ECE 153A Lab 1A: Timing Analysis

Luke Lewis | Eduardo Olmos

Purpose

In this introductory lab for the course, the objective was to familiarize ourselves with the developmental process for the Artix-100T development board. This involved modeling the hardware design of a microblaze processor interconnected with a few peripherals in the Vivado design suite from Xilinx. We then utilized the Xilinx SDK to write C code that runs on our board. Particularly, this lab involved timing various operations in terms of clock cycles and statistically analyze it with consideration that embedded systems are targeted at tightly constrained environments.

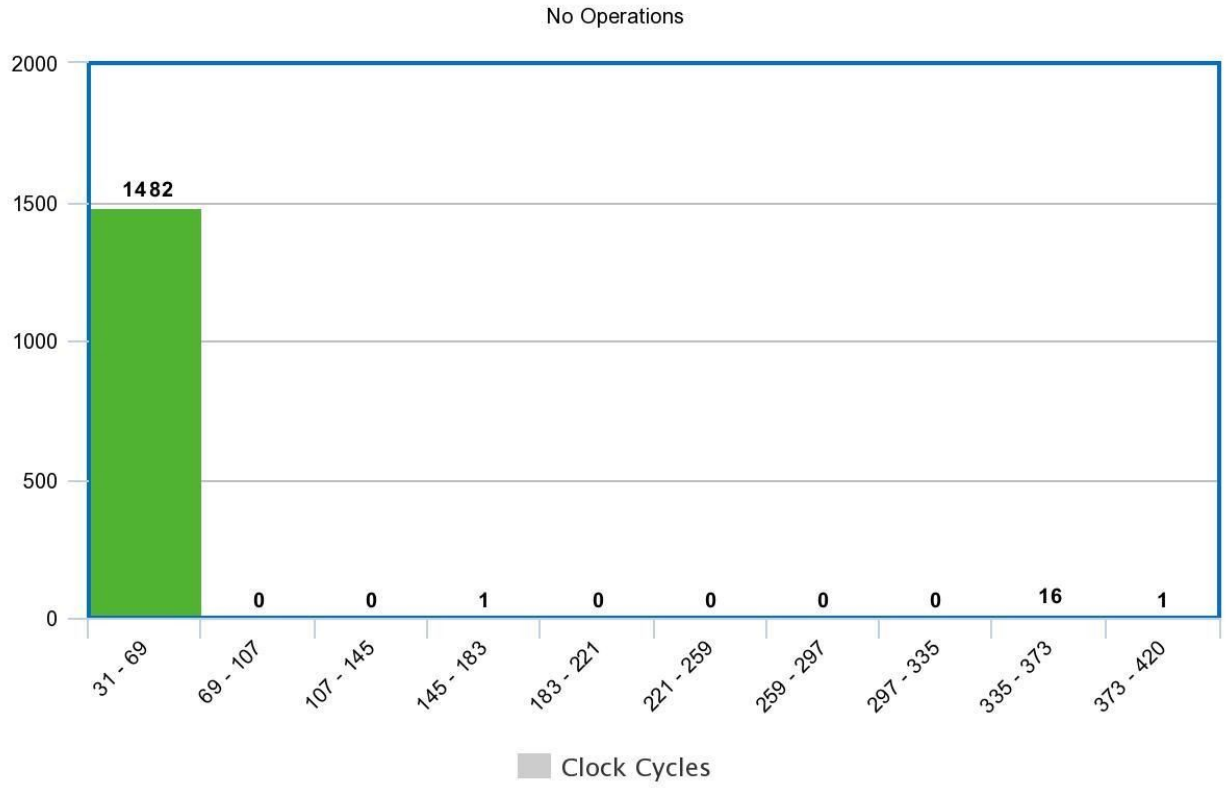
Operations Measured

1) Writing a floating point number using *printf()*
2) Writing a string of 10 characters using *xil_printf()*
3) Addition using integers
4) Multiplication using integers
5) Addition using floating point
6) Multiplication using floating point
7) Turning on LED
8) Turning off LED
9) Reading a byte from the DDR2 memory
10) Reading 8 bytes from the DDR2 memory

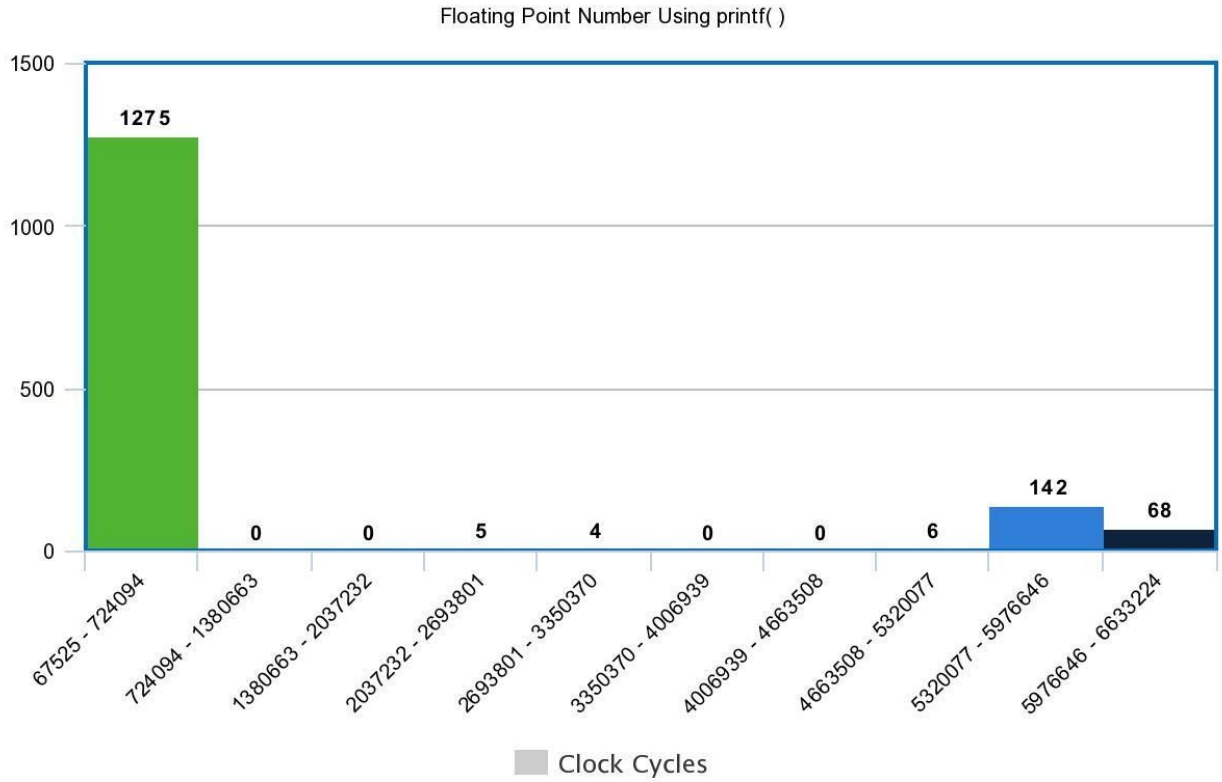
Assumptions

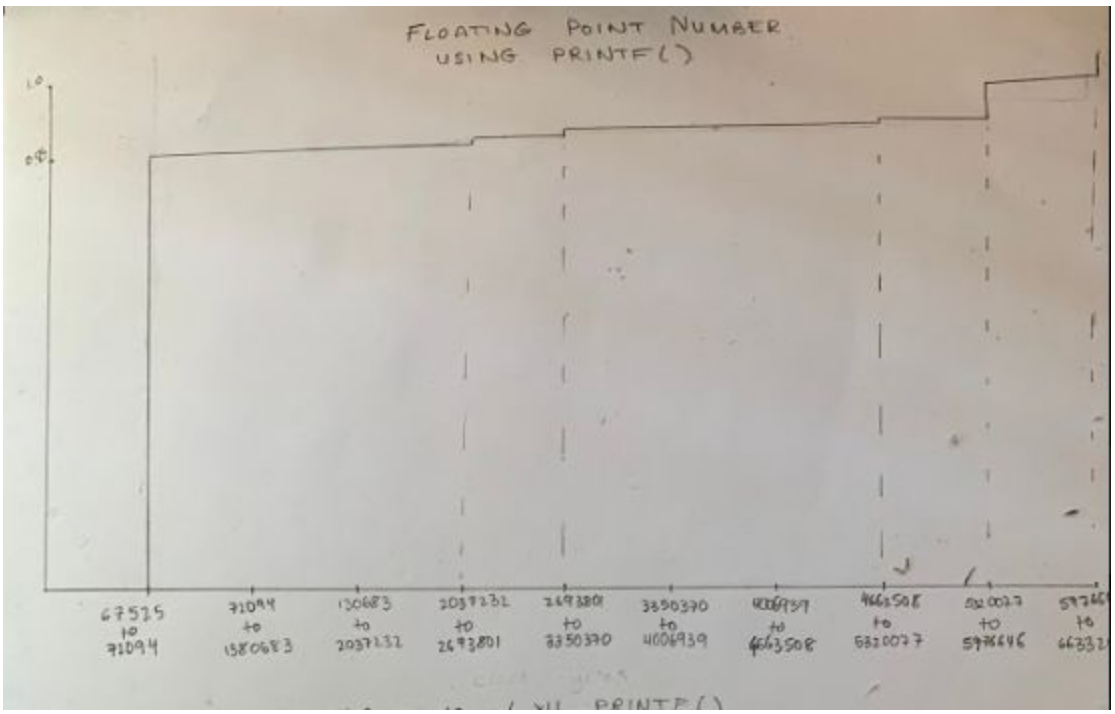
The way we will be measuring clock cycles will be through the use of a timer. Essentially, for every trial with the operation we want to measure we will have to stop the counter timer (if it is already running), enable the timer, perform the operation, and then record the value. This process loops for the number of trials we want to perform. Before we can start taking measurements, we must account for the activity for when no operations are running. As you can see by observing the histogram below, the processor does not sit idle when given no operation. As detailed in the lab, another method called extra_method( ) is called before we start measurement trials. This method enables an interrupt timer to be handled and is something we need to take into consideration. Measurements in the histogram below show that for 98.8% of the 1500 trials of no operations, clock cycles were in the range of 31-69 and 1.13% of outliers accounting for a greater than tenfold increase in cycles. This means that we also cannot

