Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра «Вычислительная техника»

Отчет по лабораторной работе №3

по дисциплине:

«Методы тестирования СВТ»

Выполнил:

ст. гр. ВТм-112

В.А. Лихачев

Принял:

С.С. Гладьо

Владимир 2012

Задание: Программно реализовать параллельный метод моделирования неисправностей, составить контролирующий и диагностирующий тесты для комбинационной схемы.

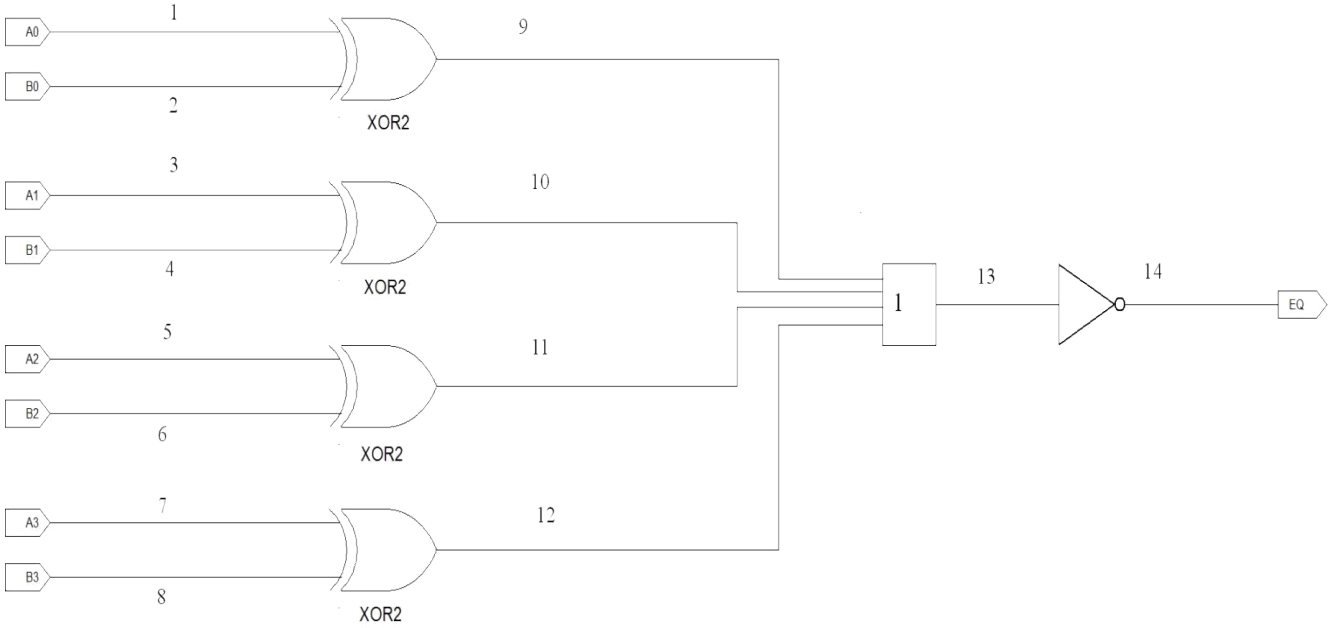
Параллельный метод тестирования неисправностей:

Приведен не весь код, а лишь основная часть, отвечающая за моделирование

```
import itertools

def perform_modelling(input_components, output_line, all_components):
    # it creates all possible combinations of 0,1 of length(input_components)
    # if len(input_components) = 2, then input_values = [(0, 0), (0, 1), (1, 0),
(1, 1)]
    input_values = [i for i in itertools.product([0, 1], repeat =
len(input_components))]
    output_values = [] # input set, output value
    #find all lines in scheme
    all_lines = []
    for component in all_components:
        for line in component.get_input_lines(): # many input lines
            all_lines.append(line)
        all_lines.append(component.get_output_line()) # but only one output lines
    for component in input_components:
        all_lines.append(component.get_output_line())
    all_lines = list(set(all_lines))
    # to return from function
    detectable_failures = [] # input, line.__id and line.get_value()
    # we will iterate over all possible failures
    # for every input first iteration must be run without failures to calculate
output
    # because of this first element in all_failures is None
    all_failures = [None]
    all_failures.extend([[line, 0] for line in all_lines])
    all_failures.extend([[line, 1] for line in all_lines])
    for input in input_values:
        current_valid_output_value = None # first iteration sets this variable
        # find all detectable failures for every input
        for failure in all_failures:
            for c in all_components:
                c.clear() # to avoid possible problems from previous iterations
            for l in all_lines:
                l.clear()
            # set failed line for this iteration
            if failure != None:
                line = failure[0]
                value = failure[1]
                line.set_output(value)
            cur_processing = input_components[:] # copy list
            i = 0
            for component in cur_processing: # set input of scheme
                component.set_input(0, input[i])
                component.out()
                i += 1
```

```
            while len(cur_processing) > 0: # for every iteration
                # lines where value can be calculated now
                lines = [component.get_output_line() for component in
cur_processing if component.ready()]
                components_to_remove = []
                i = 0
                for component in cur_processing:
                    if component.ready():
                        components_to_remove.append(i)
                        component.out() # from now get_value() on output_lines can
be called
                    i += 1
                i = 0
                for k in components_to_remove:
                    cur_processing.pop(k - i)
                    i += 1

                remained_lines = [] # if the line is splitted
                for line in lines:
                    component = line.get_output_component() # to which component
this line is attached as input
                    if component is None: # this line is connected to other lines
                        remained_lines.extend(line.get_output_lines())
                    else:
                        cur_processing.append(component) # for every input line
component will be added
                        # because of this we remove duplicates from cur_processing
by using set()
                        # we can call line.get_value() because lines[] contain
only lines with ready to use values (not None)

component.set_input(line.get_output_component_input_port(), line.get_value())
                if len(remained_lines) == 0: # it can be output line
                    if len(lines) >= 1 and (output_line in lines):
                        if current_valid_output_value == None: # first iteration
for this input (without failures)
                            output_values.append([input, output_line.get_value()])
                            current_valid_output_value = output_line.get_value()
                        else:
                            #if current_valid_output_value !=
output_line.get_value():
                            # input set, line number, line value, scheme out,
scheme out valid for this input set
                            detectable_failures.append([input, str(failure[0]),
str(failure[1]), output_line.get_value(), current_valid_output_value])
                        # break
                for line in remained_lines:
                    component = line.get_output_component()
                    cur_processing.append(component)
                    component.set_input(line.get_output_component_input_port(),
line.get_value())
                cur_processing = list(set(cur_processing)) # remove duplicates
    return [output_values, detectable_failures]
```

