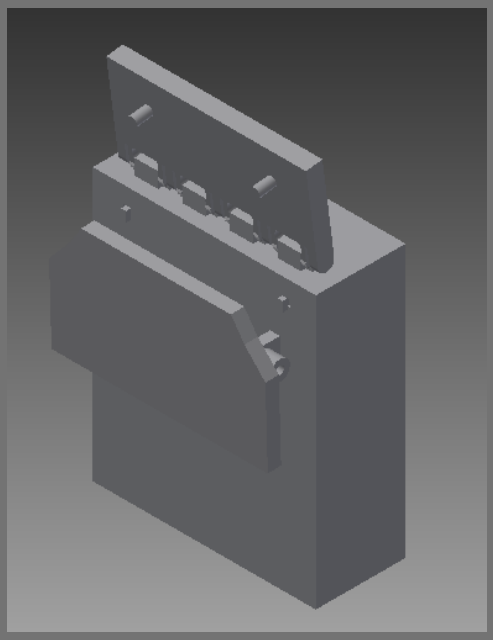
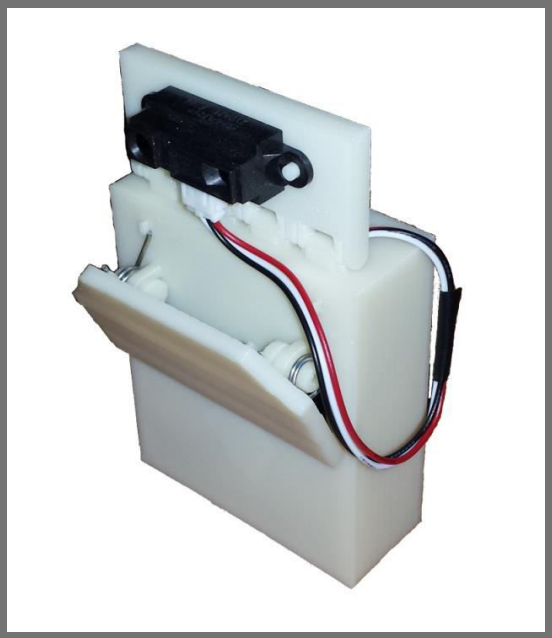
Using IR-Sensors to Limit the Screen-On Time of a Computer

Research Paper





Akshath Jain

Table of Contents

Abstract:

Electricity and power saving techniques on computers are not reliable, causing an excess use of electricity than is actually required. Through research, I concluded that the components within a computer that consume the most energy are the screen, CPU, GPU, and HDD/SSD. To mitigate power consumption, I created and coded IRES, a device designed to mitigate the power usage of the screen and CPU. After testing IRES, I determined that it was 84.81% more efficient than current power saving techniques in optimal conditions and 44.64% more efficient with actual usage. This efficiency results in approximately $4.6 billion in savings and can prevent 34 million metric tons of CO_2 from entering the atmosphere yearly.

Background:

When my laptop ran out of battery for the third time in one day, I decided that I would need to do two things. One, either get a new laptop, or two, figure out some way to save energy consumption on my current laptop. I could have just adapted to the situation and kept my laptop plugged in the entire time, but that would take away from the concept of a laptop: allowing versatility to the user in places where they can use it.

In order to solve my problem, I had to pinpoint the culprit, that is to say, the component that takes up the most energy within my laptop. I had a hunch that it would be the CPU and the screen, and I was right. After doing some research, I found out that a study had been conducted on the top power consuming components of a computer. Depending on what task you were doing, the most energy dependent processes were the screen, CPU, or GPU.[1] I decided then and there that I had to do something about it.

Hypothesis:

The hypothesis of my experiment is as follows: If I can create a device that can control when the screen and CPU are on, then I will see a decreased use in electricity. What I am trying to achieve through this experiment is creating a device that knows when the user is at the computer, turns the screen of the computer off when the user leaves, and then shortly thereafter puts the computer into a hibernate mode, with an ultimate goal to mitigate energy consumption. In order to do this, I decided that I would need some sort of sensor that can send output about the user's presence at a computer, a device that can interpret the information from the sensor, and that I would need to code both the interpreter and some sort of computer application.

[1] Mahesri, Aqeel and Vibhore, Vardhan. "Power Consumption Breakdown on a Modern Laptop"

The Engineering and Design Process:

The first step of my engineering and design process was to pick a sensor suitable to my needs. The sensors that I considered using were the camera on the computer, an infrared (IR) sensor, a pressure pad, and an ultrasonic sensor. Using logic, I immediately eliminated the camera because it would take too much energy to operate, which is counterproductive to my experiment as it is about saving energy, and I also eliminated the pressure pad because it would have been too much of hassle, with wires coming from the chair to the computer and having the user set it up properly. From there, I narrowed down my choices to two sensors, the IR sensor and the ultrasonic sensor. Both options would have fulfilled my needs, except that I had to order these components online, and the IR sensor was arriving three weeks earlier than the ultrasonic sensor.

The IR sensor that I used in my experiment was a Sharp GP2Y0A21YK0F IR sensor. The IR sensor is composed of an integrated combination of a PSD (position sensitive detector) and an IRED (infrared emitting diode). Using triangulation methods, the sensor is ideal for being a proximity sensor because the reflectivity of the object and the environmental temperature do not easily hinder the distance detection of the sensor. Optimal range for using this sensor is 10cm – 80cm, the same distance that a person sits away from their computer, distances beyond 80cm can be measured, but result in loss of accuracy. In order to operate properly, the sensor takes in 5 volts, and uses the IRED to send an IR beam out towards the user. This beam is then received by the PSD and triangulated to calculate the distance of the subject the sensor is measuring. Once the distance is gauged, the sensor outputs analog input, or varying levels of voltage. This voltage is directly proportional to the distance being measured. Shorter distances around 10cm give off higher voltages, while distances around 80cm result in lower voltages.

