

## HARDWARE DESCRIPTION LANGUAGES (HDLs)

- Textual Representation of Digital Circuit
  - Yet Another Way (truth tables, equations, circuit diagrams, etc.)
- Why Have HDLs?
  - Documentation
    - First common HDL (VHDL) Documentation Language
  - Simulation
    - First common HDL for Simulation (Verilog)
  - Synthesis
    - Both VHDL and Verilog used for this
    - Currently, more emphasis on SystemC, SystemVerilog

## HARDWARE DESCRIPTION LANGUAGES (HDLs)

- Why are HDLs Important?
  - Easy way to Describe Large Circuits
  - Large Teams of Designers can Work Concurrently Using Software Engineering Techniques
  - The Specification can be Simulated
  - The Specification can be Synthesized
  - HDLs Support Multiple Levels of Abstraction

## Definitions

- Specification
  - Description of the Desired Functionality
- Simulation
  - Given a Model and Inputs, Predict the Output
- Synthesis
  - Transform One Model into Another

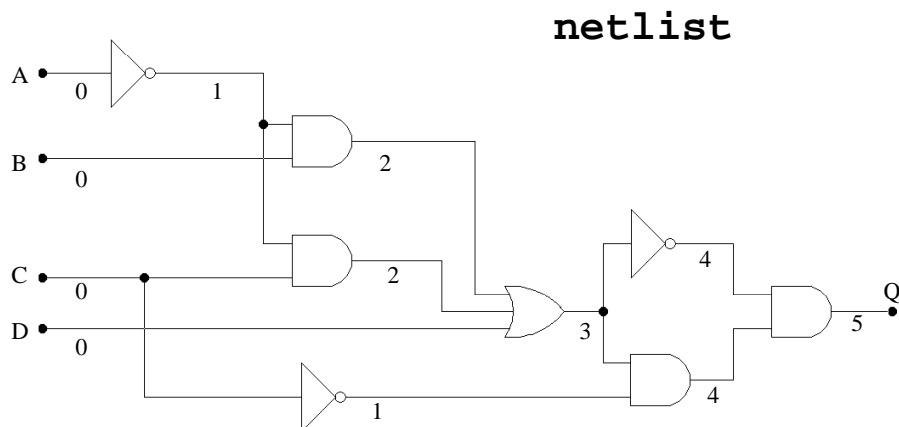
## Why Another Language?

- Sequential Languages
  - Programming Languages are *Sequential*
  - Each Statement is Executed in Order of Appearance
- Hardware is a Parallel Instance
  - Model in Terms of *Events*
  - More Natural Way to Describe and *Simulate*
- HDLs can be tricky because:
  - We are used to *Sequential Execution*
  - Different “bugs” Occur due to *Parallel Behavior*
  - Parallel Behavior Modeled Using *Event Driven* Methods

# LOGIC SIMULATION

- Different Algorithmic Approaches
- Levelization
- Levelized Compiled Code
- Event-Driven

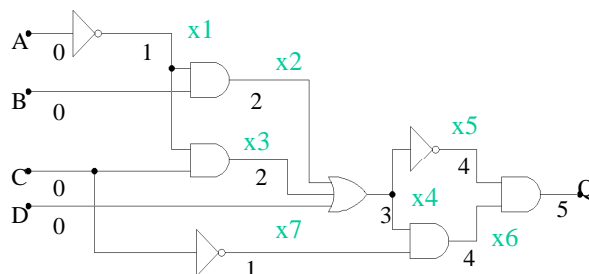
# LEVELIZATION EXAMPLE



## LEVELIZED COMPILED CODE (LCC)

- Levelize the Circuit (NETLIST)
- Sort Gates Into Ascending Order
- Generate Code (maybe C) for Each Gate
- Add I/O Routine
- Add Control Loop
- Compile the Code
- Run the Simulator

