Vance Piscitelli
CPE 493
Shamik Sengupta
November 23, 2015

eSquib Protocol Simulation

Introduction

During the previous semester, for my Senior Project, my team worked with the company DSSP to design a program that could be used to created customized sequences that could be applied to their new pyrotechnic that could be continuously fired via electrical current (see http://dsspropulsion.com/pyrotechnics/ for more details). During the course of the project, we worked with the company to create a protocol to communicate over a RS485 cable between a computer and their pyrotechnic system that was made up of multiple Arduino-controlled devices. There was no official protocol so we created one that worked but it was not necessarily the best option but it was not a priority at the time to improve it. By creating this simulation, I can create different protocols and test them out without actually having the physical hardware present which is both time-consuming to setup and transport and it is expensive.

Contribution

While my group had created a program in Java that could already simulate firing the custom sequences, it did not simulate sending the protocol to DSSP's hardware which makes up a universe. This was partially due to the fact that the protocol was not created until later into the project. Therefore, the first goal of my project is to simulate the protocol. It closely reenacts how the actual system would communicate between all the different devices, trying to simulate both accurate timing and how the data is actually transmitted. The program can create multiple types of packets that get sent between a “computer” and the “fireboxes” in the Universe via the RS485 cable that is full-duplex such that the computer can be sending packets to the fireboxes while the fireboxes are sending packets to the computer.

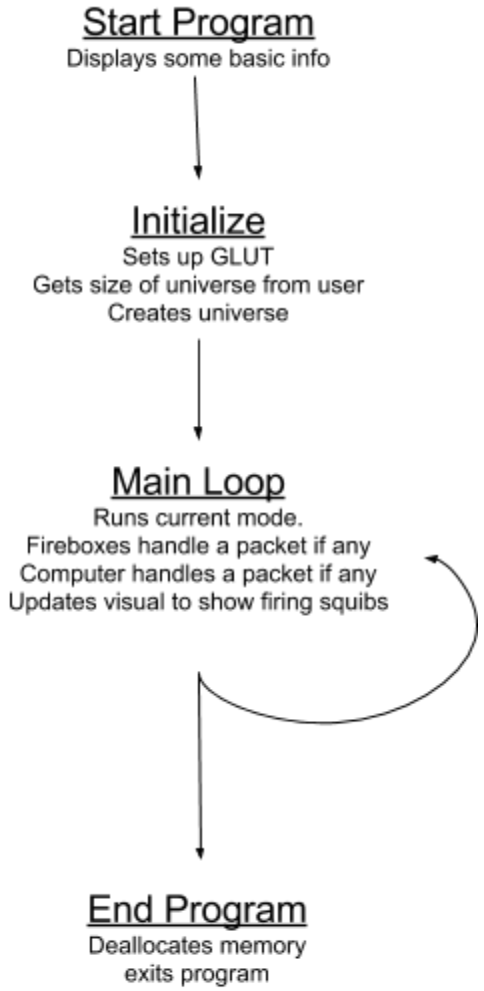
Code Explanation

This program was designed and tested to work on Ubuntu. The ECC virtual machines are able to run the program if an Ubuntu machine is needed.

OpenGl, GLUT, and GLEW must both be installed/working in order to compile and run this simulation.
GLUT and GLEW can be installed on Ubuntu with this command (Already installed on ECC machines)

```
>$ sudo apt-get install freeglut3-dev freeglut3 libglew1.6-dev
```

The program is kept simple to easily represent the actual system. After initializing the system based on input from the user, it enters a loop where packets are sent between the computer and the fireboxes. These packets can setup the universe, arm squibs, fire squibs, transfer error messages, and more.



