. Combinational [3] logic output is only dependent on the present inputs, whereas sequential logic output not only depends on the present input but also depends on the former inputs and outputs. Figure 2.1 shows the block diagrams of combinational and sequential circuits.

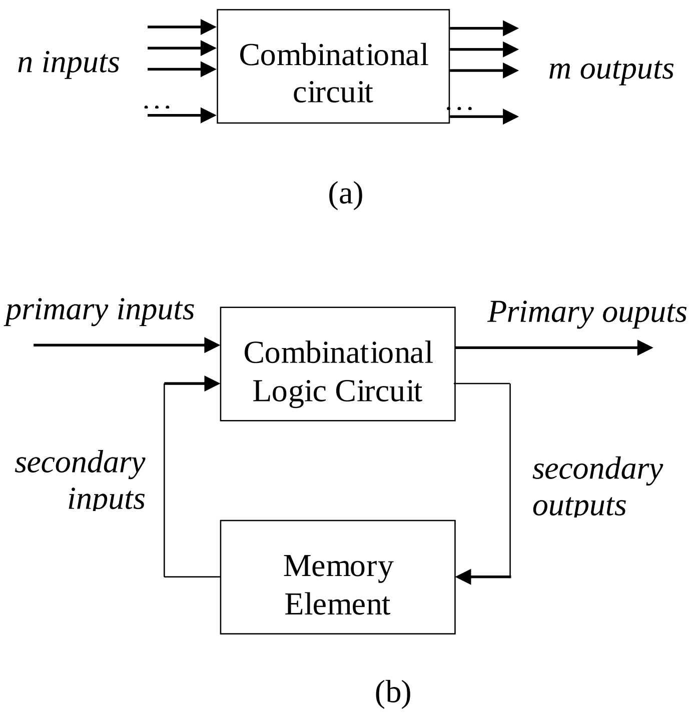


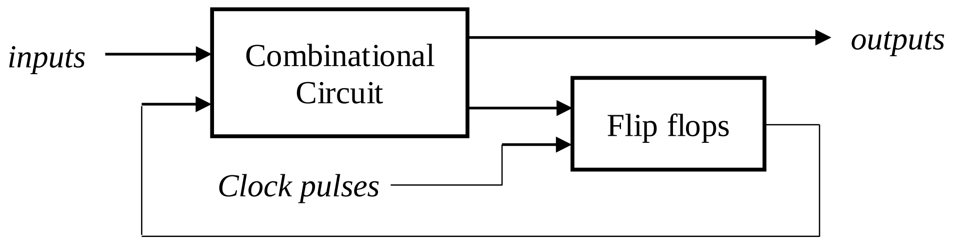
Figure 2.1: The block diagrams of combinational and sequential circuits.

At any given time, the binary that is stored in memory defines the state of the sequential circuit. The output is not only dependent on both the input and present state of the memory, but also dependent on the former outputs. There are two types of sequential circuit that can be differentiated by their relation with the timing of their signal. In the asynchronous sequential circuit, the outputs depend upon the order in which its input variables change and can be affected at any time. The synchronous sequential circuits, however, use storage devices called flip flops that are employed to change their binary value only at discrete instants of time.

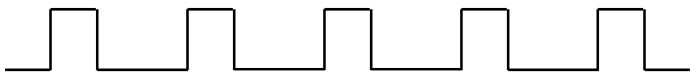
Sequential circuits use a clock signal as one of their inputs where all the state transitions in these circuits occur only when the clock value is either 0 or 1. A clock pulse generator, which is a timing device, is used to achieve synchronization as shown in Figure 2.2. Flop flops are affected only when the clock pulse from the clock pulse generator is applied.

2.2 Flip-flop basics

A flip flop [1] is a digital electric circuit that can be used to store binary data. A circuit capable of storing data is often called a memory or latch.



(a) Block diagram



(b) Timing diagram of clock pulses

Figure 2.2: Synchronous clocked sequential circuits.

