CS1220 – C++ Programming

Homework Assignment #8: Term Project
Due: 4/28/11
Points: 200

Name: Zach Schneider

I. Requirements: Restate the problem specification, and any detailed requirements

For this project, we were required to create a miniature digital logic simulator. This simulator needed to input two files, a circuit file and a vector file, and output the timing diagram of these files for sixty seconds or until nothing further changed. For error handling, the program needed to nicely handle nonexistent input files as well as malformations in the input file. However, the input file did not necessarily need to make sense; that is the user's role, not the program's.

II. Design: How did you attack the problem? What choices did you make in your design, and why? Show class diagrams for more complex designs.

I began attacking this problem by implementing classes, starting with Gate and Wire. I carefully considered what attributes each class would require and how they would need to be mutated as the program executed. I then implemented the input file opening and parsing. For the simulation itself, I created an Event class (containing a time, a wire, and what value the wire should go to). I then created a priority queue of Events, such that the program would cycle through time 0 to 60 and check the events occurring at each time. Next, I backtraced and gave the Gate update function the ability to queue new events, so it could queue a change in the output wire after its gate delay. Finally, I tested the program and worked out various errors (which were mostly with my pointer usage). At the end, I realized that my parsing did not allow for extra white space in the circuit files, so I updated my parsing to allow for this.

III. Implementation: Outline any interesting implementation details.

Instead of creating subclasses that inherited Gate for my and, or, etc. gates, I simply gave Gate another attribute: int type. The value of the type corresponded to what kind of gate it was (as explained in the program comments). This allowed me to save files; when I was updating the gate, I just used a switch statement with the type to perform the correct update function.
Also, rather than limiting the user to 60 cycles or whenever the program stops, I chose to allow the user to enter the number of cycles they wished to simulate. If the simulation was abridged since no further events existed after a time, my program provides a note informing the user that the simulation was stopped after __ cycles because of lack of further activity.

IV. Testing: Explain how you tested your program. Explain why your test set was sufficient to believe that the software is working properly, i.e., what were the range of possibilities of errors that you were testing for.

I tested my program using Dr. Shomper's two test circuits (crct and flipflop), as well as several challenging circuits of my own design. I believe that my test set was sufficient because I provided multiple examples of gates having delays generated at the same time and having events come to fruition at the same time. Also, Dr. Shomper's crct.txt test program tests the range of functions the program needs.

V. Summary/Conclusion: Present your results. Did it work properly? Are there any limitations? If it is an analysis-type project, this section may be significantly longer than for a simple implementation-type project.

My program works properly. I was able to find no limitations in its function. Every circuit that I created and tested works correctly.

VI. Lessons Learned: List any lessons learned. What might you have done differently if you were going to attack this again.

I believe that my strategy for this program worked very well. The program took me around twenty hours to complete, which I expected, but my strategy led to a successful program and I understand its inner functions better than most assignments I have completed this semester.