EE2703 - Week 4 - Logic Simulation

Nitin Chandrachoodan <nitin@ee.iitm.ac.in>

February 11, 2023

1 Logic Simulation / Evaluation

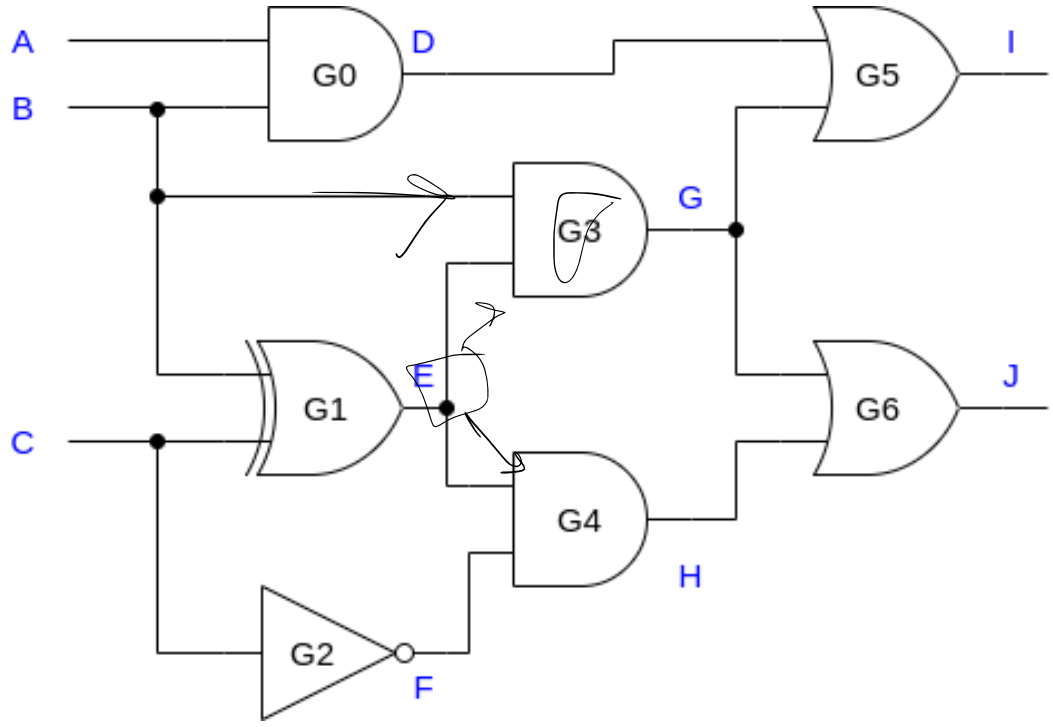
A combinational digital circuit is built using basic logic gates like AND, OR, NOT, XOR, XNOR etc. Each gate implements a specific boolean functionality. The inputs to the circuit are called the "primary inputs", and the final outputs of the circuit are called "primary outputs". The primary inputs and the outputs of all gates in the circuit are referred to as "nets". Shown below is an example of a combinational digital circuit with A,B,C as primary inputs and I,J as primary outputs.

The process of *logic simulation* involves evaluating the state of each net in the combinational circuit. Clearly we need a starting point, and here we assume that the *primary inputs* have been assigned values, and we need to evaluate the remaining net values.

1.1 Combinational circuits

An example of a combinational circuit is shown in this figure. In general, we will make some simplifying assumptions on the types of circuits we will see:

- no combinational loops are permitted (so no SR-latches for example). In general combinational loops are not permitted in digital logic except under very carefully controlled circumstances that we don't deal with in this assignment.
- only a very restricted subset of gates (detailed below) are permitted. In particular, we do not take gates with 3 or more inputs, or gates where the inputs are not symmetric.



1.2 Netlist

A circuit netlist provides a description of the circuit connectivity. Each gate is identified using a unique identifier (e.g. G0, G1 etc). The netlist contains the connectivity information for each gate in the following format.