Flip flops are functions of logic gates. If the logic gates are constructed properly and certain input values are given to them, they will be saved and executed. Flip flops are basically used to design better circuits. Individual flip flops can be combined to give us many applications, such as memory registers, counter, and shift registers.

Flip-flops [4] are the basic building blocks of digital electronics systems that are used in many fields such as computers, communications, and many other types of systems. There are two stable states is a flip-flop. Flip-flops can be used to store binary data which can be changed by applying varying inputs. A flip flop has the ability to store one bit of information. The main difference between latches and flip-flops is that for latches, their outputs are constantly affected by their inputs as long as the enable signal is applied. In other words, when they are enabled, their content changes with the inputs change. Flip-flops, on the other hand, have their content change only when there is a change in clock signal. If there is no change in clock, the flip-flop content remains constant even if there is a change in input.

There are basically four types of flip-flops:

1. S-R flip-flop
2. D flip flop
3. J-K flip flop, and
4. T flip flop

The major differences in these flip-flop types are the number of inputs they have and how they change state. For each type, there are also different variations and characteristic that enhance their operations.

2.3 Flip-flop conversion [5]

2.3.1 SR flip-flop to JK flip-flop conversion

SR and JK flip flops are devices that have the ability to take two inputs. In JK-FF, j and k are used as external inputs to S and R in SR-FF. Here, both S and R are the final outputs of the combinational circuit. The current state is expressed with Qp and Qp+1 is the next state to be found when the J and K inputs are received. When the CLK pulse is HIGH, information from the S and R inputs passes through the FF. When the CLK pulse is removed, the state of the FF is unstated. Therefore, S=1 R=1 is an invalid state. JK flip flops do not have any such invalid input state, which is why it might be useful to the user.

Table 2.1: SR flip-flop to JK flip-flop conversion table

J-K Inputs		**Outputs**		**S-R Inputs**	
J	**K**	**Q_p**	**Q_p+1**	**S**	**R**
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

2.3.2 JK flip-flop to SR flip-flop conversion

This conversion of SR to JK is the opposite of JK to SR. S and R will be taken as external inputs in this case. A conversion table is to be constructed using S, R, Qp, Qp+1, J and K. Eight combinations are possible for two inputs. Qp+1 outputs are generated for each corresponding combinations where the outputs for the combinations of S=1 and R=1 are not applicable for an SR flip flop. Thus the outputs are considered invalid and the values of J and K are taken as “don’t cares”. SR flip flops are much simpler than JK flip flops which suggests their usage.

Table 2.2: JK flip-flop to SR flip-flop conversion table

S-R Inputs		**Outputs**		**J-K Inputs**	
S	**R**	**QP**	**QP+1**	**J**	**K**
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Don’t Care	
1	1	Invalid		Don’t Care	

2.3.3 SR flip-flop to D flip-flop conversion

D flip flop can be derived from SR flip flop by providing input to S and inverted input to R. D flip flops are favored over SR as it is simpler and has less input gates. As shown in the figure, S and R are taken as inputs of the flip flop and D is the external input of the flip flop. The D FF is responsible for a delay which forces the inputs to slow down so that all inputs arrive at the same time.

Table 2.3: SR flip-flop to D flip-flop conversion table

D Inputs	**Outputs**		**S-R Inputs**	
	QP	**QP+1**	**S**	**R**
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

2.3.4 D flip-flop to SR flip-flop conversion

D is the input and S and R are given as external inputs. Since there are two inputs (S-R), eight combinations are produced. However, since S=1 and R=1 is an invalid state, the values of S, R ad Q are taken to be as "Don't Cares". The conversion table is shown below.

Table 2.4: D flip-flop to SR flip-flop conversion table

S-R Inputs		**Outputs**		**D Input**
S	**R**	**Q_P**	**Q_P+1**	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Don't care
1	1	Invalid		Don't care

2.3.5 JK flip-flop to T flip-flop conversion

J and K are the input and T is given as the external input. T and Qp gives four combinations. The conversion table is shown in Table 2.5.

