Term Project: Overview

- Overview
  1. You need to read in a circuit description
     - Consists of list of gates with their delay and what wires they connect to. Examples.
     - With that description, you'll create an internal (or in-memory) data structure which represents the circuit
       - E.g., if Wire #4 is an input to a NAND Gate, then Wire #4 will have a pointer to NAND Gate
  2. Read in a set of input tests, called the input vector, to initialize the simulation queue.
     - The input vector is simply the set of initial test conditions to exercise
       - E.g., INPUT A 4 1 means at time 4, set input A to 1
  3. Conduct simulation
     - Cause input values to propagate to outputs
  4. Visualize (i.e., print) results in a console window or with wxWidgets GUI

Term Project: Wire States

- A wire can take on any one of three values
  - 0, 1, X
  - X means that the value is unknown
    - This is different from Z, which occurs if a gate is tri-stated (i.e., some capacitive value between 0 and 1)
    - We assume gates will produce 0, 1 or X depending on the three-valued logic outlined in the HW write-up.
- At time=0, all wires are set to the unknown the value X
- If an input changes, it will force the associated wire to that particular value
  - NOTE: an input could be changed to have value X

Term Project: Key Application Classes

- When parsing the circuit, we need to represent it as a data structure
- Ideally, the data structure will support the operations we need
  - What operations do we need?
    - When a Wire changes value, we need to notify to all the Gates which have this Wire as an input so that they can (if necessary) recalculate their output value.
    - When a Gate recalculates, if its output (after the gate delay time) will change, then it needs to notify its output Wire by scheduling a value change Event on the simulation Queue.
  - So, Gates and Wires need to know of their interconnections
  - We also need Events and a priority Queue to sequence the simulation

Term Project: General Class Structure

- The most straightforward data structures would be for the Gate and Wire classes to contain pointers to one another
- We should be able to create a data structure which reflects the interconnections in the circuit
- Problem! Both classes refer to one another. Who gets included first? Answer: Use a forward declaration (in bold below).

```
class Wire;

class Gate {
    …
    private:
        Wire *in1, *in2;
        Wire *out;
}
```

```
class Gate;

class Wire {
    …
    private:
        …
        vector<Gate *> out;
}
```

Term Project: Parsing the Circuit File

- So, how do we create and connect these Gates and Wires?
  - We will be parsing an input file that has lines like:

    NAND 5ns 2 4 6
  - When we parse the keyword NAND, we know we need to create a new Gate of type NAND
    - From the rest of the Gate description line, we know that the NAND Gate has a 5 nanosecond delay and inputs on Wires 2 and 4 and output on Wire 6
    - NOTE: when you parse the Gate description, you'll need to get rid of the "ns" on the delay and convert the preceding number to an integer
    - NOTE: the three integers are Wire numbers (see the next slide for suggestions on how to handle these)

Term Project: Parsing the Circuit File

- Parsing Wire numbers
  - When we encounter a Wire number in the circuit file, this may/may not be the first time that number has appeared
  - Consider Wire #2 from NAND 5ns 2 4 6
    - If the Wire #2 has not been seen before, we want to create a new Wire object, and somehow associate it with the #2
      - One design is to give Wire an attribute wireNum
      - A better design is to keep track of all the Wires by an array (or even better, a vector), and let wireArray[2] store a pointer to the Wire #2
      - Final Process:
        - If necessary, create Wire objects with associated pointers in wireArray[2], [4] and [6]
        - Associate the new NAND Gate to these Wires.
        - Associate the input Wires to Gate they effect.

Term Project: Parsing the Circuit File (Final Details)

- Consider Wire #2 from NAND 5ns 2 4 6 (cont)
  - If the Wire #2 has already been seen before, wireArray[2] won't be NULL (because we will have stored its reference), and we can just connect this existing Wire to the associated Gate and the newly-created Gate to this Wire.

- So, as we parse a line, we create a Gate and any necessary Wires, and hook them up in the data structure!

Term Project: Simulate Circuit/Parse the Vector File

- Consider the simulation: we can accomplish it in real time or as event driven
  - Real-Time (Clock Driven)
    - A.K.A human-in-the-loop (HITL) simulation
    - Sees the simulation as advancing with each "tick" of a clock
    - Harder to implement and more costly in terms of execution time. DO NOT USE THIS METHOD!
  - Event-Driven (Priority Queue Driven)
    - Events occur at specific times
    - Current Events generate future (queued) Events
    - Easier to implement and less costly than the real-time method
- Assuming an event-driven simulation, the vector file represents the initial simulation conditions (i.e., the initial Events in the simulation Queue)
    - Specifically, we parse the vector file and for each INPUT PAD VALUE DEFINITION we insert a corresponding Event in the Queue
    - Once the vector file is parsed and the Queue is populated with its initial Events, we then simulate by:
      - Pop the top Event e concerning Wire w1 from the Queue
      - Determine if e causes a Gate g to change output Wire w2's value.
      - If g's output w2 will change, then schedule a future Event for w2
      - Apply the effects of e to w1.
- To store Events build a Queue based on a priority queue (i.e., from the STL)

Term Project: Review

- Big picture
  1. Parse circuit file to create in-memory data structure of Gates and Wires to simulate
  2. Parse the vector file to initialize the simulation Queue with initial Wire state (i.e., value) changes
  3. Simulate the circuit using Event-driven control by
     - Removing the top Event e in the Queue
     - Determining if e causes a future Wire state change
       – Create and queue any future Wire state changes as new Events
     - Apply e's effects
  4. Print the results of the simulation