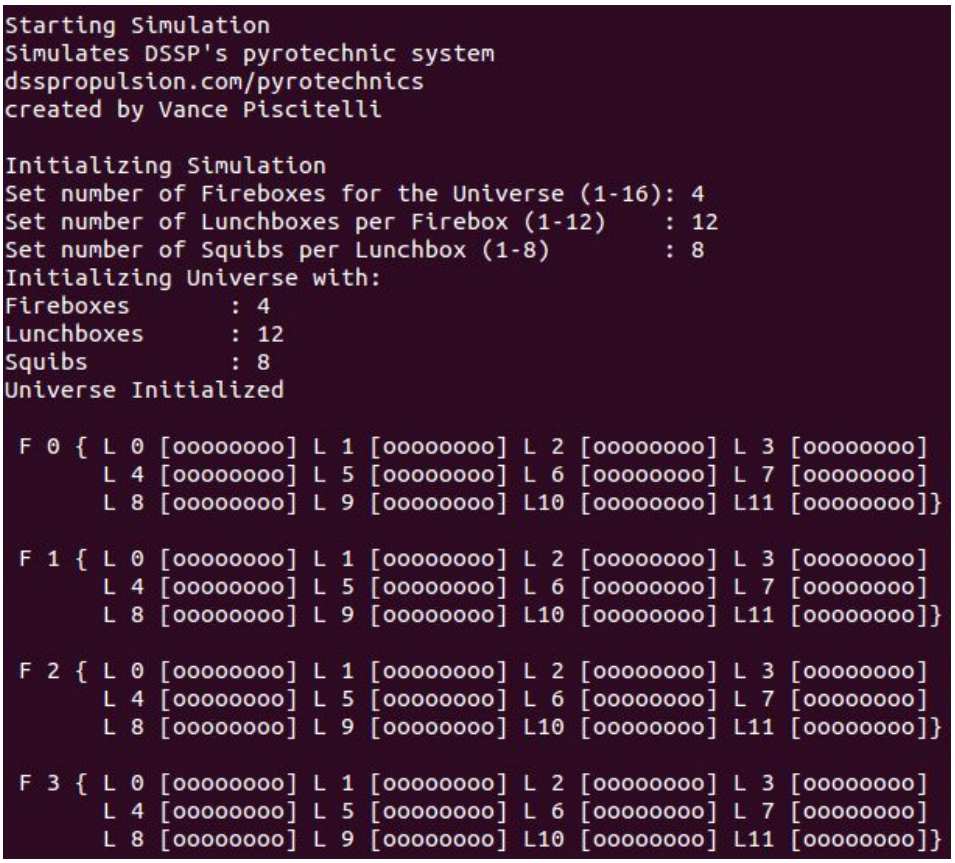
To follow how the actual system works, each part of the universe can only communicate with the objects adjacent to it. The main program only has access to the Universe which has access to all the fireboxes which have access to their own lunchboxes which each have access to their own squibs. For example, for the program to fire a squib, it must first assign addresses to the whole universe with some initializing packets. Then it must send an arm packet to the specified squib. The packet goes to the universe which is received by the fireboxes, the correct firebox takes the arm packet and then sends it to the correct lunchbox which then arms the correct squib. If the arm was successful, then a fire command can follow a similar path and fire the squib along with any other armed squibs. If there was an issue with a packet, the fireboxes can attempt to let the computer know by sending a response packet or an error packet back.

The following is just part of the function that simulates the fireboxes handling packets received from the computer. When a arm packet is received, it tries to arm the specified squib. However, the specified squib could potentially be bad so it will reply with an error packet indicating that the arm was not successful. At this point, the computer could just forget about arming the squib or try to find a solution to the problem such as by trying to arm a different squib on the same lunchbox. When a fire packet is received, all the squibs are checked and if they are already armed, they are fired and the program then handles the display to show the squibs firing.