`<gate_id> <gate_type> <input1> [input2] <output>`

List of possible logic gate types: AND2, OR2, NOT, NOR2, NAND2, XOR2, XNOR2

The netlist corresponding to the above example:

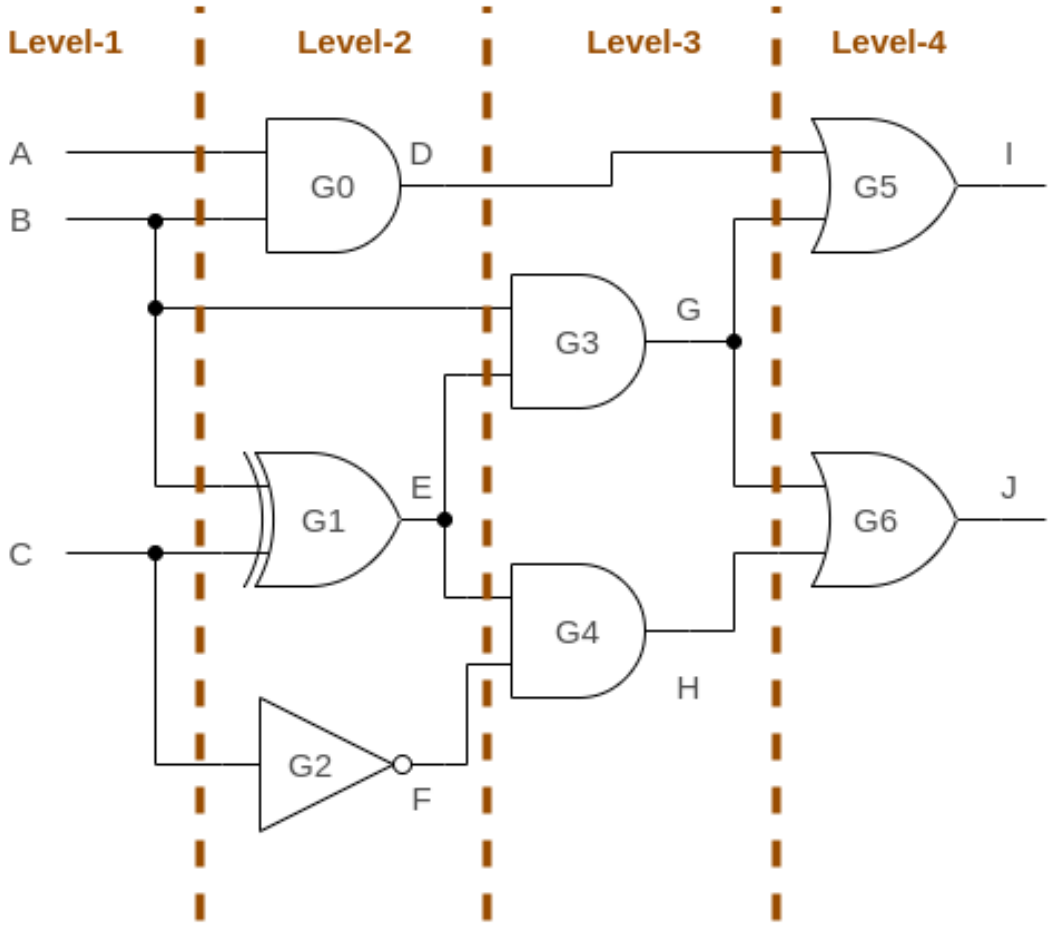
```
G0 AND2 A B D
G1 XOR2 B C E
G2 NOT C F
G3 AND2 B E G
G4 E F H
G5 D G I
G6 G H J
```

1.3 Circuit Evaluation

Given the state of the primary inputs, the state of all nets in the circuit can be obtained by traversing the circuit in the topological order wherein the primary inputs of the circuit are nets at the first topological level, the nets driven by the primary inputs lie at the second topological level and so on.

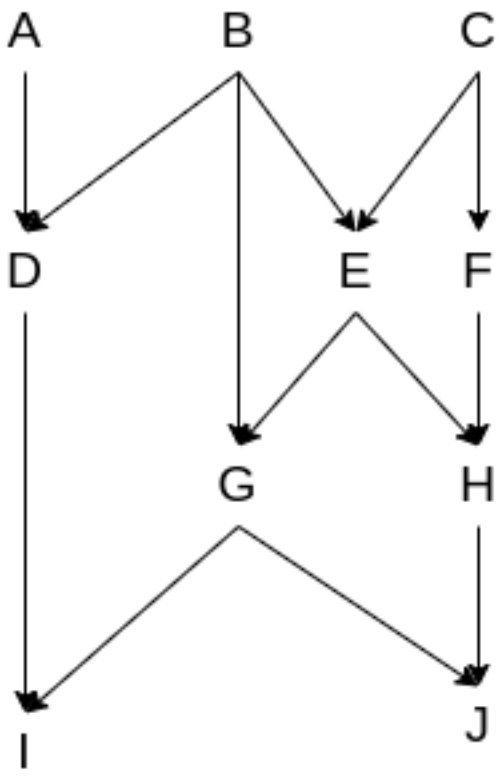
```
PrimaryInput.level = 1
Net.level = max (net.input1.level, net.input2.level) + 1
```

