Miami University ECE 448 Senior Design Project Report

Power Measurement of a Computing System (Fall 2021)

By: Sam Rutschilling ('22), Jordan Smith ('22), & Owen Hardy ('22)

Advisors: Dr. Peter Jamieson & Dr. Mark Scott

Table of Contents

Abstract

The original goal of this project was to create a power measurement system that could log the power consumption of various peripherals plugged into a power strip. This would give an idea of how much power each piece of equipment was using at any given time. After some time had passed in-between semesters, our team and advisors decided to change the scope of the project to develop a hardware solution that could measure the power consumption of various computing tasks.

With this new goal in mind, we developed a list of various computing operations we wanted to measure the power consumption for. This list included several simple scripts that performed different functions including sending email messages, writing to files, accessing webpages, and performing basic arithmetic operations. This allows us to understand the power consumption of small computing tasks and their cost. What our team wanted to understand from our data collection was how much power these different operations consume and determine tasks that required significantly more energy than others.

Introduction

Our team's focus this first semester of the project was to determine the feasibility of measuring power draw from a computing system as a whole. Very early on, a decision was made to conduct our experiments on a Raspberry Pi SBC. We weren't sure if we would be able to observe power changes when running a script, so we thought it best to initially try on a Raspberry Pi since they have very low power consumption for a system running a full OS. A typical desktop PC may **(1)** consume several hundred watts at idle and **(2)** across several different DC voltage rails. These two factors could make data collection on this sort of system complex, so we stuck with the Raspberry Pi as a proof of concept.

Project Background

Our team and advisors came up with a small list of scripts for which we wanted to monitor power consumption. This list can be seen below.

- Arithmetic (add/subtract, multiply, divide)
- Sending emails
- Writing to disk (SD card on RPi)
- Printing to console
- Pinging
- Python vs. Objective-C

As mentioned earlier, this semester's focus was a proof of concept. We knew that eventually, we'd want to monitor power consumption for more complex computing events such as the autosuggest feature on the Google search bar. We wanted to stick with more rudimentary operations so we could more easily validate our findings. Once we had our test parameters set for what sorts of operations we'd be performing on our system, we'd have to move on to research methods of measuring power draw in our specific test environment that we'd defined with our scripts.

Project Research

We needed to research the power draw characteristics for our system which was a Raspberry Pi 3B+ running Raspbian OS. The Pi was to be interfaced over the network via SSH so the only connection to the Pi besides the power supply was an ethernet cable. No external monitors/mice/keyboards we plugged into the system. To determine the power draw characteristics with the Pi, we simply needed to measure the current draw of the system at idle as well as while one of any of the test scripts was running. This would tell us the current range we should expect to be measuring with our hardware solution.

A Fluke 87V multimeter was set to its current measuring function and connected in series with the 5V lab bench power supply to the Raspberry Pi's power input. Once the Pi had booted and current fluctuations had stopped, the Fluke measured a current draw of approximately 470mA. According to the Pi's online user guide, this falls within the expected value. We ran one of our basic arithmetic scripts and measured a maximum current draw of approximately 700mA. These findings did surprise us as we weren't expecting such a significant increase in power draw during the execution of the script. This made our job easier since we knew that our current measuring sensor wouldn't have to have as high of granularity as we would have thought.

Through some research on various electronics parts distributors, we found the Texas Instruments INA219 current monitor IC. The IC's datasheet defines a maximum current measurement rating of 3.2A and a maximum voltage of 26V. This falls well within our test environment characteristics which we saw draw up to 700mA at 5V. The datasheet reports a current measuring resolution of 0.8mA which should be plenty of accuracies when we saw our system current spike from ~470mA to ~700mA of. The INA219 operates with a 3 to 5.5V power supply and communicates over I2C. These are all ideal characteristics for our test environment.

