**CS1220 – C++ Programming**

Homework Assignment: HW#6: Term Project

Due: See [course web site](http://people.cedarville.edu/Employee/kshomper/cs1220_web/schedule.htm) for due date

# Points: 150

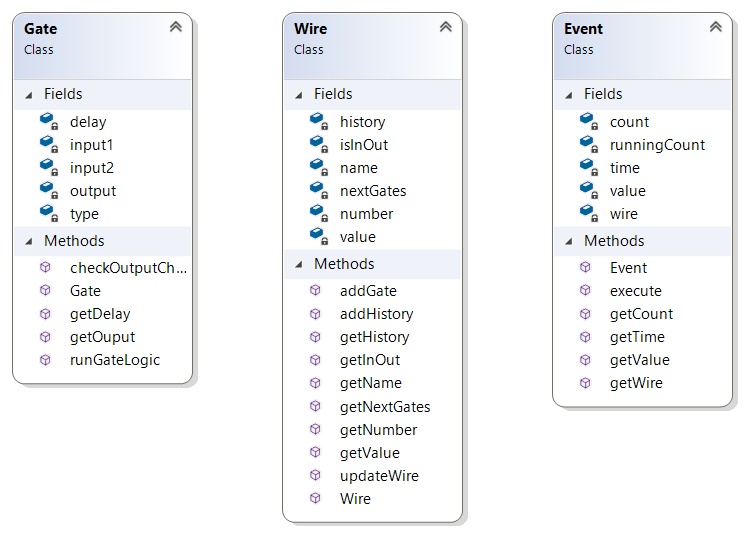
**Name: Joshua Kortje and Nathan O’Neel**

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

This assignment required us to program a circuit simulator that would be able to run basic digital logic circuits involving 2-input gates with gate delays. This involved parsing both a circuit file and a vector file. The circuit file contained information about gates, inputs, outputs, and wires. The vector files held a list of events that would need to be executed at runtime to change the input wire values. At runtime, the program had to simulate the events in the proper order and save the values in the inputs and outputs as time went on. This had to continue until either the outputs stopped changing or the time reached 60 nanoseconds. The program also had to handle 3 value logic involving high, low, and unknown values. Once the circuit was done simulating, the results then needed to be printed out as a waveform.

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.

We started by getting the classes all figured out. Namely, we coded everything we thought we would need for the Gate, Wire, and Event classes. Initially, we had resorted to using Input and Output classes as well but those ended up being more trouble than they were worth, so they were eliminated to simplify the code. The next step was to figure out the parsing of the circuit and vector files. These were wrapped up into functions to simplify the main program. The printing function came next so as to provide an easier way to test the program. This was also put in its own function. The last part was to make the actual runtime logic. Along the way we would make occasional changes to the classes as needed. The class diagrams for all of the classes we used is shown below.



III. Implementation: Outline any interesting implementation details.

The runtime logic was the most different from our previous programming experiences. This was done as an event driven program with a priority queue of Events. Within a single while() loop, one event could be handled and then we were able to check for any new events that had to be created, putting those into the priority queue. This kind of approach greatly simplified the runtime code. Another interesting part of the code was the way the Gate and Wire classes each had pointer to the other class. This allowed for events to spawn a sort of chain of events and allowed for each Gate or Wire to easily access any other relevant objects.

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.

For testing, we relied heavily on the provided circuit files. We first drew out the waveforms by had and then let the circuit run them, checking the circuit against our own work. We believe this was a thorough test of our program because by looking through the test cases we saw that it included any kind of odd situation that our code might be required to handle. These types of tests included feedback, metastability, missing files and other various things. When our code was able to successfully pass all of the test cases, we were satisfied with its functionality.

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.

Yes, our program works correctly. The only limitations that is has it that it does no error checking for the circuit and event files. If they don’t exist it will exit the program. If they do exist and are formatted incorrectly, then the program will do its best to interpret it, but depending on the errors in the file it may crash.