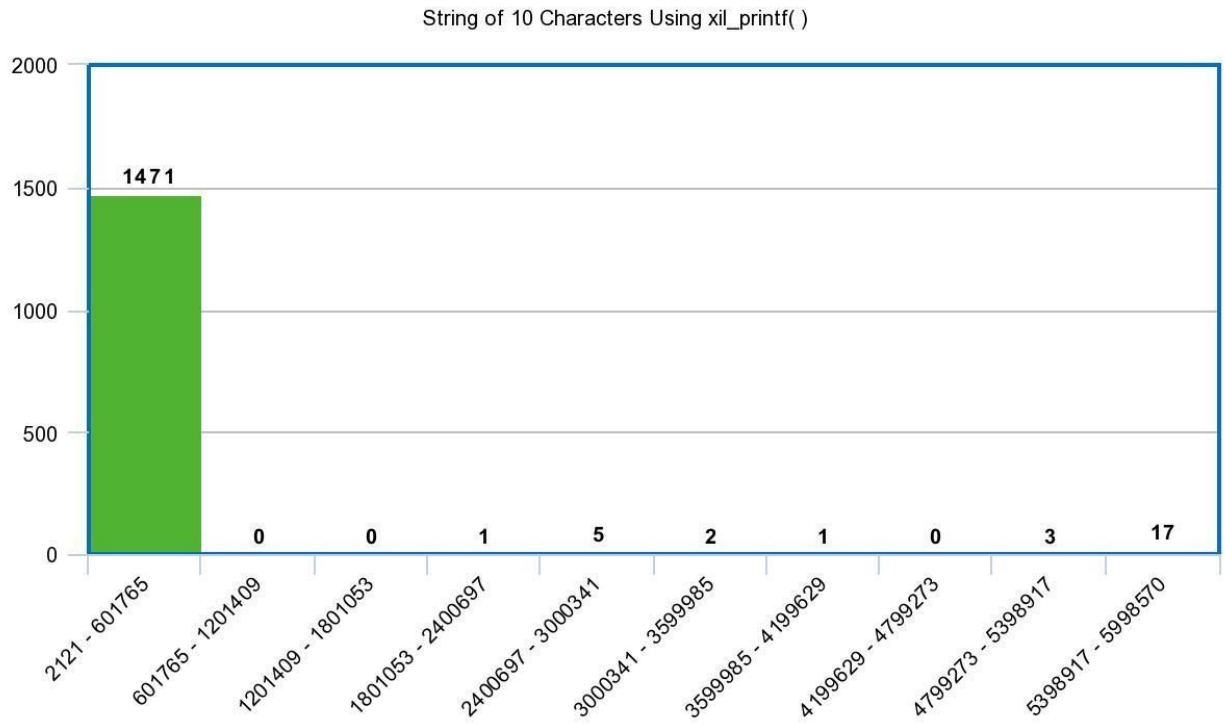
assume that the processor is consistent with each trial of operation or non operation.

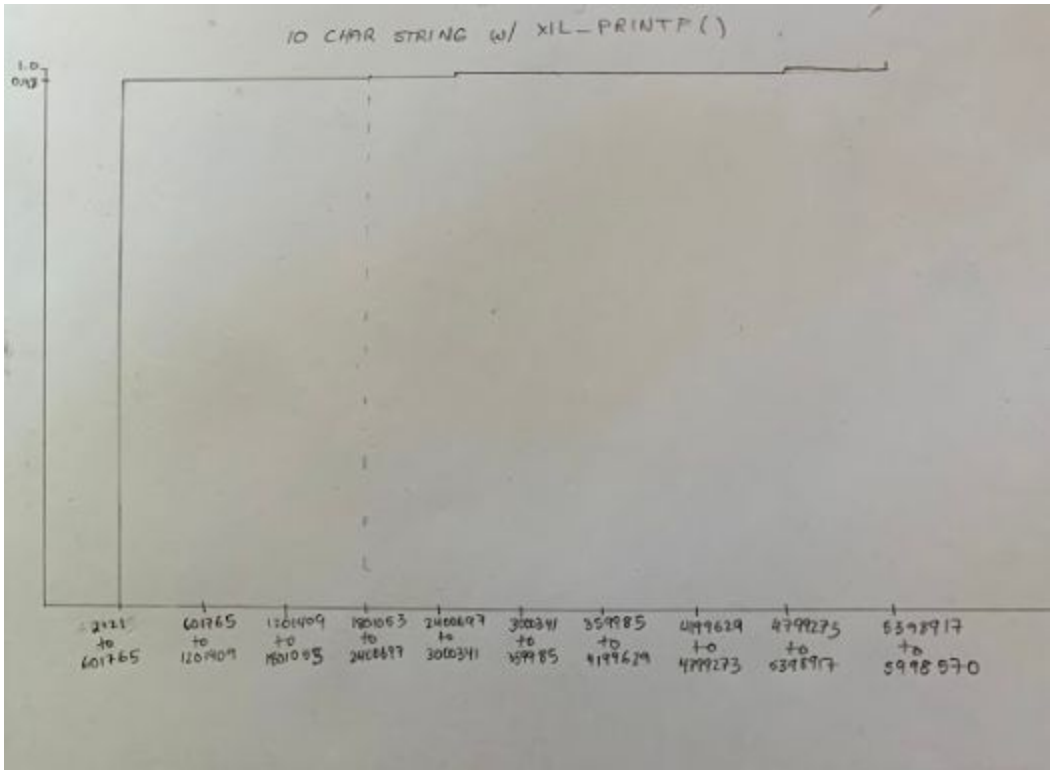


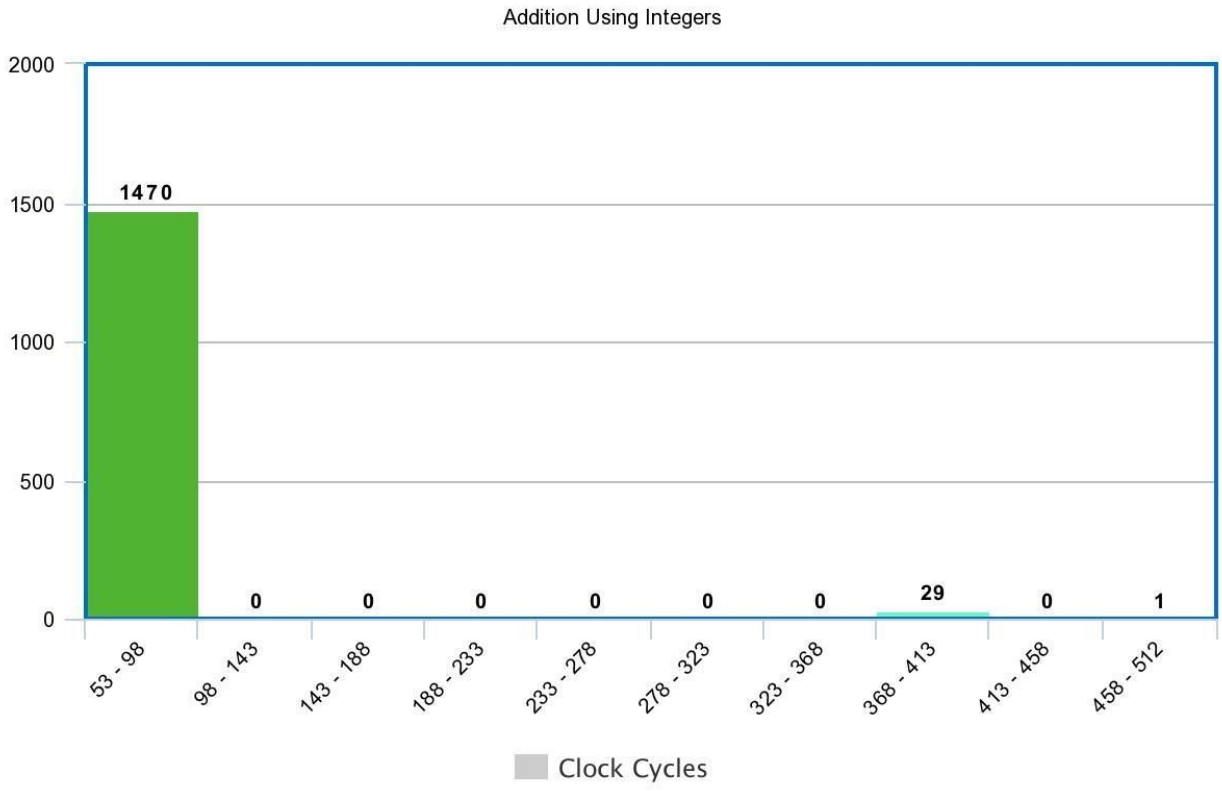
Results:

