# Project 1
# ECE 6140: Digital System Testing

## Data Structures for circuit and simulation

1. Node: class to hold the details of a node in the circuit
    a. Files: Node.h and Node.cpp
    b. Data values:
        i. “name”: Name of the node
        ii. “value”: Value of the node (default value of -1 to represented uninitialized nodes)
2. Gate: class to hold the details of a gate in the circuit
    a. Files: Gate.h and Gate.cpp
    b. Data vales:
        i. “input1”: Pointer to the input node 1 of type “Node”
        ii. “input2”: Pointer to the input node 2 of type “Node”. For single input gates, checks have been added to ensure this node cannot be accessed.
        iii. “output”: Pointer to the output node of type “Node”
        iv. “logic”: Enumerated class to hold the logic type of the gate.
        v. “simulationDone”: Flag to show whether the gate has been simulated or not. This helps avoid re-simulations of the circuit and in-turn reducing runtime of the simulator.
3. Circuit: class to hold the details of the complete circuit
    a. Files: Circuit.h and Circuit.cpp
    b. Data vales:
        i. “node_map”: Map to connect the node name to the “Node” object for the node
        ii. “node_to_gate_map”: Map to connect the node name to the list of “Gate” objects for the node
        iii. “node_to_fanout_map”: Map to connect the node name to the newly created fanout branches
        iv. “inputNodes”: list to hold the list of input nodes
        v. “outputNodes”: list to hold the list of output nodes
        vi. “nextNodeName”: integer of the next available free node. Used to uniquely name the fanout branches when fanout branches resolved.

## Logic for creating circuit data structure from netlist

1. Circuit object is instantiated
2. Netlist is read line by line into the circuit object using “read_netlist”
3. For each line
    a. First part used to identify gate and create “Gate” object
    b. Based on gate type, it reads the required number of inputs and creates “Node” objects
    c. Last element is used to create the output and its “Node” object
4. Circuit class adds the required entries to all its maps during netlist parsing
5. For circuit input list, it adds the nodes to the “inputNodes” list
6. For circuit output list, it adds the nodes to the “outputNodes” list

## Logic for creating fanout branches (Extra Credit Part)

1. For each node, it identifies whether the node has fanout by checking if it is connected to more than one gate.
2. If it has fanout, then for each gate:
    a. It creates a new node using the “nextNodeName” and replaces the stem node with the node as the gate input.
    b. Adds the new node as an entry into the “node_to_fanout_map”
    c. Removes the connection from the stem node to the gate

3. Finally, the stem node will no longer be connected to any gates as it will have connections via the fan out nodes

## Logic for simulating the circuit

1. Test vectors are applied using the “apply_test_vector” function
2. Updates the boolean value at all the input node objects. Maintains a “readyNodes” list to track the list of nodes which have received a valid Boolean value but have not been applied to the gates yet.
3. Uses a while loop to iterate over the “readyNodes” list:
    a. Identifies if the node is a stem of fanout branch using the “node_to_fanout_map”. If it does, then propagates the boolean value to the fanout branches.
    b. Applies the boolean value of each node at the appropriate gates the node is connected to using the “node_to_gate_map”.
    c. For each gate, if all its input nodes have a valid Boolean value, then it simulates the gate and adds the output node of the gate into the “readyNodes” list.
    d. If all the gates connected to the node have been simulated, then it removes the node from the “readyNodes” list.
4. The while loop runs until all the gates and nodes have been simulated.
5. Finally, prints the output boolean vector in the order in which the output nodes are defined in the circuit netlist

## Steps to run script:

```
// Pseudo code for the simulation algorithm
function apply_test_vector(testVector):
    // List of nodes with valid values
    list readyNodes;
    // Add the value to each input node
    foreach value in testVector:
        node_map[node_name].value = value
        readyNodes.append(node_map[node_name])
    // Iterate through all ready nodes till all nodes in circuit simulated
    while (readyNodes.size() > 0):
        currentNode = readyNodes.top()
        // Check if fanout stem
        if (currentNode == fanout_node):
            // Apply the stem nodes value to each branch
            foreach branch_node in node_to_fanout_map[node_name]:
                branch_node.value = currentNode.value
                // Add to ready nodes
                readyNodes.append(branch_node)

        // Check for connected gates
        foreach gate in node_to_gate_map[node_name]:
            if (gate.has_valid_inputs()):
                gate.simulate()
                if (gate.simulated()):
                    // Add the output of this gate as a readyNode
                    readyNodes.append(gate.output)

        // Check if all connected gates simulated
        if (all_connected_gates_simulated):
             readyNodes.remove(currentNode)
         else:
             // add it to the end to allow other nodes to simulate first
             readyNodes.move_to_end(currentNode)
```

1. Compile all the *.cpp and *.h files. Run either of the below commands.
    a. Command: g++ -std=c++11 *.cpp -o DSTProject1.out

   b. Command: g++ -std=c++11 DST_project1.cpp Circuit.cpp Gate.cpp Node.cpp -o DSTProject1.out
2. Input arguments:
   a. <path_to_netlist>: Path to input netlist
   b. <input_vector>: Input vectors to apply
   c. (Optional) <flag_to_fanout>: 0 or 1 to disable/enable the fanout branching
3. Run command: <exe> <path_to_netlist> <input_vector> <flag_to_fanout>
4. Example commands
   a. Default run: ./DSTProject1.out ./s27.txt 1110101
   b. Run with fanout branching: ./DSTProject1.out ./s27.txt 1110101 1
5. Outputs:
   a. Console output: "Output Vector is <output_vector>"
      i. NOTE: Output ordering same as the one in the netlist.
   b. If fanout branching activated using input argument, then modified netlist with fanout connections and updated gate connections in file "modifiedNetlist.txt"

## Tested on

1. Windows 10 machine
2. Gatech ECE Server (unix): ece-linlabsrv01.ece.gatech.edu

## Simulation Data:

| Circuit | Input Vector | Output Vector |
|---|---|---|
| s27.txt | 1110101 | 1001 |
| | 0001010 | 0100 |
| | 1010101 | 1001 |
| | 0110111 | 0001 |
| | 1010001 | 1001 |
| s298f_2.txt | 10101010101010101 | 00000010101000111000 |
| | 01011110000000111 | 00000000011000001000 |
| | 11111000001111000 | 00000000001111010010 |
| | 11100001110001100 | 00000000100100100101 |
| | 01111011110000000 | 11111011110000101101 |
| s344f_2.txt | 10101010101010101111111 | 10101010101010101010101101 |
| | 01011110000000111000000 | 00011110000000100001111100 |
| | 11111000001111000111111 | 00011100000111011000111010 |
| | 11100001110001100000000 | 00001101111001111111000010 |
| | 01111011110000000111111 | 10011101111100000100100100 |
| s349f_2.txt | 10101010101010101111111 | 10101010101010101101010101 |
| | 01011110000000111000000 | 00011110000000101011110000 |
| | 11111000001111000111111 | 00011100000111010001111100 |
| | 11100001110001100000000 | 00001101111001110010001111 |
| | 01111011110000000111111 | 10011101111100001010000100 |