دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

گزارش فاز ۱ پروژه درس آزمون پذیری

(طراحی و پیاده‌سازی شبیه‌ساز مدارهای ترکیبی)

نام اعضای گروه :

علی فراهانی - ۴۰۰۲۱۱۰۸۷

آوا دژبان - ۴۰۰۲۱۱۱۶۲

نام استاد: دکتر شاهین حسابی

تابستان ۱۴۰۱

فهرست مطالب

۱. تعریف پروژه

طبق تعریف پروژه قرار بود با استفاده از یکی از زبان‌های برنامه‌نویسی که ما ازپایتون استفاده کردیم ، یک شبیه‌ساز طراحی و پیاده‌سازی شود. تا فایل‌های استاندارد ISCAS را بخواند و مدار منطقی را تشخیص دهد و سپس با مقادیر ورودی که به مدار اعمال میشود، مقادیر تمام خروجی‌ها را مشخص کند. برای ایجاد شبیه‌ساز بایستی فایل های isc خوانده شود. که از لینک مربوطه دانلود کردیم.

۱.۱. نیازمندی های پروژه :

- شبیه‌ساز باید بتواند تمام گیت‌های منطقی اعم از AND, NAND, OR, NOR, XOR, XNOR, NOT, BUF, FANOUT را بخواند.
- تمام مقادیر ورودی را پشتیبانی کند ۰ و ۱ و U و Z .
- عمومی باشد یعنی بتواند هر مدار استانداردی را که با فرمت مدنظر است، دریافت کند.

۲. توضیحات کد

با توجه به تعریف پروژه در قسمت قبلی نیاز است تا فایل های مربوط به تعریف مدار ترکیبی با فرمت isc را که دانلود کردیم به وسیله پایتون پارس کنیم و سپس با استفاده از اطلاعات استخراج شده از این فایل ، مدار ترکیبی مورد نظر را ایجاد کنیم. ایجاد این مدار ترکیبی باید به گونه ای باشد که بتواند رفتار هر یک از اجزای مدار ترکیبی مورد نظر ما را مدل کند و اگر ورودی به وسیله فایل input به مدار مورد نظر اعمال شد در خروجی، مقادیر صحیح را تولید کند و فایلی به نام output ایجاد شود. (تمامی پوشه ها را در فولدر مربوطه قرار داده‌ایم.).

پوشه اصلی ما در این پروژه شامل ۲ فایل main.py و handler.py هست که توابع را در فایل handler.py تعریف کردیم و فایل main.py ورودی ها را دریافت و توابع را اجرا میکند. پوشه های ورودی و خروجی ها و فایل های isc را که دانلود کردیم در همان مسیر کد اصلی قرار دادیم.

```
import handler

circuit_name = input('Enter the name of iscas file: ')
input_file = input('Enter the name of input file: ')

iscas_result = handler.read_circuit_file(circuit_name)
input_data = handler.read_input_file("input/%s" % input_file)
output = handler.get_node_values(input_data, iscas_result)



with open('output/output_'+ circuit_name, 'w') as file:
    file.write('node' + '\t\t' + 'value\n')
    file.write('------\t------\n')
    for item in list(output.items()):
        file.write('  ' + item[0] + '   =>   ' + item[1] + '\n')
```

در کد بالا (در فایل main قرار دارد)، برای پاسخ به نیازمندی اول که در شرح پروژه آورده شد ابتدا نام مدار مربوطه با استفاده از متغیر circuit_name و اسم فایل ورودی را در خط بعدیش از متغیر input_file دریافت میکنیم.

وقتی ورودی توسط کاربر وارد میشود در ابتدا دو تابع read_circuit_file و read_input_file صدا زده میشوند. (در فایل handler قرار دارند)

```
def read_circuit_file(name):
    file = open("isc_sample_files/" + name,'r')
    lines = file.readlines()


    data = []
    flag = False

    for i in range(len(lines)):
        if(lines[i][0] != '*'):
            flag = True

        if(flag):
            splited = lines[i].split()
            if(len(splited) <= 3):
                data[-1] = data[-1] + splited
            elif(splited[2] == 'from'):
                for i in data:
                    if splited[3] in i:
                        splited.append(i[0])
                        break
                data.append(splited)
            else:
                data.append(splited)

    return data
```

```
def read_input_file(address):
    file = open(address, 'r')
    lines = file.readlines()
    data = []
    for i in range(1, len(lines)):
        splited = lines[i].split()
        if len(splited) == 1:
            splited.append('Z')
        data.append(splited)

    return data
```

هر کدام از مدارهای منطقی را که بصورت تابع تعریف کردیم (در فایل handler) را در ادامه آوردیم. تمامی گیت‌ها به گونه‌ای طراحی شده اند که هر چهار ورودی ۱ ،۰ ، Z و U را پشتیبانی می کنند که در پاسخ به نیازمندی دوم است که در شرح پروژه آوردیم.