Example: Net G is at topological level 3 since it is driven by nets B and E which at levels 1 and 2 respectively.



We can convert the circuit into a **Directed Acyclic Graph**, where nodes of the graph represent nets in the circuit and the output of each gate has incoming edges from the inputs.

The DAG corresponding to the above netlist is shown here:



2 Creating DAGs with Networkx

In this assignment, we will only be using very basic graph functionality. As such, it should be easy for you to implement the required classes or modules yourself in Python. However, the *topological sort* functionality can be a little complicated to implement, so you are permitted to use the `networkx` module in Python to realize graph functionality. You only need the most basic functionality here, and some examples of useful commands are given below.

[ ]:
```
import networkx as nx

# create a DAG
g = nx.DiGraph()

# add nodes and edges to the DAG
g.add_edges_from([("A", "D"), ("B", "D"), ("B", "E"), ("B", "G"), ("C", "E"),␣
 ↪("C", "F"), ("D", "I"), ("E", "G"),("E", "H"), ("F", "H"), ("G", "I"), ("G",␣
 ↪"J"), ("H", "J")])
nx.set_node_attributes(g, {"A": "PI", "B": "PI", "C":"PI", "D":"AND2", "E":
 ↪"XOR2", "F":"INV", "G":"AND2", "H":"AND2", "I":"OR2", "J":"OR2"},␣
 ↪name="gateType")
print(g.nodes(data=True))

# sort the nodes in topological order
```

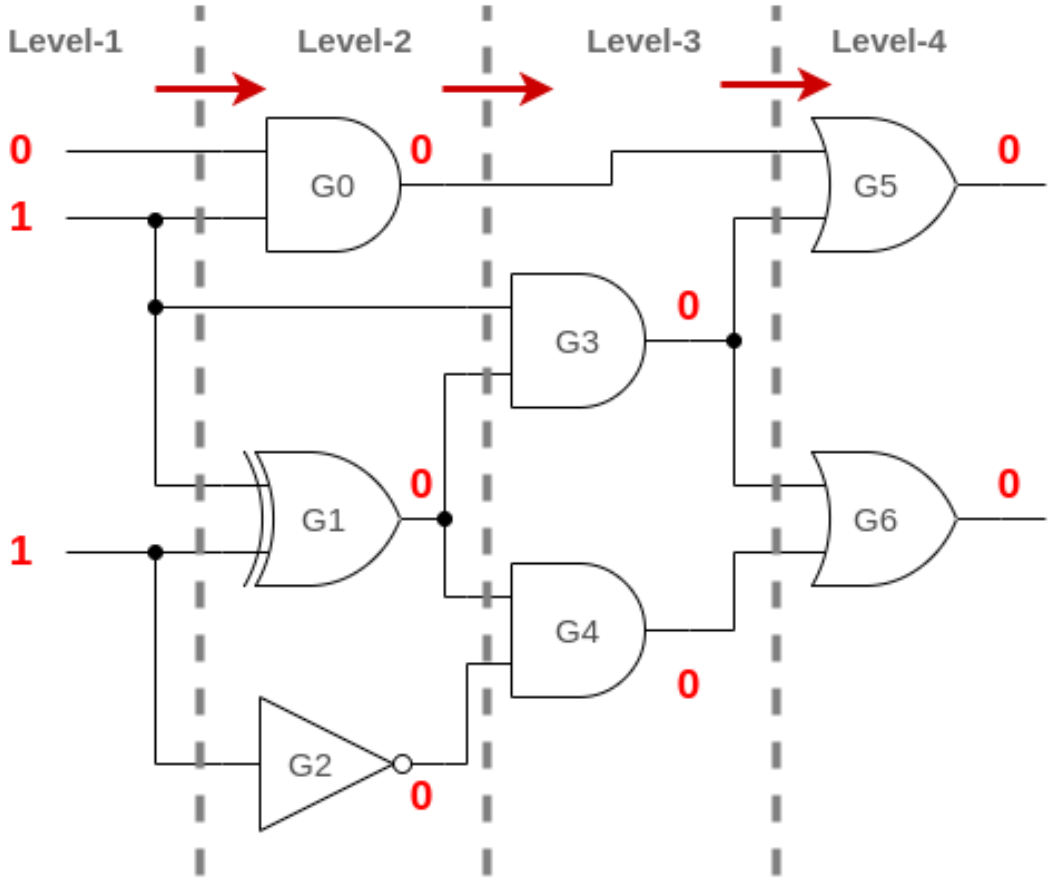
```
nl = list(nx.topological_sort(g))
print('Nodes in topological order',nl)

# find children of a node
print('Successors of node B', list(g.successors("B")))

# find parents of a node
print('Parents of node H', list(g.predecessors("H")))

['C', 'F', 'A', 'B', 'D', 'E', 'G', 'H', 'I', 'J']
```