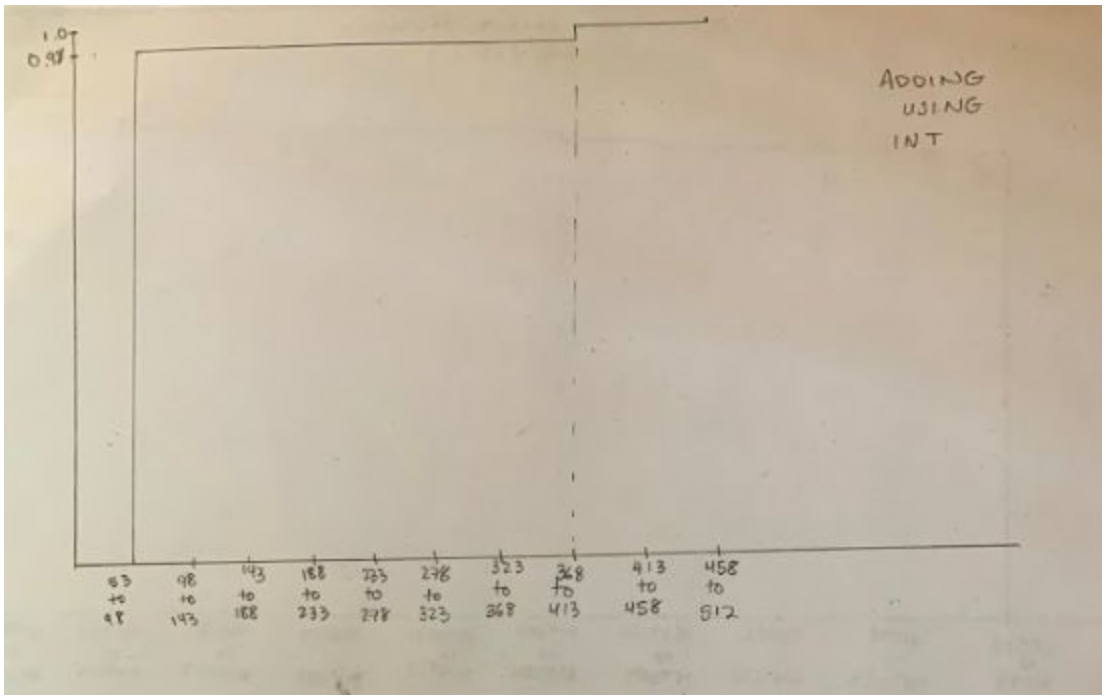

Floating Point Number Using printf( )
1500
1000
500
0
1275
0
0
5
4
0
0
6
142
68
67525 - 724094
724094 - 1380663
1380663 - 2037232
2037232 - 2693801
2693801 - 3350370
3350370 - 4006939
4006939 - 4663508
4663508 - 5320077
5320077 - 5976646
5976646 - 6633224
Clock Cycles

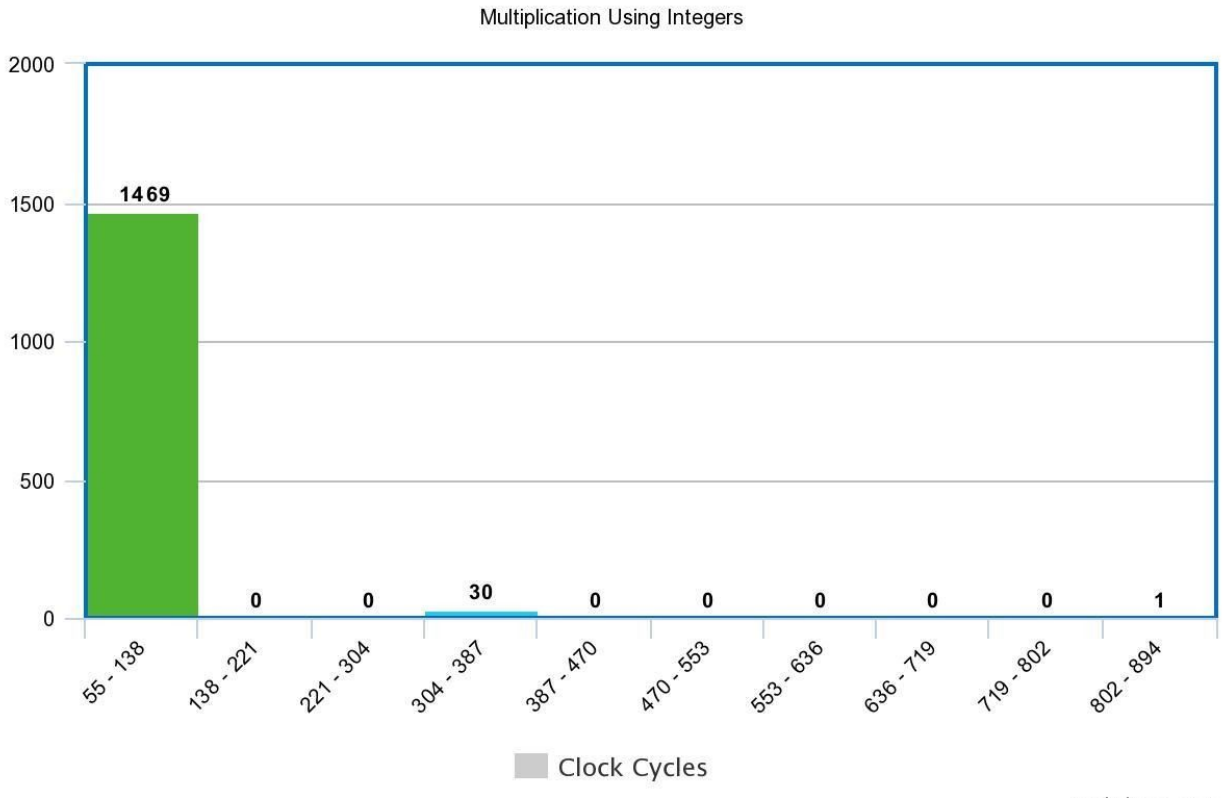

FLOATING POINT NUMBER
USING PRINTF( )