Схема компаратора:



Часть таблицы неисправностей:

Входные наборы	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Выход схемы	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
1/0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
1/1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2/0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
2/1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3/0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
3/1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4/0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
4/1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5/0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0
5/1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1

Диагностирующий тест(определяет место возникновения неисправности):

В таблице показаны определяемые на входных наборах неисправности.

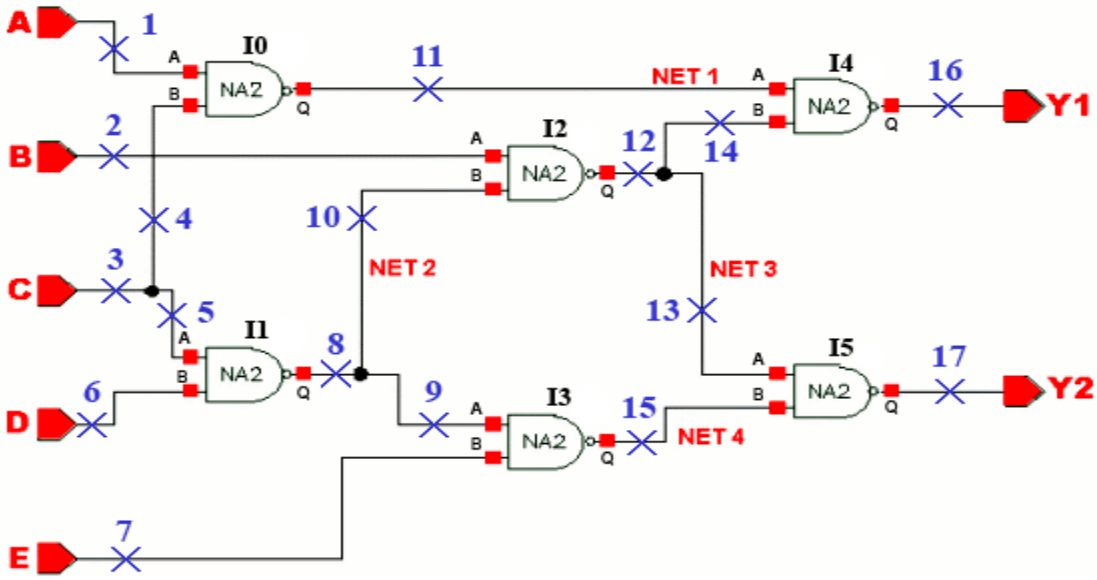
Входной набор 8	Входной набор 12
5/0	1/1
6/1	2/1
11/0	3/1
13/0	4/1
14/1	5/0
	6/0
	7/1
	8/1
	9/1
	10/1
	11/1
	12/1
	13/1
	14/0

Пересечение двух столбцов дает единственную неисправность 5/0, то есть если подать последовательно на схему сначала 8 входной набор, а затем 12 входной набор, то можно будет говорить о наличии или отсутствии неисправности вида 5/0.

Контролирующий тест(определяет наличие неисправности в схеме):

На входном наборе 00001000 можно определить неисправности, указанные в первом столбце предыдущей таблицы.

Схема ISCAS C17



Часть таблицы неисправностей для выхода y1:

Входные наборы	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Выход схемы	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
1/0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
1/1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
2/0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2/1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0
3/0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
3/1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
4/0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
4/1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
5/0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
5/1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0

Диагностирующий тест:

Входной набор 14	Входной набор 20
1/1	1/0
3/0	3/0
5/0	4/0
6/0	11/1
8/1	16/0
10/1	
11/0	
12/0	
14/0	
16/1	

Пересечение двух наборов неисправностей дает 3/0.

Контролирующий тест:

Любая из колонок пре дыдущей таблицы является контролирущим тестом для неисправностей, указанных в соответствующих столбцах.

Вывод: В ходе данной лабораторной работы был программно реализован параллельный метод моделирования неисправностей.