AND Gate:

```
def AND(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == '0' or value2 =='0'):
        return '0'
    elif(value2 =='1' and value1 == '1'):
        return '1'
    elif(value1 == 'd_f' and value2 == '1') or (value2 == 'd_f' and value1 ==
'1'):
        return 'd_f'
    elif(value1 == 'd_not_f' and value2 == '1') or (value2 == 'd_not_f' and
value1 == '1'):
        return 'd_not_f'
    else:
        return 'U'
```

NAND Gate:

```
def NAND(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == '0' or value2 == '0'):
        return '1'
    elif(value2 == '1' and value1 == '1'):
        return '0'
    elif(value1 == 'd_f' and value2 == '1') or (value2 == 'd_f' and value1 ==
'1'):
        return 'd_not_f'
    elif(value1 == 'd_not_f' and value2 == '1') or (value2 == 'd_not_f' and
value1 == '1'):
        return 'd_f'
    else:
        return 'U'
```

OR Gate:

```
def OR(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == '1' or value2 == '1'):
        return '1'
    elif(value2 == '0' and value1 == '0'):
        return '0'
    elif(value1 == 'd_f' and value2 == '0') or (value2 == 'd_f' and value1 ==
'0'):
        return 'd_f'
    elif(value1 == 'd_not_f' and value2 == '0') or (value2 == 'd_not_f' and
value1 == '0'):
        return 'd_not_f'
    else:
        return 'U'
```

NOR Gate:

```
def NOR(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == '1' or value2 == '1'):
        return '0'
    elif(value2 =='0' and value1 == '0'):
        return '1'
    elif(value1 == 'd_f' and value2 == '0') or (value2 == 'd_f' and value1 ==
'0'):
        return 'd_not_f'
    elif(value1 == 'd_not_f' and value2 == '0') or (value2 == 'd_not_f' and
value1 == '0'):
        return 'd_f'
    else:
        return 'U'
```

XOR Gate:

```
def XOR(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == 'U' or value2 == 'U' or value1 == 'Z' or value2 == 'Z'):
        return 'U'
    elif(value1 == value2):
        return '0'
    elif(value1 == 'd_f' and value2 == '0') or (value2 == 'd_f' and value1 ==
'0'):
        return 'd_f'
    elif(value1 == 'd_f' and value2 == '1') or (value2 == 'd_f' and value1 ==
'1'):
        return 'd_not_f'
    elif(value1 == 'd_not_f' and value2 == '0') or (value2 == 'd_not_f' and
value1 == '0'):
        return 'd_not_f'
    elif(value1 == 'd_not_f' and value2 == '1') or (value2 == 'd_not_f' and
value1 == '1'):
        return 'd_f'
    else:
        return '1'
```

XNOR Gate:

```
def XNOR(id1, id2, node_values):
    value1 = node_values[id1]
    value2 = node_values[id2]
    if(value1 == 'U' or value2 == 'U' or value1 == 'Z' or value2 == 'Z'):
        return 'U'
    elif(value1 == value2):
        return '1'
    elif(value1 == 'd_f' and value2 == '0') or (value2 == 'd_f' and value1 ==
'0'):
        return 'd_not_f'
    elif(value1 == 'd_f' and value2 == '1') or (value2 == 'd_f' and value1 ==
'1'):
        return 'd_f'
    elif(value1 == 'd_not_f' and value2 == '0') or (value2 == 'd_not_f' and
value1 == '0'):
        return 'd_f'
    elif(value1 == 'd_not_f' and value2 == '1') or (value2 == 'd_not_f' and
value1 == '1'):
        return 'd_not_f'
    else:
        return '0'
```

NOT:

```
def NOT(id, node_values):
    value = node_values[id]
    if(value == 'U' or value == 'Z'):
        return 'U'
    elif(value == '0'):
        return '1'
    elif(value == '1'):
        return '0'
    elif(value == 'd_f'):
        return 'd_not_f'
    elif(value == 'd_not_f'):
        return 'd_f'
```

Buffer:

```
def BUFF(id, node_values):
    value = node_values[id]
    if(value == 'U' or value == 'Z'):
        return 'U'
    elif(value == '0'):
        return '0'
    elif(value == '1'):
        return '1'
    elif(value == 'd_f'):
        return 'd_f'
    elif(value == 'd_not_f'):
        return 'd_not_f'
```

حال بعد از اینکه توابع مربوطه برای خواندن فایل های ورودی صدا زده شدند با استفاده از تابع get_node_values دو ورودی که توسط کاربر وارد شد ، مدار منطقی موردنظر را طی میکنند و خروجی ها را هم در شاخه های fanout و هم outputs بدست می آورند و در دیکشنری node_values که تعریف کردیم میریزد و این دیکشنری در فایل خروجی چاپ میشود.

```
def get_node_values(input_data, circuit_data):
    node_values = {}
    for item in input_data:
        node_values[item[0]] = item[1]

    for item in circuit_data:
        if(item[0] not in node_values):
            line = find_node_data(item[0], circuit_data)
            gate_output = find_gate_output(line, node_values)
            node_values[item[0]] = gate_output
    return node_values
```

❖ **نکته قابل توجه** : برای مقادیر Z درصورتی که به ورودی گیتی داده شوند اگر گیت ما ورودی کنترلی دیگری نداشته باشد جواب بصورت U بازگردانده می شود.

```
with open('output/output_'+ circuit_name, 'w') as file:
    file.write('node' + '\t\t' + 'value\n')
    file.write('------\t------\n')
    for item in list(output.items()):
        file.write('  ' + item[0] + '   =>   ' + item[1] + '\n')
```

برای اینکه خروجی چاپ شده مربوط به مدار به صورت یک فایل txt در اختیار کاربر قرار بگیرد قطعه کد زیر را اضافه نمودیم که خروجی را به همان صورت خواسته شده در متن پروژه به صورت فایل txt در پوشه Output ایجاد کند.

