TUM School of Computation, Information and Technology
Technical University of Munich

Technical Report

A Two-Step Algorithm for Microfluidic Large-Scale Integration Test Module Design

Jiahui Peng

Examiner:
Prof. Dr.-Ing. Ulf Schlichtmann

Supervisor:
Mengchu Li

Submitted:
Munich, 30.10.2023

Contents

Abstract

For high-throughput bio-applications, microfluidic large-scale integration (mLSI) shows a promising future. Leakage and blockage defects tend to occur on the control channels and may result in unexpected valve behavior and erroneous experimental results. Fortunately, these defects can be checked after production and before experimentation. Conventional testing can be performed by testing each valve. The first self-testing method for mLSLs designed with an integrated test module is presented in [1]. This method improves efficiency by testing the control channels instead of the previous approach of testing individual valves. However, in real applications, the actual test circuit does not need to test all possibilities for leakage, because some control channels are physically so far away from each other that leakage is essentially absent. This report therefore builds on this foundation, proposing a simplified two-step algorithm to achieve a more personalized test design strategy combining MILP modelling and greedy algorithm. As a result, in the first step it may have the chance to reduce the test number of leakage testing by generating new channel codes from the user input of combinations of channel numbers that may be compromised. Then the test groups of testing blockages are automatically generated for the user by the second step. This two-step algorithm can, in general, reduce the construct costs for independent test modules by decreasing the number of valves, and at the same time having large chances to reduce testing efforts for both leakage and blockage testing.

Key Words: Microfluidic Chips, Defect Testing, MILP Implementation

1 Introduction

Microfluidic large-scale integration (mLSI) is a promising platform for biological experiment conducting used in many application aspects. By integrating a large number of valves on the chip, it enables a combination of fluidic reactions. It focuses on ensuring the accuracy of the results of biological experiments, i.e., that different fluids can come into contact with the reaction in a precise and timely manner.

In order to further explain the factors affecting the reaction accuracy, mLSI's structure and working principle have to be introduced first. It is worth mentioning that this report only gives a brief overview of the principles, and the specific explanations should be referred to [1]. For this application scenario, there are generally two layers, one is the flow layer, which is the liquid layer responsible for providing the liquid to be reacted. The other layer is the control layer, which controls the flow and reaction of the liquid. The control layer provides control through the transmission of pressure to the membrane in the middle of the valve, which closes after pressure is applied, thus making the valve closed.

Control channel, however, is more prone to defects because of the small size. There are two general types of defects, blockages and leakages. Blockage means that pressure cannot be transmitted and therefore the channel cannot be pressurized and the valves cannot be closed. Leakage means that the channels are accidentally linked to each other, so that even if one channel is not pressurized, pressure may leak out of the other channel to achieve a pressurized result.

[1] presents a basic binary testing methodology for testing for blockages and leakages, which this report builds upon. The basic idea is that the channels are each uniquely given a binary code from one to the needed number and with a 'one' in the binary coding expression, there is a valve later for construction. Then, depending on the location and characteristics of the valves, test experiments can be designed:

For the leakage testing, the number of bits in the binary code is the number of times the test needs to be conducted. For example, each time, for testing one bit, all the channels with coding '1' on this bit which means having valves on this row will be not pressured and the channels without valves will be pressured. In this case, if the channels are perfect with no defects, theoretically the channel is perfectly smooth, as a result, the fluid will pass the channel and the end will detect the fluid. If there is a leak between any pair of valved and unvalved passages, pressure will be passed from the unvalved passage to the valved passage. Thus, this will make a valve that should not have closed to close. As a result, the fluid will not flow smoothly through the channel to reach the end. On this bit line, it is possible to detect if there is a leak between a channel with a valve and a channel without a valve on this line. As each channel has unique coding, any two channels will have at least one bit of difference between their codes, so any two valves will be tested for leakage in at least one of the sets of tests.

As for the blockage testing, if a blockage happens on one channel, it loses the ability to detect the pressure behavior, meaning that the valves for this channel can will never be closed regardless of pressure. The testing point is to test whether some valve on the channel still let the fluid pass even with pressure. In [1], the paper takes two neighboring channels as a pair with one smaller odd number code channel and one even code channel and then pressure are added to both the channels. For the odd channel, it will have a valve at the least bit and as the odd channel is bigger than it, it will have at least one bit with valve that the odd one has no valve. In this situation, the blockage situation for both channels can be detected through different bit ends. A blockage results in fluid passing through the channel of that bit.