3 Evaluation for a fixed state of primary inputs

Given the state of the primary inputs, we need to evaluate the state of all other nets. This can be done in one of 2 ways:

3.1 Topologically ordered evaluation

Once the nets are arranged in the topological order, the state output of each gate is found based on the state of the inputs and the functionality of the logic gate. For A,B,C=0,1,1, the states of all nets in the circuit can be evaluated as shown below.



3.2 Event-driven evaluation

Instead of using a topological sort, we could also use queues for updating the states in an event driven approach. To do this, we start with a state table that contains the current state of each net, initialized to 'x' at the start of the simulation. We form a *queue* that tracks the nets whose state has to be re-evaluated.

A *queue* is a data structure that has several uses in algorithms: for our purposes, it has a very simple meaning: we can insert elements into a queue, and we can *dequeue* or *pop* them from the queue. Always, the first element to be inserted will be the first element to be dequeued - the queue just enforces correct order. How the queue is implemented, the efficiency of the implementation etc. is out of scope of this course.

First, the states of the primary inputs needs to be updated so we add these to the queue. Next, we dequeue nets from the queue and evaluate the new state. If the new state is different from the current state, the sucessors of the net are added to the queue for evaluation.

The given netlist, if evaluated using an event driven approach, would go roughly as follows (note that this is not a unique ordering – depending on the order in which we insert elements into the queue, the evaluation could differ, but the final result should remain the same).

		State table										Event	Queue					Remark
		A	B	C	D	E	F	G	H	I	J							
Initialize		x	x	x	x	x	x	x	x	x	x		A	B	C			Primary inputs
Dequeue	A	0	x	x	x	x	x	x	x	x	x	A: x->0	B	C	D			
	B	0	1	x	x	x	x	x	x	x	x	B: x->1	C	D	D	G	E	Order of D, G, E not important
	C	0	1	1	x	x	x	x	x	x	x	C: x->1	D	D	G	E	F	
	D	0	1	1	0	x	x	x	x	x	x	D: x->0	D	G	E	F	I	D depends on A, B
	D	0	1	1	0	x	x	x	x	x	x	-	G	E	F	I		D evaluates same: no event
	G	0	1	1	0	x	x	x	x	x	x	-	E	F	I			G: dep on B, E but E = x, so G remains x (no events)
	E	0	1	1	0	0	x	x	x	x	x	E: x->0	F	I	G	H		E: dep on B, C. Triggers G, H evaluation
	F	0	1	1	0	0	0	x	x	x	x	F: x->0	I	G	H	H		F: dep on C. Triggers H again
	I	0	1	1	0	0	0	x	x	x	x	-	G	H	H			I: dep on D, G but G = x. No event
	G	0	1	1	0	0	0	0	x	x	x	G: x->0	H	H	I	J		G: now E is known, so eval G, trigger I, J
	H	0	1	1	0	0	0	0	0	x	x	H: x->0	H	I	J	J		H: dep on E, F. Triggers J
	H	0	1	1	0	0	0	0	0	x	x	-	I	J	J			Repeat H, remains 0. No event for 0 -> 0
	I	0	1	1	0	0	0	0	0	0	x	I: x->0	J	J				
	J	0	1	1	0	0	0	0	0	0	0	J: x->0	J					
	J	0	1	1	0	0	0	0	0	0	0	-						J already eval; no more events. Stop