۳. تست عملکرد مدار و نمایش ورودی/خروجی های مربوطه

برای اینکه از صحت عملکرد شبیه ساز مدار ترکیبی که طراحی کرده ایم مطمئن شویم سه تا از فایلهای isc که در صورت پروژه خواسته شده بود به نام های c1 ، c5 و c17 استفاده کردیم و دیدیم که شبیه ساز به درستی کار میکند.

۳.۱. تست شبیه ساز با فایل c1 :

فایل c1.isc :

```
1  *c17 iscas example (to test conversion program only)
2  *-----------------------------------------------
3  *
4  *
5  *  total number of lines in the netlist ..............    17
6  *  simplistically reduced equivalent fault set size =     22
7  *        lines from primary input  gates .......     5
8  *        lines from primary output gates .......     2
9  *        lines from interior gate outputs ......     4
10 *        lines from **     3 ** fanout stems ...     6
11 *
12 *        avg_fanin  =  2.00,     max_fanin  =  2
13 *        avg_fanout =  2.00,     max_fanout =  2
14 *
15 *
16 *
17 *
18 *
19     1     1gat inpt    1   0      >sal
20     2     2gat inpt    1   0      >sal
21     3     3gat nand    0   2      >sal
22      1     2
```

ورودی برای فایل c1 :

input	value
1	1
2	0

خروجی تولید شده توسط شبیه ساز برای c1 :

node	value
1	1
2	0
3	1

۳.۲. تست شبیه ساز با فایل c5 :

فایل c5.isc :

```
*c17 iscas example (to test conversion program only)
*-----------------------------------------------
*
*
*  total number of lines in the netlist ..............    17
*  simplistically reduced equivalent fault set size =     22
*        lines from primary input  gates .......     5
*        lines from primary output gates .......     2
*        lines from interior gate outputs ......     4
*        lines from **     3 ** fanout stems ...     6
*
*        avg_fanin  =  2.00,     max_fanin  =  2
*        avg_fanout =  2.00,     max_fanout =  2
*
*
*
*
*
    1     1gat inpt    1   0      >sal
    2     2gat inpt    1   0      >sal
    3     3gat nand    1   2      >sal
     1     2
    4     4gat inpt    1   0      >sal
    5     5gat and     0   2      >sal
     3     4
```

ورودی برای فایل c5 :

input	value
1	1
2	0
4	0

خروجی تولید شده توسط شبیه ساز برای c5 :

node	value
1	1
2	0
3	Z
4	1
5	U

۳.۳. تست شبیه ساز با فایل c17 :

فایل c17.isc :

```
*c17 iscas example (to test conversion program only)
*-----------------------------------------------------
*
*
*  total number of lines in the netlist ..............     17
*  simplistically reduced equivalent fault set size =      22
*        lines from primary input  gates .......      5
*        lines from primary output gates .......      2
*        lines from interior gate outputs ......      4
*        lines from **     3 ** fanout stems ...      6
*
*        avg_fanin  =  2.00,     max_fanin  =  2
*        avg_fanout =  2.00,     max_fanout =  2
*
*
*
*
*
    1     1gat inpt    1   0       >sal
    2     2gat inpt    1   0       >sal
    3     3gat inpt    2   0 >sa0 >sal
    8     8fan from     3gat       >sal
    9     9fan from     3gat       >sal
    6     6gat inpt    1   0       >sal
    7     7gat inpt    1   0       >sal
   10    10gat nand    1   2       >sal
     1     8
   11    11gat nand    2   2 >sa0 >sal
     9     6
   14    14fan from    11gat       >sal
   15    15fan from    11gat       >sal
   16    16gat nand    2   2 >sa0 >sal
     2    14
   20    20fan from    16gat       >sal
   21    21fan from    16gat       >sal
   19    19gat nand    1   2       >sal
    15     7
   22    22gat nand    0   2 >sa0 >sal
    10    20
   23    23gat nand    0   2 >sa0 >sal
    21    19
```

ورودی برای فایل c17 :

input	value
1	1
2	0
3	1
6	0
7	0

خروجی تولید شده توسط شبیه ساز برای c17 :

node	value
1	1
2	0
3	1
6	0
7	0
8	1
9	1
10	0
11	1
14	1
15	1
16	1
20	1
21	1
19	1
22	1
23	0