Computer Assignment #2 - Fault Simulation Gate Classes (Preliminary)

Amirmahdi Joudi 2021-04-16

In this assignment, a simple fault simulation is designed. This is line oriented fault collapsing and is used to reduce the number of wires' faults. These faults are recognized and then simulated to find wrong results.

Wire:

Wire class as shown below, has variables for name, value, faultValue, faultyWire, wire_idetifier, and inp_for_nth_time.
Variable inp_for_nth_time and those which are pointers are used to branch fanout that we talk more about it in Gate and FS classes.

```
#ifndef __WIRE__H
#define __WIRE__H "wire.hpp"

#include <string>

class Wire
{
public:
    std::string name;
    char* value;
    char* faultValue;
    bool* faultyWire;
    int wire_identifier;
    int inp_for_nth_time;
    Wire();
    bool operator==(const Wire* w);
private:
    static int number_of_wires;
};

#endif
```

Gate:

All gates are inherited from class Gate. They have variables for 1 or 2 inputs and an output. There is also gate_identifier. To do calculations; they make events on outputs. They check wires, faults, and if there is a fault, faultValue is being considered as wire actual value. Method get_faults() return a vector of strings that contains wires' name, id and faults if exist. Mrthod branch_out() makes new wires for inputs which are used as a gate inputs for more than one time. They have new id, but are connected to previous wires.

```
class Gate
{
public:
    int gate_identifier;
    Gate();
    Gate(Wire* in1, Wire* in2, Wire* out);
    Gate(Wire* in1, Wire* out);
    virtual void eval() = 0;
    virtual std::vector<std::string> get_faults() = 0;
    virtual std::vector<Wire*> branch_fanout();
    virtual std::vector<Wire*> get_inputs();
    Wire* get_output();
protected:
    Wire* input1;
    Wire* input2;
    Wire* output;
    static int number_of_gates;
};

class Not : public Gate
{
public:
    Not();
    Not(Wire* in1, Wire* out);
    void eval();
    std::vector<std::string> get_faults();
    std::vector<Wire*> branch_fanout();
    std::vector<Wire*> get_inputs();
};

class Nand : public Gate
{
public:
    Nand();
    Nand(Wire* in1, Wire* in2, Wire* out);
    void eval();
    std::vector<std::string> get_faults();
};

class Nor : public Gate
{
public:
    Nor();
    Nor(Wire* in1, Wire* in2, Wire* out);
    void eval();
    std::vector<std::string> get_faults();
};
```

Compiler:

The following rules are checked:

- each line except endmodule has to end with ‘;’.
- module has to have name and it can not be same as keywords
- The necessary ‘(‘, ‘)’, ‘#’ and ‘,’ are checked and extra spaces are removed
- all variables have to be defined and then be assigned as inputs and outputs
- wires, inputs and outputs can not have same names
- endmodule is necessary
- delay format can be #(to1,to0) or #(delay) but it can not be ignored as delay 0
- gates first argument is output and it can have any number of inputs

Finally an object of Logic is created which is actually our combinational circuit.

```
class Compiler
{
public:
    Compiler(std::string sv_file_address);
    ~Compiler();
    void compile();
    Logic* get_logic();
private:
    Logic* logic;
    std::vector<std::string> sv_lines;
    std::vector<Wire*> io;
    std::vector<Wire*> inputs;
    std::vector<Wire*> outputs;
    std::vector<Wire*> wires;
    void assign_variables(std::string data, int line_num);
    void assign_iow_variables(std::string data, int line_num, std::string io);
    void form_gate(std::string data, std::string gate, int line_num);
    void add_io_to_logic();
    void check_sem_comm_syntax(std::string data, int line_num);
    void check_module_line_syntax(std::vector<std::string> tokens, int line_num);
    void check_io_line_syntax(std::string line, int line_num);
    void check_io_correctness(std::string variable, int line_num, std::string io);
    void check_gate_line_syntax(std::string data, int line_num);
    Wire* find_output(std::vector<std::string> output);
    std::vector<Wire*> find_inputs(std::vector<std::string> _inputs);
    std::vector<int> find_delay_values(std::string data);
    std::vector<std::string> find_gate_io(std::string data);
};
```

Logic:

A vector of gate pointers are stored here. Then it sort gates suitabl for evaluations using a recursive functions and a class of FC is created which does fault collapsing operations.

```
class Logic
{
public:
    Logic();
    ~Logic();
    void calculate();
    void L_NAND(Wire* output, std::vector<Wire*> inputs);
    void L_NOR(Wire* output, std::vector<Wire*> inputs);
    void L_NOT(Wire* output, std::vector<Wire*> inputs);
    void set_inputs(std::vector<Wire*> inputs);
    void set_outputs(std::vector<Wire*> outputs);
    void set_intermediates(std::vector<Wire*> intermediates);
    FC* get_fc();
private:
    std::vector<Wire*> logic_inputs;
    std::vector<Wire*> logic_outputs;
    std::vector<Wire*> logic_intermediates;
    std::vector<Wire*> all_wires;
    std::vector<Gate*> gates;
    Gate* find_gate_with_output(Wire* output);
    void sort_gates();
    void rec_sort(Wire* gate_output, std::vector<Gate*> &sorted_gates);
    bool did_reach_to_first_level(Wire* input);
};
```

FC:

It does branch_fanout for gates and outputs. Also it finds faults and return it as a vector of string. It makes an object from class FS that does the simulation.

```
class FC
{
public:
    FC(std::vector<Gate*> _gates, std::vector<Wire*> _all_wires, std::vector<Wire*> _outputs, std::vector<Wire*> _inputs);
    std::vector<std::string> fault_collapse();
    FS* get_fs();
private:
    std::vector<Gate*> gates;
    std::vector<Wire*> all_wires;
    std::vector<Wire*> outputs;
    std::vector<Wire*> inputs;
    std::vector<std::string> l_faults;
    void branch_fanout();
};
```

FS:

The simulation of true and with faults values are done here. Then a vector of strings which true and with faults values for each case is created.

```
class FS
{
public:
    FS(std::vector<Gate*> _gates, std::vector<Wire*> _all_wires, std::vector<Wire*> _outputs, std::vector<Wire*> inputs, std::vector<std::string> _faults);
    std::vector<std::string> calculate(std::string input_values);
private:
    std::vector<Gate*> gates;
    std::vector<Wire*> all_wires;
    std::vector<Wire*> outputs;
    std::vector<Wire*> inputs;
    std::vector<std::string> faults;
    std::vector<std::string> eval_true_values();
    std::vector<std::string> eval_gates();
    void set_inputs_events(std::string line);
};
```

