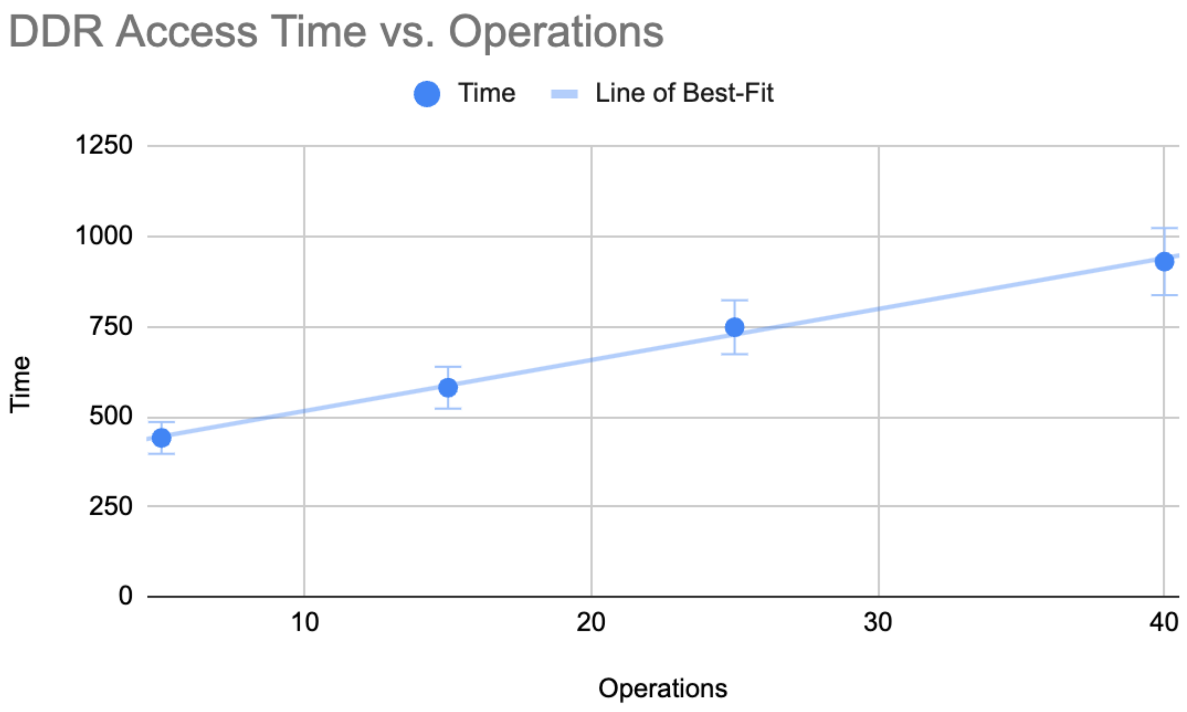
ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

PART 1

The regression estimate of the overhead from section 4 of the lab was determined to be 343 units. This was determined by running the memory access function 5 times, and recording the time it took to complete. This process was repeated for 15, 25, and 40 function repeats. The data from each of the runs is graphed below. Using linear regression, we can calculate the line of best fit to be $y = 14.1x + 343$, where x is the number of memory access function calls and 375 would be the overhead. Now we can use this calculated overhead to offset the DDR access timing measurements from Part 2.



