

Switching Theory

Assignment 1

Star and Sharp Operations for Minimization

Professor Stephen Brown

Jing Nan Chen
Student #: 1002821404

Qi Jian Huang
Student #: 1002675866

Contents

Introduction:.....	3
How to run our program:.....	3
Important features of our implementation:	4
Test Results of the Given Nodes:.....	5
Limitations:	9
Conclusion:	11

Introduction:

Cover cost minimization can be determined by using various different techniques including Quine-McCluskey, and “star” and “sharp” operations. For this assignment, our end goal is to implement a program based on C/C++ programming language to calculate and find the minimal cover cost using “star” and “sharp” operations (a cubical technique for minimization).

How to run our program:

Our program can be run using either the provided executable file (assign1.exe) or visual studio IDE (2015 version is recommended).

If by executable file:

- Use command window navigate to the directory where the executable file is located
- Make sure the test nodes also located in the same location
- Execute “assign1.exe node#.blif” where “#” is the test node number
- The program result should be displayed on the command window

If by visual studio 2015:

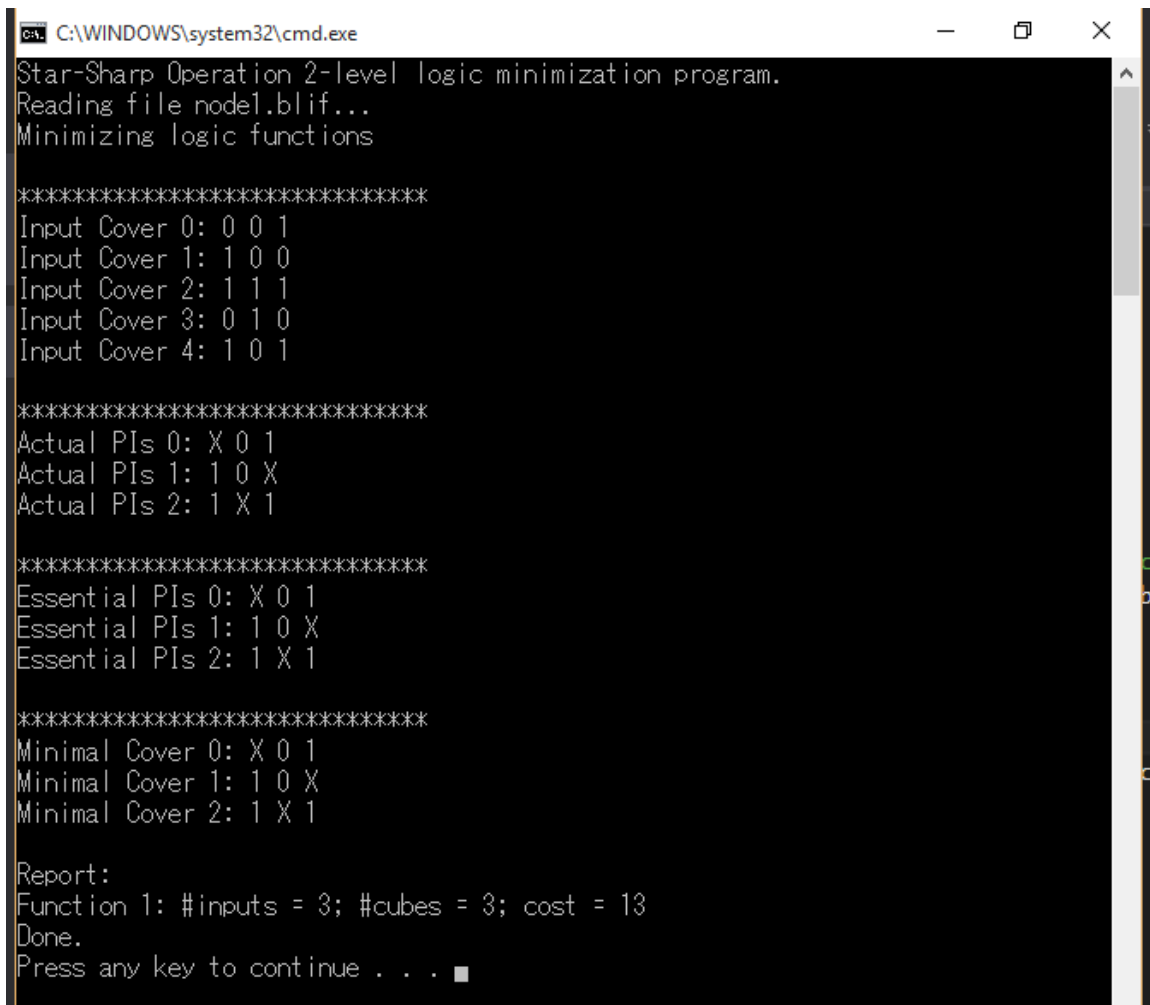
- Double click the assign1.sln, and open our project in Visual Studio 2015
- Add test node argument under “Project -> assign1 properties -> debugging -> command arguments”

