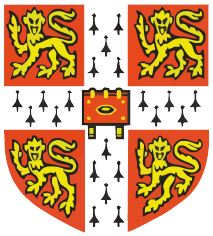

UNIVERSITY OF CAMBRIDGE

DEPARTMENT OF ENGINEERING

ENGINEERING TRIPOS PART IIA

DPO **GF2 PROJECT**

SOFTWARE
First Interim Report

Group 0

Arsalan Harris (aah36)
Ieva Cernyte (ic322)
Mark Mathews (mm2121)

---

1 Introduction

The aim of this project is to develop a logic simulation program in Python. This involves completing key modules such as the scanner, the parser and the GUI. The key tasks at the start of the project were to assign tasks to team members and plan rough time scales, specify syntax using the EBNF notation, identify all possible errors that could occur and work out how to handle and report errors.

2 Teamwork Planning

The distribution of work and the rough timeline is presented in Table 1 and Figure 1.

	Design and implementation	**Testing**
Ieva	Names, Parser	GUI, Scanner
Arsalan	Scanner, Parser	GUI, Names
Mark	GUI	Parser

Table 1: Work distribution among the team members

The general scheme of work means that for efficient progress, initially most of the work will be done separately to try and successfully complete the modules. Upon completion of the scanner, Arsalan and Ieva will collaborate to finish the parser and then all the modules will be integrated and tested by all three members of the team. Testing of the module will be done by a different team member than the author. For progress tracking and task distribution Trello app will be used. Github is going to be used for version control.

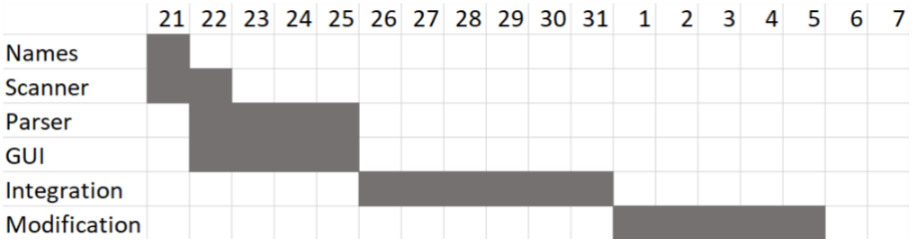

Figure 1: Rough timeline of the project

3 Syntax

The full EBNF grammar is given in section 6.1 of the Appendix. The specification file comprises two compulsory sections, DEVICES and CONNECTIONS, optionally followed by MONITOR. Each section contains a header string, a list of comma-separated items and ends in a semicolon.

3.1 Device list

In the device list, each device is assigned a name, which must begin with a letter, contain only letters, numbers and underscores and have a minimum length of 2 characters. DTYPE and XOR devices are assumed to have a fixed number of inputs and outputs. AND, NAND, NOR and OR gates must be specified with the number (between 1 and 16) of inputs corresponding to each. Switches must be specified with their initial state and clocks must be defined along with the number of cycles it takes for their output to change.

3.2 Connection list

Connections are defined between two devices using the characters '->'. Inputs for DTYPE are DATA, CLK, SET, CLEAR and inputs for all other devices are I1, I2, ... I16. Outputs for DTYPE are Q and QBAR and outputs for all others are given by the character O (all other devices are assumed to be single-output).

3.3 Monitor list and comments

The monitor list contains a list of device outputs to be monitored during the simulation. Single-line comments in the definition file can be included following the characters // and multi-line comments can be included between the characters /* and */. These will be eliminated by the scanner prior to error checking.

4 Errors

4.1 Semantic errors list

1. Device with such name already exists;
2. Specified device requires [n] connected inputs, [i] connected right now;
3. DTYPE requires 4 connected inputs, [i] connected right now;
4. XOR requires 2 connected inputs, [i] connected right now;
5. Connection on the left must be an output;
6. Connection on the right must be an input;
7. Input/output with such name doesn't exist;
8. Device with such name doesn't exist.

4.2 Semantic error detection

Semantic errors relating to device, input and output names will be detected by checking name tables. Whether a connection is an output or an input will also be checked using name tables and the information saved about the specific device. The number of connected inputs will be tracked as the network is built and after parsing of *Connections* section is done, the number of connected inputs will be compared with device specifications.

4.3 Error handling and reporting

Once an error in the definition file has been detected, an error message will be printed out and a recovery will be attempted by skipping until the next comma or semicolon. Building of a network will be stopped after the first error but parsing will continue in order to detect and report all errors. An example error message is given below. After parsing of the file is finished, the total number of errors will be displayed as well.

```
Line 3: CLICK CLK1 5,
        ^
Error 12: Not valid device type. Valid DEVICES: CLOCK, SWITCH, NAND, AND,
OR, NOR, DTYPE, XOR.
```

5 Example Definition Files and Logic Circuits

Section 6.2 in the Appendix shows two examples of a definition file with the corresponding logic circuits in Figure 2 and Figure 3. This follows the format established earlier.

In the first example, considering the first line beneath the "DEVICES:" heading, the first device type considered is a SWITCH, the name of the switch is SW1 and it is in its 0 state and each of these are separated by spaces. Considering the first line beneath the "CONNECTIONS:" heading, the output of SW1 is a 0 and this goes as the input I1 to the AND gate A1. Considering the lines beneath the "MONITOR:" heading, the two outputs monitored are for AND gates A1 and A2.