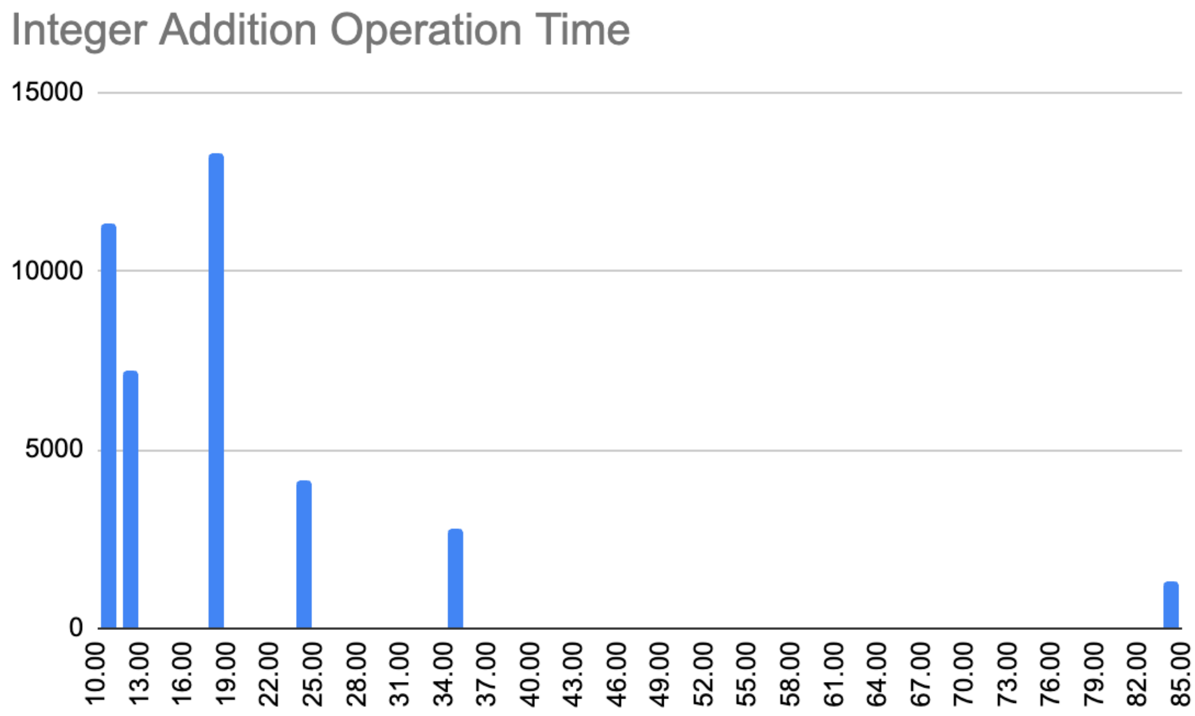
ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

PART 2

Timing of addition using integers

To calculate the timing of addition using integers, we ran a simple calculation (c = a + b, where a = 3 and b = 4) some N times, recording the time it took, then repeating the whole thing 10,000 times. This was done for N = 5, 15, 25, 40. All the resulting times were divided by N to find the actual time per operation, and aggregated into the histogram below.



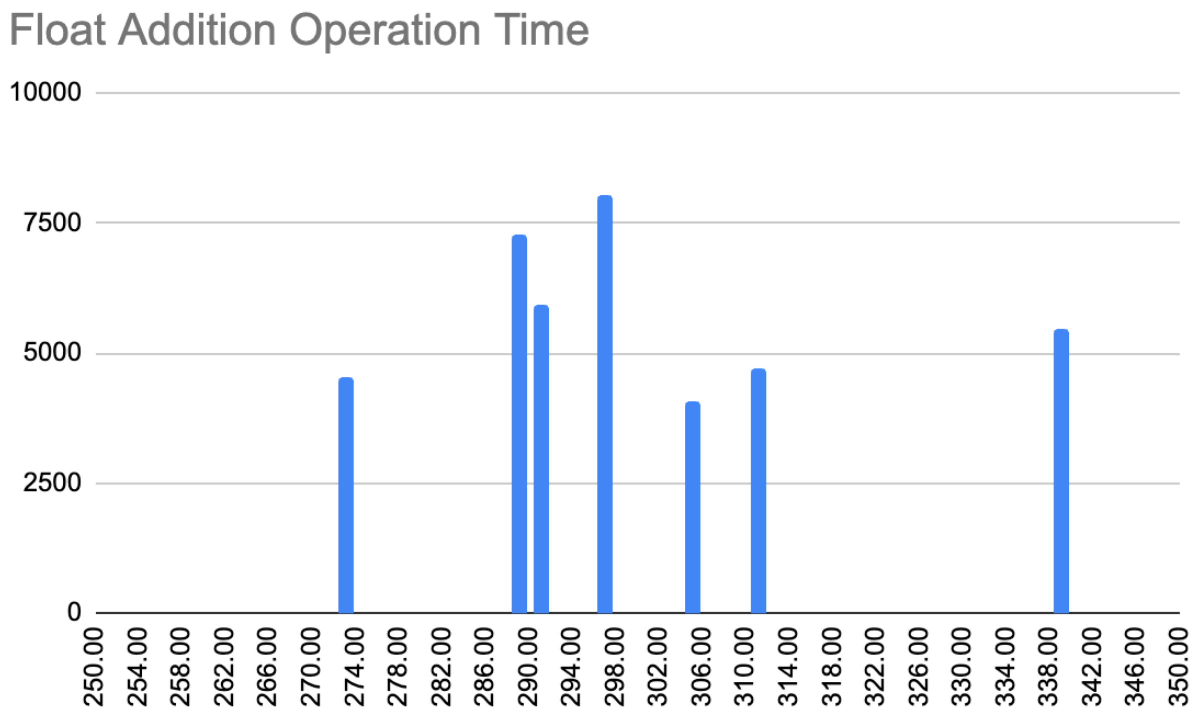
The average time was 18.8624 with a standard deviation of 13.7843.

ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

Timing of addition using floating point

We followed a similar process to calculate the timing of addition using floating point numbers, just using a different calculation (f = d + e, where d = 3.14 and e = 2.71). Again this was repeated N times, the time taken was recorded, and then the whole thing was repeated 10,000 times. This was done for N = 5, 15, 25, 40, with all the resulting times divided by N to find the actual time per operation. The aggregated results are shown in the histogram below.

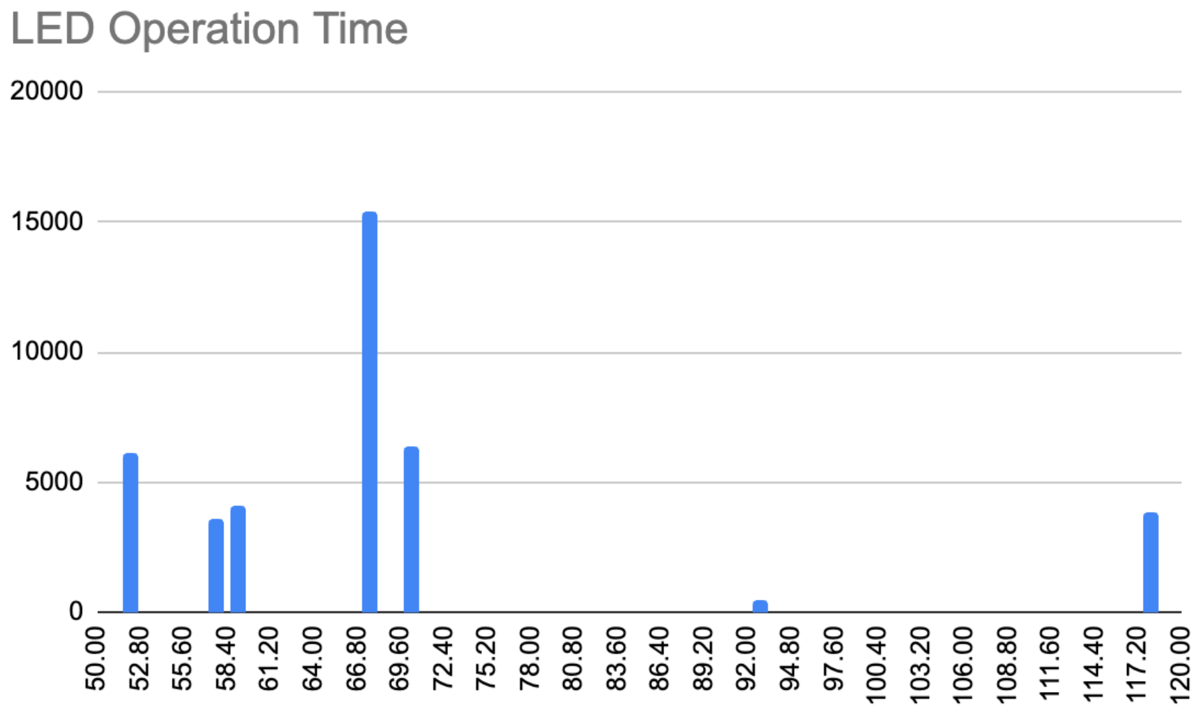


The average time was 299.9202 with a standard deviation of 18.4569.