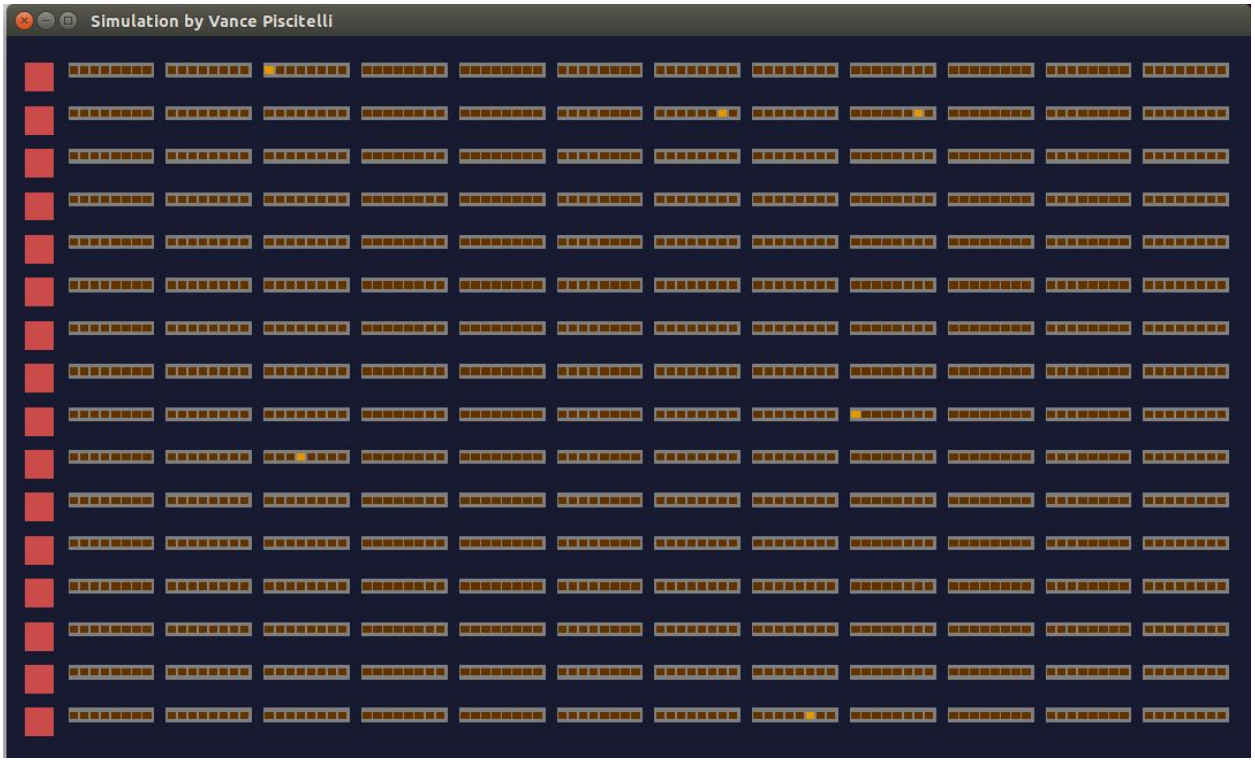
```
// the firebox-side packet handler
void fph(packet pack)
{
    else if(pack.type == 'f') // a fire packet. fires all armed & active squibs
    {
        fSquib temp;

        for(int i = 0; i < 16; i++)
        {
               for(int j = 0; j < 12; j++)
               {
                      for(int k = 0; k < 8; k++)
                      {
                             // squib will fire iff armed & active
                             universe.fireSquib(i, j, k);

                             if(universe.getFiring(i, j ,k))
                             {
                                    temp.fPos = i;
                                    temp.lPos = j;
                                    temp.sPos = k;

                                    firingSquibs.push_back(temp);
                             }
                      }
               }
        }
```

```
    }
    else if(pack.type == 'a') // arm packet
    {
        // if the specific squib is active
        if(universe.getActiveSquib(pack.fbAdd, pack.lbAdd, pack.sbAdd))
        {
               // arm specified squib
               universe.setArmedSquib(pack.fbAdd, pack.lbAdd, pack.sbAdd, true);

               // if squib not armed, send back an error packet
               if(!universe.getArmedSquib(pack.fbAdd, pack.lbAdd, pack.sbAdd))
               {
                      badSquib(pack.fbAdd, pack.lbAdd, pack.sbAdd, 'u');
               }
        }
    }

}
```

Before any actual packets are sent between the computer and the fireboxes, the simulator has to get some information from the user. The user just specifies a valid amount of fireboxes (16 max), lunchboxes (12 max per firebox), and squibs (8 max per lunchbox). a max amount of each means there could potentially be 1,536 uniquely addressable squibs. A sample initialization can be seen in the following screenshot which is taken before the actual visual is created.



Once initialization is finished, a new window appears showing the current configuration of the universe and will show squibs that are being fired. Note that the universe needs be assigned all of its addresses before packets will get to their proper locations.



Results

By running my simulation, I am able to see the packets being sent between the computer and the universe. As I was writing the code, I was able to change the protocol in order to make it run faster yet still keep the packet size from being too large. I was able to avoid common problems that a more complex, established protocol could have because there currently is not a lot of restrictions on this simple protocol. For example, I was able to create a new packet type to automatically set addresses to different fireboxes as opposed to relying on the addresses being manually set by the user. By improving the protocol, I was able to send more information in a shorter amount of time.

Conclusion and Future Direction

Overall, I got a good foundation to a program that could be further improved to have more packet types and to run multiple different protocols and compare them with the same set of data. My program can setup a semi-custom Universe configuration and then create a series of packets that sets up addresses, arms squibs, whether specified by the user or randomly selected, and can fire squibs. Additionally, the Fireboxes are capable of responding back to the computer which helps identify errors. Everything is also visualized to the user instead of just simply being text outputs in the terminal. However, with more time, I could add more functionality such as a more accurate representation of the communications between the fireboxes and the other devices in the universe (lunchboxes and eSquibs). I would also like to add more tools to analyze the efficiency of one set of protocols as compared to other protocols that I create.