Table 2.5: JK flip-flop to T flip-flop conversion table

T Input	**Outputs**		**J-K Inputs**	
	Q_P	**Q_P+1**	**J**	**K**
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

2.3.6 JK flip-flop to D flip-flop conversion

J and K are the input and D is given as external input. Four combinations are achieved from D and Qp. The conversion table is shown in Table 2.6.

Table 2.6: JK flip-flop to D flip-flop conversion table

D Input	**Outputs**		**J-K Inputs**	
	Q_P	**Q_P+1**	**J**	**K**
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

Chapter 3

Methodology

3.1 Introduction

In our method, we first generated rules from the truth table of the required and available flip-flops for constructing the characteristic tables. Using the characteristic table of the available flip-flop we constructed the excitation table of the available flip-flop. After that, for the present and next state of the characteristic table we generated output using the excitation table of the available flip-flop. By joining the generated output with the characteristic table of the required flip-flop we constructed the conversion table. To generate SOP (sum of products) from the conversion table we removed the rows that contain 0 in output. Now, a clustering technique has been used to solve KMAP. We selected a row that contains 1 in output as cluster and checked whether a feature input value is matched among 4 rows and grouped them. If the table still has more Output 1 to cover we selected new clusters in the same way and grouped them. If the table still has more output 1 to cover we selected the output row as cluster and checked whether two feature input values are matched among 2 rows and grouped them until all the Output 1 covers. If still output 1 remained in the output to cover we added them in the output. Then, we checked the quality of the 4 items and 2 items groups and selected the groups with maximum quality and also contains different indexed output 1 and stored them in output. Finally, we generated SOP by adding the selected outputs matching feature inputs as symbol.

3.2 The Algorithm

3.2.1 Input

Truth table of available and required flip-flop.

3.2.2 Output

SOP, minimized Boolean function.

3.2.3 Method

Step-1:

```
Generate rules using the truth table of required flip-flop.
Here, an Arraylist is used, <values>
        If (clock = 1) then
                Save the columns of inputs and next state.
Arraylist <values> rules = new Arraylist <values> ( ).
                For inputs, i = 1, 2
                Save them in rules input and rules. Next state.
        End if.
```

Step-2:

For inputs i = 2, 3 generate 2^n binary inputs and store them in a new Arraylist <values> {characteristic. Present(i) , characteristic. Inputs(i) }

Step-3:

Now compare and generate next state and store them in characteristic. Nextstate(i).

Step-4:

Repeat step-1 to step-3 for the available FF.

Step-5:

Generate 2^2 binary inputs and store them in a new arrraylist(). Excitation. Excitation.present(i), Excitation.next(i) and set Excitation.input(i) = -2.

<u>Step-6:</u>

```
For j = 1 to Excitation.size( )
        For k = 1 to characteristic.size ( )
                If (characteristic.size ( ) = 4) then
If (excitation.present = characteristic.present And Excitation.next =  characteristic.next)
then excitation.input(j) = characteristic.input(k)
                        Break.
                        End if.
                End if.
else if (excitation.present = characteristic.prestent And excitation.next =
characteristic.next) then
                        If (excitation.input1(i) = -2) then --------(1)
Excitation.input1(j) = characteristic.input1(k)
else if (excitation.input1(j) = characteristic.input1(k))
                                Do nothing
                        Else
                                Excitation.input1(j) = -1.
                        End if.
                End if.
End i. --------(2)
Use same conditions (1)---(2) for excitation. input2(j).
```

<u>Step-7:</u>

Here, a new arraylist() is used, <type = values>
Copy characteristic arraylist of required flip-flop in a new arraylist<values> conv.addAll (characteristic)

<u>Step-8:</u>

Generate output using the excitation arraylist.

```
If(conv.present = excitation.presnt And conv.next = excitation.next) then
        For inputs, i = 1/2
        Conv.output(i) = excitation.input(i)
```

