IIA GF2 Software: Final Report

Tim Hillel (th389)
Team 8
Downing College

Contents

1 Description **2**

2 Teamwork and Development Style **3**

3 My Contribution **4**

4 Testing **4**

5 Conclusions **4**

A Code Listings **5**

B Test Definition Files **5**
B.1 SIGGEN 5
B.1.1 Definition File 5
B.1.2 Circuit Diagram 5

C EBNF **6**

D User Guide **7**

E File Listing **8**

F Gantt Chart **9**

1 Description

Our logic simulator is able to simulate any number of circuits which include the following devices:

- Clocks
- Switches
- AND gates (Up to 16 inputs)
- NAND gates (Up to 16 inputs)
- OR gates (Up to 16 inputs)
- NOR gates (Up to 16 inputs)
- XOR gates
- D-Type flip-flops
- Signal generators

Figure 1: UML Class diagram of our logic simulator

2 Teamwork and Development Style

In the project planning stage, we split the development of our logic simulator into five major phases: specification, design, implementation, testing and maintenance. The timeframe was then decided for each task and each task was assigned to either a team member or the whole team, depending on the nature of the task. This is shown in the Gantt chart in Appendix F. Overall we found we stuck well to the planned timescale, often completing tasks before the internal deadlines we had set, which allowed longer for incremental improvements to code in the testing and debugging stages.

Each member of the team was also assigned a general project role as follows:

Project manager: (T Hillel) - Responsible for project planning including delegation of tasks and ensuring that the project runs to the set timescale.

Programming administrator: (J Magee) - Responsible for upkeep of the project directory including performing builds and keeping legacy versions of the simulator.

Client representative: (M Jackson) - Responsible for ensuring that the project meets the client's requirements for the logic simulator as defined in Appendix A of the GF2 Project Handout.

Overall this system worked relatively effectively, as each member of the team had both technical and general tasks which they were responsible for. We did find during the running of the project that there was not a strong need for a strict organisational structure. This was mainly due to the extensive use of git for revision control through GitHub. This allowed each member of the team to test, edit and change any code within the project remotely whilst fully tracking any changes made to code. GitHub allowed issues to be raised when bugs were found that would require more than a quick fix or when functionality of a class did not exactly meet the client's specification.

Occassionally this allowed for problems when multiple users edited the same file before attempting to push it to the repository, but `git merge` was used to deal with these issues effectively.

We also found that the breakdown of the tasks we had initially planned for were not completely equal, as an example the names and scanner class implemented a lot of the code written in the preliminary exercises, and so Jamie Magee, who was assigned to this task, had time to help Tim Hillel with the parser. GitHub allowed the team to track which tasks required more input and so a member could assist when they had finished their section, such as Jamie Magee helping with `parser.cc`. Whilst Martin Jackson's primary task was the GUI during the developement phase, he was also always on hand to help debug problems in the other classes.

For the maintenance phase of the project, each team member was assigned to one of the modifications required:

Nonzero Hold Time of Bistables: J Maggee

Signal Generators: T Hillel

Continious Simulation: M Jackson

Again we found that the tasks were not balanced; the nonzero hold time of bistables modification was a more involved task, and so Martin Jackson helped Jamie Maggee with this task whilst Tim Hillel implemented general maintenance changes as suggested by feedback from Interim Report 2.

Overall, the team worked very effectively through the task and there were no issues that caused any major problems. Using GitHub to manage the task definitely aided with the efficeincy of the team, and the experience has provided useful skills for the future.

3 My Contribution

I took on responsibility for writing the names and scanner classes. In addition, I wrote approximately 25% of the parser class. Once the majority of the software was written, I designed a definition file for each error and warning our logic simulator can throw and then wrote a shell script which would attempt to run each definition file, and record the output from our logic simulator. The shell script can be found in code listing **??**

4 Testing

We used two main tests of testing - unit and system testing - both of which are industry standard practices. For our unit testing, Martin wrote an errors class which compared the actual output from various units of code, to the expected output. For system testing I wrote a shell script which passed definition files to the logic simulator and recorded the output in a text file. There were two variatons on the shell script: One which ran known good definition files and therefore had to input the commands to run the simulation in addition to recording the output; Another which ran known bad definition files and only expected parsing errors which it recorded.

5 Conclusions

A Code Listings

Please see attatched Interim Report 2 for all code listings.

B Test Definition Files

Shown below are the test definition files for funcionality added in the maintenance phase of the project.

Please see attatched Interim Report 2 for test definition files for functions implemented up to section 10 of the project.

