PROJECT GF2 **SOFTWARE**

LOGIC SIMULATOR

First Interim Report

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

Contents

1 Introduction

The aim of this project is to develop a logic circuit simulator in Python. As a team of three, we will work together on the project under a real-life simulation of a professional software development environment. This will involve the five major phases of the software engineering life cycle: specification, design, implementation, testing and maintenance. This report will cover the general approach of the project including teamwork planning; EBNF for syntax; identification of all possible semantic errors; description of error handling; and two example definition files together with diagrams showing the logic circuits they represent.

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 3) 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

The logic description language will be specified using EBNF notation. The language was designed with ease of readability in mind, so English keywords and puncuation have been used such that the meaning is clear

to a human reader as well as a machine. There are four main sections to the description file:

1. `device`: Defines the device and device type by user-defined names and along with the number of input ports for each device and defines any switches or clocks.

2. `initialise`: Sets the state of the switches (which will be the inputs to the logic circuit) and the frequency of the clock. Error handling will be implemented to ensure that

3. `connections`: Defines the connections between switches, devices and input ports.

4. `monitors`: Defines the output signals which will be monitored.

The following is a formal description using EBNF of the logic description language we will be using in this project:

```
circuit_description = device_block, initialise_block, connections_block, [monitors_block];

device_block = "device", bracket_open, {gate_definition | dtype_definition | switch_definition |
    clock_definition}, bracket_close, semicolon;

initialise_block = "initialise", bracket_open, {switch_initialisation | clock_initialisation},
    bracket_close, semicolon;

connections_block = "connections", bracket_open, {device_name, bracket_open,
{connection_definition}, bracket_close, semicolon}, bracket_close, semicolon;

monitors_block = "monitors", bracket_open, {device_name | dtype_output_name | switch_name |
    clock_name}, bracket_close, semicolon;

gate_definition = device_name, {[comma, device_name]}, definition, gate_type, semicolon;

dtype_definition = device_name, {[comma, device_name]}, definition, ("DTYPE" | "DTYPES"),
    semicolon;

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

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

gate_inputs = device_name, {[comma, device_name]}, possession, {digit}, ("input" | "inputs"),
    semicolon;

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

clock_period = clock_name, ("cycle length" | "cycle"), {digit}, semicolon;

connection_definition = (device_name | dtype_output_name | switch_name | clock_name), ("is
    connected to" | "to" | "connects to" | arrow),
{(gate_input_name | dtype_input_name)}, semicolon;

gate_input_name = device_name, dot, "I", {digit};

dtype_input_name = device_name, dot, dtype_inputs;

dtype_output_name = device_name, dot, dtype_outputs;
```

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

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

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

gate_type = "AND" | "ANDS" | "OR" | "ORS" |"NOR" | "NORS" | "XOR" |
            "XORS" | "NAND" | "NANDS";

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

dtype_inputs = "DATA" | "CLK" | "SET" | "CLEAR";

dtype_outputs = "Q" | "QBAR";

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" ;

definition = "is" | "are";

possession = "has", "have";

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

bracket_open = "(";

bracket_close = ")";

semicolon = ";";

comma = ",";

dot = ".";
```

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.

NOTES:

1. AND etc can have up to 16 inputs, XOR only 2 inputs

2. Optional Monitor block (user can define it later)

3. Talk about short vs long form "Is connected to", example B is short A is long

4. clock cycle length is number of simulation cycles for clock to change

5. "BAR" now only and always applies to dtype (change above paragraph), mention QBAR is opposite of Q

3.1 Example A — Simple Circuit
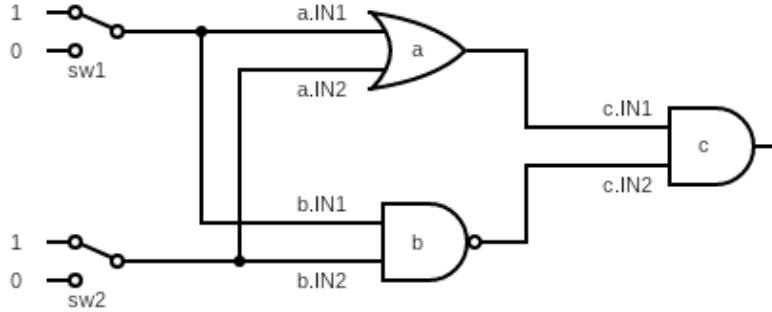
Figure 1: XOR Gate

