IP Based Device Control

Divyam Kumar Mishra, Mrinmoy Kumar Das, Prince Kumar, Varjith Anchuri

[1] *Under Graduate Student, Department of Computer Science and Engineering, SRMIST, Chennai, India*
[2]*Under Graduate Student, Department of Computer Science and Engineering, SRMIST, Chennai, India*
[3] *Under Graduate Student, Department of Computer Science and Engineering, SRMIST, Chennai, India*
[4] *Under Graduate Student, Department of Computer Science and Engineering, SRMIST, Chennai, India*

ABSTRACT

The IP based device control kit is to be used to implement a home automation console that can be easily accessible from distant places through a simple web server running inside the home. This kit IP based device control details a design for controlling devices through network. By the implementation of the design, one can control various electrical devices, using the embedded web servers from a remote location through intranet which may be extended to internet. In the kit IP based device control through this model one can off/on the device, can read their present status (whether on/off. The protocol followed here is used for LAN network. The basic functionalities in this proposed system includes automatic control of Lights and other electrical / electronic appliances. Internet-enabled hardware products are slowly becoming popular. In this kit IP based device control, the device can be controlled from two different places. One from the server end itself and other from client end which is any place connected with the internet. The interfacing of the devices with the communication port has been done using Visual Basic 6.0 interface. The socket programming of the devices, which is used to control the device from the internet

Keyword : - *home automation, server end, client end*

1. Introduction

Homes of the 21st century will turn out to be increasingly self-controlled and computerized. Straightforward gadgets, for example, a clock to turn on one's espresso creator toward the beginning of the day have been around for a long time, yet considerably more advanced components will soon be predominant in homes the world over. Envision strolling into your home and being welcomed at the entryway with lights enlightening your way without you consistently touching a light switch, with your most loved music spilling through the speakers in whichever room you enter (on the grounds that your home perceived that it was you and not some other family unit part), all while having the significant serenity realizing that your home computerization framework dealt with actuating your security framework. Moreover, such a framework could enable the client to plan occasions to happen at repeating interims (e.g., turn on sprinkler framework at 4:30a.m. each Tuesday and Thursday).

This report portrays an estimation of such a home computerization framework, to the point that was composed and worked as a last undertaking for 6.111 at M.I.T. This framework was intended to be adaptable and by and large programmable, extensible to such an extent that including extra highlights is moderately straightforward, and measured and forward-good, so new parts can be included without upgrading the whole framework. To accomplish these objectives, the framework runs a client characterized program on an uncommon reason processor, utilizing certifiable sensor contributions as operands.

In Section 2, details of the design and implementation are thoroughly discussed. Section 3 presents result and discussion & finally section 5 concludes with future scopes.

2. Design and Implementation

With Internet Protocol as the backbone of an enterprise-wide network, everything that requires a standalone network today simply becomes subsumed into the IP master. This is interoperability at the very highest level. In the building management sector of this new environment, controllers are designed *for* the IP network. More importantly, they are also designed to thrive on the IP network.

Think of today's building management system as an engine that runs on IP power. This engine, in turn, fuels all of the controllers that link to it (wired or wireless). These controllers provide comfort, security, life safety, and other necessities of a desirable work environment.

The high-level engine, which we will call a Network Control Engine (NCE), combines the building management system's network supervisory capabilities and IP Ethernet connectivity with field equipment control that does all the work.
An ideal Network Control Engine environment, as depicted in Figure 1, offers several advantages over today's status quo.

