Homework 2 - C++ Programming for Digital Systems' Applications

Taghizadeh,
810198373

I. Introduction

This homework is about event-driven gate-level circuit simulation using a timing wheel data structure. We are going to simulate a Verilog module in C++ which has timing and gates' delays are included in the simulation, so the simulation is almost same as Verilog simulation which has timing and gates' delays. In the end we will see the outputs of *"C423.v"* Verilog module in 5 different scenarios which considers timing and gate's delays and it is unordered means that there is no need to evaluate the gates in proper order and gates can be evaluated in an arbitrary order.

II. Homework Parts Report

A. *Converting Verilog module to C++ code*

First in a loop, each line of the Verilog module is read by the *getline()* command

```
81
82  void convert_verilog_to_cpp(string verilog_input_file)
83  {
84      ifstream verilog_code(verilog_input_file);
85
86      string current_line_of_verilog_code;
87
88      while(getline(verilog_code, current_line_of_verilog_code))
89      {
```

Fig.1 Reading each line of Verilog module

Base on the line read, we realise that there is a wire in that line of the Verilog module or not gate or and gate or etc.

```
90          if(current_line_of_verilog_code.substr(0, 5) == "input")
91          {
```

Fig.2 Realising input wires of Verilog module

```
153         if(current_line_of_verilog_code.substr(0, 4) == "wire")
154         {
```

Fig.3 Realising wires of Verilog module

```
194         if(current_line_of_verilog_code.substr(0, 3) == "not")
195         {
```

Fig.4 Realising not gates of Verilog module

```
202         if(current_line_of_verilog_code.substr(0, 4) == "nand")
203         {
```

Fig.5 Realising nand gates of Verilog module

Then to extract wires' names or gates' input output wires, the rest of the current line of the Verilog module is considered by the *substr()* command and split by "," with the *strtok()* command.

```
62  vector<string> extract_gate_wires(string current_line_of_verilog_code)
63  {
64      string gate_io = current_line_of_verilog_code.substr(current_line_of_
65      char gate_inputs_output[gate_io.size() + 1];
66      strcpy(gate_inputs_output, gate_io.c_str());
67
68      vector<string> gate_wires_names;
69
70      char *wire = strtok(gate_inputs_output, ",");
71
72      while(wire != NULL)
73      {
74          string gate_wire = wire;
75          gate_wires_names.push_back(gate_wire);
76
77          wire = strtok(NULL, ",");
78      }
79      return gate_wires_names;
80  }
```

Fig.6 Extracting gates' wires

Now the gate in the current line of the Verilog module and its inputs and output wires is known. So we take an instance of the gate and initialize its inputs and output wires based on the wires' names that we extracted from the current line of the Verilog module.

```
198             NOT* new_not_gate = new NOT();
199             new_not_gate->ios(&Wires[find(wires_names, gate_wires_names[0
200             gates.push_back(new_not_gate);
201         }
```

Fig.7 Getting instance of not gate

```
202         if(current_line_of_verilog_code.substr(0, 4) == "nand")
203         {
204             vector<string> gate_wires_names = extract_gate_wires(current_
205
206             if(gate_wires_names.size() == 3)
207             {
208                 NAND* new_nand_gate = new NAND(2);
209                 new_nand_gate->ios(&Wires[find(wires_names, gate_wires_na
210                 gates.push_back(new_nand_gate);
211             }
212             else if(gate_wires_names.size() == 4)
213             {
214                 NAND* new_nand_gate = new NAND(3);
215                 new_nand_gate->ios(&Wires[find(wires_names, gate_wires_na
216                 gates.push_back(new_nand_gate);
217             }
218             else
219             {
220                 NAND* new_nand_gate = new NAND(4);
221                 new_nand_gate->ios(&Wires[find(wires_names, gate_wires_na
222                 gates.push_back(new_nand_gate);
223             }
224         }
```

Fig.8 Getting instance of nand gate

This is repeated till we reach the line of the Verilog module that contains ";" at the end of, which means the end of the command.

```
ile(current_line_of_verilog_code[current_line_of_verilog_code.size() - 2] != ';')
```

Fig.9

In the end there will be some gates and wires extracted from the Verilog module, which are stored in a vector of gates and wires.

```
vector<string> wires_names;
vector<string> input_wires_names;
vector<string> output_wires_names;
vector<Wire> Wires;

vector<Gate*> gates;
```

Fig.10 Storing extracted gates and wires in vectors

B. *Taking timed inputs from a text file*

First each line of a text file (where all timed data inputs are stored) is read by the *getline()* command and stored in a vector of string (where each string of this vector represents a timed data input).

```
// Getting timed inputs from timed_inputs_scenario5.txt file

ifstream timed_inputs_file("timed_inputs_scenario5.txt");
string current_line_of_timed_inputs_file;
vector<string> timed_inputs;
while(getline(timed_inputs_file, current_line_of_timed_inputs_file))
    timed_inputs.push_back(current_line_of_timed_inputs_file);
```

