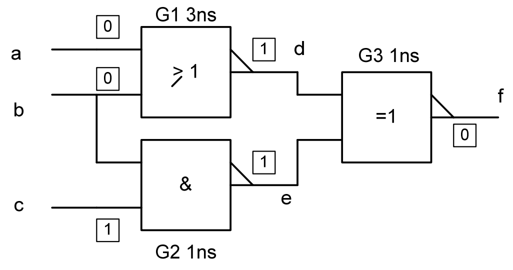
Assessed Assignment Part 2

- The objective of this assignment is to write a simple event-driven logic simulator.
- This document explains the following:
  1. Format of the input file that specifies the circuit.
  2. The commands that are used to drive the simulator and the way that output events are reported to the console window.
  3. How an event-driven simulator works.
  4. A basic structure of your software package (as a suggestion).
  5. Deliverables and deadlines

A Simple Circuit Specification

- This circuit can be described in a file, say, “example.cct” as:

```
.input a b c           # primary inputs
.output f              # primary outputs
.gate nor 2 d a b 3
.gate nand 2 e b c 1
.gate xor 2 f d e 1
.end
```



Circuit Specification Format

- All circuit files starts with `.input` and `.output` keywords, specifying the primary inputs and outputs of your circuit respectively.

```
.input <node_name_list>
.output <node_name_list>
```

- All node names are separated by space of tab characters, and must start with an alphabet, but can contain numbers or underscore. DIN, d3, b_input are all valid node names.
- All comments, inline or otherwise, starts with “#”.
- Gates are specified with:

```
.gate <gate_type> <#inputs> <output> <i/p_list> <delay>
```

- The gate types you must implement are AND, NAND, OR, NOR and XOR. You are welcome to extend this list.
- For those with ambition (not required), you may also implement a D-flipflop:

```
.ff D <output> <input> <clock> <initial_value> <delay>
```

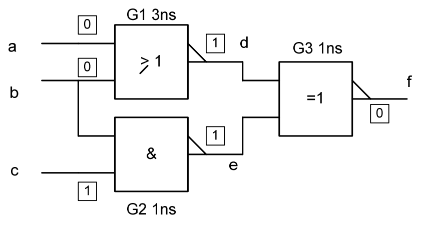
- `.include <file_name>` can be used to read in another circuit from a file in the current director.

User interface

- The logic simulator should be a console program. Users should see a prompt character ‘>’, after which the user will enter one of the following commands (not case-sensitive):
  - `> H <node_list>` – set all nodes in node_list to high
  - `> L <node_list>` – set all nodes in node_list to low
  - `> D <node_list>` - display the values of nodes in node_list (‘*’ = all)
  - `> W <node_list>` - report any changes in nodes in node_list (‘*’ = all)
  - `> I` – initialize circuit (see later)
  - `> S <time>` – simulate for <time>ns (if missing, simulate until circuit settles)
  - `> L <circuit>` - load the circuit from the file <circuit.cct>
  - `> R <filename>` – execute all the commands in <filename.cmd>
  - `> Q` – quit the simulator
- When the simulator is first started, the circuit filename may or may not be one of the command line arguments.
- The simulator should also automatically execute the commands in the file “default.cmd” if one exists in the current directory. You can use this feature to help the testing of your programme without typing command in each time.

How it works?

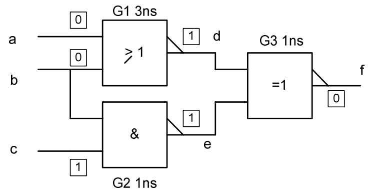
- All nodes can take on 3 values: ‘0’, ‘1’ or ‘x’ (unknown).
- Consider the simple circuit, assuming that the initial values at all nodes are as shown, let us now issue the command: `> h b`
- The sequence of events that follows are:



Time	Node	Transition
0	b	0 → 1
1	e	1 → 0
2	f	0 → 1
3	d	1 → 0
4	f	1 → 0

How it works?

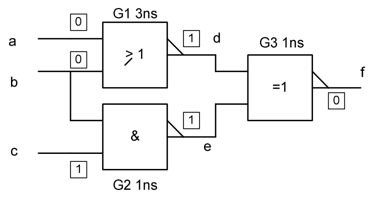
- In the simulator, the following steps should happen:
  1. Node b goes from 0 to 1 creates event 1 at time 0ns. This is inserted in an event list.
  2. G1 and G2 are potentially affected by event 1 because b drives both.
  3. G1 is evaluated and node d goes from 1 to 0 at time 3ns. This is inserted in the event list.
  4. G2 is evaluated and node e goes from 1 to 0 at time 1ns. This is inserted in the event list BEFORE the node d event.
  5. Node b event 1 is now completed and is removed from list.
  6. Next event is e 1->0 at 1ns, this affects G3. G3 evaluated. Node f goes from 0 to 1 at 2ns. This is inserted BEFORE node d event.
  7. Repeat for all events. When all events on the event list is dealt with, the simulation is completed.



Time	Node	Transition
0	b	0 → 1
1	e	1 → 0
2	f	0 → 1
3	d	1 → 0
4	f	1 → 0

What should be displayed?

- Here is an example of what should be displayed, assuming the state of the circuit as shown, when you set b to high and simulate:



```
…..
> h b
> s
0ns: b 0->1
1ns: e 1->0
2ns: f 0->1
3ns: d 1->0
4ns: f 1->0
> d *
a: 0
b: 1
c: 1
d: 0
e: 0
f: 0
```

- Everything that is displayed on the console should also be recorded in a log file “log.txt”.

Program Structure

- Here are guidelines for writing to this program:
  1. If command line argument exists, read circuit file and create an internal data structure for the circuit. You need to design the structure carefully remembering that you don’t know how many gates there are!
  2. You need a routine that interpret the circuit keywords (input, output, gate etc.), you will also need to create an addition data structure for all the nodes in the circuit. This should include its current value, what gates it drives, whether it is primary input/output or an internal node and whether the node is a watch node (i.e. if change occurs, report this).
  3. You need to write a routine to deal with the user interface commands.
  4. You need to maintain a linked list of events. Each item should have the node of the event, the time it occurs and the changed value of the event.
  5. For initialisation, you should initially make all nodes to ‘X’. Then create at time 0, one event for each primary input going from ‘X’ to ‘0’.
  6. The main simulation engine (invoked when you either initialize the circuit or run a simulation step) should be written. This effectives perform the steps outline in previous slides.

Deliverables

- The deadline for this assignment is Friday 5th May at 17.00. Submission is through the web as before. An email will be sent out nearer the time to give you detailed instructions.

- Your submission should be a zipped file containing:
  1. A text or WORD file explaining BRIEFLY the structure of your program (in the form of structure diagram).
  2. A circuit specification file of the test circuit you have tried.
  3. A default.cmd file that exercise your simulator properly.
  4. A log.txt file that records the output of the simulator when you circuit is exercised by the default.cmd file.
  5. All source codes of your program (no object or .exe files).