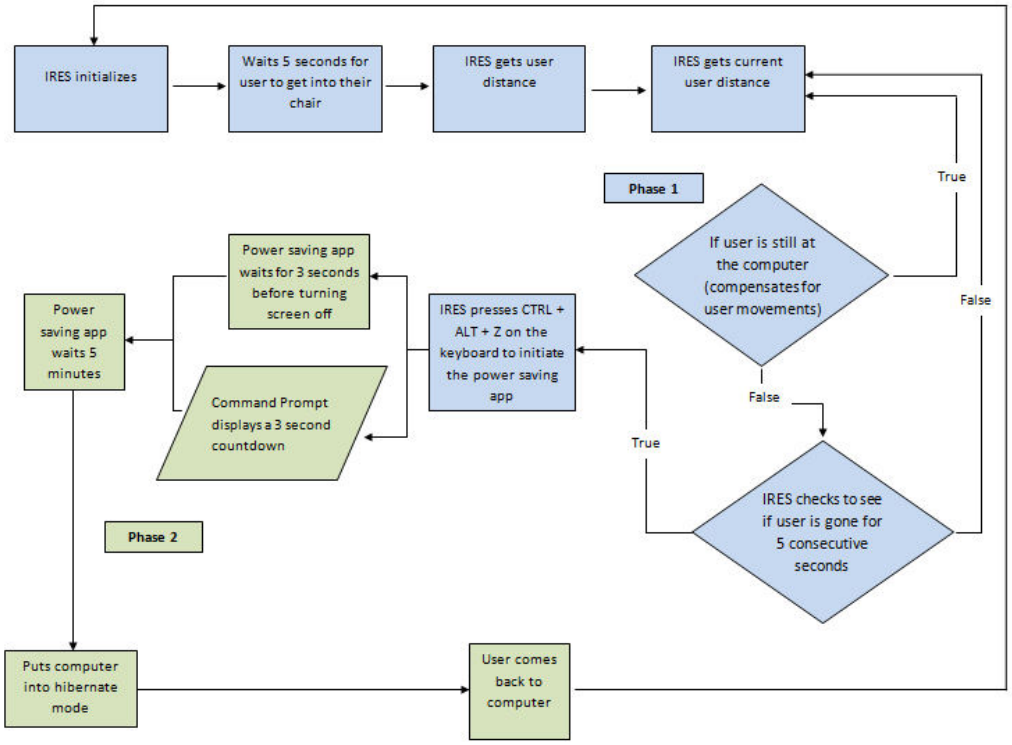
Since the sensor outputs information in a format unreadable to the computer, I needed some sort of interpreter that could decipher this information and process it. For that, I chose the Arduino Leonardo, a microcontroller based around an integrated ATMega32u4 chipset, designed to be programmed using the Arduino IDE (integrated development environment), which is a program that is based off of the Processing IDE. The language used by the Arduino IDE is similar to C++, following much of the same syntax. The Arduino Leonardo's distinguishing feature is that it, unlike many other Arduino platforms, has the ability to act as a USB

(universal serial bus) 2.0 device, a feature plays an integral role within my project. Other appealing aspects of the Arduino Leonardo include its small size and slim profile in addition to requiring very little energy to operate, only 8 mAh, which less than that of an external wireless mouse, something that I found out during testing my device. After acquiring both a sensor and interpreter, I got in touch with my six year old self. Yes, I dug up some dusty old Legos that I hadn't touched in years and started playing, (cough) I mean building. Using a combination of Legos and K'NEX, I built a structure that would clip onto my computer and provide a stable platform for testing (the pictures on the cover page), giving me more accurate and consistent results throughout my tests, and preventing myself from manually holding the Arduino and IR sensor, thus eliminating human error. Although I used this Legos and K'NEX device for testing, to improve upon it, I created a 3D-printed casing for it and a GUI to monitor information coming from the device.

To 3D print the device out, I had to first design a 3D rendering of it in Autodesk Inventor. Once I had done that, I needed to export the file into a format readable by the 3D printer[2], which was designed by uPrint. After printing for 7 hours, the ABS Plus plastic components needed to go into a sodium-hydroxide bath, which would get rid of all support materials used by the printer during printing. Although I was close, I could not get every dimension perfect during my first try, resulting in me re-printing this out again[3]. I call this device IRES, or Infrared Energy Saving Device, but in order to turn IRES from a paperweight to an energy saving device, I needed to code it.

[2] Autodesk Inventor saves it's files as type .ipt whereas the 3D printer uses a .stl file type.

[3] I printed out my device using my school's 3D printer, knowing that these printers were meant for prototyping, printing again was just a matter of redesigning the 3D rendering.



During Phase 1, the initial boot up of IRES, the first action done is gauging the user's initial distance, that is, the distance from IRES to the user. From there, iterating once every second, IRES acquires the user's current distance and compares it to the user's initial distance. During this comparison, IRES determines whether the user has left the computer. To make these assumptions as precise as possible, I designed IRES's algorithm to account for the user leaning back, slight movements backwards and forwards, and any other superfluous movements. IRES give the user twenty centimeters to move in order to avoid false positives. IRES iterates this loop continuously until it determines that the user has left the computer. In order to make sure that these assumptions are correct, IRES checks to see that the user is gone for at least five consecutive seconds. If IRES discovers that it was a false positive and the user has returned, then IRES continues to get the user's current distance. If IRES determines that the user has left, it emulates pressing the Control, Alt, and 'z' keys on the keyboard simultaneously, as it has the ability to act as a USB 2.0 device, to initiate Phase 2, the screen control app that I designed.