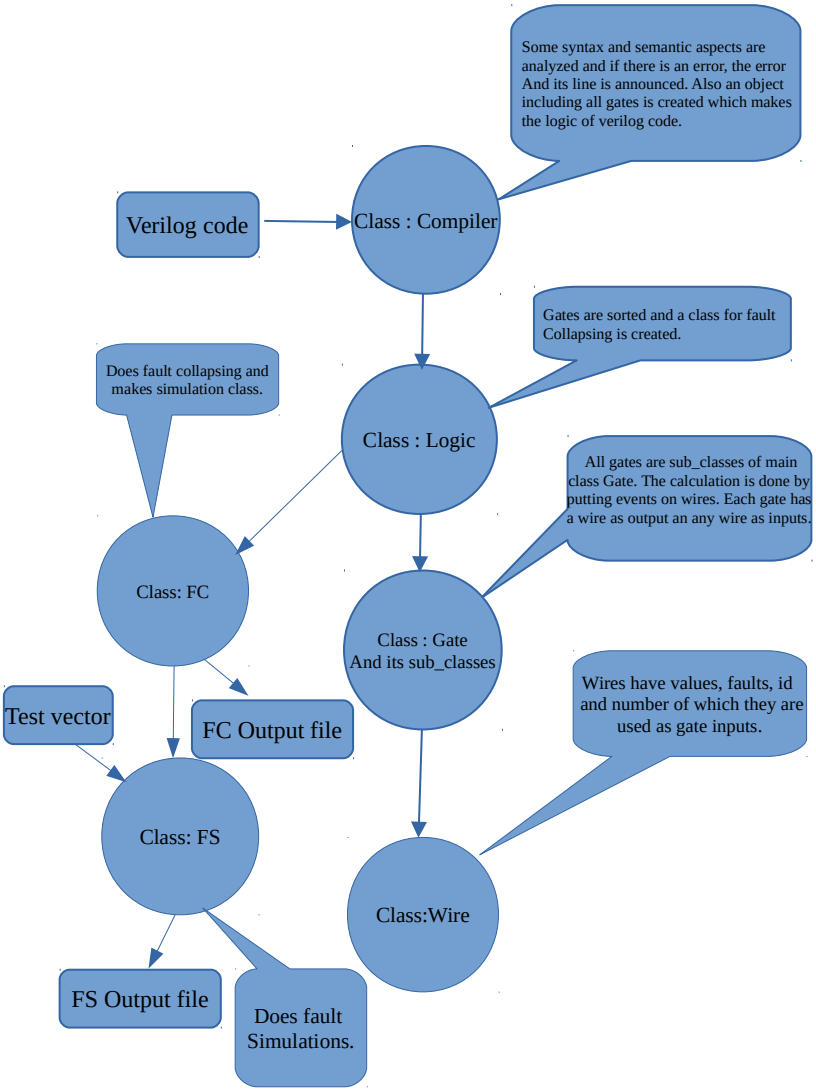
CA2:

```cpp
int Wire::number_of_wires = ZERO;
int Gate::number_of_gates = ZERO;

using namespace std;

int main(int argc, char* argv[])
{
    if(argc >= TWO)
    {
        Compiler compiler(argv[ONE]);
        try
        {
            compiler.compile();
            if(argc >= THREE)
            {
                vector<string> faults;
                vector<string> sim;
                ofstream fc_output_file("fc_output_file.txt");
                ofstream fs_output_file("fs_output_file.txt");
                Logic* logic = compiler.get_logic();
                logic->calculate();
                FC* fc = logic-> get_fc();
                faults = fc-> fault_collapse();
                for(auto line:faults)
                    fc_output_file << line << endl;
                fc_output_file.close();
                FS* fs = fc-> get_fs();
                sim = fs-> calculate(argv[TWO]);
                for(auto line:sim)
                    fs_output_file << line << endl;
                fs_output_file.close();
            }
        }
        catch (invalid_argument& ex)
        {
            cerr << ex.what();
        }
    }
}
```

```
a     4   1   000
aa    9   1   001
b     5   1   010
bb    10  1   011
s     12  1   100
s     6   0   101
s     6   1   110
sbar  8   1   111
y     7   0
y     7   1
```




```
Output values without fault: 0
a    4  1   1
aa   9  1   0
b    5  1   0
bb   10 1   0
s    12 1   0
s    6  0   0
s    6  1   0
sbar 8  1   0
y    7  0   0
y    7  1   1
Output values without fault: 0
a    4  1   0
aa   9  1   0
b    5  1   1
bb   10 1   0
s    12 1   0
s    6  0   0
s    6  1   0
sbar 8  1   0
y    7  0   0
y    7  1   1
Output values without fault: 0
a    4  1   1
aa   9  1   0
b    5  1   0
bb   10 1   0
s    12 1   1
s    6  0   0
s    6  1   1
sbar 8  1   0
y    7  0   0
y    7  1   1
```

```
Output values without fault: 1
a    4  1   1
aa   9  1   1
b    5  1   1
bb   10 1   0
s    12 1   1
s    6  0   0
s    6  1   1
sbar 8  1   1
y    7  0   0
y    7  1   1
Output values without fault: 1
a    4  1   1
aa   9  1   0
b    5  1   1
bb   10 1   1
s    12 1   0
s    6  0   1
s    6  1   0
sbar 8  1   1
y    7  0   0
y    7  1   1
Output values without fault: 0
a    4  1   0
aa   9  1   0
b    5  1   1
bb   10 1   0
s    12 1   0
s    6  0   1
s    6  1   0
sbar 8  1   1
y    7  0   0
y    7  1   1
```

```
Output values without fault: 1
a    4  1   1
aa   9  1   0
b    5  1   1
bb   10 1   1
s    12 1   1
s    6  0   1
s    6  1   1
sbar 8  1   1
y    7  0   0
y    7  1   1
Output values without fault: 1
a    4  1   1
aa   9  1   1
b    5  1   1
bb   10 1   0
s    12 1   1
s    6  0   1
s    6  1   1
sbar 8  1   1
y    7  0   0
y    7  1   1
```