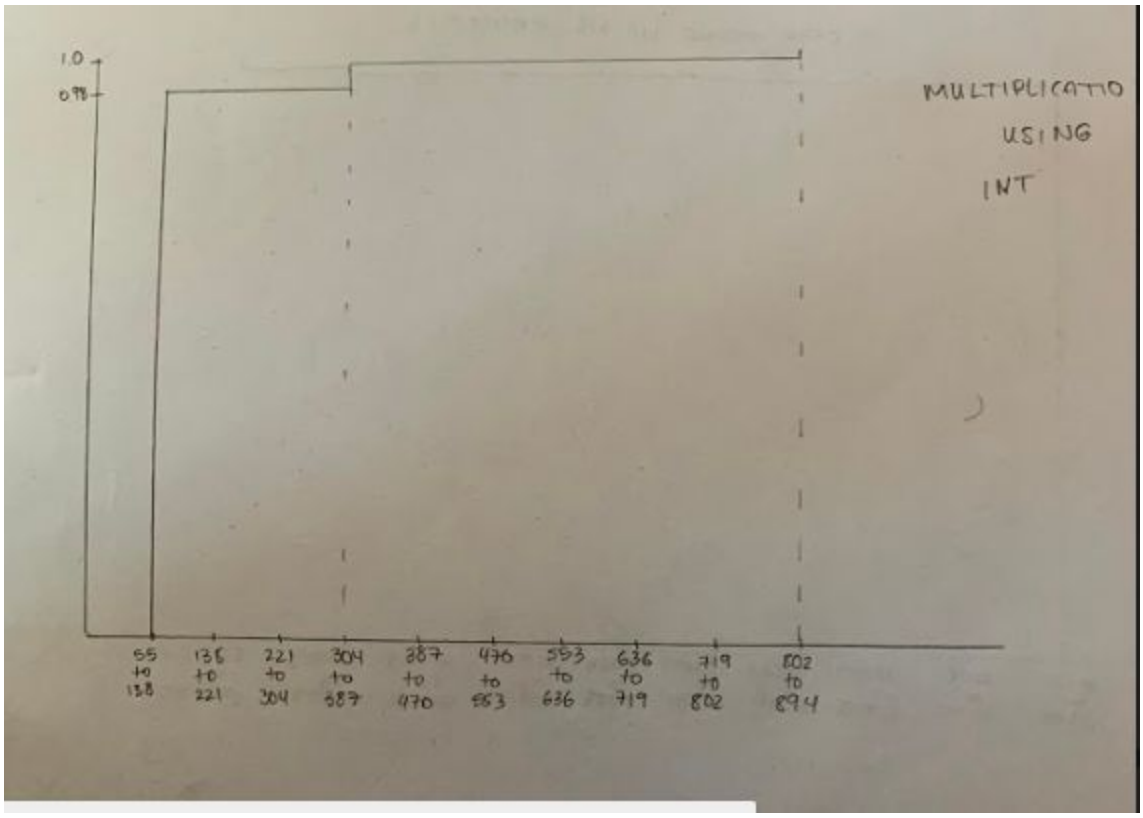

String of 10 Characters Using xil_printf( )
2000
1500
1000
500
0
1471
0
0
1
5
2
1
0
3
17
2121 - 601765
601765 - 1201409
1201409 - 1801053
1801053 - 2400697
2400697 - 3000341
3000341 - 3599985
3599985 - 4199629
4199629 - 4799273
4799273 - 5398917
5398917 - 5998570
Clock Cycles

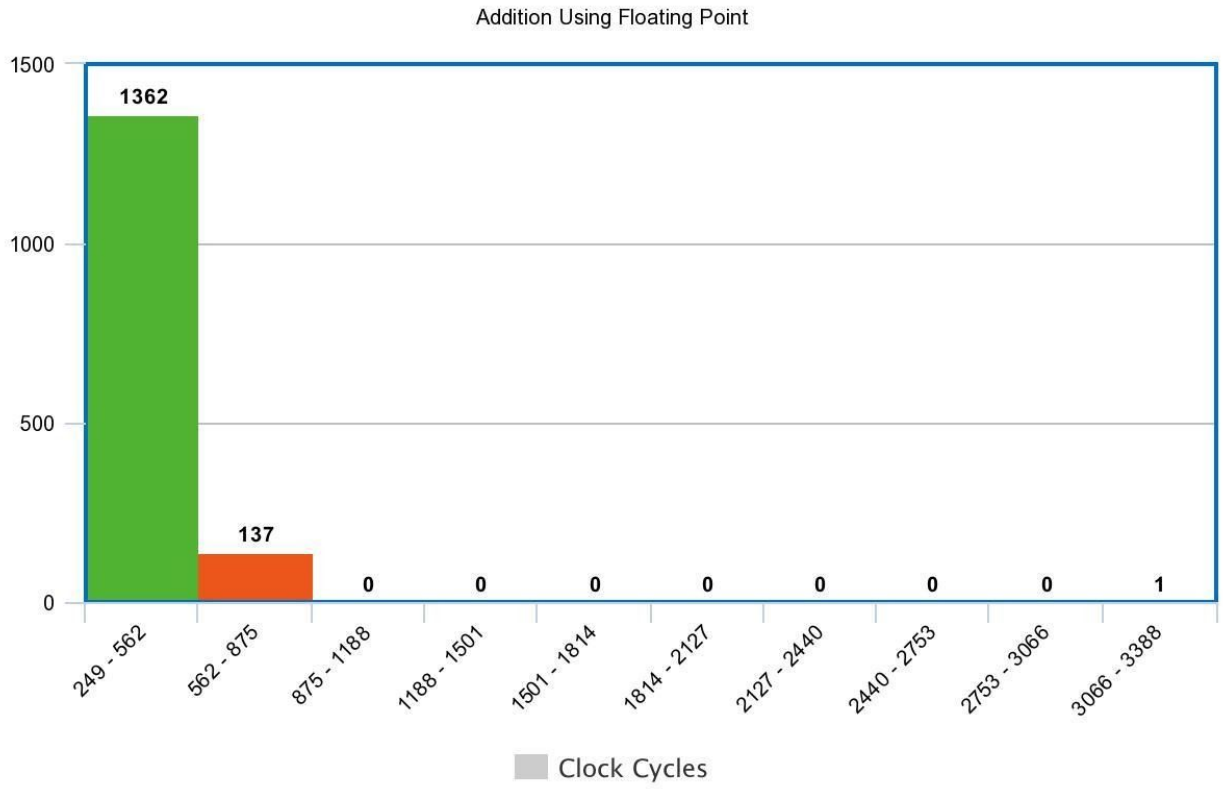

10 CHAR STRING w/ XIL_PRINTF()
2121 to 601765
601765 to 1201409
1201409 to 1801053
1801053 to 2400697
2400697 to 3000341
3000341 to 3599985
3599985 to 4199629
4199629 to 4799273
4799273 to 5398917
5398917 to 5998570

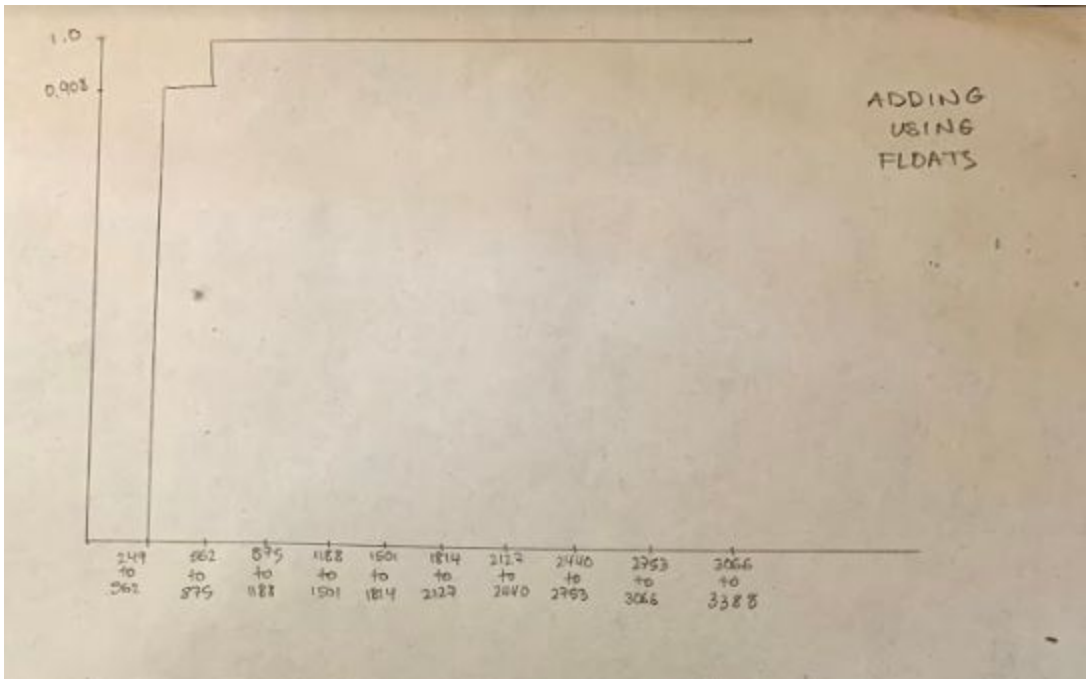

Addition Using Integers
2000
1500
1000
500
0
1470
0
0
0
0
0
0
29
0
1
53 - 98
98 - 143
143 - 188
188 - 233
233 - 278
278 - 323
323 - 368
368 - 413
413 - 458
458 - 512
Clock Cycles