Solution Implementation

With our current measuring IC selected, we had to develop our hardware implementation. We knew that we would have to have another device that served as our data collection system. We knew this would have to be isolated from the Raspberry Pi since if we used that to perform the data logging, our power draw would be artificially elevated from this operation running on the Pi at the same time we were running our test script. Luckily, the INA219 interfaces over I^2C and there is an Arduino library for interfacing with this sensor. An Arduino Uno was used to read the data from the sensor and print out the data over USB serial to a PC where it was logged using a script that utilized the PySerial library.

We decided to use SQLite to store the data instead of other formats such as .txt (tab-delimited text) or .csv (comma-separated value) for several reasons: SQLite stores data in columns and tables so it is much less likely to become corrupt if it is incorrectly closed or not closed at all. SQLite also allows easy storage of the date/time value as well, which allows us to store the exact time of a data point. SQLite also allows easy searching and calculation of averages of data sets as it allows the use of all common SQL (structured query language) queries and commands. This allowed calculation of the average idle and operation values to be done using just a single query line in the open-source DB Browser for SQLite. These same queries were also used to calculate the time values for each of these scripts. That data was then used to calculate the power consumption for each computing event.

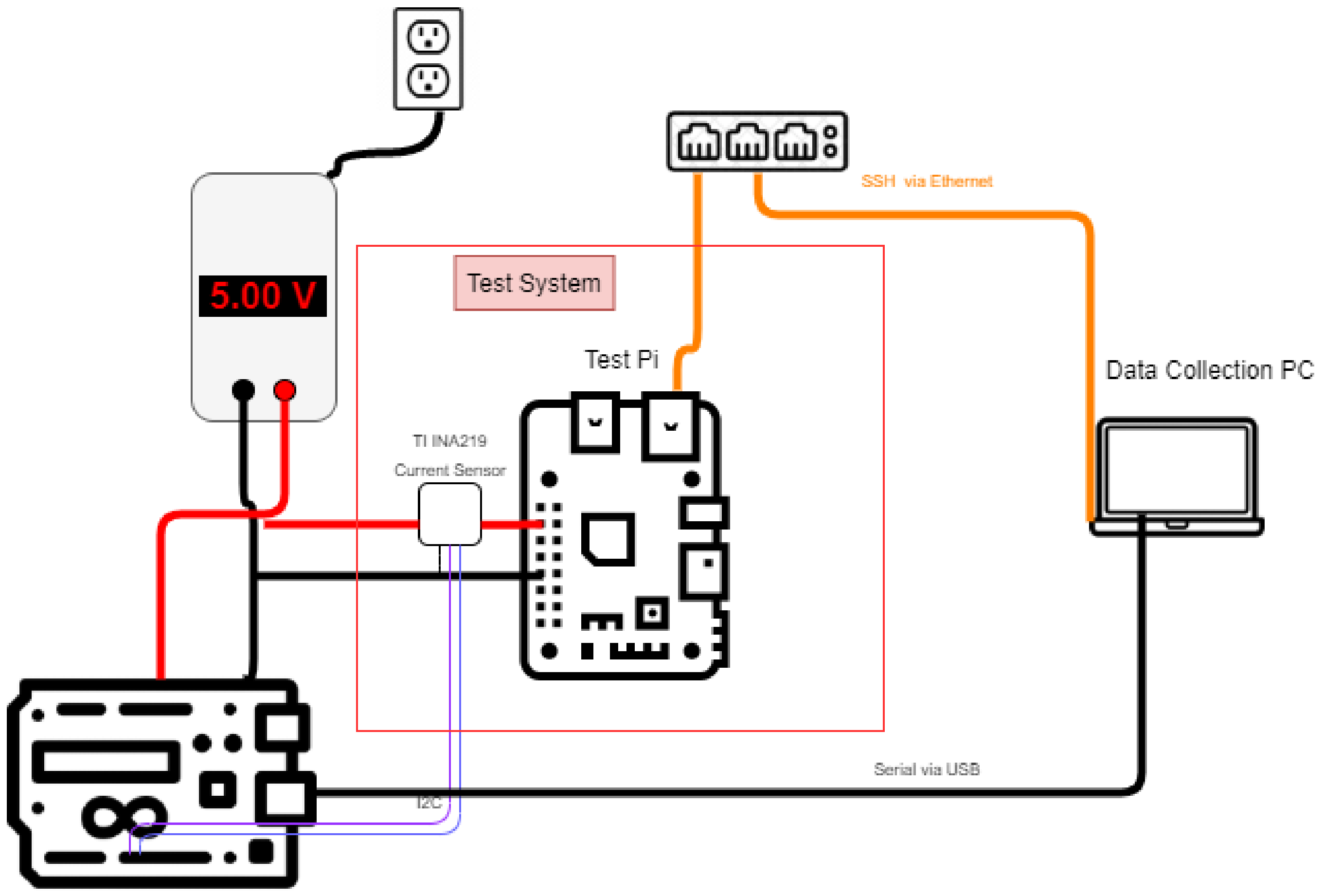

Figure 1: Diagram of the complete computing system. The measured system is the Raspberry Pi 3B+ at the center of the layout. The Arduino is used for data collection, and the router is used for communicating with the Raspberry Pi as well as planning for future use.

Most of the data collected was collected with a timing of 10ms, which means for every second there are 100 data points, which means that there are several thousand data points for most of the trials. This frequency of data collection is high enough to create enough data so that we can be confident our results are a true representation of the system's power consumption, both at idle and while performing a computing operation.

In the recorded data for each of the trials, some jumps and increases can be seen in the data, especially when viewed visually as in Figure 2 below. These are background tasks operating on the OS level and should not cause an issue as they are present in both the average baseline power consumption and the average operation power consumption.

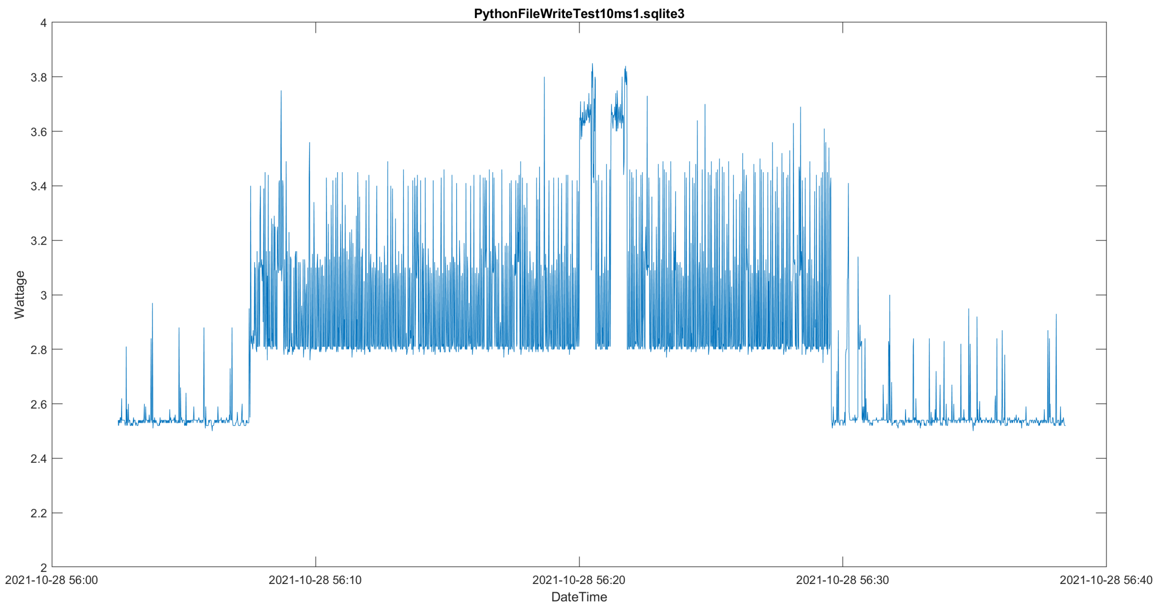

Figure 2: Shows a visual plot of the data from one of the trials, specifically a file write test using Python. The jumps in the figure can be attributed to background OS calls in the Debian-based Raspberry Pi OS. The areas of lower consumption at the beginning and end of the figure are the standby power consumption, and the increased "plateaued" area in the center is the power consumption during the Python file write test.