- Run the program by clicking “Debug -> start without debugging”
- The program result should be displayed on the command window

Important features of our implementation:

- A star function to perform star operation between two cubes
- A sharp function to perform sharp operation between two cubes
- A combine function to combine two covers and eliminate redundant cubes
- A equality function to compare if two cover equals to each other
- An implementation to identify all prime implicants
- An implementation to identify the DC cubes from prime implicants
- A function to find all essential prime implicants from actual prime implicants
- A function to identify all uncovered minterms that essential prime implicants does not cover in the ON-set cover
- A function to identify all uncovered cubes after choosing one prime implicant in order to determine if the ON-set cover is fully covered
- An implementation to find a good minimal cover for the input function

Test Results of the Given Nodes:



```
C:\WINDOWS\system32\cmd.exe
Star-Sharp Operation 2-level logic minimization program.
Reading file node1.blif...
Minimizing logic functions

*****
Input Cover 0: 0 0 1
Input Cover 1: 1 0 0
Input Cover 2: 1 1 1
Input Cover 3: 0 1 0
Input Cover 4: 1 0 1

*****
Actual PIs 0: X 0 1
Actual PIs 1: 1 0 X
Actual PIs 2: 1 X 1

*****
Essential PIs 0: X 0 1
Essential PIs 1: 1 0 X
Essential PIs 2: 1 X 1

*****
Minimal Cover 0: X 0 1
Minimal Cover 1: 1 0 X
Minimal Cover 2: 1 X 1

Report:
Function 1: #inputs = 3; #cubes = 3; cost = 13
Done.
Press any key to continue . . . █
```

Figure 1: Node1 Test Result

```
C:\WINDOWS\system32\cmd.exe
Star-Sharp Operation 2-level logic minimization program.
Reading file node2.blif...
Minimizing logic functions

*****
Input Cover 0: 0 0 X 0
Input Cover 1: 1 0 0 X
Input Cover 2: 1 0 1 0
Input Cover 3: 1 1 1 1
Input Cover 4: 0 0 X 1
Input Cover 5: 0 1 1 X

*****
Actual PIs 0: 0 0 X X
Actual PIs 1: X 1 1 1
Actual PIs 2: X 0 0 X
Actual PIs 3: 0 X 1 X
Actual PIs 4: X 0 X 0

*****
Essential PIs 0: X 1 1 1
Essential PIs 1: X 0 0 X
Essential PIs 2: X 0 X 0

*****
Minimal Cover 0: X 1 1 1
Minimal Cover 1: X 0 0 X
Minimal Cover 2: X 0 X 0

Report:
Function 1: #inputs = 4; #cubes = 3; cost = 14
Done.
Press any key to continue . . .
```

Figure 2: Node2 Test Result

```
C:\WINDOWS\system32\cmd.exe
Input Cover 16: 1 1 1 1 0
Input Cover 17: 1 1 1 1 1
Input Cover 18: 0 1 0 0 0
Input Cover 19: 1 0 0 0 0
Input Cover 20: 1 0 1 0 1
Input Cover 21: 1 0 1 1 0

*****
Actual PIs 0: 0 1 0 0 X
Actual PIs 1: X 1 0 1 0
Actual PIs 2: 0 1 0 X 0
Actual PIs 3: 0 1 X 0 0
Actual PIs 4: X X 0 0 1
Actual PIs 5: X 0 1 X 0
Actual PIs 6: X X 1 0 0
Actual PIs 7: X 0 1 1 X
Actual PIs 8: X X 1 1 1
Actual PIs 9: 1 0 X X 1
Actual PIs 10: 1 X 0 X 1
Actual PIs 11: 1 0 X 0 X
Actual PIs 12: 1 X X 1 1
Actual PIs 13: 1 X 1 X 0
Actual PIs 14: 1 0 1 X X
Actual PIs 15: 1 X 1 1 X
Actual PIs 16: 1 1 X 1 X

*****
Essential PIs 0: X X 0 0 1
Essential PIs 1: X X 1 1 1

*****
Minimal Cover 0: X X 0 0 1
Minimal Cover 1: X X 1 1 1
Minimal Cover 2: 1 1 X 1 X
Minimal Cover 3: X 0 1 X 0
Minimal Cover 4: X X 1 0 0
Minimal Cover 5: 1 0 X X 1
Minimal Cover 6: X 1 0 1 0

Report:
Function 1: #inputs = 5; #cubes = 7; cost = 37
Done.
Press any key to continue . . . ■
```

