ENGINEERING TRIPOS PART IIA

PROJECT GF2   **SOFTWARE**

LOGIC SIMULATOR

First Interim Report

Name: Yi Chen Hock, Michael Stevens, Cindy Wu
College: Queens', Robinson, St. 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. In particular, the development of unit tests for each module will likely be developed by a different team member than the module's author, however, a single member will be responsible for the overall completion of these unit tests. 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. `devices`: 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 output and input ports of the devices. This is subdivided into sections which contain all inputs to a given device.

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. Note the top-down structure starts from the overall file format and finishes at the definition of allowed letters, symbols and numbers:

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

device_block = "devices", 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" | 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" | "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 = ".";
```

The gate and D-type device names are defined by the user, whilst their input and output ports are defined as per the logic description language. The clock is defined by a cycle length, which is the number of simulation cycles for which the output state remains constant. The number of inputs and outputs for different gate types are not defined in the EBNF, instead being classed as semantic errors. The switch output is set by switch_level.

Since devices can be named with any combination of letters and numbers, subject to the condition that the start must be a letter, we have chosen lowercase letters only for 'letter', reserving capital letters for fixed keywords and gate types. This avoids common naming clashes.

In the choosing of the tradeoff between flexibility and complexity, we have given three options for defining connections (arrow, 'is connected to' and 'to') which can be ideally chosen for depending on the client's preference, but kept consistent in usage throughout any one file. The same applies for switch levels (1/0 or 'high/low') as well as when defining the clock cycle length ('cycle length' or 'cycle').

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

Figure 1: XOR Gate
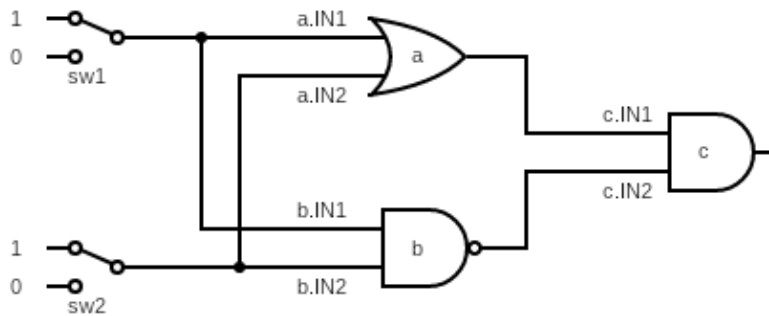
```
1    devices(                              17          );
2        a is OR;                          18
3        b is NAND;                        19      b(
4        c is AND;                         20          sw1 is connected to b.I1;
5        a, b, c have 2 inputs;            21          sw2 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  );
```
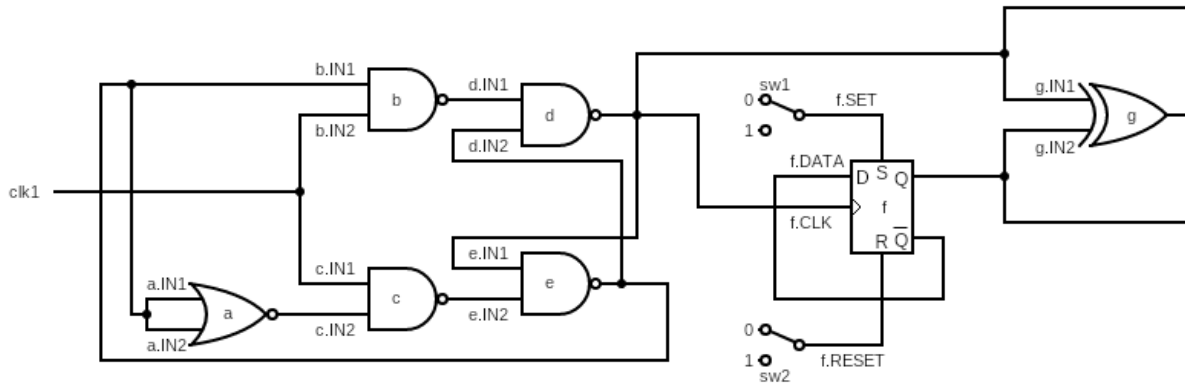
3.2   Example B — Complex Circuit

Figure 2: 2 Bit Counter with Outputs XOR

```
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

All errors encountered when scanning and parsing the circuit definition file as well as during simulation will be reported to the user, indicating the nature of the error. When an error occurs during parsing, a cursor shows exactly at which point in the circuit definition file failure occurs, using a function in the scanner class. A counter of number of errors will be included in an error function to record how many errors have occured. Each line in the specification should be ended by a semicolon, so the parser skips to past the next semicolon and resumes reading from there when an error is encountered. Error messages specific to the error type (listed by error code) will be printed, and use of the names module allows the variable associated with the error to be included in the error message.

