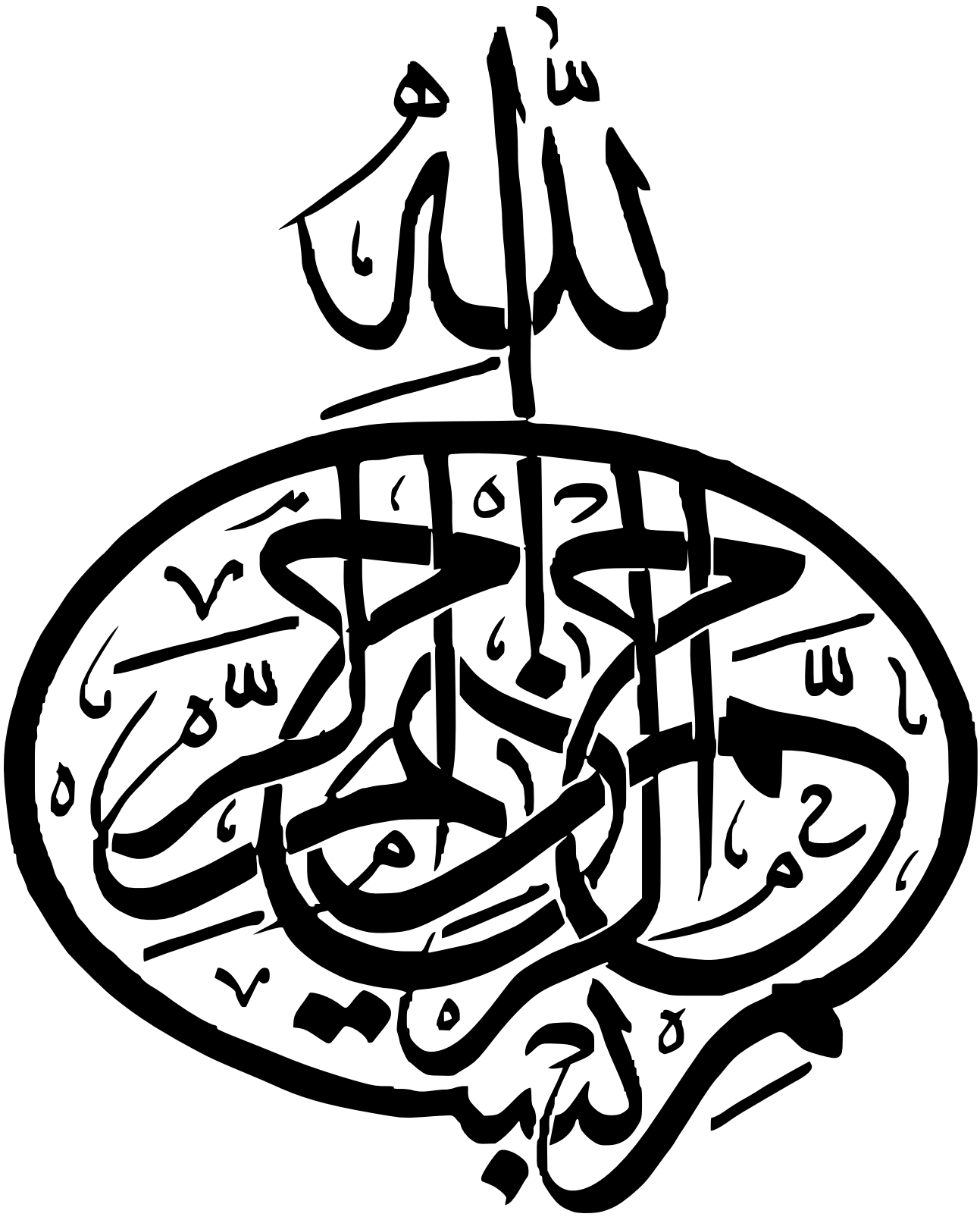


دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

درس آزمون پذیری
مهندسی کامپیوتر

گزارش پروژه - فاز دوم

نگارش
سیدمحمد روزگار - ۴۰۱۲۰۳۳۸۲

استاد درس
دکتر شاهین حسابی

آذر ۱۴۰۲

فهرست مطالب

۱ مقدمه و شرح مسئله

در این گزارش قصد داریم به بررسی فاز دوم پروژه درس آزمون‌پذیری بپردازیم. در این پروژه فایل‌هایی با پسوند bench که حاوی اطلاعاتی در مورد ورودی، خروجی و همچنین گیت‌های مدار است را خوانده و موارد زیر را اجرا می‌کنیم:

۱. در گام اول ابتدا بر اساس الگوریتم استنتاجی شبیه‌سازی اشکال [۱] که در فاز قبلی به طور مفصل به آن پرداخته شد، به تعداد ۲^n ، که n همان ورودی‌های مدار است (روش جامع [۲]) ، ورودی به مدار تزریق می‌کنیم و در هر گام اشکالات چسبیدگی تکی [۳] ای که در خروجی مدار پدیدار شده‌اند را ذخیره کرده و در نهایت این بردارهای تست را به همراه اشکالات شناسایی شده توسط هر بردار تست را در یک فایل با پسوند csv ذخیره می‌کنیم تا در نهایت بتوانیم به Fault Dictionary دست یابیم.

۲. در گام بعدی بر اساس قوانین موجود در معادل‌بودن اشکال [۴] تعداد اشکالات موجود در مدار را کاهش داده و بر اساس این قوانین مجدداً Fault Dictionary را تشکیل می‌دهیم. همچنین در این گام بردارهای تست ضروری یا Essential Test Vectors را نیز مشخص کرده و همه را در یک فایل csv دیگر ذخیره می کنیم.

۲ مراحل تولید فایل دیکشنری اشکال

در این قسمت قصد داریم درمورد تولید فایل دیکشنری اشکال صحبت کنیم. از این‌رو، در ابتدا بردارهای تست را به روش جامع به مدار تزریق کرده و در ادامه برای هر بردار تست، اشکالاتی که در خروجی نمایان می‌شود را مشخص کنیم.

۱.۲ تزریق بردارهای تست به‌صورت جامع

همانطور که گفته شد، نیاز است در گام اول، بردارهای تست را به صورت جامع به مدار تزریق کنیم. قطعه کدی که این موضوع را بیان می‌کند به شرح زیر است:

```
1  for test_vector in range(2**len(self.__network_controller.input_gates)):
2          test_vector_bin: str = bin(test_vector)[2:].zfill(
3                  len(self.__network_controller.input_gates)
4          )
5
6          test_vector_list: list[str] = list(test_vector_bin)
7
8          self.__network_controller.inject_and_execute(
9                  inject_values=test_vector_list
10         )
11         self.__fault_simulation_controller.run()
12         self.__detected_fault_dict[test_vector_bin] = self.
               __fault_simulation_controller.detectable_faults
```

[1] Deductive fault simulation
[2] Exhaustive
[3] Stuck-at-fault
[4] Fault Equivalence

```
13
14          self.__fault_simulation_controller.reset()
15          self.__network_controller.reset()
```

بر اساس قطعه کد بالا، نیاز است به تعداد ۲^n که n تعداد ورودی های مدار است، ورودی باینری مختلف به مدار تزریق کنیم (روش جامع). در هر گام پس از تزریق ورودی مربوطه، لیست اشکالاتی که توسط بردار تست مربوطه تولید می شود را به عنوان یک المان از یک دیکشنری ذخیره می کنیم. در نهایت کنترلرهای مربوطه را جهت تزریق ورودی بعدی، reset می کنیم.

۲.۲ تشکیل جدول اشکال

درگام بعد، پس از مشخص شدن اینکه هر بردار تست، چه اشکالاتی را کشف می کند، نیاز است را تولید کنیم. تولید دیکشنری اشکال به این صورت است که بر اساس دیکشنری که مرحله قبل بدست آوردیم، آن را در یک فایل csv ذخیره می کنیم. به جهت سادگی کد مربوط به این قسمت، از آوردن کد اجتناب کرده و تنها به خروجی که از مدار c17 بدست آورده ایم، اکتفا می کنیم:

test vector	10_s-a-1	7_s-a-0	3_1_s-a-1	1_s-a-0	16_s-a-1	2_s-a-1	16_1_s-a-1	22_s-a-0	19_s-a-1	11_2_s-a-0	19_s-a-0	16_1_s-a-0	6_s-a-0	11_2_s-a-1	1_s-a-1	2_s-a-0	16_2_s-a-1	23_s-a-1	6_s-a-1	3_1_s-a-0	22_s-a-1	23_s-a-0	11_1_s-a-1	3_2_s-a-0	3_s-a-1	11_s-a-0	11_s-a-1	16_s-a-0	11_1_s-a-0	3_2_s-a-1	16_2_s-a-0	3_s-a-0	7_s-a-1	10_s-a-0
00000																					Y						Y			Y		Y	Y
00001		Y				Y			Y	Y		Y									Y	Y				Y		Y						
00010						Y					Y	Y						Y			Y							Y			Y		Y	Y
00011		Y				Y			Y	Y		Y									Y	Y			Y	Y		Y		Y				Y
00100						Y					Y	Y			Y			Y			Y							Y			Y		Y	Y
00101		Y				Y			Y	Y		Y			Y				Y		Y	Y				Y		Y						Y
00110											Y	Y			Y			Y			Y							Y			Y			Y
00111											Y	Y	Y	Y	Y			Y			Y			Y			Y	Y			Y	Y		Y
01000					Y		Y	Y								Y	Y					Y				Y			Y					
01001					Y		Y	Y								Y						Y				Y			Y					
01010					Y		Y	Y								Y	Y					Y			Y	Y			Y	Y				
01011					Y		Y	Y								Y						Y			Y	Y			Y	Y				
01100					Y		Y	Y								Y	Y		Y			Y				Y			Y					
01101					Y		Y	Y								Y			Y			Y				Y			Y					
01110											Y	Y	Y		Y			Y			Y		Y	Y			Y	Y			Y	Y		Y
01111											Y	Y	Y	Y	Y			Y			Y		Y	Y			Y	Y			Y	Y		Y
10000			Y			Y					Y	Y						Y			Y				Y			Y			Y		Y	Y
10001		Y	Y			Y			Y	Y		Y									Y	Y			Y	Y		Y						Y
10010			Y			Y					Y	Y						Y			Y				Y			Y			Y		Y	Y
10011		Y	Y			Y			Y	Y		Y									Y	Y			Y	Y		Y		Y				Y
10100	Y			Y		Y		Y			Y							Y		Y								Y			Y	Y	Y	
10101	Y	Y		Y				Y	Y	Y									Y	Y		Y				Y						Y		
10110	Y			Y				Y			Y							Y		Y								Y			Y	Y		
10111	Y			Y				Y			Y		Y	Y				Y		Y				Y			Y	Y			Y	Y		
11000					Y		Y	Y								Y	Y					Y				Y			Y					
11001					Y		Y	Y								Y						Y				Y			Y					
11010					Y		Y	Y								Y	Y					Y			Y	Y			Y	Y				
11011					Y		Y	Y								Y						Y			Y	Y			Y	Y				
11100					Y			Y								Y	Y		Y			Y				Y			Y					
11101								Y											Y			Y				Y								
11110	Y			Y				Y			Y		Y					Y		Y			Y	Y			Y	Y			Y	Y		
11111	Y			Y				Y			Y		Y	Y				Y		Y			Y	Y			Y	Y			Y	Y		

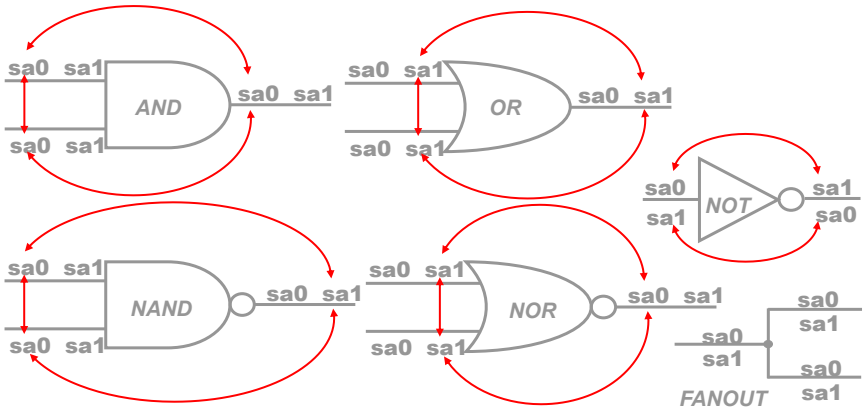
جدول ۱: جدول اشکال مدار c17 قبل از ادغام

۳ ادغام اشکال ها و تشکیل مجدد دیکشنری اشکال

در این گام نیاز است که اشکالاتی که در مراحل قبل کشف شده‌اند، را ادغام کنیم. برای این منظور با توجه به این که اشکالات با هم معادل هستند(Fault Collapsing) ، آن ها را با هم ادغام می‌کنیم.