2.1 System Architecture

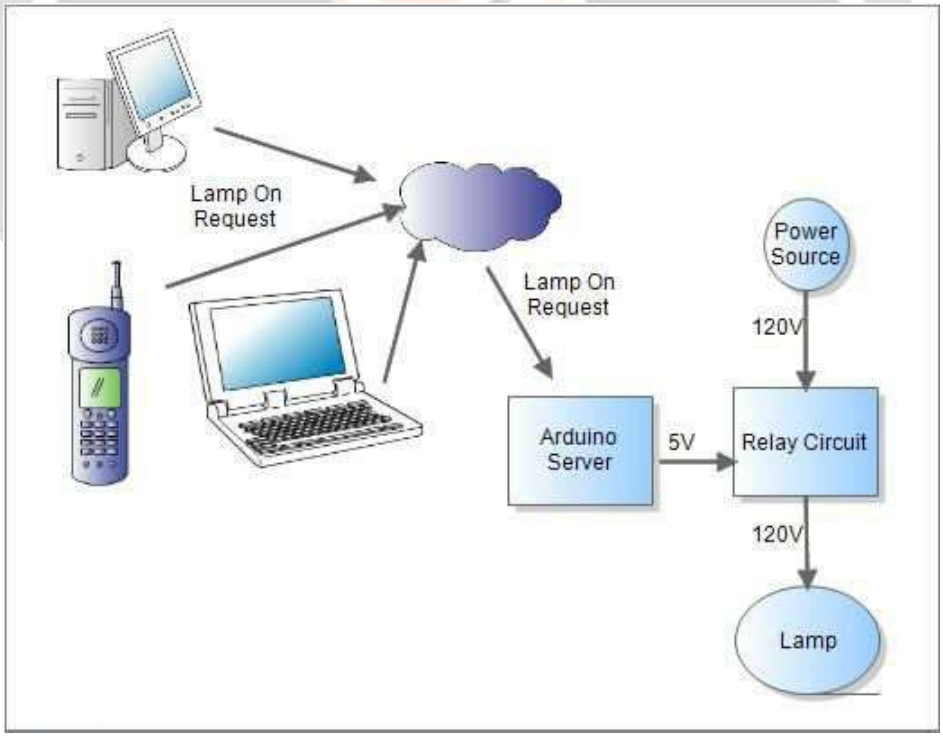

Fig -1: System Architecture

2.2 Hardware Requirements

Raspberry Pi 3 (Any other Version will be nice)

Memory card 8 or 16GB running Raspbian Jessie

5v Relays

2n222 transistors

Diodes

Jumper Wires

Connection Blocks

LEDs to test.

AC lamp to Test

Breadboard and jumper cables

220 or 100 ohms resistor

2.3 Software Requirements

Asides the Raspbian Jessie operating system running on the raspberry pi, we will also be using the WebIOPi frame work, notepad++ running on your PC and filezilla to copy files from the PC to the raspberry pi, especially the web app files.

Also you don't need to code in Python for this Home Automation Project, WebIOPi will do all the work.

3. Results and Discussion

This section will comprise of the Pi updating procedures and installing the WebIOPi framework which will help us handle communication from the webpage to the raspberry pi.

3.1 Update the raspberry Pi

To update the raspberry file typr below commands and then reboot the RPi;

sudo apt-get update

sudo apt-get upgrade

sudo reboot

3.2 Install the webIOPi framework

cd ~

Use wget to get the file from their sourceforge page;

wget http://sourceforge.net/projects/webiopi/files/WebIOPi-0.7.1.tar.gz

When download is done, extract the file and go into the directory;

tar xvzf WebIOPi-0.7.1.tar.gz

cd WebIOPi-0.7.1/

Then we can run the setup installation for the WebIOPi using;

sudo ./setup.sh

Keep saying yes if asked to install any dependencies during setup installation. When done, reboot your pi;

sudo reboot

Test WebIOPi Installation:

Before jumping in to schematics and codes, With the Raspberry Pi back on, we will need to test our WebIOPi installation to be sure everything works fine as desired.

Run the command;

sudo webiopi -d -c /etc/webiopi/config

After issuing the command above on the pi, point the web browser of your computer connected to the raspberry pi to http://raspberrypi.mshome.net:8000 or http;//thepi'sIPaddress:8000. The system will prompt you for username and password.

Username is *webiopi*

Password is *raspberry*

After the login, look around, and then click on the GPIO header link.