Phase 2 of this process takes place on the computer through an application designed in Microsoft Visual Studios and programmed in C++. Running through the command prompt enables this application to be as efficient as possible, in addition to requiring the least amount of storage space and energy possible. This application is "Hot Keyed" into my computer, meaning that whenever I press Ctrl + Alt + 'z' on the keyboard,

it will start up anywhere on the computer, regardless of what task is occurring (it starts up on top of the activity open, it does not close the activity itself). When this application is called upon, it displays a three second countdown as a final failsafe, if the user is still at the computer, to alert the user the screen will be shutting off momentarily. Once the screen is off, it begins a five minute countdown until it puts the computer into a hibernate mode. As the computer goes into the hibernate mode, all open applications and tasks will be written to the hard drive temporarily so all applications open prior to hibernation will be running at the boot up of the computer, when the user comes back. Once the hibernate function is called upon, the screen control application automatically terminates itself, resulting in an encapsulated user experience.

Going back to IRES, when it initiates the computer application, IRES continues to check whether the user has returned to the computer, so when the user is back, the entire process can begin again, given that the user is back within five minutes. If, however, the user is gone long enough for the computer to be entered into a hibernate mode, more than five minutes, IRES turns off, so when the user comes back to the computer, the process starts again at the beginning.

As IRES iterate (once every second), it sends the data to a GUI[4] that I coded within the Processing IDE, with the Java programming language. This application that I coded enabled me to receive data from IRES in a readable format, and also make the user aware of their status, such as total time spent working, distance from the computer, and when they should take a break from the computer[5].

After I successfully created IRES and made sure that all aspects of it worked properly, I proceeded to test it, designing numerous test cases to properly assess the efficiency of IRES.

Experimental Procedure:

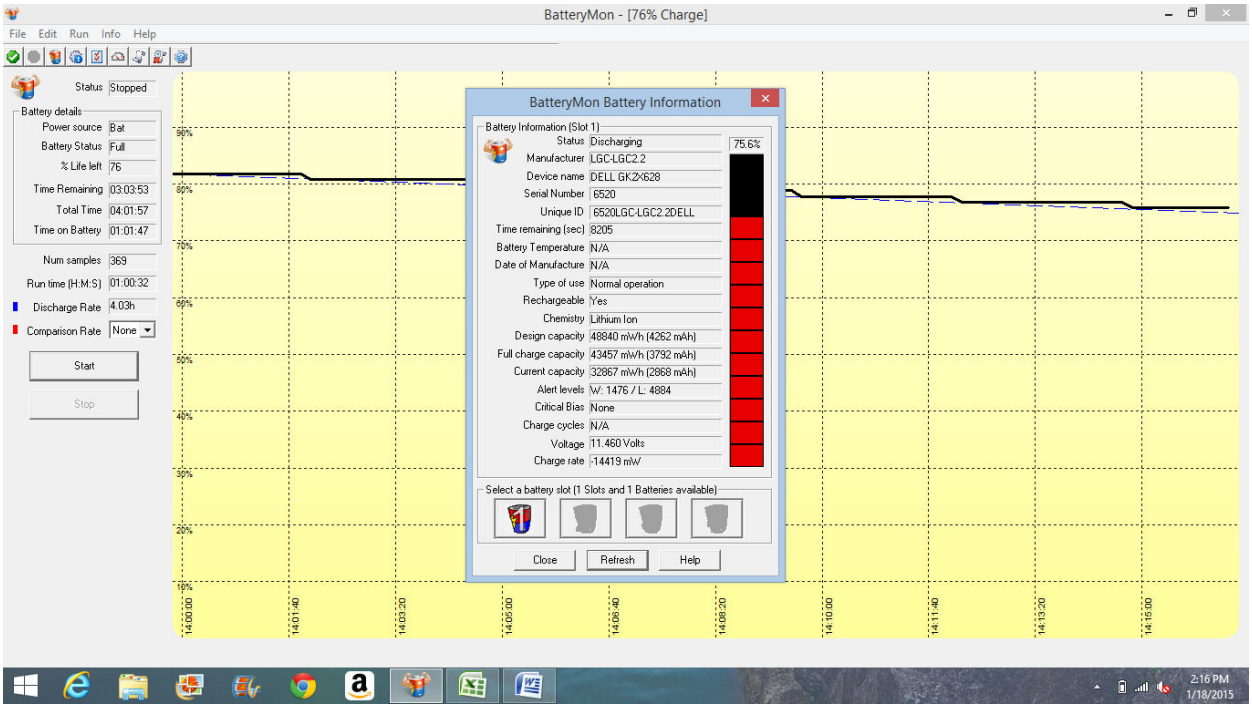
To test IRES, I designed 6, 60 minute tests, in addition to a control test. The testing unit was a Dell Inspiron running an Intel Core i5, clocked at 2.5 GHz with 64-bit architecture, a Lithium-Ion battery, and 4GB of memory. Each test case had three trials, and the average of those trials was used to calculate the overall efficiency. In addition, surveys from fifteen people were acquired to help formulate three of the test cases. The

[4] A GUI is a graphical user interface; it is the part of software that the user interacts with.
[5] The user is alerted after two hours of continuous use that they need to take a short break, and then once every ten minutes after that

control was designed to test how much energy is required to run a computer for 60 minutes. Test Case 1 was designed to test how much energy is required to run IRES for 60 minutes. Test Case 2 tested how efficient a computer's power saving techniques are. Test case 3 was designed to assess how efficient IRES's power saving techniques are. Test Cases 4, 5, and 6 were all designed using the survey data I collected from fifteen people. Test Case 4 was designed to analyze how efficient a laptop is with actual usage habits. Test Case 5 was designed to test how efficient IRES is with actual usage habits. And finally, Test Case 6 was designed to test how efficient a desktop is with actual usage habits.