<u>Step-9:</u>

```
Clustering (number of inputs, table, output)
    If (noi  = 2) groupnumber = 1.
    End if.
    Else if (noi = 3) groupnumber = 3.
    End if.
ArrayList <values> visited.
Remove rows that contain output 0.
    If (noi = 2 and table. Size ( ) = 4)
        Output SOP 1. Return.
    End if.
    If (noi = 3 and table. Size ( ) = 8)
        Output SOP 1. Return.
    End if.
Select a row that contains output 1 randomly that is not in the visited ArrayList.
    While (!visited.contains (all the output 1)
    Select a random row that contains output 1.
        If (visited.contains (Select). Do Nothing
        End if
    Else
        Count = 0.
        Cluster.add (Select)
            Compare the input values of the cluster with all the other rows.
    If (Cluster.input[i].value = Row.input[j].value and count <groupnumber)
        Count++
        Save the column number and row numbers.
        If( count = groupnumber)
```

```
            Save the saved row numbers in selected groups and matching input values and symbol.
        End if
    If (noi = 2)
        Select the unselected output 1 as a group individually.
    End if
        Visited.add (Select)
    Else if. (noi =3 )
        Compare the input values of the cluster with all the other rows.
    If (2#Cluster.input[i].value = 2#Row.input[j].value and count < 1)
        Count++
        Save the column number and row numbers.
        If( count = 1)
Save the saved row numbers in selected groups and matching input values and symbol.
        End if
Select the unselected output 1 as group individually.
Visited.add (Select).
```

Step-10:

```
SOP(Selected Groups)
    Final Output = the maximum covering output 1 group
    While(!FinalOutput.containAll(output1)
        Final Output = maximum covering output 1 group that contains different output1 rows
    End while
```

Step-11:

```
Obtain SOP by adding all the Final Outptuts symbol respect to the saved values.
IF(SavedValue=0)
    SOP =( Symbol)¯
End IF
Else
```

```
        SOP = Symbol
End IF
```

3.3 An example: JK to SR conversion

Input:

Table 3.1: Truth table of available flip flop

C	J	K	N
0	X	X	P
1	0	0	P
1	0	1	0
1	1	0	1
1	1	1	$\bar{P}$

Table 3.2: Truth table of required flip flop

C	S	R	N
0	X	X	P
1	0	0	P
1	0	1	0
1	1	0	1
1	1	1	X

Step-1: Generate rules using truth table of required flip flop.

When clock = 1, store the rules in arraylist <values>. rules.

Generated Rules:

When S = 0 R = 0 N.state = P

When S = 0 R = 1 N.state = 0

When S = 1 R = 0 N.state = 1

When S = 1 R = 1 N.state = X

Step-2: Generate binary inputs and store them in characteristic array list.

P	S	*R*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Step-3: Using the rules array list generate characteristic table with next state.

P	**S**	**R**	**N**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Step-4: Repeating Step 1 to step 3 for available flip flop.

When J = 0 K = 0, N = P

When J = 0 K = 1, N = 0

When J = 1 K = 0, N = 1

When J = 1 K = 1, N = P^{-}

P	**J**	**K**	**N**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Step-5:

P	**Qn**	**J**	**K**
0	0	-2	-2
0	1	-2	-2
1	0	-2	-2
1	1	-2	-2

Step-6:

P	**Qn**	**J**	**K**
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

For P = 0, Qn = 0,

We get 2 values for each J and K

J = 0,0

K = 0,1

So, J = 0, K = X.

Step-7: Copy characteristic flip flop in conversion table array list.

Step-8: Conversion table generated using excitation table.

P	**S**	**R**	**N**	**J**	**K**
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	X
0	1	1	X	X	X
1	0	0	1	X	0
1	0	1	0	X	1
1	1	0	1	X	0
1	1	1	X	X	X

Step-9:

Select = J.

Remove 0 valued rows.

Index	**P**	**S**	**R**	**N**	**J**
1	0	1	0	1	1
2	0	1	1	X	X
3	1	0	0	1	X
4	1	0	1	0	X
5	1	1	0	1	X
6	1	1	1	X	X

Select index 1 as cluster.

Check for 4 number of rows for grouping.