Fig -2: WebIOPi Main Menu

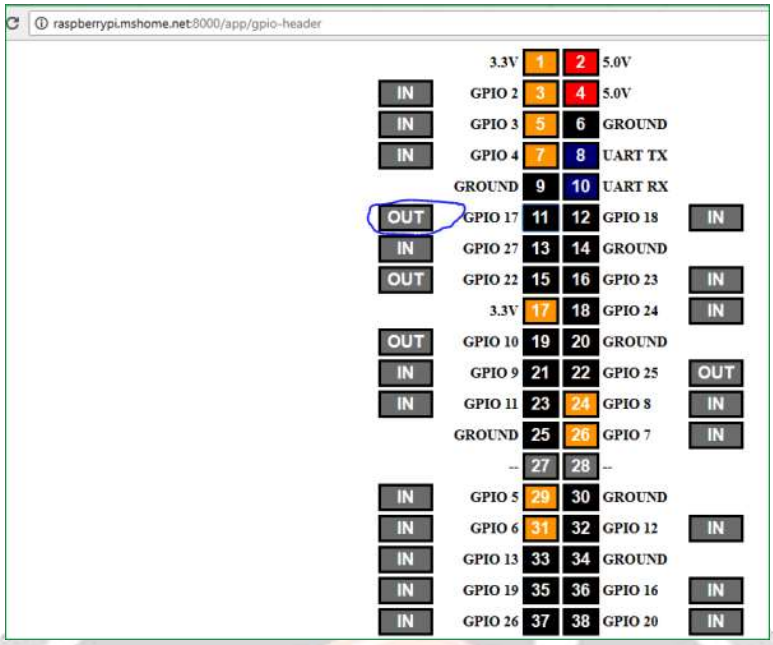

Fig -3: gpio header

With this done, connect the led to your raspberry pi as shown in the schematics below.

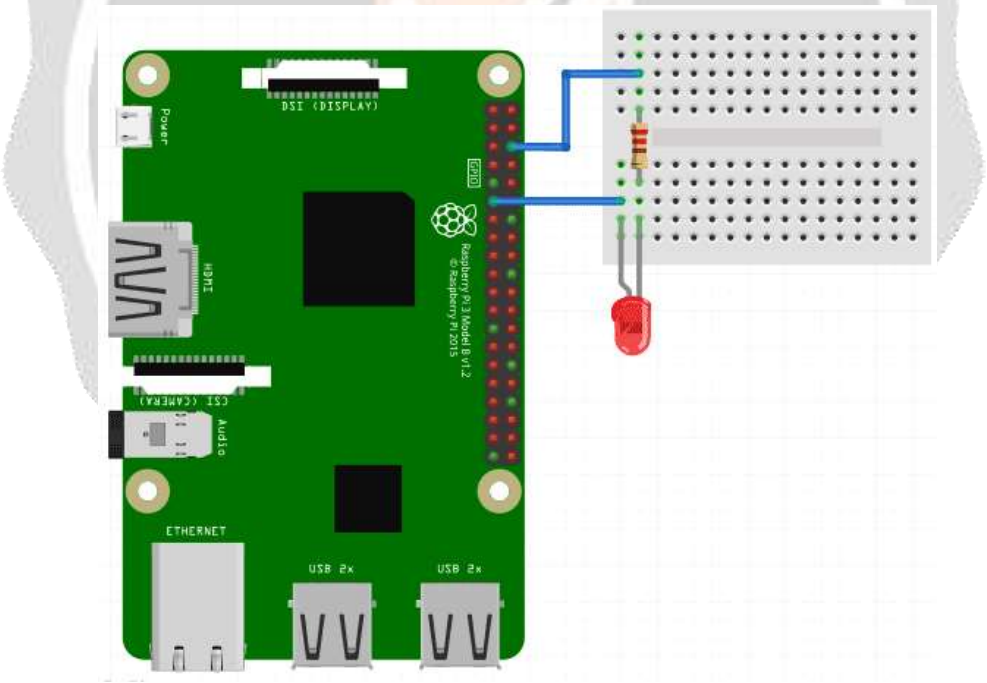

Fig -4: Raspberry Pi Schematics

After the connection, go back to the webpage and click the pin 11 button to turn on or off the LED. This way we can control the Raspberry Pi GPIO using WebIOPi.