Constants through the entirety of testing was the screen brightness, set at 100%, the time for each trial set to 60 minutes, and both Microsoft Excel and BatteryMon (I explain this in the next paragraph) were open throughout the entire test.

To acquire battery statistics before and after each test, along with the runtime of each test, I used BatteryMon by PassMark Software[6]. This application gave me the battery statistics in milliamp hours (mAh) and milliwatt hours (mWh) in addition to tracking the time elapsed during the test and the battery level depletion throughout the test[7].



[6] "Monitor Laptop and UPS Batteries." *PassMark BatteryMon*.

[7] This is shown in the figure above, the depletion is shown in the back, but is partly covered by the statistics menu. This figure is from the control, test 1.

IRES Test Results

Tests	Start			End			Time Elapsed	Variables			
	Starting Percentage	Starting Battery Level		Ending Percentage	Ending Battery Level			Applications Open			IR-Sensor
	%	mAh	mWh	%	mAh	mWh	min	Microsoft Excel	BatteryMon	Microsoft Word	Enabled
Control											
1	100.00%	3792	43457	75.60%	2868	32867	60	X	X		
2	100.00%	3770	43457	76.00%	2866	33034	60	X	X		
3	100.00%	3770	43457	75.60%	2851	32856	60	X	X		
Average	**100.00%**	**3777**	**43,457.00**	**75.73%**	**2862**	**32,919**	**60**				
Test Case 1											
1	100.00%	3783	43457	75.10%	2840	32623	60	X	X		X
2	100.00%	3778	43457	75.20%	2840	32667	60	X	X		X
3	100.00%	3615	41614	75.40%	2725	31369	60	X	X		X
Average	**100.00%**	**3725**	**42,842.67**	**75.23%**	**2802**	**32,220**	**60**				
Test Case 2											
1	100.00%	3590	43457	93.80%	3368	40770	60	X	X		
2	100.00%	3611	43457	91.70%	3313	39871	60	X	X		
3	100.00%	3665	43457	87.30%	3119	37929	60	X	X		
Average	**100.00%**	**3622**	**43,457.00**	**90.93%**	**3266.67**	**39,523.33**	**60**				
Test Case 3											
1	100.00%	3573	43457	97.80%	3492	42480	60	X	X		X
2	100.00%	3561	43457	98.10%	3493	42624	60	X	X		X
3	100.00%	3561	43457	98.20%	3497	42680	60	X	X		X
Average	**100.00%**	**3565**	**43,457.00**	**98.03%**	**3494**	**42,594.67**	**60**				
Test Case 4											
1	100.00%	3788	43457	73.90%	2800	32123	60	X	X	X	
2	100.00%	3802	43457	73.30%	2788	31168	60	X	X	X	
3	100.00%	3629	41614	72.80%	2641	30292	60	X	X	X	
Average	**100.00%**	**3739.67**	**42,842.67**	**73.33%**	**2743.00**	**31,194.33**	**60**				
Test Case 5											
1	100.00%	3696	43457	84.50%	3122	36708	60	X	X	X	X
2	100.00%	3692	43457	84.50%	3119	36719	60	X	X	X	X
3	100.00%	3517	41614	85.50%	3007	35587	60	X	X	X	X
Average	**100.00%**	**3635.00**	**42,842.67**	**84.83%**	**3082.67**	**36,338.00**	**60**				
Test Case 6											
1	100.00%	3818	43457	71.90%	2744	31224	60	X	X	X	
2	100.00%	3793	43457	73.60%	2793	32001	60	X	X	X	
3	100.00%	3652	41614	71.20%	2600	29626	60	X	X	X	
Average	**100.00%**	**3754.33**	**42842.67**	**72.23%**	**2712.33**	**30950.33**	**60**				

Control:

Control						
1	100.00%	3792	43457	75.60%	2868	32867
2	100.00%	3770	43457	76.00%	2866	33034
3	100.00%	3770	43457	75.60%	2851	32856
Average	**100.00%**	**3777**	**43,457.00**	**75.73%**	**2862**	**32,919**

The control test was designed to test how much energy is required to run my laptop (Dell Inspiron) for 60 minutes. Going through with the test, I had three trials, which I then took the average of to find the overall net decrease. Power saving techniques used by the computer were not enabled during this test, it was just running my laptop for 60 minutes without any interruption. At the end of the control, I found out that it takes 915 mAh (Starting mAh – Ending mAh) to run my computer for 60 minutes.

Test Case 1:

Test Case 1						
1	100.00%	3783	43457	75.10%	2840	32623
2	100.00%	3778	43457	75.20%	2840	32667
3	100.00%	3615	41614	75.40%	2725	31369
Average	**100.00%**	**3725**	**42,842.67**	**75.23%**	**2802**	**32,220**

Test Case 1 was designed to test how much energy IRES takes in a one hour time period. In order to do this, IRES was set up with the same program that it normally uses, except that the cable I used only had power and ground connections, which enabled IRES to run for 60 minutes without any data transfer, meaning that power saving techniques would not be activated. In addition, while testing this, the computer's default power saving techniques were turned off. At the end of my trials, I found out that IRES only takes 8 mAh to operate, which is a minuscule number compared to the amount of power it takes to power a laptop for an hour. To arrive at this result, I used the change in (Δ) mAh from this test, and subtracted it from the ΔmAh from the control. This got me how much energy it takes to power IRES for 60 minutes.