### LCC EXAMPLE



```

SimCkt0()
{
    x1 = ~A;           1
    x7 = ~C;           1
    x2 = x1 & B;       2
    x3 = x1 & C;       2
    x4 = x2 | x3 | D;  3
    x5 = ~x4;          4
    x6 = x4 & x7;      4
    Q = x5 & x6;      5
}
LEVEL

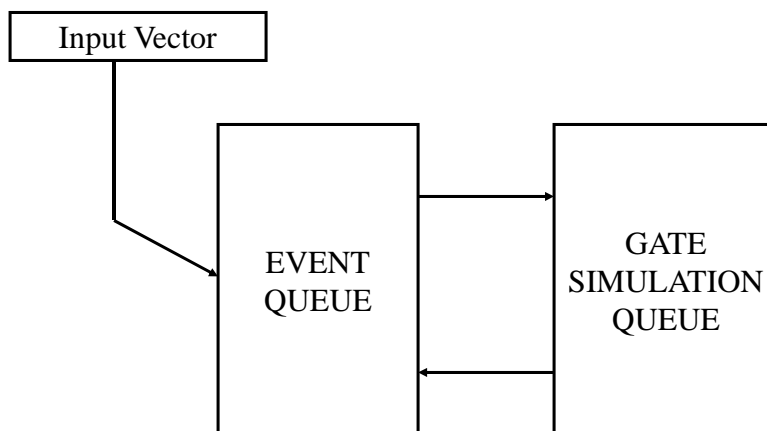
```

- 0-delay simulation
- very fast

## EVENT DRIVEN SIMULATION

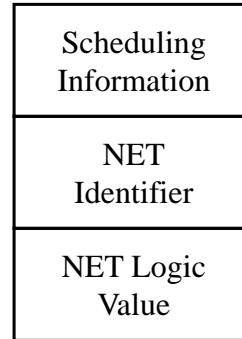
- Test Input Vector For Changes
- A Change is an EVENT
- Schedule Events for Primary Inputs
- Repeat Scheduling Until Finished
  - Process Each Event (Change in a net Value)
  - Schedule Gate Simulations
  - Simulate All (in no particular order)
  - Check for New Events

## ED SIMULATOR STRUCTURE

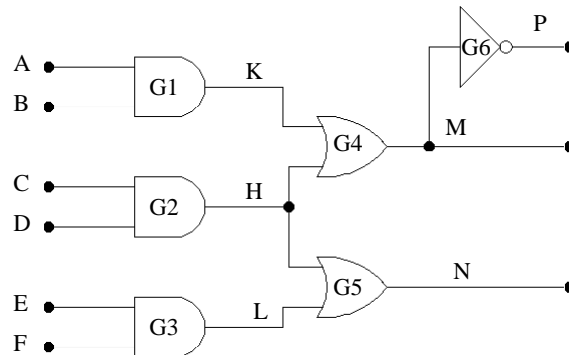


## ED Versus LCC Simulation

- LCC – Each Gate is Simulated Once Per Input Vector
- ED – Eliminates Unnecessary Simulations
  - Only Simulates Gates with Events on an Input
- Event – Change in a NET Value
  - Each Net has a Data Structure
- ED Simulator
  - Detects Events
  - Schedules the Simulations in Response
- Dynamic Scheduling
- Test For Event When
  - New Input Vector
  - Immediately After Simulating a Gate

