Cambridge University Engineering Department
Engineering Tripos Part IIA
PROJECTS: Interim and Final Report Coversheet

IIA Projects

TO BE COMPLETED BY THE STUDENT(S)

Project:	GF2 Software
Title of report:	First Interim Report Group Report / ~~Individual Report~~ (delete as appropriate)

Name(s): (capitals)	crsID(s):	College(s):
JAMES THOMPSON	jhrt2	Emmanuel
ANNA MILLS	asm87	Emmanuel
NEELAY SANT	ns758	Emmanuel

Declaration for: Interim Report 1 / ~~Interim Report 2 / Final Report~~ (delete as appropriate)

~~I~~/we confirm that, except where indicated, the work contained in this report is ~~my/~~our own original work.

Instructions to markers of Part IIA project reports:

Grading scheme

Grade	A*	A	B	C	D	E
Standard	Excellent	Very Good	Good	Acceptable	Minimum acceptable for Honours	Below Honours

Grade the reports by ticking the appropriate guideline assessment box below, and provide feedback against as many of the criteria as are applicable (or add your own). Feedback is particularly important for work graded C-E. Students should be aware that different projects and reports will require different characteristics.

Penalties for lateness: Interim Reports: 3 marks per weekday; Final Reports: 0 marks awarded – late reports not accepted.

Guideline assessment (tick one box)

A*/A	A/B	B/C	C/D	D/E

Marker:		Date:	

Delete (1) or (2) as appropriate (for marking in hard copy – different arrangements apply for feedback on Moodle):

(1) Feedback from the marker is provided on ~~the report itself.~~ moodle.

~~**(2) Feedback from the marker is provided on second page of cover sheet.**~~

	Typical Criteria	Feedback comments
Project Skills, Initiative, Originality	Appreciation of problem, and development of ideas	
	Competence in planning and record-keeping	
	Practical skill, theoretical work, programming	
	Evidence of originality, innovation, wider reading (with full referencing), or additional research	
	Initiative, and level of supervision required	
Report	Overall planning and layout, within set page limit	
	Clarity of introductory overview and conclusions	
	Logical account of work, clarity in discussion of main issues	
	Technical understanding, competence and accuracy	
	Quality of language, readability, full referencing of papers and other sources	
	Clarity of figures, graphs and tables, with captions and full referencing in text	

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.

good to see different team members writing the unit tests ✓

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” ;

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" ;

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

this allows any ordering of blocks (or even no blocks), though in your example files there is a fixed order

```
definition =  “define”, name, {name}, “as”, (“XOR” | “DTYPE” | switch | gate | clock), “;” ;
```

not LL(1) if "as" is allowed as a name

```
switch = “SWITCH”, (“0” | “1”), “state” ;
gate = (“NAND” | “AND” | “OR” | “NOR” ), digit, {digit}, “inputs”;
clock = “CLOCK”, “period”, digit, {digit} ;
name = letter, {letter | digit} ;

connection = “connect”, output, “to”, input, “;” ;
output = name, “.”, [“Q” | “QBAR”] ;
```