The recording procedure utilized for all of the tests was quite simple. The test Raspberry Pi was powered on and connected to our test ethernet network router. The raspberry pi is powered through the INA219 power measurement IC. Next, an SSH (secure shell) connection was made over the network connection to the Raspberry Pi using Visual Studio Code Remote SSH. Through VSCode, the test scripts were uploaded to the pi and modified. Then for each test, the data collection python script was started on the data collection PC. After several seconds of standby data were collected, the test script for the Raspberry Pi was started through VSCode. The performance of both the recording script and the test script was monitored during the test to make sure there were no unexpected abnormalities. After the test script was done running, several seconds of standby data at the end were collected before stopping the data collection script. Both the SSH connection and the data collection were done using the same laptop. This process was repeated several times for each test operation to get a more accurate average of the power consumption.

Knowing the increase in power while an operation was functioning is useful, but because the events happen so quickly and we used repeated functions to show their power consumption the results need to be normalized to each individual operation. This is because some operations take longer than others to complete. While doing trials, we chose arbitrary values of the number of operations so that each trial would be at a minimum, approximately 0.25s long. This required significantly more trials for some of the operations than others.

To calculate the average time each individual computing operation took we simply took the average time it took for a recorded trial to take place, which is calculated from our SQLITE data and divided it by the arbitrary number of operations we tested during that trial. This calculated value is the average time per individual trial. Multiplying that value by the average increase in wattage between the standby and operation recorded for each trial gives the power consumed by each operation in watts per second. That value is then multiplied by a constant value of 2.778 E-6 to output the results in kWh, which is the industry standard value power is billed to consumers in.

Data Findings & Interpretation

ID	Script/operation	Average increase in power from idle (W)	Power use per computing event (W/s)	Energy use per computing event (kWh)
A	Python LAN Ping & Print	0.25	2.860 E-3	7.944 E-10
B	Python LAN Ping No Print	0.19	1.968 E-3	5.465 E-10
C	Python Addition & Print	0.904	3.478 E-5	9.662 E-12
D	Python Subtraction & Print	0.896	2.007 E-6	5.574 E-13
E	Python Multiplication by Constant 2 & Print	0.145	4.725E-4	1.312 E-10
F	Python Division & Print	0.189	7.652 E-4	1.426 E-10
G	Python Addition No Print	0.447	1.154 E-6	3.206 E-13
H	Python Subtraction No Print	0.643	1.057 E-6	2.9364 E-13
I	Python Multiplication by Constant 2 No Print	0.195	4.829E-4	1.341 E-10
J	Python Division No Print	0.124	2.785 E-4	9.029 E-11
K	Python Write To File	0.420	1.184 E-6	3.290 E-13
L	Objective-C Write to File	0.414	7.276 E-8	2.021 E-14

In our tests, the least consuming Python operations the add and subtract functions that did not print to the console, tests **G** and **H** in the table above which consumed 3.206 E-13 kWh and 2.936 E-13 kWh respectively. This is in comparison to the highest consuming Python operation we measured, which was the Python ping and print to console trial. This trial, **A** in the table, consumed 7.994 E-10 kWh per computing operation. All of these tests were run with the same Raspberry Pi 3B+ and the same Python 3 interpreter. The large disparity in the power consumption of these operations can largely be attributed to the large difference in resources the two operations will take. The ping and print operation will have OS calls for networking, domain name services, printing to the screen, as well as counting calls. The Python add and subtract operations, only have OS calls for the addition and subtraction operations, which is why their power consumption is so similar and significantly smaller than the other Python operations.