In this report, a new two-step algorithm is developed from above. The problem with the testing method in [1] is that in most cases, not all the channel pairs have the possibility for leakage, therefore assigning unique numbers to all channels is redundant. And for test groups for blockage, actually more channels can be combined in one group instead of only two channels. Therefore, in the new algorithm, two channels can be given exactly the same binary code if there is absolutely no possibility of leakage between them. And after binary coding accordingly, the test groups for blockage will be generated automatically, combining more channels than two, up to the bit number as a pair in one group.

Specific algorithmic details will be presented in the next part and the experimental design and results will be presented in the third part. Conclusions are drawn in the fourth part, and the fifth part will summarize this internship work and look into the future. There is also a sixth part for reference.

2 Model Design

2.1 Leakage Testing Model Design

In this section, the experimentally used model 2.1.1 and an introductory-only supplementary model 2.1.2 are presented.

2.1.1 Leakage Testing Model

Input:

1. Supposing the number of to-be-tested control channels is denoted as an integer number *n*;
2. For each control channel k with $1 \le k \le n$, a set of control channels that may have leakage defect with channel k: L_k.

Preparation:

1. For the worst case, *n* channels can be represented by $\lceil \log_2(n+1) \rceil$ binary digits, which is denoted as δ;
2. *M* is considered as a large number later for constraint construction.

Variable Declaration:

1. $\forall k \in [1, n]$, δ binary variables $b_{k,i}$ with $i \in [1, \delta]$ are declared to represent the value of the i^{th} binary digit of channel k;
2. $\forall k \in [1, n]$, *n* binary variables $x_{k,j}$ with $j \in [1, n]$ are used to show whether the k^{th} channel after binary decimal conversion is coded as integer *j*;
3. $\forall i \in [1, \delta]$, there are total δ binary variables t_i meaning if this bit position are useful, in other words, if at least one of the *n* channels has been coded with a "one" on this bit;
4. $\forall i \in [1, \delta]$, there are also total δ integer variable sum_i with lower bound 0 and upper bound *n*, counting the number of 1's on the i^{th} bit, also, the number of valves later on the bit;
5. There is an integer variable that equals the maximum value among the δ sum_i variables, which has the same range as sum_i variables and is denoted as *max_value.*

Constraints Declaration:

To begin with, avoiding duplication, for two channels, k and l, only the combinations where l is greater than k are considered once. The same coding cannot be given to l and k if they have the possibility of leakage:

$$x_{k,j} + x_{l,j} \le 1, \forall j \in [1, n], \forall k \in [1, n], \forall l \in [k+1, n] \text{ and } l \in L_k$$

And a channel can only have one coding result:

$$\sum_{1 \le j \le n} x_{k,j} = 1, \forall k \in [1, n]$$

Then the binary form needs to be expressed:

$$x_{k,j} = 1 \Rightarrow \sum_{1 \le i \le \delta} b_{k,i} \cdot 2^i = j, \forall j \in [1, \mathrm{n}], \forall k \in [1, \mathrm{n}]$$

Which can be illustrated as equation combination:

$$\sum_{1\le i\le\delta} b_{k,i}\cdot 2^i \le j+(1-x_{k,j})\cdot M, \forall j \in [1,\mathrm{n}], \forall k \in [1,\mathrm{n}]$$
$$\sum_{1\le i\le\delta} b_{k,i}\cdot 2^i \ge j-(1-x_{k,j})\cdot M, \forall j \in [1,\mathrm{n}], \forall k \in [1,\mathrm{n}]$$

Further, constraints are made to get t_i:

$$\sum_{1\le k\le n} b_{k,i} = 0 \Rightarrow t_i = 0, \forall i \in [1,\delta]$$

Which equals:

$$\sum_{1\le k\le n} b_{k,i} \le t_i\cdot M, \forall i \in [1,\delta]$$

The valves on one bit are calculated for optimization:

$$sum_i = \sum_{1\le k\le n} b_{k,i}, \forall i \in [1,\delta]$$

And then get the maximum number of valves among variables sum_i, calling it as *max_value.*

Objective:
This model has several optimization objects with priority: the coding bit number is considered as the most important, and then the total valve number. These two can minimize the cost of building the test module. Minimizing the maximum number of valves among all bits is also the goal we want to optimize. This allows the valves to be distributed as evenly as possible, which facilitates the subsequent grouping of the blockage model. The object can be expressed as :

$$Minimize\colon \alpha \sum_{1\le i\le\delta} t_i + \beta \sum_{1\le i\le\delta} sum_i + \gamma\cdot max\_value$$

The above weights can be assigned as needed.
Output: coding result *code*, binary bit number *q*.