3.3 Building the Web Application for Raspberry Pi Home Automation

Here we will be editing the default configuration of the WebIOPi service and add our own code to be run when called. The first thing we will do is get filezilla or anyother FTP/SCP copy software installed on our PC. You will agree with me that coding on the pi via the terminal is quite stressful, so filezilla or any other SCP software will come in handy at this stage. Before we start writing the html, css and java script codes for this IOT Based Control Device and moving them to the Raspberry Pi, lets create the project folder where all our web files will be.

cd ~

mkdir webapp

cd webapp

mkdir html

Create a folder for scripts, CSS and images inside the html folder

mkdir html/css

mkdir html/img

mkdir html/scripts

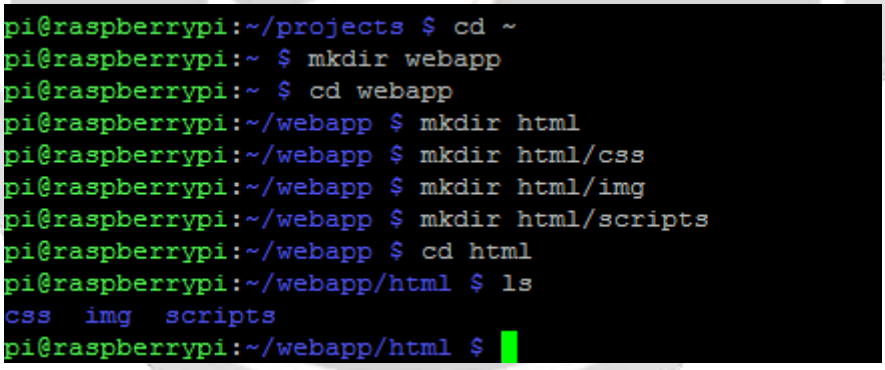

Fig -5: Raspberry Pi Coding

With our files created lets move to writing the codes on our PC and from then move to the Pi via filezilla.

The JavaScript Code:

The first piece of code we will write is that of the javascript. Its a simple script to communicate with the WebIOPi service.

```
webiopi().ready(function() {
        webiopi().setFunction(17,"out");
        webiopi().setFunction(18,"out");
        webiopi().setFunction(22,"out");
        webiopi().setFunction(23,"out");
                var content, button;
        content = $("#content");
                button = webiopi().createGPIOButton(17," Relay 1");
        content.append(button);
                button = webiopi().createGPIOButton(18,"Relay 2");
        content.append(button);
                button = webiopi().createGPIOButton(22,"Relay 3");
        content.append(button);
                button = webiopi().createGPIOButton(23,"Relay 4");
        content.append(button);
    });
```

HTML Code:

The html code pulls everything together, javascript and the style sheet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="viewport" content = "height = device-height, width = device-width, user-scalable = no" />
    <title>Smart Home</title>
    <script type="text/javascript" src="/webiopi.js"></script>
    <script type="text/javascript" src="/scripts/smarthome.js"></script>
```

```
    <link rel="stylesheet" type="text/css" href="/styles/smarthome.css">
    <link rel="shortcut icon" sizes="196x196" href="/img/smart.png" />
</head>
<body>
            </br>
            </br>
            <div id="content" align="center"></div>
            </br>
            </br>
            </br>
            <p align="center">Push button; receive bacon</p>
            </br>
            </br>
</body>
</html>
```

Within the head tag exist some very important features.

```
<meta name="mobile-web-app-capable" content="yes">
```

The line of code above enables the web app to be saved to a mobile desktop is using chrome or mobile safari. This can be done via the chrome menu. This gives the app an easy launch feel from the mobile desktop or home.

The next line of code below gives some sort of responsiveness to the web app. It enables it occupy the screen of any device on which its launched.

```
<meta name="viewport" content = "height = device-height, width = device-width, user-scalable = no" />
```

The next line of code declares the title shown on the title bar of the web page.

```
<title>Smart Home</title>
```

The next four line of codes each perform the function of linking the html code to several resources it needs to work as desired.

```
<script type="text/javascript" src="/webiopi.js"></script>

<script type="text/javascript" src="/scripts/smarthome.js"></script>

<link rel="stylesheet" type="text/css" href="/styles/smarthome.css">

<link rel="shortcut icon" sizes="196x196" href="/img/smart.png" />
```

The first line above calls the main WebIOPi framework JavaScript which is hard-coded in the server root. This needs to be called every time the WebIOPi is to be used.

The second line points the html page to our jQuery script, the third points it in the direction of our style sheet. Lastly the last line helps set up an icon to be used on the mobile desktop in case we decide to use it as a web app or as a favicon for the webpage.

The body section of the html code just contains break tags to ensure the buttons aligned properly with the line below telling our html code to display what is written in the JavaScript file. The *id=”content”* should remind you of the content declaration for our button earlier under the JavaScript code.

