ENGINEERING TRIPOS PART IIA

GF2 First Interim Report

Team 11

James Thompson (jhrt2)
Anna Mills (asm87)
Neelay Sant (ns758)

1 Introduction

The aim of this project is to develop a logic simulation program. The software for processing a given circuit has been provided and so the focus is on designing and implementing a language for describing logic circuits, and on providing a suitable Graphical User Interface (GUI). At this stage, the logic description language and error handling protocols have been specified, and the preparatory Python exercises have been completed.

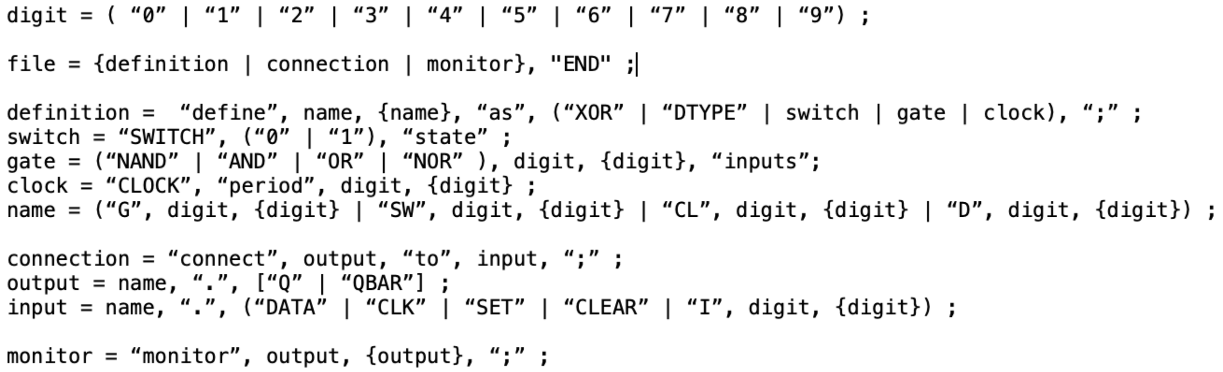
1.1 Teamwork Planning

The team consists of James Thompson, Anna Mills and Neelay Sant (all Emmanuel College). So far, all work has been completed together, however for coding, the tasks have been divided among the team members as specified by the Gantt chart (see Appendix). James and Anna will write the Names and Scanner modules respectively while Neelay prepares the tests for them. Then, Neelay will focus on the GUI while James develops the Parser and Anna prepares pytest files for each. We will individually prepare our second and final reports. The maintenance work will be planned once the memo is released on 4th June.

2 Logic Description Language

2.1 Syntax Specification

The syntax for our Logic Description Language is described in Extended Backus-Naur Form as follows:



```
digit = ( “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”) ;

file = {definition | connection | monitor}, "END" ;

definition =  “define”, name, {name}, “as”, (“XOR” | “DTYPE” | switch | gate | clock), “;” ;
switch = “SWITCH”, (“0” | “1”), “state” ;
gate = (“NAND” | “AND” | “OR” | “NOR” ), digit, {digit}, “inputs”;
clock = “CLOCK”, “period”, digit, {digit} ;
name = (“G”, digit, {digit} | “SW”, digit, {digit} | “CL”, digit, {digit} | “D”, digit, {digit}) ;

connection = “connect”, output, “to”, input, “;” ;
output = name, “.”, [“Q” | “QBAR”] ;
input = name, “.”, (“DATA” | “CLK” | “SET” | “CLEAR” | “I”, digit, {digit}) ;

monitor = “monitor”, output, {output}, “;” ;
```

The syntax is LL(1) and so can be processed by a top-down, single look-ahead parser. Comments in the file are to be enclosed in hashes (eg. #a comment#). All whitespace is ignored. The language is designed such that definitions, connections and monitors can be written in any order, as long as any devices referenced have already been defined.

2.2 Error Handling

A syntax error occurs when the received text from the description file does not conform to the rules laid out in the EBNF specification. When an unexpected character is encountered, a syntax error will be reported with the offending character labelled by ˆ. The Scanner will move to the next semicolon and continue to read the description file, keeping a count of the number of lines containing syntax errors.

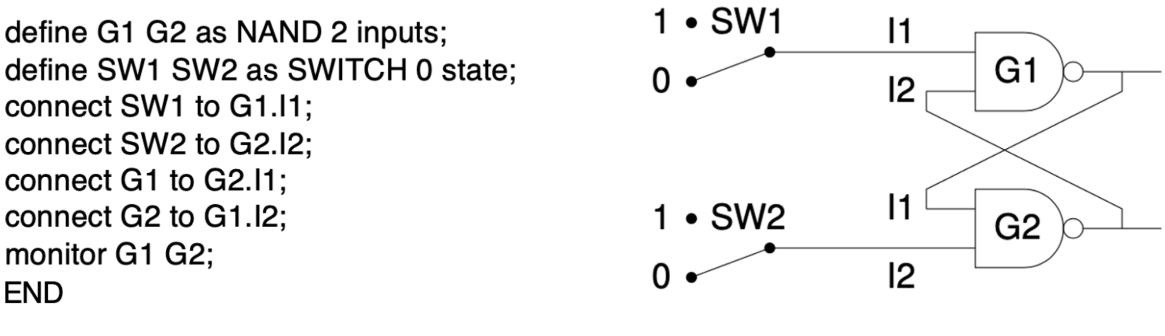
A semantic error occurs when the file is syntactically sound but has ambiguous or nonsensical meaning. Once a semantic error is found that would prevent the circuit from being completed, the program will cease to attempt to construct the network. The following table lists all possible semantic errors and how they will be detected:

Error	**Detection**
Same name used to define multiple different devices.	Use query() in Names class to check name does not already have an ID (i.e. 'None' should be returned)
Inconsistent name chosen in definition (e.g. SW1 for a gate).	Check name matches device type. If not, warn user.
Number of inputs defined for gate not in range 1-16.	'INVALID_QUALIFIER' error returned by make_device() in Devices class.
Defined clock period not interpretable.	Check period does not start with 0.
Multiple connections to same input.	'INPUT_CONNECTED' error returned by make_connection() in Network class. If get_connected_output() returns the output specified by the user (i.e. the user has specified the same connection twice), report to the user and move on. Otherwise report to user and stop.
Invalid port for device type (for connection or monitor).	'PORT_ABSENT' error returned by make_connection() in Network class.
Non-existent device specified (for connection or monitor).	'DEVICE_ABSENT' error returned by make_connection() in Network class.
Monitor already exists.	'MONITOR_PRESENT' error returned by make_monitor() in Monitors class. Report to user and carry on.
Input unconnected.	Call check_network() in Network class at end of parsing. Report error if False returned (i.e. there are floating inputs).
No monitor given.	If monitors_dictionary in Monitors class is empty at end of parsing, alert user and carry on.

Table 1: Semantic Error Handling

Input-to-input and output-to-output errors are syntactically forbidden (and therefore do not need to be specified as semantic errors). The same is true of the "QUALIFIER_PRESENT" error in the Devices class.

2.3 Example Circuit Definition Files



```
define G1 G2 as NAND 2 inputs;
define SW1 SW2 as SWITCH 0 state;
connect SW1 to G1.I1;
connect SW2 to G2.I2;
connect G1 to G2.I1;
connect G2 to G1.I2;
monitor G1 G2;
END
```

Figure 1: Example Definition File 1

```
define CL1 as CLOCK period 1;
define SW1 as SWITCH 1 state;
define SW2 as SWITCH 0 state;
define D1 D2 D3 as DTYPE;
connect SW1 to D1.SET;
connect SW2 to D1.CLEAR;
connect SW1 to D2.SET;
connect SW2 to D2.CLEAR;
connect SW1 to D3.SET;
connect SW2 to D3.CLEAR;
connect CL1 to D1.CLK;
connect D1.Q to D2.CLK;
connect D2.Q to D3.CLK;
connect D1.QBAR to D1.DATA;
connect D2.QBAR to D2.DATA;
connect D3.QBAR to D3.DATA;
monitor D1.Q D2.Q D3.Q;
END
```

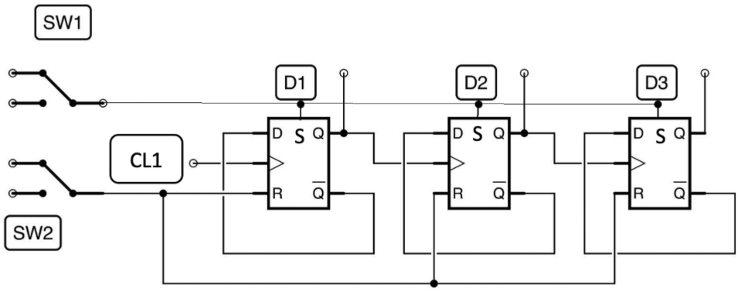


Figure 2: Example Definition File 2

A Gantt Chart

											Second Report Deadline						Final Report Deadline
	24-May	25-May	26-May	27-May	28-May	29-May	30-May	31-May	01-Jun	02-Jun	03-Jun	04-Jun	05-Jun	06-Jun	07-Jun	08-Jun	09-Jun	10-Jun
Write Names Module	🟧	🟧	🟧															
Write Scanner Module	🟪	🟪	🟪															
Write Names Unit Tests	🟦	🟦	🟦															
Write Scannner Unit Tests	🟦	🟦	🟦															
Debug Names				🟧														
Debug Scanner				🟪														
Write Parse Module					🟧	🟧	🟧											
Write GUI Module				🟦	🟦	🟦	🟦											
Write Parse Unit Tests					🟪	🟪	🟪											
Write GUI Unit Tests					🟪	🟪	🟪											
Debug Parse								🟧	🟧									
Debug GUI								🟦	🟦									
Integrate Modules										🟩	🟩							
Maintenance													🟩	🟩	🟩	🟩	🟩	
	🟧 James	🟪 Anna	🟦 Neelay	🟩 All														