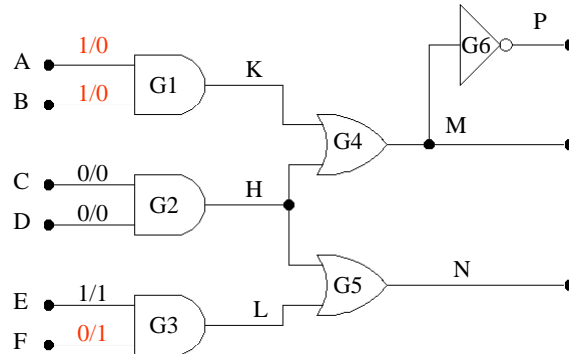


## ED SIMULATION EXAMPLE



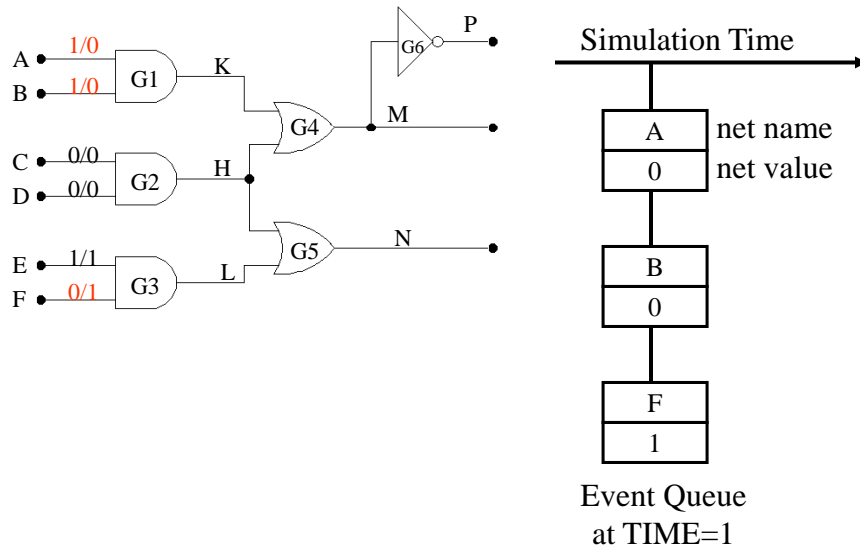
Suppose Input Test Vector Changes From  
(1, 1, 0, 0, 1, 0) → (0, 0, 0, 0, 1, 1)

## ED SIMULATION EXAMPLE

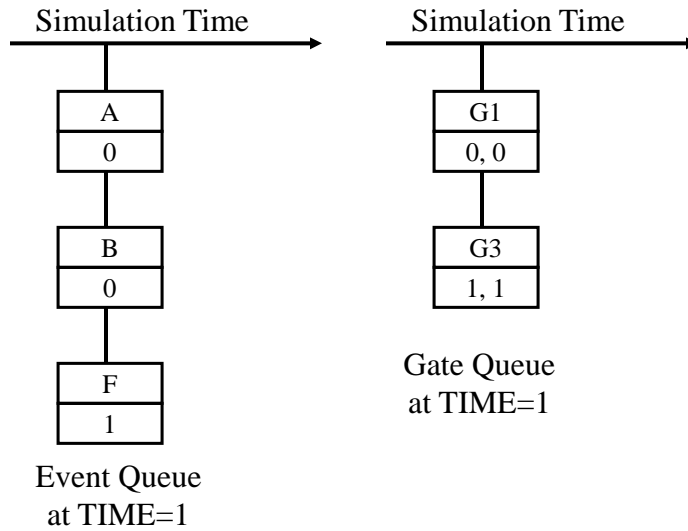


Suppose Input Test Vector Changes From  
 $(1, 1, 0, 0, 1, 0) \rightarrow (0, 0, 0, 0, 1, 1)$   
 Events are in **Red**