```
<div id="content" align="center"></div>
```

WebIOPi Server Edits for Home Automation:

At this stage, we are ready to start creating our schematics and test our web app but before then we need to edit the config file of the webiopi service so its pointed to use data from our html folder instead of the config files that came with it.

To edit the configuration run the following with root permission;

sudo nano /etc/webiopi/config

Look for the http section of the config file, check under the section where you have something like, *#Use doc-root to change default HTML and resource files location*

Comment out anything under it using # then if your folder is setup like mine, point your doc-root to the path of your project file

doc-root = /home/pi/webapp/html

Save and exit. You can also change the port from 8000, in case you have another server running on the pi using that port. If not save and quit, as we move on.

Its Important to note that you can change the password of the WebIOPi service using the command;

sudo webiopi-passwd

Lastly run the WebIOPi service by issuing below command:

sudo /etc/init.d/webiopi start

The server status can be checked using;

sudo /etc/init.d/webiopi status

It can be stopped from running using;

sudo /etc/init.d/webiopi stop

To setup WebIOPi to run at boot, use;

sudo update-rc.d webiopi defaults

If you want to reverse and stop it from running at boot, use;

sudo update-rc.d webiopi remove

3.4 Circuit Diagram and Explanation

With software set up done, we are all set to start creating the schematics for IOT Based Control Device.

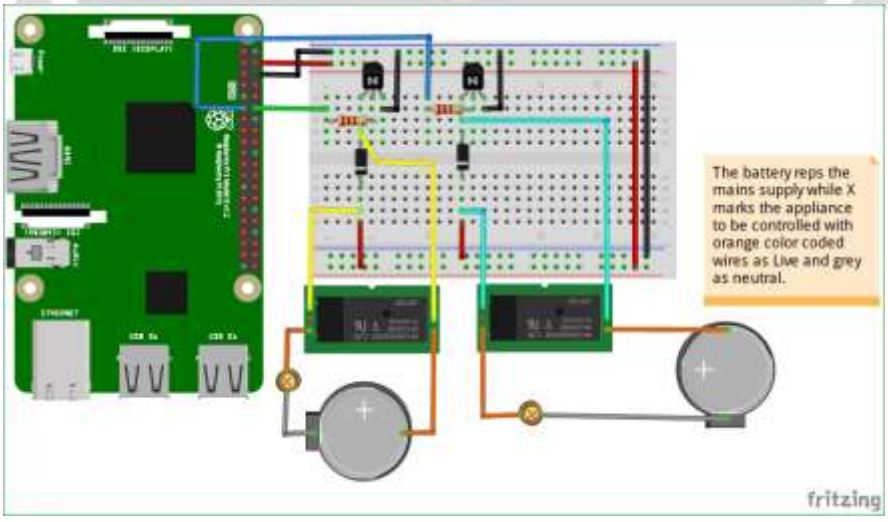

Figure 6

Connect your circuit as shown in the fritzing circuit above. COM, NO (normally open) and NC (normally Close) of your own relay may be located at different sides/points.

With components connected, fire up server, from a webpage, go to the IP of your Raspberry Pi and indicate the port as described earlier, login with your username and password, and you should see a webpage that looks exactly like the image below.

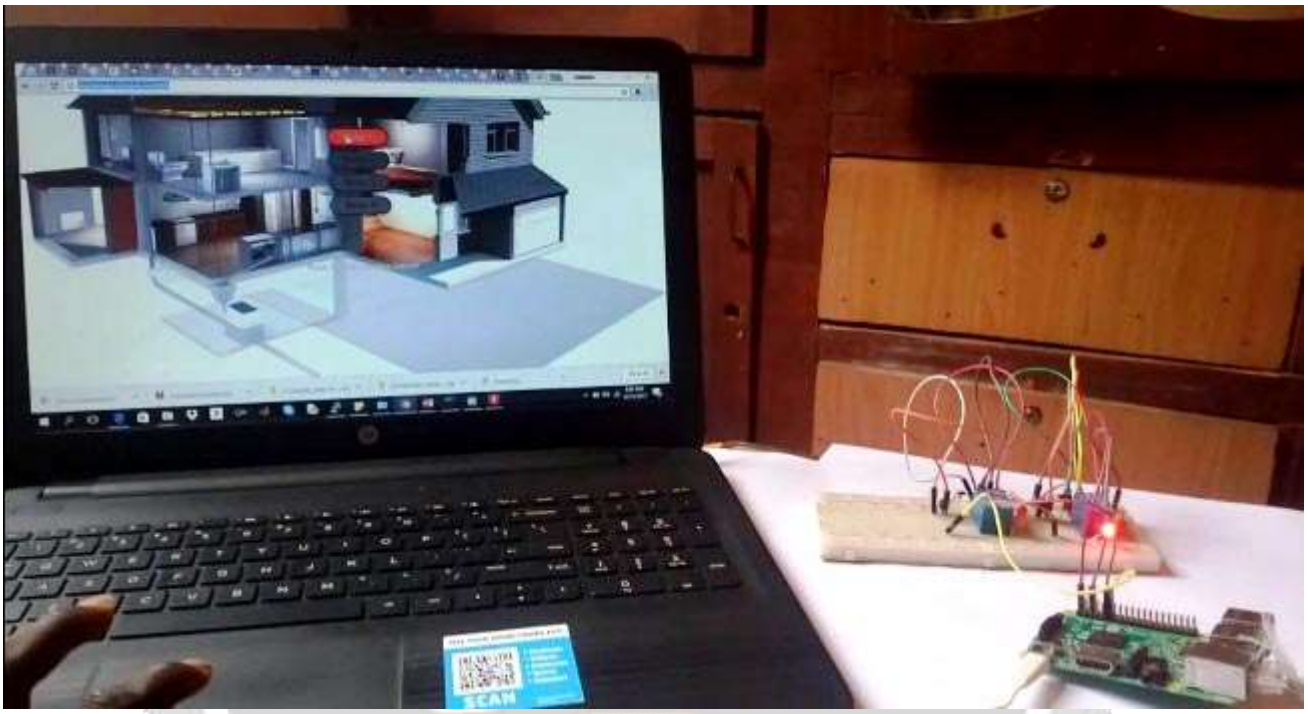

Fig -7: Webpage for controls

Now we can easily control four AC Home appliances from anywhere wirelessly, just by clicking on the buttons. This will work from Mobile phone, tablet etc. and you can add more buttons and relays to control more devices.

4. Conclusion and future work