0 1 0

0 1 1

1 1 0

1 1 1

It covers all the rows with output 1.

So, break.

Select = K

Remove 0 valued rows.

Index	**P**	**S**	**R**	**N**	**K**
1	0	0	0	0	X
2	0	0	1	0	X
3	0	1	0	1	X
4	0	1	1	X	X
5	1	0	1	0	1
6	1	1	1	X	X

Select index 5 as cluster.

Check for 4 number of rows to group.

1 0 1

0 0 1

0 1 1

1 1 1

As it covers all the rows with output 1. Breaks.

Step-10:

For output 1,

Pattern = 1

Pattern symbol = S

For output 2,

Pattern = 1

Pattern symbol = K

Step-11: Final SOP

Output 1 = J = S

Output 2 = K = R

3.3 Pseudocode

Input: Truth table of available and required F-F.

Output: Sum of Products (SOP)

Steps:

Conversion Table Generation:

1. Read truth table of the required F-F.
2. Determine the number of inputs n and input symbol and store it.
3. When Clock = 1 store the rules of next state.
4. Generate 2^(n) binary numbers and store it as characteristic table's input.
5. Using the rules of the next state complete the characteristic table.
6. Repeat step 1 to 5 for the available F-F.
7. Generate 2^2 binary numbers for the present and next state of excitation table for available F-F.
8. Using the generated characteristic table of the available F-F and comparing the present and next state with the excitation's present and next state complete the excitation table's input.
9. Compare the present and next state of both characteristic table of the required F-F and excitation table of the available F-F complete the conversion table.

<u>SOP Generation (Conversion Table, input number):</u>

10. Remove the rows that contain 0 in output.
11. If no rows has been removed then SOP is 1 and return.
12. Set group size as 4 if input number is 3 or 2 if input number is 2.
13. Select a row that contains 1 in output as cluster and mark it as visited.
14. If(group size = 4) then
    A. For each input bit of the selected cluster check if 4 rows including the cluster contains the same input value and group them and save the symbol and matched bit value.
    B. If step A is not possible for each pair of input bit of the selected cluster check if 2 rows including the cluster contains the same input value and group them and save the symbols and matched bit value's.
15. If(group size = 2) then

For each input bit of the selected cluster check if 2 rows including the cluster contains the same input value and group them and save the symbol and matched bit value.

16. Repeat step 13 to 15 until all the output 1 of the conversion table is visited.
17. Check the groups if they contains all the output 1 of the conversion table and if there is still output 1 remaining select them as an individual group and save the symbols and matched bit value.
18. Repeat step 10 to 17 for each output.
19. Select the maximum sized group that contains maximum output 1 and remove it.
20. Repeat Step 10 until all the groups that contains different output 1 of the conversion table is selected.
21. Generate the symbols of the selected groups using the matched bit.
22. Publish SOP by adding all the symbols of the selected groups.

Chapter 4

Findings

The Benchmark analysis [6] between the proposed algorithm and Brute force method is given in the following table. Table 4.1 is constructed by implementing our proposed algorithm and brute force method in java programming language. We have conducted tests on both implementations with random run and continuous 100 runs and observed that the time taken to convert the flip flops and find the Sum of Products is faster in our proposed algorithm.

In Figure 4.1 A time analysis graph between our proposed algorithm and the brute force method.is carried out where the x-axis denotes different flip-flop conversions while the y-axis denotes the execution times in seconds. From the table we can determine that our algorithm converts at a linear time complexity whereas brute force method increases exponentially.