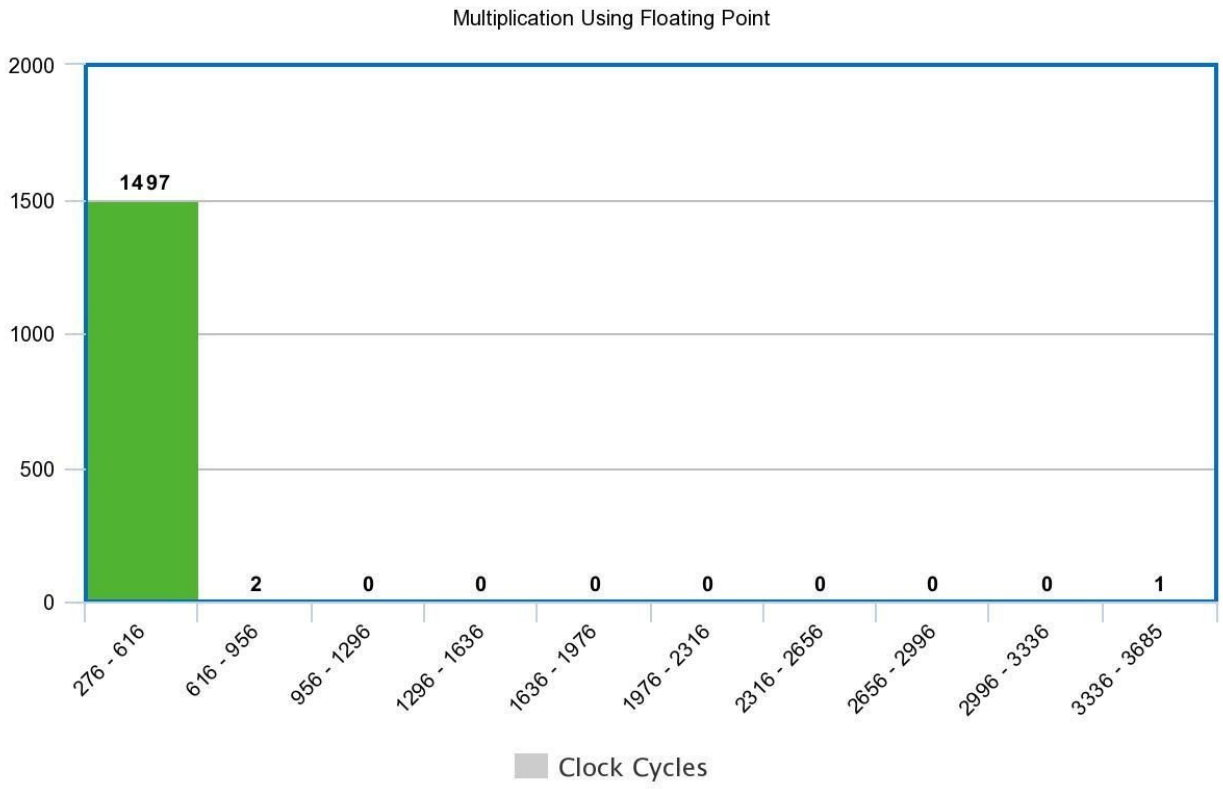

1.0
0.98
ADDING
USING
INT
53 to 98
98 to 143
143 to 188
188 to 233
233 to 278
278 to 323
323 to 368
368 to 413
413 to 458
458 to 512

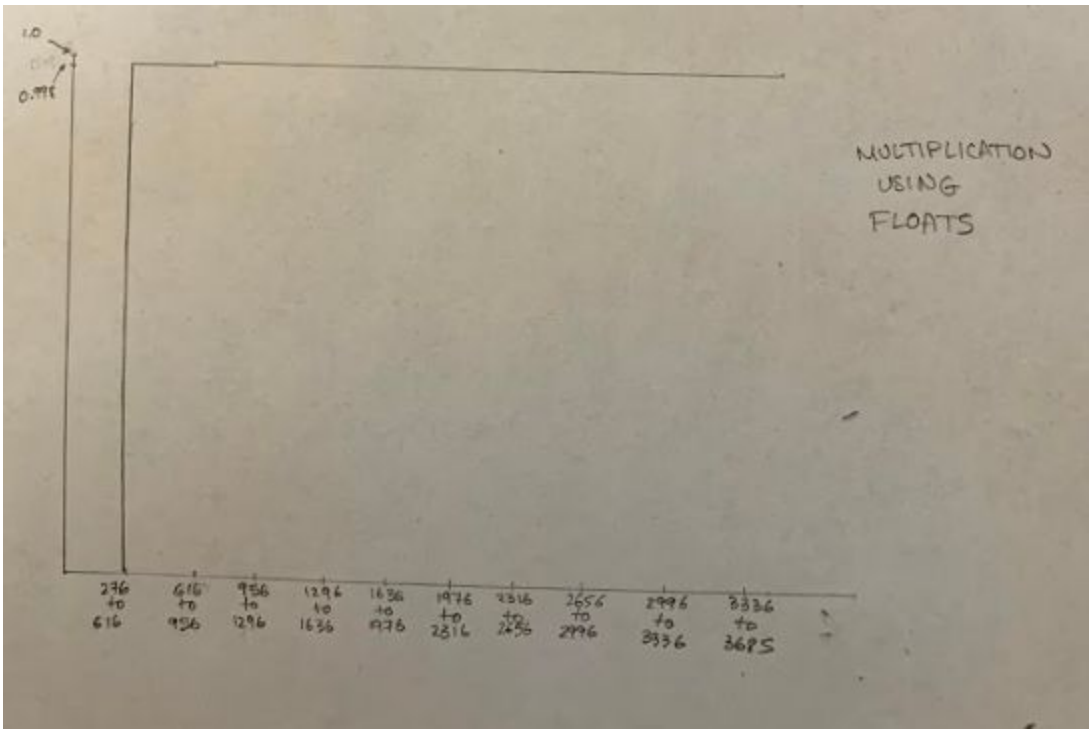

Multiplication Using Integers
2000
1500
1000
500
0
1469
0
0
30
0
0
0
0
0
1
55 - 138
138 - 221
221 - 304
304 - 387
387 - 470
470 - 553
553 - 636
636 - 719
719 - 802
802 - 894
Clock Cycles

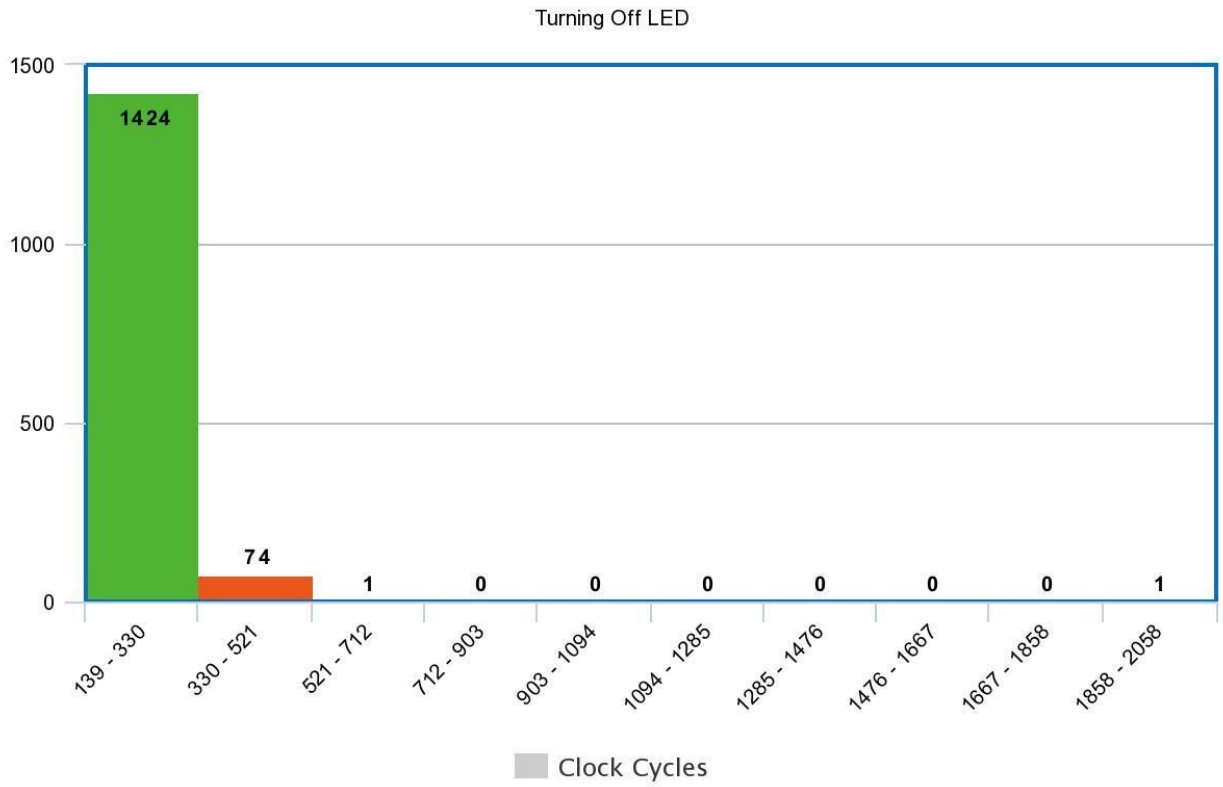

1.0
0.5
MULTIPLICATIO
USING
INT
55 to 138
135 to 221
221 to 304
304 to 387
387 to 470
470 to 553
553 to 636
636 to 719
719 to 802
802 to 894