Test Case 2:

Test Case 2						
1	100.00%	3590	43457	93.80%	3368	40770
2	100.00%	3611	43457	91.70%	3313	39871
3	100.00%	3665	43457	87.30%	3119	37929
Average	**100.00%**	**3622**	**43,457.00**	**90.93%**	**3266.67**	**39,523.33**

Test Case 2 was designed to test how efficient a computer is with default power saving techniques. With the default techniques enabled, the computer goes to sleep after 5 minutes of inactivity, and then 10 minutes after that (for a total of 15 minutes) the computer goes into a sleep mode, which is like the hibernate mode IRES uses, except that it does not turn the computer completely off, resulting in more power being used. After three

60 minute trials, I found out that it takes 355.33 mAh (ΔmAh from start to finish) to power my computer for 60 minutes with the default power saving techniques.

Test Case 3:

Test Case 3						
1	100.00%	3573	43457	97.80%	3492	42480
2	100.00%	3561	43457	98.10%	3493	42624
3	100.00%	3561	43457	98.20%	3497	42680
Average	**100.00%**	**3565**	**43,457.00**	**98.03%**	**3494**	**42,594.67**

Test Case 3 was to test how efficient IRES is. In this test, the computer's default power saving techniques were turned off, so as not to interfere with IRES, and IRES and all of its components were enabled. After the three, 60 minute trials, I found out that using IRES, it takes only 71 mAh (ΔmAh from start to finish) to run a computer using IRES, showing that this new power saving solution is already significantly better than traditional power saving techniques.

The problem with the control, Test Cases 1, 2, and 3 are that these tests are best case scenarios. This means that these tests simulate a person getting up and leaving for one hour. The problem with this is that people do not just get up and leave for an hour. These tests are not completely representative of the entire population. What these tests do show however, is the max efficiency of IRES. These scenarios can happen, but they do not happen very frequently. In order to do that, I would need to survey people to get data that represents what people's habits are when they work.

When I created the survey, I designed it with three parts. Part one inquires about how frequently the person uses their computer, part two inquires if they know any power saving techniques used by their computer, and part three inquires about their habits while they work with a computer. When I surveyed people, I surveyed 6 of them by hand (going to their houses, and physically asking them questions), but then I realized that it would be much more efficient if I made an online survey. The online survey was through Google Forms, and was designed exactly like the original, except in the Google Forms format. The advantage of using the online survey was that I was able to get a farther reach, and I was able to save time and work on other aspects of this project in the mean time.

Proctor's Name:
Participant's Name:
Date:

Computer Usage

1. Do you use a computer daily?
2. How long do you use it for on average?
3. Do you primarily use a desktop or a laptop?
    a. (If laptop) Is your laptop plugged in and charging while you work?
        i. (If yes) Why do you keep your laptop plugged in? Is it because of a short battery life?
            1. (If yes) Would you benefit if your battery life was better, giving you more versatility in places you are able to work?

Power Saving Techniques

4. Are you aware of any power-saving techniques used by your computer (i.e. screen turns off, computer goes to sleep, etc.)?
    a. (If Yes) How long until the screen times out? (With absolute certainty)
    b. How long until your computer goes to sleep? (With absolute certainty)

Work Habits

5. When you work, do you ever take short breaks away from your computer? This can include anytime that you are away from your computer.
    a. (If yes) For how long do you work at a time, and then take a break?
    b. What is the duration of your break?

Questionaire Results

	Computer Usage						Power Saving Techniques			Computer Habits		
	1	2	3	3a	3ai	3ai1	4	4a	4b	5	5a	5b
	Y/N	Hours	Laptop (L)/ Desktop (D)	Y/N	Convenience (C)/ Short Battery Life (SBL)/ Other (O)	Y/N	Y/N	Minutes	Minutes	Y/N	Hours	Minutes
Person												
1	Yes	10	L	Yes	C	Yes	Yes	10	15	Yes	2	10
2	Yes	10	L	Yes	SBL	Yes	Yes	5	10	Yes	1	45
3	Yes	10	L	Yes	C	Yes	No	5	15	Yes	2	10
4	Yes	10	L	Yes	C	Yes	Yes	15	30	Yes	1	5
5	Yes	10	D				No	0	0	Yes	3	30
6	Yes	10	L	Yes	C	Yes	Yes	5	15	Yes	1	10
7	Yes	10	L	Yes	C	Yes	Yes	5	10	Yes	0.5	5
8	Yes	8	L	Yes	C	Yes	Yes	5	15	Yes	2	5
9	Yes	10	L	Yes	C	Yes	Yes	10	15	Yes	4	5
10	Yes	8	L	Yes	C	Yes	Yes	5	15	Yes	1	32.5
11	No	1/2	D				No	0	0	No		
12	Yes	8	L	Yes	SBL	Yes	No	5	15	Yes	1.75	12.5
13	Yes	8	L	Yes	C	Yes	Yes	5	15	Yes	1	10
14	Yes	8	D				Yes	0	0	Yes	2.50	12.5
15	Yes	8	L	Yes	C	Yes	Yes	5	15	Yes	1.25	7.5
Average		**9.14**						**5.33**	**12.33**		**1.71**	**14.29**

