ENGINEERING TRIPOS PART IIA

Project GF2 **SOFTWARE**

LOGIC SIMULATOR

First Interim Report

Name: Yi Chen Hock, Michael Stevens, Cindy Wu
College: Queens', Robinson, John's

Contents

1 Introduction

2 General Approach

The general approach to the project is to split it into four main sections. The first section involves defining a precise specification for the logic description language, which the client will use to define their logic circuit. The aim of this is to create a simple and coherent language using EBNF notation, which is readable to both the user as well as the computer. In addition, the specification will identify the semantic constraints which apply to the language as well as consider how any error conditions will be handled. The details of this section are outlined in this report.

The second section involves developing the individual modules for the bulk of the logic simulator. The scanner and parse modules will directly use the logic description language as well as the specification of error handling as defined in the first section. The approach for this is to create a detailed specification in the first section to allow for easier and efficient development of the code. The user interface will be designed to be straight forward as well as intuitive to use. Throughout this section, unit tests will be implemented to ensure that each module works as expected, even when given unexpected inputs.

The third section is to integrate the various modules together and ensure that the program works as one. The unit tests for each module developed in the previous section will greatly improve the speed and efficiency of integration as they will ensure that any bugs found will be confined to the integration of the modules rather than the modules themselves. Therefore, it is vital that well written and complete unit tests are developed to allow for smooth integration. In addition, clear and easy to read documentation will be produced for the client.

Finally, the last section is involved with implementing any modifications to the system which the client proposes. These modifications will not be known until after the logic simulator is developed and hence, the code will be structured with plenty of modularisation. This will allow system modifications to be implemented more efficiently and with less bugs.

2.1 Teamwork Planning

The Gantt chart in the Appendix (Figure 1) shows how the tasks for this project have been split up and allocated to each team member. Note that the allocated person(s) for each task will not necessarily be the only person working on that task since the workload for each task may not be predictable at this stage of the project. The Gantt chart also highlights the deadlines given by the client. It's also to be noted that the aim is to complete the first iteration of the code within the first two weeks of the project, to allow sufficient time for integration and testing of the system. Throughout this project, git is used for collaborative coding and version control. The repository can be found at https://github.com/yichenhock/GF2.

3 EBNF for Syntax

- Define
- User-defined name
- Specify parameters
- Define device connections
- Specify output signal to monitor

```
letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
         "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
         "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

device_name = letter, {[letter|digit]};

clock_name = "clk", {[digit]};

switch_name = "sw", {[digit]};

port_name = "IN", {[digit]};

devices_type = "AND" | "ANDS" | "OR" | "ORS" |"NOR" | "NORS" | "XOR" |
    "XORS" | "NAND" | "NANDS" |"DTYPE" | "DTYPES";

arrow = "=>" | "->";

definition = "is" | "are";

possession = "has", "have";

bracket_open = "(";

bracket_close = ")";

semicolon = ";";

comma = ",";

dot = ".";

order_factor = "K" | "M" | "G" | "T";

switch_level = "HIGH" | "LOW" | "1" | "0";

device_definition = {device_name,[comma]}, definition, devices_type, ";";

switch_definition = {switch_name,[comma]}, definition, ("SWITCHES" | "SWITCH"), ";";

clock_definition = {clock_name,[comma]}, definition, ("CLOCK" |"CLOCKS"), ";";

switch_initialisation = {switch_name,[comma]}, definition, switch_level, ";";

inputs = {device_name,[comma] | switch_name,[comma]}, possession, {digit}, "inputs", ";";

clock_frequency = clock_name, "frequency", {digit}, order_factor, ";";

connection_definition = (device_name | switch_name), ["BAR"], ("is connected to" | "to" | "connects to"
{(device_name | switch_name), dot, port_name}, ";";
```

```
device_block = "device", bracket_open, {device_definition | switch_definition | clock_definition}, brack

initialise_block = "initialise", bracket_open, {switch_initialisation | clock_initialisation}, bracket_c

connections_block = "connections", bracket_open, {(device_name | switch_name), bracket_open,
{connection_definition}, bracket_close}, bracket_close, ";";

monitors_block = "monitors", bracket_open, {device_name | switch_name}, bracket_close, ";";
```

"BAR" following a `device_name` or `switch_name` denotes the inverse of the output. We do not include comment syntax in EBNF as the scanner removes comments from input file. The syntax for comments will be: `# ;`

where "#" denotes the start of the comment section, and ";" denotes the end of the comment section. There will be no line comments as line breaks and spaces hold no significance.

3.1 Example A — Simple Circuit

```
devices(
    a, b are NANDs;
    a, b have 2 inputs;
    sw1, sw2 are SWITCHES;
    clk1 is CLOCK;
);

initialise(
    sw1, sw2 are LOW;
    clk1 frequency 10M;
);

connections(
    a(
        sw1 is connected to a.IN1;
        b is connected to a.IN2;
    );

    b(
        a is connected to b.IN1;
        sw2 is connected to b.IN2;
    );
);

monitors(
    a, b;
);

```

5 Appendix

Figure 1: Gantt Chart