line 1: Christopher Kingsley   
line 2: *Electrical & Computer Engineering Department   
(of Affiliation)*  
line 3: *North Carolina Agricultural & Technical State University   
(of Affiliation)*line 4: Greensboro, North Carolina United States  
line 5: [christopherkingsley123@gmail.com](mailto:christopherkingsley123@gmail.com) / cakingsley@aggies.ncat.edu

**Abstract**  
 This paper presents hardware accelerated FPGA implementations designed to significantly enhance the performance of two critical statistical computations within the bioinformatics tool FastQC: the getMean() and getPercentile() functions. Traditional Java implementations rely heavily on sequential loops and redundant calculations, leading to inefficiencies when analyzing large genomic datasets. To address this issue, we developed two optimized hardware designs using VHDL implemented on an Intel/Altera DE2i-150 FPGA board equipped with a Cyclone IV FPGA operating at 50 MHz. The getPercentile() function employs a parallel prefix accumulation architecture, achieving deterministic calculation of percentile thresholds within 4 clock cycles (80 ns). The getMean() function utilizes a pipelined binary reduction tree architecture, resulting in a deterministic latency of 15 clock cycles (300 ns). Compared to Java implementations averaging 830 ns for getPercentile() and 2460 ns for getMean(), our FPGA implementations significantly reduced latency and provided deterministic performance. These results illustrate FPGA technology’s considerable potential for addressing computational bottlenecks in bioinformatics analyses.

**Keywords: FPGA, VHDL, FastQC, Bioinformatics, Parallel Accumulation, Binary Reduction Tree**

**Introduction** Bioinformatics plays an increasingly critical role in modern biological research, particularly in the management and analysis of extensive genomic datasets produced by advanced sequencing technologies. As genomic data volumes grow rapidly, conventional software tools encounter significant performance limitations, especially during intensive statistical analyses. FastQC is a widely used bioinformatics tool designed for assessing sequencing data quality. However, its Java based implementations of critical statistical functions, namely mean quality score calculation (getMean) and percentile determination (getPercentile), rely predominantly on sequential algorithms. This reliance results in computational inefficiencies and reduced performance with large scale datasets.

Field Programmable Gate Arrays (FPGAs) present an attractive alternative to general purpose CPUs because they enable highly parallel operations, exhibit deterministic execution patterns, and offer enhanced energy efficiency. FPGAs permit custom hardware optimization tailored specifically to application requirements, making them particularly well suited for accelerating bioinformatics workflows.

This paper introduces two FPGA implementations targeting FastQC’s statistical functions, getMean() and getPercentile(). These implementations are optimized using VHDL hardware designs implemented on an Intel/Altera DE2i-150 development board containing a Cyclone IV FPGA. Performance evaluations against original Java implementations highlight substantial latency improvements, validating FPGA acceleration as a practical solution for computational challenges in bioinformatics.

**Background** Bioinformatics tools like FastQC are essential for quality control and assessment of genomic sequencing data. Nevertheless, many current software implementations predominantly use sequential, iterative computation patterns. CPU based software executes tasks serially or with limited parallelism, constraining computational performance when processing increasingly large genomic datasets.

Recent advances have emphasized hardware acceleration as an effective solution to these constraints. While GPUs effectively handle floating point operations and large scale parallel workloads, FPGAs provide precise control over fine grained parallel computation and deterministic execution characteristics favorable for bioinformatics tasks. Using hardware description languages such as VHDL, developers can design customized circuits specifically optimized for targeted algorithms, thus significantly outperforming general purpose CPUs in relevant computational tasks.

**Research Methodology** We first profiled the FastQC Java implementation to identify performance critical computational routines. Through instrumentation of timing measurements within the Java code, we determined that getMean() and getPercentile() functions represented the primary computational bottlenecks.

Based on these insights, functionally equivalent FPGA implementations were developed in VHDL to replicate the original Java algorithms. Once the correctness was verified, hardware specific optimizations were introduced. For percentile computations, a parallel prefix accumulation architecture was implemented, partitioning the input data into concurrent processing blocks. For the mean calculation, a balanced pipelined binary reduction tree was developed to compute the sum of weighted values efficiently with minimal latency. An integrated cycle counter managed by a finite state machine precisely measured FPGA computation latency to enable accurate performance evaluation.