Referring to events **K** and **L**, we can observe a much smaller power use for the Objective-C file write test at 2.021 E-14 kWh used compared to the Python file write test at 3.29 E-13 kWh used. These findings make sense since Python scripts are executed in a virtual environment whereas Objective-C does not, so generally, Objective-C is considered to be a faster language.

Referring to the non-printing scripts **B**, **G**, **H**, **I**, **J** and the printing scripts **A**, **C**, **D**, **E**, **F**, we can see that printing to the console uses roughly about 55% more energy (taken as an average across all 5 test variants) than its non-printing equivalent. Again, this findings makes sense since printing to the console requires more OS calls. For one, since we're interfacing with the Pi via SSH, the print statement needs to be sent through than channel rather than something like an external display connected via HDMI.

The overall significance of this data is that we were able to successfully detect and measure changes in current consumption when various computing tasks were executed. Our proof-of-concept was a success which will lead us into next semester's objectives where we want to measure energy use for more abstract computing tasks (see Future Project Goals).

Future Project Goals

So far, the majority of what we have done is setting up the problem and basic testing to ensure our setup works as we'd expect. In the future, we plan to move on to the next step for the project, that is, the meat of the project. We will be setting up an email server and a script to send emails. Using these, we will be testing the different kinds of content that can be sent via email and how they affect power consumption.

After gathering data from the email servers, we will be looking into the Google Search Bar. Or, more specifically, the auto-fill feature of the search bar. The auto-fill feature of the Google Search Bar is a process that gives the user options to fill in what they've already typed. For instance, if the user typed "How to" into the search bar, Google might suggest that the query they're searching for is "how to commit murder", or "how to hide a body", or "how to remove the evidence". Our goal in this instance is to determine how much power having that auto-fill feature on at all times will take up over not having it on at all.

Conclusion

In this paper, we have presented an isolated system running on a Raspberry Pi. Using this Raspberry Pi and an INA219 current monitor, we have proved that it is feasible to measure the power draw of a computing system as a whole. However, we have only tested this concept using basic processes, and we plan to test further using more complicated processes.

The processes we have tried are basic arithmetic, server pinging, and file writing alongside each of these processes accompanied by a print to console and a comparison between Objective-C and Python. Our results have shown that Objective-C consumes less power than Python, making Objective-C a more power-efficient programming language. Arithmetic such as addition and subtraction cost less than arithmetic such as multiplication and division with pinging costing more than addition and subtraction as well. Testing the process with printing to the console increased the power draw even further. So far, our collection of data has shown that there is a difference between when the system is running on idle and when the processes are running, the processes increasing the consumption of the system. This difference is rather small, but the power draw can add up over time if the processes are run continually.

Reflection

It is our responsibility as engineers to ensure that the systems we create are power efficient. Using power costs money and an inefficient program running on an inefficient system implemented into an inefficient machine will compound to cost a lot. It is not right to design systems this way, to intentionally make it so it consumes more power than is reasonable. Our goal, our job, as engineers is to design systems that a customer can rely on and doesn't cause them a net loss overall.

It is important to never stop learning. Learning is one of the most important facets of life that allows us as the human race to advance society, technology, and, even, our personal beings. If we stop learning, we'll never improve, we'll be stuck as we are, never changing. It is also important to continue to learn to keep our minds sharp and engaged, to prevent them from dulling and losing information.

A broad education is needed for an engineer to be able to design products that can work for a wide range of people or be able to design products for a wide range of people over-designing products for one type of person. What works for a computer engineer might not work for an English major. What works for an American might not work for an Indonesian. We need to be able to create solutions for problems worldwide, and if we don't have the broad education needed to understand nations outside of our own, our solutions won't integrate very well for those other nations, or, perhaps, they could cause problems instead of helping.