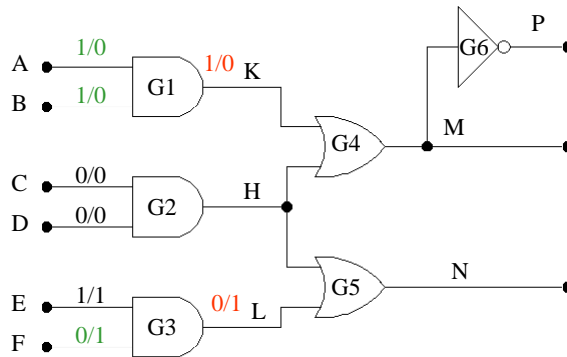
## The EVENT QUEUE at TIME=1



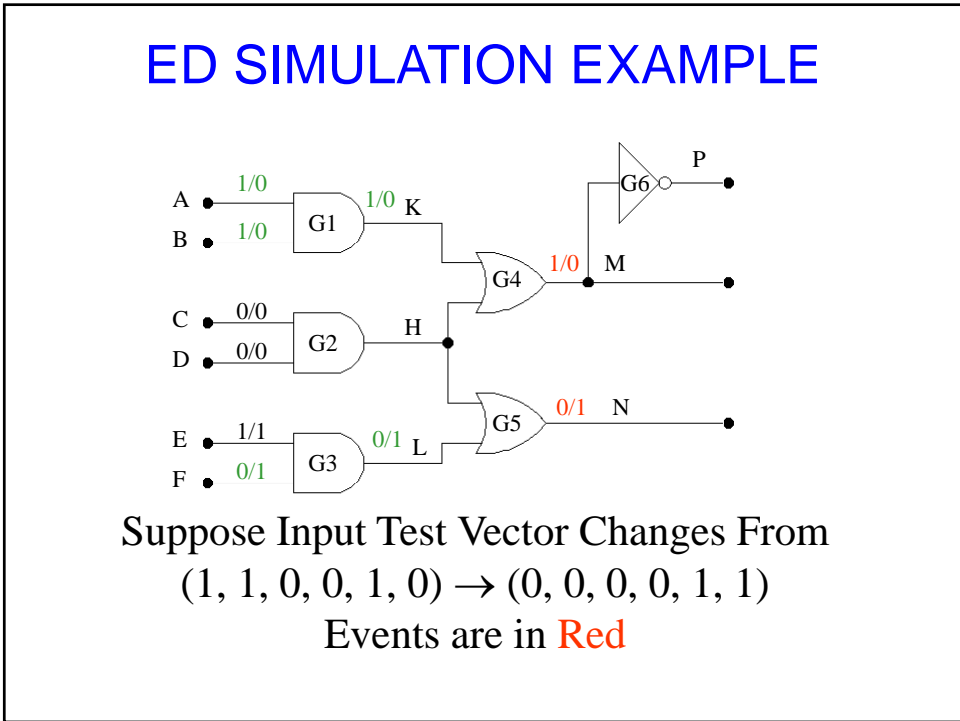
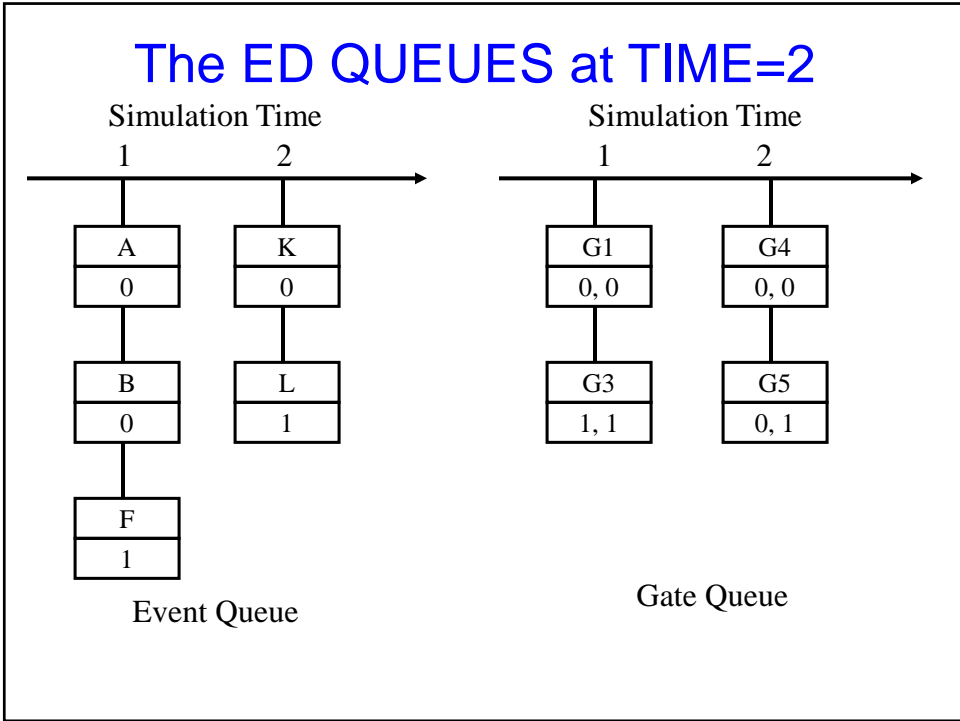
## The ED QUEUES at TIME=1