**Architecture** Both FPGA implementations utilize a shared internal RAM module containing 150 memory entries, each entry being 64 bits wide. This RAM module facilitates efficient indexed access based on provided offset values.

For percentile computations, the FPGA hardware divides input data into blocks containing ten elements each, performing local prefix sums concurrently within each block. After quickly identifying the block where the percentile threshold is crossed, a localized final scan determines the exact threshold index. This entire operation is completed deterministically in four clock cycles.

The mean computation uses a pipelined binary reduction tree structure that simultaneously processes all 150 weighted data points and associated count values. These inputs traverse multiple reduction stages, progressively reducing data until producing the total sum and total count after precisely 15 pipeline stages. Both architectures employ a start and done handshake protocol, ensuring deterministic execution and simplifying integration into larger workflows.

**Experimental Setup** FPGA designs were implemented and validated on an Intel/Altera DE2i-150 development board containing a Cyclone IV FPGA operating at 50 MHz. An internal cycle counter triggered by a finite state machine precisely recorded the number of clock cycles from the initiation of computation until completion. The recorded measurements exclusively represent FPGA computation latency and exclude external data transfer overhead.

Java software performance tests were conducted using an Intel i7-9750H CPU running at 2.7 GHz with Java version 11. Each statistical function was evaluated individually using five distinct offset values (0, 32, 64, 96, 128), which represented realistic test scenarios. Before timing each function, a preliminary dummy computational loop was executed to eliminate Java Virtual Machine startup costs and warm up overhead, providing a fair and accurate performance comparison between FPGA and Java implementations.

**Results**  
 FPGA implementations demonstrated substantial latency improvements over Java based counterparts. Specifically, the FPGA percentile computation consistently completed in exactly 4 clock cycles (80 ns at 50 MHz), significantly outperforming the Java implementation, which averaged approximately 830 ns. Similarly, FPGA mean computations consistently finished in exactly 15 clock cycles (300 ns at 50 MHz), substantially outperforming the Java average runtime of about 2460 ns.

FPGA computations provided fully deterministic, fixed cycle execution. In contrast, Java implementations exhibited runtime variability depending on data offsets, ranging approximately from 600 to 1200 ns for getPercentile() and from approximately 900 to 4300 ns for getMean(). The comprehensive comparative results are summarized in Table 1.

| Offset | FPGA getMean (cycles) | FPGA getPercentile (cycles) | FPGA time @ 50 MHz (Mean) | FPGA time @ 50 MHz (Percentile) | FPGA time @ 500 MHz (Mean) | FPGA time @ 500 MHz (Percentile) | Java getMean (ns) | Java getPercentile (ns) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 15 | 4 | 300 | 80 | 30 | 8 | 4,250 | 1,200 |
| 32 | 15 | 4 | 300 | 80 | 30 | 8 | 2,800 | 750 |
| 64 | 15 | 4 | 300 | 80 | 30 | 8 | 2,300 | 800 |
| 96 | 15 | 4 | 300 | 80 | 30 | 8 | 1,550 | 700 |
| 128 | 15 | 4 | 300 | 80 | 30 | 8 | 900 | 600 |

**Conclusion and Future Work** This research demonstrated the significant advantages of FPGA hardware acceleration in overcoming computational bottlenecks within bioinformatics applications, specifically FastQC’s getMean() and getPercentile() functions. Our customized VHDL implementations provided deterministic and substantially reduced execution latency compared to original Java based methods. At 50 MHz, the FPGA designs completed percentile computations within 80 ns and mean computations within 300 ns, approximately ten times faster than Java.

Future research will investigate further FPGA architectural enhancements by extending pipeline depth to increase clock frequencies, improving memory bandwidth, developing advanced data buffering strategies for larger datasets, and integrating these optimized modules into comprehensive FPGA accelerated bioinformatics workflows. These enhancements promise additional performance gains and scalability, providing robust computational support for next generation sequencing analyses.