Fig.11 Storing timed inputs in a vector of string

The variable *"current_time"* is defined and set to 0. This variable represents current time which is increased whenever time is advanced by the *advanced_time_one_step()* method of the *time_wheel* class.

```
int main ()
{

    // Getting timed inputs from timed_inputs_scenario5.txt file

    ifstream timed_inputs_file("timed_inputs_scenario5.txt");
    string current_line_of_timed_inputs_file;
    vector<string> timed_inputs;
    while(getline(timed_inputs_file, current_line_of_timed_inputs_file))
        timed_inputs.push_back(current_line_of_timed_inputs_file);

    convert_verilog_to_cpp("c432.v");

    time_wheel Time_Wheel = time_wheel();

    int current_time = 0;

    do
    {
        for(int i = 0; i < timed_inputs.size(); i++)
            if(stoi(timed_inputs[i].substr(1, timed_inputs[i].find(" "))) == curr
                initialize_inputs(timed_inputs[i].substr(timed_inputs[i].find("
        Time_Wheel.advance_time_one_step();
        current_time++;
    }while(!Time_Wheel.is_activity_list_empty());


    // printing the values of output wires

    for(int i = 0; i < output_wires_names.size(); i++)
        cout << output_wires_names[i] << " = " << Wires[find(wires_names, output

    return 1;
}
```

Fig.12 Representation current time with *"currren_time"* variable

At each time we search all timed data inputs by a loop, if a timed data input has the time equal to the current time, so this time is the time that inputs must be initialized according to that timed data input.

```
for(int i = 0; i < timed_inputs.size(); i++)
    if(stoi(timed_inputs[i].substr(1, timed_inputs[i].find(" "))) == current_time
        initialize_inputs(timed_inputs[i].substr(timed_inputs[i].find(" ") + 1, t
```

Fig.13 Initialize inputs which must be initialized at current time

To initialize inputs we wrote a function which iterates through input wires' names, finds their corresponding input wire and set their values based on timed data input (which is a string extracted from a line of the text file where all timed data inputs are stored)

```
void initialize_inputs(string given_input_values)
{
    for(int i = 0; i < input_wires_names.size(); i++)
        Wires[find(wires_names, input_wires_names[i])].set_value(given_input_valu
}
```

Fig.14 Initialize input wires

C. *Event driven simulator for gate-level circuits implementation*

First a class called *"time_wheel"* is defined, which has an array of time called *"time_array"*, which each time has a vector of gates which must be evaluated in that time. There is a variable *"current_time"* which shows the current time and is increased whenever time is advanced by the *"advance_time_one_step"* method of the *"time_wheel"* class.

```
vector<Gate*> gates;

const int max_delay_of_gates = 6;

class time_wheel {
    int current_time = 0;
    vector<Gate*> time_array[max_delay_of_gates];
    public:
        time_wheel() {} // constructor
        void advance_time_one_step() {
            for(int i = 0; i < time_array[current_time].size(); i++)
                time_array[current_time][i]->evl();
            time_array[current_time].clear();
            for(int i = 0; i < gates.size(); i++)
                if(gates[i]->is_gate_ready_to_evl())
                    if(!is_gate_in_activity_list(gates[i]))
                        time_array[(current_time + gates[i]->get_gate_delay()) %
            current_time = (current_time + 1) % 6;
        };
        bool is_gate_in_activity_list(Gate* gate) {
            for(int i = 0; i < 6; i++)
                for(int j = 0; j < time_array[i].size(); j++)
                    if(time_array[i][j] == gate)
                        return true;
            return false;
        }
        bool is_activity_list_empty() {
            for(int i = 0; i < 6; i++)
                if(time_array[i].size() > 0)
                    return false;
            return true;
        };
};
```

Fig.15 time wheel class attributes

Whenever time is advanced, at first all gates which are in the current time of the time array (time_array[current_time]) must be evaluated in a loop. Then time_array[current_time] must be cleared because all gates which must be evaluated in this time (current time) were evaluated before (previous discussed loop in this paragraph) and no gates is remain.

```
vector<Gate*> gates;

const int max_delay_of_gates = 6;

class time_wheel {
    int current_time = 0;
    vector<Gate*> time_array[max_delay_of_gates];
    public:
        time_wheel() {} // constructor
        void advance_time_one_step() {
            for(int i = 0; i < time_array[current_time].size(); i++)
                time_array[current_time][i]->evl();
            time_array[current_time].clear();
            for(int i = 0; i < gates.size(); i++)
                if(gates[i]->is_gate_ready_to_evl())
                    if(!is_gate_in_activity_list(gates[i]))
                        time_array[(current_time + gates[i]->get_gate_delay()) %
            current_time = (current_time + 1) % 6;
        };
        bool is_gate_in_activity_list(Gate* gate) {
            for(int i = 0; i < 6; i++)
                for(int j = 0; j < time_array[i].size(); j++)
                    if(time_array[i][j] == gate)
                        return true;
            return false;
        }
        bool is_activity_list_empty() {
            for(int i = 0; i < 6; i++)
                if(time_array[i].size() > 0)
                    return false;
            return true;
        };
};
```