By outlining the IP Based Control Device in a particular way, it was conceivable to make a framework that was finished with sensors, running project, and status screen in an incremental manner. Building the framework along these lines made discovering bugs from the get-go a simple undertaking to achieve. Subsequent to finishing a gathering task, for example, the HAS, it winds up evident that a solid accentuation on pecking order and measured quality right on time in the outline procedure brings about an unmistakable determination. This determination enables the implementers of the framework to keep an unmistakable thought of what undertakings should be expert and never get confounded by the many-sided quality of the job needing to be done. On the off chance that, while building the framework, any part neglected to be acknowledged, it was conceivable to work around the issue and still figure out how to execute a framework that was working towards the last objective.

IP networks are the most common element linking intelligent devices today, and they will only become more prevalent within buildings of all types. IP-enabled devices can take advantage of the intelligence embedded in other IP-enabled devices, thereby adding value to all of them.

Look at what is happening in the telephone industry. Telephones no longer require physical wires running from a central office to your home or business, they have become mobile devices. And, with the addition of an IP address, voice-mail and other features can be retrieved via computer as well as through the phone itself. This trend will expand exponentially as the pool of IP-enabled equipment mushrooms in the years ahead. We can only guess what advantages this will bring to consumers and businesses.

IP-based control is equally comfortable in a wired or wireless environment. However, wireless is clearly the wave of the future, so this should be carefully considered when moving to IP-based control.

References

[1] http://www.raspberrypi.org
[2] http://www.arduino.cc/
[3] http://www.dhcpserver.de/
[4] http://learn.adafruit.com/reading-a-analog-in-and-controlling-audiovolume-with-the-raspberry-pi
[5] http://pi4j.com/
[6] http://www.susa.net/wordpress/2012/06/raspberry-pi-relay-using-gpio/Raspberry Pi Measurement Electronics:
[7] hardware and software [Kindle Edition]
[8] http://www.zdnet.com/build-your-own-supercomputer-out-of-raspberry-pi-boards-7000015831/
[9] http://fritzing.org/home/
[10] http://www.adafruit.com