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

Timing of writing the LEDs to turn on or off

To calculate the timing of writing the LED's to turn on, we wrote to the LED GPIO pin N times, recording how long it took, and then repeated that 10,000 times. Just like before, this was done for N = 5, 15, 25, 40, again dividing all the resulting times by N to find the actual time per operation, and aggregated them into the histogram below.



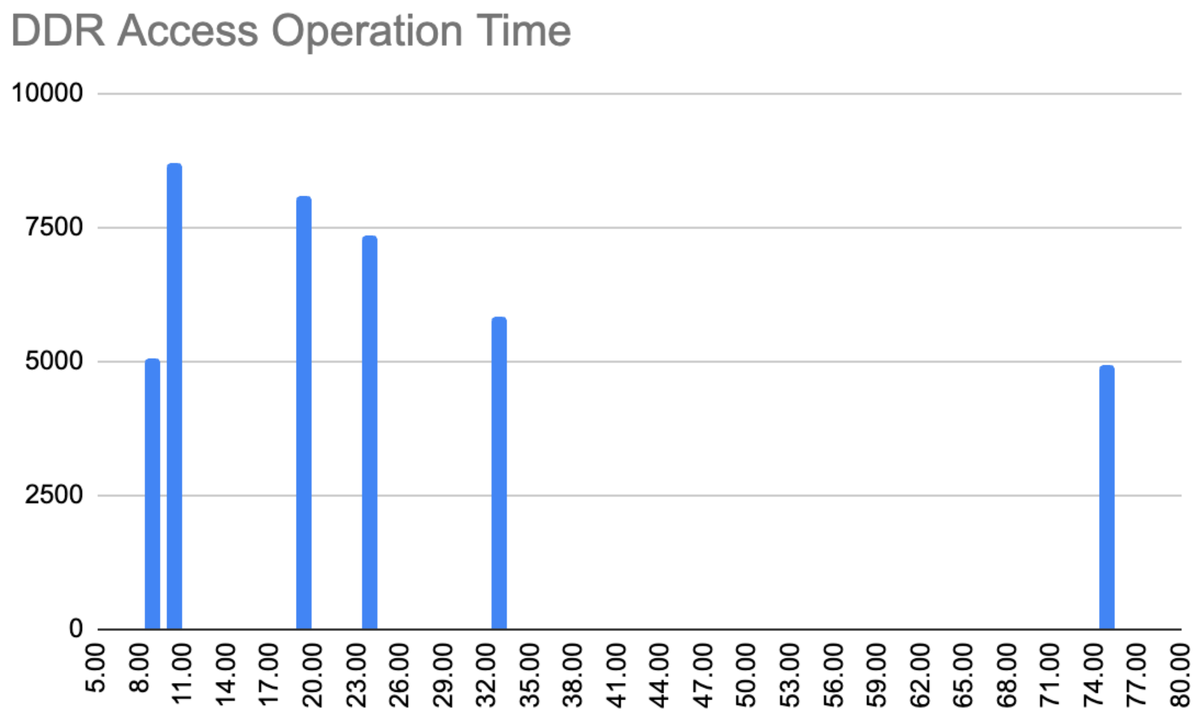
The average time was 69.0062 with a standard deviation of 17.4924.

ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

Timing of reading a word from the DDR2 memory at a random location

To calculate this timing, we first generated a random address inside the size of the buffer we created. Then we accessed this location in the buffer N times, recorded the time taken, then repeated the whole process 10,000 times. This was done for N = 5, 15, 25, 40, with the estimated overhead for memory access that we found in Part 1 subtracted from each of the measurements. Then we divided all the resulting measurements by N to find the actual time per operation. The aggregated results are shown in the histogram below.



