Measuring And Analyzing Variations in FPGA Operation Timings

Oviya Seeniraj

2 | Purpose and expected goals

The objective of this lab is to acquaint ourselves with Vivado, Xilinx, the Nexys Artix A7 board, and the Microblaze processor. Our primary goal is to analyze the timing of various operations on the FPGA by creating a full RTL project and configuring the processor to interface with DDR2 memory and AXI timer peripherals. Our final goal is to understand how the timing of different FPGA operations such as DDR2 memory access, USB port function, integer and float addition, and LED toggle functions vary, and analyze what factors may lead to these variations.

3 | Methodology

A. Design Tasks: we will create a Vivado RTL project, configure the Microblaze processor to use caches and be equipped with enough memory, interface with peripherals using the AXI bus interface, measure timing with the help of Vivado's Vitis IDE, and perform regression / statistical analysis on the collected data.
B. Assumptions required for design analysis and procedures: Operations are executed in a tightly constrained, cache-enabled embedded environment. Timing measurements are linear. Memory and buses can be shared.
C. Observations from the design tasks: Will use my designs to measure operation timings from the FPGA
D. Plan for Design Testing: Measure operation timings 10,000 times to find rarer event timings (only 1,000 times for USB port operations due to their high complexity). Generate histograms and CDF plots to visualize timing data. Analyze the expected values and confidence intervals of the recorded data.

4 | Results

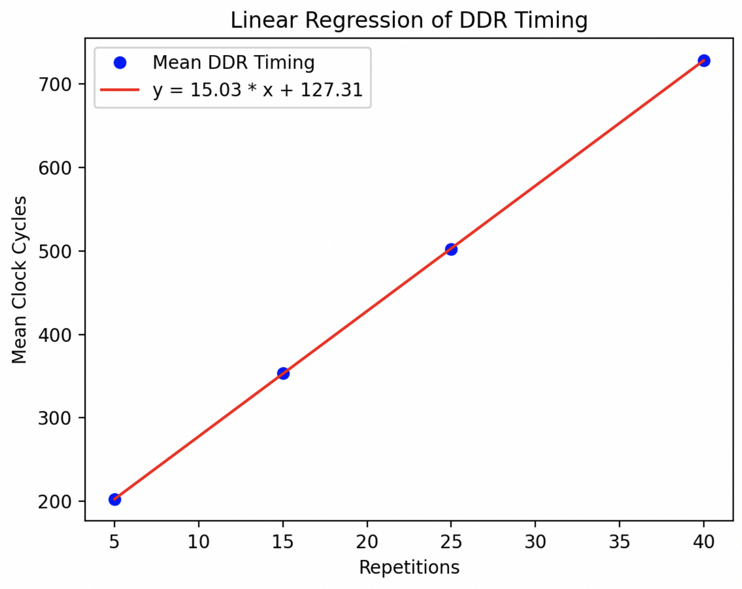
A. METRIC RESULTS

I met all my design goals. I successfully deployed my project to my FPGA and measured its operation timings, and then generated visualizations and statistically analyzed my data using Python.

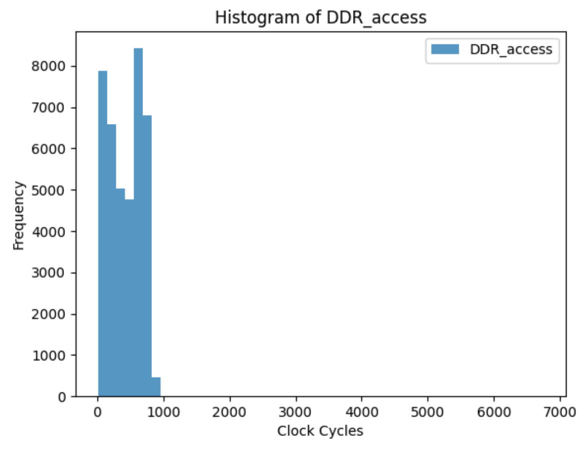
1 | Regression estimate of overhead and DDR timing