Syntax errors involve errors which do not conform to the format specified by the EBNF grammar. Below we identify instead semantic errors, which conform to the EBNF grammar but not circuit logic rules.

4.2 Runtime errors and warnings

Circuits may conform to syntax and semantic errors and still generate logic circuits that may break upon simulation. For example:

1. A 'NOT' gate (NAND or NOR gate with all input ports connected) with its output fed back to its input acts as a clock with an infinitely fast cycle frequency, as no propagation delay is specified for the simulation of these gates.

2. If both SET and CLEAR of D-type are set to 1 at the same time during simulation.

These errors should be flagged with a runtime error warning pointing to the device that has caused the issue.

4.3 Semantic Error Identification

The table below summarises types of semantic error, detection and reporting.

Table 1: List of Semantic Errors

Semantic constraint	Example of error	Reporting error message
Clock/switch must have 1 output, 0 inputs	devices(clk1 has 1 input;);	Incorrect number of inputs specified for clock clk1. 1 specified, but allowed number is 0.
AND, NAND, and NOR must have 1 output, 1-16 inputs	devices(a is NOR; a has 17 inputs;);	Incorrect number of inputs specified for NOR gate a. 17 specified, but allowed range is 1-16.
XOR must have 1 output, 2 inputs	devices(a is XOR; a has 1 input;);	Incorrect number of inputs specified for XOR gate a. 1 specified, but allowed number is 2.
DTYPE gates must have 4 inputs, one each of CLK, DATA, SET and CLEAR	devices(d is DTYPE;); connections(a to d.CLK; d.QBAR to d.DATA; sw1 to d.SET;);	Incomplete set of inputs to DTYPE. Missing CLEAR.
DTYPE SET and CLEAR inputs cannot both be logic HIGH at the same time	initialise(sw1, sw2 are HIGH;); connections(sw1 to d.SET; sw2 to d.CLEAR;);	SET and CLEAR ports cannot both set to logic high.
Device names must not overlap with reserved phrases in syntax	devices(to is NOR;);	Illegal device name specified. Please observe the list of words to be avoided.
The number of connection inputs defined cannot be different to the number of inputs specified	devices(a has 1 input;); connections(sw1 is connected to a.I1; sw2 is connected to a.I2;);	Different number of inputs defined in connection section to number of inputs listed in devices section.
The same device name cannot be used for two different devices	devices(a is XOR; a is AND;);	Device names must be unique.
Each input channel can admit only a connection to one output	connections(a to b.IN1; c to b.IN1;);	Input channel b.IN1 can only connect to one output.
Devices that have not been defined in the devices section cannot be used in other sections	initialise(sw1 is HIGH;);	Please define the device sw1 before initialisation.
Switch initial value or clock cycle length is not defined in initialisation	devices(sw1, sw2 are SWITCHES;);	Please define switches sw1, sw2 initial values.
Devices that have been defined in the devices section are not connected to or used later	devices(clk1 is CLOCK;);	clk1 defined but not used.

Reserved words list (cannot define these as variable names):
"to", "connected", "is", "are", "have", "has", "inputs", "input", "connections", "initialise", "device", "monitors", "cycle", "length".