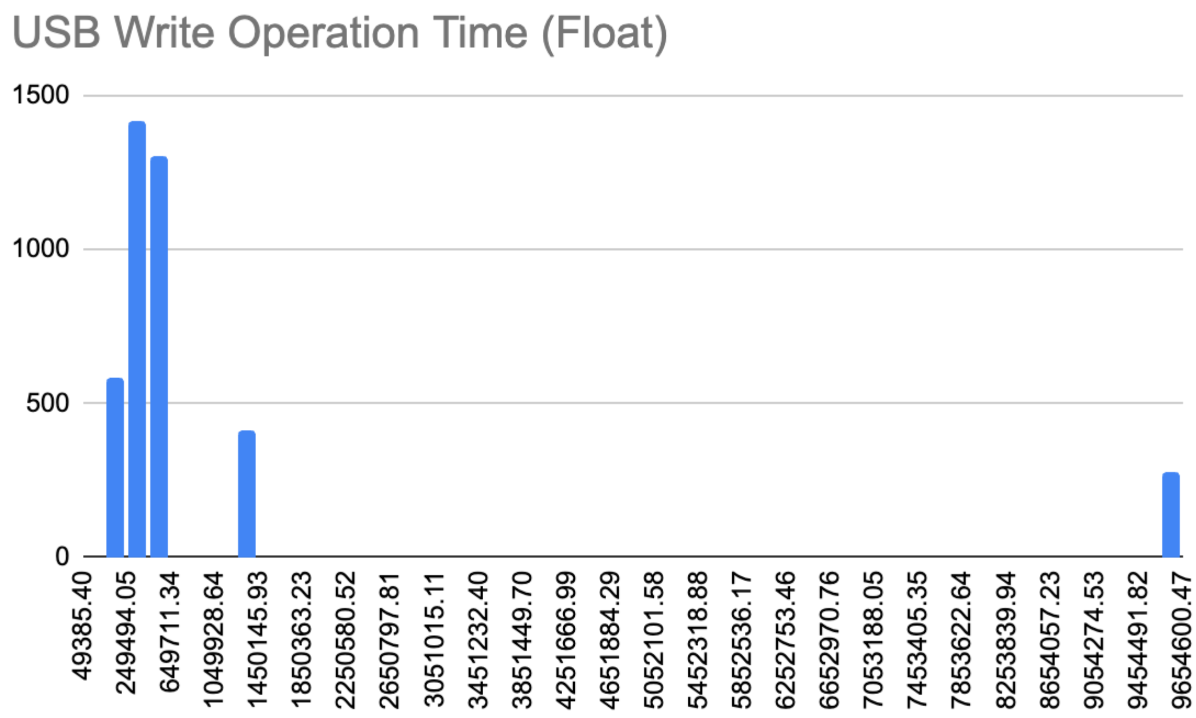
The average time was 25.8245 with a standard deviation of 20.0871.

ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

Timing of writing to the USB Port (floating point)

To calculate the timing of writing a floating point number to a USB port, we ran `printf("%f\n\r", 0.1)` N times, recorded the time taken, then repeated the whole process 1,000 times (instead of 10,000 for the sake of time). This was done for N = 5, 15, 25, 40. The results were all divided by N to find the actual time per operation. The aggregated results are shown in the histogram below.