Using the results from my questionnaire, I can conclude that from my sample size, people work for an average of about 102 minutes (1.71 * 60), and then take a 14 minute break. Since my tests were 60 minutes each, all I really need to know is that people's breaks are around 14 minutes. I do not need to know people's time before they take a break because I am trying to get as many work/break iterations in my tests as possible. If I turned these results into one hour intervals, I would have approximately 53 minutes of work, and a 7 minute

break, and if I were to use those parameters, then it would be extremely similar to the control, test cases 1, 2, and 3. In order to make my parameters representative of my survey results, I had to keep the break time, because that is what I am ultimately testing in this project (I am comparing how efficient power saving techniques are when people are gone for a given period), and use an arbitrary work time to help provide distinct results between power saving techniques. For my parameters, I chose a 1:1 ratio of work to break, using 15 minutes of work, to 15 minutes of break (the break time being based off of the survey). I could, in theory, use a 1:15 minute ratio (or any other time for work) except then the amount of energy used for each test case would be too close in value (or it would not have run for enough iterations if I used a larger value), resulting in less distinct results. Using these parameters (the 15:15 ratio), I would be able to get a total of two work/break iterations during my testing, getting that sweet spot of number of iterations and distinct results between tests, allowing me to simulate what it would be like if someone were actually using IRES with their computer.

After deciding on what the times would be for my work to break ratio, I created my tests, but to simulate the work, I used the Arduino to write the phrase "Power saving device test" over and over for 15 minutes in Microsoft Word, and then take a 15 minute break. But in order to account for the energy that the Arduino took, I would have to add it back on to the end result. The problem with this would be that during test case 2 (the test in which I found out how much energy IRES/Arduino took), the IR sensor was enabled, so for these tests, I also needed the IR sensor to be enabled and using power, but not actually sending back any information. This means that I could use the results from test case 2 to determine how much power the Arduino was using these tests, and add back on the proper amount.

Test Case 4:

Test Case 4						
1	100.00%	3788	43457	73.90%	2800	32123
2	100.00%	3802	43457	73.30%	2788	31168
3	100.00%	3629	41614	72.80%	2641	30292
Average	**100.00%**	**3739.67**	**42,842.67**	**73.33%**	**2743.00**	**31,194.33**

Test Case 4 was designed to test how efficient a laptop is with actual usage habits. Using the 15:15 work to break ratio and three, 60 minute trials, I determined that a laptop uses 988.67 mAh (ΔmAh from start to finish) of electricity (this amount is including the Arduino). The applications open in this were Microsoft Excel, BatteryMon, and Microsoft Word. The laptop's default power saving techniques are set to turn the screen off

after 5 minutes of inactivity, and then put the laptop into a sleep mode after another 10 minutes of inactivity (for a total of 15 minutes). I used the laptop's default techniques because the average results from my survey were 5.33 minutes, and 12.33 minutes. The options on the laptop are in five minute increments, so for consistency, I chose the default settings. From this test I determined that it takes 988.67 mAh (ΔmAh from start to finish) of electricity to run a laptop with actual usage habits for 60 minutes.

Test Case 5:

Test Case 5						
1	100.00%	3696	43457	84.50%	3122	36708
2	100.00%	3692	43457	84.50%	3119	36719
3	100.00%	3517	41614	85.50%	3007	35587
Average	**100.00%**	**3635.00**	**42,842.67**	**84.83%**	**3082.67**	**36,338.00**

Test Case 5 was designed to test how efficient IRES is with actual usage habits. For this test, I used the same Arduino program as Test Case 4, except in this I disabled the computer's default power saving settings, and instead programmed to Arduino to initiate my power saving app after 15 minutes of work. Also, whenever the computer entered into hibernate mode, I had to manually restart it every thirty minutes because the Arduino was no longer on. Just like Test Case 4, it had the same applications running, Microsoft Excel, BatteryMon, and Microsoft Word, and like the other tests, it was also 60 minutes with three trials. Unlike Test Case 4, when I calculated the overall efficiency, I did not add the power of the Arduino back on because using the Arduino in this test just takes the place of using IRES. From this test I determined that it takes 552.33 mAh (ΔmAh from start to finish) of electricity to run a computer with IRES with actual usage for 60 minutes.

Test Case 6:

Test Case 6						
1	100.00%	3818	43457	71.90%	2744	31224
2	100.00%	3793	43457	73.60%	2793	32001
3	100.00%	3652	41614	71.20%	2600	29626
Average	**100.00%**	**3754.33**	**42842.67**	**72.23%**	**2712.33**	**30950.33**

Test Case 6 was designed to test the efficiency of a desktop with actual usage. The usage was simulated using the same Arduino program used in Test Case 4, with the 15 minutes of work, writing "Power saving device test" proceeded by a 15 minute break. There were no power saving settings enabled for this because desktops do not require power saving, as they are connected directly into a power source, but when desktops do have power saving settings, they are usually just a screen saver that is activated around 30-40 minutes of

inactivity, which is beyond the parameters for testing. Like the other Test Cases, there were three, 60 minute trials with Microsoft Excel, BatteryMon, and Microsoft Word open. Like Test 4, I will be adding the amount of energy the Arduino took during testing while determining overall efficiency. From this test, I determined that it takes 1034 mAh (ΔmAh from start to finish) of electricity to run a desktop for one hour with actual usage.