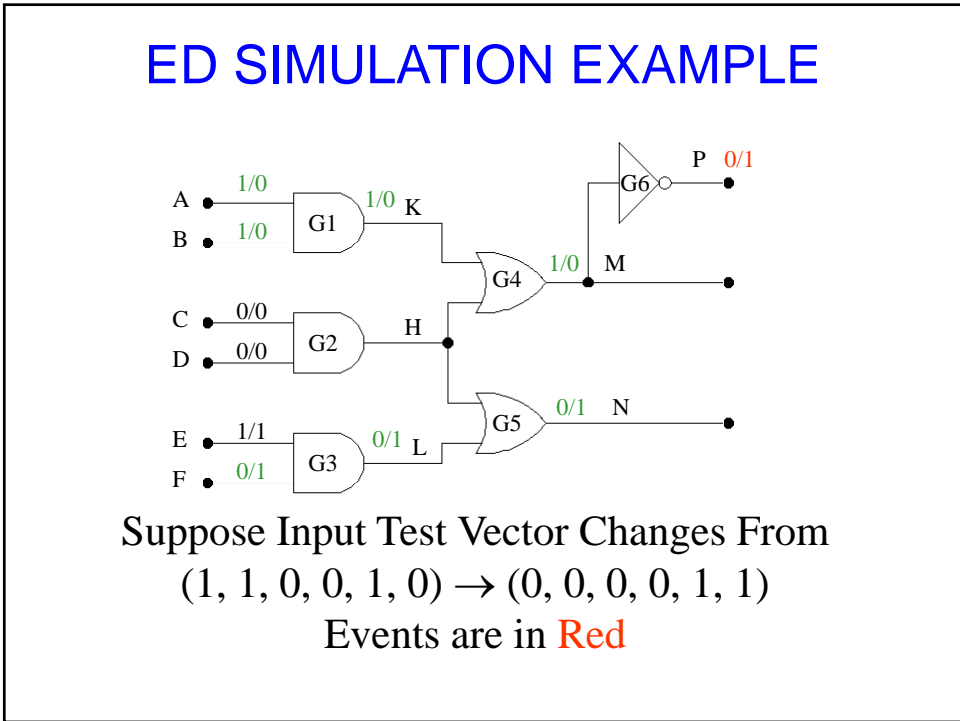
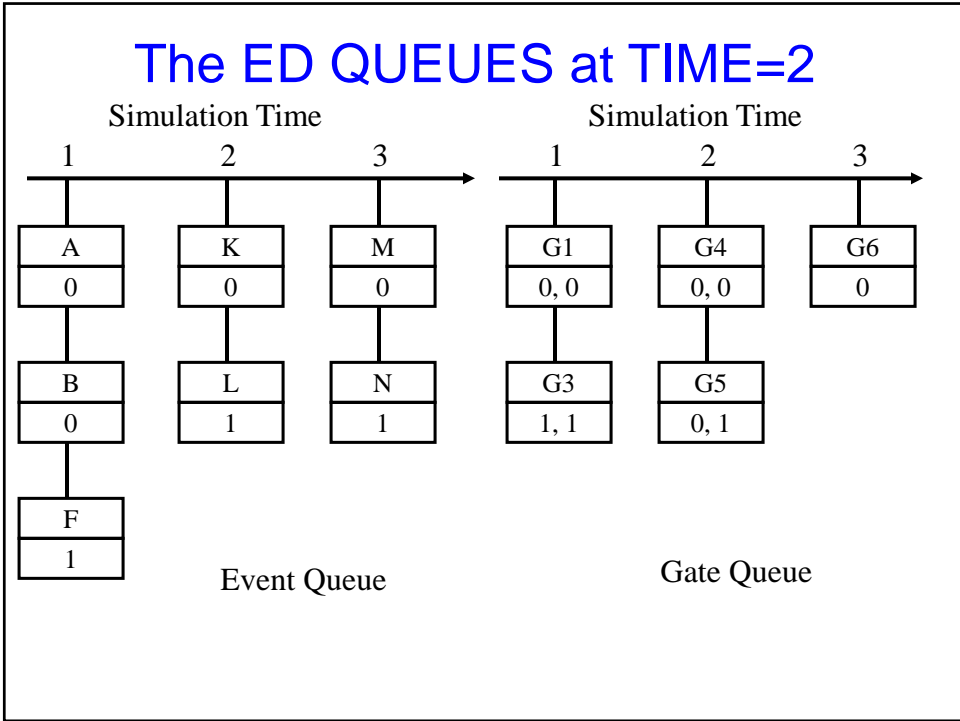


## ED SIMULATION EXAMPLE



Suppose Input Test Vector Changes From  
 $(1, 1, 0, 0, 1, 0) \rightarrow (0, 0, 0, 0, 1, 1)$   
 Events are in **Red**





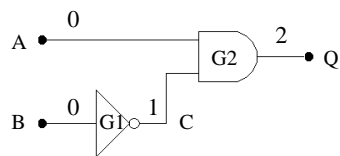
## ED SIMULATION COMMENTS

- ORDER In Which Gates Are Simulated  
(at a Given Time-Epoch) DOES NOT MATTER!
  - The Event Queue is Processed Causing the Gate Queue to be Filled
  - The Simulator Alternately Processes the Event then the Gate Queue
1. Process All Events in Queue
  2. Simulate All Gates in Gate Queue
  3. If Queues Not Empty, GO TO Step 1