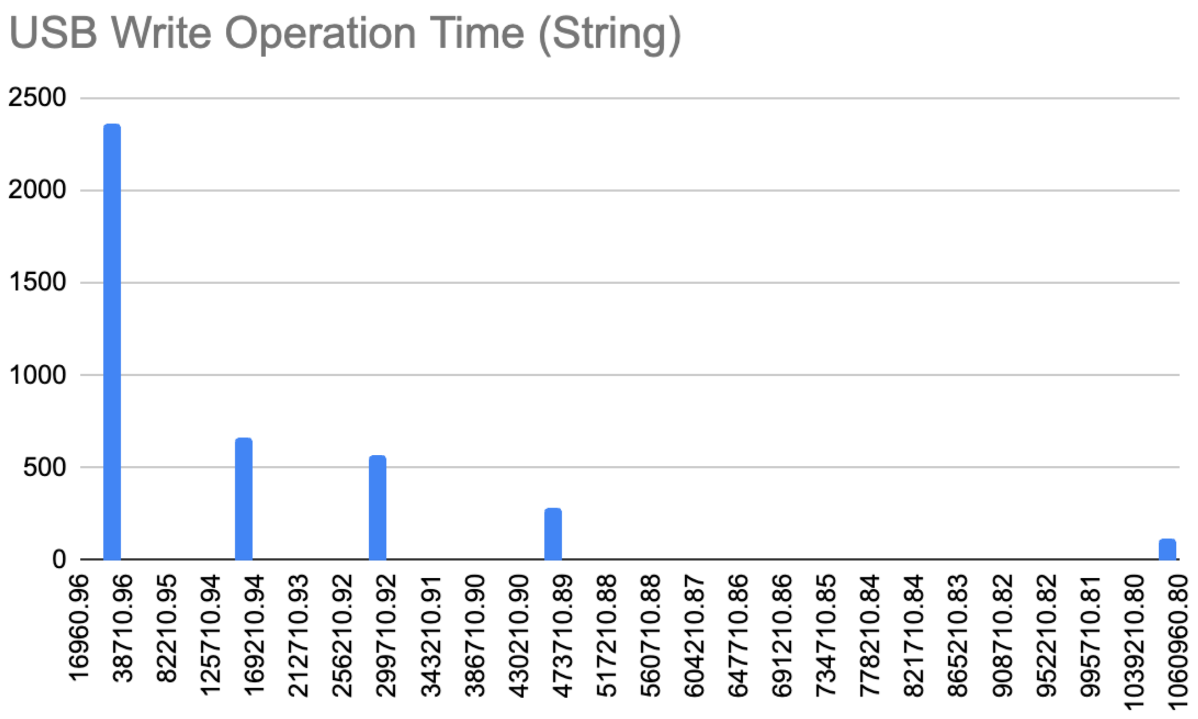
The average time was 1,128,391.274 with a standard deviation of 2,397,689.91.

ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

Timing of writing to the USB Port (string)

To calculate the timing of writing a string to a USB port, we ran **`xil_printf`**`("helloworld\n\r")` N times, recorded the time taken, then repeated the whole process 1,000 times (instead of 10,000 for the sake of time). This was done for N = 5, 15, 25, 40. The results were all divided by N to find the actual time per operation. The aggregated results are shown in the histogram below.



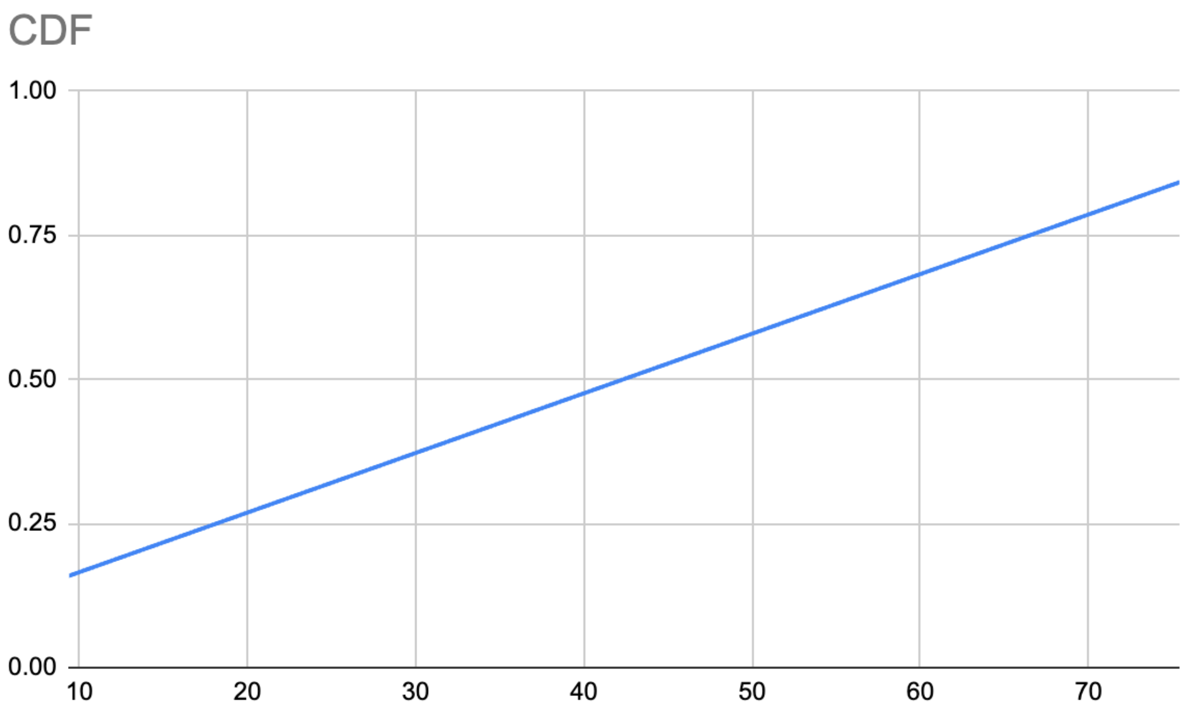
The average time was 149,378.5851 with a standard deviation of 208,063.0432.