Then leakage can be tested as the method mentioned in [1], with *q* testing times. This is not in fact the most optimal result for testing times, because when the channels are finally coded as *q* bits, in some cases *q*-1 times to test leakage is enough. For example, under the situation for 7 channels with the leakage impossibility pairs (1,5), (2,6) and (3,7), the channels can be coded as “001”, “010”, “100”, “011”, “001”, “010”, “100”. This uses 3 bits coding but only need to test two times . The reason is that for testing, an all zero situation such as 00 is enough but not proper for channel design. All 0 equals no valve, meaning that the channel cannot close with pressure. For the all 0's, the actual design has to add another 1 as valve to ensure functionality. This very small difference in the number of tests, however, can be ignored in the application and the above model result can be then considered as the most optimal. As a supplementary introduction, the model for obtaining the optimal group is also presented in 2.2, but this extra step is not used in the experiments reported later because of the really tiny improvement in results.

2.1.2 Leakage Testing Complementary Model

Input:

1. Coding result *code* from 2.1.1
2. Final binary bit number q from 2.1.1
3. The same integer number n of to-be-tested control channels as 2.1.1
4. For each control channel k with $1 \leq k \leq n$, a set of control channels that may have leakage defect with channel k: L_k.

Preparation:

1. $\forall k \in [1, n]$, $\forall l \in [1, n]$, $\forall z \in [1, q]$, $xor_{k,l,z}$ means the xor relationship between the k^{th} channel and the l^{th} channel at the z^{th} *bit.*

Variable Declaration:

1. $\forall z \in [1, q]$, total q binary variables $test_z$ are declared to represent if the z^{th} bit need to be tested.

Constraints Declaration:

Same as before, to avoid duplication, for two channels, k and l, only the combinations where l is greater than k are considered once. For two channels that may have leakage defects, they need to be tested at at least one bit:

$$\sum_{1 \leq z \leq q} xor_{k,l,z} \cdot test_z \geq 1 \text{ , } \forall k \in [1, n], \forall l \in [k+1, n] \text{ and } l \in L_k$$

Objective:

This simple model aims to make the test time number the smallest, so the objective can be expressed as:

$$Minimize: \sum_{1 \leq z \leq q} test_z$$

However, after 2.2.1, the objective result can be known as either q or $q\text{-}1$.

2.2 Blockage Testing Model Design

It is found that the modeling approach is slower when the size of the computation becomes larger, so faster algorithms can be used without pursuing the optimization of the solution in large-scale computations. In this part, two algorithms are proposed, one for MILP modelling and the other as an alternative greedy algorithm.

2.2.1 MILP Modelling Solution

Input:

1. u is the given group number upper limit;
2. The total channel number is still denoted as n;
3. The new needed bit number from 2.1 is q;
4. The channel coding result *code* from leakage testing algorithm in 2.1 is also introduced.

Preparation:

1. $\forall k \in [1, n]$ and $\forall p \in [1, q]$, $c_{k,p}$ represents the binary coding result for the k^{th} channel on the p^{th} bit;
2. $\forall p \in [1, q]$, m_p is the total valve number on the p^{th} bit (1's number) minus 1;
3. M is considered as a large number later for constraint construction.

Variable Declaration:

1. $\forall k \in [1, n]$, u binary variables $g_{k,w}$ with $w \in [1, u]$ are declared to represent if the k^{th} channel is assigned to the w^{th} group;
2. $\forall w \in [1, u]$, total u binary variables f_w are introduced to represent whether the w^{th} group is used for assignment;
3. $\forall k \in [1, n]$, $\forall w \in [1, u]$, $\forall p \in [1, q]$, $v_{k,w,p}$ and $v_{k,w,0}$ are some binary variables later for constraints construction.