Fig.16

Then by evaluation of the gates which must be evaluated in the current time, some gates will be ready to evaluate because their inputs were initialized by the gates which evaluated in this time (current time). So we iterate through all gates and by calling *"is_gate_ready_to_evl()"* method of the *"time_wheel"* class, we find the gates, which are ready to evaluate. In the end for the gates, which are ready to evaluate, we add these gates to the time array. We must add each gate to the time which is gate delay times greater than the current time, so we push back the gates to time_array[current_time _ gate_delay % max_gate_delay]. It must be mentioned that "%max_gate_delay" is added to make the time array like a time wheel, because there is no need to have a large time array, instead we can add gates relative to the current time, so the time array with the max gate delay size would be enough.

```
for(int i = 0; i < gates.size(); i++)
    if(gates[i]->is_gate_ready_to_evl())
        if(!is_gate_in_activity_list(gates[i]))
            time_array[(current_time + gates[i]->get_gate_delay()) % max_delay_of
```

Fig.17 Adding gates ready to evaluate to time wheel

In the end, in the main function we take an instance of "time_wheel" class. Then in a do – while loop, time is advanced till there are no gates in the time array and all gates have been evaluated.

```
int main ()
{

    // Getting timed inputs from timed_inputs_scenario5.txt file

    ifstream timed_inputs_file("timed_inputs_scenario5.txt");
    string current_line_of_timed_inputs_file;
    vector<string> timed_inputs;
    while(getline(timed_inputs_file, current_line_of_timed_inputs_file))
        timed_inputs.push_back(current_line_of_timed_inputs_file);

    convert_verilog_to_cpp("c432.v");

    time_wheel Time_Wheel = time_wheel();

    int current_time = 0;

    do
    {
        for(int i = 0; i < timed_inputs.size(); i++)
            if(stoi(timed_inputs[i].substr(1, timed_inputs[i].find(" "))) == curr
                initialize_inputs(timed_inputs[i].substr(timed_inputs[i].find(" "
        Time_Wheel.advance_time_one_step();
        current_time++;
    }while(!Time_Wheel.is_activity_list_empty());


    // printing the values of output wires

    for(int i = 0; i < output_wires_names.size(); i++)
        cout << output_wires_names[i] << " = " << Wires[find(wires_names, output_

    return 1;
}
```

Fig.18 Instanciation of time wheel class in main

D. Implementation verification

Scenario 1 timed inputs

```
#0 11111111111111111111111111111111
```

Fig.19 Scenario 1 timed inputs

Scenario 1 simulation outputs

```
N223 = 0
N329 = 0
N370 = 0
N421 = 0
N430 = 1
N431 = 1
N432 = 1
```

Fig.20 Scenario 1 simulation outputs

Scenario 2 timed inputs

```
1 #0 00000000000000000000000000000000
```

Fig.21 Scenario 2 timed inputs

Scenario 2 simulation outputs

```
N223 = 0
N329 = 0
N370 = 0
N421 = 0
N430 = 0
N431 = 0
N432 = 0
```

Fig.22 Scenario 2 simulation outputs

Scenario 3 timed inputs

```
1 #0 10101010101010101010101111111111
```

Fig.23 Scenario 3 timed inputs

Scenario 3 simulation outputs

```
N223 = 0
N329 = 0
N370 = 0
N421 = 1
N430 = 0
N431 = 0
N432 = 1
```

Fig.24 Scenario 3 simulation outputs

Scenario 4 timed inputs

```
1 #0 10101010101010101010101000000011111
```

Fig.25 Scenario 4 timed inputs

Scenario 4 simulation outputs

```
N223 = 0
N329 = 0
N370 = 0
N421 = 1
N430 = 0
N431 = 0
N432 = 0
```

Fig.26 Scenario 4 simulation outputs

Scenario 5 timed inputs

```
1 #0 10101010101011111111111111111111111
```

Fig.27 Scenario 5 timed inputs

Scenario 5 simulation outputs

```
N223 = 1
N329 = 0
N370 = 0
N421 = 1
N430 = 1
N431 = 0
N432 = 1
```

Fig.28 Scenario 5 simulation outputs