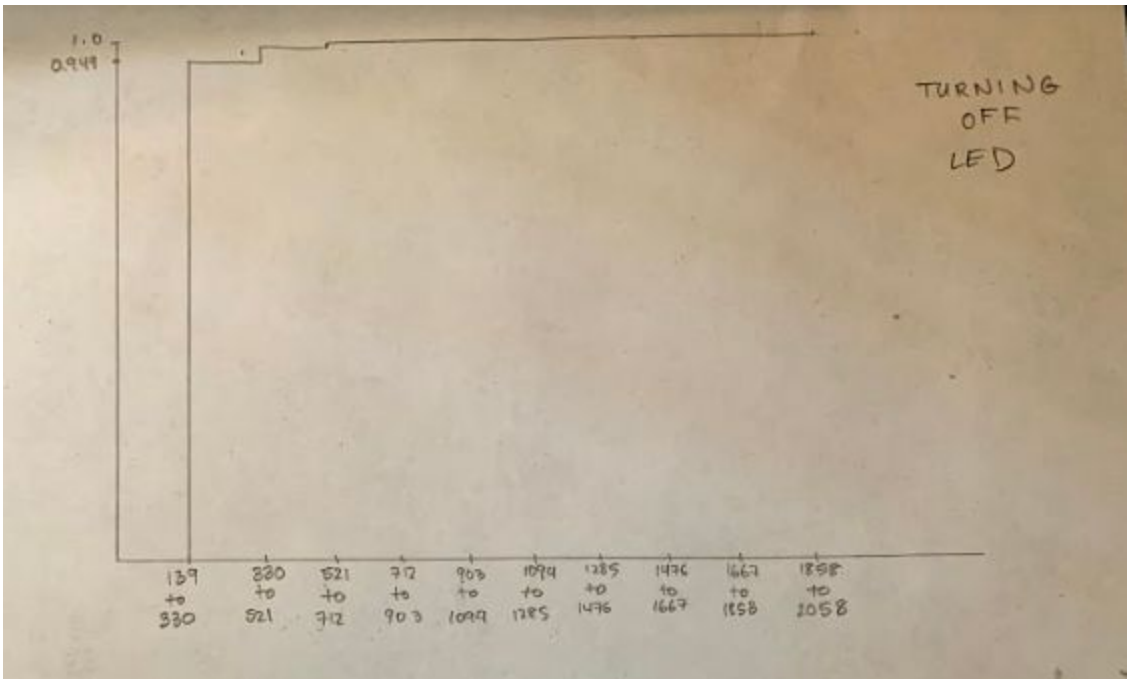

Addition Using Floating Point
1500
1000
500
0
1362
137
0
0
0
0
0
0
0
1
249 - 562
562 - 875
875 - 1188
1188 - 1501
1501 - 1814
1814 - 2127
2127 - 2440
2440 - 2753
2753 - 3066
3066 - 3388
Clock Cycles

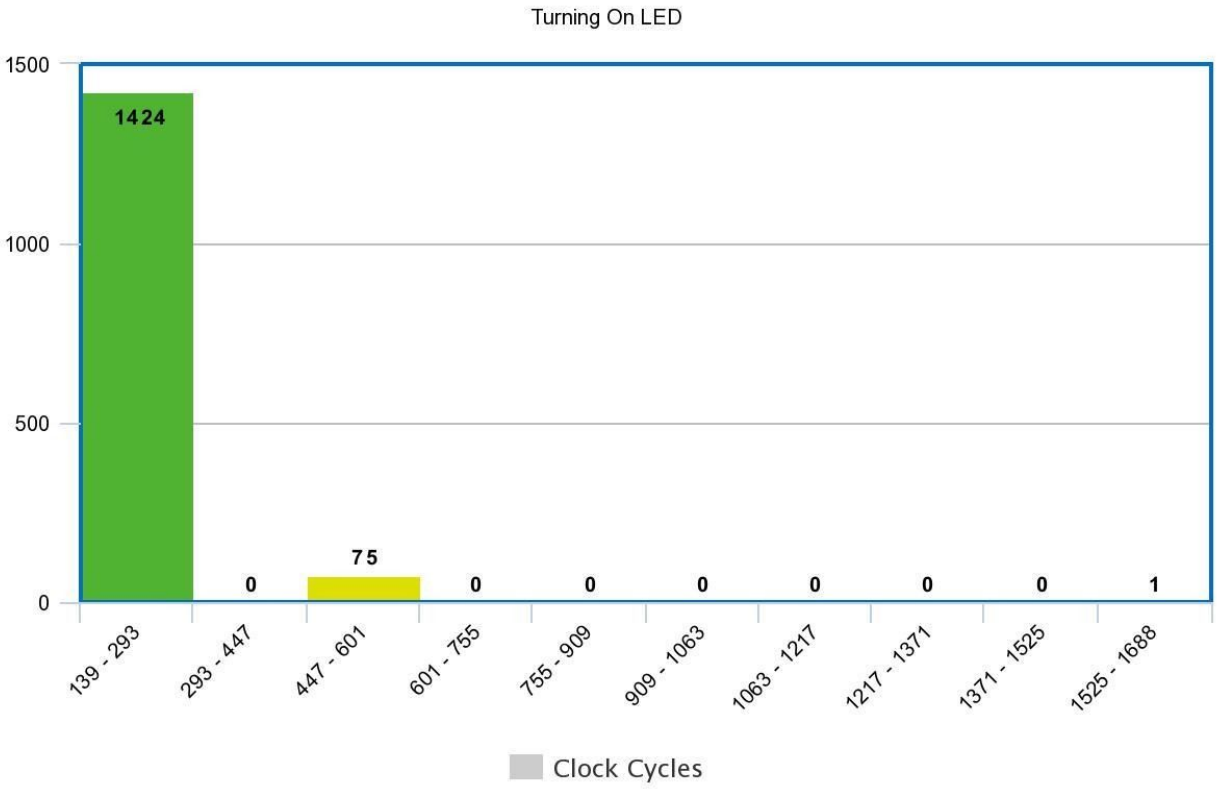

1.0
0.90s
ADDING
USING
FLOATS
249 to 562
562 to 875
875 to 1188
1188 to 1501
1501 to 1814
1814 to 2127
2127 to 2440
2440 to 2753
2753 to 3066
3066 to 3388

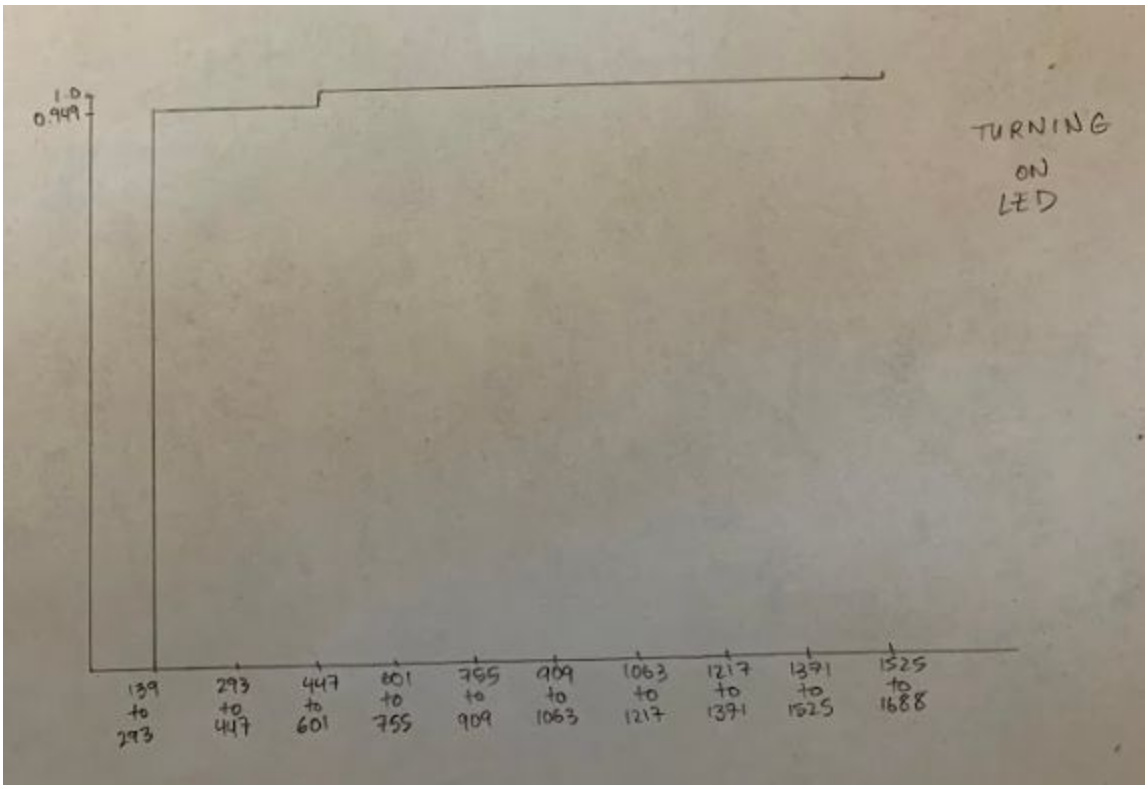

Multiplication Using Floating Point
2000
1500
1000
500
0
1497
2
0
0
0
0
0
0
0
1
276 - 616
616 - 956
956 - 1296
1296 - 1636
1636 - 1976
1976 - 2316
2316 - 2656
2656 - 2996
2996 - 3336
3336 - 3685
Clock Cycles