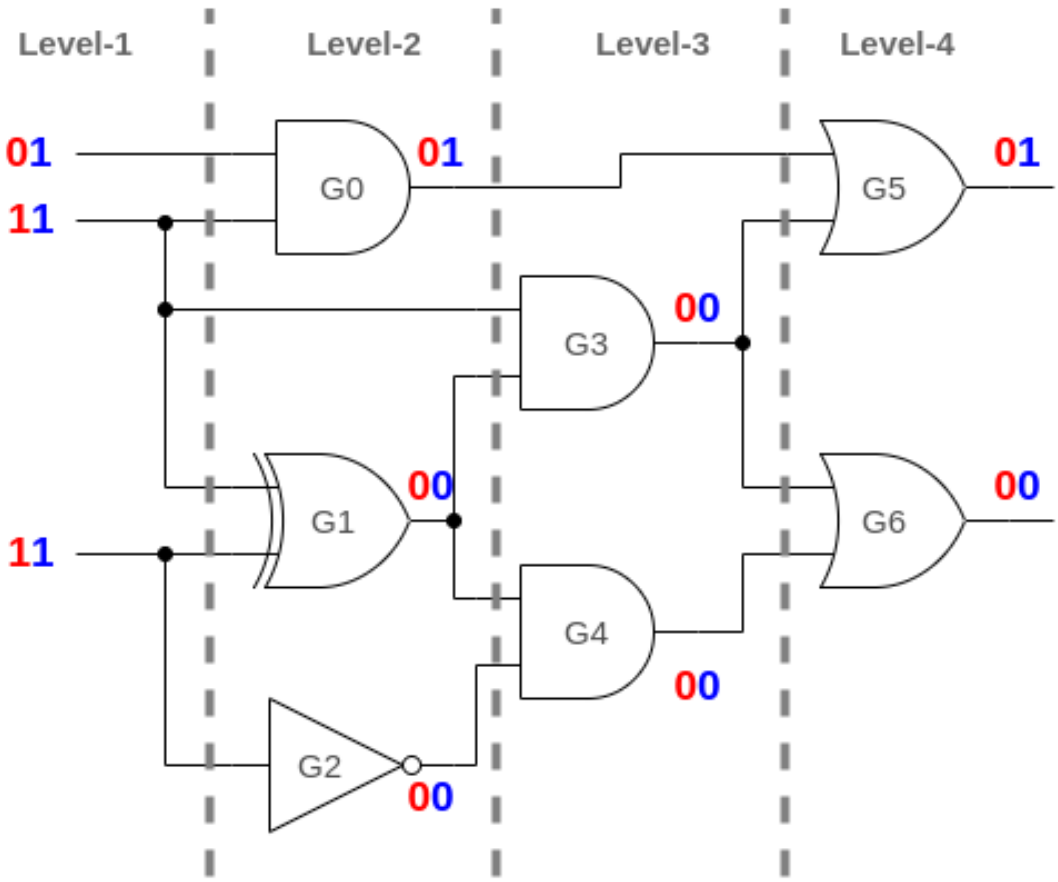
4 Evaluation for inputs changing over time

The inputs to a circuit could change over time. Therefore, the state of all nets needs to be re-evaluated. For example, consider the case where the primary inputs change from A,B,C=0,1,1 at t=0sec to A,B,C=1,1,1, at t=1sec.

The evaluation of the circuit can be performed in two ways.