## TIMING

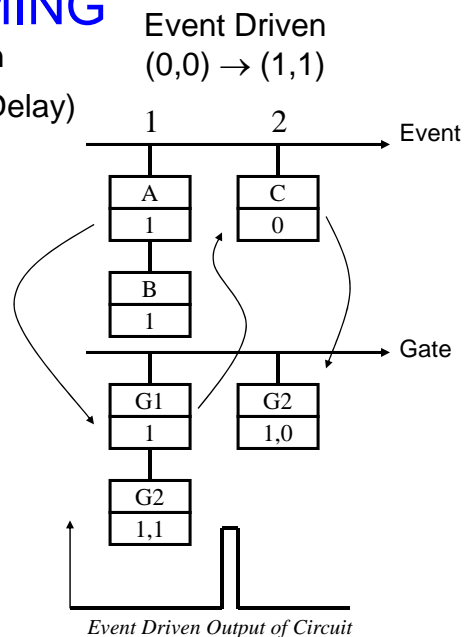
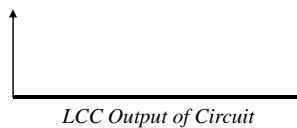
- LCC is 0-Delay Simulation
- Basic ED is 1-Delay (Unit Delay)

### EXAMPLE



```

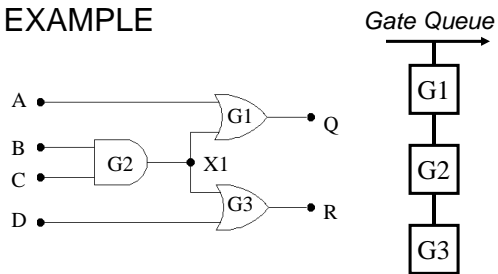
simCkt() {
  C = ~B;
  Q = C & A; }
  
```



## TIMING

- Gate G2 was Simulated Twice!
- At Time=1 and Time=2
- 2 Events for Net Q
- ED Simulators Can Show Glitches (Hazards/Races)
- Intermediate Values are Stored in Event Queue Instead of Netlist Itself

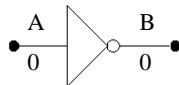
### EXAMPLE



- G2 after G1, but if G2 causes X1 Event, G1 and G3 sims use a different value!
- X1 must be held constant until all entries in Gate Queue are simulated

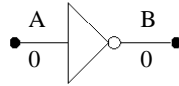
## ED SIMULATOR INITIALIZATION

- So Far, Considered Events (NET changes) Only
- How do we initialize for the First set of Inputs?
- Set Every Internal Net to Zero?



## ED SIMULATOR INITIALIZATION

- So Far, Considered Events (NET changes) Only
- Cannot Initialize All NETS to 0 Initially!



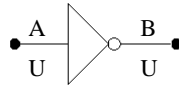
- Commonly Used Method is Multiple-Valued Logic

EXAMPLE: 3-Valued (ternary) Logic

Use {0, 1, U} U is Unknown

(VHDL/Verilog uses "X" for "U" – Not a Don't Care!!!!)

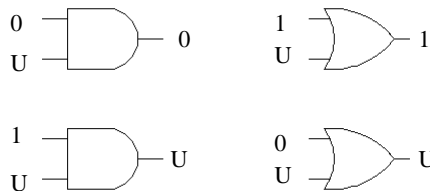
Initialize All Nets to U



- Event Occurs on B Whether A is 0 or 1

## Unknown Values – X in HDLs

- X (or U) means Simulator DOES NOT KNOW the Logic Value
- Don't Cares are Assigned to 0 or 1 in REAL Circuits
- Sometimes a Simulator Can Schedule a Gate for Simulation with an Unknown Input



- Unknown Values in HDL Simulations are Usually a Sign of Trouble With Your Design!!!!!!

## Summary

- Basic Ideas in Logic Circuit Simulation
  - Levelization
  - Event Driven
  - Zero-Delay, Unit-Delay
  - Unknown Values
- HDL Bugs Can be Hard to Find
  - Event-Driven Execution Rather than Sequential
  - Things can happen at (apparently) the Same Time
  - Hardware Operates in Parallel not Sequential Manner
- Knowing how Simulators Work can Help Debug HDL
- Verilog is used For:
  - Circuit Simulation
  - Circuit Synthesis