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

One thing that we saw very quickly was that the idea of having Input and Output classes, while sounding nice at first, actually only made things more complicated later. This was very instructional in showing how preferable it is to have a simple solution, and how necessary it is to make sure you do not make to problem more complicated than it has to be. We also realized the importance of commenting on each important line of code. Since we didn’t meet together much, we had to read each other’s code and it made it a lot easier being able to see what each line doing (or trying to do) from the comments made by the other person.

**CS1220 – C++ Programming**

Homework Assignment: Term Project

**Objective**

Create a circuit simulator which reads a circuit definition (as described below) and an input vector definition (also described below) and simulates the operation of the circuit over time (up to 60 time steps of simulation time or until the circuit output no longer changes).

**Description**

The purpose of this assignment is to pull together in a single object-oriented and \*optionally\* GUI-based assignment all of the characteristics of the C++ language which you’ve learned in this class: file I/O, pointers, classes, containers, inheritance, and polymorphism.

*Be aware, for most students this project is significantly more difficult than previous assignments. It is in your best interest to begin the assignment early; otherwise, you may not complete it on time—if at all.*

*Detailed Problem Statement*: Develop a digital simulator which reads a circuit description and input vector from files (formats described below) and performs the digital simulation based on these definitions. The simulation is to be visualized using a console window or \*optionally\* a GUI via wxWidgets.

* *Circuit File Format*: In general, a circuit definition has the following format:

CIRCUIT HEADER

INPUT PAD DEFINITIONS (as many as necessary)

OUTPUT PAD DEFINITIONS (as many as necessary)

GATE DEFINITIONS (as many as necessary)

Where:

1. The CIRCUIT HEADER consists of the keyword “CIRCUIT” and a circuit name. You may use this name to label the circuit, or simple ignore the line. For example: *CIRCUIT Circuit1*
2. The CIRCUIT HEADER will be followed by an unspecified number of INPUT PAD DEFINITIONS. An INPUT PAD DEFINITION will consist of the keyword “INPUT” followed by a name label and a wire number. For example, the following line denotes that input pad “A” is associated with wire number two: *INPUT A 2*

NOTE: although the [example circuit files](http://people.cedarville.edu/Employee/kshomper/cs1220_web/lecture_notes/definition_files.htm) use single-letter names, your simulator should accommodate names of more than one letter.

1. The INPUT PAD DEFINITIONS will be followed by an unspecified number of OUTPUT PAD DEFINITIONS. An OUTPUT PAD DEFINITION will have the same format as an INPUT PAD DEFINITION, except the keyword will be “OUTPUT” rather than INPUT”. For example, the following line denotes that output pad “E” is associated with wire number six: *OUTPUT E 6*
2. The OUTPUT PAD DEFINITIONS will be followed by an unspecified number of GATE DEFINITIONS. A GATE DEFINITION consists of the gate type (one of “NOT”, “AND”, “OR”, “XOR”, “NAND”, “NOR”, and “XNOR”) followed by an integer delay value with its nanosecond units, followed by the input wire numbers (two input wires for all gates, except a NOT gate which uses only one), followed finally by an output wire number: For example, the following line defines an AND gate with a 5 nanosecond delay and having wire 1 and 2 for input and wire 4 for output: *AND 5ns 1 2 4*

**Result:** Your program should read the circuit file and produce an in-memory representation of the circuit which can be simulated using the information from the vector file (see below) as the initial starting conditions.