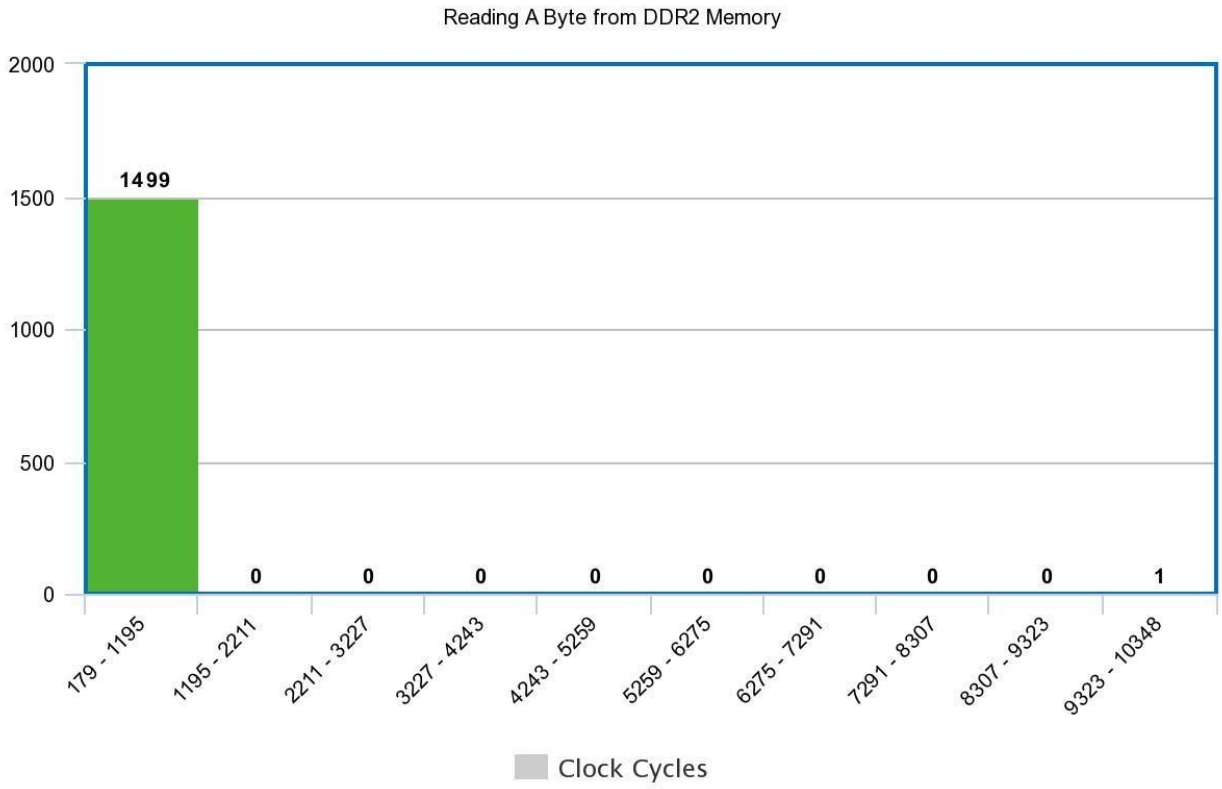

MULTIPLICATION
USING
FLOATS

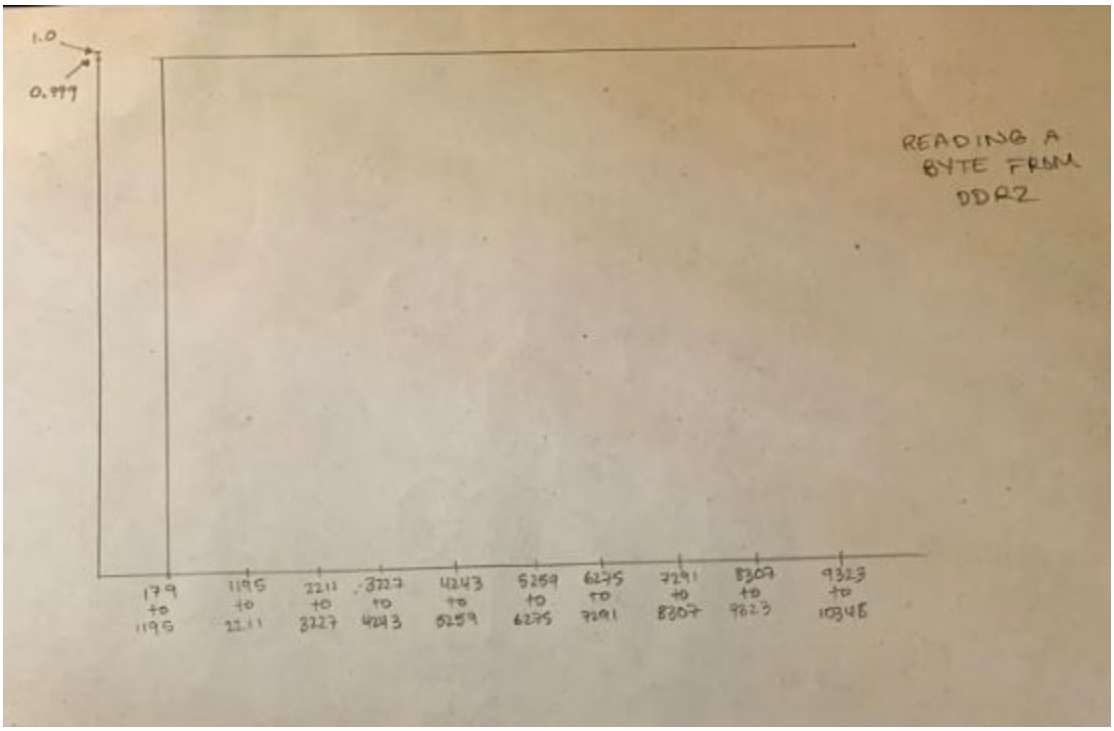

Turning Off LED
1500
1000
500
0
1424
74
1
0
0
0
0
0
0
1
139 - 330
330 - 521
521 - 712
712 - 903
903 - 1094
1094 - 1285
1285 - 1476
1476 - 1667
1667 - 1858
1858 - 2058
Clock Cycles

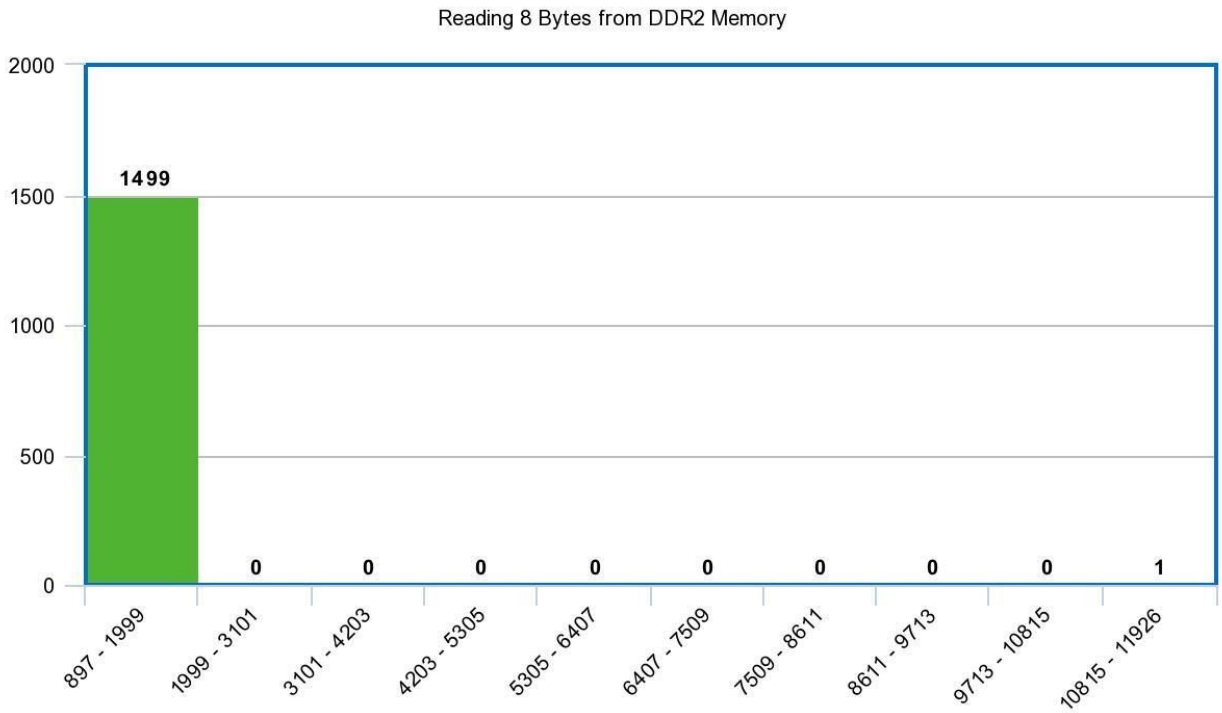

1.0
0.944
TURNING OFF LED
139 to 330
330 to 521
521 to 712
712 to 903
903 to 1094
1094 to 1285
1285 to 1476
1476 to 1667
1667 to 1858
1858 to 2058