Results:

Overall Statistics				
12:3 Laptops to Desktops	Status Qou	IRES	Savings	IRES Energy Reduction
	ΔmAh	ΔmAh	mAh	%
Desktops	915.67	71.00	844.67	92.25%
Laptops	355.33	71.00	284.33	80.02%
Average w/ 12:3	**467.40**	**71.00**	**396.40**	**84.81%**
Desktops w/ Usage	1034.00	552.33	481.67	46.58%
Laptops w/ Usage	988.67	552.33	436.33	44.13%
Average w/ 12:3	**997.73**	**552.33**	**445.40**	**44.64%**

From my tests, I can conclude that it takes 915.67 mAh to power a desktop (from the control test) for 60 minutes, 355.33 mAh to power a laptop for 60 minutes, and 71 mAh to power IRES for 60 minutes. Using these results, coupled with the statistic from my survey of twelve laptops for every three desktops, results in 84.81% of cost reduction if IRES were to be implemented. To get to these results, I first took the average of the ΔmAh of laptops and desktops, and averaged them out using the 12:3 ratio I got from my survey ((12ΔmAh(laptops) + 3ΔmAh(desktops))/15). I then found out how much energy I could save if IRES were implemented, which was 396.40 mAh (467.40 – 396.40). To calculate IRES's energy reduction, I simply divided 396.40/467.40, which gets me 84.81%.

This number however, is the best case scenario, and there is not always a best case scenario, so using the statistics from the tests with usage, the energy reduction is 44.64%. To get this, I used the same method as described above, except I subtracted 8 mAh to account for the Arduino's energy consumption for the ΔmAh for desktops with usage and laptops with usage. Using the same methods (including the 12:3 ratio), I get 44.64% of energy reduction if IRES were to be implemented.

Applications:

Savings Statistics								
Energy Statistics							Savings	
	# of Computers	Energy Consumption	Computer Usage	Energy Cost	Work Days	CO_2	IRES Optimal	IRES w/ Usage
	Computers (million)	kW (w/ 12:3)	Hours	¢ / kWh	Days	lbs	%	
Yes	289.89	0.12	9.14	12.46	261	1.855	84.81%	44.64%
No	20.71	0.12	0.5	12.46	121	1.855	84.81%	44.64%

In the United States, there are 310.6 million computers[8], the national average for a kWh of electricity is 12.46¢[9], and each computer uses approximately 0.12 kW of energy (100W for laptops, 200W for desktops[10], applying the 12:3 ratio). To calculate the reduction of cost in electricity within the United States, I first have to divide the country into two groups of people: people who use their computer every day, and people who do not.

Using data from my survey, I determined that 14 out of 15 people use their computer every day. This means that there are 289.89 million computers that are being used for 9.14 hours every day for 261 days a year (workdays). Combining that with electricity usage of a computer and cost of electricity in the United States equates to $10,340,376,300 spent every year on computers alone (289,890,000 * 0.12 * 9.14 * 0.1246 * 261), which means that every person that uses a computer daily spends $35.67 on a computer yearly. Applying IRES's savings of 44.64% means that as a country, the part of America that uses their computers every day could save $4,615,943,980 every year.

On the flip side, I determined that 1 out 15 people do not use their computer every day, but once about every 3 days. This equates to 20.71 million computers that are being used for 30 minutes a day for 121 days a year. Coupled with the cost of electricity and the power usage of a computer equates to $18,734,183.16 (20,710,000 * 0.12 * .5 * 0.1246 * 121) which means that every person who does not use a computer daily spends $0.90 on a computer yearly. Applying IRES's 44.64% energy reduction, the part of America that does not use their computer every day could save $8,362,939.36 every year.

Adding these results up, the United States as a whole spends $10,355,900,180 every year on electricity to power their computers. If IRES were to be implemented, the United States could see savings in electricity

[8] "Top Ten Countries with Highest number of PCs." *Maps of World*.
[9] "Recent Data." *U.S. Energy Information Administration*.
[10] "How much power does a computer use? And how much CO_2 does that represent?" *Energuide*.

usage by up to $4,624,306,919 every year, allowing people to cut their computer's electricity bill by almost half, but there are many more benefits of IRES than just cost reduction.

On average one kWh of electricity produces 1.855 lbs of CO_2[11]. Using the same usage statistics as above, people who use their computers every day emit 76,968,868.41 tons of CO_2 on a yearly basis ((289,890,000 * 0.12 * 9.14 * 1.855 * 261)/2000) and people who do not use their computers every day emit 254,921.71 tons of CO_2 yearly ((20,710,000 * 0.12 * 9.14 * 1.855 * 121)/2000). IRES could prevent 34,358,902.86 and 113,797.05 tons of CO_2 from entering the atmosphere respectively, which equates to a total of 34,472,699.91 tons of CO_2 every year. To put that into perspective, automobiles and the transportation sector in the U.S put out 1,522,000,000 tons of CO_2 every year[12].

Discussion:

After building, coding, testing, and analyzing IRES, I did some research and found that many other people have created power saving solutions for their computer, whether through software, or through hardware approaches. For example, a United States patent[13] depicted a piece of software that would run alongside your computer to limit the clock speeds of various pieces of hardware depending on what task you were doing. Other studies in 2005 by the University of California[14] looked at the architecture of CPUs, architectures of system software, and system software to applications to find that through new techniques used to create smaller, more efficient computers, resulted in reduced computer power usage. These techniques are not limited to technological advances either, but some such studies, such as a 1998 study from the University of California[15] shows the use of software to save computer battery power through remote process executions, where larger tasks from computers are wirelessly transmitted to remote servers to be processed, then the results are sent back. All these studies show that making even one change to how a computer operates could result in drastic power reductions. Like my own study, they conclude that changing how one or more aspects of a computer operates, concludes with results similar to my own, that is, immense power reduction.