* *Vector File Format*: An input vector definition has the following format:

VECTOR HEADER

INPUT PAD VALUE DEFINITIONS (as many as necessary)

Where:

1. The VECTOR HEADER consists of the keyword “VECTOR” and a vector name. You may use this name to label the simulation output, or you may simply ignore the line. For example: *VECTOR vector1*
2. The VECTOR HEADER will be followed by an unspecified number of INPUT PAD VALUE DEFINITIONS. An INPUT PAD VALUE DEFINITION consists of the keyword “INPUT” followed by a name label, followed by a time stamp at which the wire associated with the name value changes its value, and then by the value to which the wire changes. For example, the line below indicates that input A changes value at time 0 to a value of 1: *INPUT A 0 1*

**Result:** Your program should read the vector file and initialize a priority queue of events (one event for each INPUT PAD VALUE DEFINITION).

*Three-valued Digital Logic:* The wires in our digital circuit can take on 3 values: 0, 1, and X (undefined). The outputs pads and all gate outputs should be initialized to X at time zero. The truth values for three-valued AND and OR operations appears below, from these tables you should be able to determine the remainder of the gate operations.

Three-Value Truth Table

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | X AND Y | X OR Y |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | x | 0 | x |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | x | x | 1 |
| X | 0 | 0 | x |
| X | 1 | x | 1 |
| X | x | x | x |

*Interim Deliverables:* To encourage an early start on this assignment, you are required to turn in (as a minimum) a printed listing of your class specifications AND implementations for the **Gate** and **Wire** objects by the date shown on our course web site. The interim deliverable is worth 20% of the overall assignment. Please note, these classes are not all of the classes you will implement for your solution. An additional class which you need is Event. You may also consider classes for Input, Output, Circuit, and/or Simulation.

*Other Requirements*: Your program must perform basic error handling on the input files (however, exception handling is not required). For example, you should gracefully handle “File not found” and detect format errors in input files. Any ill-formed input lines should simply be ignored. You do not have to check that the circuit being input makes sense; e.g., you don’t have to test that circuit inputs can effect the outputs, etc.

Your program should provide a record of the simulation by showing the input and output pad histories of the simulated circuit in a console window or by drawing on a wxWidgets canvas. If developing the \*optional\* GUI, please use a wxFileDialog to choose the circuit and vector files. NOTE: *you do* ***not*** *need to actually draw the black-box circuit, change the wire colors, or allow for resizing as demonstrated in the example program.*

*Sample Programs:* To get a better feel for the program, you can copy a working executables for both the console and GUI versions from:

S:\DEPT\EG\Computer Science\CS1220\Circuit Sim (Text Version)\text circuit sim.exe or S:\DEPT\EG\Computer Science\CS1220\Circuit Sim (GUI Version)\circuit sim gui.exe

If you copy the GUI version of the app to your own computer, please be sure to also copy associated button bitmaps (open.bmp, etc.)

*Sample Circuits:* Sample circuit and vector definitions are also provided in the aforementioned folders or on our course web [here](http://people.cedarville.edu/Employee/kshomper/cs1220_web/lecture_notes/definition_files.htm).

*Teamwork:* You may (and are encouraged to) work in teams of two persons. However, if you choose to work alone, you may.

*Output*: Your simulator should have the capability to display the input and output waveforms. The display can just use text characters in the console window to show the waveform. For example:

Input A XXXXX000000000001111111111111111111

Time 0 5 10

(or)

Input A xxxxxx\_\_\_\_\_\_\_\_\_-------------------------

Time 0 5 10

\*Optionally\* use wxWidgets to display the waveforms. Please select this option **only after** you have completed the basic requirements. You will be given 10% (15 points) extra credit for using wxWidgets of file selection and output (assuming the basic requirements are also met).

**Final Details**

Required for turn-in (NOTE differences in a. and b.):

1. The completed coversheet for this assignment. If you choose, your coversheet can be printed and handed in on the last day of class OR included as part of your electronic zipped submission.
2. Program listings of your application source code are NOT required.
3. Provide an electronic zipped submission of ALL your source files (including main program)via the “submit20” command. Please, do NOT send any other files in your zipped file other than .h and .cpp, or .docx files.
4. Certain working programs may be demonstrated in class on the due date by their developers. Please indicate if you have an interest in doing so.