University of Tehran

Electrical and Computer Engineering Department

ECE (8101) 342

Object Oriented Modeling of Electronic Circuits – Spring 1401-02

Homework 2: C++ Programing for Digital Systems' Applications
Due Date: Esfand 23

Event-driven gate-level circuit simulation using a timing wheel data structure:

Logic simulation is a form of **design verification testing** that uses a model of the designed system. Figure 1 shows a schematic view of the simulation process. As shown, logic simulation is to apply an input vector to primary inputs (PIs) of the circuit model and compute the response values at primary outputs (POs), considering the gate and wire delays.

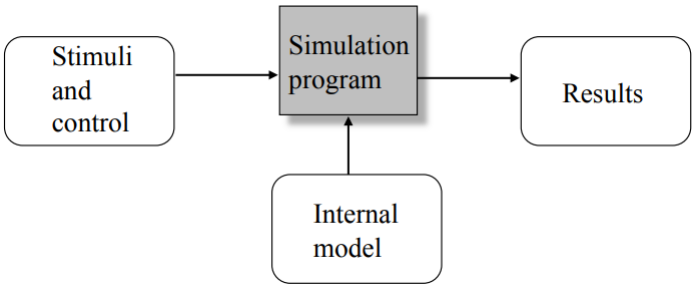


Figure 1 – Simulation process.

To perform logic simulation, a very effective procedure is event-driven simulation that is based on the recognition that an event on a signal causes new events. Gates whose inputs have events are called **active** and are placed in an **activity list** (also named **evaluation list**). The simulation proceeds by removing a gate from the activity list and evaluating it to determine whether its output has an event. The process of evaluation stops when the activity list becomes empty.

An event-driven simulator only does the necessary amount of work. For logic circuits, in which typically very few signals change at a time, this can result in significant savings of computing effort. In general, the activity of a digital circuit is between 1% to 5%. However, the biggest advantage of this technique is its ability to simulate any arbitrary delays. This is done by a procedure known as **event scheduling** or **event list**.

Suppose the evaluation of an active gate generates an event at its output. If the gate has a switching delay of x units, then the event should take effect x-time units later. To correctly consider the effects of delays, the simulator distributes the activity list in time. Event scheduling is the procedure of distributing the activity caused by events over time according to the specified

delays. Using separate event and activity lists avoids any dependence on the order in which events are processed since the evaluations occur only after all nodes have been updated.

In a nutshell, Figure 2 shows the conceptual flow of event-driven simulation. The simulation time is advanced for the next time for which events are pending. This becomes the current simulation time. Next, the simulator retrieves from the event list the events scheduled to occur at the current time and updates the values of the active signals. The active signals are then followed to determine the activated elements. The evaluation of the activated elements may result in new events. These are scheduled to occur in the future time according to delays associated with the gates. The simulator inserts the newly generated events into the event list. The simulation continues as long as there is a logic activity in the circuit. That is until the event list becomes empty.

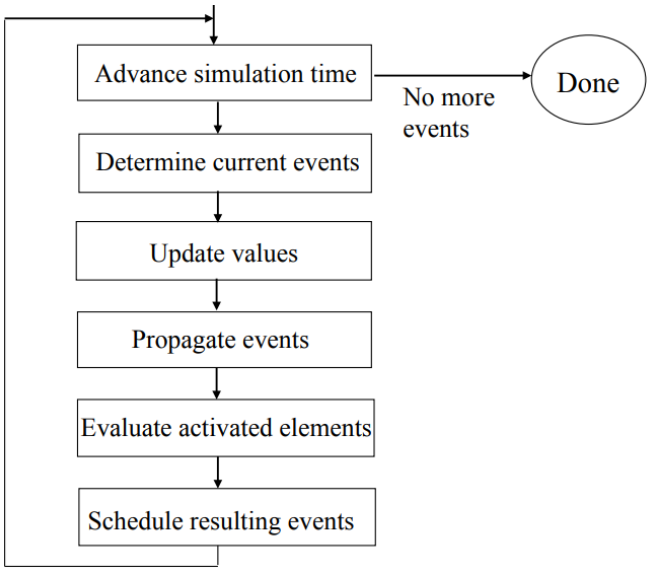
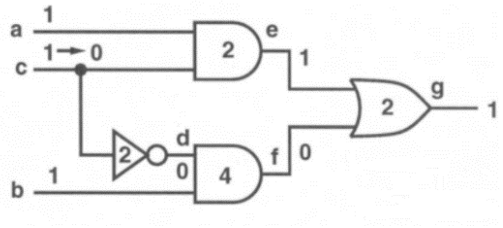
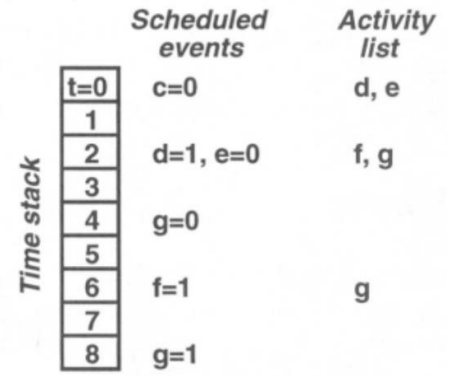