not all outputs have a ".", I think you put the [] in the wrong place

```
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#). Whitespace carries no syntactic meaning other than to separate symbols. 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.

an open comment option would also be nice

OK, now I see it was intentional, though not something you felt the need for in either of your examples?

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:

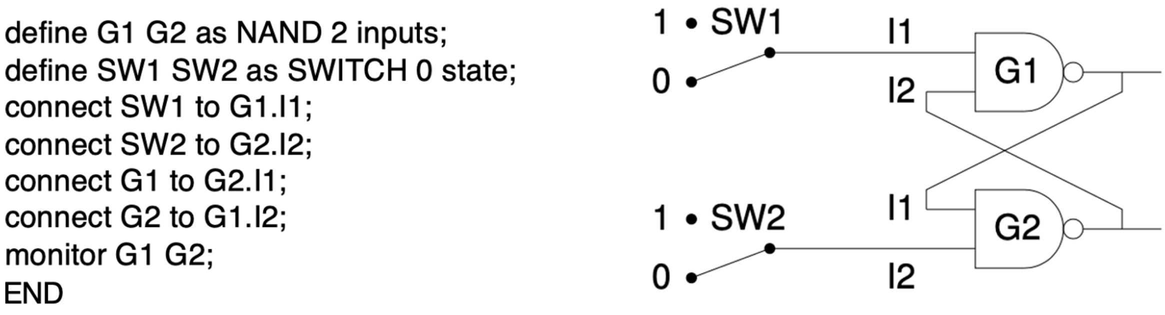
Error	**Detection**
Same name used to define multiple different devices or name is keyword.	Use query() in Names class to check name does not already have an ID (i.e. 'None' should be returned)
Number of inputs defined for gate not in range 1-16.	'INVALID_QUALIFIER' error returned by make_device() in 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

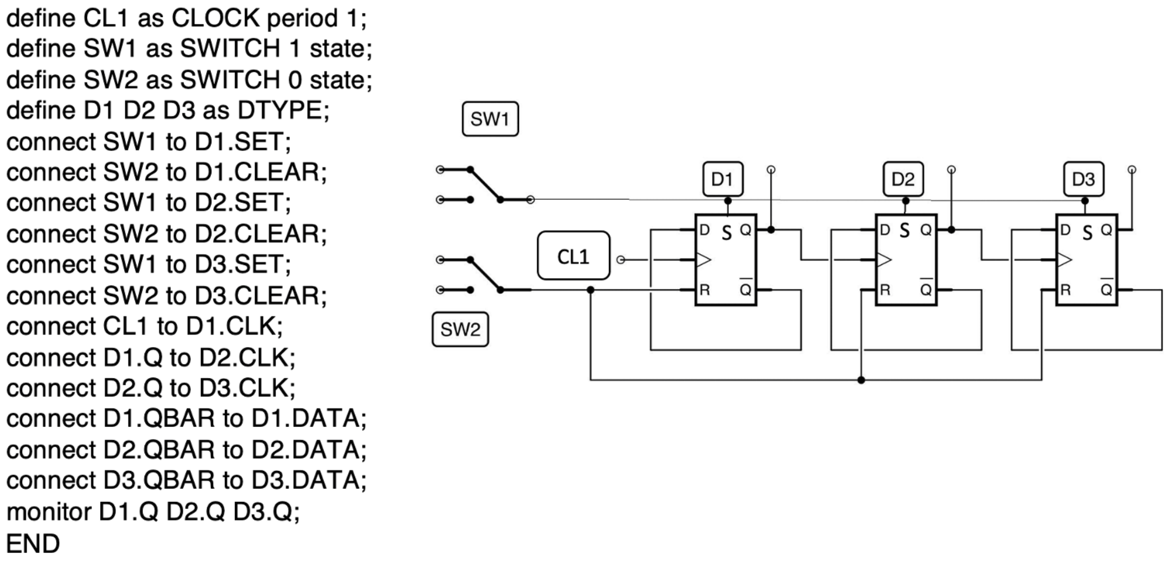
Input-to-input and output-to-output errors are syntactically forbidden (and therefore do not need to be specified as semantic errors). The same is true of the "QUALIFIER_PRESENT" error in the Devices class.

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

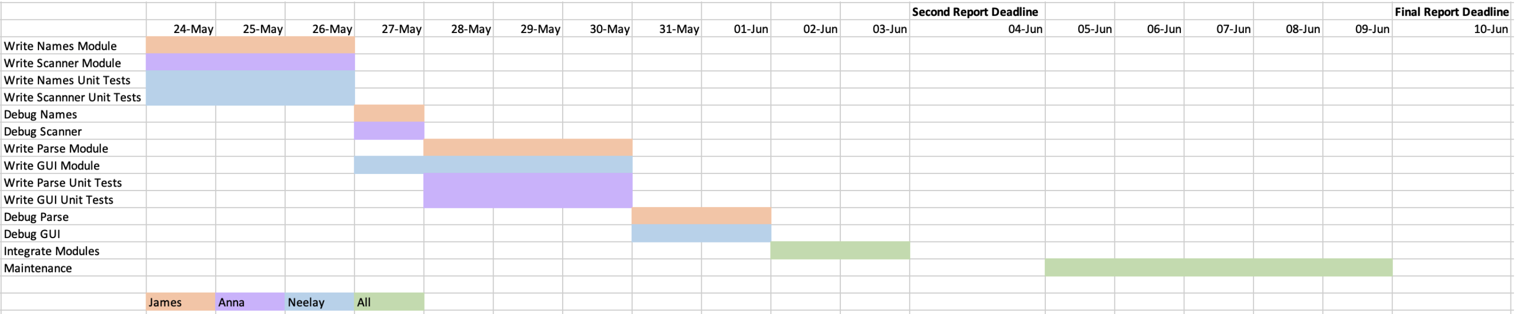


```
define CL1 as CLOCK period 1;
define SW1 as SWITCH 1 state;
define SW2 as SWITCH 0 state;
define D1 D2 D3 as DTYPE;
connect SW1 to D1.SET;
connect SW2 to D1.CLEAR;
connect SW1 to D2.SET;
connect SW2 to D2.CLEAR;
connect SW1 to D3.SET;
connect SW2 to D3.CLEAR;
connect CL1 to D1.CLK;
connect D1.Q to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

no AND, OR. NOR, XOR gates

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														