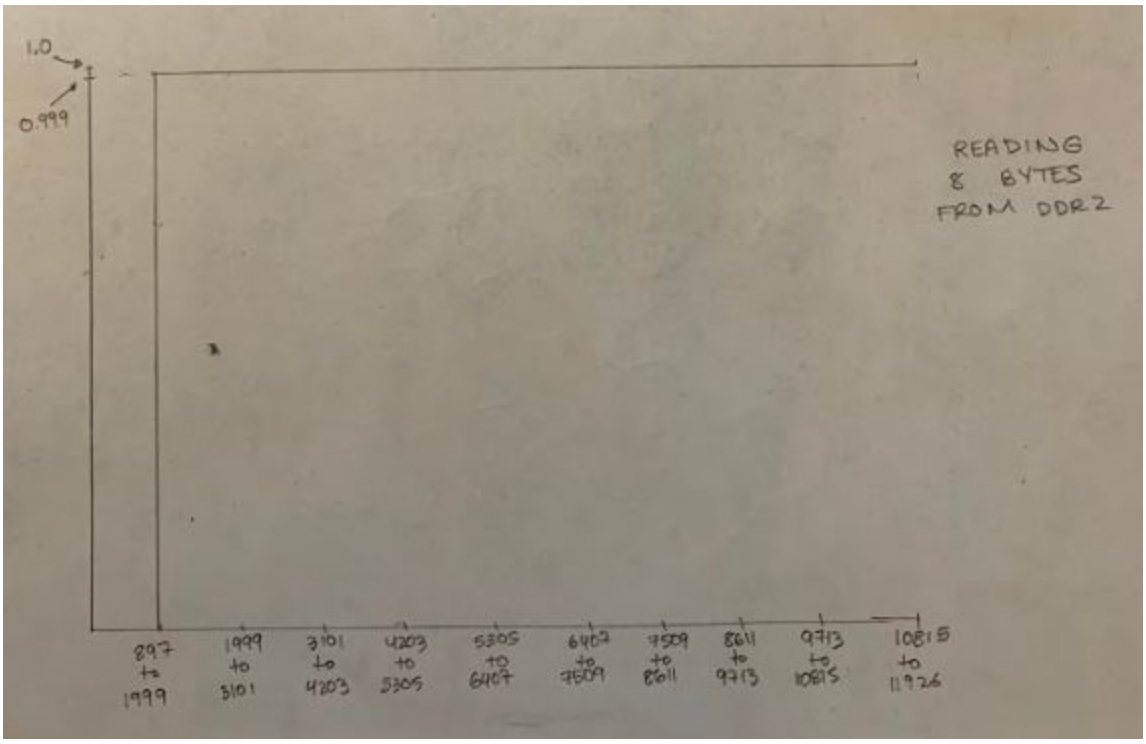

Turning On LED
1500
1000
500
0
1424
0
75
0
0
0
0
0
0
1
139 - 293
293 - 447
447 - 601
601 - 755
755 - 909
909 - 1063
1063 - 1217
1217 - 1371
1371 - 1525
1525 - 1688
Clock Cycles


1.0
0.999
TURNING
ON
LED
139
to
293
293
to
447
447
to
601
601
to
755
755
to
909
909
to
1063
1063
to
1217
1217
to
1371
1371
to
1525
1525
to
1688


Reading A Byte from DDR2 Memory
2000
1500
1000
500
0
1499
0
0
0
0
0
0
0
0
1
179 - 1195
1195 - 2211
2211 - 3227
3227 - 4243
4243 - 5259
5259 - 6275
6275 - 7291
7291 - 8307
8307 - 9323
9323 - 10348
Clock Cycles


1.0
0.999
READING A BYTE FROM DDR2
179 to 1195
1195 to 2211
2211 to 3227
3227 to 4243
4243 to 5259
5259 to 6275
6275 to 7291
7291 to 8307
8307 to 9323
9323 to 10348


Reading 8 Bytes from DDR2 Memory
2000
1500
1000
500
0
1499
0
0
0
0
0
0
0
0
1
897 - 1999
1999 - 3101
3101 - 4203
4203 - 5305
5305 - 6407
6407 - 7509
7509 - 8611
8611 - 9713
9713 - 10815
10815 - 11926
Clock Cycles


1.0
0.999
READING
8 BYTES
FROM DDR2
897 to 1999
1999 to 3101
3101 to 4203
4203 to 5305
5305 to 6407
6407 to 7509
7509 to 8611
8611 to 9713
9713 to 10815
10815 to 11926

Operation	Min	Max	Expected Value	Inconsistency from Min $\frac{Expected\ Value - Min}{Min} \times 100$
No Operation	31	411	34.873	12.49%
Printf( ) Float	67525	6633217	918792.438	1260.67%
xil_printf( ) String of 10 Characters	2121	5998566	105817.898	4889.06%
Addition Using Integers	53	508	59.683	12.61%
Multiplication Using Integers	55	889	62.156	13.01%
Addition Using Float	249	3382	281.431	13.02%
Multiplication Using Float	276	3678	311.585	12.89%
Turning LED On	139	1682	156.815	12.82%
Turning LED Off	139	2054	156.815	12.82%
Reading 1 Byte From DDR2 Memory	179	10343	232.784	30.05%
Reading 8 Bytes From DDR2 Memory	897	11923	1168.307	30.25%

Analysis

In looking at the data we collected, there's a few observations that can be made. Since there is a disparity and spread of values with each operation, we need to take a look at the expected values of these operations as well. The fastest operations, which we will define as the operations with the smallest minimum values of clock cycles, was addition and multiplication using integers. These operations are cost effective relatively compared to the rest of the operations. They are also very similar to each other, with the exception being the 75% greater maximum cycles measurement found with multiplication. These operations are very well implemented and commonly optimized for in hardware so it is no surprise that they are close to the no operation measurements. The expected values found for both operations tell us that they lean close to the minimum values similar to no operation. As a way to measure this distance of the expected value relative to the min, I define the inconsistency value as shown in the chart. We can see that the inconsistency values of addition and multiplication are close to the performance of no operation.

Interestingly, the next quickest group of operations are turning the LED on and off. This involves initializing the GPIO instance and writing on or off at every trial. In looking at the

expected values, they are very close to their minimum value, which are the same for both of them. This is likely due to having a very similar physical procedure in enabling and setting voltages at the GPIO pin. We weren't really sure why the maximum value was higher to turn the led off by 22%. Comparing the inconsistency values, they are actually lower than multiplication using integers. You may say that averaged out these two operations are quick more consistently because their expected values are closer to their minimum.

The next observation we made was that reading a byte from DDR2 memory is faster than addition and multiplication using floating point numbers. While this seems to go against our previously specified point of optimized hardware for addition and multiplication, it seems the more precision that we require due to floating points, an increase in cycle time is to be expected. We do see expected values close to their minimums which means that there must not be much complexity happening.

Reading a byte from DDR2 memory does however have an inconsistency score of 30.05% which means that its expected value is to be found further away than its minimum value. This is reflected in its quite high expected value which is 5778% higher than its minimum value. Looking at the minimum value for an 8 byte read from DDR2 tells us that there is not exactly 8 times the amount of work to be done compared to reading a single byte but it is certainly higher. On the other side of the spectrum, its maximum value is really close to a reading a single byte, so there must be something substantial that affects both operations equally.

Lastly, printf( ) and xil_printf( ) were the most expensive operations giving reason to the cautionary tale of bloated printing timing delays in embedded system courses. xil_printf( ) has the smaller minimum value, it has nearly the same maximum value of printf( ). The expected value of printf( ) is higher than xil_printf( ) but don't let the inconsistency value fool you. While it is lower than xil_printf( ), it is relative to its own minimum value, which is a lot higher than xil_printf( ). This is perhaps a failing point of this measurement unit we came up with, but it does allow us to highlight the fact that the expected value of xil_printf( ) is really far away from its minimum. We also must say that they were both printing different data types and amounts, so that is also a factor.

In conclusion, the data gathered in our trials gave valuable insight on the different amount of cycles that different operations might take, which we can definitely take into account and apply when we create tasks for the processor. The large number of trials, as well as taking measurements of the min, max, expected values, as well as the inconsistency from the min is important in giving a clearer picture of the processor, and a clear picture of expected behaviour through gathered data is absolutely necessary when designing embedded systems, as failure can be very costly and possibly dangerous.