Note that for all devices other than dtype, by definition, the number of outputs is 1. One output can be included in more than one connection, so it is not a semantic error if the same switch is connected to more than one input port.
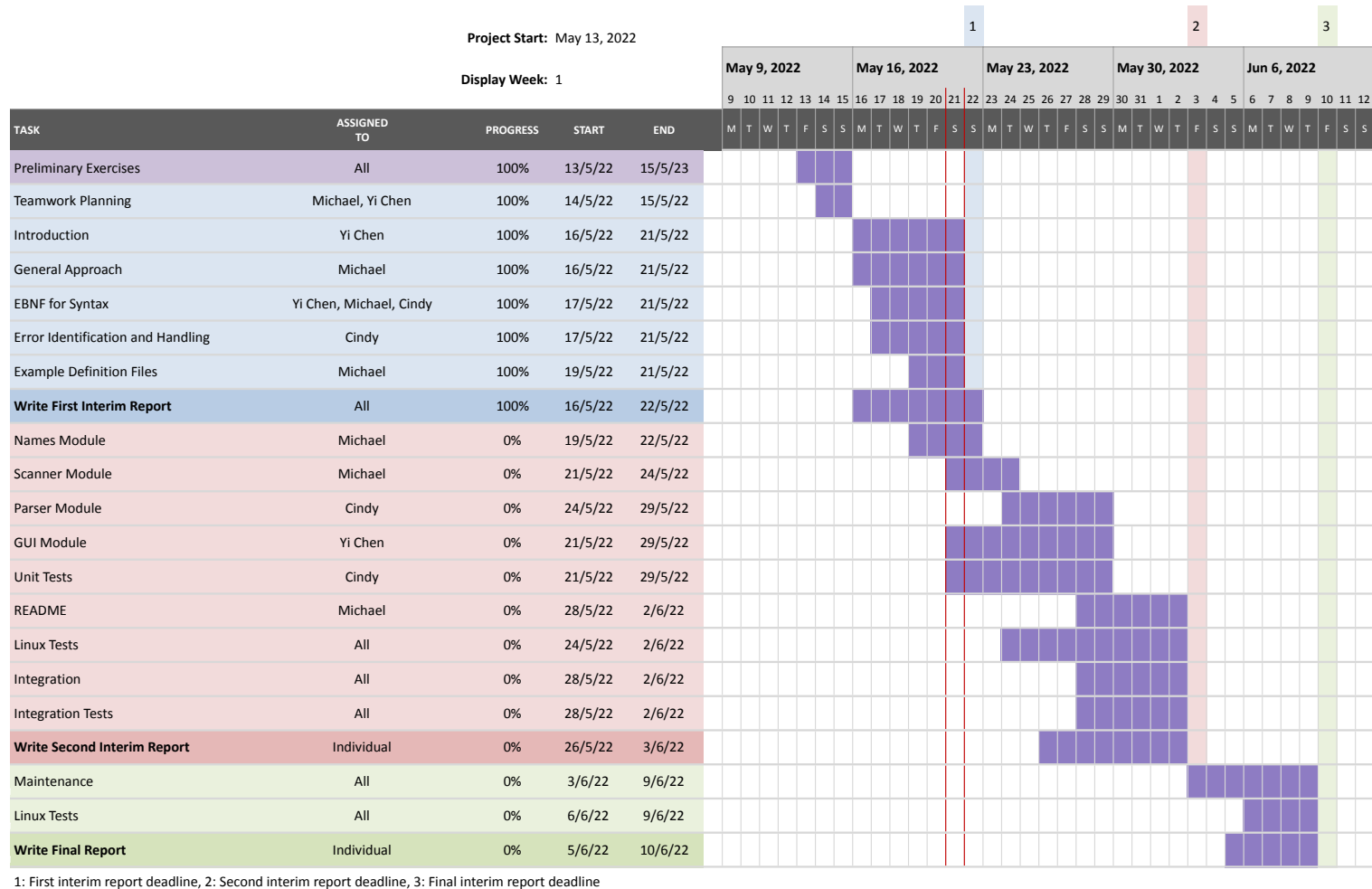
5 Appendix

Project Start: May 13, 2022

Display Week: 1

							1			2		3

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, Michael, Cindy	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
Write First Interim Report	All	100%	16/5/22	22/5/22
Names Module	Michael	0%	19/5/22	22/5/22
Scanner Module	Michael	0%	21/5/22	24/5/22
Parser Module	Cindy	0%	24/5/22	29/5/22
GUI Module	Yi Chen	0%	21/5/22	29/5/22
Unit Tests	Cindy	0%	21/5/22	29/5/22
README	Michael	0%	28/5/22	2/6/22
Linux Tests	All	0%	24/5/22	2/6/22
Integration	All	0%	28/5/22	2/6/22
Integration Tests	All	0%	28/5/22	2/6/22
Write Second Interim Report	Individual	0%	26/5/22	3/6/22
Maintenance	All	0%	3/6/22	9/6/22
Linux Tests	All	0%	6/6/22	9/6/22
Write Final Report	Individual	0%	5/6/22	10/6/22

1: First interim report deadline, 2: Second interim report deadline, 3: Final interim report deadline

Figure 3: Gantt Chart