ECE 153A, Friday @ 4:00pm
Lab #1A

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

PART 3

The expected value of the DDR access timing was 25.8245. The 95% interval for this measurement is (25.5875, 26.2489).



PART 4

The average time of the integer addition operation was 18.8624 with a standard deviation of 13.7843. The average time of the floating point addition was 299.9202 with a standard deviation of 18.4569. Both of these operations have relatively low variance. The integer addition was significantly faster (on the magnitude of x10) than the floating point addition. This makes sense since it's a lot easier to not only represent but also interact with integers than it is with floating point numbers. The CPU would have dedicated hardware in the form of ALUs that can interact

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

with integers but floating point numbers would require some software virtualization, which tends to make operations quite a bit slower.

The average time for the LED write operation was 69.0062 with a standard deviation of 17.4924. This is still a relatively low variance, but a higher operation time. This makes sense since interacting with the LEDs requires sending signals to the GPIO pins. This adds some overhead since we are now also limited by the characteristics of the GPIO hardware itself.

The average time for the DDR access operation was 25.8245 with a standard deviation of 20.0871. This had a much larger variance compared to the previous operations. This was also visible in the histogram since the data had higher spread. This could be due to the nature of memory access on the CPU. It's a more involved process than simply calculating some arithmetic, which qualifies the increased spread.

The average time for the USB float write operation was 1,128,391.274 with a standard deviation of 2,397,689.91. The average time for the USB string write operation was 149,378.5851 with a standard deviation of 208,063.0432. Both the average times and the variances were a lot greater for these two than the rest of the operations. Since these operations dealt with actually sending data back to a host PC over the USB, they would also be affected by the performance of the host PC. This could be why the variance was so high as the host PC was somewhat of an unknown random variable during these operations. What was particularly interesting was that writing a string to the USB was a lot faster than writing a floating point integer. The reason for this is that the floats were serialized using formatted strings. So there would be extra work being done to convert the floats to a character array before sending them to the USB versus just directly sending a predetermined set of characters to the USB.

PART 5

Bharat Kathi
bkathi@ucsb.edu
Lab Partners: Joshua Thomas

There were a few outliers in the data we collected. These were time measurements that were over 3 times the average for that operation in some cases. When looking at the raw data, we can see that the first measurement for any batch is always one of these high numbers, and immediately goes back down for the next measurement. A reason for this could be that the CPU may be caching certain data such as the numbers we are using in our operations. This saves the CPU having to look in memory for the variables all the time as it fetches the same variables for subsequent fetches. Looking through the data again, we also saw some large timing numbers repeatedly throughout the data. We set up each operation in its own function, and called this function from the main for-loop. This could lead to some additional overhead as the CPU needs to find the function code and return back to the original flow. This could be a source of variance across all the operations.

PART 6

The operation with the smallest standard deviation was adding two integers (13.7843). The operation with the largest was writing a float to the USB port (2,397,689.91). It makes sense that the integer addition along with the float addition and writing to led operations, have significantly less variance than the reading from memory or writing to USB port operations. The first 3 operations only deal with manipulating static components on the board, which resulted in more consistent operations. For example, the ALU used to add the integers is very consistent in its performance. Reading from memory, however, has a lot more overhead and can be affected based on the current state of the cache on the CPU, leading to higher variation in the data (≈ 21 vs ≈ 13-15 for the first 3). Finally, while the first 4 all deal with the FPGA board itself, writing to USB requires interfacing with the host PC. This results in a lot more moving parts, each of which adds its own overhead and variable delays.