Figure 2 – The main flow of event-driven simulation.

The example shown in Figure 3 illustrates event-driven simulation with circuit delays. Gate delays are shown inside the gate symbols while wire delays are assumed to be zero. For an input *abc=111*, the gate-level simulator should be able to compute the output response *g=1* at time *t=8*. The gates and their output signals are given the same name. The signal values shown in the figure are the steady-state values for the initial vector. We will denote the simulated time by variable *t*, which can take discrete values of *0, 1, 2, etc*. These values are assigned to the slots of a time stack. Attached to each time slot there is an event list of scheduled events and an activity list as shown in Figure 3. We will perform the simulation for an input event *1→0* on *c* occurring at *t=0*. The simulation processes the event *c=0*, which had been initially scheduled for this time *t=0* (*t=0* is now the current simulated time). Due to the event *c=0* from the event list, the signals *d* and *e* are placed in the activity list, which has as a result the insertion in the event list of the new events *d=1*, *e=0*, scheduled at time *t=2*. At time *t=2*, the scheduled events are *d=1* and *e=0* that resulting in placing signals *d* and *e* in the activity list. Evaluations of *f* and *g*, using the currently scheduled values of *d* and *e*, produce two new events, namely, *f=1* scheduled at *t=6* and *g=0* scheduled at *t=4*. The simulation produces the other events for all signals in the circuit as shown in Figure 3.



(a) A gate-level example circuit



(b) Illustration of Scheduled events (event list) and Activity list (evaluation list)

Figure 3 – An example of event-driven simulation.

In this example, the time stack would continue to grow as the simulation progresses. This can be avoided by scheduling events **relative** to **current time**. For any current time, all new events are scheduled only at future times. Also, the difference between the current time and the farthest time at which we can schedule an event is limited by the maximum delay a gate has in the entire circuit. Thus, in an efficient implementation, all event scheduling occurs relative to the current time. The number of slots in the time wheel equals *max +1*, where the *max* unit is the largest delay experienced by any event in the circuit.

We can view the time array as a circular stack, called the **time wheel** as shown in Figure 4. As discussed above, the number of slots in the time wheel has to be large enough so that the largest delay in the circuit can be accommodated. For example, the minimum size of the timing wheel necessary to simulate the circuit in Figure 3 is *5*, so that events that are *4*-time units ahead in time can be scheduled (*4* is the largest delay among all the gates in the circuit netlist).

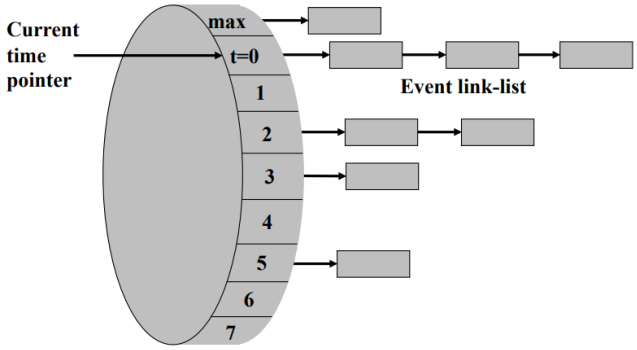


Figure 4 – Time Wheel (Circular Stack) representation.

In this homework, you are to consider the C432 circuit with the following delay values:

- NOT: 2
- NAND: 5
- NOR: 4
- AND: 3
- XOR: 6

A) Write a C++ parser program to read a gate-level Verilog description and turn it into the necessary data structure for the above event-driven simulation. The netlist consists of primitive gates and their interconnections in standard Verilog format.

B) Write a C++ program for taking timed data inputs using a vector format and apply to the internal data structure above.
An example of timed data inputs:
#10 0101
#20 1100
#30 0100

C) Write a C++ program to implement an event-driven simulator for gate-level circuits as discussed above.

D) Verify your implementation using different scenarios (at least five scenarios).