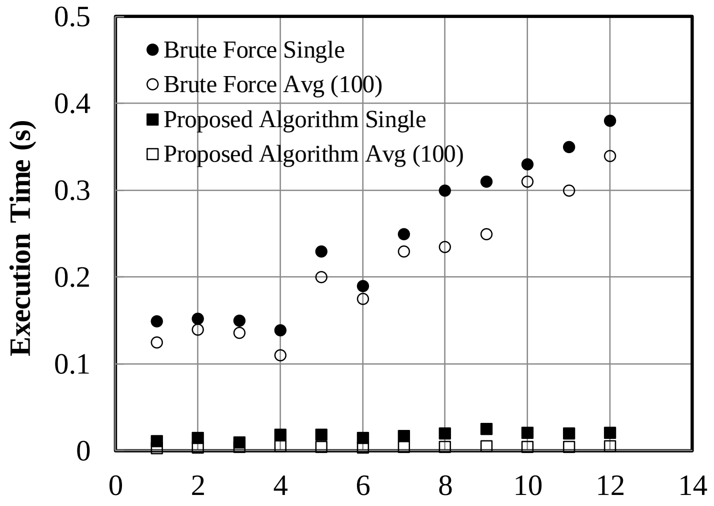


Figure 4.1: Execution time analysis graph between proposed algorithm and Brute Force algorithm.

Table 4.1: Benchmark analysis between proposed algorithm and Brute Force algorithm.

No.	**Available**	**Target**	**Output**	**Execution Time (seconds)**			
				Brute Force		**Proposed Algorithm**	
				Single	**Avg (100)**	**Single**	**Avg (100)**
1	SR	D	S = D and R = $\overline{D}$	0.1490	0.1250	0.01100	0.00306
2	JK	SR	J = S and K = R	0.1520	0.1400	0.01500	0.00392
3	SR	T	S = $\overline{Qp}$.T and R = Qp.T	0.1500	0.1359	0.01000	0.00463
4	JK	T	J = T and K = T	0.1390	0.1100	0.01900	0.00548
5	JK	D	J = D and K = $\overline{D}$	0.2300	0.2000	0.01900	0.00498
6	T	D	T = $\overline{Qp}$.D + Qp. $\overline{D}$	0.1900	0.1750	0.01500	0.00423
7	D	T	D = $\overline{Qp}$.T + Qp. $\overline{T}$	0.2500	0.2300	0.01700	0.00449
8	D	JK	D = J. $\overline{Qp}$ + $\overline{K}.Qp$	0.3000	0.2350	0.02000	0.00478
9	D	SR	D = S + $\overline{R}$. Qn	0.3100	0.2500	0.02500	0.00511
10	T	JK	T = J. $\overline{Qp}$ + K.Qn	0.3300	0.3100	0.02100	0.00493
11	T	SR	T = S. $\overline{Qp}$ + R.Qp	0.3500	0.3000	0.02000	0.00488
12	SR	JK	S = $\overline{J}$.Qp and R = K.Qp	0.3800	0.3400	0.02100	0.00568

Chapter 5

Conclusions

This thesis work has provided an approach to convert flip-flops. The findings in this work show that our algorithm is the most efficient way to accomplish the goal compared to the brute force method. It concludes with the following decisions:

The principal advantages are:

1) Linear time complexity
2) Converts all available flip flops
3) Converts custom made flip flops with input 2 or less.

Further improvements of the present work can be made by optimizing the proposed algorithm by introducing other machine learning techniques to compared with.

References

[1] V. A. Pedroni, Digital Electronics and Design with VHDL, Massachusetts: Morgan Kaufmann, 2008.

[2] M. Rafiquzzaman, Fundamentals of Digital Logic and microcontrollers, 6th Edition, John Wiley & Sons, 2014.

[3] V. K. Puri, Digital Electronics and Circuits and Systems, Faridabad: Tata McGraw-Hill, 1997.

[4] "electronicsforu.com," EFY Group, [Online]. Available: https://electronicsforu.com/resources/learn-electronics/flip-flop-rs-jk-t-d. [Accessed 2/23/2018 December 2018].

[5] John, "circuitstoday," 11 August 2018. [Online]. Available: http://www.circuitstoday.com/flip-flop-conversion. [Accessed 2/11/23 December 2018].

[6] Md Abu Raihan Srabon, Hossain Md. Tareq, Omar Faruq Osama, MD. Tanvir Hoassain, "Automatic Minimization of Karnaugh Map for SOP," Dhaka, 2018.