Constraints Declaration:

For each group, at most q channels can be held in it:

$$\sum_{1\leq k\leq n} g_{k,w} \leq q, \forall w \in [1, u]$$

For each channel, it can only be assigned into one group:

$$\sum_{1\leq w\leq u} g_{k,w} = 1, \forall k \in [1, n]$$

About the assigning rules, for a certain channel k, it has some certain bits coded with 1, this bit position set is denoted as P_k. If it is assigned to group w, for the other channels k' that are also assigned to w, this set without k is called W_k. They need to satisfy that for at least one bit coded with 1 for channel k, the other channels in W_k must be all coded with 0 on this bit. This can be expressed as:

$$g_{k,w} = 1 \Rightarrow \prod_{p \in P_k} \left(\sum_{k' \in Wk} c_{k',p} \right) = 0 \text{ , } \forall\, k \in [1, n], \forall w \in [1, u], \forall p \in [1, q]$$

Then it can be further written as:

$$g_{k,w} \cdot \prod_{p \in P_k} \left(\sum_{k' \in W_k} c_{k',p} \right) = 0 \,, \forall\, k \in [1, n], \forall w \in [1, u], \forall p \in [1, q]$$

Further using former variables, k' ∈ [1, n] represents the channels except k, $\forall k$ ∈ [1, n] and $\forall w$ ∈ [1, u], above equation can be separated into binary equation combinations:

$$g_{k,w} \leq v_{k,w,0}$$
$$\sum c_{k',p} \cdot g_{k',w} \leq v_{k,w,p} \cdot m_p \,, \forall p \in [1, q] \, and \, p \in P_k$$
$$v_{k,w,0} + \sum_{1 \leq p \leq q} v_{k,w,p} = \sum_{1 \leq p \leq q} c_{k,p}$$

Also, to record the total group number, if the w^{th} group is occupied, f_w will have the value of 1:

$$\sum_{1 \leq k \leq n} g_{k,w} \neq 0 \Rightarrow f_w = 1 \,, \forall w \in [1, u]$$

This can be further written as:

$$\sum_{1 \leq k \leq n} g_{k,w} \leq 0 + f_w \cdot M \,, \forall w \in [1, u]$$

Objective:
The object of this model is to minimize the testing groups' number, thus the objective function is:

$$Minimize\!: \sum_{1 \leq w \leq u} f_w$$

Output: grouping result

2.2.2 Alternative Solution

The basic idea of grouping is to make each channel in the group has a unique encoding of 1 in at least one bit, meaning that in that one bit, all channels except it are encoded with 0. This alternative solution uses a greedy algorithm that firstly sorts all channels from largest to smallest based on the number of valves, then adds the ones that match the criteria to a group in order, and then keeps grouping until the end. The pseudocode is introduced below:

Input: channel number n, bit number q, and the channel coding result *code* from leakage testing algorithm in 2.1

Preparation: Each channel is initialized with several properties:

Variable Name:	Explanation:
assigned_notice	1 representing the channel has been allocated to groups
coding	The coding result for this channel
group	The group number it has been allocated
satisfied_bits	Which bit can be used for testing

Among them, *coding* is from *code* and the others will be initialized as 0 or none. The channels are also firstly sorted from largest to smallest according to the valves that they have and the sorted list is called *sorted_array*.