۱.۳ مشخص کردن اشکالات معادل

در این گام نیاز است که اشکالات معادل را مشخص کنیم. برای این منظور مشابه با شکل زیر عمل می کنیم:



شکل ۱: چگونگی ادغام اشکال ها در هر گیت منطقی

کد نوشته شده برای این قسمت به شرح زیر است:

```
1  class FaultCollapseOperation(Operation):
2          @classmethod
3          def buffer_operation(cls, gate: BufferGate) -> list[tuple[str,
               list[str]]]:
4          return [
5                  (
6                          f'{gate.output_wires[0].id}_s-a-0',
7                          [f'{gate.input_wires[0].id}_s-a-0']
8                  ),
9                  (
10                         f'{gate.output_wires[0].id}_s-a-1',
11                         [f'{gate.input_wires[0].id}_s-a-1']
12                 ),
13         ]
14
15         @classmethod
16         def not_operation(cls, gate: NotGate) -> list[tuple[str, list[str
               ]]]:
17         return [
18                 (
19                         f'{gate.output_wires[0].id}_s-a-0',
20                         [f'{gate.input_wires[0].id}_s-a-1']
21                 ),
22                 (
23                         f'{gate.output_wires[0].id}_s-a-1',
24                         [f'{gate.input_wires[0].id}_s-a-0']
25                 ),
26         ]
27
```

```
28          @classmethod
29          def and_operation(cls, gate: AndGate) -> list[tuple[str, list[str
                ]]]:
30          return [
31                  (
32                          f'{gate.output_wires[0].id}_s-a-0',
33                          [f'{input_wire.id}_s-a-0' for input_wire in gate.
                                input_wires]
34                  ),
35          ]
36
37          @classmethod
38          def or_operation(cls, gate: OrGate) -> list[tuple[str, list[str
                ]]]:
39          return [
40                  (
41                          f'{gate.output_wires[0].id}_s-a-1',
42                          [f'{input_wire.id}_s-a-1' for input_wire in gate.
                                input_wires]
43                  )
44          ]
45
46          @classmethod
47          def nand_operation(cls, gate: NandGate) -> list[tuple[str, list[
                str]]]:
48          return [
49                  (
50                          f'{gate.output_wires[0].id}_s-a-1',
51                          [f'{input_wire.id}_s-a-0' for input_wire in gate.
                                input_wires]
52                  )
53          ]
54
55          @classmethod
56          def nor_operation(cls, gate: NorGate) -> list[tuple[str, list[str
                ]]]:
57                  return [
58                  (
59                          f'{gate.output_wires[0].id}_s-a-0',
60                          [f'{input_wire.id}_s-a-1' for input_wire in gate.
                                input_wires]
```

```
61                  )
62                  ]
63
64          @classmethod
65          def xor_operation(cls, gate: XorGate) -> list[tuple[str, list[str
                ]]]:
66                  return []
67
68          @classmethod
69          def xor_operation(cls, gate: XnorGate) -> list[tuple[str, list[
                str]]]:
70                  return []
71
72          @classmethod
73          def fanout_operation(cls, gate: FanoutGate) -> list[tuple[str,
                list[str]]]:
74                  return []
```

همانطور که از کد بالا مشخص است، نیاز است خطاهای معادل هر خط ورودی گیت را در صورت لزوم مشخص کنیم. حال نیاز است، اشکالات معادل را بر مدار اعمال کنیم. برای این منظور از کد زیر زیر استفاده می کنیم:

```
1  def apply_fault_collapse(self) -> None:
2          assert self.__equivalent_fault_dict
3
4          for test_vector_bin, detected_faults in self.
               __detected_fault_dict.items():
5                  temp_detected_faults: list[str] = list(detected_faults)
6
7          for gate_level in range(self.__network_controller.
               max_network_level):
8          for _, gates in self.__network_controller.total_gates_with_level.
               items():
9          for gate in gates:
10         if not isinstance(gate, FanoutGate):
11                 for input_wire in gate.input_wires:
12                         s_a_0_fault: str = f'{input_wire.id}_s-a-0'
13                         s_a_1_fault: str = f'{input_wire.id}_s-a-1'
14
15                         if s_a_0_fault in temp_detected_faults:
16                                 temp_detected_faults[temp_detected_faults
                                       .index(
17                                             s_a_0_fault)
18                                 ] = self.__get_eqivalent_fault(fault_name
```

```
                                        =s_a_0_fault)
19
20                          if s_a_1_fault in temp_detected_faults:
21                                  temp_detected_faults[temp_detected_faults
                                        .index(
22                                          s_a_1_fault)
23                                  ] = self.__get_eqivalent_fault(fault_name
                                        =s_a_1_fault)
24
25                  self.__detected_fault_dict[test_vector_bin] = set(
26                          temp_detected_faults
27                  )
```