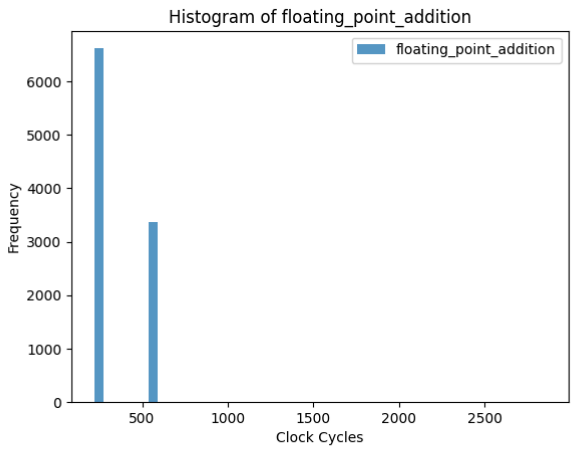
y = 15.03 * x + 127.30

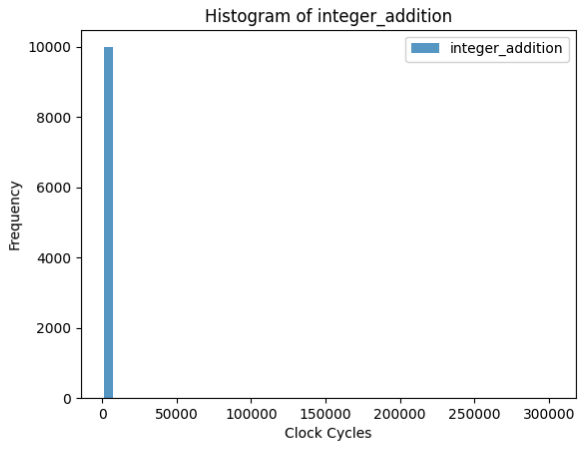
Overhead cycles = 127.30
Clock cycles needed per repetition: 15.03