Pseudo code:

```
while sorted_array do
  element_allocating ← sorted_array[0]
  get ← 0
  if element_allocating.assigned_notice == 0 then
    for each object in sorted_array do
      if sum (element_allocating.coding[h]· object.coding[h]) < sum (element_allocating.coding[h]) for (h ← 0 to h < q by h++) and sum ((1 - element_allocating.coding[h])· object.codingt[h])≥ 1for (h ← 0 to h < q by h++) then
        add [element_allocating, object] to group
        group_id ← len(group)
        remove element_allocating and object from sorted_array
        element_allocating.assigned_notice ← 1
        get ← 1
        for (h ← 0 to h < q by h++) do
          if element_allocating.coding[h] == 1 and object.coding[h] == 0 then
            add h to element_allocating.satisfied_bits
          if element_allocating.coding[h] == 0 and object.coding[h] == 1 then
            add h to object.satisfied_bits

          check[h] = element_allocating.coding[h] + object.coding[h]
          if check[h] > 1 then
            check[h] ← 1
        if sum(check) == q then
          break
        else then
          add a new channel object with coding as check and assigned_notice as 1 on the top of sorted_array
          break

  else then
    for each object in sorted_array do
      if sum ((1 - element_allocating.coding[h])· object.coding[h])≥ 1for (h ← 0 to h < q by h++) then
        previous_ok ← 1
        for item in group[element_allocating.group-1] do
          num ← len(item.satisfied_bits)
          for (h ← 0 to h < q by h++) do
            if object.coding[h] == 1 and h is in item.satisfied_bits then
              num ← num -1
          if num == 0 then
            previous_ok ← 0

        if previous_ok == 1 then
          element_allocating.assigned_notice ← 1
          get ← 1
          for item in group[element_allocatin.group-1] do
```

```
                if object.coding[h] == 1 and h is in item.satisfied_bits then
                    remove h from item.satisfied_bits
            if element_allocating.coding[h] == 0 and object.coding[h] == 1 then
                add h to object.satisfied_bits
            add object in group[element_allocatin.group-1]
            remove element_allocating and object from sorted_array
            for (h ← 0 to h < q by h++) do
                check[h] = element_allocating.coding[h] + object.coding[h]
                if check[h] > 1 then
                    check[h] ← 1
            if sum(check) == q then
                break
            else then
                add a new channel object with coding as check on the top of sorted_array
                break

    if get == 0 then
        if element_allocating.assigned_notice == 0 then
            add [element_allocating] to group
            remove element_allocating from sorted_array
        else then
            remove element_allocating from sorted_array

Output: group
```

3 Experiment Design and Results

3.1 Experiments Design

Two experiments including both easy cases and complicated cases have been made to test the two-step algorithm and specific experimental data and results can be found in the Appendix. The parameters used in the experiment are listed in Table 1:

Device, software or parameters:	Value:
CPU	i5-8250u
Gurobi version	10.0.1
Python version	3.9.7
M	100000
α	1000
β	5
γ	1

Table 1: Experiment parameters

3.1.1 Test Case 1: Random Leakage Generating

In this experiment design, for n channels, there are total $\frac{n(n-1)}{2}$ pairs, 10%, 30%, 50% and 100% possible leakage pairs are generated randomly for 15, 30, 45 and 60 channels.

3.1.2 Test Case 2: Neighboring Leakage Generating

This simulates the situation most likely happens in real applications because if the distance between the channels is far enough, leakage basically cannot happen. For the cases of 15, 30, 45, and 60 channels, 2.3.4.5 neighboring channels at risk of leakage are simulated respectively.

3.2 Experiment Results

3.2.1 Results of Leakage Test Modelling

First, the experimentation of the algorithm is completed according to the original parameter design. For the leakage testing part, the numbers of coding bits for test case 1 and test case 2 compared to baseline from [1] are shown in Figure 1 and Figure 2. When comparing the coding bits, it can be seen that except for the extreme case where 100% of pairs need to be considered for leakage as in the case of baseline, the number of bits is reduced in all other cases. In the worst-case scenario, at least the outcome will remain the same.

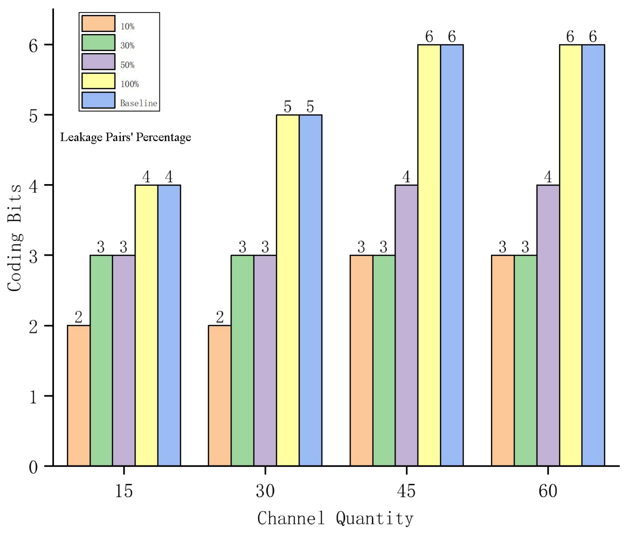

Figure 1: Coding Bits under Different Conditions (Test Case 1)