Figure 3: Node3 Test Result

```
ca Select C:\WINDOWS\system32\cmd.exe
Star-Sharp Operation 2-level logic minimization program.
Reading file node4.blif...
Minimizing logic functions

*****
Input Cover 0: 0 0 X 0
Input Cover 1: 1 0 0 X
Input Cover 2: X 0 1 0
Input Cover 3: 1 1 1 1
Input Cover 4: 0 0 X 1
Input Cover 5: 0 1 1 X

*****
Actual PIs 0: 0 0 X X
Actual PIs 1: X 1 1 1
Actual PIs 2: X 0 0 X
Actual PIs 3: X 0 X 0
Actual PIs 4: 0 X 1 X

*****
Essential PIs 0: X 1 1 1
Essential PIs 1: X 0 0 X
Essential PIs 2: X 0 X 0

*****
Minimal Cover 0: X 1 1 1
Minimal Cover 1: X 0 0 X
Minimal Cover 2: X 0 X 0

Report:
Function 1: #inputs = 4; #cubes = 3; cost = 14
Done.
Press any key to continue . . .
```

Figure 4: Node4 Test Result


```
C:\WINDOWS\system32\cmd.exe
Star-Sharp Operation 2-level logic minimization program.
Reading file node5.blif...
Minimizing logic functions

*****
Input Cover 0: X X 0 0
Input Cover 1: 1 1 0 X
Input Cover 2: 1 X 1 1
Input Cover 3: 1 0 X 0

*****
Actual PIs 0: X X 0 0
Actual PIs 1: 1 1 0 X
Actual PIs 2: 1 X 1 1
Actual PIs 3: 1 0 X 0
Actual PIs 4: 1 1 X 1
Actual PIs 5: 1 0 1 X

*****
Essential PIs 0: X X 0 0

*****
Minimal Cover 0: X X 0 0
Minimal Cover 1: 1 1 X 1
Minimal Cover 2: 1 0 1 X

Report:
Function 1: #inputs = 4; #cubes = 3; cost = 15
Done.
Press any key to continue . . .
```

Figure 5: Node5 Test Result

Limitations:

Within our implementation, the cover set variables (number of cubes that a function can hold) are declared in a fixed size, which it might have some performance bottleneck if using this program to calculate some large test nodes. For example, with this fixed cover size, the program might not be able to perform minimization if there are a lot of inputs. But if we set the fixed cover size too large, there might be some wasted memory allocation which will not be needed. During the program execution, we have to iterate the code multiple times, and we tried to do

dynamically memory allocations to the covers, but this will cause some memory allocation issue from malloc function.

The choice of nonessential prime implicants to be included in the cover is often not obvious. Besides, we need to consider the cost. For functions with many inputs, there are a lot of possibilities, and some heuristic approach has to be used. Our branching heuristic strategy in selecting the non-EPIs to be included in the EPIs can be summarized below:

- Sort the non-EPIs in ascending order in terms of cube cost, and start from the lowest cube cost non-EPI to check if it covers all the remaining uncovered ON cubes: If yes, add this non-EPI into the EPI set, and finish the minimization; if non-EPI covers partial uncovered ON cubes, add this non-EPI into the EPI set and then continue with the second lowest cost non-EPI.
- Keep looping until it covers the whole ON-set cover
- Then we can calculate the minimal cover cost
- Finding a good minimal cover: if the given ON-set is not covered by the essential prim implicants, we developed a simple algorithm to find a better minimal cover. First, we find out the number of cubes with both lowest cost, and second lowest cost respectively. Then we choose the greater number as the iterations of the optimization loop, and start with different lowest cost or second lowest non-EPIs, then iterate the above code. Then we compare the previous cover and the new cover, and keep the lower cost one.

Conclusion:

We successfully implemented the software program by utilizing “star” and “sharp” cubical technique to calculate the minimal cost. Even though the minimization technique has few limitations, but the results are accurate and the minimal cost is acceptable.