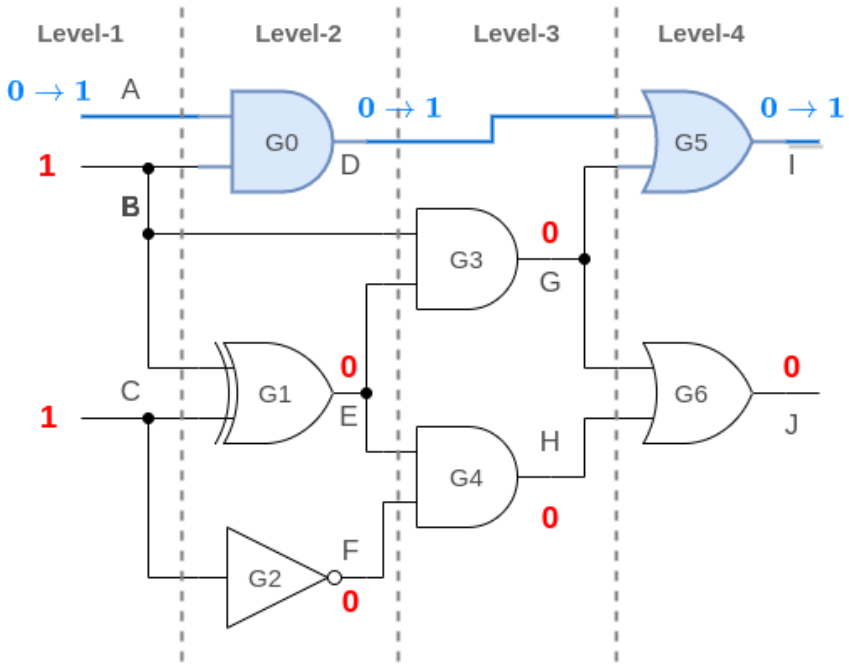
4.1 Method1:

Evaluate the entire circuit twice. The time complexity for this method is *O(2N)* where N is the number of nets in the circuit.



4.2 Method 2:

The entire circuit does not need to be re-evaluated. We can use the event-driven approach and propagate the events from the primary inputs that have been altered.



The events corresponding to this are captured in the event table here:

		State table										Event	Queue	Remarks
		A	B	C	D	E	F	G	H	I	J			
Initialize		0	1	1	0	0	0	0	0	0	0		A	A changed: triggers D
Dequeue	A	1	1	1	0	0	0	0	0	0	0	A: 0->1	D	D changed: triggers I
	D	1	1	1	1	0	0	0	0	0	0	D: 0-> 1	I	I changed. No new events
	I	1	1	1	1	0	0	0	0	1	0	I: 0->1		

5 Assignment

You will be given certain input netlists, and corresponding sets of input events, in the formats given here:

5.1 Input format for netlist

The file will consist of a number of lines of the form

`<gate_id> <gate_type> <input1> [input2] <output>`

where

- `<gate_id>` will start with an alphabet and can contain alphanumeric characters (A-Z, 0-9, upper or lower case).
- `<gate_type>` will be one of the following logic gate types: `AND2`, `OR2`, `NOT`, `NOR2`, `NAND2`, `XOR2`, `XNOR2`
- `<input1>` will be an alphanumeric value - this is the name of a *net*
- `<input2>` is also alphanumeric (for the `NOT` gate there may be no `<input2>`)

- `<output>` is also alphanumeric

Each of the values `<input1>`, `<input2>` and `<output>` are *net* names and indicate connections. You need to read in these values and construct the corresponding connectivity graph.

The netlist corresponding to the above example:

```
G0 AND2 A B D
G1 XOR2 B C E
G2 NOT C F
G3 AND2 B E G
G4 E F H
G5 D G I
G6 G H J
```

5.2 Input format for input events

We will use a generic format for inputs as shown below: each line corresponds to a different *time instant*, but at each time instant we will specify all primary input values, even if they have not changed. You need to read these in, and decide which ones need to be inserted into the evaluation queue (for the queue based solution). For the topological ordered evaluation, you will of course just evaluate all values.

```
line 1  : <List of primary inputs>
line 2  : <State of each PI>
```

Example:

```
A B C
0 1 1
1 1 1
```

5.3 To Do

Write a python code for the following

1. Read the netlist and sort the nets in the toplogical order.
2. Read the list of input vectors, evaluate the circuit and find the state of all nets in the circuit.
   - Using topological sort and multiple rounds of circuit evaluations.
   - Using event driven simulation.
3. Briefly discuss your results: which approach is faster/more efficient, and for what types of inputs. Your discussion should be brief (not more than 1 page) but should give logical reasons rather than just blanket statements.

Output file format

```
line 1: List of all nets in the circuit (sorted in alphabetical order of net-names)
line 2: List of states for input vector 1
line 3: List of states for input vector 2
.
.
.
```