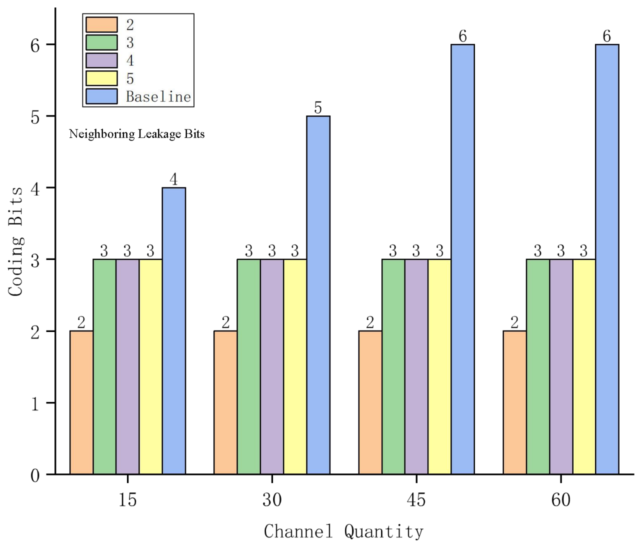

Figure 2: Coding Bits under Different Conditions (Test Case 2)

It is worth mentioning that for test case 2, which is closer to the actual simulation scenario, the enhancement is more pronounced. The number of coding bits is fixed once the number of neighboring channels that may be at risk of leakage has been determined, which is a great enhancement for high-volume applications. When it comes to n channels, the baseline can be represented by $\lceil \log_2(n+1) \rceil$ binary digits. However for n channels with neighbouring leakage defect number of a fixed μ, the coding bits can be calculated as $\lceil \log_2(\mu+2) \rceil$ bits as a constant. For example, for a constant amount of neighboring channels as in test case 2 and infinite channels, the comparison is shown in Figure 3.

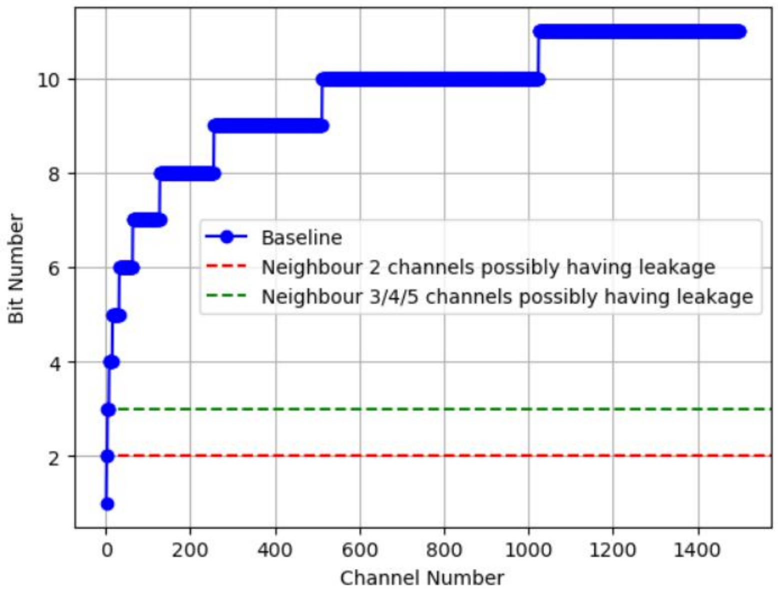

Figure 3: Bit Number for Neighboring Cases

The reduction of coding bits means that the area of testing module decreases and then the costs are also reduced. According to [1], the optimized test module area's decreasing percentage as $P_{decreased}$ for test case 1 and test case 2 are calculated in Table 2 and Table 3, supposing this $Area_t$ (mm^2) can be calculated as $0.2\rho q$, in which ρ is a constant number related to different applications and q represents the bit number. For ease of expression, test cases are numbered, for example, for test case 1, 10%, 30%, 50%, 100% for 15 channels are coded as 15-0.1, 15-0.3, 15-0.5, 15-1 respectively. For test 2, 2, 3, 4 and 5 neighbouring for also 15 channels are coded as 15-2, 15-3, 15-4. 15-5.