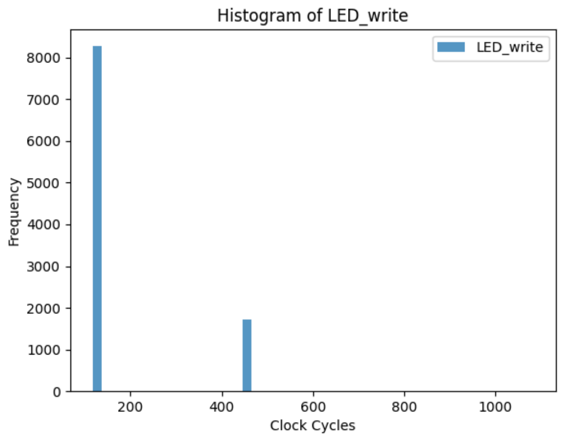


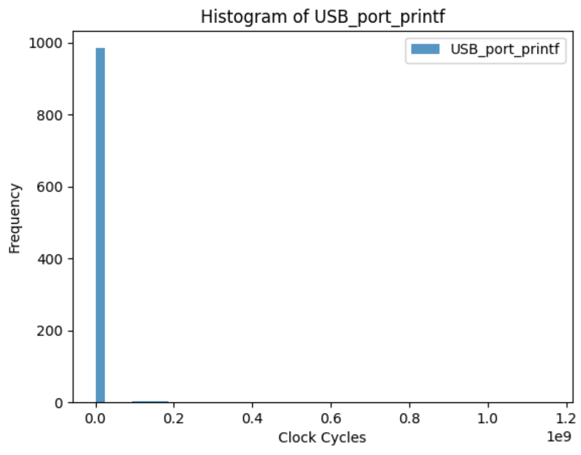
2 | RAW Histograms of measured data

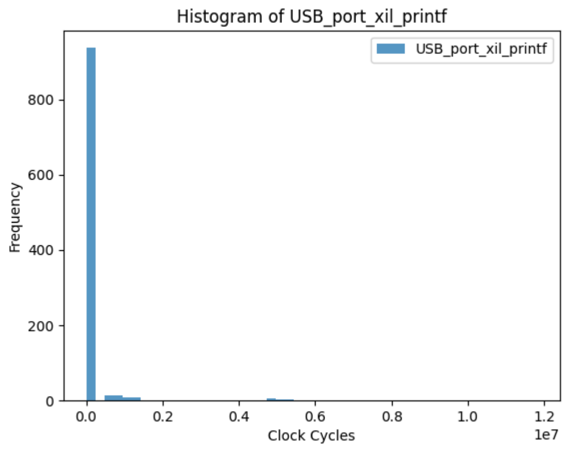




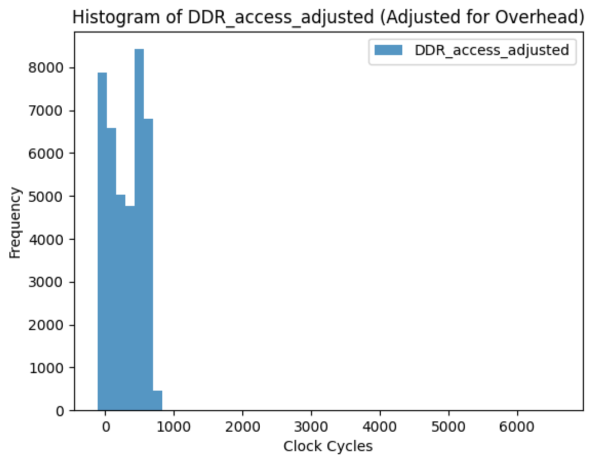


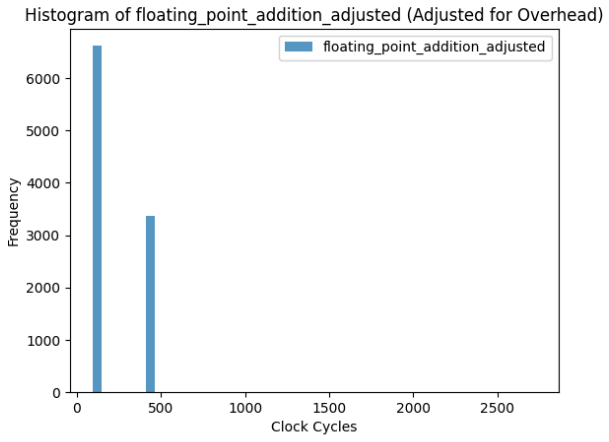


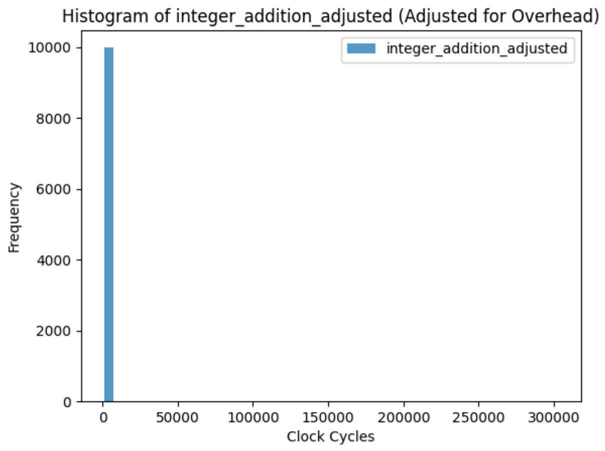


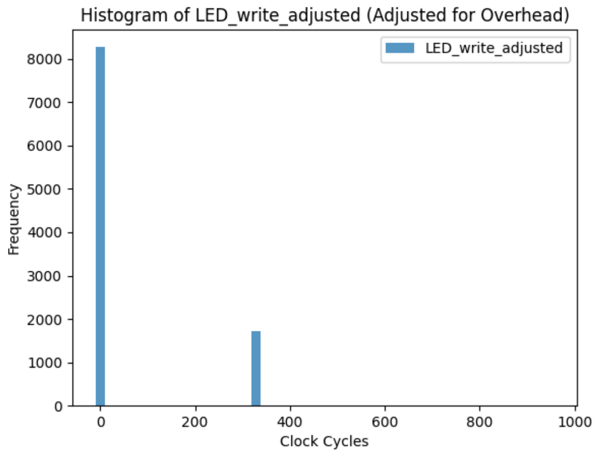


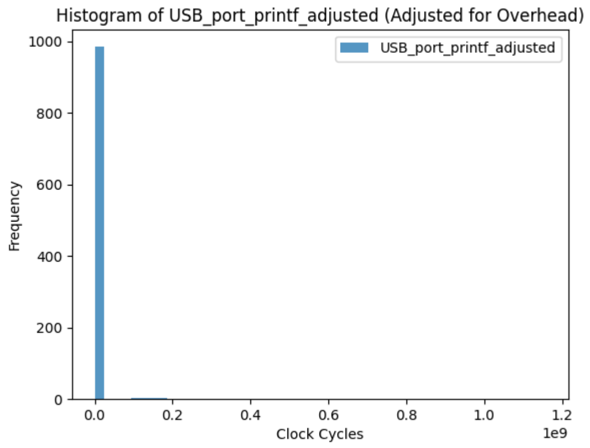
2 | ADJUSTED FOR OVERHEAD Histograms

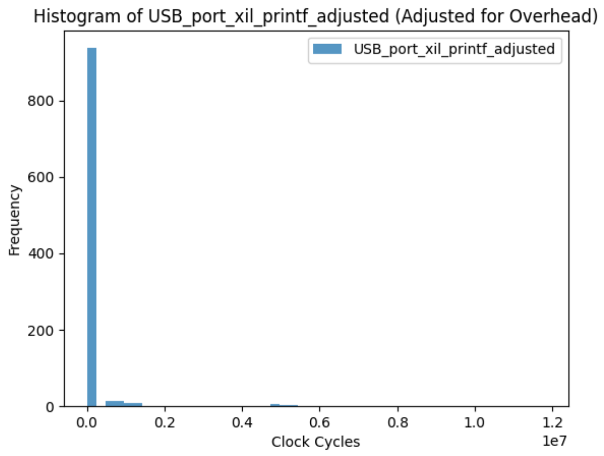












3 | DDR Access Timing Analysis

Expected Value for DDR Access Timing:

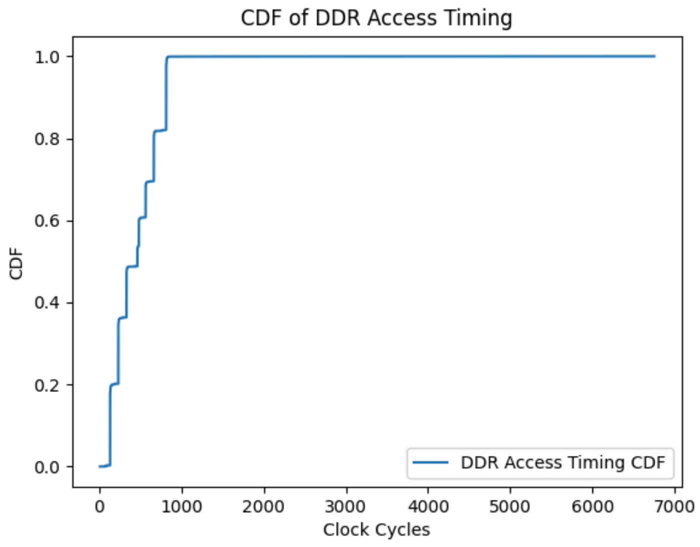
Mean: 446.70 cycles

Standard deviation: 259.54 cycles

95% Confidence Interval:

Empirically determined using Python:

(444.1546, 449.2439)



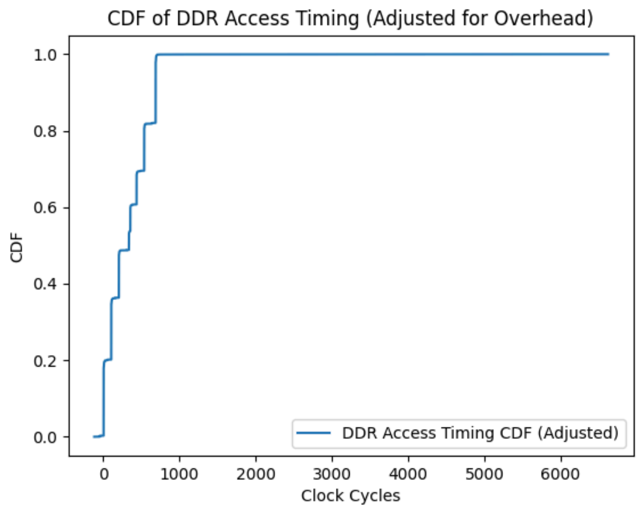
Expected Value for DDR Access Timing Adjusted for Overhead:

Mean: 319.40 cycles

Standard Deviation: 259.54

95% Confidence Interval (Adjusted for overhead):

(316.8563, 321.9455)



4 | Timing and Uncertainty Observations

- The magnitude of measured values are relatively low for DDR access, integer and floating point addition, and LED on/off changes, with magnitudes in the $10^2/10^3$ range.
- On the other hand, The USB port functions have incredibly high magnitudes of access timing cycles, with a magnitude of 10^6 cycles.
- All the timing uncertainties for every operation have almost the same magnitude as their corresponding average accesses as well, which is interesting.
- NOTE: My data shows that DDR access is faster than integer addition, which is theoretically unexpected. My explanations will delve into the expected result rather than this average from trials, which is likely due to CPU occupation and several cache misses during testing.

DDR2 memory access:
- Mean: 446.70 cycles
- Std Dev: 259.54 cycles

Integer addition:
- Mean: 3109.87 cycles
- Std Dev: 4245.70 cycles

Floating point addition
- Mean: 338.33 cycles
- Std Dev: 159.23 cycles

LED turning on/off
- Mean: 174.28 cycles
- Std Dev: 125.69 cycles

USB port printf
- Mean: 2728746.37 cycles
- Std Dev: 39433615.25 cycles

USB_port_xil_printf
- Mean: 144275.37 cycles
- Std Dev: 838204.68 cycles

After adjusting for overhead:

DDR2 memory access:
- Mean: 319.40
- Std Dev: 259.54

Integer addition:
- Mean: 2982.57
- Std Dev: 4245.70

Floating point addition:
- Mean: 211.04
- Std Dev: 159.23

LED write:
- Mean: 46.98
- Std Dev: 125.69

USB port printf:
- Mean: 2728619.07
- Std Dev: 39433615.25

USB port xil_printf:
- Mean: 144148.07
- Std Dev: 838204.68

5 | Outliers

I used Python to calculate outliers as access timings with z-scores > 3 (z-scores calculated from the statistics reported in the "Timing and Uncertainty Observations" section of this report). These outliers are the same regardless of accounting for overhead.