۲.۳ مشخص کردن بردار های تست ضروری

همچنین نیاز است، بردار های تست ضروری [۵] را نیز مشخص کنیم. روش کار به این صورت است که اگر یک اشکال توسط یک بردار شناسایی شود، آن بردار به صورت ضروری است. برای این منظور از تکه کد زیر استفاده می کنیم.

```
1  def essential_test_vectors(self) -> list[str]:
2          detected_fault_test_vectors: dict[str: list[str]] = dict()
3          for test_vector_bin, detected_faults in self.
               __detected_fault_dict.items():
4                  for detected_fault in detected_faults:
5                          if detected_fault not in
                               detected_fault_test_vectors:
6                                  detected_fault_test_vectors[
                                       detected_fault] = [test_vector_bin]
7                          else:
8                                  detected_fault_test_vectors[
                                       detected_fault].append(test_vector_bin
                                       )
9
10         essential_test_vectors_: list[str] = list()
11
12         for _, test_vectors in detected_fault_test_vectors.items():
13                 if len(test_vectors) == 1:
14                         essential_test_vectors_.append(test_vectors[0])
15         return essential_test_vectors_
```

[5] Essential Test Vector

۳.۳ تشکیل مجدد جدول اشکال

پس از ادغام اشکالات، جدول اشکال به صورت زیر خواهد بود:

test vector	3_2_s-a-1	16_s-a-1	11_s-a-1	16_1_s-a-1	3_s-a-0	11_1_s-a-1	11_s-a-0	19_s-a-1	3_s-a-1	10_s-a-1	11_2_s-a-1	2_s-a-1	1_s-a-1	16_2_s-a-1	23_s-a-1	22_s-a-0	3_1_s-a-1	6_s-a-1	7_s-a-1	23_s-a-0	16_s-a-0	22_s-a-1	Essential / Not Essential
00000												Y			Y				Y		Y	Y	Not Essential
00001							Y	Y				Y								Y	Y	Y	Not Essential
00010												Y			Y				Y		Y	Y	Not Essential
00011	Y						Y	Y	Y			Y								Y	Y	Y	Not Essential
00100												Y	Y		Y				Y		Y	Y	Not Essential
00101							Y	Y				Y	Y					Y		Y	Y	Y	Not Essential
00110													Y		Y						Y	Y	Not Essential
00111			Y		Y						Y		Y		Y						Y	Y	Not Essential
01000		Y		Y			Y							Y		Y				Y			Not Essential
01001		Y		Y			Y									Y				Y			Not Essential
01010	Y	Y		Y			Y		Y					Y		Y				Y			Not Essential
01011	Y	Y		Y			Y		Y							Y				Y			Not Essential
01100		Y		Y			Y							Y		Y		Y		Y			Not Essential
01101		Y		Y			Y									Y		Y		Y			Not Essential
01110			Y		Y	Y							Y		Y						Y	Y	Not Essential
01111			Y		Y	Y					Y		Y		Y						Y	Y	Not Essential
10000									Y			Y			Y		Y		Y		Y	Y	Not Essential
10001							Y	Y	Y			Y					Y			Y	Y	Y	Not Essential
10010									Y			Y			Y		Y		Y		Y	Y	Not Essential
10011	Y						Y	Y	Y			Y					Y			Y	Y	Y	Not Essential
10100					Y					Y		Y			Y	Y			Y		Y		Not Essential
10101					Y		Y	Y		Y						Y		Y		Y			Not Essential
10110					Y					Y					Y	Y					Y		Not Essential
10111			Y		Y					Y	Y				Y	Y					Y		Not Essential
11000		Y		Y			Y							Y		Y				Y			Not Essential
11001		Y		Y			Y									Y				Y			Not Essential
11010	Y	Y		Y			Y		Y					Y		Y				Y			Not Essential
11011	Y	Y		Y			Y		Y							Y				Y			Not Essential
11100		Y					Y							Y		Y		Y		Y			Not Essential
11101							Y									Y		Y		Y			Not Essential
11110			Y		Y	Y				Y					Y	Y					Y		Not Essential
11111			Y		Y	Y				Y	Y				Y	Y					Y		Not Essential

جدول ۲: جدول اشکال c17 پس از ادغام اشکالات