---

[11] "How much carbon dioxide is produced per kilowatt hour when generating electricity with fossil fuels?" *U.S. Energy Information Administration*.
[12] "How much carbon dioxide is produced by burning gasoline and diesel fuel?" *U.S. Energy Information*
[13] "Power Management for a Laptop Computer with Slow and Sleep Modes." *Google Patents*.
[14] Venkatachalam, Vasanth, and Michael Franz. "Power Reduction Techniques for Microprocessor Systems."
[15] Rudenko, Alexey, et al. "Saving Portable Computer Battery Power through Remote Process Execution."

Of course, there may have been potential errors during my study, resulting in somewhat skewed results. For example, during testing, trials may have run at different intervals, creating inconsistent results. Also, human error during Test Case 5 (the one testing IRES's efficiency with actual usage habits) could have thrown off the final results. Some of these errors would have been uncontrollable, such as the degradation of my laptop's Lithium-Ion battery over time. Not to mention that to get more definitive analysis on cost reduction, I would need to obtain a sample size larger than the one I currently have which is at 15 responses, but not just a larger sample size, but also a wider spread, focusing not just on Western Pennsylvania, but trying to get a reach into other states, where professions and lifestyles might differ, resulting in different computer usage statistics. Of course, more data is always better, so if I were to do this experiment again, I would collect more data, running 4 or 5 trials for each Test Case; in fact, this is just of one of my plans for the future.

Plans for the future include encapsulating the screen control app so the user does not have to see, it happens behind the scenes and expanding upon the GUI to enhance the user experience and allow user programmability are plans for the immediate future. Slimming down the logic board and the case are also plans further down the road in order to create a more practical and aesthetically pleasing compact device. A future goal would be to work with OEMs for possible integration into a computer itself to provide a hassle free experience to the user.

Conclusion:

In conclusion, at the end of study, I was able to prove my hypothesis correct. If I can create a device that can control when the screen and CPU are on, then I will see a decreased use in electricity. I successfully created and coded IRES, which was the device that controlled when the screen and CPU were on, and I use various analytic techniques, such as my survey and additional research, to conclude that IRES directly causes a decrease in electricity consumption of a computer. Using IRES, I got a 44.64% mitigation in energy consumption, which could lead to approximately $4.6 billion in savings each year, and also prevent approximately 34 million tons of CO_2 from entering the atmosphere yearly.

The practicality of IRES is extremely high. For a device that costs approximately $30 to build, $20 for the Arduino and $10 for the IR sensor (add $2 if the user does not have a USB cable lying around), it has such a

great impact, but it's not only that; everyone that I interviewed with a laptop said that they would be interested in better battery life, meaning that people would actually use IRES if they could. If it were to be mass produced, the price of manufacturing would go down significantly, resulting in a retail price of under $30. But creating a version of IRES that could clip onto the top of computers would only be one step (what I did with the 3D printer); working with companies for possible integration into a computer would be another, much larger, step. All in all, the creation of IRES is just the beginning into more efficient energy saving techniques, allowing for a reduction in energy consumption in world with ever growing energy demands.

Acknowledgments

Keith Banks,

North Allegheny Intermediate High School – Technology Education Department

Works Cited

"How much carbon dioxide is produced by burning gasoline and diesel fuel?" *U.S. Energy Information Administration*. US Department of Energy, 2015. Web. 3 Feb. 2015. <http://www.eia.gov/tools/faqs/faq.cfm?id=307&t=10>.

"How much carbon dioxide is produced per kilowatthour when generating electricity with fossil fuels?" *U.S. Energy Information Administration*. US Department of Energy, 2015. Web. 3 Feb. 2015. <http://www.eia.gov/tools/faqs/faq.cfm?id=74&t=11>.

"How much power does a computer use? And how much CO2 does that represent?" *Energuide*. Energuide, 2015. Web. 2 Feb. 2015. <http://www.energuide.be/en/questions-answers/how-much-power-does-a-computer-use-and-how-much-co2-does-that-represent/54>.

Mahesri, Aqeel, and Vibhore Vardhan. "Power Consumption Breakdown on a Modern Laptop." *Springer Link*. Springer International, 2015. Web. 23 Jan. 2015. <http://link.springer.com/chapter/10.1007%2F11574859_12>.

"Monitor Laptop and UPS Batteries." *PassMark BatteryMon*. PassMark Software, 2015. Web. 5 Jan. 2015. <http://www.passmark.com/products/batmon.htm>.

O'Reilly, Dennis. "Calculate Your PC's Energy Use." *CNET*. CBS Interactive, 2014. Web. 11 Nov. 2014. <http://www.cnet.com/news/quirky-ge-unveil-new-affordable-smart-home-line/>.

"Power Management for a Laptop Computer with Slow and Sleep Modes." *Google Patents*. Google, 24 Nov. 1992. Web. 2 Mar. 2015. <https://www.google.com/patents/US5167024>.

"Recent Data." *U.S. Energy Information Administration*. U.S. Department of Energy, 2015. Web. 2 Feb. 2015. <http://www.eia.gov/electricity/>.

Rudenko, Alexey, et al. "Saving Portable Computer Battery Power through Remote Process Execution." *ACM Digital Library* (1998): n. pag. Web. 2 Mar. 2015. <http://dl.acm.org/citation.cfm?id=584008&preflayout=flat#source>.

"Top Ten Countries with Highest number of PCs." *Maps of World*. Compare Infobase, 2015. Web. 2 Feb. 2015. <http://www.mapsofworld.com/world-top-ten/world-top-ten-personal-computers-users-map.html>.

Venkatachalam, Vasanth, and Michael Franz. "Power Reduction Techniques for Microprocessor Systems." *ACM Digital Library* (2005): n. pag. Web. 2 Mar. 2015. <http://dl.acm.org/citation.cfm?id=1108957>.