- DDR Access: 21 outliers over ~40,000 data points
- Integer addition has 2 outliers over ~10,000 data points
- Floating point addition has 1 outlier over ~10,000 data points
- LED on/off changes have 1 outlier over ~10,000 data points
- USB port's printf() func has 9 outliers over ~1,000 data points
- USB port's xil_printf() func has 17 outliers over ~1,000 data points

These outliers are likely caused by the following issues:

- ***Cache misses***: Cache misses result in direct DDR memory data fetching (slower than main memory). Integer and floating point addition sometimes result in cache misses (intermediate values/instructions)
- ***Interrupt Handling***: USB port functions rely on I/O operations, which get interrupted by higher-priority tasks or system interrupts, explaining why their average is high.
- ***Bus Contention***: peripheral access and multiple devices all try to access buses at once, which especially slows down USB port functions (also DDR (although this is moreso affected by memory contention))
- ***Task Scheduling:*** The processor switching between tasks often creates delays and slows down tasks such as floating point/integer addition
- ***Pipeline Stalls****:* Not as applicable to this lab, but when the processor is waiting for data due to a cache miss or I/O operation, addition operations can experience pipeline stalls
- ***USB passthrough:*** USB port operations took even longer as they had to go through USB passthrough between my Mac's hardware and the NoMachine VM, which is where I had to connect my board to port over the program

6 | Explanation for Variations

In essence, low-complexity operations have lower variation. For example, LED toggling directly interfaces with registers and is a basic I/O operation requiring very little processing. On the other hand, USB operations are complex I/O operations, so their access timings are high. Similarly, addition is faster because it's computationally straightforward (especially with an enabled cache for the addition). However, DDR operations are more dependent on memory access which causes higher timing variations. CPU/IO bounds also explain variation: addition is CPU bound, which does not experience I/O or memory bottlenecks like USB port functions would, so its variation is lower. Similarly, peripheral devices can cause delays through interrupts and bus usage, so USB ports and DDR access, which use shared resources, have greater timing variability than LED toggling/addition.

B. DESIGN LIMITATIONS

I was limited by our simulated cache - it being enabled led to highly variable memory, although this led to valuable insights and understanding of memory in embedded systems.

C. DESIGN TEST RESULTS THAT DID NOT MEET EXPECTATIONS

My integer addition, though less complex than floating-point addition operations, was far higher than floating-point by a magnitude of 10. More surprisingly, it had more access cycles than DDR2 memory access, which should have taken more cycles due to bus access and cache misses, as explained in 4.A.5 of this lab report.

D. ROAD BLOCKS / PARTS THAT DID NOT MEET SPEC

I was able to successfully complete this lab and accurately analyze timing operations based on my measurements. However, some problems I ran into on the way were storing console output: copying and pasting 10,000 outputs was not viable, which I solved by changing the run configurations to automatically store console output to a file. Connecting my Nexys board to my laptop was quite difficult as well, as I ran into unexpected errors when working with USB passthrough.

E. ISSUES AND SOURCES OF ERRORS IN THE LABS

As described throughout the sections of this lab, I had errors with the timing operations of integer addition. Though I reran the trials, I continually observed that despite its lower complexity than floating point addition and lower likelihood of cache misses than DDR2 memory access, it took longer and required more cycles. A source of error in the high USB port timings is the need for USB passthrough: measurements may have differed if I ran the program entirely locally.

F. SUGGESTED IMPROVEMENTS TO LAB

Clearer instructions on what graphs to generate and how to account for some skew would be much appreciated. Also, the advice to account for overhead came a bit late on Campuswire

---

Conclusion

This lab introduced the essential tools we will use to create hardware/software interfaces using Vivado and Vitis. By conducting timing analyses for various operations, we understood timing uncertainties and peripheral interactions in a constrained embedded system. Seeing these high variabilities in USB and DDR access times emphasizes the challenges and importance of managing memory and I/O in embedded systems. We understand how complexity has a large impact on timing, which is a crucial aspect of user experience and interface design.