```
1   devices(                              17          );
2       a is OR;                          18
3       b is NAND;                        19       b(
4       c is AND;                         20           sw1 is connected to b.I1;
5       a, b, c have 2 inputs;            21           sw2 is connected to b.I2;
6       sw1, sw2 are SWITCHES;            22       );
7   );                                    23
8                                         24       c(
9   initialise(                           25           a is connected to c.I1;
10      sw1, sw2 are HIGH;                26           b is connected to c.I2;
11  );                                    27       );
12                                        28   );
13  connections(                          29
14      a(                                30   monitors(
15          sw1 is connected to a.I1;     31       c;
16          sw2 is connected to a.I2;     32   );
```

3.2 Example B — Complex Circuit
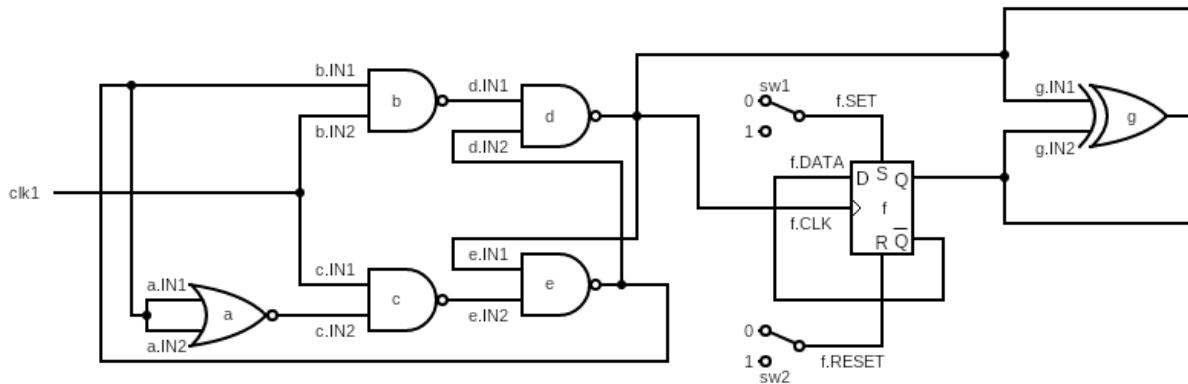
Figure 2: 2 Bit Counter with Outputs XOR

5

```
 1    devices(
 2        a is NOR;
 3        b, c, d, e are NANDS;
 4        f is DTYPE;
 5        g is XOR;
 6        a, b, c, d, e, g have 2 inputs;
 7        sw1, sw2 are SWITCHES;
 8        clk1 is CLOCK;
 9    );
10
11    initialise(
12        sw1, sw2 are LOW;
13        clk1 cycle length 1000;
14    );
15
16    connections(
17        a(
18            e to a.I1;
19            e to a.I2;
20        );
21
22        b(
23            e to b.I1;
24            clk1 to b.I2;
25        );
26
27        c(
28            clk1 to c.I1;
29            a to c.I2;
30        );
31
32        d(
33            b to d.I1;
34            e to d.I2;
35        );
36
37        e(
38            d to e.I1;
39            c to e.I2;
40        );
41
42        f(
43            f.QBAR to f.DATA;
44            d to f.CLK;
45            sw1 to f.SET;
46            sw2 to f.CLEAR;
47        );
48
49        g(
50            d to g.I1;
51            f.Q to g.I2;
52        );
53    );
54
55    monitors(
56        d, f.Q, g;
57    );
```

4 Errors

4.1 Error Handling

4.2 Semantic Error Identification

5 Appendix

GF2 Software

Part IIA project

Project Start: May 13, 2022

Display Week: 1

Yi Chen Hock, Michael Stevens, Cindy Wu

TASK	ASSIGNED TO	PROGRESS	START	END
Preliminary Exercises	All	100%	13/5/22	15/5/23
Teamwork Planning	Michael, Yi Chen	100%	14/5/22	15/5/22
Introduction	Yi Chen	100%	16/5/22	21/5/22
General Approach	Michael	100%	16/5/22	21/5/22
EBNF for Syntax	Yi Chen	100%	17/5/22	21/5/22
Error Identification and Handling	Cindy	100%	17/5/22	21/5/22
Example Definition Files	Michael	100%	19/5/22	21/5/22
First Interim Report (Deadline)	All			22/5/22
Names Module	Michael	0%	21/5/22	23/5/22
Scanner Module	Michael	0%	23/5/22	26/5/22
Parser Module	Cindy	0%	21/5/22	26/5/22
GUI Module	Yi Chen	0%	21/5/22	30/5/22
Unit Tests	Cindy	0%	21/5/22	27/5/22
README	Michael	0%	28/5/22	2/6/22
Integration	All	0%	28/5/22	2/6/22
Integration Tests	All	0%	28/5/22	2/6/22
Second Interim Report (Deadline)	Individual			3/6/22
Maintenance	All	0%	3/6/22	9/6/22
Final Report (Deadline)	Individual			10/6/22

| May 9, 2022 | May 16, 2022 | May 23, 2022 | May 30, 2022 | Jun 6, 2022 |

Figure 3: Gantt Chart