B.1 SIGGEN

B.1.1 Definition File

```
DEVICES
SIGGEN S1: 1 0 1 0 1 0 0 1 0 0 1 0 0;
SIGGEN S2: 1 1 1 0  0 1 0 1 0 1 0 1   0;
AND A1: 1 6;
END

CONNECTIONS
A1.I1=S1;
A1.I2=S2;
END

MONITORS
S1;
S2;
A1;
END
```

Listing 1: siggen.gf2 - Note defined a very long waveform to test ability to take waveform of arbitrary length; defined other waveform with spaces in sequence to test this ability

B.1.2 Circuit Diagram

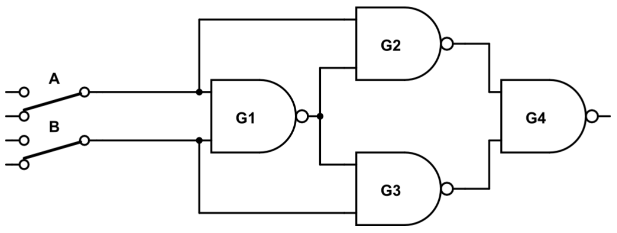

Figure 2: Circuit diagram of two Signal Generators conenected to a 2-input AND gate

C EBNF

```
specfile = devices connections monitors

devices = 'DEVICES' dev ';' {dev ';'} 'END'
connections = 'CONNECTIONS' {con ';'} 'END'
monitors = 'MONITORS' {mon ';'} 'END'

dev = clock|switch|gate|dtype|xor|siggen
con = devicename'.'input '=' devicename['.'output]
mon = devicename['.'output]

devicename = letter {'_'|letter|digit}
input = 'I'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|
  '10'|'11'|'12'|'13'|'14'|'15'|'16')|'DATA'|'CLK'|'SET'|'CLEAR'
output = 'Q'['BAR']

clock = 'CLOCK' devicename':'digit{digit}
switch = 'SWITCH' devicename':'('0'|'1')
gate = ('AND'|'NAND'|'OR'|'NOR') devicename':'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|
  '9'|'10'|'11'|'12'|'13'|'14'|'15'|'16')
dtype = 'DTYPE' devicename
xor = 'XOR' devicename
siggen = 'SIGGEN' devicename':'('0'|'1'){'0'|'1'}
```

Listing 2: EBNF

D User Guide

E File Listing

F Gantt Chart

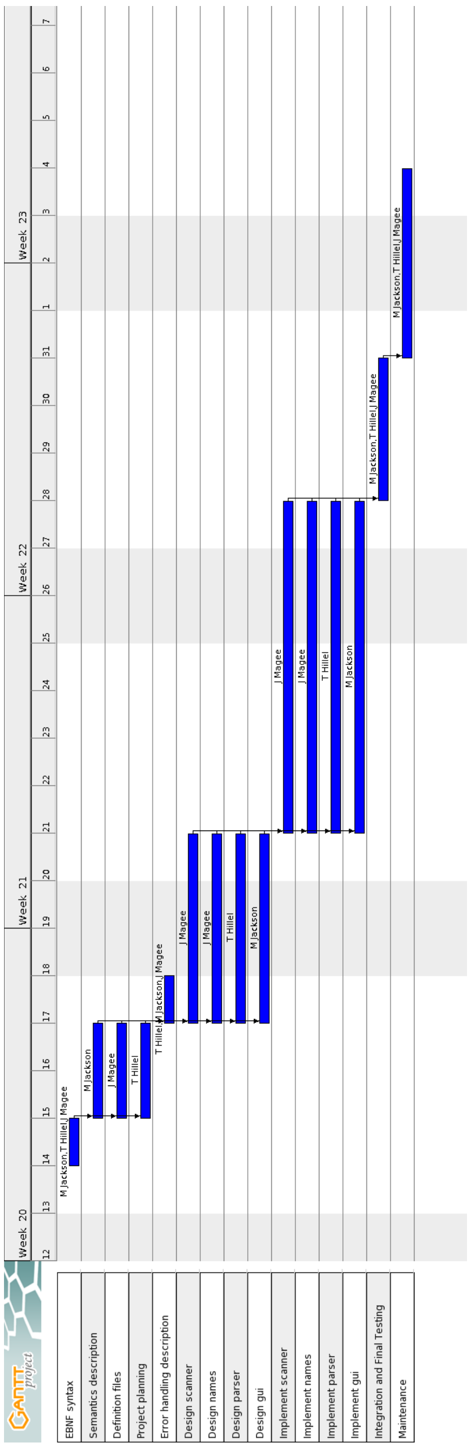

Figure 3: Gantt chart showing key events in development cycle