Case	$P_{decreased}$	Case	$P_{decreased}$	Case	$P_{decreased}$	Case	$P_{decreased}$
15-0.1	50%	30-0.1	60%	45-0.1	50%	60-0.1	50%
15-0.3	25%	30-0.3	40%	45-0.3	50%	60-0.3	50%
15-0.5	25%	30-0.5	40%	45-0.5	33.3%	60-0.5	33.3%
15-1	0%	30-1	0%	45-1	0%	60-1	0%

Table 2: Percentage of Area Decreasing for Test Case 1

Case	$P_{decreased}$	Case	$P_{decreased}$	Case	$P_{decreased}$	Case	$P_{decreased}$
15-2	50%	30-2	60%	45-2	66.7%	60-2	66.7%
15-3	25%	30-3	40%	45-3	50%	60-3	50%
15-4	25%	30-4	40%	45-4	50%	60-4	50%
15-5	25%	30-5	40%	45-5	50%	60-5	50%

Table 3: Percentage of Area Decreasing for Test Case 2

The running time for test case 1 is shown in Table 4 and test case 2 is shown in Table 5. For test case 1, as some cases are complicated and will last a really long time, the upper bound of running time is set to be 10 minutes, and this not so optimal results will have nothing to do with the smallest bits, also the smallest area, however it may have result in more blockage grouping later. For test case 2, none of the cases runs more than 6 minutes, which is a quite good result.

Case	t	Case	t	Case	t	Case	t
15-0.1	<0.1s	30-0.1	0.4s	45-0.1	5.4s	60-0.1	444s
15-0.3	0.3s	30-0.3	4.7s	45-0.3	10min	60-0.3	10min
15-0.5	0.3s	30-0.5	170s	45-0.5	10min	60-0.5	10min
15-1	<0.1s	30-1	0.5s	45-1	1.8s	60-1	2.7s

Table 4: Running Time for Test Case 1

Case	t	Case	t	Case	t	Case	t
15-2	<0.1s	30-2	0.3s	45-2	1.3s	60-2	3.5s
15-3	<0.1s	30-3	0.4s	45-3	7s	60-3	352s
15-4	0.1s	30-4	0.5s	45-4	5.4s	60-4	18s
15-5	0.1s	30-5	0.7s	45-5	6.5s	60-5	16s

Table 5: Running Time for Test Case 2

3.2.2 Results of Blockage Test Grouping

It should be noted that for blockage test groupings, the experiment adopted the strategy of using alternative algorithms for computation when the modeling algorithm is slow. Figure 4 and Figure 5 shows the results for test case 1 and test case 2 compared to baseline from [1]. As for the groups number, the results show that in most cases the number of groups decreases or remains the same, but there are still some times when the number of groups becomes more instead, for example, 15-0.1, 30-0.1, 15-2, 30-2. 45-2 and 60-2. The results have a common reason that the coding before reduces the bits to 2. For the grouping method in [1], every 2 channels can be assigned into one group. For 2 bits, the channel may have coding of "01", "10" and "11". Only "01" and "10" can form a pair and "11" can only act as a group itself. That is the reason for group increasing. However, 2 bits coding reduces the area a lot, especially in large cases. This sacrifice is acceptable in relation to the construction effort.

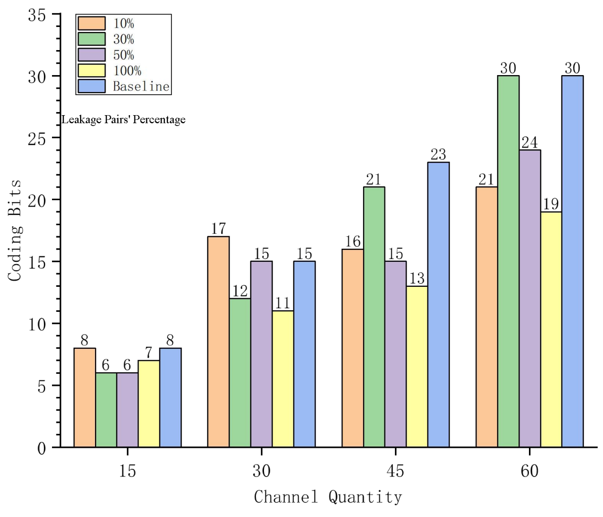

Figure 5: Groups for Testing Blockage under Different Conditions (Test Case 1)