The second example definition file is slightly more complicated, but works very much in the same way. Considering the third device, the NAND gate G1 has 2 inputs, considering the third connection, the NAND gate G1 has output O which leads to input I1 of G2. Considering the fifth connection, the same NAND gate also leads to the input I1 of NAND gate G3, so each path takes a new line and each connection is accounted for. Under the "MONITOR:" heading, the outputs of G1, G2, G3 and G4 are being monitored. Therefore, this is all simple to understand and laid out to make it easy for the reader to work out.

6 Conclusions

Having specified the syntax and semantics for the logic description file, the next task is to design the individual sub-systems for the final simulation program. When doing this, it is now clear that the most important factors to consider are keeping to time and carefully checking that the program meets the consumer specification. Furthermore, to prepare for the final modification, it is essential to keep the code readable and well structured.

7 Appendix

7.1 EBNF grammar

```
#File Structure:

specfile = "DEVICES:" , device_list , ";" , "CONNECTIONS:" , connection_list ,
";" , [ "MONITOR:" , monitor_list , ";" ] ;

#Device List:

device_list = { (dtype | xor | andnandornor | switch | clock) , "," } , (dtype
| xor | andnandornor | switch | clock);
name = letter , (letter | digit | "_") , { letter | digit | "_"};
dtype = "DTYPE" , name;
xor = "XOR" , name;
andnandornor = ("AND"|"NAND"|"OR"|"NOR") , name, one_to_sixteen, "inputs";
switch = "SWITCH" , name , bool , "state";
clock = "CLOCK" , name , nonzeronumber , "cycles";

#Connection list:

connection_list = { name , "." , output , "->" , name , "." , input , "," } ,
(name , "." , output , "->" , name , "." , input);
output = "O" | "Q" | "QBAR";
input = ( "I" , one_to_sixteen ) | "DATA" | "CLK" | "SET" | "CLEAR";

#Monitor list:

monitor_list = { name , "." , output , "," } , (name , "." , output) ;
bool = "0" | "1";

#Primitive character sequences:

nonzeronumber = nonzerodigit, { digit };
nonzerodigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
digit = "0" | nonzerodigit;
one_to_sixteen = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" |
"11" | "12" | "13" | "14" | "15" | "16";
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" | "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" ;
```

7.2 Example definition files and logic circuits

Example 1

```
DEVICES:
SWITCH SW1 0 state,
SWITCH SW2 0 state,
SWITCH SW3 0 state,
AND A1 2 inputs,
AND A2 2 inputs;
CONNECTIONS:
SW1.O->A1.I1,
SW2.O->A1.I2,
SW3.O->A2.I2,
A1.O->A2.I1;
MONITOR:
A1.O,
A2.O;
```

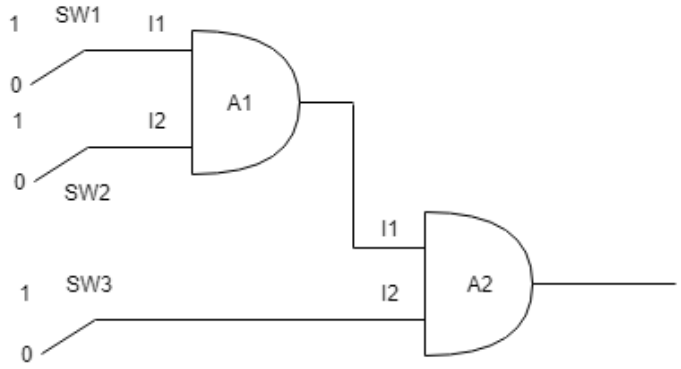

Figure 2: First example of a logic circuit

Example 2

```
DEVICES:
SWITCH SW1 0 state,
SWITCH SW1 0 state,
NAND G1 2 inputs,
NAND G2 2 inputs,
NAND G3 2 inputs,
NAND G4 2 inputs;
CONNECTIONS:
SW1.O->G1.I1,
SW2.O->G2.I2,
G1.O->G2.I1,
G2.O->G1.I2,
G1.O->G3.I1,
G2.O->G4.I2,
G3.O->G4.I1,
G4.O->G3.I2;
MONITOR:
G1.O,
G2.O,
G3.O,
G4.O;
```

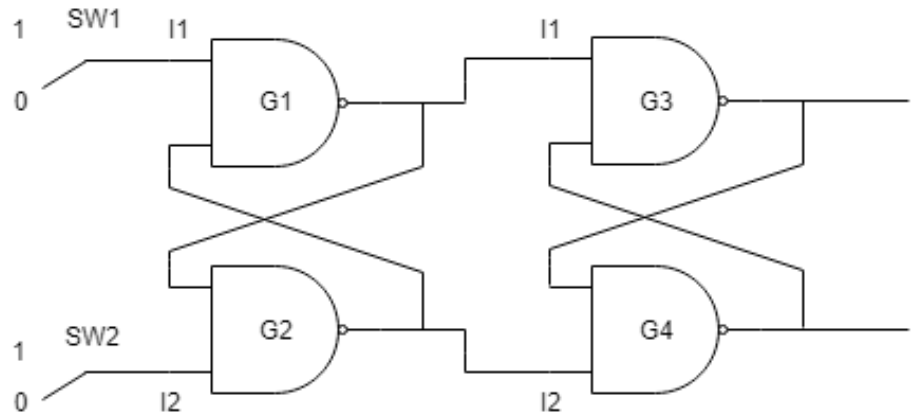

Figure 3: Second example of a logic circuit