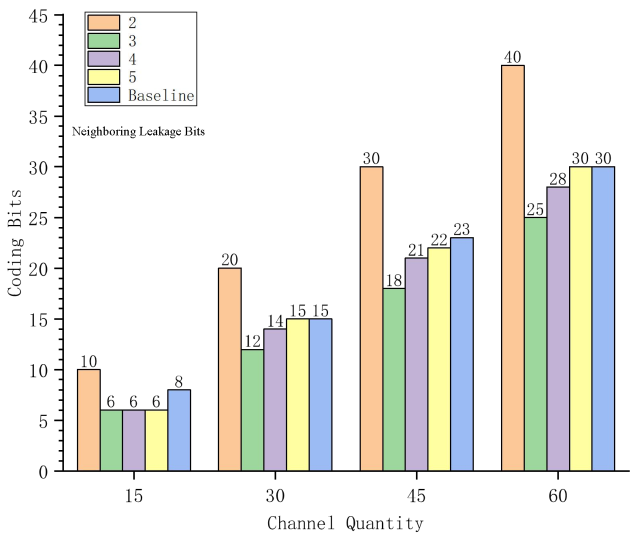

Figure 5: Groups for Testing Blockage under Different Conditions (Test Case 2)

3.2.3 Comparison of Blockage Test Modelling and Greedy Algorithm

To compare the performance of the two algorithms, two metrics were taken, time $t_{modelling}$, t_{greedy} and number of groups $n_{modeling}$ and n_{greedy} . Actually the comparison is incomplete and less computationally intensive due to the fact that the results of the modeling could not be derived when the problem size was large. The results are shown in Table 6. According to the table, it can be seen that the greedy algorithm is much faster, especially when the number of channels increases. For the number of groups, it is almost the same as the modeling results when they can be compared. However, there are isolated cases where the number of groups increases slightly by one, and since not all cases can be measured, it is not possible to determine how large the difference actually is.

Case	$t_{modelling}$	t_{greedy}	$n_{modeling}$	n_{greedy}
15-0.1	<0.1s	<0.1s	8	8
15-0.3	<0.1s		6	6
15-0.5	<0.1s		6	6
15-1	42.3s		7	7
30-0.1	31.5s		17	17
30-0.3	3224s		12	13
45-0.1	0.6s		16	16
60-0.1	1.7s		21	21
15-2	<0.1s		10	10
15-3	<0.1s		6	6
15-4	0.1s		6	7
15-5	0.1s		6	7

Table 6: Time and Group Comparison between Modelling and Greedy algorithm

4 Conclusion

In this report, a two-step algorithm for testing mLSI circuits' leakages and blockages based on [1] is proposed. It combines mathematical modelling and greedy algorithm. Compared to [1], this algorithm reduces the area and cost of building the test module by decreasing the number of coding bits and the number of valves required, and reduces the test effort with high probability.

5 Reference

1. M. Li et al., "Integrated Test Module Design for Microfluidic Large-Scale Integration," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 6, pp. 1939-1950, June 2023.

6 Personal Summarization and Future Work

Looking back on this internship experience, I achieved the actual practice of transforming knowledge from EDA class to application. Thanks to my supervisor for patiently guiding and advising me throughout the process. It meant a lot to me when I was helpless and she told me not to research on my own, but to communicate in order to make new advances.

In addition to modeling and programming skills, this internship made me realize the importance of algorithmic complexity. Earlierly I wrote an algorithm for leakage, but this algorithm took too long to compute on my computer, so I abandoned the old algorithm. However, that discarded algorithm achieves the optimal solution for the optimal solution for the number of tests with a fixed number of bits if you don't seek to minimize the number of bits. Blockage's alternative algorithm was also written because of the speed problem of the modeling method, and although it doesn't reach a strict optimal solution, it still performs well, and most importantly, it's fast. It also taught me that sometimes modeling is not the only solution, learning the algorithm is thus really important.

The downside of this internship is that blockage's modeling algorithm is slow on my computer under large-scale computation, so I wasn't able to fully compare it to alternative algorithms in terms of optimality for large-scale computation.

For future work, I hope to improve blockage's modeling algorithm to make it faster, or use better device to compare the two algorithms in full depth. I also wish there was a better way to deal with very large scale leakage problems, as modeling methods have speed limitations when the problem size gets bigger.